ANGLIA RUSKIN UNIVERSITY

FACULTY OF SCIENCE AND ENGINEERING

INTRADAY MACHINE LEARNING FOR THE SECURITIES MARKET

VITALIY MILKE

A thesis in partial fulfilment of the requirements of Anglia Ruskin University for the degree of Doctor of Philosophy

Submitted: July 2022

To my family.

Acknowledgements

This dissertation was prepared in fulfilment of the requirements of the degree of Doctor of Philosophy under the supervision of Dr Cristina Luca, Dr George Wilson and Dr Arooj Fatima at Anglia Ruskin University.

This PhD journey has been an exciting and new experiment in my life. It would not have been possible without the support that I have received from many people. Special thanks to my family for all the sacrifices that they have made on my behalf.

I would like to express my special gratitude and thanks to my first supervisor Dr Cristina Luca; she has been a fabulous mentor for me. I would like to thank her for all the encouragement, trust and goal setting which were vital via my PhD journey.

I would like to show my sincere appreciation to my supervisors:

Dr George Wilson and Dr Arooj Fatima for their help and essential advice and their immense knowledge and patience in reading this thesis and associated papers.

I would also like to thank my head of the School of Computing and Information Science, Professor Marcian Cirstea, for his guidance, advice and opportunities to deliver lectures and practicals to students in machine learning and core mathematics.

Thank you to David Cobbett, for helping me with the final proofreading for this thesis.

Finally, I would like to extend my thanks to everyone else who has helped me in one way or another. These include many other staff members at the School of Computing and Information Science at Anglia Ruskin University.

ANGLIA RUSKIN UNIVERSITY

ABSTRACT

FACULTY OF SCIENCE AND ENGINEERING

DOCTOR OF PHILOSOPHY

INTRADAY MACHINE LEARNING FOR THE SECURITIES MARKET

VITALIY MILKE

APRIL 2021

A major issue in financial market trading is knowing when to undertake a transaction for the purpose of generating profit. Trading on large national/international financial systems can be analysed by various sophisticated techniques including neural networks. This work focuses on using deep neural networks and other machine learning tools to analyse financial markets (such as Forex which specialises in trading currencies and is the focus of this work with respect to EUR/US Dollar rates) by identifying patterns in the behaviour of major financial market participants: funds and market makers. Current techniques have drawbacks in that market uncertainty limits the confidence traders have in such predictive aides. This research investigates the use of convolutional neural networks to identify subtle patterns that precede significant movements in financial markets. A new approach is taken which focuses on intraday trading features in order to reduce the risks associated with overnight price gaps which have increased in recent years due to financial instability and the COVID-19 pandemic. Particular emphasis is placed on the advanced preliminary analysis of big financial data, including all minimal price changes (ticks) and all transaction volumes, before feeding them into various neural network architectures. An innovative approach to predicting financial markets is described based on the vector of the probability of significant price movements. This makes it possible for the analysis to easily transition from a standard regression task that predicts prices to a classification task, partly mitigating a common issue of balancing re-training frequency versus application. Critical to this approach is the ability to identify the intensity and intraday volatility based on time intervals between each trade. So time analysis is added to the commonly analysed variables of price and volume to reduce the probability of received losses due to stop-loss orders. The big data processing uses open platforms with GPU processors, and the current work also presents a novel method for reducing the amount of data for training neural networks.

Keywords: Machine Learning, Algorithmic Trading, Neural Networks, Artificial Intelligence, Financial Markets Analysis, Convolutional Neural Networks

Table of Contents

List of Figures 9
List of Tables 12
List of Algorithms (Neural Network Architectures) 13
Glossary and Abbreviations 14
List of Related Publications and Conference Oral Reports 16
1. Introduction 18
1.1. Central hypothesis 19
1.2. Research Questions 20
1.3. Research Aim and Objectives 21
1.4. Research Gap and Motivation 21
1.5. Contributions to Knowledge 23
1.6. Thesis Overview 24
2. Underlying Financial Concepts 25
2.1. Intraday Trading (Day Trading) 25
2.2. Futures Contracts 26
2.3. FOREX 27
2.4. Technical Indicators 27
2.5. Trend-Based and Countertrend-Based Strategies 29
2.6. Liquidity 31
2.7. Volatility 32
2.8. Price Gaps 33
2.9. Stop-Loss 34
2.10. Summary 35
3. Literature Review 37
3.1. ATS Application 37
3.2. Financial Market Analysis 39
3.2.1. Financial Market Structure 40
3.2.2. Time-Series Analysis of Financial Markets 44
3.2.3. Trading Volume Analysis 45
3.3. Machine Learning and Deep Learning Applications 46
3.4. Combination of ML and DL with ATS Practices 47
3.4.1. Multilayer Perceptron 47
3.4.2. Recurrent Neural Networks (RNNs) 49

3.4.3. Convolutional Neural Networks (CNNs) 51
3.4.4. Other Techniques used Big Financial Raw Data 54
3.5. Evolution of Technical Analysis and Japanese Candlestick Charting Techniques. 55
3.6. Comparison of Neural Network Architectures for Financial Market Prediction 58
3.7. Metrics 62
4. Data Processing 64
4.1. Introduction 65
4.2. Conceptual Research Framework 66
4.3. Research Method Tools 67
4.3.1. Financial Market Terminals 67
4.3.2. APIs and High-Speed Connectors for Day Trading 68
4.3.3. Machine Learning Software 71
4.3.4. Environments: Operation Systems, Anaconda, TensorFlow, Keras and Google Colab 72
4.3.5. PC Hardware 74
4.4. Data Collection Tools 74
4.5. Databases Used 76
4.5.1. S&P-500 Index and Stock Market Databases 76
4.5.2. EUR/USD Forex Tick Database 76
4.6. Nuances of Validation and Valuation Neural Networks Models for Financial Markets 78
4.6.1. Overfitting and Validation of Trading ANN Models 78
4.6.2. Nuances of Valuation Neural Networks Models for Financial Markets 82
4.7. Preliminary Data Analysis 82
4.7.1. t-Distributed Stochastic Neighbor Embedding (t-SNE Method) 83
4.7.2. t-SNE for EUR/USD FOREX Database 84
4.8. Data Preparation for Experiments with CNNs 89
4.8.1. Time-Formats Conversions 89
4.8.2. Data Cleaning 91
4.8.3. Normalisation 93
4.9. Preliminary Testing of Models and Topologies of NNs 102
4.9.1. Tests with Multilayer Perceptron for Prediction of S&P-500 Index 103
4.9.2. Initial Tests of CNN Models with Transfer Learning and Fine-Tuning 105
4.10. Novelty of data analysis 108
5. Prediction of the Forex EUR/USD Next Tick Rate 111
5.1. Transforming Raw Data 112
5.2. Prediction of Forex EUR/USD Tick Rates by the Multilayer Perceptron 115
5.3. Prediction of Forex EUR/USD Tick Rates by the CNNs 117
5.4. Evaluation 121

5.4.1. Statistical Evaluation 121
5.4.2. Comparative Evaluation 125
5.4.3. Evaluation against baseline state-of-the-art techniques 128
5.5. Limitations and Discussion 130
6. Management of the Probability of Significant Movement and Stop-loss Prediction 131
6.1. Introduction 131
6.2. Additional Dataset with Japanese Candlesticks 134
6.3. Advanced Data Analysis and Volatility 135
6.4. Fractal Structure of Financial Markets and a Fractal Comparison of Volatility. 139
6.5. Probabilities Vector 141
6.6. Reduction of Calculations 145
6.7. Prediction of the Start of a Significant Movement in Forex EUR/USD Rates by the CNNs 147
6.7.1. Data Used 147
6.7.2. CNN Architectures 148
6.7.3. Dynamic Boundaries and Splitting of the Probability Vector 151
6.7.4. Binary Classification: (Trend or Flat) 156
6.8. Evaluation 158
6.8.1. Statistical Evaluation 158
6.8.2. Comparative Evaluation 160
7. Prediction of the Directions of a Significant Movement Based on the Probabilities Vector 162
7.1. Prediction of the Directions of Significant Movements by the CNNs 162
7.2. Evaluation 164
7.2.1. Statistical Evaluation 164
7.2.2. Comparative Evaluation 166
8. Evaluation and Discussions 173
8.1. Comparison of the Results of Key Experiments 173
8.2. Evaluation of the key experiments 175
8.3. Limitations and Discussion 177
9. Conclusions and Future Work 179
9.1. Review of the Work 179
9.2. Summary of Contributions to Knowledge 180
9.3. Results and Discussion 182
9.4. Future Work and Challenges 183
References 185
Appendix I. Results of experiments on Prediction of Forex EUR/USD Tick Rates 199
Appendix II. Results of the binary classification experiments on Trend or Flat 203
Appendix III. Results of the binary classification experiments on Sell or Wait 205

Appendix IV. Results of the binary classification experiments on Buy or Wait 207

Appendix V. Machine Learning and Deep Learning Methods Abbreviations.....209

Appendix VI. The Connector-1 to MetaTrader-5.....214

Appendix VII. t-SNE-Algorithm-EUR/USD - Visualisation of Multidimensional Data.....239

Appendix VIII. Prediction of S&P-500 Index Prices by Multilayer Perceptron.....275

Appendix IX. Cleaning, Normalisation and Preparation Data for the First Series of the Experiments.....282

Appendix X. Regression Task of the EUR/USD Price Prediction by CNN (One of the 45 Experiment Codes)...299

Appendix XI. Evaluation of the Regression Task by Recurrent Neural Networks (GRU and LSTM).....314

Appendix XII. Volatility Analysis.....321

Appendix XIII. CNN Binary Classification on Trend or Flat.....352

Appendix XIV. Evaluation of the Classification Task by Recurrent Neural Networks (GRU and LSTM).....371

In addition to the appendices, the code used in this research was published on Github at the following link:

https://github.com/vmilke/IntradayML_Thesis.git

The Big Financial Data used in this research is stored at:

https://drive.google.com/drive/folders/1PH1Cam0Z2nz07cOF6jXba-tQG8RveeeO?usp=sharing

List of Figures

Figure 1.1: Stock Market management (The Economist, 2019). 20
Figure 2.1: Two Exponential Moving Averages (EMA) with Apple Inc shares in May 2020. Screenshot from (Interactive Brokers LLC, 2022). 28
Figure 2.2: The MACD technical indicator with EUR/USD Forex in May 2020. Screenshot from (Dukascopy Swiss Banking Group, 2022b) 29
Figure 2.3: Trend following trading with shares of The Boeing Company in January 2019. Screenshot from (Interactive Brokers LLC, 2021). 30
Figure 2.4: Countertrend strategy with shares of Microsoft Corporation in May 2020. Screenshot from (Interactive Brokers LLC, 2022). 30
Figure 2.5: Apple Inc shares price gaps. Screenshot from (Interactive Brokers LLC, 2022). 33
Figure 3.1: Average decision terms for HFT, Arbitrage, Intraday Trading, Middle- and Long-term investments (logarithmic horizontal axis). 38
Figure 3.2: An example of a time-series chart with integrated data (15 min) of EUR/USD Forex. Screenshot from (Dukascopy Swiss Banking Group, 2022b) 40
Figure 3.3: Number of research papers with published open-source code (Jiang, 2020). 41
Figure 3.4: An example of a real chart of the Forex EUR/US Dollar which is based on raw data of ticks and volume. Screenshot from (Dukascopy Swiss Banking Group, 2022b). 43
Figure 3.5: Tick chart (price & volume) of the Forex EUR/US Dollar. Tick size is the smallest upward or downward price movement. Screenshot from ((Dukascopy Swiss Banking Group, 2022b) 43
Figure 3.6: The relation of volume and absolute value of the relative value of the price change (Blume, Easley and O'hara, 1994). 46
Figure 3.7: Multilayer Perceptron Architecture Example (LeDell, 2016). 48
Figure 3.8: An unrolled Recurrent Neural Network (Olah, 2015). 49
Figure 3.9: The repeating module in an LSTM contains four interacting layers (Olah, 2015). 51
Figure 3.10: Convolutional Network (Saha, 2018). 52
Figure 3.11: Support and resistance lines and price-break-out points based on Technical analysis (Jitpakdee and Pravithana, 2017). 55
Figure 3.12: Some examples of simple Japanese candlestick patterns (Nison, 1994). 56
Figure 3.13: An example of a formal description in words of one of the Japanese candlestick patterns (Chen and Tsai, 2020) 57
Figure 3.14: An example of a formal description in numbers of one of the Japanese candlestick patterns (Nison, 2011). 57
Figure 3.15: Sine wave middle-term (4 hours) EUR/USD Forex chart in May 2020. Screenshot from (Dukascopy Swiss Banking Group, 2022b). 59
Figure 3.16: Sine wave model of middle-term price movement with support and resistance levels (green lines) of prices. 60
Figure 3.17: Stepped patterns in the tick-term GBP/USD Forex chart in February 2020. Screenshot from (Dukascopy Swiss Banking Group, 2022b). 61
Figure 3.18: Stepped model (patterns) of short-term price movement. 61
Figure 3.19: Hypothetical growth of US $10,000 invested in the S&P-500 Index from 1st January 1980 to 31st March 2020 (Fidelity Investments Inc., 2020). 62
Figure 4.1: Research Conceptual Framework. 67
Figure 4.2: Interconnections between the modules. 69
Figure 4.3: Order book. Screenshot from (CQG Inc, 2022) 75

Figure 4.4: S#.Data terminal. Screenshot from (Stock Sharp, 2021). 75
Figure 4.5: Price and volume graph of EUR/USD Forex from 01.01.2018 util 30.04.2018. 77
Figure 4.6: Price and volume graph of EUR/USD Forex: first 400 ticks. 78
Figure 4.7: Indicator of overfitting (Gulli and Pal, 2017). 79
Figure 4.8: Splitting an original dataset into training, validation and testing parts 80
Figure 4.9: Cross-validation on a rolling basis (Bergmeir, Hyndman and Koo, 2018). 81
Figure 4.10: Cross-validation with the 'rolling training set' (Jiang, 2020). 81
Figure 4.11: t-SNE for raw EUR/USD Forex Dataset. 85
Figure 4.12:t-SNE for EUR/USD Forex Dataset with the intervals between adjacent ticks. 85
Figure 4.13: The distribution of the patterns of price movements for the first half of 2019. 87
Figure 4.14: t-SNE projection for a random 10,000 rows with significant patterns of price movements. 88
Figure 4.15: t-SNE projection for a random 100,000 rows with significant patterns of price movements. 88
Figure 4.16: Falsely predicted prices based on missing data with a large amplitude (red and orange graphics), as opposed to real prices (blue and green). 92
Figure 4.17: Distribution of 1,000 random intervals between ticks. 94
Figure 4.18: Distribution of 10,000 random intervals between ticks. 94
Figure 4.19: Distribution of 100,000 random intervals between ticks. 95
Figure 4.20: Distribution of random 1,000 ticks' volume. 98
Figure 4.21: Distribution of random 10,000 ticks' volume. 98
Figure 4.22: Distribution of random 100,000 ticks' volume. 99
Figure 4.23: Ask distribution. 100
Figure 4.24: An inconsistency of the dimensionality of normalisations. The green line is normalised volume. The red line is normalised price. 101
Figure 4.25: Actual and predicted prices of the S&P-500 index (normalised). 104
Figure 4.26: Inaccuracy, %. 104
Figure 4.27: Training and validation absolute errors. 105
Figure 4.28: Top models in last 5-10 years in ImageNet competitions (Spark, 2017). 106
Figure 4.29: Part of the new VGG-18, which has to be trained (Sozikin, 2022) 108
Figure 5.1: Two hundred ticks sliding window (images) for finding patterns by two-dimensional CNNs. 113
Figure 5.2: Two hundred ticks sliding window (2D-array) for finding patterns by one-dimensional CNN. 114
Figure 5.3: White noise structure. 116
Figure 5.4: Error chart for bid and ask EUR/USD Tick Rates (Prices) for 120,000 random ticks. 120
Figure 5.5: Error chart for bid and ask EUR/USD Tick Rates (Prices) for 1,800 random ticks. 120
Figure 5.6: Error chart for bid and ask EUR/USD Tick Rates (Prices) for 300 random ticks. 121
Figure 5.7: Training and validation mean absolute error (MAE) graph. 122
Figure 5.8: Real and predicted bid and ask EUR/USD tick rates (prices) for 1,000 random ticks. 123
Figure 5.9: Real and predicted bid and ask EUR/USD tick rates (prices) for 300 random ticks. 124
Figure 5.10: GRU Training and validation mean absolute error (MAE) graph. 126
Figure 5.11: LSTM Training and validation mean absolute error (MAE) graph. 126
Figure 5.12: Comparative evaluation of the quality of training and validation of the neural network architectures (CNN, LSTM, GRU). 127
Figure 5.13: Comparison and evaluation of MAE results. 129
Figure 6.1: Japanese candlestick types. 135
Figure 6.2: Daily volatility of the Forex EUR/USD for 2018 and 2019. 136
Figure 6.3: Daily volatility of the Forex EUR/USD from 2018 till 2020. 136
Figure 6.4: Hourly volatility of the Forex EUR/USD for 2018 and 2019. 137
Figure 6.5: Hourly volatility of the Forex EUR/USD from 2018 to 2020. 138

Figure 6.6: Boxplot of the hourly volatility of the Forex EUR/USD from 2018 to 2020. 138
Figure 6.7: 30-minute volatility of the Forex EUR/USD from 2018 to 2020. 139
Figure 6.8: Fractal lines of the volatility. 140
Figure 6.9: Comparisons of total volatilities at different time frames. 140
Figure 6.10: Probabilities vector. 141
Figure 6.11: Margin distribution. 142
Figure 6.12: Daily number of ticks (settlements) in March 2020 (working days). 144
Figure 6.13: Initial distribution of the number of values of this probability vector. 144
Figure 6.14: Distribution of the number of values of this probability vector without the zero level. 145
Figure 6.15: Distribution of this probability vector with only significant financial market movements. 146
Figure 6.16: One of the most valuable points of entry into the market without significant retracements. 148
Figure 6.17: Distribution of the probability vector over on each day for the first half of 2020. 151
Figure 6.18: A typical probability vector with a correct prediction. 152
Figure 6.19: A probability vector, which predicted up-trend but mixes neighbouring fields. 153
Figure 6.20: A probability vector, which predicted down-trend but mixes neighbouring fields. 154
Figure 6.21: A probability vector, which predicted flat but mixes neighbouring fields. 154
Figure 6.22: Splitting a complex multi-parameter classification task with dynamic bounds of the probability vector into a sequence of simpler tasks. 155
Figure 6.23: Probability (Trend or Flat) and absolute prediction error for each tick. 157
Figure 6.24: One of the accuracy charts for recognising trend or flat. 158
Figure 6.25: One of the recall charts for recognising trend or flat. 159
Figure 6.26: Binary cross-entropy loss-function 159
Figure 6.27: Comparative evaluation of the quality of training and validation of the neural network architectures (CNN, LSTM, GRU) for binary classifier (Trend or Flat). 161
Figure 7.1: Probabilities (Sell or Wait), (Buy or Wait) and absolute prediction error for each tick. 163
Figure 7.2: One of the accuracy charts for recognising (Sell or Wait) signals. 164
Figure 7.3: One of the accuracy charts for recognising (Buy or Wait) signals. 165
Figure 7.4: Binary cross-entropy loss function for recognising Sell or Wait and Buy or Wait signals. 165
Figure 7.5: Accuracy of the LSTM neural network classifier (Sell or Wait). 169
Figure 7.6: Accuracy graph of training the GRU neural network classifier (Sell or Wait). 169
Figure 7.7: Precision graph of training the GPU neural network classifier (Sell or Wait). 170
Figure 7.8: Recall graph of training the LSTM neural network classifier (Buy or Wait). 170
Figure 7.9: Comparative evaluation of the quality of training and validation of the neural network architectures (CNN, LSTM, GRU) for binary classifier (Sell or Wait). 171
Figure 7.10: Comparative evaluation of the quality of training and validation of the neural network architectures (CNN, LSTM, GRU) for binary classifier (Buy or Wait). 171
Figure 8.1: Comparative evaluation of the neural network architectures 174
Figure 8.2: Predicting the trend and direction of subsequent movements (Accuracy, %). 177

List of Tables

Table 4.1: Period for learning a Convolutional Neural Network (CNN) by Central Processing Unit (CPU). 73
Table 4.2: Period for learning a CNN by Graphics Processing Unit (GPU). 73
Table 4.3: PC hardware with GPU used. 74
Table 4.4: Patterns of important price changes. 86
Table 4.5: Size of EUR/USD Forex Raw Tick Financial Data 2018...2020. 89
Table 4.6: Raw data of EUR/USD Forex 2018. 90
Table 4.7: Converting the dates and times to 'datatime64' type. 90
Table 4.8: Example of the identification of incorrectly collected transaction times. 93
Table 4.9: Input data where the inter-tick intervals are more than 100,000. 97
Table 5.1: Integrated parameters and error metrics results. 121
Table 5.2: Comparative evaluation of the quality of training and validation of the neural network architectures (CNN, LSTM, GRU). 127
Table 5.3: Mean absolute error (MAE) of baseline state-of-the-art techniques. 128
Table 6.1: Japanese candlesticks and big financial tick data sizes. 134
Table 6.2: The probability vector with only significant financial market movements. 146
Table 6.3: Hyperparameters, training parameters and error metrics results. 158
Table 6.4: Comparative evaluation of the quality of training and validation of the neural network architectures (CNN, LSTM, GRU) for binary classifier (Trend or Flat). 160
Table 7.1: Training parameters, hyperparameters and error metrics for binary classifications: (Sell or Wait) and (Buy or Wait). 164
Table 7.2: Comparative evaluation of the quality of training and validation of the neural network architectures (CNN, LSTM, GRU) for binary classifiers (Sell or Wait) and (Buy or Wait). 166
Table 8.1: Comparative evaluation of the Accuracy of training and validation of the neural network architectures (CNN, LSTM, GRU) for key binary classifiers on the testing datasets. 173
Table 8.2: Classification accuracy for the EUR/USD currency pair in determining the direction of the next price movement (Galeshchuk and Mukherjee, 2017). 175
Table 8.3: Classification Accuracy for the EUR/USD currency pair to determine Trend-Flat and Buy-Sell-Wait options of the next price movement (Moghaddam and Momtazi, 2021). 176

List of Algorithms (Neural Network Architectures)

Algorithm 4.1: The VGG-16 topology (Simonyan and Zisserman, 2014). 107
Algorithm 5.1: MLP with four hidden layers. 116
Algorithm 5.2: The four hidden layers CNN. 117
Algorithm 5.3: Three layers Gated Recurrent Unit (GRU) neural network for regression. 125
Algorithm 6.1: Eleven layer CNN for the probability vector analysis. 149
Algorithm 6.2: Thirteen layer CNN for the probability vector analysis. 150
Algorithm 7.1: Gated Recurrent Unit (GRU) neural network architecture for classification. 167
Algorithm 7.2: Long short-term memory (LSTM) neural network architecture for classification. 168

Glossary and Abbreviations

ANN	Artificial Neural Network
API	Application Programming Interface
ARIMA	Autoregressive Integrated Moving Average
ATS	Algorithmic Trading System
BB	Bollinger Bands
CCI	Commodity Channel Index
CME	Chicago Mercantile Exchange
CNN	Convolutional Neural Network
CQG	Commodities/Comprehensive Quote Graphics
DL	Deep Learning
EMA	Exponential Moving Average
ETF	Exchange Traded Funds
ETS	Exponential Smoothing
FQA	Finance Quantitative Analysis
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HFT	High-Frequency Trading
HSC	High-Speed Connector
IB	Interactive Brokers
ICTT	Integrated Client Trading Terminals
LSTM	Long Short-Term Memory
MA	Moving Average
MACD	Moving Average Convergence/Divergence
MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multilayer Perceptron

MSE	Mean Squared Error
MT-5	MetaTrader-5
OLAP	Online Analytical Processing
OTTM	Online Trade Test Mode
QAF	Quantitative Analysis in Finance
ReLU	Rectified Linear Unit (function)
RNN	Recurrent Neural Network
ROC	Return on Capital
ROCh	Rate of Change
ROIC	Return on Invested Capital
RSI	Relative Strength Index
SL	Stop-Loss
SVM	Support Vector Machines
TPU	Tensor Processing Unit
t-SNE	t-Distributed Stochastic Neighbor Embedding
TF	Target Function
TVA	Trading Volume Analysis
VSA	Volume Spread Analysis

List of Related Publications and Conference Oral Reports

- Milke, V., Luca, C. and Wilson, G., 2017, May. Minimisation of parameters for optimisation of Algorithmic Trading Systems. In *2017 International Conference on Optimization of Electrical and Electronic Equipment (OPTIM) & 2017 Intl Aegean Conference on Electrical Machines and Power Electronics (ACEMP)* (pp. 1114-1119). IEEE. Available at: <https://ieeexplore.ieee.org/abstract/document/7975121> [Accessed 27 March 2021].

- Milke, V., Luca, C., Wilson, G. and Fatima, A., 2020. Using Convolutional Neural Networks and Raw Data to Model Intraday Trading Market Behaviour. In *the 12th International Conference on Agents and Artificial Intelligence (ICAART)* (2) (pp. 224-231). Available at: <https://www.scitepress.org/PublicationsDetail.aspx?ID=CqOGCTFIr30%3d&t=1> [Accessed 27 March 2021].

- Milke, V., 2020. Artificial General Intelligence (AGI) from Theory to Practice. First Draft of the Road Map. In *Artificial Intelligence Journey - 2020*. [online] Available at: <https://www.youtube.com/watch?v=3hSzLTowZRA&feature=emb_logo> and <https://ai-journey.ru/en/media/broadcast#galery-video/363> [Accessed 19 March 2021].

- Milke, V., 2020, February. INSTICC conference oral report. In *the 12th International Conference on Agents and Artificial Intelligence ICAART-2020, Valletta, Malta*.

- Milke, V., 2020. Ethics in the National Strategy for Artificial Intelligence including Finance. In: RANEPA, ed. (2020), Ethics and digital: ethical issues of digital technologies. RANEPA, Moscow (pp. 79-80). [online] Available at: <https://ethics.cdto.center/3_9> [Accessed 27 March 2021].

- Milke, V., 2019. Artificial Intelligence. In: Shklyaruk, M., ed. (2019), The state as a digital platform: people and technologies. In *the 10th Gaidar Economic Forum, Moscow* (pp. 21-28). [online] Available at: <https://www.ranepa.ru/images/News/2019-01/16-01-2019-GovPlatform.pdf> [Accessed 19 March 2021].

INTRADAY MACHINE LEARNING FOR THE SECURITIES MARKET

VITALIY MILKE

COPYRIGHT

Attention is drawn to the fact that copyright of this Thesis rests with:

(i) Anglia Ruskin University for one year and thereafter with

(ii) Vitaliy Milke

This copy of the Thesis has been supplied on condition that anyone who consults it is bound by copyright.

This work may:

(i) be made available for consultation within the University Library, or

(ii) be lent to other libraries for the purpose of consultation or may be photocopied for such purposes.

(iii) or be made available in Anglia Ruskin University's repository and made available on open access worldwide for non-commercial educational purposes, for an indefinite period.

1. Introduction

In recent years, leaders of high-tech industries and universities have developed powerful new machine learning technologies (Abadi, et al., 2016; Chollet, 2022; Paszke, et al., 2017). The software is able to tackle problems that were challenging for previous technologies. One of the areas where Machine Learning (ML) and Deep Learning (DL) techniques can be applied is trading in financial markets - in particular, the development of Algorithmic Trading Systems (ATS) with a ML / DL basis. Due to perpetually improving software modules (frameworks) for deep learning, it is becoming easier to implement, train and test new models and architectures of Deep Neural Networks.

Some of the largest investment banks, such as JPMorgan and Goldman Sachs, have declared themselves technology companies. They recommended that their analysts and employees have training in Python coding, mainly for ATS creation (Noonan, 2018). By 2014, the Hong Kong Stock Exchange's trading floor accounted for less than 1% of the exchange trading volume. On October 27, 2017, it was closed due to the transition to online electronic trading. The trading floor has since been renamed Hong Kong Connect Hall and converted into a museum.

There are many kinds of ATS that are used to predict stock market prices. These are automated systems and technologies/techniques and include (Lien, 2008; Kaufman, 2013; Lin, 2013):

- Intraday trading or day trading (trading that takes place within 24hrs)
- High-frequency trading (HFT)
- Arbitrage
- Swing trading
- Long-term investments
- Technical indicators/stochastic theory
- Volume spread analysis (VSA)
- Elliot wave principles
- Japanese candlestick charts
- Mathematical functions including uninterrupted analogue functions
- Statistical-based methods.

This research focuses only on day trading due to the difficulties inherent in the other systems, including infrastructural expense, an inapplicability of the applied models, use of poor-quality data, or unverifiable data not in the public domain.

The focus of this research project is on the ML analysis of publicly-available stock market data as the primary source of information with an over-arching objective to successfully predict stock market behaviour. Existing automatic trading technologies (except for HFT) classify current behaviour based on historical data sets which are average time series and do not reflect subsequent changes in market trends during a one-day trading session. This means the models on which the systems are based are unverifiable, the results are often unreliable and they have a limited time and market-location window where they are potentially useful.

This research will connect the reliable basic principles of ATS with ML libraries. As a consequence, the main aim is to produce an advanced operating system for trading in financial markets. One of the challenge to address is that the financial market is constantly changing. Therefore, it is not enough to train an implementation of an ML model once. This model should have the capability to change and retrain periodically.

Ethical Considerations: all collected data is in the public domain, and no private data by which an individual could be recognised (other than the researcher) will be stored.

1.1. Central hypothesis

There is a widespread belief that algorithms drive markets and that machines determine the majority of significant movements (Vezeris, Kyrgos and Schinas, 2018). This opinion arose due to the comparison of the exchange turnover initiated by robots and humans. Due to the significant turnover generated by trading robots, a high level of market liquidity is maintained. However, according to statistical research (The Economist, 2019), only about 35% of assets are controlled by ATS. The remaining 65% of the shares are managed based on decisions made by people. Of these, 40% of financial assets are passive investments (without significant operational management) of enterprises and organisations, such as governments and insurance companies, as well as private investors. Thus, the number of shares actively managed by humans is significantly higher than that of algorithmic systems driven ones (Figure 1.1). It is logical to conclude that humans initiate most of the significant movements in financial markets.

During trading with significant amounts of financial assets on an exchange, humans display particular patterns of behaviour and also can create trading algorithms in which such patterns of behaviour are entered in the process of trading in financial markets. Thus, if these behaviour patterns exist, they can be recognised using artificial neuron networks (Bouchaud, et al., 2018). This is the central hypothesis of the current research, the confirmation of which we will assess by analysing data and training neural networks.

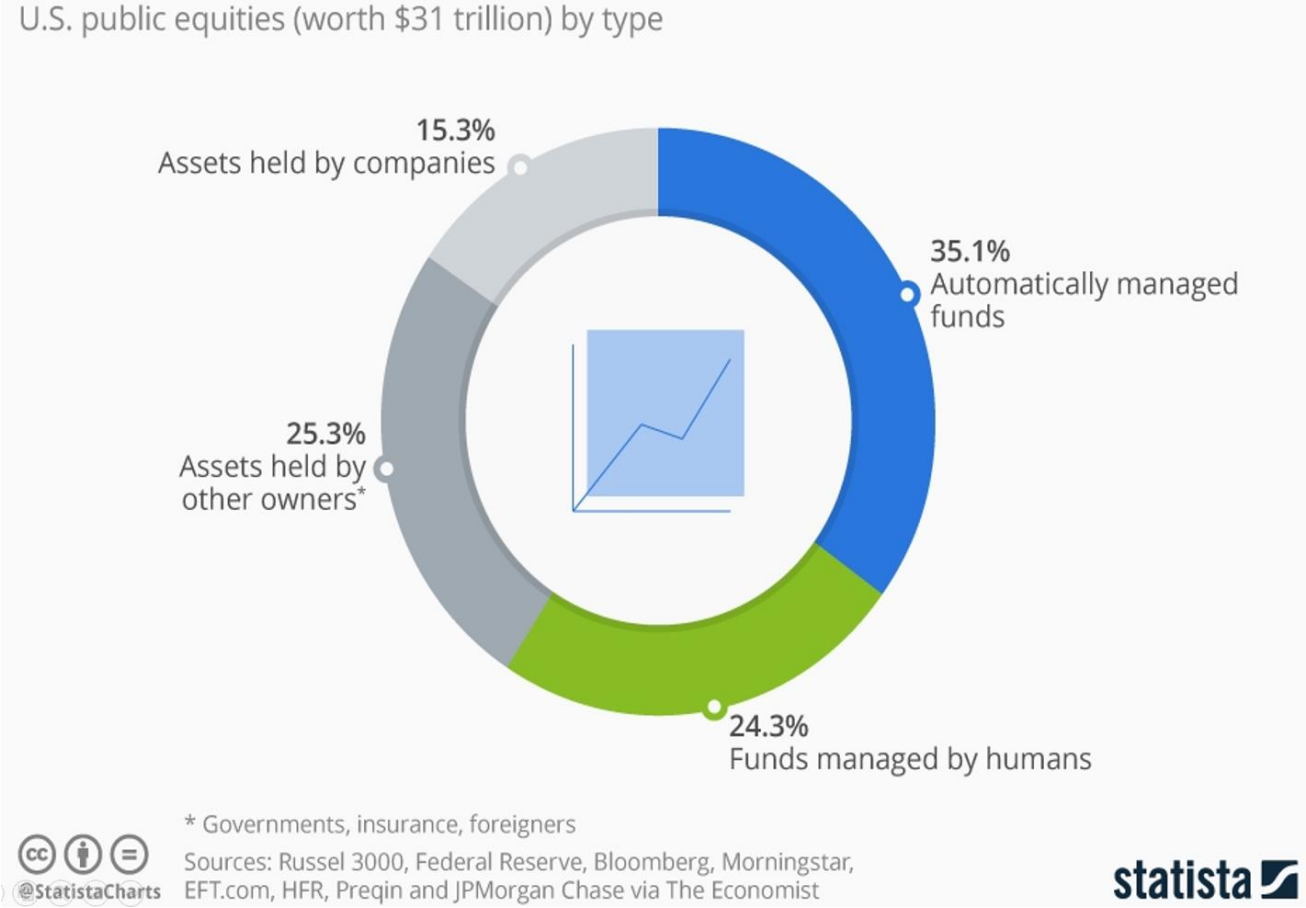


Figure 1.1: Stock Market management (The Economist, 2019).

1.2. Research Questions

The main research questions this study aims to answer are:

RQ1. How can a high-speed intraday ATS be successfully created for use with several stock exchanges and trading platforms?

RQ2. How can a ML model be developed that accurately describes continually changing stock market behaviour through the use of a adapting learning strategy?

RQ3. How can existing software be used in conjunction with publicly available stock market data to implement an ATS that successfully predicts future market behaviour?

RQ4. What techniques and/or strategies can be adopted to test, optimise, improve and evaluate the implemented system?

1.3. Research Aim and Objectives

The aim of this research is to improve the performance of intraday trading. The research will investigate deep learning algorithms to attempt to achieve increased accuracy in intraday trading predictions, identify market entry points with the highest probability of significant movements and reduce losses from the execution of stop-loss orders. To achieve this aim, the research objectives are defined below:

OB-1. To identify the best performing machine learning algorithms used for intraday trading and the most relevant data collection software following a thorough survey of the state-of-the-art.

OB-2. To formulate a sequence of actions that allow the neural networks to adapt to dynamically changing financial data for minimising systematic risks.

OB-3. To break up the complex task of creating an intraday trading system into its constituent parts and identify standard error metrics and loss functions for each. These will be tested further and evaluated against existing state-of-the-art research separately.

OB-4. To test and evaluate different models, ML frameworks, tools and data collection software by preprocessing and analysing publicly-available financial market data in order to predict the most probable directions of significant market movements and identify market entry points with minimal risk.

1.4. Research Gap and Motivation

The use of machine learning methods and applied artificial intelligence in financial markets is of great interest to researchers, banks and asset management companies. Active use of neural networks is gradually replacing the classical methods of observational Technical analysis and Japanese charting techniques and often complements the classical statistical methods of quantitative research of financial markets. Many recent publications have tried to

predict price movement based on historical integral data (Aggarwal and Sahani, 2020; Gudelek and Ozbayoglu, 2020; Yıldırım, Toroslu and Fiore, 2021). This type of data is mainly daily or hourly Japanese candles that do not show detailed price movements; thus, crucial information is lost. In the last couple of years, some papers have investigated the use of neural networks to analyse shorter time frames such as 30, 15 and 5 minutes, and they also try to predict the future price of one or more future Japanese candles of the respective time frames (Jiang and Liang, 2017; Orimoloye, et al., 2020). A similar approach is also relevant for medium-term or long-term investments, which implies assessing the Return on Invested Capital (ROC or ROIC). An attempt to analyse raw big financial tick data (that is not integrated data), as used in this research, has been done in some papers that explore High-Frequency Trading (HFT) methods that primarily use quantitative statistical analysis (Hirano et al., 2020; Petukhina, Reule and Hardle, 2021).

Any transaction in the financial market consists of three parts: the point of entry into the market, tracking the potential of an open position, and the point of exit from the market. This research uses various neural network architectures to enter the market and track the potential of an open position. The majority of experiments in this research are based on the analysis of microstructural fluctuation patterns of financial markets in the form of ticks, which are rarely used in scientific papers since they constitute big data that is difficult to analyse. However, this research does not use ATS methods to exit the market which are well described in the scientific and practical literature on trading and are based on risk management techniques (Grant, 2011).

For the practical use in intraday trading, which has become popular in recent years, the price prediction for a certain period in future is less important than the prediction of the direction of a potential significant movement; in other words, the determination of the start points of intraday trends with the highest probability. This problem of determining the trend (direction) using raw (non-integrated) data has not been investigated yet, and this research is trying to address it.

The motivation for this research is to explore methods for determining the beginning of intraday trends with the highest probability using raw data, which will aid further researchers in developing these approaches and allow developers to use this research, as part of further creation of successful intraday trading strategies.

1.5. Contributions to Knowledge

The current research is novel in the following ways:

- It proposes a system that integrates convolutional neural networks with different architectures and an original approach to classifying potential expected market movements using a probability vector. This novel probability vector makes it possible to transit from the regression task in ML to the task of classifying potential prospective movements of the price of financial assets. This transition enables a focus on finding better entry points to the financial market.
- It negates the need to predict a specific price of a financial instrument after a particular time; instead, it predicts the direction of recoilless movement and its depth to calculate the probability vector.
- It uses public domain historical raw data in the form of ticks (multidimensional time-series data with irregular timestamps) to develop a ML system (principally based on existing neural network technology). The difference between raw data and regular time series is explained in Section 3.2.
- For accelerating calculations and analysis of big financial data, an original approach is proposed to reduce the size of the big data by preliminary analysis of potential prospective movements and to remove flat movements, which are noise for a trading system.
- The Target Function (TF) of the utilised ANN models adopts a minimising stop-losses (SL) strategy rather than the maximising profits approach that is most popular with other researchers. The minimising stop-losses order execution is set at the stage of data preprocessing and does not require additional calculations in the process of neural network training. Thus, calculations can be carried out in batches and in parallel. Profit maximisation as a goal can lead to unstable results and large drawdowns in trading capital, which certainly increases the risks. For example, often, most of the gross-profits are made from one or two random trades. In contrast, minimising SL allows the model to achieve a graph of capital growth without significant local dips.
- Contrary to the majority of approaches in financial market research, this research uses Recurrent Neural Network (RNN) architectures only to validate the results obtained with convolutional neural networks.
- The integrated system has been developed in such a way that it is adaptable to changing market trends and specific market features.

1.6. Thesis Overview

This thesis has 9 chapters, as described below.

Chapter 1 introduces the aim, objectives and motivation of this research.

Chapter 2 presents the key financial concepts used in this research at various stages.

Chapter 3 reviews the state-of-the-art papers in the algorithmic trading systems and financial market analysis domains, as well as the application of well-known neural network architectures to trading in the financial markets.

Chapter 4 describes the preliminary data analysis, preparation, collection and visualisation data used in this research.

Chapter 5 describes intermediate experiments to predict the Forex EUR/USD next tick rate.

Chapter 6 describes the first key experiments to predict the probability of the following significant movement of the Forex EUR/USD rate and identify market behaviour as a trend or flat.

Chapter 7 describes the second and third key experiments to predict the directions of the following significant financial market movements (up or down) of the Forex EUR/USD rate.

Chapter 8 summarises, analyses, and evaluates the results of the key intraday trading experiments.

Chapter 9 presents the conclusions and future work.

2. Underlying Financial Concepts

This chapter describes the key concepts, fundamental terms and definitions that are used in this research at various stages. It also details two primary financial trading approaches: trend-based and countertrend-based strategies (Section 2.5). The intraday trading definition is given, in which all positions in financial instruments are closed at the end of the trading day (Section 2.1). Because this research uses futures and Forex databases, this chapter also describes the differences between exchange-traded futures contracts and Forex currency trading (Sections 2.2 and 2.3). The chapter describes the disadvantages of trading based on financial indicators (Section 2.4) and reviews the relevant essential parameters of the securities markets, such as liquidity, price gaps, volatility and stop-loss (Sections 2.6, 2.7, 2.8 and 2.9).

2.1. Intraday Trading (Day Trading)

Intraday trading is characterised by active trading throughout the working day; however, all positions should be closed before the financial market session has ended (U.S.Securities and Exchange Commission (SEC), 2022). This is one of the vital concepts and principles on which this research is based. All subsequent experiments will be based on the axiom that no positions in financial assets or instruments are held overnight.

In contrast to one of the well-known trader's proverbs, 'A trend is your friend' around 70% of the time market prices trade in a sideways or range-bound pattern (Weissman, 2005). However, this assertion is correct only for long or medium-term investment. The range-bound pattern for medium-term investment is the trend for intraday and short-period trading. Thus, short-period activity has more potential points of entry into the financial market. This assertion will be proven in the volatility analysis (Section 6.3). For short-period trading, it is not crucial whether the market is going up or down at any particular moment, but it should not stay on the same level for successful intraday trading. Thus, it is easier for short-term trading to collect a large number of potentially profitable entry points for training neural networks. Day trading and short-period data is suitable for training Deep Learning (DL) models because they allow the system to analyse a large number of data points entering into any one particular financial market. Consequently, it allows the model to be trained with high-quality data because a DL model requires plenty of data.

During a short-term period, such as a day, price instability does not reflect any fundamental factor, such as the profit of the particular company or budget deficit of a particular country. Excluding the previously known moments of publication of macro-economic reports and statistics, the financial market's shift is entirely formed by the market participants' attitude. Nevertheless, it is possible to recognise the behaviour of the market participants through the use of ML / DL methods, which can identify the behavioural patterns. These circumstances are ideal for DL.

2.2. Futures Contracts

Futures (or futures contracts) are an exchange-traded financial derivative pegged to the price of an underlying asset such as stocks, metals, oil, sugar, prices of currency pairs and other exchange-traded commodities or financial indexes. A futures contract is almost always a deal that involves a specified delivery date in the future of an underlying asset. This research includes preliminary experiments trading the S&P-500 index, which is a set of exchange futures contracts on this financial index. The S&P-500 index is calculated based on the price of the 500 largest companies, and the futures contracts for this index are traded on the Chicago Mercantile Exchange in parallel and have different expiration dates. This research analyses historical data published specifically as prices of the futures contracts.

Each of the underlying asset's main parameters, such as quality, packaging, labelling, expiration date, etc., have already been determined when this futures contract is issued. The underlying asset must also be traded on the exchange (not necessarily on the same exchange).

A futures contract is similar to a betting deal. During the trading day, the price of the underlying asset changes and the exchange continuously calculates the difference between the purchase price and the current price and transfers this difference from the losing party to the winning party at that moment.

In order for the losing party to pay this variable difference, both parties must have on their account the what is known as guaranteed collateral in the form of money in the brokerage account at the conclusion of the deal. This amount is usually several times less than what is required to buy the same quantity of the underlying asset on the spot market. Thus, when trading futures contracts, the parties seem to trade with leverage, since they only require for guaranteed collateral around 15% of the spot value. The result of this internal leverage is an

increased risk, which must be taken into account in the subsequent volatility analysis in this research.

An essential consequence of trading on one exchange is the known volume of these futures contracts, which is concluded in each transaction. Therefore, it possible to calculate the intensity of trading by analysing the volume of transactions (deals) at any given moment.

The key experiments of this research use the historical data of the EUR/USD currency pair in the Forex market.

2.3. FOREX

The Forex (foreign exchange market) is a system of interconnected trading platforms (trading floors) that brings together foreign banks and brokerage firms to trade currency pairs, precious metals, oil and other financial instruments. Thus, it is not an exchange trade, as it is not regulated or controlled by exchange rules. As a result, all interested parties know the prices of currency pairs; however, information on the total volume of all transactions that have taken place on a particular currency pair at a particular moment in the time is unknown. This makes it very difficult to analyse market data, even using neural networks. However, some major currency trading platforms provide information on the volume of transactions made on these platforms.

These foreign banks and brokerage firms provide clients with an opportunity to trade currency pairs with very high leverage, which makes Forex trading very risky. There are liquid and non-liquid currency pairs. The most liquid currency pair is EUR/USD.

The key experiments of this research use the historical data of the EUR/USD currency pair in the Forex market.

2.4. Technical Indicators

For approximately forty years (from the 1970s to 2010), the majority of professional traders actively used what are known as technical indicators for algorithmic trading, and this was the gold standard for trading in the financial markets. Many researchers still use them as input data to neural networks. However, this section describes the weaknesses of using technical indicators and explains why the author does not use them as inputs for training neural networks. However, the concept of technical indicators should be included as some of

the state-of-the-art approaches published in existing papers are used to validate the results of this research (Yıldırım, Toroslu and Fiore, 2021).

Technical indicators are functions built based on historical data (prices, volumes, open interests of futures contracts) that have the goal of predicting the future price and movement direction of particular financial instrument (Chan, 2019). There are well-known and popular technical indicators, such as the Moving Average (MA) (Figure 2.1) and the Moving Average Convergence/Divergence (MACD) (Vezeris, Kyrgos and Schinas, 2018) (Figure 2.2). Also, professional traders develop unique and complex proprietary technical indicators for their own specific professional activity. However, the vast majority of these are different combinations of average statistical values for a certain period in the past. Financial indicators integrate and average data, and as a result, valuable information about the behaviour of market participants is lost. This averaging was done with one goal - to reduce the number of calculations so that the computers of earlier generations could process them. Modern hardware and neural network algorithms can identify patterns in raw data without preliminary data averaging.

Figure 2.1: Two Exponential Moving Averages (EMA) with Apple Inc shares in May 2020. Screenshot from (Interactive Brokers LLC, 2022).

Figure 2.2: The MACD technical indicator with EUR/USD Forex in May 2020. Screenshot from (Dukascopy Swiss Banking Group, 2022b).

Due to their using averaged data and lagging signals, the use of technical indicators in professional trading is decreasing. However, technical indicators can be added as inputs to neural networks in addition to integral data to improve the quality of some models' predictions.

2.5. Trend-Based and Countertrend-Based Strategies

There are two types of trading strategies in financial markets. A trend-based strategy is often referred to as a trend following trading, and it is commonly used for medium or long-term trading. Figure 2.3 shows a typical chart used for a trend-based strategy. This slow form of trading has been popular for more than 80 years, but it has drawbacks. It is challenging to determine when a trend begins and when it ends. Also, the increased market volatility in recent years does not aid this strategy. In this type of trading strategy, various technical indicators and RNNs are often used.

Figure 2.3: Trend following trading with shares of The Boeing Company in January 2019. Screenshot from (Interactive Brokers LLC, 2021).

Figure 2.4: Countertrend strategy with shares of Microsoft Corporation in May 2020. Screenshot from (Interactive Brokers LLC, 2022).

In contrast, the countertrend-based strategy, also known as Swing trading, looks for market turning points to open long (buying) or short (selling) positions (Figure 2.4). It is a dynamic type of trading with increased risk, but with greater efficiency, which uses short-term trading for a duration of no more than 2-3 days.

For these reasons, the countertrend-based strategy for intraday trading has been selected to be used in this research.

2.6. Liquidity

Slippage is the difference between the price at the time of sending an order to the exchange and the actual trade price. Slippage can arise for several reasons, for example, the speed of communication via the Internet and a lack of volumes to complete the order.

Liquidity is the possibility to open or to close a position without slippage. In contrast, “illiquidity in the context of short-term systems trading means that the total net profits generated by a system are insufficient to compensate for the wide intraday slippage (or bid/ask spreads) plus commissions.” (Weissman, 2005). Moreover, liquidity becomes even more critical when trading time frames are shortened.

The testing of the DL models that the author will develop will operate with only the least volume (one lot) of liquid financial Instruments (Futures contracts) and will occur in an Online Trade Test Mode (OTTM) with a supposition that the liquidity is absolute. In other words, OTTM will not consider any slippage. Live testing will use the most liquid and well-known Futures contracts such as S & P-500 futures or EUR/USD currency pairs on FOREX.

Future researchers who use the results from this research must understand these liquidity limitations. All obtained results from the experiments on historical data imply absolute liquidity. This can apply to the S&P-500 futures contracts or the EUR/USD currency pair on Forex through a respectable broker but only using small lots. For example, using the results of this research for tens of million USD, or using low-liquid financial instruments, may require additional analysis of the order book's depth, which is beyond the scope of this research.

2.7. Volatility

Volatility in financial markets is the rate that the price of a financial instrument increases or decreases. Volatility is measured by calculating the standard deviation of the returns for a given period (Wilmott, 2013). The classical daily volatility formula is shown below (Poon, 2005).

$$\sigma = \sqrt{\frac{1}{T-1}\sum_{i=1}^{T}(r_i - \mu)^2} \quad \textbf{(2.1)}$$

where r_i is the difference between the maximum and minimum prices during the day i;

μ is the average difference between the maximum and minimum prices over the T-days period.

Volatility can also be calculated for any time-series period such as one minute, five minutes or one hour.

Since 2008, stock and currency markets have shown increased volatility. High levels of volatility have a high risk for ATSs, which are based on trends such as negative political events and unexpected governmental decisions. In 2020, everyone faced the COVID-19 pandemic, which also affected financial markets in the form of extreme volatility. It is impossible to predict these events. Consequently, the trend-based systems may have disadvantages and countertrend trading systems give better results. A countertrend-based strategy is able to determine the pivot points of the financial market for opening positions.

The DL neural network should be trained to determine which type of trading strategy (trend-based or countertrend-based) is more suitable for intraday trading at any given time. One of the ML model's target functions may be recognition of a certain level of volatility because volatility changes could indicate a transition of the financial market from a flat to a trend-mode and vice versa, which is essential for determining the point of entry into the market.

The volatility of financial markets is one of the key concepts used in this research to create the probabilities vector, which is one of the crucial innovations of this research.

2.8. Price Gaps

Another important reason for focusing this research on day trading is price gaps. The term price gaps, also known as morning gaps or opening gaps, refers here to a significant difference, usually between the closing prices of the previous trading day and the opening prices of the new trading day (Caporale and Plastun, 2017). Figure 2.5 shows price gaps in stocks of Apple Inc in January 2020. These price gaps, which are common to all financial instruments, usually happen at night, when the stock market is closed. Reasons for these happening include negative political events and unexpected government decisions, quarterly earnings reports, as well as other news, either general economic news or news on a particular company, that becomes known when the stock exchange is closed (Nazário, et al., 2017).

Because prices between these daily trading sessions change dramatically, it is risky to hold long positions (purchased financial instruments) or short positions (sold financial instruments) overnight (between day trading sessions), especially if these positions are opened with leverage (borrowed funds). Therefore, in order to reduce the risk, all the positions (long or short) close before the end of the daily trading session. A price gap can happen either upwards or downwards, so traders should not hold a position overnight. This is a key principle of intraday trading.

Figure 2.5: Apple Inc shares price gaps. Screenshot from (Interactive Brokers LLC, 2022).

2.9. Stop-Loss

Stop-loss is an order to sell/buy a Financial Instrument when its market price reaches a certain level (Gitman, et al., 2015). The maximum loss allowed to be received in each transaction is regulated using stop-loss orders.

> *"Traders should employ stop-loss orders as a way of specifying the maximum loss they are willing to accept. By using stop-loss orders, traders can avoid the common predicament of being in a scenario where they have winning trades but a single loss large enough to eliminate any trace of profitability in the account."* (Lien, 2006).

Having a stop-loss value is an obligatory parameter of each trading strategy (Baviera and Baldi, 2019). Due to insufficient attention to risk-management systems, of which stop-loss is a part, private investors lose about 2% of their assets annually in the US financial markets. It is noteworthy that the profit of institutional investors (banks, funds, etc.), which do use these systems, is approximately 1.5% per year from the same small transactions (Barber, et al., 2009).

The most advanced risk-management strategy involves calculating stop-loss as a function at each point in time, depending on each tick (Leung and Li, 2015). This does not mean that the stop-loss must be moved after each tick, but it is desirable to change it periodically following this calculation. The target functions of a DL model will be finding patterns and points of entry into the financial market, which will allow the minimisation of stop-losses occurring during trading.

Minimising the execution of stop-loss orders is part of the aim of this research as one of the target functions for data preprocessing.

2.10.Summary

This section gives a summary of the relationship between the financial concepts introduced in the previous sections, the research experiments described in Chapters 4-7 and the research objectives identified in Section 1.3. This section also explains how three key experiments, which are proposed as the solution, relate to this research's aim and objectives and how they will compare them with state-of-the-art.

This research focuses on **Intraday Trading** (Section 2.1) because it is a potentially more profitable trading style due to greater overall **Volatility** (Section 2.7) than medium- and long-term trading styles , which are usually based on the values of **Technical Indicators** (Section 2.4). The second reason for focusing on **Intraday Trading** is to remove the significant risk of losses due to frequent overnight **Price Gaps** (Section 2.8) when there is no way to close a losing position as the financial market is closed. This research uses raw bid financial tick data on S&P-500 **Futures Contracts** (Section 2.2) and EUR/USD **Forex** (Section 2.3) currency pair.

Intraday Trading could be profitable if it uses a **Swing Trading** (Section 2.5) technique based on the intraday **Countertrend-Based Strategy** (Section 2.5) and determines entry points into the market with the minimum probability of **Stop-Loss** (Section 2.9) orders being executed.

Limitations are also established for applying the results of this research, implying absolute **Liquidity** (Section 2.6), in other words, the ability to buy or sell a financial instrument at the current price without influencing the price by doing so.

The preliminary experiments (Section 4.9.1 and Section 4.9.2), including the visualisation of the results (Section 4.7), and the first series of experiments to determine the price of the next tick (Chapter 5), have helped the author to come upon the vital concept of a **Probabilities Vector** (Section 6.5). The required processing power was also measured, and as a result, the necessity to reduce the size of incoming data (Section 6.6) in the following neural networks was determined.

As already noted, it is not necessary to predict the future prices for successful intraday trading; it is only necessary to predict the direction of the future **Swing Movement** (Section 2.5). It is also vital not to enter the market during a flat. Thus, the prediction task is separated into components (Section 6.7.3 and Figure 6.22), and for each next tick, one of three ‘simple

decisions' must be made: buy, sell, or wait. At this point, this research moves from a regression task to a classification task.

The preprocessing and data size reduction (Section 6.6) show the necessity 'to wait' (to be out of the market - in the money) more than 99% of the time. Thus, one of the key experiments (the first binary classifier – Chapter 6) on separating the potential beginning of trends from flats solves the problem of the long waiting.

The next step is determining the most probable direction of potential movement. This problem is solved by the second and third binary classifiers described in Chapter 7 of the second and third key experiments. The results of these three key experiments are evaluated based on the standard error metrics and loss functions (Section 3.7) against the outcomes of existing state-of-the-art published research (Section 8.2).

All the above experimental results should be consistently applied during the operation of the trading system in the future. At the same time, they could both be used on each next tick if the existing processor capacity allows or on any number up to 10 ticks. It was found (Fang, et al., 2021) that in order to take into account changes in the market participants' behaviours, it is sufficient to retrain neural networks weekly (5working days) on additionally obtained data.

Thus, all these experiments, quantitative and qualitative results briefly described above, are related to this research's aim to improve the state-of-the-art performance of intraday trading, as well as to the research objectives indicated in Section 1.3.

3. Literature Review

This chapter reviews the state-of-the-art scientific research and papers in the ATS, and financial market analysis domains, as well as the application of well-known neural network architectures to trading in the financial markets. It also describes the differences in the time structure of financial market data in different time frames, and how this significantly affects the analysed financial market data size and the neural network architectures to be used.

Practical and useful information about successful ATS using ML for stock markets is often highly confidential because these systems use sensitive information and are often proprietary. However, from 2019 to 2022, high-quality publications appeared which used ML-frameworks in the domains mentioned above.

In this chapter, the state-of-the-art concepts from seven significant areas of research are going to be considered:

i. ATS application (discussed further in Section 3.1)
ii. Financial market analysis (discussed in Section 3.2)
iii. ML and DL applications (discussed in Section 3.3)
iv. Combination of the ML and DL with ATS practices (discussed in Section 3.4)
v. Evolution of Technical analysis and Japanese candlestick charting techniques (discussed in Section 3.5)
vi. CNN vs MLP and RNN for financial market prediction (discussed in Section 3.6)
vii. Error metrics and loss functions for evaluation of the results (discussed in Section 3.7).

3.1. ATS Application

Chapter 1 listed potential different approaches and types of ATSs which use financial markets' imperfections utilising different time frames. Much empirical literature has been written on each of the above approaches. Mike Halls-Moore has tried to systemise present-day methods of algorithmic trading (Halls-Moore, 2019). Chebotarev (2011) describes the ATS in the Russian financial market. Figure 3.1 demonstrates the average decision terms for the different types of trading technology, leading to different approaches to creating ATSs, depending on the time frames used.

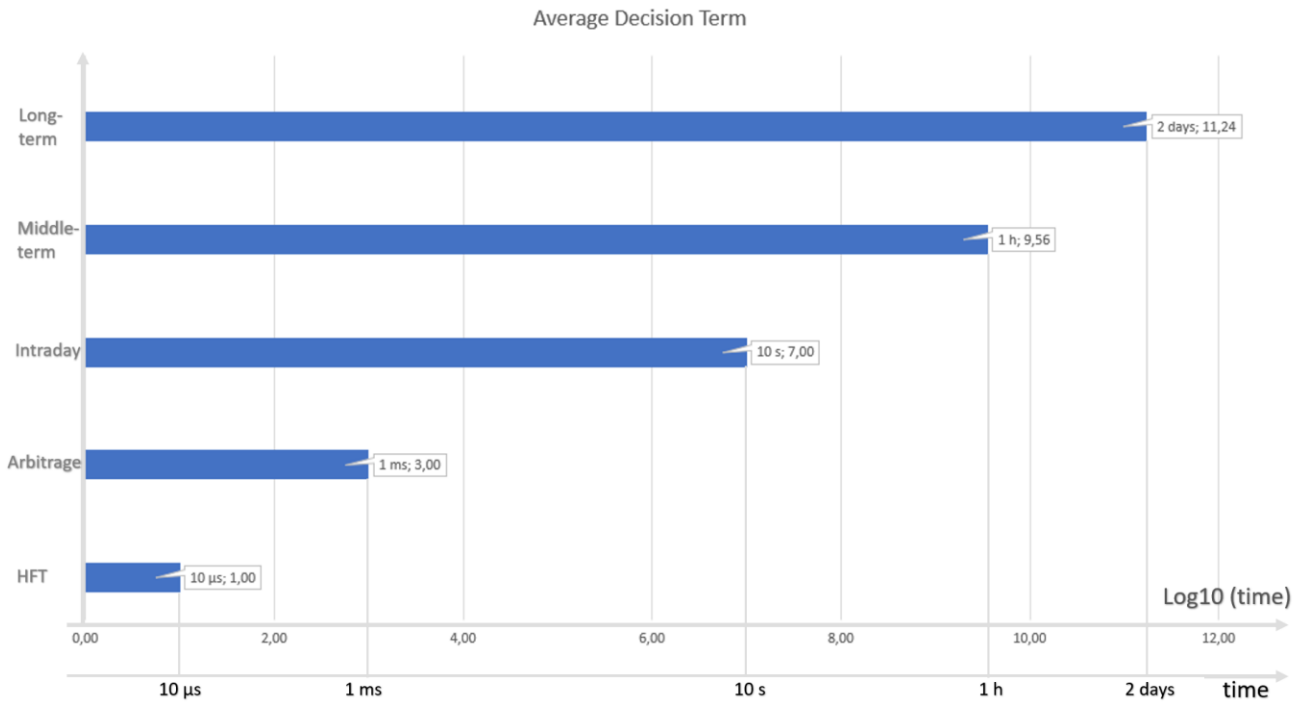


Figure 3.1: Average decision terms for HFT, Arbitrage, Intraday Trading, Middle- and Long-term investments (logarithmic horizontal axis).

The current research does not consider High-Frequency Trading (HFT), which is discussed in the following papers (Benos et al., 2017; Chong, Han and Park, 2017; Brogaard, et al., 2018; Dixon, 2018; Baron, et al., 2019; Van Kervel and Menkveld, 2019; Aldrich and Vargas, 2020; Guasoni, Mishura and Rásonyi, 2021). HFTs elicit colossal infrastructure costs and require powerful computer servers that must be located as close as possible to the stock exchange servers. Currently, the fastest HFT systems can process up to 10,000,000 orders per second (Tilen and Van Waeyenberge, 2020). Only large banks and brokerage companies can afford such investments in the expensive infrastructure. HFT algorithms are quite simple and are based on the advantages of the speed of placing and removing trade orders in an order book.

Many researchers concentrate on arbitrage (Eraslan, 2019; Jarrow and Chatterjea, 2019; Nakajima, 2019; Pichl, Nan and Kaizoji, 2020), which is considered one of the least risky strategy in financial markets (Cho, 2020). Arbitrage is a method to make a profit by trading on the differences in the price of the same financial instrument in different financial markets simultaneously (Chaboud, et ai., 2014). A comparative analysis of the effectiveness of neural networks as well as other methods of machine learning, such as gradient-boosted trees and random forests, is made by Krauss, Do and Huck (2017) in the context of arbitrage. However, arbitrage also needs expensive systems and large infrastructures, similar to HFT; hence, this

research does not consider arbitrage nor HFT, even though they are extremely short-term methods. The current research also does not consider long-term investments (Gerrard, et al., 2020) due to significant price-gaps and unpredictable currently increasing market volatility.

ATSs have become increasingly widespread since the 1970s when computing first allowed online econometric calculations. As a result, ATS concentrates on Quantitative Analysis in Finance (QAF) (Wilmott, 2013) which essentially consists of stochastic mathematics (Shreve, 2005), probabilistic methods and risk valuation (Baxter and Rennie, 1998; Wang, 2020). A little later, the concept of 'Quants' and 'Quantitative Trading' appeared in the financial world. Quants are analysts who use only quantitative analysis to make deals in the financial markets. Quantitative trading is mostly used by financial institutions and hedge funds (Hull, 2014). The most famous and one of the most successful quantitative funds is Renaissance Technologies LLC (Renaissance Technologies LLC, 2022). Proprietary ATSs have driven the recent success of London-based company XTX-Market Limited, founded only in 2015, which ranked third in 2018 as a global market liquidity provider with a 7.36% market share, overtaking HSBC and Deutsche Bank (Euromoney, 2018).

The advantages and disadvantages of a wide variety of flexible ATSs are discussed in monographs, such as (Solabuto, 2011; Wilmott, 2013). They describe the mandatory elements and sequence of actions that are necessary to develop ATSs. They also show advanced methods for back-testing and risk quantification of ATSs that are used by the current research. It is necessary to narrow the scope of this research, which exploits the analysis of big financial tick data using neural networks, and the current research concentrates on only intraday trading (day trading). Classical modelling of stock market intraday activity and behaviours has been demonstrated in books and papers (Nison, 2004; Frost, 2005; O'Neil, 2005; Lien, 2006; Morris, 2006; Murphy, 2009; Bauwens and Giot, 2013). These models could be used for testing, verification and valuation of ML models in accordance with OB-4 (Section 1.3).

3.2. Financial Market Analysis

This section describes the structure of the financial market and explains how raw tick data differs from financial time-series based on integrated data. This section also describes the role of trade volume and its role in analysing future potential movements.

3.2.1. Financial Market Structure

The dynamic structure of the financial markets is the central challenge of the modelling task. The markets are constantly changing. That is why the majority of ATS and ML models for trading are not successful. Financial markets have a chaotic structure (Dwyer and Hafer, 2013). However, the chaos is quasi-stationary. This means that it is possible to find patterns over a short-term period because big buyers and sellers shape the financial markets. Each buyer and seller have their unique behaviour. The aim of the desired DL neural network is to find repetitive behaviour before it changes. A dynamically changing DL model for the analysis of videos is described by Ashangani, et al. (2016). This could potentially be repurposed for analysis of continually changing financial markets.

Figure 3.2: An example of a time-series chart with integrated data (15 min) of EUR/USD Forex. Screenshot from (Dukascopy Swiss Banking Group, 2022b).

Before Google introduced the TensorFlow ML-framework in November 2015, ML models on financial markets were based on classical ML algorithms and simple self-made neural networks and tried to solve the problem by different approaches (Larsen, 2010; Flach, 2012; Patel, et al., 2015). Since then, several papers have been published trying to solve the task

of prediction of stock markets by using TensorFlow. Nevertheless, some of the researchers continue to use other ML libraries (Qiu and Song, 2016; Dixon, Klabjan and Bang, 2017), in particular, PyTorch (Hu and Lin, 2019). Furthermore, the majority of the researchers use integral (interval) financial time series with different durations (Chou, Truong and Le, 2020), as well as Bayesian statistics (Halls-Moore, 2018).

Figure 3.2 shows a chart of Forex EUR/US Dollar on 15-minute intervals that is an example of integral (interval) financial time series. Each 15-minute bar consists of open, close, maximum and minimum prices, as well as the volume for a specified period.

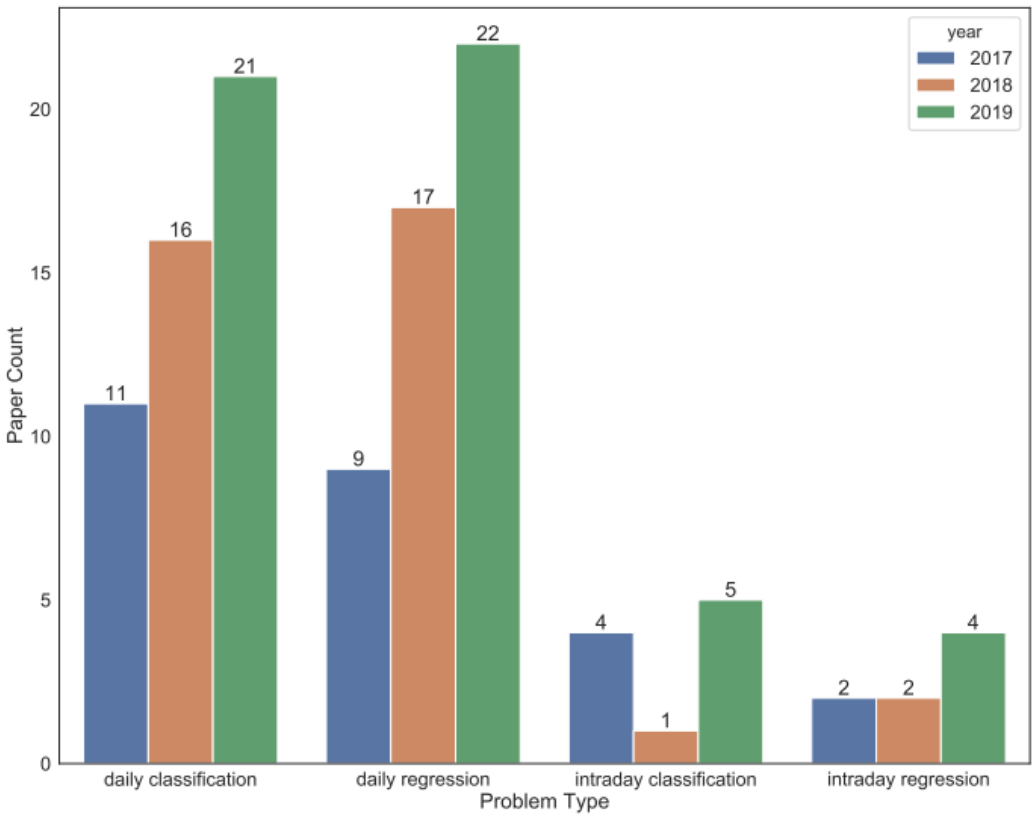


Figure 3.3: Number of research papers with published open-source code (Jiang, 2020).

However, in real financial markets, settlements are at irregular intervals, resulting in an uneven intensity. The problem with the majority of the above-mentioned research is that financial markets do not have time series of equivalent intervals. In raw data, such as tick data, deals are taken into account at irregular intervals as they occur. However, the majority of research (Udagawa, 2018; Wu, et al., 2020b; Heinz, et al., 2021) use time series at equivalent intervals (interval time series) that sum all ticks over a particular interval, such as one minute, five minutes, or one day, but only a small minority of these researchers have published open-source code. Inside this integral data, useful information is lost that can show

the patterns of behaviour of market participants. Stasiak (2020) describes integral financial data as a significant mistake of economic research in intraday trading and HFT. For example, Figure 3.3 illustrates the quantitative relationship between research that uses integrated long-term data from a period of one day or longer vs short-term intraday data for analysis with neural network deep learning. However, all this intraday research uses integral dates with a period of 5 minutes or longer. The lack of research in DL using tick data has arisen for two reasons. Firstly, it is difficult for academic communities to access full tick data. Secondly, training neural networks with tick data requires significant GPU computing power.

This research concentrates on analysing raw data from multidimensional time series using neural networks. Figure 3.4 shows a real chart of Forex EUR/US Dollar ticks and volume, which are raw data. EUR/US Dollar futures on US financial markets have similar charts. The periods between deals or between tick changes are not equivalent (Sandubete and Escot, 2020). Visualisation of tick data of prices and volumes can explain the financial market behaviour; however, it is impossible for humans to observe and analyse this big data without automation. Tick size is the smallest upward or downward price movement. Each point on the chart shows the volume of market orders' transactions that were at the corresponding price. The upper line corresponds to market buy orders that are executed with ask-limit sale orders (Figure 3.5). Conversely, the bottom line corresponds to market sell orders that are executed with bid-limit buy orders. Each tick can consist of many deals with different market participants, which take place at the same price at the same time. If any of the prices (bid or ask) changes, or the time in milliseconds changes, then the deals belong to the next tick. The unequal intervals between transactions and significant differences in the volumes of each transaction (tick) are obvious.

As was shown by Stasiak (2020) and confirmed in further experiments, the integral data summed up in each regular time frame (for example, 1, 5, 30 minutes, one hour or one day) unfortunately loses vital information about the market participants' behaviours. The results of further analysis of big financial tick data will show the benefits of this approach.

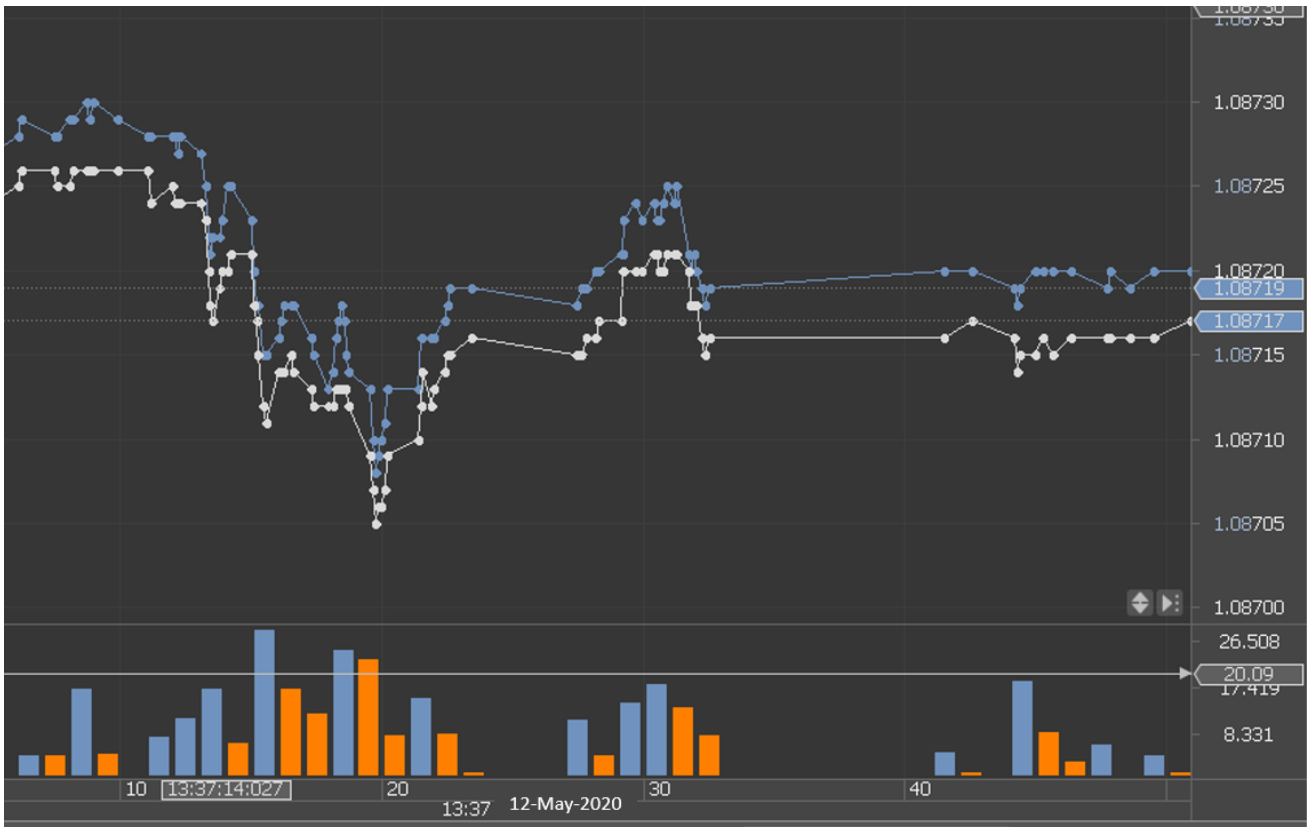


Figure 3.4: An example of a real chart of the Forex EUR/US Dollar which is based on raw data of ticks and volume. Screenshot from (Dukascopy Swiss Banking Group, 2022b).

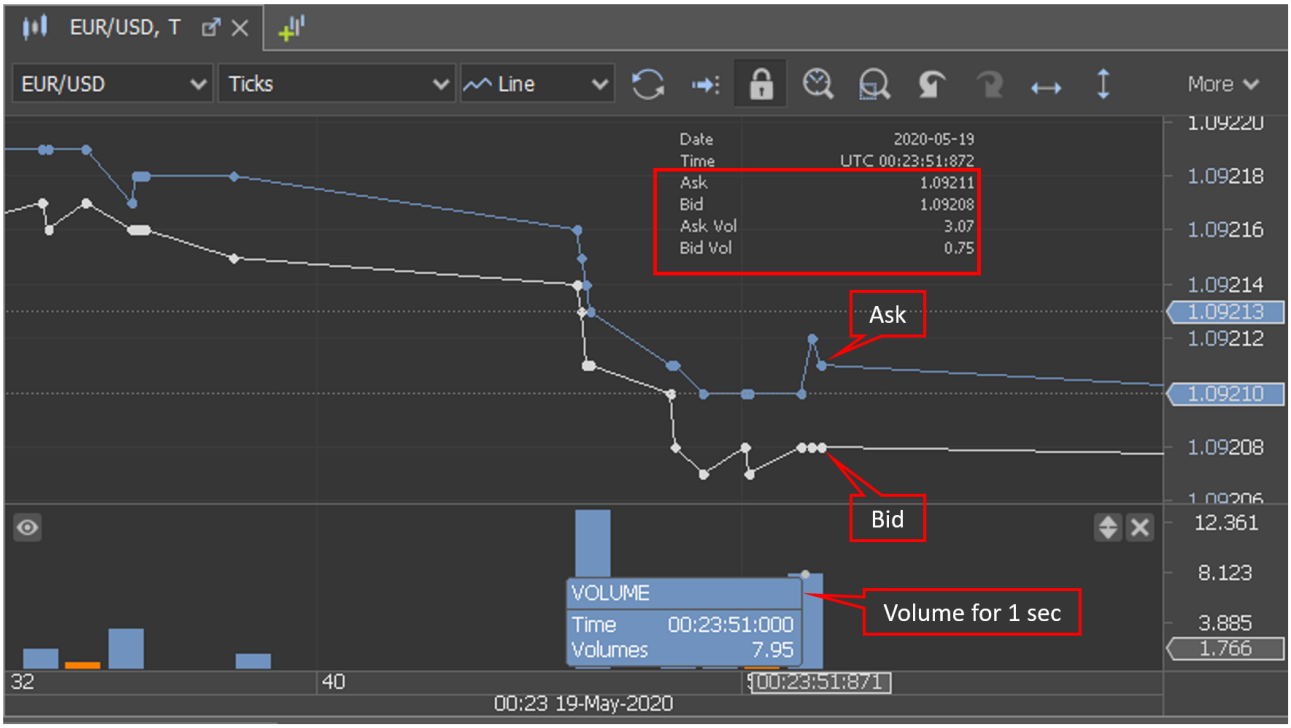


Figure 3.5: Tick chart (price & volume) of the Forex EUR/US Dollar. Tick size is the smallest upward or downward price movement. Screenshot from ((Dukascopy Swiss Banking Group, 2022b).

The majority of statistical methods of time-series analysis based on frequency or time-domain, such as spectral analysis and wavelet analysis, do not have predictive power for the time series of financial markets since these have an irregular fractal structure (Mandelbrot, 1983; Mandelbrot and Hudson, 2010; Karp and Van Vuuren, 2019; Ku, et al., 2020).

Thus, the time-series statistical analysis methods lose essential information if they use integral (interval) time-series intervals. Consequently, the quality of DL model prediction and the capability to find patterns could decrease if the DL model uses integral data. Using simple examples, Stasiak (2020) shows the problems that the majority of researchers have when using integral data in the analysis of financial markets using neural networks. This paper compares using big financial tick data of EUR/USD exchange rates and integral Japanese candlestick data. Particular attention is paid to the problem of isolating existing research from the real practice of trading and investing. For example, as will be shown in this research, it is not necessary for successful intraday trading to predict the prices of a financial instrument at a certain point in the future. This paper also focuses on the problem of information loss for historical data in integral form. Due to isolation from the needs and aims of practical trading, many theoretical solutions, despite their formal correctness and expediency, in practice lead to financially inefficient investments and transactions. Data preprocessing and the original binary-temporal representation method are also proposed in this paper to improve the results.

3.2.2. Time-Series Analysis of Financial Markets

Statistical analysis of time series of various financial markets was prevalent in the 2000s and early 2010s among researchers before the widespread use of neural networks (Tsay, 2005; Mills and Markellos, 2008; Kim, et al., 2011). Some financial market data shows impulse, the time-series momentum and asset pricing anomalies, which were investigated by Moskowitz, Ooi and Pedersen (2012) for a large number of stocks, forward and futures contracts, commodities, equity indexes and currencies. A detailed analysis of the behaviour of speculators and hedgers was conducted. Using 58 liquid financial instruments, Moskowitz, Ooi and Pedersen (2012) were able to reduce the trading risks.

Another relationship which has been investigated using methods of statistical analysis of time series is that between investor sentiment and impulse movement (Han and Li, 2017).

The problem with the analysis of financial markets using statistical methods is that the time series of prices, volumes and orders are non-stationary with a lot of noise, jumps and

stochastic movements. Thus, using various variables and mathematical methods to extrapolate time series, it is possible to predict their behaviour only two or three fixed time frames in advance (Ican and Celik, 2017). This is a very short horizon of prediction. For example, Galeshchuk and Mukherjee (2017) analysed two statistical methods for the analysis of time series: ARIMA and Exponential Smoothing (ETS) in comparison with one of the machine learning methods, such as SVMs, and deep neural networks, such as ANNs and CNNs. The analysis was carried out on three Forex currency pairs: EUR/USD, GBP/USD and JPY/USD. An important innovation of this paper is the purpose of determining the direction of the future movement of these currency pairs in addition to predicting their future prices. As a result, it was found that the classical basic models of time series analysis have low loss functions when predicting future prices in a short period; however, deep neural networks better predict the direction of future movement with an accuracy of 78.53%, which is 18 to 31% higher than other models.

Using algorithms based on ML is superior to the classical approaches of statistical analysis of financial time series in terms of accuracy, which was convincingly demonstrated in (Rundo, et al., 2019; Sezer, Gudelek and Ozbayoglu, 2020). This research does not use time-series analysis.

3.2.3. Trading Volume Analysis

Many researchers use Trading Volume Analysis (TVA), which is extremely important for building successful trading systems (Dinh and Kwon, 2018; Wang, Ho and Li, 2020). One TVA approach uses the Volume Spread Analysis (VSA) theory, where the spread is the length of each bar on the price chart (Linh, 2019). Most of the significant movements in market prices are associated with large volumes of transactions made immediately before these movements. A large volume of transactions signals that large buyers or sellers have brought new funds to the market. Also, a large volume may signal the closing of a position by a significant market participant, including market makers. In any case, a large volume of deals signals a change in the composition of significant market participants, and therefore, a change in their total behaviour.

Blume, Easley and O'hara (1994) were one of the first to demonstrate a positive correlation between an increase in the volume of transactions and absolute price changes (Figure 3.6) which can be written as:

$$\left|\frac{price + \Delta price}{price}\right| = f(volume)\,. \qquad \textbf{(3.1)}$$

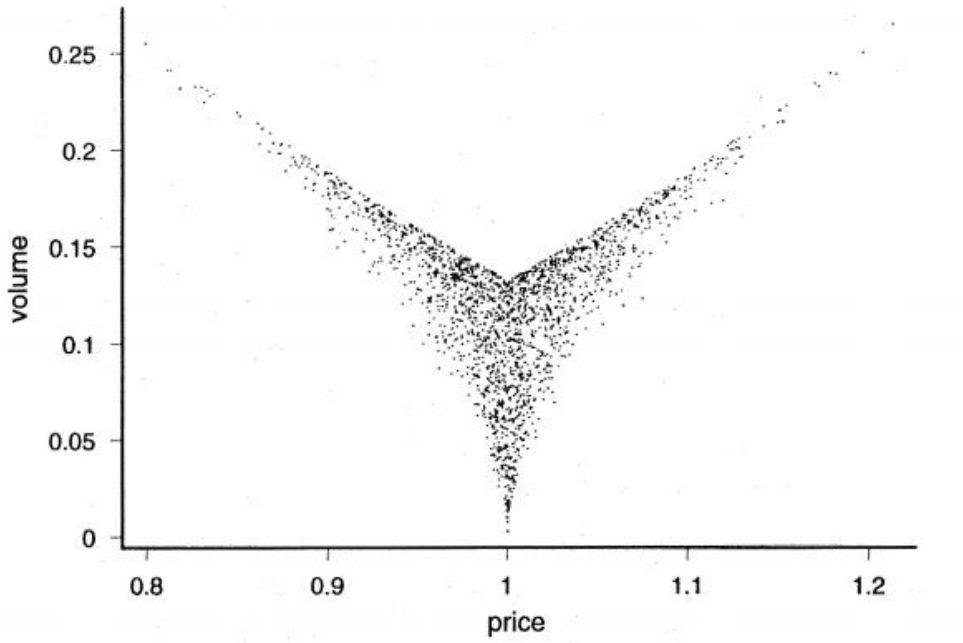


Figure 3.6: The relation of volume and absolute value of the relative value of the price change (Blume, Easley and O'hara, 1994).

This research will actively use large-volume analysis as a part of the preparation of data for training neural networks.

3.3. Machine Learning and Deep Learning Applications

Much research into the application of ANNs to analyse financial markets has been done by mathematicians and data scientists without the collaboration of professional traders. Such research often aims to test a new neural network architecture previously used for other purposes, for example, for pattern, sequence, computer vision or time-series recognition. This section briefly describes this research and its historical development.

Basic principles of ML have been described since the 1950s. However, the computing power was insufficient for solving real-world tasks. Basic neural network architectures, a mathematical analysis of the networks, learning rules, non-linear regression, pattern recognition, and other practical aspects are introduced in the monograph (Hagan, et al.,

2014). The issues of validation and assessing the quality of financial analysis using neural networks are described in detail by De Prado (2018). A detailed description of the principles of the Keras ML-framework is described by its creator in (Chollet, 2022). Evolutionary and genetic algorithms, differential evolution, brainstorm optimisation algorithms, hybrid methods, fuzzy methods, and data mining approaches are described in (Tan, et al., 2015; Wirsansky, 2020). Examples of the practical application of ML in financial markets, as well as methods for limiting risks and creating ATSs based on neural networks, are considered in (Yan, 2017; Hilpisch, 2018). Typical mathematical and statistical applications for using machine learning in finance are described in (Weiming, 2019). More advanced end-to-end ML for the trading process, from idea development and model optimisation to strategy development and backtesting, are described in (Jansen, 2020). Furthermore, a collection of various structures of deep neural networks is described in detail in (Goodfellow et al., 2016). ML and DL method abbreviations are detailed in (Jiang, 2020) which are shown in Appendix V.

All of the above-listed books and papers describe the basic structures and principles of ML and training neural networks in finance for using time frames with integrated data. These sources differ mainly in the architecture of neural networks and the ML methods used. However, hands-on training based on tick data requires serious refinement of neural network architectures and methods and applications used to operate with big tick data.

3.4. Combination of ML and DL with ATS Practices

3.4.1. Multilayer Perceptron

Researchers actively use Multilayer Perceptrons (MLPs) to analyse financial markets. MLPs are one of the most common ANN architectures and contain more than one sequential layer. Figure 3.7 demonstrates a generic ANN with two hidden layers, which is described in detail in (LeDel, 2016). The MLP is fully bound (dense). Each element of the ANN is connected with every element from the previous and following layers. Often this type of layer is called ‘densely connected’.

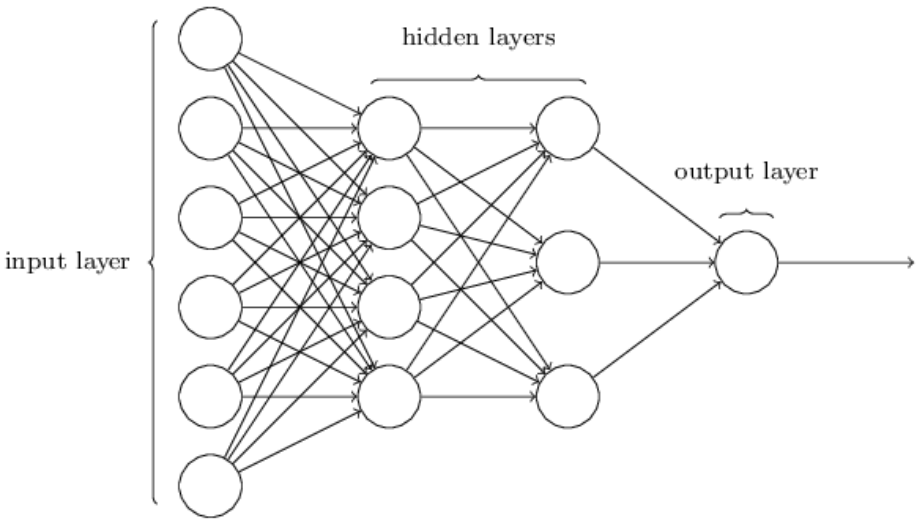


Figure 3.7: Multilayer Perceptron Architecture Example (LeDell, 2016).

An essential property of a multilayer perceptron is the influence of each cell of the previous layer on each cell of the subsequent layer. The number of elements in each layer depends on the size of the input data.

Multilayer perceptrons have a relatively simple architecture that does not require many computations; therefore, they are commonly used by researchers. For example, Sezer, Ozbayoglu and Dogdu (2017) describe MLP training using financial indicators as inputs. Nevertheless, the result of this experiment is only comparable to the Buy and Hold strategy. However, when adding data on the volume of transactions, the quality of forecasting using MLP significantly increases compared to the Buy and Hold strategy (Dinh and Kwon, 2018). This once again suggests the need to analyse not only prices but also trading volumes in financial markets.

Diversified variations of the analysis of different time frame data of financial markets using MLP neural network architectures were described in (Li and Tam, 2018). HFT systems are also created based on MLP architectures (Hernández, Hoyos-Reyes and Martínez-Preece, 2018). However, the weakness of MLP for price forecasting in financial markets is in the network architecture itself. The influence of each cell of the previous layer on each cell of the next layer sometimes leads to overfitting of MLP on noisy financial data (Namdari and Li, 2018).

In this research, MLP has been used for prediction of the price of the next tick in the regression task.

3.4.2. Recurrent Neural Networks (RNNs)

The next problem of the ML models in financial markets, which should be indicated, is the duration of memory. Humans have a memory, and they start to think based on previous conclusions or events. A significant weakness of a simple traditional ANN is that it has no memory. The problem may be solved by using a Recurrent Neural Network (RNN) architecture, which is a type of ANN. It is particularly well suited to forecasting and sequence classification, such as text, music, and time-series data (Olah, 2015). RNNs are actively used by other researchers to analyse time-series data from the financial market, mainly for medium and long terms (Karpathy, 2015; Jarrah and Salim, 2019). The connections between cells inside RNN can be cycles. Figure 3.8 demonstrates an unrolled RNN.

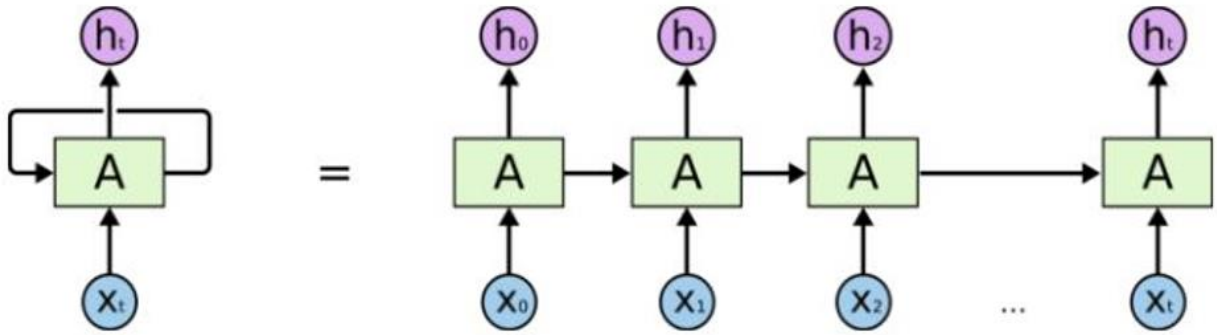


Figure 3.8: An unrolled Recurrent Neural Network (Olah, 2015).

Unfortunately, RNN can save only short-term connected information. The problem was explored in detail first by Hochreiter (1991) and Bengio, Simard and Frasconi (1994), and was continued by (Hagan, et al., 2014), who found some fundamental reasons why it might be difficult. An application of the RNN is considered in (Dixon, 2018) for HFT. Here it is attempting to describe the microstructure of sub-millisecond movements called 'price-flips', observed after the appearance of a large market order, which takes part of the liquidity from the financial market. These movements are very short-term and occur almost instantly. Therefore, they can only be predicted and exploited with the use by powerful computers. For such short-term movements, RNN architectures are applicable.

Companies and researchers often use the RNN to analyse the news to predict financial markets. For example, Thomson Reuters launched the MarketPsych Indices system, which makes market forecasts based on more than 2,000 news sources and 800 social networks. This tool uses many ML algorithms, including natural language processing technology, to

analyze quantitative indicators, as well as emotional metrics of traders, i.e. behavioural trading (Fafula and Peterson, 2015).

A new exotic ANN architecture for trading, named a ‘spatial neural network’, was described in (Sirignano, 2019). The central idea of the paper is that market price depends only on a situation in the limit order book. The statement is quite controversial, but the neural network architecture should be analysed and tested. However, while this has potential for financial analysis, this research does not use it, because the analysis of order books with tick data will require using a supercomputer. For example, the model mentioned above was trained with a supercomputer cluster of 50 GPUs because it has significant computation. This would be an interesting subject for further research.

A Long Short-Term Memory (LSTM) network is a type of RNN that is capable of learning long-term dependencies (Figure 3.9). This architecture was introduced by Hochreiter and Schmidhuber (1997), and can be employed for sequence learning (Nelson, Pereira and de Oliveira, 2017). Fischer and Krauss (2018) assert that the LSTM can formalise a rules-based monthly-term reversal strategy, which has the capability to achieve the highest annualised returns of 82.29%. However, it does not consider short-term strategies for intraday trading. Other papers, such as (Troiano, Villa and Loia, 2018) and (Sang and Di Pierro, 2019), also use LSTM to forecast financial markets using time series. However, they all use time frames with integrated data. Fischer and Krauss (2018) compare LSTM networks with other classical methods of ML, such as random forest, and a standard DL neural network. Maneejuk and Srichaikul (2021) use RNN and LSTM for analysing the historical data of EUR/USD, GBP/USD and JPY/USD Forex currency pairs together with data sets for macroeconomic analysis and technical indicators, emulating the most popular methods of Fundamental and Technical analysis. Experiments are carried out using various methods, such as classical ARIMA (Autoregressive Integrated Moving Average), SVM (Support Vector Machines), and various architectures of neural networks: ANNs, RNNs, including LSTMs. As additional databases for Technical analysis, the researchers use several technical indicators, such as Relative Strength Index (RSI), Bollinger Bands (BB), Momentum, Moving Average Convergence/Divergence (MACD), Moving Average (MA), Rate of Change (ROCh) and the Commodity Channel Index (CCI). Experimental results for the EUR/USD currency pair are MAE from 0.0295 to 0.1836 for different methods.

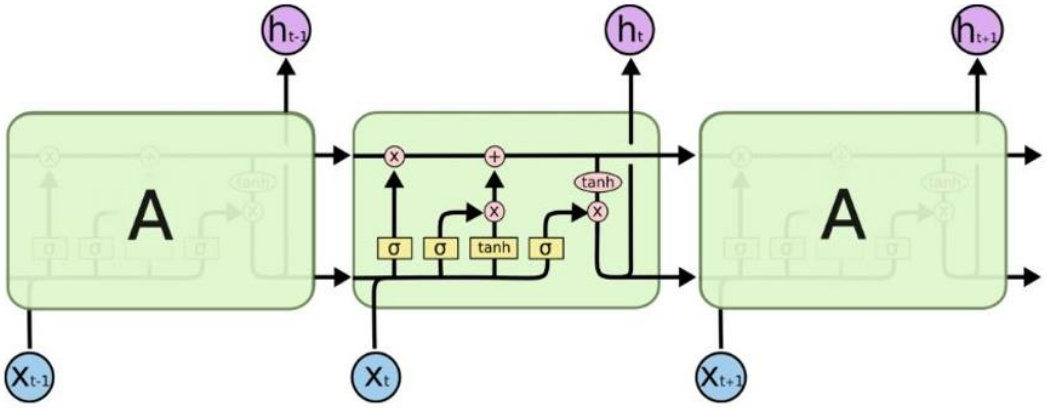


Figure 3.9: The repeating module in an LSTM contains four interacting layers (Olah, 2015).

Aggarwal and Sahani (2020) compare using various recurrent neural network architectures, such as simple recurrent neural networks (SRNNs), long short-term memory (LSTM) and Gated Recurrent Unit (GRU), to predict 22 countries' currency rates against the United States dollar (USD). The results show that when predicting future prices, the LSTM models perform better with MAE equal to 0.0475, which is almost twice as good as the GRU result.

Currently, LSTM neural network architecture would seem to be the best choice for analysing financial markets over medium-term time-terms. Using this integrated data allows researchers to use a relatively low-power Graphics Processing Unit (GPU) and Tensor Processing Unit (TPU) rather than expensive high-performance computer servers. However, the current research aims to conduct more advanced analysis of financial markets based on tick data using available GPUs and TPUs.

In this research, the RNN architectures have been used to validate the results obtained with convolutional neural networks.

3.4.3. Convolutional Neural Networks (CNNs)

A Convolutional Neural Network (CNN) (Figure 3.10) is a type of ANN that has achieved impressive results for the analysis and recognition of visual images. The CNN aggregates elementary features into more abstract details on each subsequent hidden layer, right up to the configuration of simplified images or patterns that can correspond to the answers. In this case, all non-essential details are filtered out. The CNN has convolution layers followed by pooling layers. Before the widespread use of Deep CNN from 2014, researchers manually constructed these abstract attributes. They are currently more focused

on choosing architectures of CNNs for particular goals in Applied AI, such as computer vision, natural language processing, medical image analysis, pattern search in various types of data, speech recognition, and more (Mane and Kulkarni, 2020).

However, the same principle may be employed for analysing financial market data. CNN architecture has the ability to cope with noisy data successfully. This property is required to successfully analyse and find patterns in the extremely noisy big data from financial markets (Doering, Fairbank and Markose, 2017). Several studies identified patterns in financial market transactions using technical indicators. (Dymova, Sevastjanov and Kaczmarek, 2016; Sezer, Ozbayoglu and Dogdu, 2017; Chen and Liao, 2018; Li and Dai, 2020).

The idea of converting time series of technical indicators into images was proposed in (Sezer and Ozbayoglu, 2018). A CNN was trained on fifteen images of each of fifteen various technical indicators formed over a fifteen-day period, in order to analyse the long-term behaviour of the stock market and Exchange Traded Funds (ETF) market. Despite this research being successful, it should be noted that important market information is lost in the case of technical indicators used because these are integral parameters (Göçken, et al., 2016). In other words, they average all independent raw data. Nevertheless, this method of converting time series into images has exciting potential, and will be used in this current research for analysing linear independent market raw data for short-term prediction of movements during the trading day. However, in this current research, images will be replaced with 3D tensors.

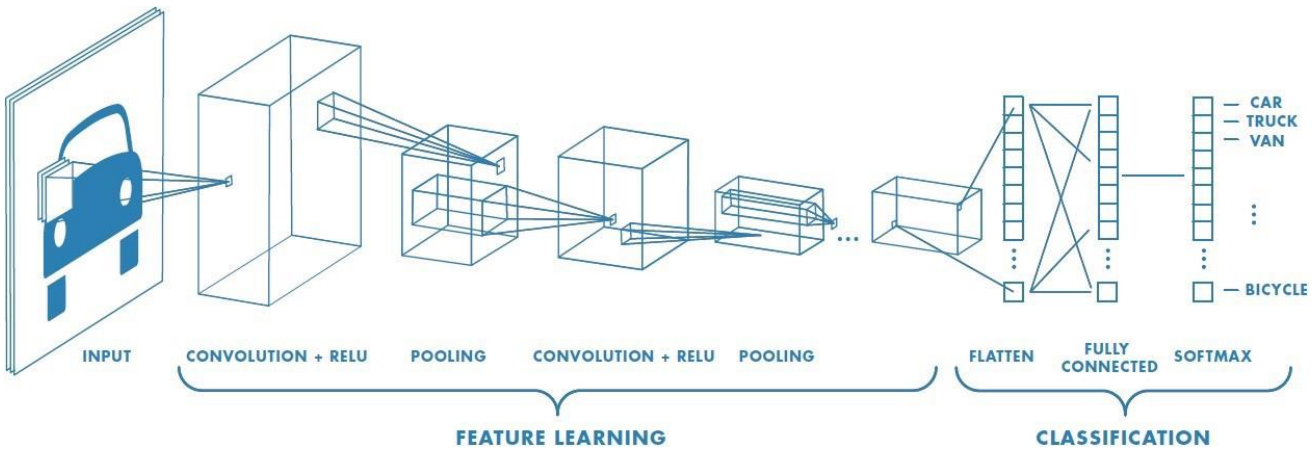


Figure 3.10: Convolutional Network (Saha, 2018).

Also, there are papers whose authors combine various properties of the above-mentioned basic types of ANNs. For example, Lee and Soo (2017) combine CNN and RNN to analyse financial news at the same time as technical analysis indicators. However, the results of the

predictions of such a symbiosis do not exceed 53% so far, and for some financial instruments, they are at 28%, which is not enough to be considered successful. An analysis of news from the Internet employing an attention mechanism for analysing short-term stock movements is used in (Wang, Huang and Li, 2019). Evaluation of the use of technical indicators as inputs of CNN is discussed in detail in (Göçken, et al., 2016). Some researchers combine market data analysis not only with technical indicators but also with news, fundamental analysis (Liu, Lu and Du, 2019) and Wikipedia traffic (Weng, Ahmed and Megahed, 2017).

An unusual approach for analysing HFT data using a combination of CNN and LSTM was demonstrated in (Zhang, Zohren and Roberts, 2019). The resulting model was able to give weakly positive results even in the markets in which it was not trained. The 1D CNNs actually perform as a form of aggregation operation. The aggregation operation is determined automatically by the trained filters of the CNNs.

Also, CNNs have fewer parameters (weights), which are needed for calculations during the training process of ANNs. It is an advantage in the speed of training CNNs over RNNs. At the same time, CNNs may solve typical problems of analysis of time series that historically were learned by recurrent-type ANNs. Moreover, CNNs usually solve problems with higher quality and more time-efficiency (Borovykh, Bohte and Oosterlee, 2017). The paper demonstrates the result exclusively on long-term financial data.

Neural networks outperform standard statistical analysis methods, such as logistic regression and naive empirical models. As demonstrated in the paper (Sirignano, 2019), the accuracy of prediction of financial market behaviour is better with neural networks.

Deep neural networks are especially suitable for processing dense data with a high level of redundancy, such as recognition or generation of images, sounds, texts, as well as pattern recognition in financial data.

This research accepts that for some types of raw data and financial instruments, various CNN architectures are more suitable than other ANNs for recognising the behaviour of intraday traders by analysing short patterns of big financial tick data (Orimoloye, et al., 2020). These data patterns are recognised according to the same principles as ordinary pictures and images, for which convolutional neural networks are by far the most suitable architecture.

Combined neural network models from CNN and RNN can also be useful. For example, Moghaddam and Momtazi (2021) analysed and applied a complex model consisting of CNN and LSTM elements to analyse Japanese candlestick patterns for the EUR/USD Forex

currency pair. They examined binary classifiers (profitable/non-profitable) and triple classifiers (buy/sell/non-profitable). This paper explores a similar task as this research in terms of focusing on analysing the directions of future significant movements (trends). The (profitable/non-profitable) classifier is essentially a classifier (Trend-Flat). The intersection of moving averages is used to confirm the profit, which can be controversial. This model showed an accuracy of 76.09% for the binary classifier and 63.95% for the triple classifier.

3.4.4. Other Techniques used Big Financial Raw Data

Another interesting approach is the use of the reinforcement learning method for tick data of the EUR/USD Forex currency pair, as demonstrated in (Carapuco, Neves and Horta, 2018). The research used simple neural networks inside the Q-learning algorithm. It was noted that in addition to the standard reward and punishment for profit and loss, it is necessary to impose a penalty for capital drawdown. Otherwise, the system holds a losing position for a very long time, waiting for it to break even. As a result, on test data, the system showed a yield of 12.8% per annum, which was significantly higher than inflation. However, the experiments assumed a constant margin between the bid and ask prices, which is often not true in real big financial tick data. About 550 trades per year were made, which roughly corresponds to an average of 2 trades per trading day. This intensity correlates with the expected intraday trading intensity based on swing trading. Thus, the self-learning system with Reinforcement learning, which initially did not include requirements for the number of transactions and the duration of holding a position, indirectly confirmed the highest efficiency of the Swing trading technique. It can also be concluded that in order to reduce the risk and ultimately increase the profitability of the system, it is necessary to set punishment in the form of stop-loss orders.

The following two sections of the chapter describe the evolution of Technical Analysis into research using recurrent neural networks and the evolution of Japanese Candlestick Charting Techniques into research using convolutional neural networks. It also describes how differences in the structures of different time frame financial market data significantly affect the research success using the above neural network architectures.

3.5. Evolution of Technical Analysis and Japanese Candlestick Charting Techniques.

Ever since the most famous early successful stock trader, Jesse L. Livermore, all portfolio managers have attempted to create various visual or tabular charts to visually and numerically analyse historical price and volume data for financial instruments (Livermore, 1940). All ideas about Technical analysis at that time were collected, analysed and summarised in (Edwards and Magee, 1948). The Technical analysis of price charts (Schwager, 1995) has gained the most significant popularity since the widespread use of computers in the 1970s and later. An alternative to Technical analysis is Fundamental analysis of specific companies, which allows investors to understand the current fair valuation of the business and its prospects based on financial statements. Fundamental analysis is usually used for long-term strategic investments.

Figure 3.11: Support and resistance lines and price-break-out points based on Technical analysis (Jitpakdee and Pravithana, 2017).

The essential part of the Technical analysis was finding medium or long-term trend lines. Figure 3.11 shows charts with sloping trendlines indicating resistance lines, support lines, and break-out points or trend reversal. Also, horizontal lines of support and resistance find practical application. Obviously, for such a Technical analysis, it is necessary to take into

account the historical prices of a particular financial asset. It could be assumed that the success of using recurrent neural networks, including LSTM, in price forecasting using integrated data with medium-term time frames, is based on the ability of these neural network architectures to remember previous levels of trend reversals and recognise patterns within themselves similar to the sloping and horizontal support and resistance lines shown above. A more important inference from the above assumption is the analogy between the Japanese candlestick charting techniques and intraday data analysis using convolutional neural networks, which is described below.

Similar Technical analysis graphs were popular until the early 2010s and were built on different time frames. They allowed, with a certain degree of probability, to determine the critical price levels (areas) where the trend should reverse, but they provided virtually no information on what particular point is the best to enter the market at with minimal risk.

In the 1990s, a technique based on analysing Japanese candlesticks gained popularity in the United States. Japanese merchants used this method hundreds of years ago to identify trends in the rice trade. Asset management companies and investors in the United States began to analyse Japanese candlesticks stock charts to find new patterns of price movement that are possible in the near future and to predict short-term market reversals (Nison, 1994).

According to (Bulkowski, 2012) there are 103 basic candlestick patterns. Some examples of simple patterns are shown in Figure 3.12. These basic patterns can consist of one to five or more candlesticks. Each basic Japanese candlestick pattern can be formally described in words Figure 3.13 or numbers Figure 3.14.

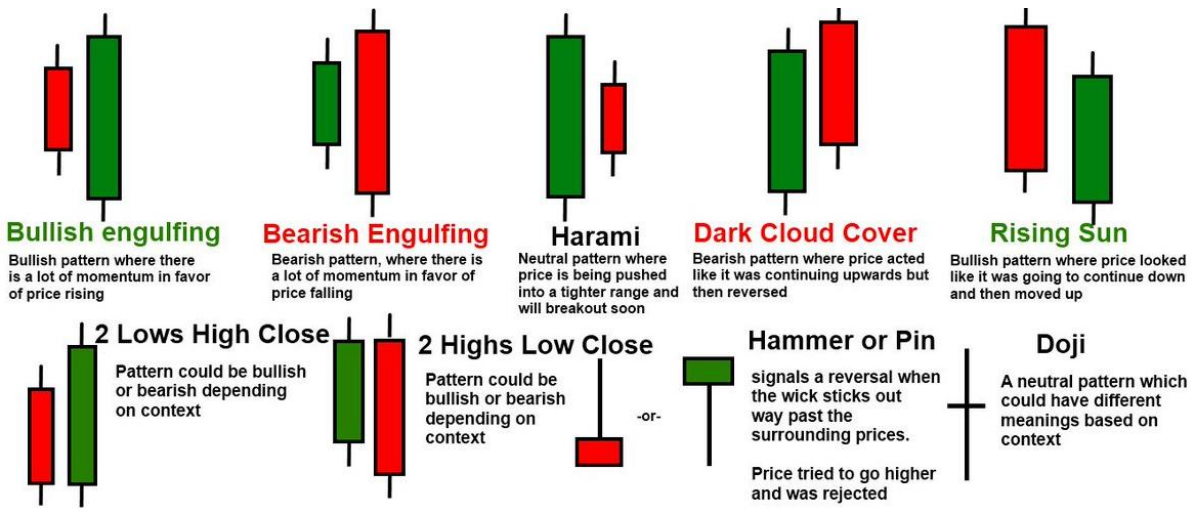


Figure 3.12: Some examples of simple Japanese candlestick patterns (Nison, 1994).

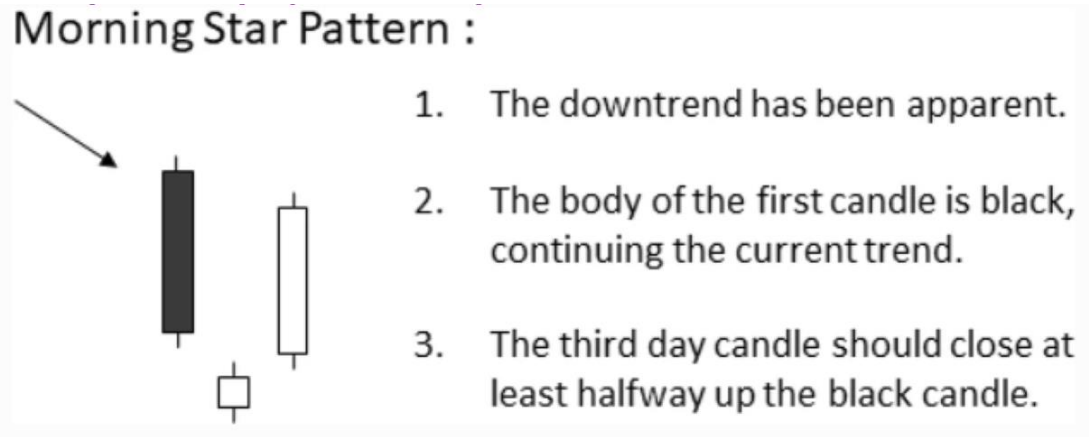


Figure 3.13: An example of a formal description in words of one of the Japanese candlestick patterns (Chen and Tsai, 2020).

session	open	high	low	close
1	23	26	21	25
2	26	28	23	27
3	23	27	22	22
4	25	28	22	25
5	27	27	25	25

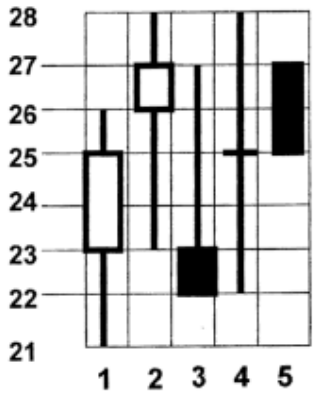


Figure 3.14: An example of a formal description in numbers of one of the Japanese candlestick patterns (Nison, 2011).

From the figures and descriptions presented above, it becomes evident that analysing and recognising Japanese candlestick patterns can be successfully carried out using convolutional neural networks since these patterns are images. Also, such process could be carried out using 1D CNNs, using a numerical description of these patterns as inputs. There are existing papers (Naranjo and Santos, 2016; Chen and Tsai, 2020) that recognise, classify and encode Japanese candlestick patterns using CNNs at the state-of-the-art level, which take into account the fuzzy nature of the modern financial market.

Since the 1970s, the practice of sequential analysis of market prices has formed, which was actively used until the early 2010s for practical trading. This practice can be demonstrated (with some simplification) in three steps as the analysed time frame decreases:

- using fundamental analysis, determining a set of promising companies for purchase, or a list of companies that have lost their development dynamics to sell;

- using Technical analysis, analysing charts of prices and volumes, a preliminary time and price range was determined for a medium-term or long-term trend reversal for each stock from the lists defined above;

- using Japanese candlestick charting techniques, a specific moment was determined to enter the market with a minimal probability of a stop-loss order being executed (minimal risk).

To sum up this section, it could be stated that for intraday trading, it is necessary to analyse only the third step of the above plan in detail. In other words, it could be enough to analyse only the patterns of Japanese candlesticks, which can be done mainly using CNNs. Also, this theory explains more about papers using RNNs than CNNs as most papers use integral data on medium and long-term time frames.

This assumption to prioritise the use of CNNs to recognise patterns in the behaviour of market participants for intraday trading is only a theory, which the key experiments of this research will partially confirm to determine the direction of future significant movements in financial instruments within the trading day.

3.6. Comparison of Neural Network Architectures for Financial Market Prediction

This section discusses another more detailed reason for prioritising using CNNs for intraday trading that follows from financial market structure.

The CNNs are more suitable because they are invariant compared with MLPs. The main difference between the densely connected layer in the MLP and the convolutional layer lies in the breadth of analysis. The densely connected layer in MLP analyses the influence of all input parameters on the global pattern (i.e., the whole image or the whole dataset, which is analysed at this step of the calculation). In contrast, the convolution layer in a CNN analyses only a local pattern (part of the image or dataset). Thus, the data near the point being analysed have the most significant impact on CNN analysis (Chollet, 2022). As a result of this local focus, convolutional networks have an essential property of transfer invariance (Mei, et al., 2017). In other words, after training, a CNN can identify this pattern in any location of a full

dataset, and not only in the location where this pattern was initially located in the training dataset. This property accelerates both learning and recognition.

CNNs also possess a property called ‘spatial hierarchies of patterns’. The first layer recognises the most straightforward patterns, on which more complex patterns are built in the next layers.

While Recurrent Neural Networks have memory emulation, which makes them suitable for processing sequences (Wang, et al., 2016), they require too much computer time for training. RNNs can memorise in what sequence the same patterns appear because they have internal loops that iterate over and remember these sequences. However, due to this cycle, the number of calculations increases dramatically.

As will be shown below, short-term patterns are essential for intraday trading. There is no need to remember the sequence of occurrence of these patterns since this sequence can be changeable.

Figure 3.15: Sine wave middle-term (4 hours) EUR/USD Forex chart in May 2020. Screenshot from (Dukascopy Swiss Banking Group, 2022b).

The majority of middle-term and long-term charts of financial instruments during flat movements (Figure 3.15) look like sine waves (Figure 3.16).

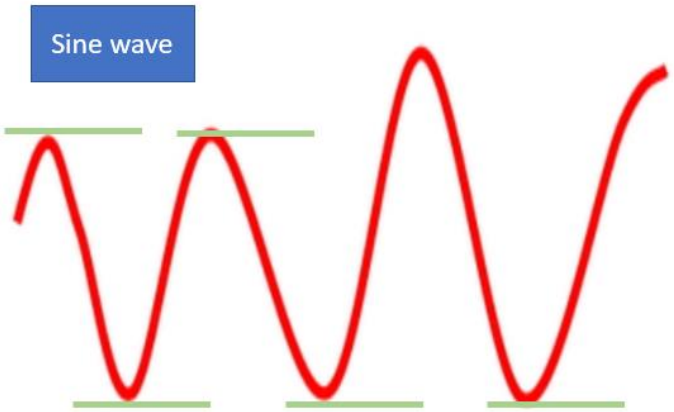


Figure 3.16: Sine wave model of middle-term price movement with support and resistance levels (green lines) of prices.

In addition, the price levels (lines) of support and resistance are essential to determine the potential turning points for medium-term trading in financial markets (Hsu, Taylor and Wang, 2016). These are the levels of prices where the market made U-turns or long slight oscillating movements (flat) in the near past. These levels can be explained in terms of the behaviour of participants in financial markets: sellers and buyers. At these levels, large buyers bought securities or futures contracts, and large sellers sold and took profit or fixed stop loss from these positions. Thus, with the next price moving to these support levels from top to bottom, it is very likely that large buyers will buy more positions. A similar process occurs with resistance levels as the financial instrument price comes near them from the bottom up. Large holders will sell positions to take profits. The RNN topologies can remember these previous levels, which is undoubtedly an advantage of RNNs. It was this advantage that made RNNs famous for predicting the price level in financial markets in the medium and long-term. However, support and resistance levels are not crucial in intraday trading compared to medium-term trading (Schwager, 2017).

If the chart scale is increased, then, in contrast to long periods, before significant movements, intraday tick charts tend to form a stepped pattern (of increments or decrements) rather than a sine wave (Figure 3.17). For intraday trading, patterns of behaviour of sellers and buyers are more crucial to determining whether the market will unfold, or break through the level of support or resistance and move further.

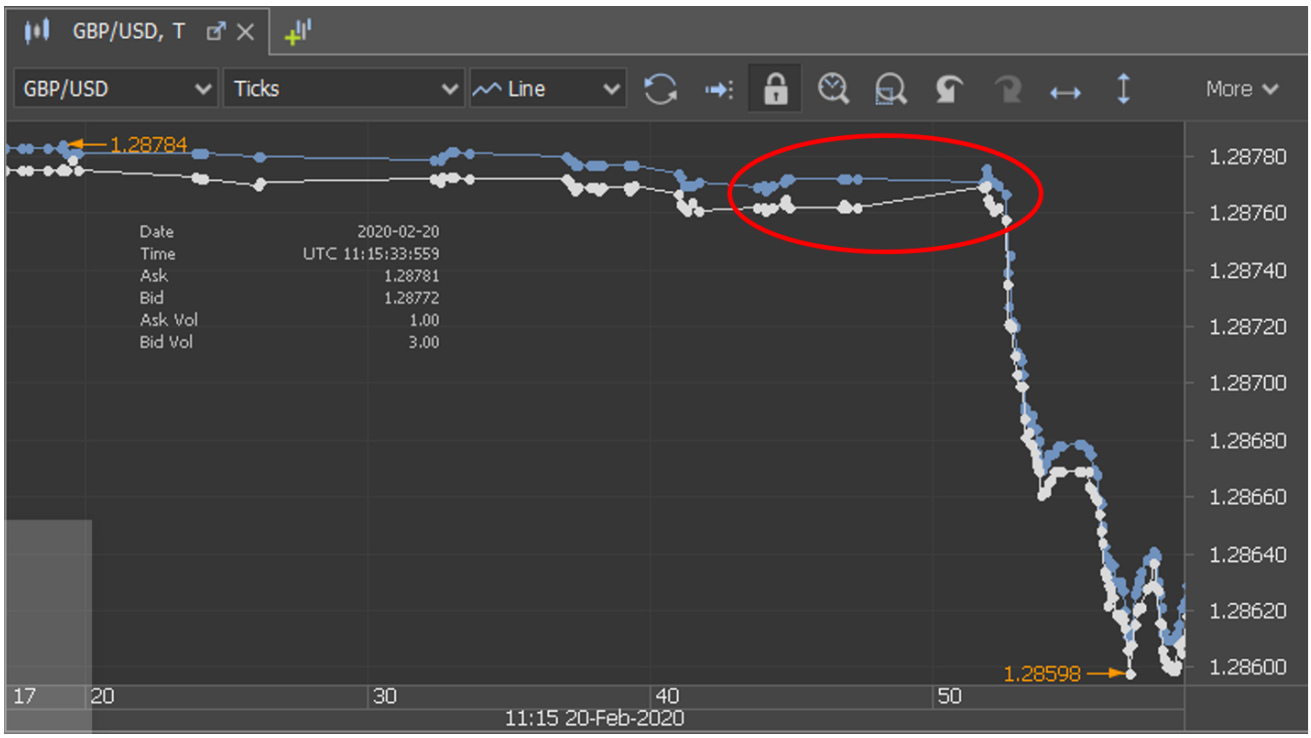


Figure 3.17: Stepped patterns in the tick-term GBP/USD Forex chart in February 2020. Screenshot from (Dukascopy Swiss Banking Group, 2022b).

The current research will focus on areas of ticks that immediately precede a significant movement (indicated by red circles in Figure 3.18). Analysing and finding patterns in these areas is similar to the analysis of images for which CNNs are so successful. Consequently, to determine local patterns, the CNN topology is more suitable (Milke, 2020).

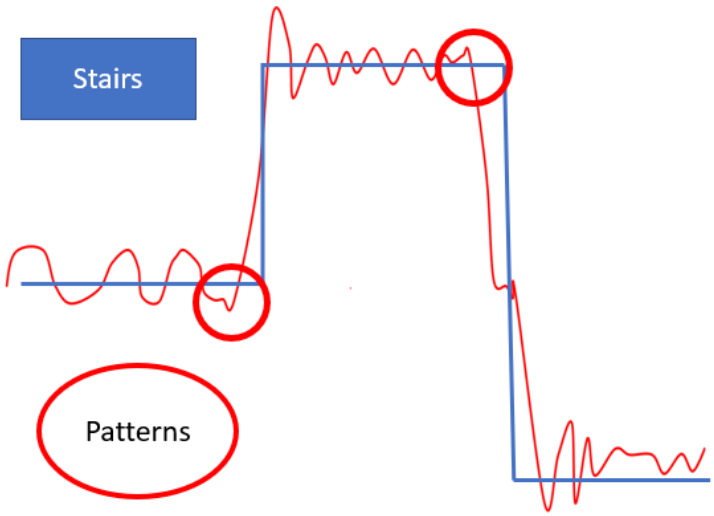


Figure 3.18: Stepped model (patterns) of short-term price movement.

Interestingly, on super long-term investing periods, the financial markets also demonstrate this stepped pattern. Figure 3.19 indicates that despite the significant return on hypothetical

investment in the S&P 500 Index over 40 years, the main profit was obtained in a matter of days. For example, if this hypothetical investor had missed only five of the most profitable days in these 40 years, then he would have lost 38% of the total potential profit; and if he had missed the 50 most profitable years in these 40 years, then he/she would have lost 93% of this profit. Night gaps also gave part of this profit (Fidelity Investments Inc., 2020).

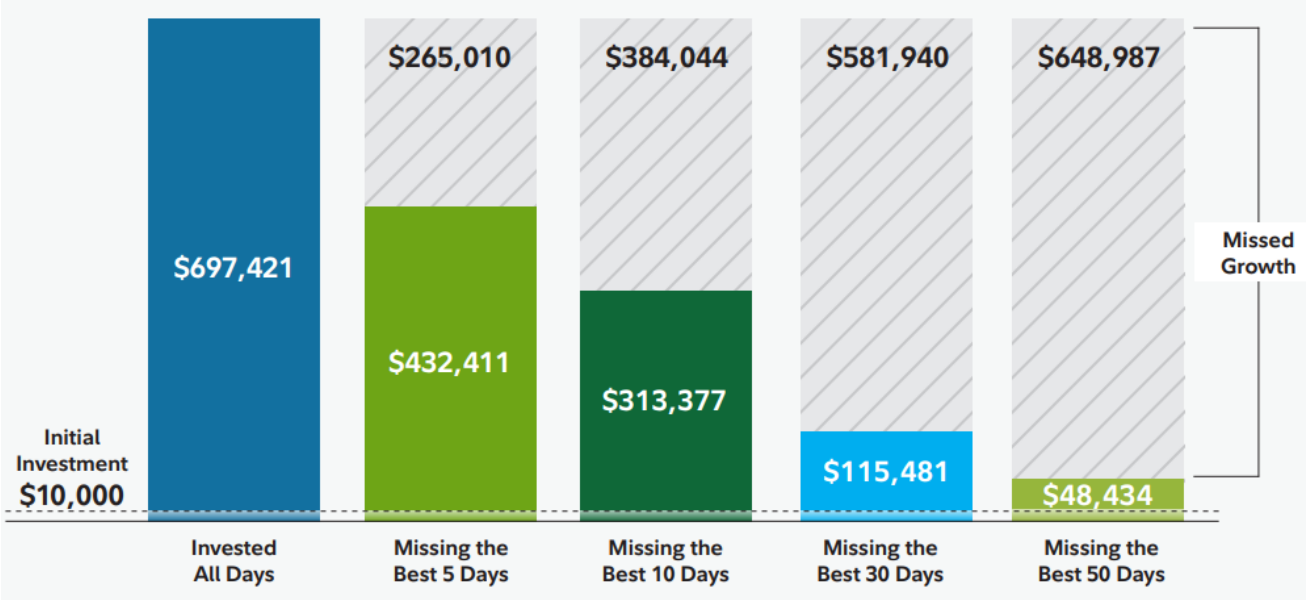


Figure 3.19: Hypothetical growth of US $10,000 invested in the S&P-500 Index from 1st January 1980 to 31st March 2020 (Fidelity Investments Inc., 2020).

3.7. Metrics

The research done by Galeshchuk and Mukherjee (2017), Aggarwal and Sahani (2020), Maneejuk and Srichaikul (2021) Moghaddam and Momtazi (2021) will be used to validate the results of the intermediate and key experiments of this research, as these papers demonstrate good performance of some neural network architectures such as RNNs (GRUs and LSTMs) and CNNs. These papers contain estimates of the **Mean Absolute Error (MAE)** loss function for intermediate experiments of this research to predict the prices of the next tick. They also demonstrate estimates of error metrics, such as **Accuracy**, for predicting the probable direction of future movements of the currency pairs' prices in the Forex market which are the key experiments of this research.

The four papers mentioned above performed a comparison of the financial data analysis results using standard statistical time series analysis methods, such as ARISMA, standard machine learning algorithms in the finance domain, such as SVM, Random Forest and

Boosting (Sezer, Gudelek and Ozbayoglu, 2020; Maneejuk and Srichaikul, 2021), and various neural network architectures. The performance of neural networks is better in most papers (Section 5.4.3 and Section 8.2), which justifies the use of standard ANNs, RNNs (GRUs and LSTMs) for validation. Thus, suitable existing approaches have been identified to compare to the results of this research.

In this chapter, a literature review of various approaches to creating ATS architectures and analysing financial markets has been conducted. It has critically analysed existing research papers that compared financial data analysis results using standard classical econometric approaches to time series analysis, standard machine learning algorithms and various deep neural network architectures. Existing papers using new ANN architectures to analyse linearly independent raw financial market data have been evaluated for their suitability for predicting price movements of various financial instruments at different time intervals.

Some interesting analytical approaches have also been demonstrated, such as using market charts as inputs to train deep convolutional neural networks. Based on existing research, the theory has been shown that RNN analysis could be seen as an evolution of Technical analysis for identifying trends, support and resistance lines. On the contrary, CNN's data analysis could be seen as an evolution of Japanese candlestick charting techniques, which have always been used mainly for short-term decisions, for example, for intraday trading.

The microstructural fluctuation patterns of financial markets and the time frame limitations of the research are also described in this chapter. It has also been established the set of standard error metrics and loss functions that allow evaluation and comparing of the results of this research with existing results.

4. Data Processing

This chapter describes initial data preparation for future experiments. It demonstrates research tools and methodology and describes preliminary experiments with visualising results.

This chapter is organised in several sections, as described below.

Sections from 4.2 to 4.4 describe the data and research tools that will be used in this research. ML and DL frameworks, programming languages that are employed in this research are listed. Methods and tools for data collection are displayed.

Section 4.5 describes the databases used. Specific methods and metrics used to evaluate trading neural network systems for financial markets are highlighted. The classic approach for avoiding overfitting as well as two cross-validation methods for financial data are also described in Section 4.6.

Section 4.7 is a preliminary analysis of the data undertaken in order to gain non-obvious correlations in data using and insights into the best neural network architecture(s) to use; this involves a statistical approach for visualising high-dimensional data known as t-Distribution Stochastic Neighbour Embedding (t-SNE method), as described by Maaten and Hinton (2008). The technique reduces a set of high-dimensional points (or features) to two dimensions with the intention that any clustering of the data can be visualised on a 2D-graph, the values of each axis of the graph being derived units generated by the t-SNE method.

Section 4.8 describes initial data preparation for future experiments.

Additionally, Section 4.9 demonstrates some preliminary testing of models and ANN architectures that allow topological appraisal without the overhead requirement for processing power and long execution times that would otherwise be needed for use with big financial data. At the end of the chapter, Section 4.10 provides some general conclusions and describes key experiments.

4.1. Introduction

In order to begin this machine learning research in finance and intraday trading, it is necessary to identify the most applicable technology for collecting the trading data. The software for visualisation (the Trading Terminal) of the trading data and results also has to be identified. Visualisation is beneficial to understanding the results and to implementing further changes to the model.

Secondly, it is necessary to select the financial market, stock exchanges and financial instruments. This choice depends on liquidity, which is the central requirement of the financial market used, and allows development beyond the results of this research to the management of financial portfolios that are bigger than were used in this research experiments.

Thirdly, it is necessary to create High-Speed Connectors (HSCs) to the stock exchanges directly or to trading terminals, which already have connections to the stock exchanges, once these trading terminals have been identified. In the second case, the connectors will tie together the trading terminals and the ML models. An alternative way is to connect through APIs of these trading terminals.

Fourthly, a widely used 'end-to-end' approach in other areas of machine learning, where raw data is fed to neural networks at the inputs, does not work for big financial data due to noise and constantly changing financial markets. Therefore, a deeper preliminary analysis of the raw input data is needed. Consequently, an essential part of this research is the transformation and advanced analysis of data before it is directly fed into the neural network. In a more generalised way, this research relates to a positivist experimental research approach (Park, Konge and Artino, 2020). The central part of this research is somewhat iterative with each stage consisting of creating and coding new ideas for the algorithmic trading and ML models, with these models being improved at each stage. Each change has to be checked first with historical data and then via online trading. The testing against historical data is known as backtesting or paper-trading. One problem associated with classical methods for evaluating machine learning models is that there are error metrics, such as accuracy, precision and recall. ML models and ANNs must be evaluated, not in isolation, but in the context of a real trading system.

Methods and tools for analysing financial markets that use ML and ANN architectures have been described as indicated in the literature review (Chapter 3). However, problems lie in the formulation of the goals, the paradox being that neural network models aimed at predicting

the future price of securities and maximising profits can give unstable or negative results. This is despite the reasonable formal estimates of error metrics, such as accuracy, MSE and MAE (Wu, et al., 2020a). It is necessary to determine other atypical research goals in order to obtain stable predictions, such as the probability of any particular direction and depth of movement of a financial instrument in the short term. It is also necessary to determine the target search functions for whole trading system and add simple crash tests that signal in advance a possible significant drop in the market, such as those that occurred in March 2020 due to the COVID-19 epidemic (Tabar, Sharma and Volkman, 2020). It is likely that this objective target function is not the maximisation of profit, but rather the minimisation of losses and stop-losses. Since there is not enough processing power for training large ANNs on an end-to-end basis, preliminary preparation of financial market data and the search for unobvious combinations of input data (features) are needed to obtain stable results.

4.2. Conceptual Research Framework

Figure 4.1 shows the conceptual framework of the central iterative process of this research, which is based on Miles and Huberman's principals (Miles and Huberman, 1984). The word 'YES' on the arrows means that the neural network model can recognise and predict patterns of the entry and exit points to/from the financial market. These points should provide a probability of successful intraday trading of more than 50%. Thus, the probability is one of the possible target functions of the neural network model. The word 'NO' means that the neural network model should be changed because it produces unstable results, leads to losses, or the neural network error metrics do not meet the required values. Even if the target probability is achieved, tools of risk and money management must still be used in order to make a profit.

The purpose of this framework is to clearly show how a successful trading system should be created, tested and improved (research aim). However, dividing this aim into the research objectives made it easier to compare every step and validate with the standard error metrics reported in existing papers.

The online trade test mode on new historical data is necessary to detect overfitting. Many brokerage companies allow trading on a testing account. The real trading test will be conducted only to confirm results and to assess the impact of delays in the execution of market orders.

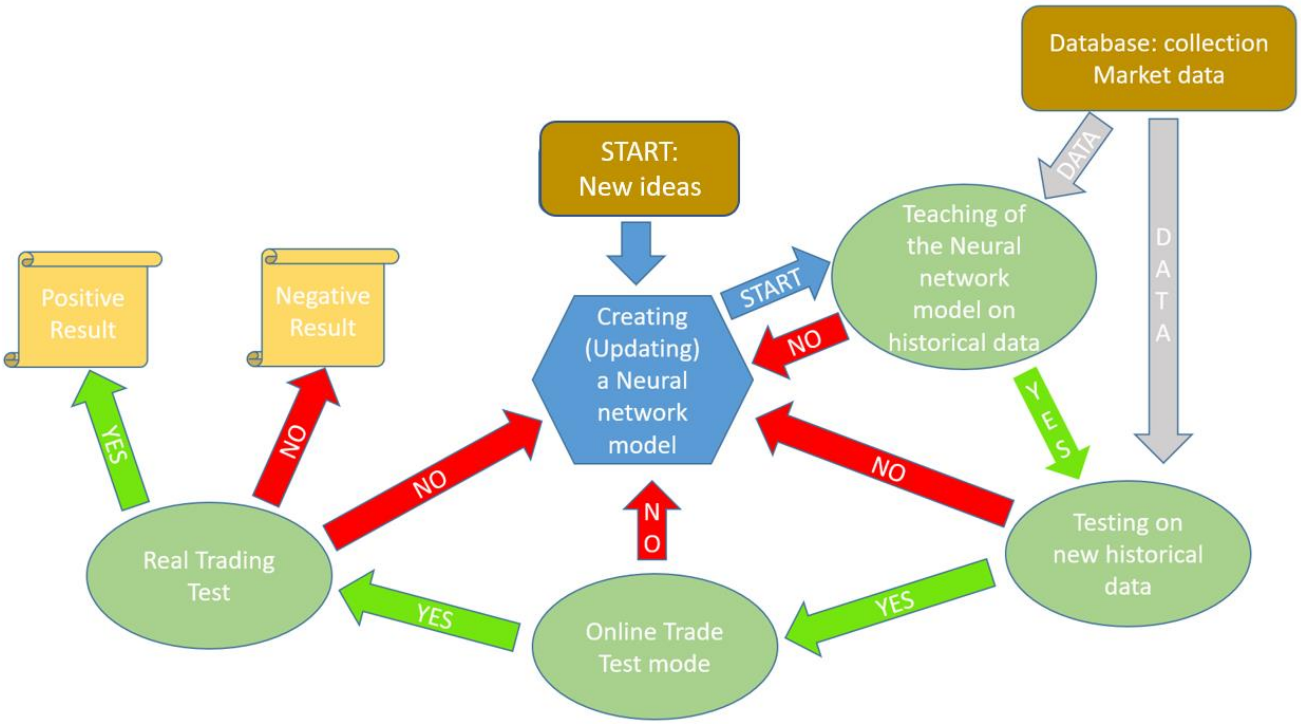


Figure 4.1: Research Conceptual Framework.

4.3. Research Method Tools

In the context of the Research Objective 1 (OB-1) (Section 1.3), the author has analysed existing financial market terminals and ML frameworks that are most suited to achieving the research objectives.

4.3.1. Financial Market Terminals

Even though it would be possible to create entirely new ATS, a far better way is to use well-known trading terminals with programming options. The market has many different trading terminals; JForex Dukascopy, CQG, Interactive Brokers (IB) and MetaTrader-5 (MT-5) were chosen to support these research tests after scrutinising a wide range of options.

- **JForex Dukascopy**

The JForex Dukascopy platform is owned by Dukascopy Bank (Switzerland), which is a bank and brokerage company and provides access to the FOREX market and historical data, including big tick data and tick volumes. Providing information on tick volumes of the non-centralised Forex currency and commodities markets is a significant competitive

advantage of the JForex Dukascopy platform compared to other Forex trading platforms that provide information only on the prices of financial instruments.

- **Commodities/Comprehensive Quote Graphics (CQG)**

The leading software solutions for online trading for the American financial market is Commodities/Comprehensive Quote Graphics (CQG Inc, 2022), which provides trading data from more than one hundred exchanges. CQG has an Application Programming Interface (API) that can export data to another application, which can then be used to analyse and create trading signals. This is the most useful option for organising the connectors to the Chicago Mercantile Exchange (CME), where most American futures contracts are traded.

- **Interactive Brokers**

The Interactive Brokers (Interactive Brokers LLC, 2022) platform is owned by the well-known American brokerage company of the same name. It is one of the largest electronic brokerage platforms in the United States and a principal Forex broker (Interactive Brokers Group, 2022). The platform also has an API connector for receiving data and for direct connection to external systems.

- **MetaTrader - 5**

One of the well-known trading terminals for connection to the Moscow Exchange (Moscow Exchange, 2022) is the MetaTrader-5 (MT-5) (MetaQuotes Software Corp, 2022a). It is a standard functional trading platform with a technical analysis interface and algorithmic trading capabilities. The MT-5 uses MetaQuotes Language 5 (MQL5), which is a high-level object-oriented language based on C++ (MetaQuotes Software Corp, 2022b). The language has a vast community, which supports and creates good quality open-source code for intraday trading, high-frequency trading (HFT), statistical analysis, arbitrage and optimisation.

4.3.2. APIs and High-Speed Connectors for Day Trading

This section describes the APIs and high-speed connectors, which are necessary for testing the created neural network models in the live financial market. It is not possible to fully simulate the financial market environment, in particular, the market slippage parameter. Therefore, after testing and evaluating ANNs using conventional methods of evaluation of ML algorithms, the final test should be on a live financial market, which can be carried out with minimum volume (a minimum lot).

- **The High-Speed Connectors**

Integrated Client Trading Terminals (ICTT) allow high-speed connection to the stock exchanges (Figure 4.2). Each brokerage company chooses one or some ICTTs, which its clients use. Usually, the terminals are used manually. Some brokerage companies provide and support high-speed channels to stock exchanges with ICTTs that support automatic trading by ATS. In the case of simple algorithmic trading, the ATS with a defined strategy replaces the role of the connector. When neural network models are in charge of the strategy, connectors between the ICTTs and the neural network models are needed. It is possible to create such connectors based on the program code of ATS. The author modified the connector-1 based on the ATS that is described in (Mashnin, 2016). The code of the connector is shown in Appendix VI.

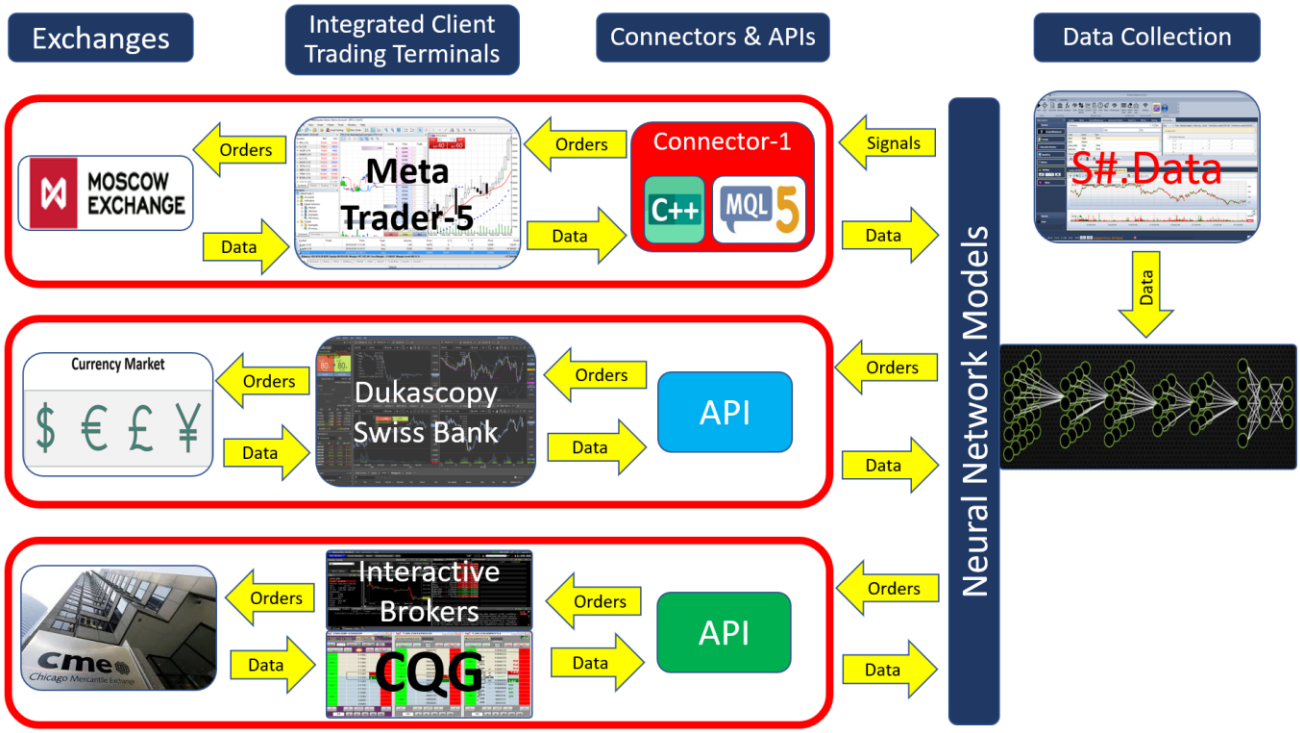


Figure 4.2: Interconnections between the modules.

- **Real up and real down movements should be analysed separately since the patterns Implementation and Testing**

Two versions of the high-speed connector-1 for day trading on the futures market on the Moscow Exchange were created. One used object-orientated programming, and the other one used procedural programming. Both versions had been implemented in C++. Some functions use MQL-5 that has a similar syntax to the C++ language. MQL-5 is an event-driven

C++ language. It analyses events from the client trading terminal MetaTrader-5, such as putting a new tick or any change inside an order book. The event may also be a periodical signal from the internal timer that is a function of MQL-5. The internal timer may send the signal each minute or each second or millisecond. In reality, the minimum time-period that may be calculated is about ten milliseconds because it depends on the Internet channel connection speed. The most useful function of MQL-5 is the sending off different orders, such as Stop-Loss Order, Limit-Order, Stop-Order, Trailing-Stop Order, Market-Order, Hidden ('Iceberg') Order, Time-in-Force Order, and 'All-or-None' Order, among others.

The code of the two connector-1 was tested using three different strategies with two different futures contracts. The connector works correctly. All types of orders are sent to the financial market. The code checks itself as well as the parameters of the connection and trading account.

- **API**

Recently, most major brokers provide their clients with direct access to the trading system through APIs. This saves time on developing full-stack applications that include neural networks. In this way, these brokers take responsibility for consistently transmitting orders and receiving real-time data necessary to train and use neural network models. For connecting to the foreign exchange market, the author used the API from Dukastcopy Swiss Bank. Detailed connection instructions are provided in the Dukascopy FIX API Programming Guide (Dukascopy Swiss Banking Group, 2022a). Also, for connecting to the futures markets, API from Interactive Brokers was used. The company provides direct access to data through their Trader Workstation Application Programming Interface (TWS API), which allows you to create custom trading applications in Java, .NET (C #), C ++ or Python, including using ANNs (Interactive Brokers, 2022). Both organisations provide direct access to the market data stream in the condition that the client has to be a professional securities market participant with confirmed licensed status.

- **Open-Source Connectors**

OS.Engine (Van, 2022) is an open-source framework for creating connectors to popular trading platforms and making ATSs. It has connections to the Interactive Brokers (Interactive Brokers LLC, 2022), OANDA (OANDA Corporation, 2022), Quick (ARQA

Technologies, 2022), Plaza-2 (Moscow Exchange, 2018), etc. The framework is based on C# programming language and is used in this research to test the final model using paper-trading.

4.3.3. Machine Learning Software

- **TensorFlow**

At the end of 2015, Google developed and presented a new software library for machine intelligence called ‘TensorFlow’.

> It is *“for numerical computation using flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows deployment of computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API”* (Google LLC, 2022).

The main difference between TensorFlow and other similar systems is that it can support different software platforms. Moreover, the documentation on Python API (Application Programming Interface) is qualitative. The technology is described in more detail in the Preliminary White Paper from the Google Research Department (Abadi, et al., 2016).

The TensorFlow is the basis of many DL frameworks which use TensorFlow as a 'mother platform' because it achieves fast calculation and rational use of processor resources.

- **Scikit-Learn**

The scikit-learn (Cournapeau, 2022) is an ML library in Python programming language (Mueller and Guido, 2017). It has a large active society, which supports and develops the library. The scikit-learn has high-quality documentation for each algorithm, such as classification and regression. Moreover, it has clustering algorithms and other ML-methods, such as support vector machines, t-Distributed Stochastic Neighbor Embedding (t-SNE), random forests, k-means, gradient boosting, etc.

- **Keras and TensorFlow 2.0**

Keras (Chollet, 2022) is an neural network library in Python. The framework aims to build on top of other DL low-level ML frameworks, such as TensorFlow, Theano, etc (Gulli and Pal, 2017). It simplifies the organisation of complex DL models. Many researchers prefer to use Keras (Rampasek and Goldenberg, 2016; Fischer and Krauss, 2018), and it is actively

used in this research. At the current time, the new version of this framework is combined with TensorFlow in the complex framework 'TensorFlow 2.0' and is supported by the Google Corporation. TensorFlow is a generic software library for numerical computation and machine learning using data flow graphs, whilst Keras is a neural network library that interfaces and supplements TensorFlow. Therefore, its use in conjunction with TensorFlow which is from the same corporation is the most logical. Also, it is possible to use Keras in JavaScript programming language with a Keras.js framework and run neural network models in the browser (Chen, 2022). Most of the neural network models created in this research use Keras.

- **PyTorch**

The PyTorch (Paszke, et al., 2017) ML framework was developed by the Facebook company and was widely introduced in 2017. PyTorch is quickly gaining popularity due to the ability to work with dynamic computational graphs and a lower level of program code.

Keras, TensorFlow 2.0 and Scikit-learn were used for this research. They were chosen after scrutiny of all existing ML and DL software.

- **Programming Languages**

Most of the programming code in this research is written in Python, one of the most popular (TIOBE, 2021) and mainstream languages used for machine learning, with an advanced set of specialised libraries and a developed professional community (Portugal, Alencar and Cowan, 2016). However, some highly loaded parts of the code use the C language, as well as MQL5, which is a specialised programming language for the MetaTrader-5 trading platform and is based on C++.

4.3.4. Environments: Operation Systems, Anaconda, TensorFlow, Keras and Google Colab

In this research, three different programming environments (operation systems) were used for testing and timing the DL models:

- Windows 10
- MacOs High Sierra

- Google Colab, which is a research tool for machine learning (Google Colaboratory, 2022).

Installing the Anaconda, Python-based TensorFlow-2 with Keras software on Windows 10 and macOS are standard tutorials that are available online (Google LLC, 2022).

```
Train on 45000 samples, validate on 5000 samples
Epoch 1/25
45000/45000 [==============================] - 281s 6ms/step - loss: 1.5065 - acc: 0.4488 - val_loss: 1.1188 - val_ac
c: 0.5956
Epoch 2/25
45000/45000 [==============================] - 286s 6ms/step - loss: 1.1245 - acc: 0.6017 - val_loss: 0.9034 - val_ac
c: 0.6862
Epoch 3/25
45000/45000 [==============================] - 290s 6ms/step - loss: 0.9694 - acc: 0.6534 - val_loss: 0.8388 - val_ac
c: 0.7094
Epoch 4/25
45000/45000 [==============================] - 289s 6ms/step - loss: 0.8755 - acc: 0.6922 - val_loss: 0.7461 - val_ac
c: 0.7502
Epoch 5/25
45000/45000 [==============================] - 290s 6ms/step - loss: 0.8055 - acc: 0.7183 - val_loss: 0.7278 - val_ac
c: 0.7476
Epoch 6/25
45000/45000 [==============================] - 299s 7ms/step - loss: 0.7523 - acc: 0.7358 - val_loss: 0.7103 - val_ac
c: 0.7544
Epoch 7/25
45000/45000 [==============================] - 300s 7ms/step - loss: 0.7112 - acc: 0.7514 - val_loss: 0.6596 - val_ac
c: 0.7772
Epoch 8/25
45000/45000 [==============================] - 292s 6ms/step - loss: 0.6726 - acc: 0.7629 - val_loss: 0.6520 - val_ac
c: 0.7770
Epoch 9/25
45000/45000 [==============================] - 395s 9ms/step - loss: 0.6403 - acc: 0.7744 - val_loss: 0.6625 - val_ac
c: 0.7826
```

Table 4.1: Period for learning a Convolutional Neural Network (CNN) by Central Processing Unit (CPU).

```
Train on 45000 samples, validate on 5000 samples
Epoch 1/25
 - 25s - loss: 1.5541 - acc: 0.4328 - val_loss: 1.1850 - val_acc: 0.5846
Epoch 2/25
 - 24s - loss: 1.1328 - acc: 0.5988 - val_loss: 0.9571 - val_acc: 0.6638
Epoch 3/25
 - 24s - loss: 0.9748 - acc: 0.6567 - val_loss: 0.8255 - val_acc: 0.7058
Epoch 4/25
 - 25s - loss: 0.8732 - acc: 0.6924 - val_loss: 0.7750 - val_acc: 0.7388
Epoch 5/25
 - 25s - loss: 0.8031 - acc: 0.7184 - val_loss: 0.7865 - val_acc: 0.7320
Epoch 6/25
 - 25s - loss: 0.7567 - acc: 0.7342 - val_loss: 0.6900 - val_acc: 0.7646
Epoch 7/25
 - 25s - loss: 0.7159 - acc: 0.7476 - val_loss: 0.6885 - val_acc: 0.7690
Epoch 8/25
 - 25s - loss: 0.6785 - acc: 0.7629 - val_loss: 0.6633 - val_acc: 0.7774
Epoch 9/25
 - 24s - loss: 0.6422 - acc: 0.7742 - val_loss: 0.6624 - val_acc: 0.7698
Epoch 10/25
 - 25s - loss: 0.6196 - acc: 0.7804 - val_loss: 0.6660 - val_acc: 0.7812
```

Table 4.2: Period for learning a CNN by Graphics Processing Unit (GPU).

Google Colab is a cloud-hosted development environment for creating machine learning applications and does not require any special installation procedure. It is constructed as a Jupyter Notebook for Python-3 and uses GPU and TPU technology for accelerated learning of ANNs. Table 4.1 and Table 4.2 show the epoch periods taken during the learning of a CNN

with about 1.7 million variables. The improvement in error minimisation for each iteration (Epoch) is about ten times longer than the non-GPU approach. Unfortunately, the Google Colab environment has limitations and is available for no more than 12 hours of continuous operation.

4.3.5. PC Hardware

For prototyping neural network models, in addition to the publicly available GPU cloud platform Google Colab, a laptop with a GPU processor was also used, the configuration details of which are indicated in Table 4.3.

Hardware Configuration Details: (Lenovo Legion Y540-17IRH)	
CPU	Intel Core i7-9750H (2.60 GHz, up to 4.50 GHz with Turbo Boost, 6 Cores, 12 MB Cache)
GPU	NVIDIA GeForce RTX 2060 6 GB GDDR6 192bits
RAM	32 GB DDR4 2666MHz SoDIMM
Hard Drive	2TB, 5400RPM, 2.5'', 7mm
Solid State Drive	256 GB, M.2 2280

Table 4.3: PC hardware with GPU used.

4.4. Data Collection Tools

For this research, the neural networks for the trading systems have been trained and tested with linearly-independent, publicly-available raw data, such as price, time, volume and order book (Milke, Luca and Wilson, 2017). A library of completed transactions (known as ‘deals data’) should collect raw data based on the smallest unit of integrated data called a ‘tick’. A tick excludes detailed information about each deal but mainly contains the slightest upward or downward price fluctuation on a financial market.

Integral market data for prices and volumes, such as hourly or daily time frames, are widely available. On the contrary, access to tick data is usually paid. However, some global brokerage companies, such as Dukascopy Bank or Interactive Brokers, provide access to historical data, including tick data to those clients with a live account. Some data aggregators, such as TS-Lab (TSLab, 2022), specialise in selling trading software while providing access to tick data. One of these companies is Stock Sharp. Their product, S#.Data, was chosen as a database because it is capable of accumulating and storing raw data (Stock Sharp, 2021).

Figure 4.3: Order book. Screenshot from (CQG Inc, 2022).

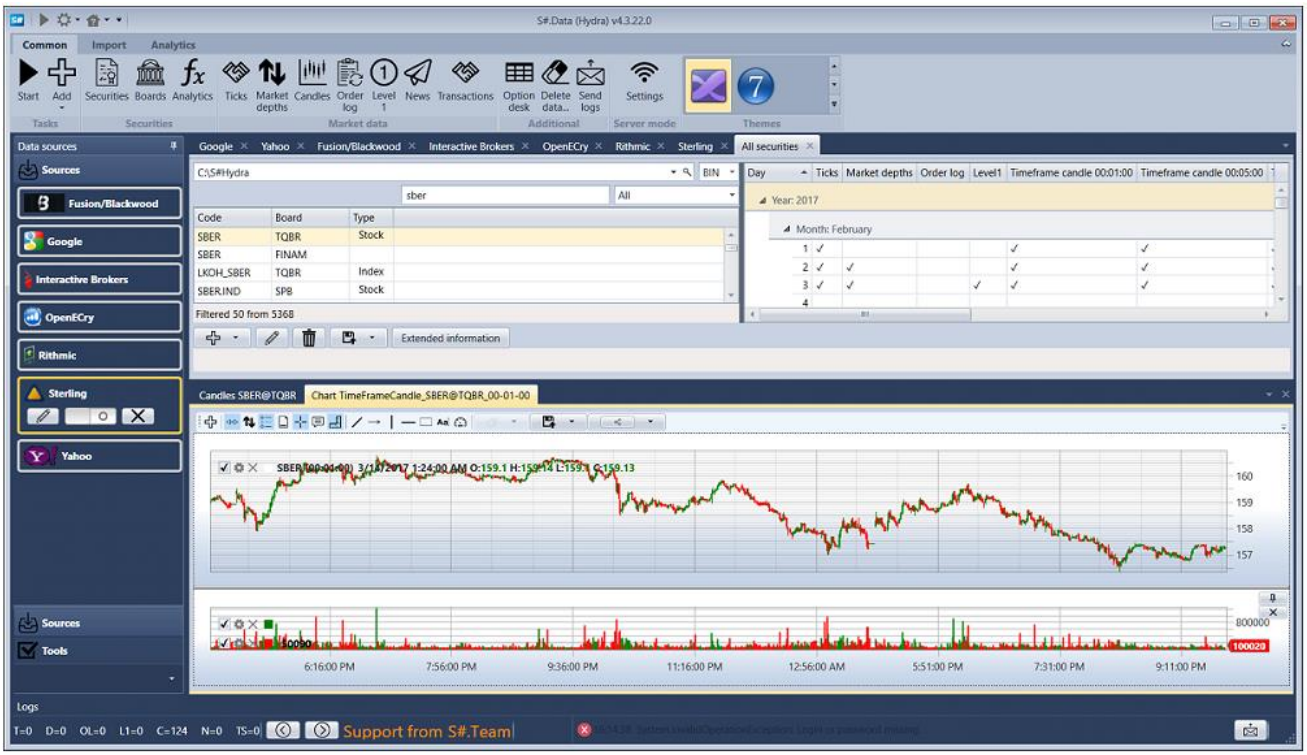


Figure 4.4: S#.Data terminal. Screenshot from (Stock Sharp, 2021).

The framework has the capability to download the market maker's data from different sources (Stock Sharp, 2020) and keep it in a local database. The popular text format for machine

learning ‘csv’ can be used by the program as much as other formats including ‘xml’ and ‘txt’. The data may be used for training and testing DL models. Also, S#.Data may collect real-time data, including an order book (Figure 4.3), which consists of many uncompleted buy and sell orders. The interface of the program is displayed in Figure 4.4.

After starting the online testing, the data from the selected trading systems will be used directly. At the same time, the data will continue to be collected for future ML models. In this case, the database will collect not only information about completed deals but also about all uncompleted orders. This data may appreciably change the ML result.

Online data available in the public domain is in obtained from the Chicago Mercantile Exchange (CME) and the Moscow Exchange (directly accessible by the author as a client of brokerage companies which operate on these stock and futures exchange markets). This publically available collated data include information about all uncompleted orders.

4.5. Databases Used

4.5.1. S&P-500 Index and Stock Market Databases

The S&P-500 (Standard & Poor’s 500) (S&P Dow Jones Indices, 2022) index is an American stock market index, which is calculated on the basis of the five hundred largest stock market listed companies. Consequently, it depends on the prices of the stocks of the 500 companies. Therefore, the price of S&P-500 might be predicted based on previous prices of all 500 shares that are included in the index.

Many stock market databases with integral data are available as open-source. For checking new ANN models, this current research uses one-minute time frame data of market prices and volumes. On one hand, this time frame is quite detailed for a description of intraday trading. On the other hand, a one-minute time frame does not conclude a massive amount of data because it does not consist of tick information. This research uses the S&P-500 Index (one-minute time frame) from the S#.Data.

4.5.2. EUR/USD Forex Tick Database

For the main experiments with CNNs this research uses tick-term data of market prices and volume. The research uses the publicly available EUR/USD Forex tick data, which was

downloaded from the Dukascopy Bank (Switzerland) historical data resources (Dukascopy Swiss Banking Group, 2022b). The data consists of the prices of supply and demand (ask and bid), volumes of supply and demand (ask volume and bid volume), as well as the date and time of transactions with an accuracy of milliseconds. The size of the volume is specified in millions of lots. The author understands that the amount of volume lots on Forex through any market maker does not carry complete information about global volumes in that moment. Thus, it is possible to miss a significantly large amount that occurred through other market makers. However, further training of the neural network will take into account the important parameter of the time interval between two adjacent ticks, which partially compensates for the incompleteness of information in the amount of volume, since the interval between ticks usually decreases as the volume of transactions increases. Figure 4.5 shows the charts of prices and volumes separately for the four months of 2018. Figure 4.6 shows only the 400 first ticks in 2018, which allows for a more detailed demonstration of the structure of the analysed data.

Figure 4.5: Price and volume graph of EUR/USD Forex from 01.01.2018 util 30.04.2018.

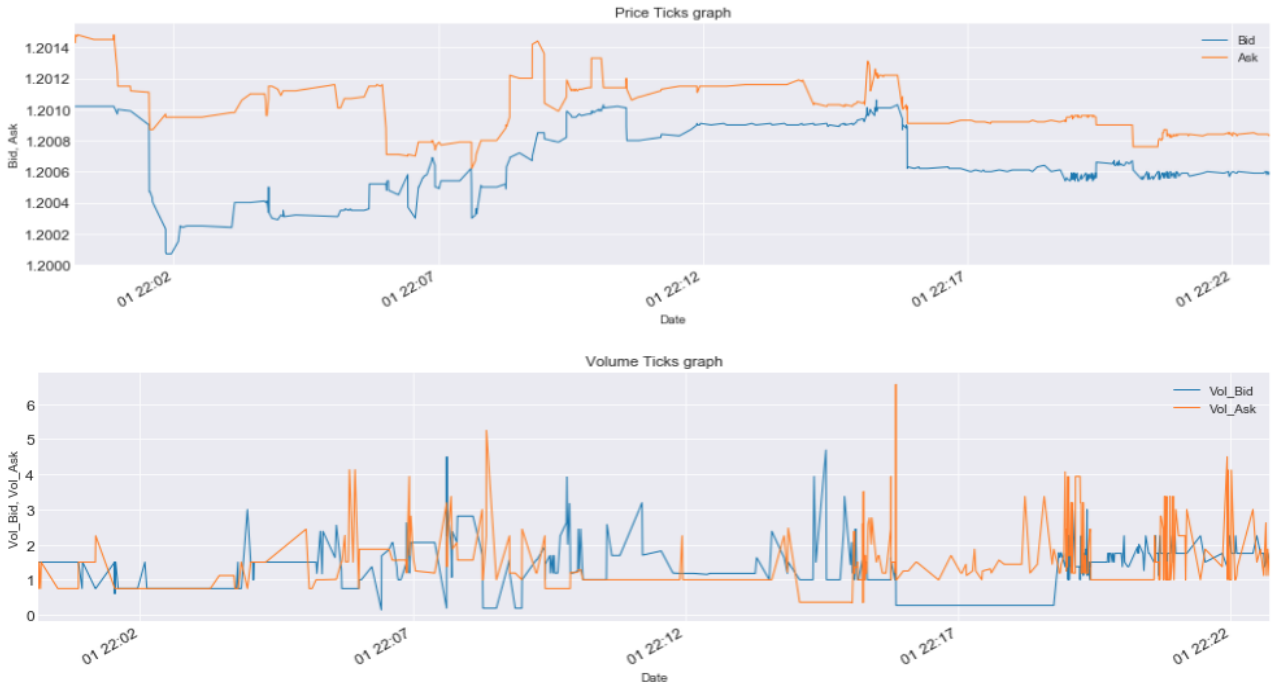


Figure 4.6: Price and volume graph of EUR/USD Forex: first 400 ticks.

4.6. Nuances of Validation and Valuation Neural Networks Models for Financial Markets

Most researchers create neural networks for financial market analysis and use standard error metrics and loss functions, such as accuracy, recall, precision, F1 and Mean Squared Error (MSE), Mean Absolute Error (MAE) or Huber loss. These metrics evaluate various error parameters and ultimately maximise the profitability of the trading system. However, it is not enough and not always correct to evaluate the results of neural networks for financial markets only based on conventional methods for evaluating the quality of the neural network results because there are nuances. This subchapter indicates some specific methods and metrics used to evaluate trading neural network systems for financial markets. This subchapter also describes both the classic approach to avoiding overfitting and two cross-validation methods used by this research.

4.6.1. Overfitting and Validation of Trading ANN Models

- **Overfitting**

Sometimes, ANNs may be too restrictive in scenarios where they capture all relations and dependencies of a particular dataset to such an extent that they can only recognise and correctly classify data that is identical to the training data. In these situations, the ANN does not have an excellent performance on the validation (test) data and is said to be overfitted.

Figure 4.7 demonstrates this where the error (loss) decreases during the iterative training events (epochs) and then increases in the validation sets.

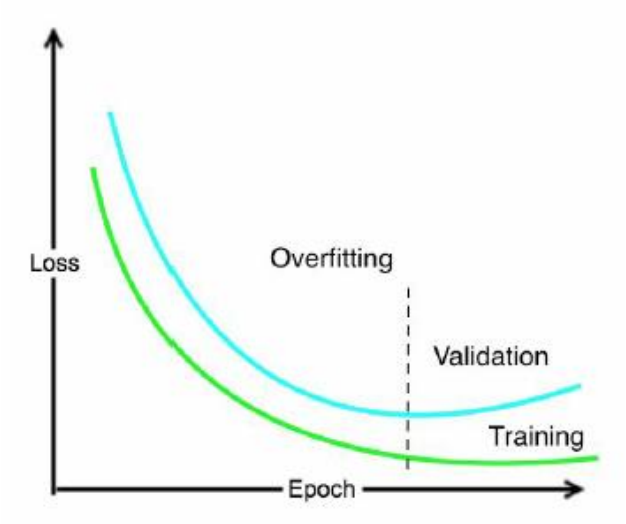


Figure 4.7: Indicator of overfitting (Gulli and Pal, 2017).

The most well-known methods of avoiding overfitting are stopping the training of an ANN as soon as the accuracy has started to decrease on the validated dataset; and the 'dropout' method when randomly selected cells of the ANN and their connections (weights) to other cells are dropped (Srivastava, et al., 2014). The second method is challenging to apply in cases when the additional weekly retraining should be done on weekly replenished online market data.

- **Training, Validation and Testing**

To avoid overfitting, 80% of the dataset was used for training, and 20% of it was reserved for validation, as shown in Figure 4.8. A typical public database consists of two parts: training data and testing data. However, it is a good practice to keep about 20% of the training data for the validation process, which is necessary for identifying the overfitting, if any occurs. Each epoch used by the ML algorithm uses training data and then checks the model with the validation part. The test data must also be separated from the training process. Otherwise, the result of the training will not be reliable.

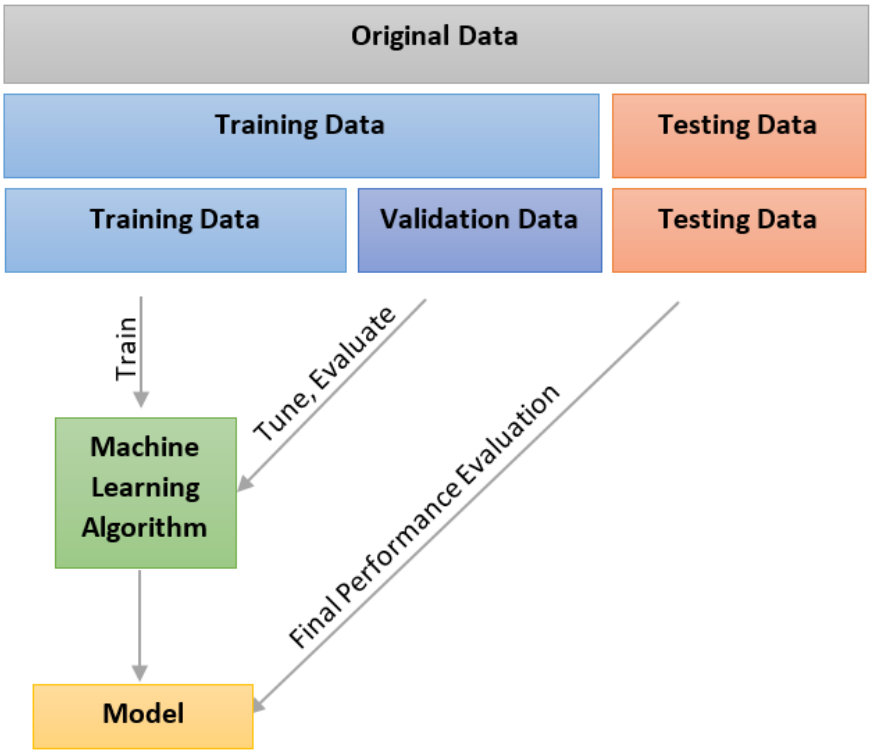


Figure 4.8: Splitting an original dataset into training, validation and testing parts (Eissa, 2016).

This method allows the estimation of the actual recognition accuracy. The accuracy of the validation part of the dataset is an objective function. If the accuracy does not change, the process of the training should be stopped.

- **Cross-Validating**

There is a significant difference between the time series and a simple data sample as, for the first one, the analysis has to take into account the time associated with the measurements and not just the characteristics and statistical diversity of the sample. Conventional cross-validation is not applicable for time series. It is not permissible to mix the values of a time series in an arbitrary order. Otherwise, interconnections will be lost, and values from the future will be known, which is impossible in the process of financial trading (Bergmeir, Hyndman and Koo, 2018). In this case, 'cross-validation on a rolling basis' is used. The sequence of this type of cross-validation is shown in Figure 4.9. At the first step, the neural network is trained on a small part of the time series, from the beginning to time t. A forecast is made for $t + n$ ticks (steps) forward, and an error is considered. Next, the training

segment is increased to $t + n$ ticks, and the values are predicted from $t + n$ to $t + 2n$, etc. Training and validation continue until the end of the historical data.

With the arrival of new data (time series), retraining of the neural network will continue in the same manner as the cross-validation on a rolling basis. Sometimes this type of cross-validation is called the 'successive training set'.

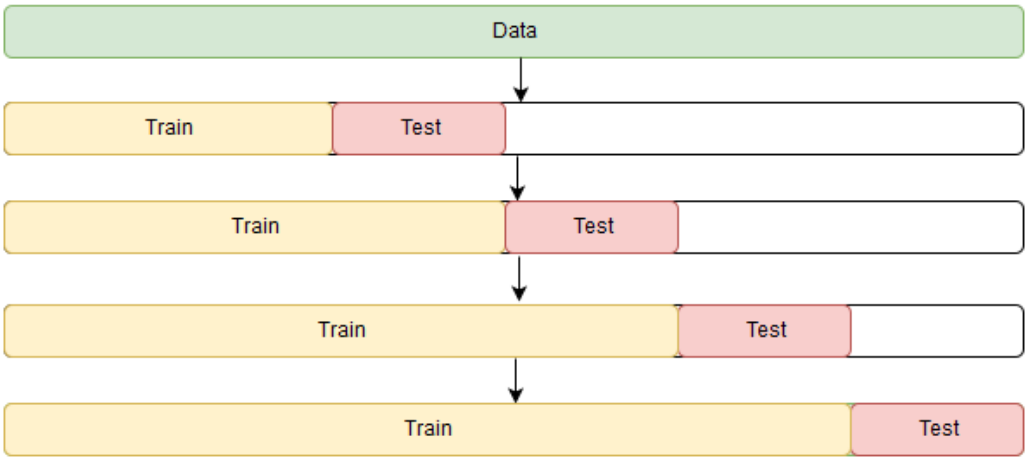


Figure 4.9: Cross-validation on a rolling basis (Bergmeir, Hyndman and Koo, 2018).

For particularly large data sets, cross-validation with the 'rolling training set' is used (Figure 4.10). This validation is often used when retraining ANNs with new data already obtained in the process of using the model. In the case of intraday trading, this validation can be done on a daily or weekly basis after the market closes overnight. This research will use both of the types of rolling cross-validations mentioned-above.

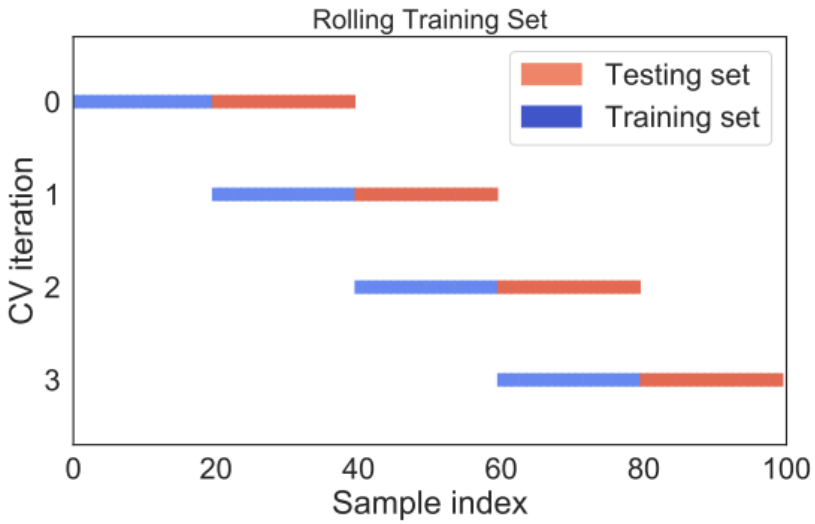


Figure 4.10: Cross-validation with the ‘rolling training set’ (Jiang, 2020).

4.6.2. Nuances of Valuation Neural Networks Models for Financial Markets

Sometimes strategies with high returns also have large drawdowns of equity in the middle of the assessed period. In other words, often, a strategy with lower returns is more attractive due to the reduced risk. To assess profitability and risk at the same time, the Sharpe ratio is used (Sharpe, 1998). The Sharpe ratio evaluates the relationship between profit potential and risk strategy. The higher the Sharpe ratio, the better the results that the trading strategy shows concerning the accepted risks.

The Sharp Ratio definition is:

$$S_a = \frac{R_a - R_f}{\sigma_a} \quad \textbf{(4.1)}$$

Where R_a is the asset return for the fixed time-period;

R_f is the risk-free return, such as bank deposits or U.S. Treasury security;

σ_a is the standard deviation, which is equivalent to the average volatility of a currency pair or futures contract for the time-period.

The equation shows that the value of the Sharpe ratio is directly proportional to the percentage of profitability of a trading system, which is based on an ANN, and inversely proportional to the spread of its performance. In other words, the more significant and more stable the average income of a trading system, the higher the value of this ratio.

There are other metrics for the comprehensive assessment of the profitability and risk of financial models, such as Sortino ratio (Goel, Sharma and Mehra, 2019) and Calmar ratio (Almahdi and Yang, 2019).

4.7. Preliminary Data Analysis

This section discusses some methods of preliminary data analysis, including visualisation. The preliminary data analysis was an essential element in the development of

this research, as it helped understand the data structure before using neural networks, which require massive GPU computing power. Furthermore, this preliminary data analysis showed which of the available features are more important. This analysis allowed the author to formulate preliminary ideas on one of the key novelties of this research – the probability vector of future significant movements. It also focused efforts on searching for deeper patterns, instead of simultaneously exploring all possible options. This approach reduced resources and sped up this research. For this preliminary data analysis, several random sequential data sets of big financial tick data for 2019 were used.

4.7.1. t-Distributed Stochastic Neighbor Embedding (t-SNE Method)

For visualisation, multidimensional data is transformed into two or three-dimensional space. As a tool to achieve this goal, the t-Distributed Stochastic Neighbor Embedding (t-SNE) method is well established. The t-SNE is a method of non-linear dimensionality reduction and visualisation of multidimensional variables. The method was developed by Maaten and Hinton (2008).

The classical t-SNE method is formulated in the following description. There is a data set with points (rows in the data set) described by a variable, which has more than three space dimensions. In other words, the number of features (columns) for each row in the data set is more than three. The goal of this transformation is to get a new variable, that exists in two-dimensional or three-dimensional space, which preserves the maximum extent of the structure and patterns in the source data. The t-SNE begins by transforming the multidimensional Euclidean distance between points into conditional probabilities that reflect the similarity of points of the data set. Equation (4.2) below demonstrates the calculation of these probabilities.

$$p_{j|i} = \frac{exp\left(-\left\|x_i - x_j\right\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq j} exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)} \quad \textbf{(4.2)}$$

Where σ_i is the variance of the Gaussian that is centered on datapoint x_i ;

x_j and x_i are datapoints.

The similarity of datapoint x_j to datapoint x_i is the conditional probability $p_{j|i}$.

This equation shows how close the point x_j is to the point x_i with a Gaussian distribution around x_i with a given deviation σ. The deviation will be different for each point. Areas with a higher density should have more minor variance than neighbouring points.

The resulting two-dimensional or three-dimensional points are called y_i and y_j . This pair corresponds to the original multidimensional points x_i and x_j . The conditional probability $q_{j|i}$ for y_i and y_j is calculated using the previous equation. Standard deviation equals $1 / \sqrt{2}$ (Equation (4.3)).

$$q_{j|i} = \frac{exp\left(-\left\|y_i - y_j\right\|^2\right)}{\sum_{k \neq i} exp\left(-\|y_i - y_k\|^2\right)} \tag{4.3}$$

Where y_i and y_j are the low-dimensional counterparts of the high-dimensional datapoints x_i and x_i .

If the low-dimensional points y_i and y_j correctly model the similarity between the initial high-dimensional points x_i and x_i , then the corresponding conditional probabilities $p_{j|i}$ and $q_{j|i}$ will be equivalent.

4.7.2. t-SNE for EUR/USD FOREX Database

The t-SNE method requires an extensive computer resource for a large database. The necessary computing time grows in proportion to the quadratic function of the number of rows of the dataset. For example, for 1,000 rows of the EUR/USD Forex dataset, it took about four seconds for Intel I-7 CPU 2.6 GHz and GPU NVIDIA GeForce RTX 2060 (Section 4.3.5). For 10,000 rows, it took more than a minute, and for 100,000 rows it took 20 minutes. Obviously, it is almost impossible to display 16 million lines of this tick dataset for the first half of 2019. The author analysed several random sequential data sets of 1,000, 10,000, 50,000, and 100,000 rows.

The all Python code for using the t-SNE method for EUR/USD forex dataset is attached in Appendix VII as the Jupyter Notebook.

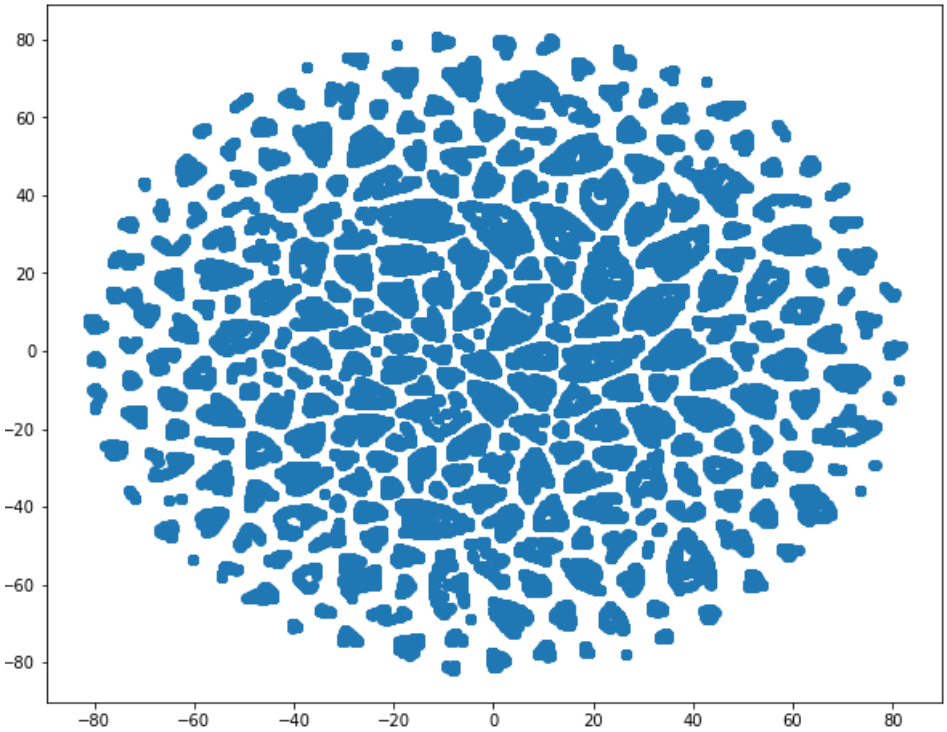


Figure 4.11: t-SNE for raw EUR/USD Forex Dataset.

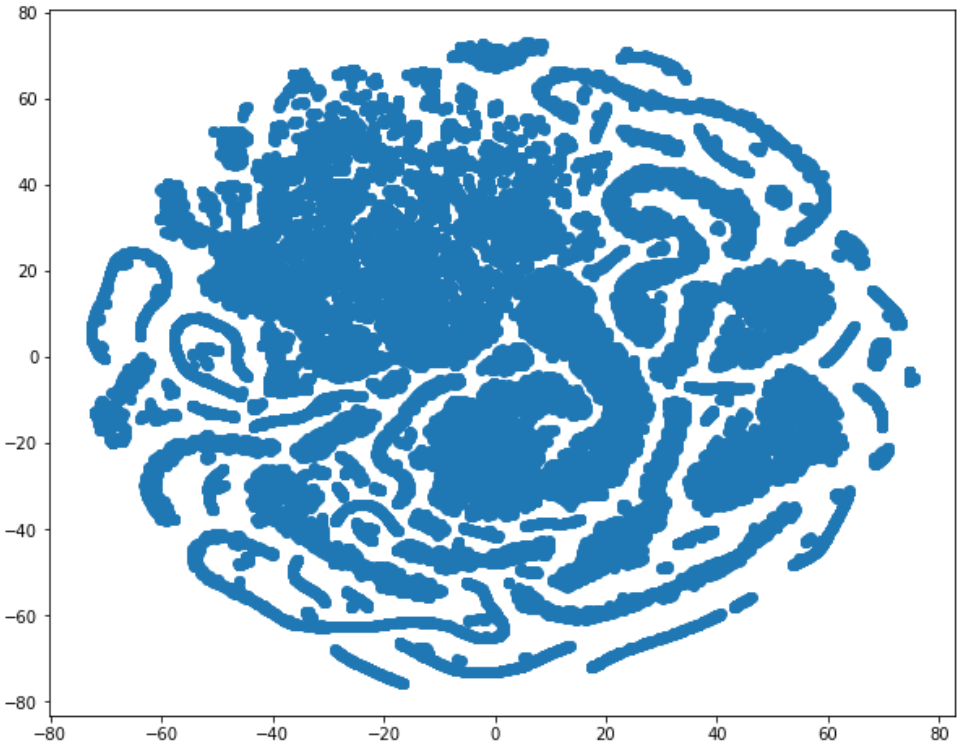


Figure 4.12:t-SNE for EUR/USD Forex Dataset with the intervals between adjacent ticks.

Initial analysis of a random 50,000 rows of raw data of a EUR/USD Forex dataset showed a specific structure. However, this structure is determined by successively increasing Unix-time, not patterns that indicate that the market is ready for significant movements (Figure 4.11).

If the absolute values of Unix-time are converted to the values of the intervals between adjacent ticks, then the structure in the data is lost at first sight (Figure 4.12). However, the presence of a less pronounced data structure in the form of patterns of market readiness for significant movements with a certain probability will be shown below.

Number	Tick characteristic	Comment
1	A real move up	Ask and Bid of the next tick went up
2	The margin expanded up	Ask of the next tick went up, and Bid remained the same
3	The margin expanded down	Bid of the next tick went down, and Ask remained the same
4	A movement down	Ask and Bid of the next tick went down
5	The margin has diverged	Ask has increased, and Bid has decreased
6	The next tick did not exceed the margin	The vast majority of ticks

Table 4.4: Patterns of important price changes.

It is essential to clarify that the t-SNE projections shown above are trying to find common properties for individual tick lines. However, of greater interest is the forecasting of price changes at the next tick depending on the volume of transactions and the time interval at the previous tick. This goal can be achieved by merely shifting price changes one tick back. Thus, in each row, the intervals and volumes of the previous tick and the price changes of the next tick will be indicated. Also of interest is not a simple change in prices, but a significant change. For example, if the change in prices is more than the margin at the previous tick that may indicate the beginning of a substantial price movement.

For this experiment, the following patterns of important price changes were identified (Table 4.4).

Figure 4.13 demonstrates the code and distribution of the above patterns of price movements for the first half of 2019, consisting of 16,680,595 ticks.

```
Y.value_counts()
```

```
6    14315067
5      782772
2      492248
3      488110
4      303347
1      299051
dtype: int64
```

```
Y.value_counts(sort=False).plot.pie()
plt.show()
```

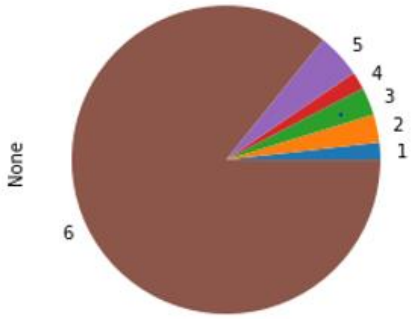


Figure 4.13: The distribution of the patterns of price movements for the first half of 2019.

In a randomly selected dataset of 10,000 ticks from the mentioned above EUR/USD Forex dataset of 2019, the data structure is already visible (Figure 4.14). Of most significant interest are real up and real down movements, that is, numbers one and four (black and bright green). Obviously, several areas of the upward movement patterns are divided into different places, since different patterns are possible to lead to upward movements. The same situation is observed with patterns of downward movements. It is important to note that there is a bunching of patterns of significant price movements.

Real up and real down movements should be analysed separately since the patterns and behaviour of markets during up and down movements can be completely different; for example, it is widely known that markets fall much faster than they rise.

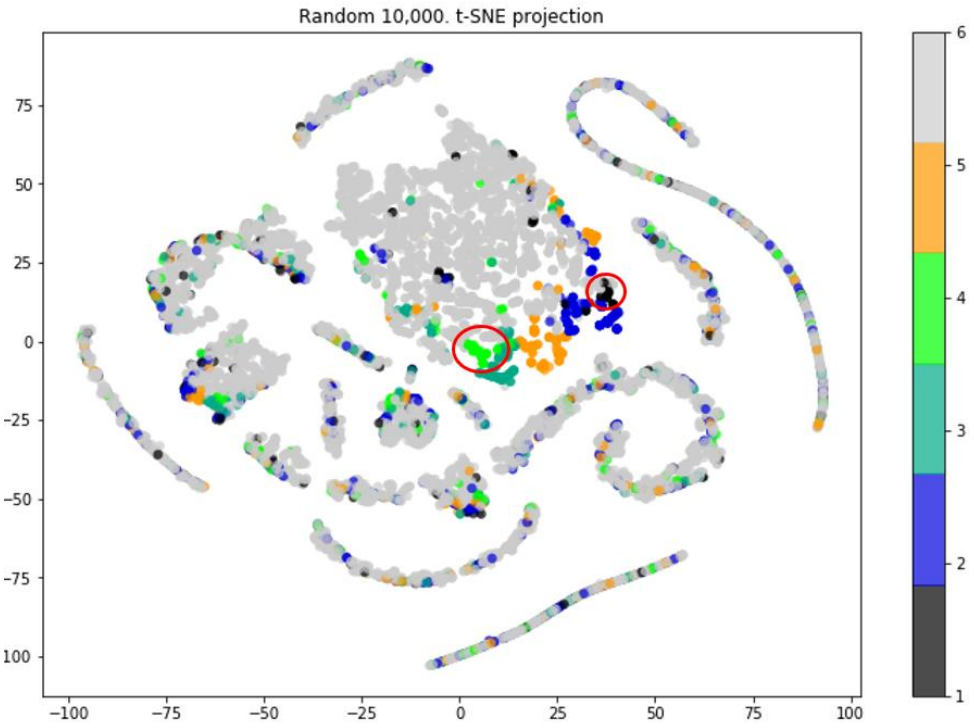


Figure 4.14: t-SNE projection for a random 10,000 rows with significant patterns of price movements.

A randomly selected dataset of 100,000 ticks confirms the presence of a structure in the data (Figure 4.15).

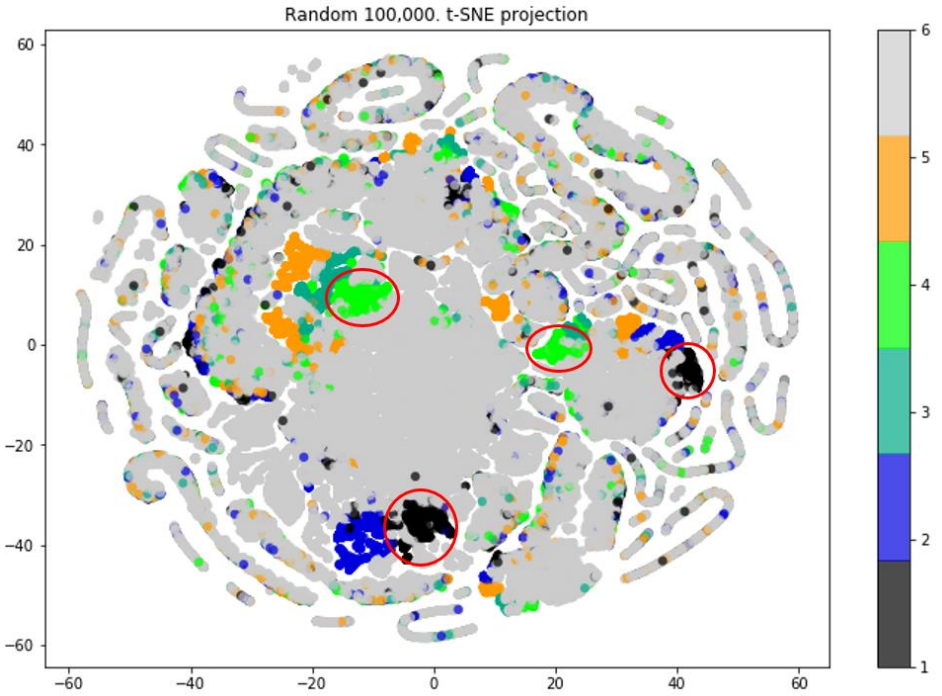


Figure 4.15: t-SNE projection for a random 100,000 rows with significant patterns of price movements.

A random selection of data for creating t-SNE projections was made using the random() function of the Python language. In this way, the first point was chosen at random for counting 10,000 or 100,000 subsequent ticks. The experiments were repeated 15 times for each quantity (10,000 or 100,000 ticks). The visual results of concentration of significant movements on the t-SNE projections have always been similar.

Not all real up and real down movements stick together. Some are located separately. These present potential problems in predicting movements, as well as reducing the accuracy of more complex neural network models. On the other hand, the t-SNE method analyses and searches for patterns in a specially prepared dataset based on information from only two neighbouring ticks. Neural networks will analyse sequences of hundreds of ticks, which are likely to contain more patterns.

4.8. Data Preparation for Experiments with CNNs

For crucial experiments with CNNs to predict the price of the next tick, as well as for experiments with probabilistic movements, this research uses the publicly available EUR/ESD Forex tick-term database of market prices and volume from Dukascopy Bank (Switzerland), which was described in detail in Section 4.5.2.

A period of 36 months, from January 2018 to December 2020 inclusive, was used in most of the experiments. This dataset consists of around 88 million rows Table 4.5. The data structure consisting of prices, volumes and tick times was visualised on the charts in Section 4.5.2.

2018	26,048,774
2019	29,186,310
2020	32,758,627
Total:	**87,993,711**

Table 4.5: Size of EUR/USD Forex Raw Tick Financial Data 2018...2020.

4.8.1. Time-Formats Conversions

These raw input databases contain the time in the format date: (hour: minute: second. millisecond) (Table 4.6), which are stored as the type ‘object’. Data of type ‘object’ is slowly processed by Python and Python-based analysis and data manipulation libraries such as Pandas and NumPy. For convenience of operation, accelerating calculation and training neural networks, all dates and times have been converted to "datatime64" type (Table 4.7).

```
# Import FOREX EURUSD: from 01.01.2018 to 31.12.2018
data = pd.read_csv('/Users/Documents/Python 3/Tick/EURUSD_ticks_2018.csv')
# print raw data
print(data.head(n=10)) # print 10 first rows
print(data[-5:]) # print 5 last rows
```

```
                Time (UTC)      Ask      Bid  AskVolume  BidVolume
0  2018.01.01 22:00:08.661  1.20143  1.20102       0.75       1.50
1  2018.01.01 22:00:08.895  1.20148  1.20102       1.50       1.50
2  2018.01.01 22:00:10.634  1.20147  1.20102       0.75       1.50
3  2018.01.01 22:00:11.223  1.20148  1.20102       1.50       1.50
4  2018.01.01 22:00:29.530  1.20145  1.20102       0.75       1.50
5  2018.01.01 22:00:52.173  1.20145  1.20102       0.75       1.50
6  2018.01.01 22:00:52.224  1.20148  1.20102       1.50       1.50
7  2018.01.01 22:00:56.269  1.20124  1.20097       1.50       0.75
8  2018.01.01 22:00:56.836  1.20115  1.20100       1.50       1.50
9  2018.01.01 22:01:10.848  1.20115  1.20099       1.50       0.75
                       Time (UTC)      Ask      Bid  AskVolume  BidVolume
26048770  2018.12.31 22:00:00.325  1.14697  1.14639       0.75       0.75
26048771  2018.12.31 22:00:00.378  1.14697  1.14641       1.50       1.50
26048772  2018.12.31 22:00:02.145  1.14698  1.14642       0.75       0.75
26048773  2018.12.31 22:00:02.681  1.14703  1.14627       0.75       0.75
26048774  2018.12.31 22:00:03.048  1.14711  1.14627       0.75       0.75
```

Table 4.6: Raw data of EUR/USD Forex 2018.

```
data['DateT'] = pd.to_datetime(data['Time (UTC)'])
print(data.dtypes)
print(data.head(n=5))
```

```
Time (UTC)            object
Ask                  float32
Bid                  float32
AskVolume            float32
BidVolume            float32
DateT         datetime64[ns]
dtype: object
                Time (UTC)      Ask      Bid  AskVolume  BidVolume  \
0  2019.01.01 22:02:37.254  1.14682  1.14598       0.75       3.75
1  2019.01.01 22:02:38.590  1.14682  1.14599       0.75       2.25
2  2019.01.01 22:02:39.138  1.14684  1.14599       0.75       3.75
3  2019.01.01 22:02:55.787  1.14684  1.14598       0.75       3.75
4  2019.01.01 22:03:02.060  1.14684  1.14598       0.75       3.75

                   DateT
0 2019-01-01 22:02:37.254
1 2019-01-01 22:02:38.590
2 2019-01-01 22:02:39.138
3 2019-01-01 22:02:55.787
4 2019-01-01 22:03:02.060
```

Table 4.7: Converting the dates and times to ‘datatime64’ type.

As already mentioned in the Chapter's introduction, for intraday trading, the size of the intervals between neighbouring ticks in milliseconds is essential. Thus, all settlement times are converted from Unix-time into the array of values of inter-tick intervals, which will demonstrate the intensity of the trading market at each moment during the intraday trading session. Thus, in order to train CNNs, it is necessary to convert the time array starting on January 1, 1970, into an array of intervals between ticks. The idea of the above conversion is essential and innovative.

In addition to indicating the intensity of the market, inter-tick intervals allow the normalising of the dataset with higher quality. It is possible because the range of the majority of these intervals between neighbouring ticks is relatively narrow. Therefore, it is possible to choose some maximum number and normalise these intervals so that they are approximately evenly distributed in the range [0;1]. Such a conditionally uniform distribution increases the quality of the training of neural networks.

4.8.2. Data Cleaning

After completing the first experiments, significant instability and volatility of the predicted prices relative to the actual prices were found. The predicted prices looked like an oscillatory chart with large amplitudes. After preliminary analysis, it was found that such oscillations were due to errors in the raw data.

There are often moments of exceptionally high market intensity, which usually occur at the moments of the opening of trading days or significant or unexpected news for the market from politicians or government market regulators. The increased intensity of trading at the moments of the market opening is associated with client orders accumulated overnight, which brokerage companies are obliged to execute as soon as possible. Thus, there are regularly recurring periods in which automated data collection systems have to process market data quickly. Sometimes these systems do not cope with the load, and as a result, errors, gaps, or zero values appear in the databases. Figure 4.16 shows incorrect predicted prices based on missing prices with a large amplitude (red and orange graphics), as opposed to actual prices (blue and green).

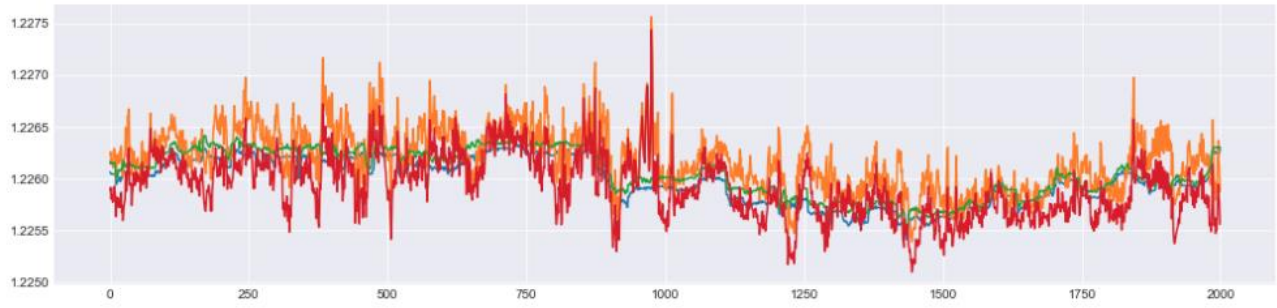


Figure 4.16: Falsely predicted prices based on missing data with a large amplitude (red and orange graphics), as opposed to real prices (blue and green).

Thus, before processing, transformation, and analysis, the data should be checked for the absence of obvious problems, including very large or zero values and gaps in the data. For example, if price data is not available at some point, this is perceived as a zero price. The next price can be accurate. The zero prices and gaps clearing takes significant processing time but is achieved with simple code. Clearing erroneous data is a more difficult task because it is not always evident what out of this data is wrong. Erroneous prices are more easy to identify than erroneous trading volumes. If the price unexpectedly jumps significantly during one or several ticks and then unexpectedly returns to the previous level, it can be identified as an erroneous price with a high degree of probability. If the rates of currency pairs in the Forex market jump above or below the exchange rate, which they have never been for these currency pairs, then the probability of the erroneous rate is 100%. Erroneous data could also be identified by the incorrectly collected transaction time, the value of which is not within the interval between two neighbouring times.

In these cases, it is necessary to do preliminary data cleaning. Otherwise, ANNs during training perceive these unexpected changes as significant price volatility, and it is remembered as a false pattern. As a result, this leads to a decrease in the accuracy of the predictive models and is demonstrated by a large amplitude of the predicted prices.

Table 4.8 demonstrates an example erroneous data identification from incorrectly collected transaction times. These erroneous times become obvious after calculating the intervals between neighbouring ticks. If some intervals are equal to large negative values, then all lines with such data should be deleted without further analysis of prices and volumes in these lines.

```
In [14]: #  DateDeltal <= 0
         data[ data.DateDeltal <= 0]
```

	Date	Bid	Ask	Vol_Bid	Vol_Ask	DateT	Bid_N	Ask_N	Bid_N_use	Ask_N_use	Vol_Bid_N	Vol_Ask_N	DateDelta	DateDelta1
0	2018.01.01 22:00:08.661	1.20102	1.20143	1.5	0.75	2018-01-01 22:00:08.661000000	0.578714	0.581643	0.157429	0.163286	0.15	0.075	1514844008661	0
3933	1	1.00000	1.00000	1.0	1.00	1970-01-01 00:00:00.000000001	1.000000	1.000000	1.000000	1.000000	1.00	1.000	0	-1514850430212
3937	1	1.00000	1.00000	1.0	1.00	1970-01-01 00:00:00.000000001	1.000000	1.000000	1.000000	1.000000	1.00	1.000	0	-1514850437617
4287	1	1.00000	1.00000	1.0	1.00	1970-01-01 00:00:00.000000001	1.000000	1.000000	1.000000	1.000000	1.00	1.000	0	-1514851111326
4288	1	1.00000	1.00000	1.0	1.00	1970-01-01 00:00:00.000000001	1.000000	1.000000	1.000000	1.000000	1.00	1.000	0	0
4476	1	1.00000	1.00000	1.0	1.00	1970-01-01 00:00:00.000000001	1.000000	1.000000	1.000000	1.000000	1.00	1.000	0	-1514851547542
4510	1	1.00000	1.00000	1.0	1.00	1970-01-01 00:00:00.000000001	1.000000	1.000000	1.000000	1.000000	1.00	1.000	0	-1514851635697

Table 4.8: Example of the identification of incorrectly collected transaction times.

4.8.3. Normalisation

For increasing the accuracy and other error metrics of the results, it is desirable to train neural networks on data that has been normalised. Data normalisation is necessary for several reasons. The original data values could vary over a vast range, and the training of analytical models, such as neural networks, with this data, may be incorrect. Without normalisation, the most significant impact will be the inputs with a more variable range, even if this is contrary to the actual situation. Furthermore, an imbalance between input feature values could cause instability of the neural network model, worsen learning outcomes and slow down the training and validation processes. In particular, parametric machine learning methods, which include ANNs, are usually trained using the Batch Gradient Descent method, that is, in several directions at once. Therefore, to increase the stability and training speed, all input variables should be of the same order or, even better, they should be in a particular range, for example, within the intervals [0;1] or from [-1;1].

However, for complex heterogeneous, unstable and noisy financial data, the above classical approaches to normalisation are insufficient. This research uses three types of data normalisation, each with its own nuances. These normalisations of tick prices, tick volumes and intervals between ticks are described in detail below. The method of discarding long, thin tails of the statistical distribution and outliers is also applied. As a result, in this research, the matrix of calculated coefficients (weights), the output response vector and the matrix of input parameters take values in the interval of 0 to 1 inclusive [0;1].

- **Normalisation of Intervals Between Ticks**

The time intervals between neighbouring ticks are enormously diverse. This is the reason why standard classical normalisation methods alone are not enough. Figure 4.17 shows the statistical distribution of the time intervals between neighbouring ticks for a random 1,000 ticks in 2018.

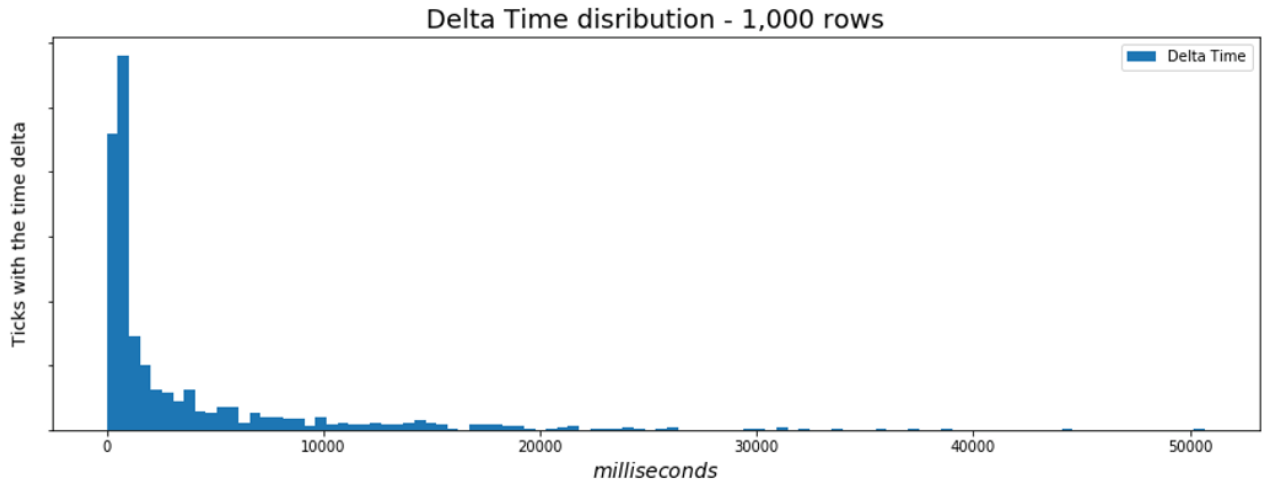


Figure 4.17: Distribution of 1,000 random intervals between ticks.

The graph shows that the vast majority of the time, intervals are no more than 20 seconds. Precisely the same conclusion is gained from the graphs of random 10,000 ticks (Figure 4.18) and 100,000 ticks (Figure 4.19).

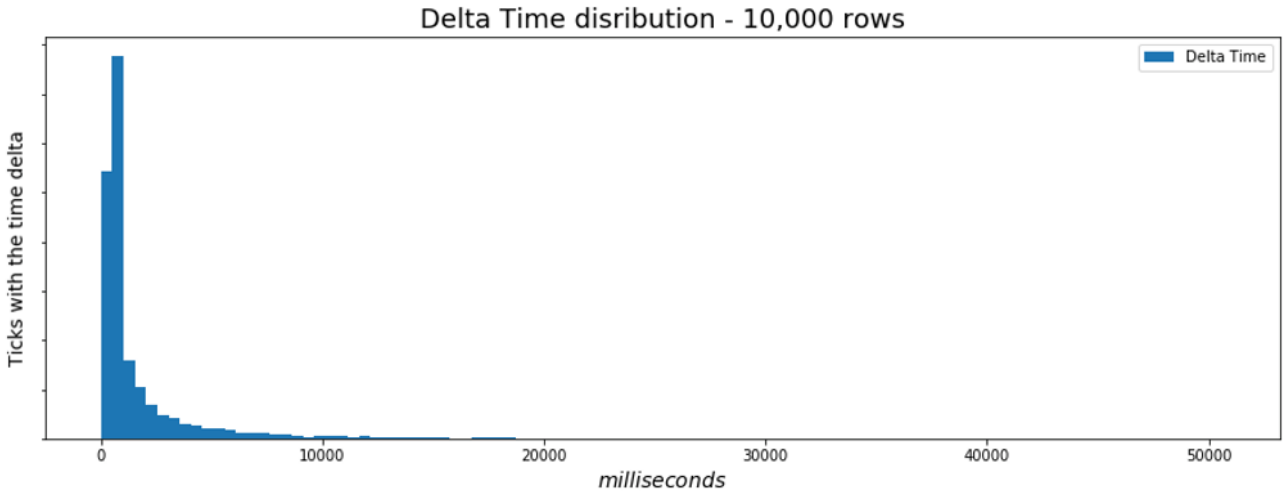


Figure 4.18: Distribution of 10,000 random intervals between ticks.

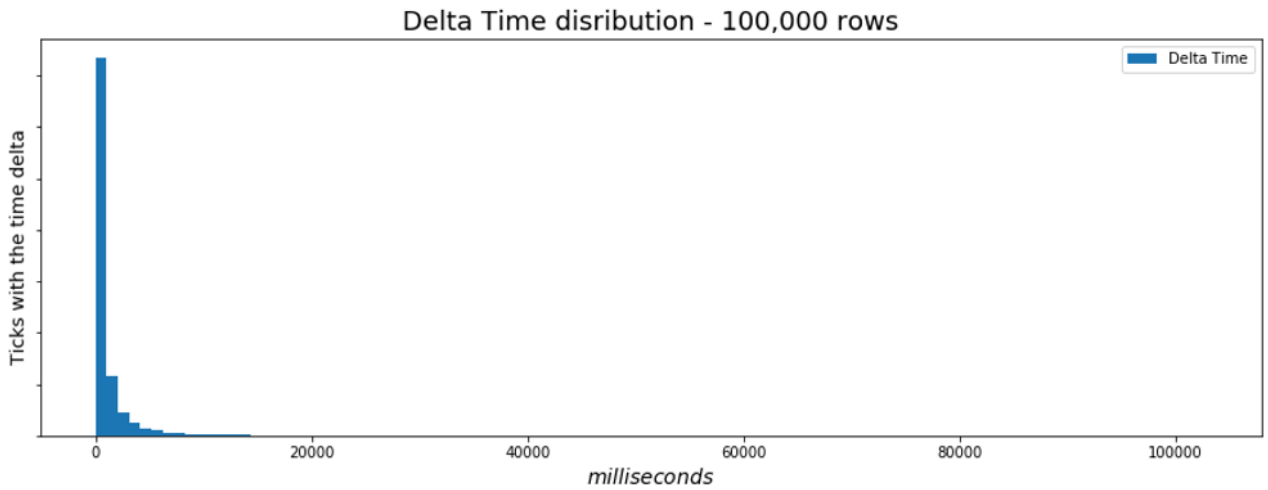


Figure 4.19: Distribution of 100,000 random intervals between ticks.

Analysis shows that ticks with time intervals of more than 20 seconds account for just 0.086% of the total.

With such an uneven distribution, the application of standard normalisation by Equation (4.4):

$$\hat{x}_i = \frac{x_i - x_{min}}{x_{max} - x_{min}} \tag{4.4}$$

Where x_i is the value of the discrete input variable to be normalised (one of the intervals between ticks);

$\hat{x}_i$ is the value of the normalised discrete input variable;

x_{max} is the maximum value of the input data feature;

x_{min} Is the minimum value of the input data feature.

The normalised input matrix $\hat{X}$ would be unrepresentative with x_{max} greater than 100,000 because all other intervals between ticks are normalised to values close to zero, and this will dramatically reduce the accuracy of future trained neural network.

Classical non-linear normalised methods, such as sigmoid or hyperbolic tangent, are often used to normalise such asymmetric distributions. The first one converts the values to the

range [0;1], and the second one to [-1;1]. In both these non-linear normalisation, different intervals of the original range are 'compressed' to varying degrees, which in some practical cases allows one to achieve an acceptable result by choosing the parameters that control the shape of the normalising function. However, in actual big financial data, outliers in the form of large values of inter-tick intervals are not associated with traders' activity but reflect the stock exchanges' organisational rules. For example, Table 4.9 demonstrates that most inter-tick intervals over 100,000 seconds are either weekends, the ends of trading days, or technical stops of trading. Thus, from the point of view of the quality of recognition of patterns of behaviour of market participants, in other words, the predictive value of neural networks, these large intervals are not necessary. Moreover, these organisational intervals reduce the quality of the training model, as well as gaps in the data (which is discussed in Section 4.8.2), creating oscillator plots of the forecast prices.

Thus, it is necessary to reduce the influence of these long inter-tick intervals. However, a simple deletion of these rows, as in the case of data gaps (Section 4.8.2), would be incorrect because many of these rows involved with the daily opening and closing of the markets and contain the correct prices and volumes. This research does not use any extrapolations or interpolations that could distort the raw tick data and lose the patterns of behaviour of financial market participants. The most obvious solution used in this research is all intervals of more than 20 seconds should be equal to 20 seconds (20,000 milliseconds). This original method allows:

- exclusion of the oscillatory nature of the predicted prices,

- avoidance of concentrating most of the normalised inputs around zero,

- normalisation of the inter-tick intervals without using the sigmoid and hyperbolic tangent, which would require additional processor time,

- avoidance of the situation when the neural networks would concentrate on searching for patterns for trading interruptions rather than patterns of behaviour of market participants.

```
data[data.DateDelta1 > 100000]
```

	Ask	Bid	AskVolume	BidVolume	DateT	Bid_N	Ask_N	Vol_Bid_N	Vol_Ask_N	DateDelta	DateDelta1
93764	1.20467	1.20463	2.25	4.69	2018-01-03 03:44:03.184	0.619925	0.620224	0.469	0.225	1514951043184	102833
322405	1.20297	1.20265	0.75	0.75	2018-01-07 22:00:04.850	0.605149	0.607538	0.075	0.075	1515362404849	172808585
781883	1.21962	1.21933	0.75	0.75	2018-01-14 22:00:04.641	0.729627	0.731791	0.075	0.075	1515967204641	172812314
1318286	1.22708	1.22702	0.37	0.37	2018-01-21 22:00:14.074	0.787015	0.787463	0.037	0.037	1516572014073	172817937
1932893	1.24291	1.24232	0.75	0.75	2018-01-28 22:00:34.413	0.901194	0.905597	0.075	0.075	1517176834413	172839215
2560927	1.24391	1.24320	0.75	0.75	2018-02-04 22:00:04.598	0.907761	0.913059	0.075	0.075	1517781604598	172804645
3262667	1.22501	1.22483	0.75	0.37	2018-02-11 22:00:14.329	0.770672	0.772015	0.037	0.075	1518386414329	172818210
3818770	1.24035	1.24015	0.37	0.37	2018-02-18 22:00:14.462	0.885000	0.886493	0.037	0.037	1518991214462	172820488
4316953	1.22953	1.22928	0.75	0.75	2018-02-25 22:00:04.559	0.803881	0.805746	0.075	0.075	1519596004559	172808442
4840754	1.23296	1.23248	0.37	0.37	2018-03-04 22:00:08.704	0.827762	0.831343	0.037	0.037	1520200808704	172812099
5371812	1.23176	1.23094	0.37	0.37	2018-03-11 21:00:07.329	0.816269	0.822388	0.037	0.037	1520802007329	169215702
5804131	1.22857	1.22817	1.12	1.12	2018-03-18 21:02:03.226	0.795597	0.798582	0.112	0.112	1521406923225	172931549
6298964	1.23551	1.23470	0.37	0.37	2018-03-25 21:00:10.831	0.844328	0.850373	0.037	0.037	1522011610831	172814900
6738174	1.23267	1.23199	0.75	0.75	2018-04-01 21:00:18.456	0.824104	0.829179	0.075	0.075	1522616418456	172822683
6738589	1.23269	1.23229	1.50	1.00	2018-04-01 21:55:26.956	0.826343	0.829328	0.100	0.150	1522619726956	106094
6902514	1.22825	1.22822	1.00	2.36	2018-04-04 04:43:06.917	0.795970	0.796194	0.236	0.100	1522816986917	117996
7169706	1.22896	1.22824	0.75	0.75	2018-04-08 21:00:05.291	0.796120	0.801493	0.075	0.075	1523221205291	172809138
7265381	1.23241	1.23235	2.89	3.94	2018-04-09 22:38:45.987	0.826791	0.827238	0.394	0.289	1523313525987	116847
7636948	1.23360	1.23278	0.75	0.75	2018-04-15 21:00:12.838	0.830000	0.836120	0.075	0.075	1523826012838	172816678
8076797	1.22772	1.22699	0.52	0.75	2018-04-22 21:00:13.972	0.786791	0.792239	0.075	0.052	1524430813972	172817806
8273969	1.22264	1.22262	1.00	1.00	2018-04-25 03:17:46.392	0.754179	0.754329	0.100	0.100	1524626266392	100509
8541118	1.21250	1.21224	0.75	0.75	2018-04-29 21:00:04.454	0.676716	0.678657	0.075	0.075	1525035604454	172808681
8942695	1.19880	1.19873	2.59	0.15	2018-05-03 21:55:06.070	0.575895	0.576418	0.015	0.259	1525384506070	105762
9031200	1.19637	1.19557	0.75	0.75	2018-05-06 21:01:02.100	0.552313	0.558284	0.075	0.075	1525640462100	172865801
9116131	1.19249	1.19243	2.44	1.69	2018-05-07 22:37:58.115	0.528881	0.529328	0.169	0.244	1525732678115	124396
9503143	1.19463	1.19385	0.75	0.52	2018-05-13 21:00:17.400	0.539478	0.545299	0.052	0.075	1526245217400	172820973
10002469	1.17766	1.17684	0.75	0.75	2018-05-20 21:01:33.818	0.412537	0.418657	0.075	0.075	1526850093817	172897570
10002752	1.17733	1.17708	1.24	0.20	2018-05-20 21:21:06.478	0.414329	0.416194	0.020	0.124	1526851266478	110737
10531591	1.16878	1.16862	0.75	0.75	2018-05-27 21:00:10.180	0.351194	0.352388	0.075	0.075	1527454810180	172813891
11105652	1.16644	1.16561	0.75	0.75	2018-06-03 21:00:51.903	0.328731	0.334925	0.075	0.075	1528059651903	172859215
...	...	...	...	...	...	...	...	...	...	...	...
25386601	1.14138	1.14065	1.00	0.19	2018-12-25 01:34:24.990	0.142463	0.147910	0.019	0.100	1545701664990	170843
25387175	1.14171	1.14128	1.75	1.25	2018-12-25 02:42:03.178	0.147165	0.150373	0.125	0.175	1545705723177	107992
25387403	1.14139	1.14126	1.37	1.25	2018-12-25 03:11:15.178	0.147015	0.147985	0.125	0.137	1545707475177	108232
25391245	1.14183	1.14137	0.75	0.75	2018-12-25 06:54:33.042	0.147836	0.151268	0.075	0.075	1545720873041	106147
25391403	1.14185	1.14131	1.00	0.19	2018-12-25 07:16:59.689	0.147388	0.151418	0.019	0.100	1545722219689	107592
25391948	1.13972	1.13901	0.75	0.37	2018-12-25 22:01:18.411	0.130224	0.135522	0.037	0.075	1545775278411	50549336
25916866	1.14398	1.14331	1.50	1.50	2018-12-30 22:00:22.563	0.162313	0.167314	0.150	0.150	1546207222563	172826438
25917148	1.14398	1.14365	1.50	1.00	2018-12-30 22:27:45.800	0.164851	0.167314	0.100	0.150	1546208865800	125553

83 rows × 11 columns

Table 4.9: Input data where the inter-tick intervals are more than 100,000.

- **Normalisation of Volume**

A similar method of normalisation is applicable to the volumes, which also display substantial irregularity. Figure 4.20, Figure 4.21 and Figure 4.22 show the statistical distribution of the volumes of random 1,000 ticks, 10,000 ticks, and 100,000 ticks. These distributions demonstrate that the vast majority of volumes do not exceed ten million lots.

Using the Python random number function, any row from the main financial database was selected, which became the starting point for selecting the next 1,000, 10,000, or 100,000 rows. The experiment was repeated 15 times for each specified different number of rows. The distributions of volumes were similar each time.

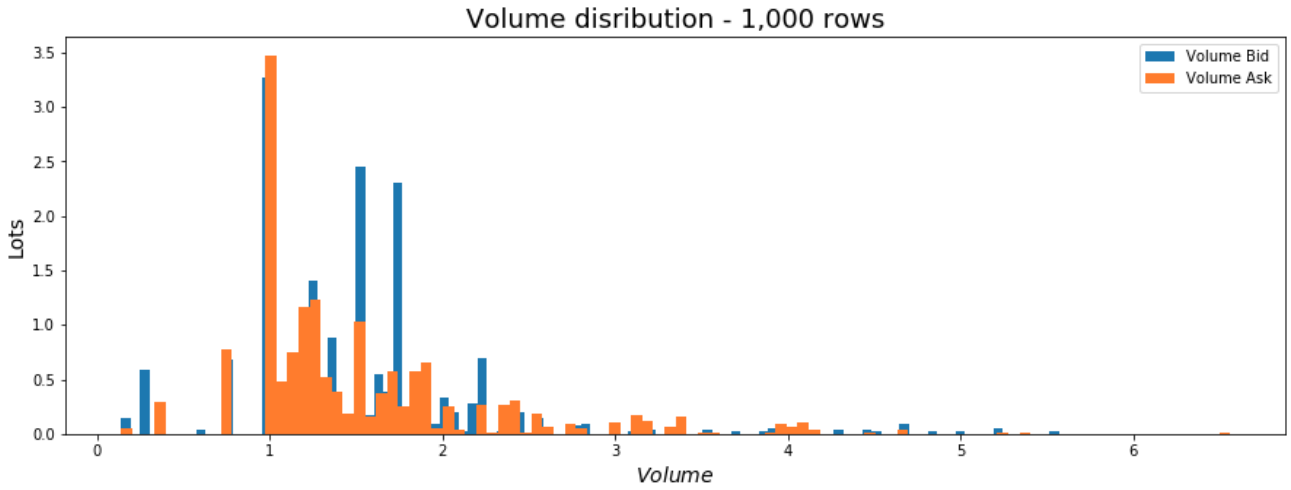


Figure 4.20: Distribution of random 1,000 ticks' volume.

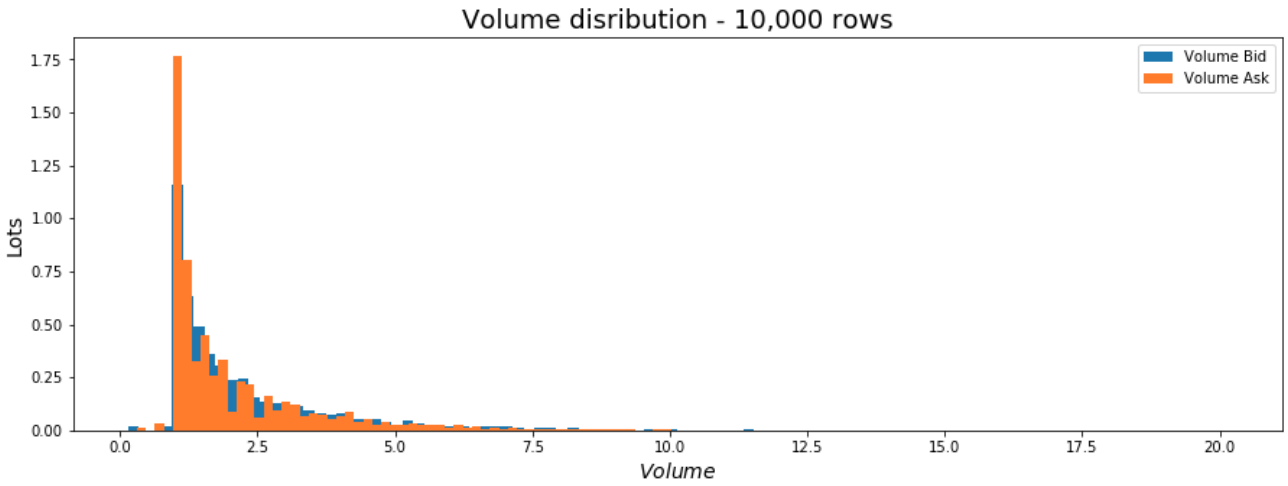


Figure 4.21: Distribution of random 10,000 ticks' volume.

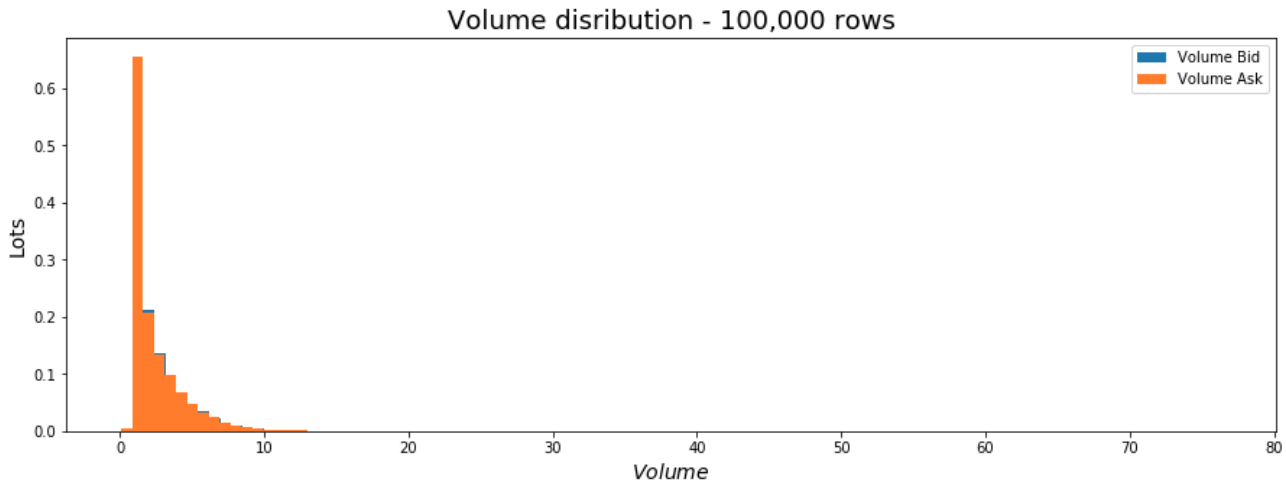


Figure 4.22: Distribution of random 100,000 ticks' volume.

Thus, as in the previous example, all volume lots over 10 million equate to 10 million lots. The loss of information on long time intervals between ticks is less sensitive than on extra-large volume lots. However, in transactions with extra-large lots, there are often substantial price changes, and there are also possible delays with the execution of market orders, which usually leads to slippage. These are not the best times to enter the market in terms of risk management.

In further research, the author is aiming to add additional significant volume analysis. At least three ways are possible:

- create a separate parallel path to the current neural network for analysing these significant volumes,

- use machine learning methods faster than neural networks, such as decision trees, etc,

- add two additional features (columns) to the input data for training the current neural network model, containing normalised data of only these significant ask and bid volumes.

- **Normalisation of Prices**

A common mistake is to normalise data before splitting it into training and test data because extrema, i.e. maximum and minimum data, are used for normalisation. When making forecasts in real stock trading, it is not known what the maximum and minimum prices will be. Taking information from the future for training, such as extrema, is incorrect because this will create system errors.

Thus, there are two big problems with the normalisation of tick prices. Firstly, when training and validating a neural network, information on the maximum and minimum prices for normalisation cannot be used since this will tell the neural network the price boundaries above or below which there will be no movement. This creates false patterns in the neural network that do not correlate with correct patterns of behaviour of market participants. Secondly, the preparation of financial data for further feeding of the neural network will be performed by the sliding window method consisting of only 200 ticks (described in detail in Section 5.1). Thus, the normalisation of tick prices within each step of the sliding window using the global extrema of the whole database, that are minimum and maximum tick prices, will be unrepresentative. This is because the price changes within each step of the sliding window are small compared to the global price difference over three years.

```
plt.figure(num=1, figsize=(15,7)) # size of the chart
plt.hist(data["Ask"], bins=100, normed=True) # histogram with 100 column

plt.title('Ask disribution', size=18) # title and font size
plt.ylabel('Numbers', size=14)
plt.xlabel('$Ask$', size=14)
# saving in PNG
plt.savefig('/Users/v135790/Documents/Python 3/December 2018/Figures/AskDistribution2.png')
```

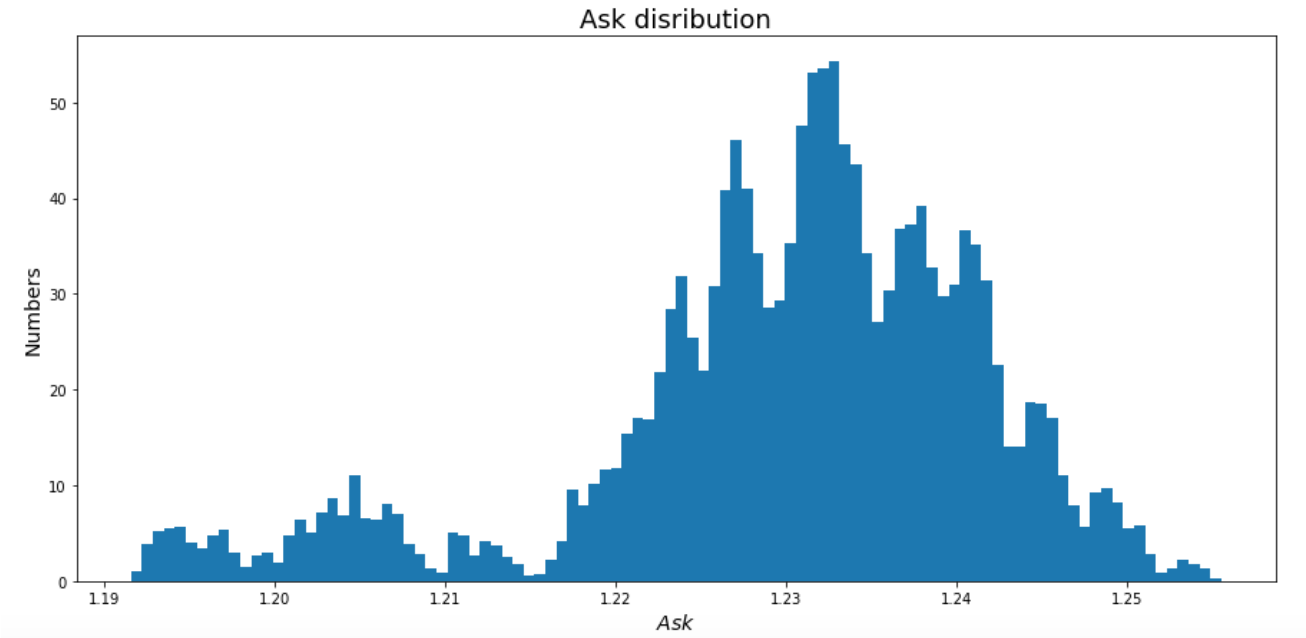


Figure 4.23: Ask distribution.

For example, Figure 4.23 shows the statistical distribution of the ask price over 18 months. In other words, half of the whole main database was used to create this distribution. The bid price distribution is similar. On one hand, the global price difference for 18 months varies slightly: from 1.11 to 1.26. On the other hand, tick price changes during any 200 ticks of the sliding window are insignificant relative to 18 months. For example, if prices normalise based on the minimum and maximum prices for only four months, the normalised price chart will

stretch to the line (Figure 4.24). Consequently, the accuracy of training a neural network will be low.

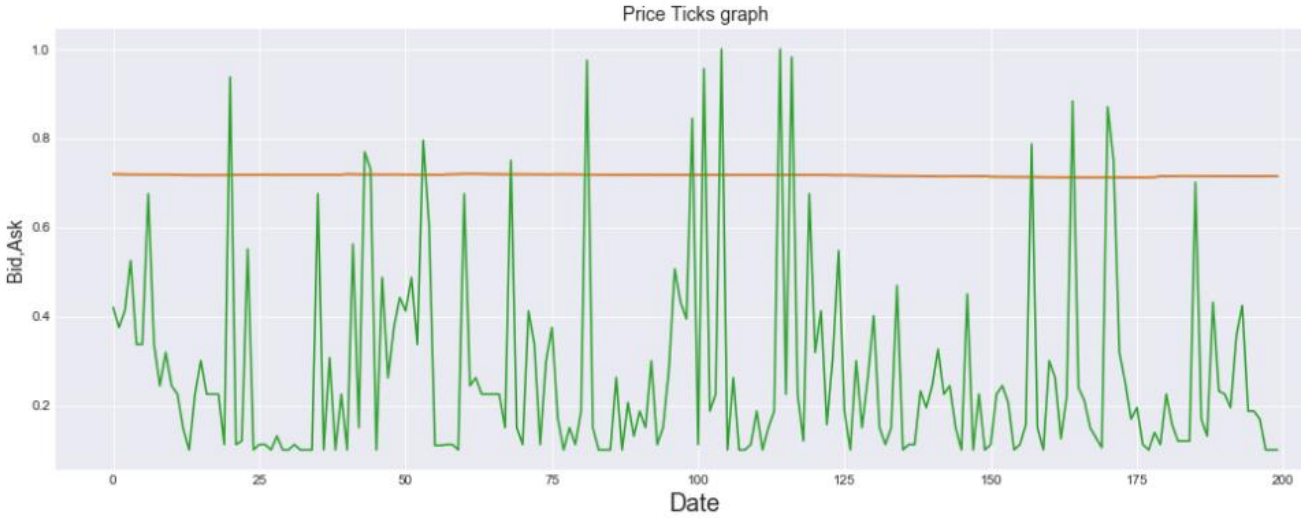


Figure 4.24: An inconsistency of the dimensionality of normalisations. The green line is normalised volume. The red line is normalised price.

Intermediate conclusion: it is necessary to normalise prices not in the range of minimum and maximum prices for the whole period: three years, but for a shorter period. There are two options for normalising the 200 tick width of the sliding window:

The first is the normalisation at the minimum and maximum price of each 200-tick interval. This option requires increased processing power since it is necessary to find the minimums and maximums of each 200-tick interval. Additionally, this option slightly distorts the search for patterns of market participants' behaviour since each 200-tick interval will have a different normalisation scale. Equation (4.5) describes this type of normalisation:

$$\hat{p}_i = \frac{p_i - p_{200_min}}{p_{200_max} - p_{200_min}} \quad \textbf{(4.5)}$$

Where p_i is the current price of i-tick;

$\hat{p}_i$ is the normalised price of i -tick;

p_{200_max} is the maximum price of current 200-tick interval;

p_{200_min} Is the minimum price of current 200-tick interval.

The second is the normalisation of prices based on maximum volatility within all 200-tick intervals. In other words, there is such a 200-tick interval during all three years, where the price volatility is maximum (Equation (4.6)). This price difference is taken to normalise every 200 tick interval throughout the incoming big financial database.

$$\hat{p}_i = \frac{p_i - p_{min}}{(p_{max} - p_{min})max} \quad \textbf{(4.6)}$$

Where p_i is the current price of i-tick;

$\hat{p}_i$ is the normalised price of i -tick;

$(p_{max} - p_{min})max$ is the maximum between prices inside each of 200-tick intervals during three years (called Maximum delta of prices);

p_{max} is the maximum price of the Maximum delta of prices;

p_{min} Is the minimum price of the Maximum delta of prices.

This research uses the second way. The Python code, which is used for cleaning, normalisation and preparation data for the first series of the experiments, is demonstrated in Appendix IX.

4.9. Preliminary Testing of Models and Topologies of NNs

This section aims to demonstrate the results of the preliminary experiments with different topologies of ANNs. The functionality of the topologies chosen should be checked for their correctness before being used with the full set of big financial tick data. The majority of the ANN models are created with Keras, which is a ML library that uses Python and TensorFlow 2.0 as a backend, one of the most potent ML frameworks of today. Keras does not use a mathematical graph representation. Consequently, it is more user-friendly than the original TensorFlow. The following subsections describe different architectures of ANNs. The difficulty of the topologies listed below increases from simple models as multilayer perceptron to deep enough CNNs.

This preliminary analysis of the neural network models intended for use in main experiments allows one to write a preliminary code and test it on integrated financial data or limited sets of

big financial tick data. Thus, the code is pre-tested, and bottlenecks in the code that require much computational time are identified. Therefore, it is possible to improve the performance of this code in advance with simple and limited examples.

4.9.1. Tests with Multilayer Perceptron for Prediction of S&P-500 Index

The convolutional effects of MLP are significantly different from a CNN. The MLP accepts vectors as inputs. Therefore, an image, for example, has to be transformed from a matrix (two-dimensional array) into a vector (one-dimensional array). As a result of this, convolutional effects manifest themselves in only the vertical direction and are lost in the horizontal direction.

Section 4.5.1 explained in detail the S&P-500 index, which is compiled by Standard & Poor's and includes about 500 shares of the largest companies traded on major United States stock exchanges, such as the New York Stock Exchange (NYSE) and NASDAQ. In this test, the prices of all stocks of these 500 companies are analysed using MLP to predict the S & P-500 index price for the next minute.

The MLP used for this test has four hidden layers. The number of elements in each layer depends on the size of the input data. In this case, the MLP can recognise the majority of the properties of the data. For convolutional effects, the number of cells in each subsequent layer must be reduced after the first hidden layer. In this architecture, a rule was used in which each subsequent hidden layer has half the number of cells than the previous one. This rule is often used in practical neural network modelling, including MLP and CNN. The first hidden layer, which consists of about 1000 cells, is double of the number of the shares considered. Following the rule mentioned above, each of the subsequent hidden layers has half the number of the previous layer cells.

This is a regression task. In other words, an answer is a real number or a numeric vector. As an output, the topology has one cell with a real number that is the equivalent of the predicted price of the S&P 500 for the next minute. A similar model is written in TensorFlow (Heinz, 2017). The learning time was about ten seconds for each epoch. An epoch means the training of the ANN on a whole dataset once. The code written in Python of the MLP is shown in Appendix VIII. Figure 4.25 shows two graphs – the blue one is for the actual prices, and the

orange one is for the predicted prices of the S&P-500 index. As can be seen, the difference between the two prices is minimal.

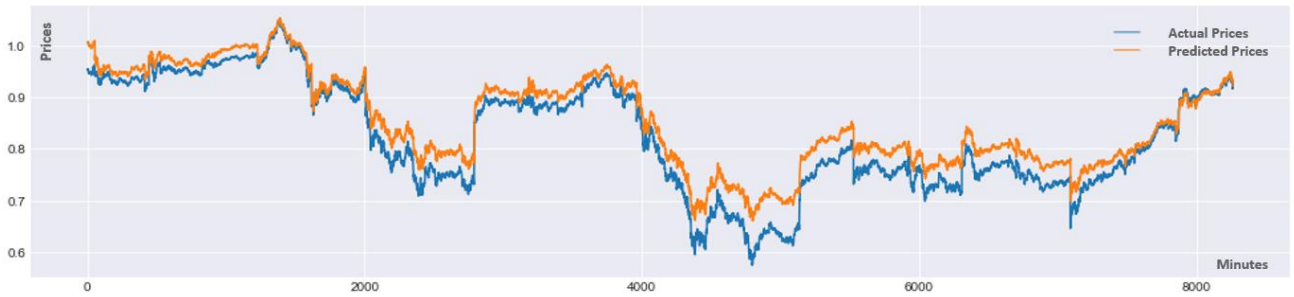


Figure 4.25: Actual and predicted prices of the S&P-500 index (normalised).

As a result of training the topology, the average inaccuracy is 3.1% that is the response rate. However, the inaccuracy graph, which is shown in Figure 4.26, has some significant statistical outlier (up to around 20%) of inaccuracy. Usually, it happened in the time of maximum volatility on the securities market. Thus, the spikes could be explained by price gaps after the weekends. For reducing these statistical outliers in the inaccuracy graph, the remedial action would be to reduce the time frame below one minute and carry out pattern analysis on intraday data only without predicting prices at the beginning of the next day.

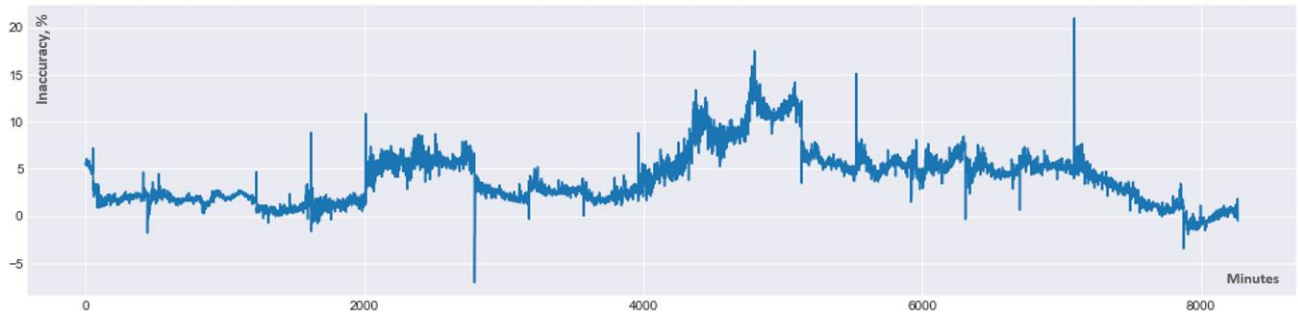


Figure 4.26: Inaccuracy, %.

Figure 4.27 demonstrates absolute errors of the training and the validation. What is interesting in the chart, is the growth of the absolute error validation after nine epochs, at the same time as training absolute error declines. This is the indicator of overfitting. Accordingly, the training of the topology with the dataset should be limited to five epochs because that was when the minimum level of the absolute error was achieved.

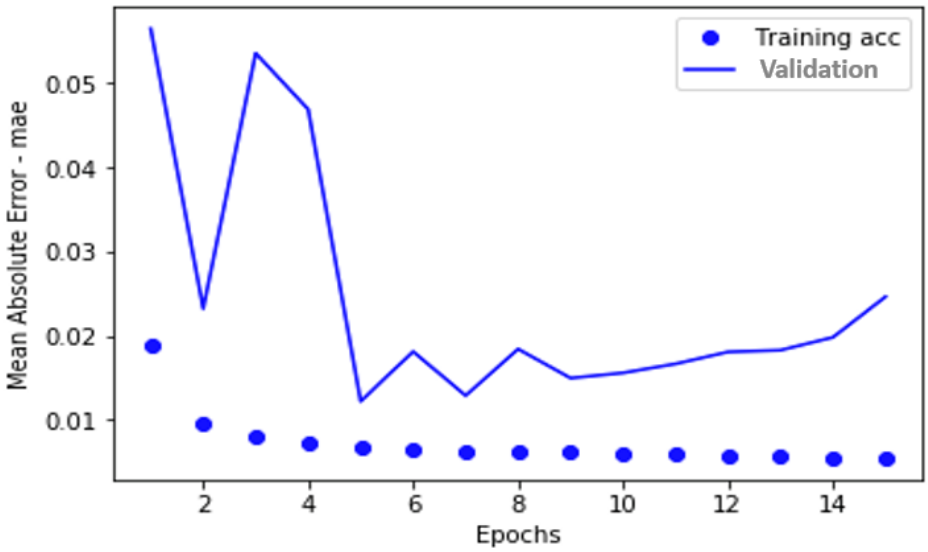


Figure 4.27: Training and validation absolute errors.

4.9.2. Initial Tests of CNN Models with Transfer Learning and Fine-Tuning

A CNN is based on the spatial structure of information. Thus, a CNN is more suitable for analysing data with spatial localities, such as images or sets with time-related data (market price, volume, order book). A CNN has a structure that is similar to a human visual cortex, which was gained from physiological experiments. The visual cortex is composed of multiple layers; each of them is in charge of more complex characteristics of the input image and for a higher level of abstractions. The first layer recognises pixels; the second one aggregates simple geometric forms; the next layers identify patterns, objects, subjects, etc.

In initial experiments, the author analysed 2D price and volume charts using pre-trained CNNs, such as VGG-16, in conjunction with advanced methods such as Transfer Learning and Fine-Tuning, which do not have to train on all layers. However, the time spent on each set of hyperparameters was significant, and the Google Colab service often interrupted training and dumped the big financial tick data. VGG-16 is one of the deep CNN models (Figure 4.28), which has 16 hidden layers and was designed by the Oxford Visual Geometry Group (University of Oxford, 2021). Algorithm 4.1 demonstrates the VGG-16 topology, which contains more than 138 million weights (parameters).

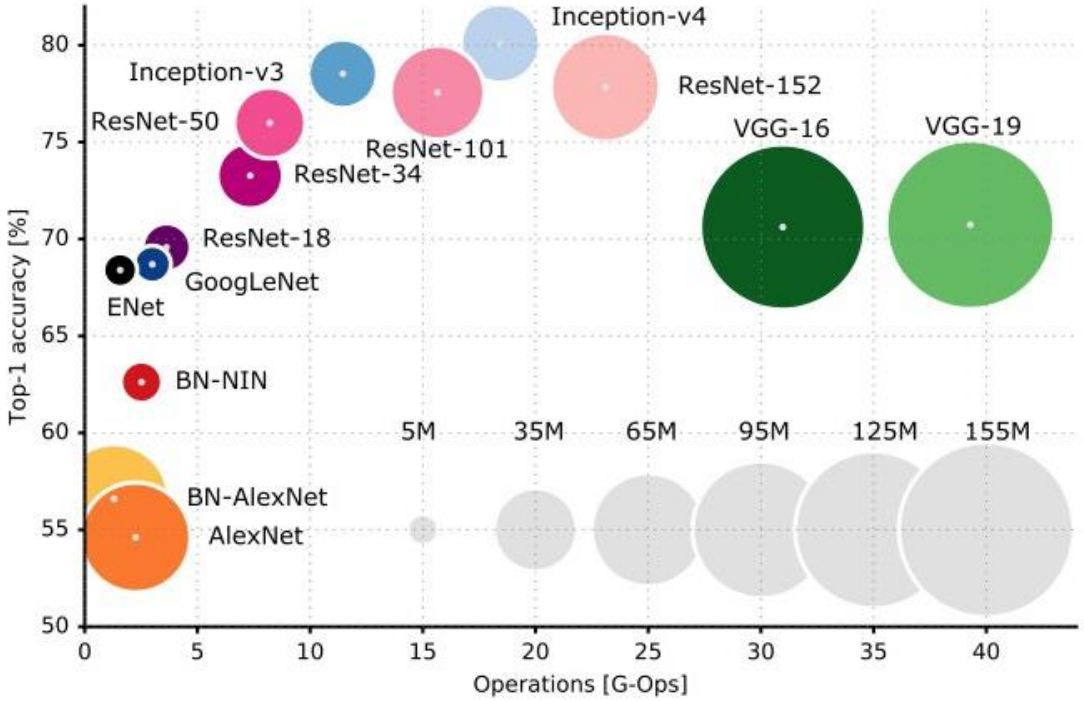


Figure 4.28: Top models in last 5-10 years in ImageNet competitions (Spark, 2017).

Pre-trained neural networks are usually used for image recognition. It is possible to use these pre-trained deep neural networks to predict the prices of financial instruments using the approach of feeding these pre-trained ANNs by datasets with price and volume charts of various time frames as images.

The transfer learning technology uses pre-trained ANNs for solving problems that differ from those tasks on which these the ANNs were trained. The fine-tuning method is an improvement on the transfer learning technology. During the fine-tunng, not only the new classifier is trained with a new dataset. The next step is training one or two of the last convolution layers (or blocks) with the new dataset. Simultaneously, all other convolution layers and the new classifier of the deep ANN have to be frozen. Those the weights (parameters) of the other convolution layers will not be changed.

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
```

Algorithm 4.1: The VGG-16 topology (Simonyan and Zisserman, 2014).

Figure 4.29 demonstrates part of the changed VGG-16 topology, which has been trained during the fine-tuning.

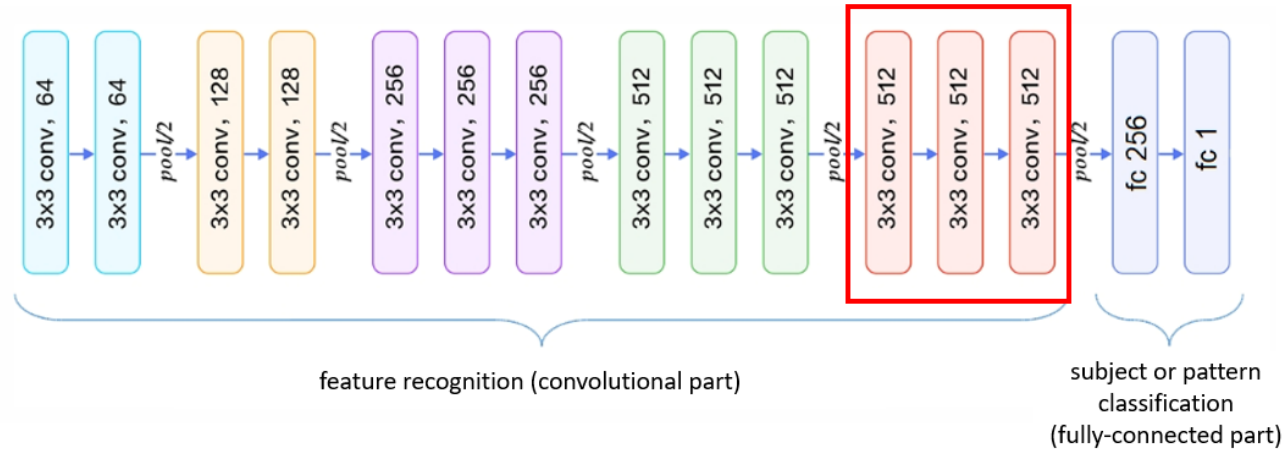


Figure 4.29: Part of the new VGG-18, which has to be trained (Sozikin, 2022).

The additional training of pre-trained CNNs by feeding them short-term (up to 200 ticks) sets of tick price and tick volume data in the form of charts is evident and elegant. However, the implementation required access to the significant computing power of GPUs. The lack of such access prompted the author to formulate a method for advanced preprocessing of big financial tick data, which essentially trains 1D CNNs instead of 2D CNNs. That significantly saves processing resources. The next chapter will provide a detailed description of this data preprocessing and the creation of a data tensor.

4.10.Novelty of data analysis

An important novelty of this research is the analysis of the prices and volumes of each tick, which are rarely used to train neural networks, but also the analysis of time intervals between neighbouring ticks. Raw financial data usually contains information about the transaction time, stored in the standard Unix-time format, which is an integer number of seconds from January 1, 1970 (raw data structure is shown in Section 4.8.1, Table 4.6). However, for intraday trading, it does not matter how many seconds have passed since this date over 50 years ago. More important is the size of the interval in seconds or milliseconds between neighbouring ticks because it demonstrates the intensity of trading, the frequency of transactions and could show patterns in market participants' behaviour. Inter-tick intervals precisely demonstrate the intensity of trading, on which all other parameters depend, such as trading volumes, volatility and safe stop-losses. In this research, all tick time data is converted into values of inter-tick intervals, which are fed into the convolutional neural networks along with parameters commonly analysed by other researchers, such as prices and volumes.

Another crucial novelty of this research is the transformation of irregular time series into regular time series. This innovation is possible as a consequence of the transformation of the whole array of transaction times into the array of inter-tick intervals, which, together with data on prices and volumes of tick transactions, is input data to neural networks. Thus, this conversion of irregular time series to regular allows applying all the analysis methods typical for typical time series. Thus, this research avoids the typical methods of converting irregular time series to regular ones using various variants of mathematical interpolation methods. In the case of data from financial markets, this has the benefit of avoiding the loss of critical information about the financial market participants' behaviour.

The first series of intermediate experiments (Chapter 5) use the data pre-processed, as described in this chapter, and introduce the following novelties.

- using tick data together with convolutional neural networks without using typical recurrent neural networks,

- non-linear unique normalisation of data on the price and volume of each tick (minimum price change during trading),

- non-linear unique normalisation of time intervals between tick data, which includes crucial information about the structure and intensity of the market at each moment. It also allows conversion of irregular time series into regular time series and the application of all the analysis methods typical for typical time series,

- using a three-dimensional (3D) data tensor consisting of a set of data slices for feeding convolutional neural networks,

- reducing the dimension of data by replacing two-dimensional convolutional neural networks with one-dimensional while preserving the three-dimensional structure of the input data tensor.

The additional novelties are applied in Chapters 6 and 7, describing the key series of experiments:

- avoiding the need to predict a specific price of a financial instrument after a particular time and instead predicting the direction of recoilless movement and its depth,

- using a unique vector of the probability of market movements, which makes it possible to move from a regression task to a classification task,

- a new approach to significantly reducing data and removing flat market movements.

Each of these series of experiments requires preliminary data cleaning since there are errors, nulls and a lack of raw data, which significantly negatively affect the result. Convolutional neural networks have been chosen for the following experiments based on the concept described in Section 3.5 and Section 3.6 because they are heirs of Japanese candlestick charting techniques, which have previously been used to find entry points to the financial market in short-term (intraday) trading. The predicted movements should be recoil-free so as not to fix stop-losses, which can destroy the deposit (Section 2.9).

Results, problems, and directions for the subsequent experiments are discussed at the end of each of the above mentioned chapters.

This chapter described the research methods and tools used in this research. The iterative algorithm and conceptual framework of this research were described in detail. It also listed the utilised ML and DL frameworks and described the financial market terminals. Methods and tools for data collection were demonstrated. Databases used in this research and preparation of this data for subsequent experiments were also described. Some methods of preliminary data analysis, including visualisation, were demonstrated. The features of training, validation and testing for time-series of financial data were indicated. An algorithm for avoiding overfitting and the nuances of validation and advanced ratio and metrics for valuation of the success of trading systems were described.

The following chapters describe the main experiments of this research which trained neural networks with big financial tick data. Such tick datasets are rarely used in scientific research due to problems with their size and availability for academic researchers. This neural network analysis focuses on intraday trading and, therefore, analyses only short-term movements, identifying only small but stable profitable movements that should end before the close of that trading day.

5. Prediction of the Forex EUR/USD Next Tick Rate

This chapter describes the first of the four series of experiments that use big tick financial data. At the beginning of each series of experiments, the raw data used, its transformation and non-linear normalisation are described in detail. The topologies and hyperparameters of the neural networks used and the output data are described.

The aim of the first series of experiments is predicting the Forex EUR/USD next tick price based on previous tick prices and volumes during the preceding short-term period. Predicting the next tick, which reflects any minimum change in the trading price, was chosen as the most straightforward regression task for testing all MLP and CNN training algorithms, analysing database cleanliness, error validation, and controlling times for calculations. These experiments are also necessary to evaluate the quality of the architecture of the ANNs used, determine the required processing time for such calculations based on big data, and determine future trading strategies.

The first series of experiments is a short-term regression task, which makes it possible to predict tick prices by analysing whole big financial tick data. However, it has some disadvantages in terms of error metrics and requires massive computer (GPU) power or extensive processor time to train these ANNs.

The expediency of such short predictions as the price of an index for the next minute or the next tick price for stocks and currencies can be explained by a simple logical statement that follows from the concept of causal determinism in economics (Chen, Kaboudan and Du, 2018) and is vividly described by the conception of Laplace's demon (Johnson, 2017; Grünke, 2019; Gruner and Bartelmann, 2020). If it is possible to predict the future one second or a minute ahead, it is possible to predict the future any time ahead. Of course, in the classic formulation, Laplace's demon is impossible. However, if the purpose is significantly narrowed down to predicting the price of a randomly chosen financial asset within the next tick, second, or minute; and if the purpose is considered successfully achieved, in case a certain probability of the prediction is reached, then the purpose can already be achieved using ML methods. It would be added to the formulation of the target that at each step, unlike Laplace's demon, the problem of predicting the stock price for the next short period will be solved based on the actual input data obtained from financial markets, which correct all possible inaccuracies in

this prediction at the previous step and thermodynamic effects. The paraphrasing definition of the limited Laplace's demon is that the prediction is possible by calculating the future price of a financial asset per tick, second tick or minute ahead with a certain probability. However, for predicting the prices of this financial asset for an unlimited time ahead, it is vital to do this calculation every second based on the current and correct input data received every second from the financial market.

5.1. Transforming Raw Data

This section aims to describe in detail how pre-processed and normalised data can be transformed into a three-dimensional tensor using a sliding window. These transformations are necessary to simulate datasets for further comfortable analysis using convolutional neural networks. Initially, convolutional neural networks were created and proved themselves in the analysis of images. Due to the fact that they are able to find patterns anywhere in images, their ability to find patterns therefore invariant to scales, shifts and rotations of objects in images. CNNs analyse images of identically labelled objects and find patterns common to all these objects, inherent only to these identically labelled objects. Other identically labelled objects in this database also have their own patterns by which CNNs combine them and subsequently recognise them.

In the main big financial database, the role of the patterns mentioned above is taken on by certain combinations of prices, trading volumes and values of inter-tick intervals. These patterns are searched for on a certain number of rows, which is equal to the depth of this sliding window. Thus, analysis of short-term trading can be compared with the analysis of images. The role of pictures in the financial analysis is performed by a chart of tick prices and volumes, where abscissa is time. These 2D prices and volumes charts can also be converted into a time series set that can be analysed using a sliding window. When a new tick appears, the CNN will try to predict the next tick, looking 200-ticks back from the last tick, which is an average from 2 to 10 minutes ago, depending on the intensity of the market at that moment. Thus, at each step, CNN sees only prices, volumes and inter-tick spacing for the last 200-ticks. It tries to find patterns, which have already appeared before and that changed the market's direction or preceded a significant price movement. These are some aims of the training of the CNNs in this research.

Short-term patterns are repeated because, in each specific market, there are large sellers and buyers, as well as market makers, who have their own behaviours when making deals (Khedr and Yaseen, 2017). It is these behaviours that create patterns. Behaviour changes and patterns also change over time. However, these changes do not occur every moment, which allows the identification of duplicate patterns (Gurav and Sidnal, 2018).

There are two ways to analyse big financial data using convolutional neural networks. In the first classical method, graphs of changes in prices and transaction volumes can be presented as two-dimensional (2D) images. In this case, the analysis should be processed by a two-dimensional CNN.

Figure 5.1 shows a two hundred ticks sliding window for finding patterns. These charts are fed as inputs for the two-dimensional CNN, for training and subsequent pattern recognition.

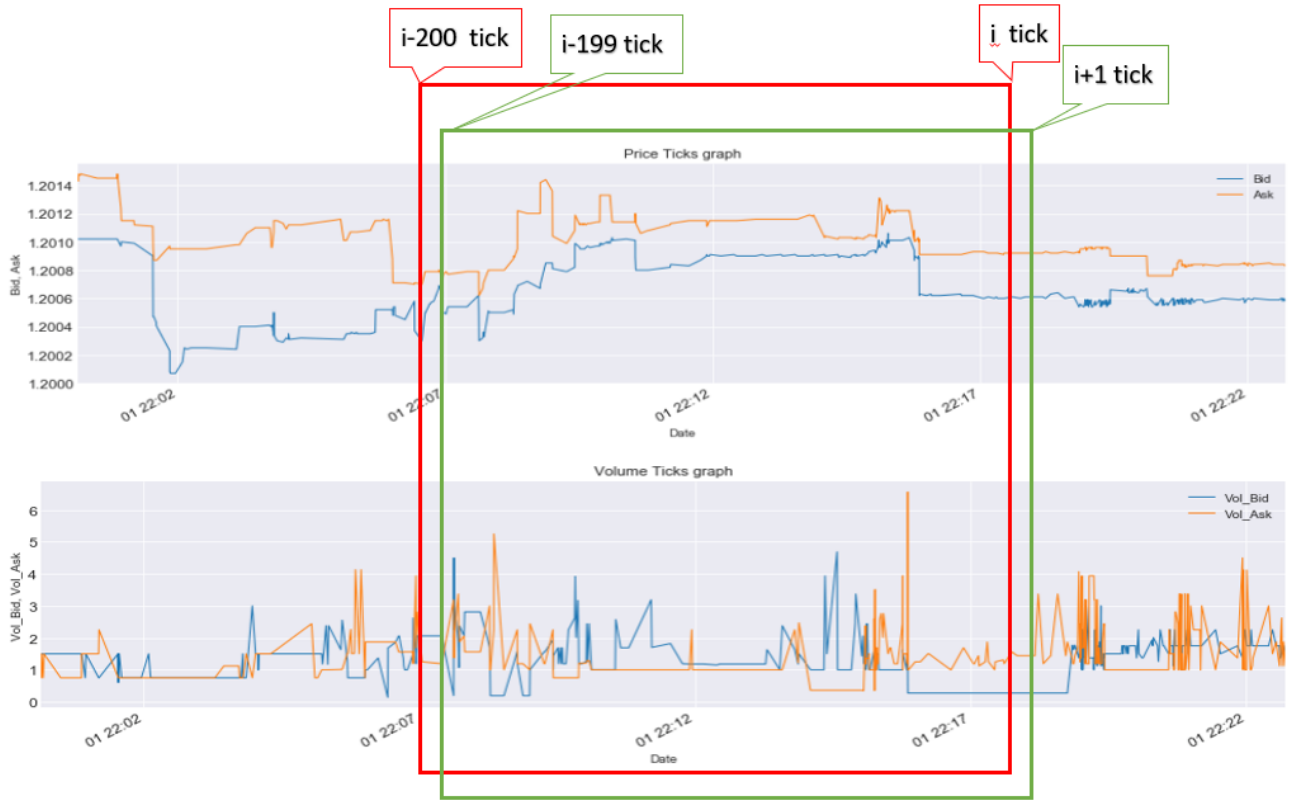


Figure 5.1: Two hundred ticks sliding window (images) for finding patterns by two-dimensional CNNs.

This approach has both advantages and disadvantages. This method's disadvantage is the increased requirements for processing power due to the significant size of the calculated parameters of the CNN. One more disadvantage of graph analysis is the sparseness of the

graphs compared to images, which can lead to a large number of near zero elements in the weight matrix, which can be the reason for the instability of the neural network training. The advantage of the analysis using two-dimensional CNNs is the possibility of using pre-trained neural networks, such as VGG-16 and others, using transfer learning and fine-tuning technologies. However, when analysing big financial tick data, the requirements for processor power become significant, as described in detail in the chapter on preliminary tests and experiments (Section 4.9.2).

The second method uses a more straightforward input data structure. This data flow of normalised tick prices, volumes, and inter-tick intervals is transformed in arrays and analyses using a one-dimensional CNN. Figure 5.2 demonstrates the second approach. This approach is less computationally expensive. Nevertheless, this method preserves the main advantage of CNN - an analysis of patterns of data combinations.

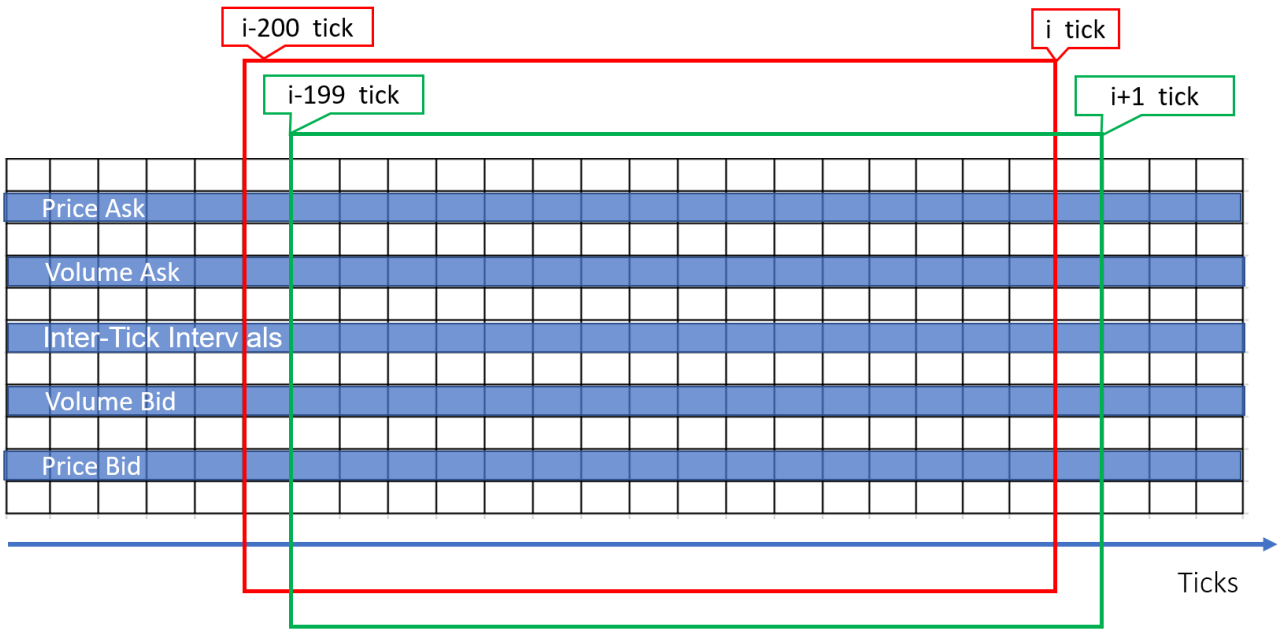


Figure 5.2: Two hundred ticks sliding window (2D-array) for finding patterns by one-dimensional CNN.

In both of the approaches mentioned above, two-dimensional charts (pictures) or arrays of normalised tick prices, volumes and inter-tick intervals are transformed into the three-dimensional tensor of charts (pictures) or 2D-arrays. The negative result of this approach is that the size of the data increases dramatically, namely by 200 times. A positive result is that it is possible to use high-performance CNNs, and there is no need to use RNNs based on cycles, which increase the number of calculations.

When using one-dimensional CNNs, the full third-dimension tensor with dimension [(87,993,711 - 200) × 200 × 5] ≈ 87.99 billion

is split into 1 million pieces with dimension

[1,000,000 × 200 × 5] = 1 billion.

Where 200 is the number of ticks equal to the depth of the sliding window;

5 – is the number of features in the database, consisting of ask and bid prices, ask and bid volumes and inter-tick intervals.

The product of three dimensions [1,000,000 × 200 × 5] of each part of the three-dimension tensor was chosen as the maximum possible RAM for the CNN training on the publicly available GPU cloud Google Colab.

The above conversions require significant processing time since the total number of numbers is about 88 billion ≈ (87,993,711 × 200 × 5) of float-32 format. More than 45 hours of net processor time were spent on normalization, transformation and preparation of input data for feeding into the neural networks during the experiments.

On the other hand, the above transformations of big financial data convert time series with an irregular structure of time intervals to standard time series formed into arrays ready for feeding as input data into convolutional neural networks. Also, it became possible to apply all the analysis methods for typical time series.

5.2. Prediction of Forex EUR/USD Tick Rates by the Multilayer Perceptron

This experiment with the MLP was made to further compare accuracy with the results from the CNN. The same four hidden layers MLP (described in Section 4.9.1) was used in this example too but with historical data of Forex EUR/USD ticks that was described above.

The aim of the test is the prediction of the price of the next tick based on previous tick prices during the preceding short-term period. In this short period, 400 ticks are considered to be the equivalent of approximately 20 minutes. At every step, 400 previous ticks are used to teach the ANN and try to predict the next one.

Algorithm 5.1 demonstrates the topology of the MLP. It has four hidden layers (from dense_6 up to dense_9) and the exit (dense_10) layer. The term ‘dense’ in Keras means a fully-connected layer. The exit layer has two cells for predictions: the first for the ask price and the second for the bid price. This topology has more than 4.8 million weights of the matrix for the calculation.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 2048)              2050048
_________________________________________________________________
dense_7 (Dense)              (None, 1024)              2098176
_________________________________________________________________
dense_8 (Dense)              (None, 512)               524800
_________________________________________________________________
dense_9 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_10 (Dense)             (None, 2)                 514
=================================================================
Total params: 4,804,866
Trainable params: 4,804,866
Non-trainable params: 0
```

Algorithm 5.1: MLP with four hidden layers.

Figure 5.3 shows the result of the prediction. The graph has a ‘white noise’ structure. It means that the prediction of the next tick price might be impossible using a multilayer perceptron. However, the probability of prices for the next short integral period, for example, on a five-minute time frame, might be predicted. More complex ANNs, such as CNNs could predict the probability of the prices for the next short period. The following experiments will demonstrate the assumption.

Figure 5.3: White noise structure.

The fully bound (dense) MLP has weaknesses. The most significant of them is that MLP architecture does not have the invariance of pattern recognition, which is in CNN models. MLP architecture also has a large number of weights in the matrix for calculation. One of the ways for accelerating and improving the result of MLP training is a dropout. Dropout is a procedure of randomly dropping some of the cells inside hidden layers (Srivastava, et al., 2014). However, another way to recognise patterns could be to use CNN models, as in this

application, CNNs are an extension of Japanese candlestick analysis on a more detailed tick scale.

5.3. Prediction of Forex EUR/USD Tick Rates by the CNNs

For predicting the price of the next tick, one-dimensional CNNs were used, which are often used to process time sequences along with RNNs. However, this research uses irregular intervals between neighbouring ticks and a three-dimension tensor (as described in Section 4.8.3 and Section 5.1), which is fed to CNNs as a separate feature (column). A supervised learning method was used for training the CNN. The answers (output data) were the next tick's ask and bid prices that the CNN could not see during the training.

The experiments were carried out with a four-hidden-layers and five-hidden-layers one-dimensional CNN. Algorithm 5.2 demonstrates the four hidden layers CNN.

```
model = Sequential()
model.add(Conv1D(200, 9, padding='same', input_shape=(200, 5), activation='relu', kernel_initializer="normal"))
model.add(MaxPooling1D(7))
model.add(Conv1D(400, 7, padding='same', activation='relu', kernel_initializer="normal"))
model.add(GlobalMaxPooling1D())
model.add(Dense(100, activation='relu', kernel_initializer="normal"))
model.add(Dense(2))
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 200, 200)          9200
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 28, 200)           0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 28, 400)           560400
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 400)               0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               40100
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 202
=================================================================
Total params: 609,902
Trainable params: 609,902
Non-trainable params: 0
_________________________________________________________________
None
```

Algorithm 5.2: The four hidden layers CNN.

For both initial training and the testing of new ideas, a dataset of 1 million rows was used, which is approximately two weeks. For training convolutional neural networks, sets of 1 million

row datasets were formed, the starting points of which were chosen randomly throughout all the mentioned-above three years using the Python random number function. Training on the whole three-year database was not possible due to limited processor resources. Nevertheless, the received results of training and validation accuracy of these CNNs are stable, and it can be assumed that the sample used is statistically relevant.

Due to a massive amount of input data, the training of CNNs was carried out iteratively, approximately one million ticks each on Google Colab. This approach to training emulates further training in real trading. CNNs are trained over the weekend when financial markets are closed by using the previous week's market data.

This series of experiments was carried out with data from three years, separately. The results of this series of experiments are presented in Appendix I. The summary of these experiments is shown in Table 5.1 and analysed in Section 5.4.

One of the 45 Python codes from this series of experiments is demonstrated in Appendix X. These codes of prediction of the next forex EUR/USD tick rates by the CNNs are similar to each other but differ in the input data used, the neural network training parameters, and the results.

Based on the size of the financial ticks database specified in Section 4.8, the three-dimensional tensor of each year is divided into 27 to 33 blocks, each of them consists of one million rows. Conducting continuous training on all 89 blocks would require a long processor time on the hardware used (specified in Section 4.3.5) and on the Google Colab platform, comparable to 160 hours of net processor time. By this time, it is necessary to add the time for uploading data to the Google Drive cloud for use with Google Colab and organisational time charges associated with the iterative nature of the series of these experiments. However, Google Colab declares the ability to train neural networks for only 12 hours continuously. However, in reality, this period is significantly less since Google Colab is taking measures against the mining of cryptocurrencies on their portal. As a result, the series of experiments presented in this section required about 34 hours of net processor time (excluding the time for normalisation, transformation and preparation of the input data).

Therefore, in each of the three years, three neighbouring blocks were randomly selected, in which they were sequentially fed to this CNN. In this case, the CNN trained on the previous data block was used as an input model for training on the following data block. Thus,

additional training of the CNN was simulated in actual trading, when additional retraining occurs on weekends according to the past week's data.

The hyperparameters of the one-dimensional CNN (as described in Section 5.3) did not change during the series of these experiments. This CNN has four layers, and the convolution kernel consisted of 9 elements, representing a one-dimensional vector. The number of epochs varied from 10 to 15 to cover the beginning of overfitting. This number of epochs after which the overfitting of the neural network model begins is specified for each experiment consisting of three consecutive input data blocks. The time required for each block's workout was also measured, and the average time required for each epoch was calculated. One of the variables was the batch size in the direction in which the gradient descent was simultaneously performed. This batch size varied from 15 to 100. If the batch size was below 15, then training the network took significantly longer and tended to the classic (stochastic) gradient descent. On the other hand, it makes no sense to increase the number of batches over 100 since this number becomes comparable to the depth of the sliding window (200 ticks), which is equal to one of the three-dimensional dimensions tensor of the input data.

Table in Appendix I consists of blocks, which indicate both the synthetic results of training this CNN on three sequential blocks of data, consisting of one million rows each, and randomly selected sets of 10 samples of absolute errors of bid and ask price prediction. Each of these ten samples was randomly selected, using the random generation function in Python, from 3 million samples, which are the results of the CNN predictions on the test datasets (as described in Section 4.6.1) using these three block datasets as inputs. Thus, the result of this series of experiments is a dataset of 45 million lines, including the predicted ask and bid prices, as well as the absolute prediction errors for ask and bid. These prediction errors for bid and ask prices are more visually demonstrated by Figure 5.4, Figure 5.5 and Figure 5.6.

These charts' scale is also automatically determined based on randomly selected intervals of 120,000, 1,800 and 300 ticks. It can be seen from the graphs that the prediction errors over these randomly selected intervals are minor, but there are outliers when the errors reach from 8% to 26,5%. This is primarily due to new patterns of behaviour of market participants that appear periodically.

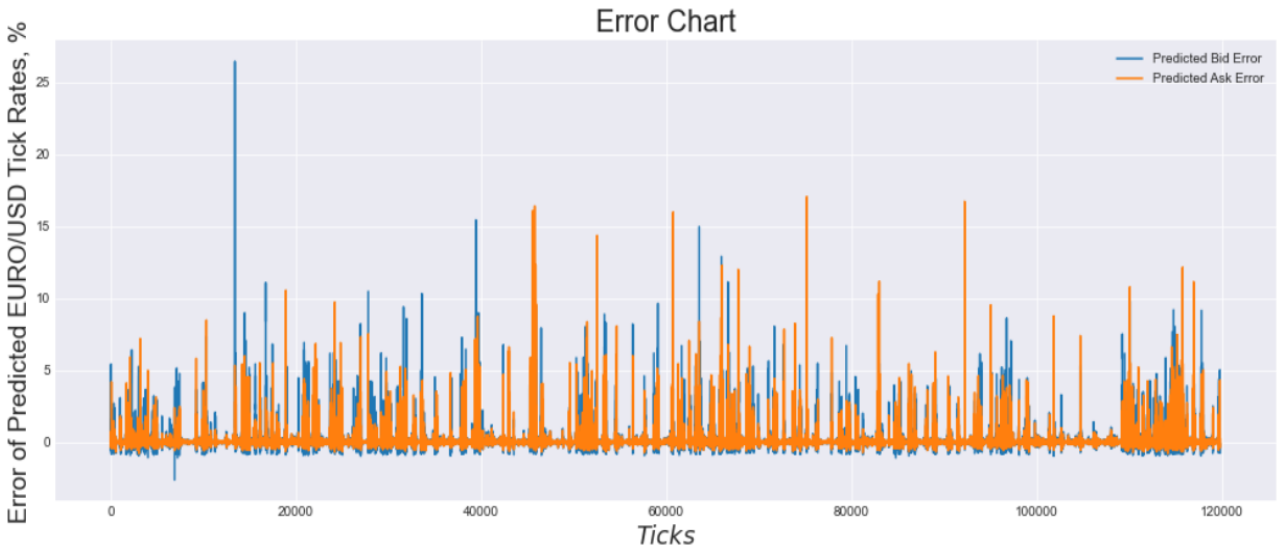


Figure 5.4: Error chart for bid and ask EUR/USD Tick Rates (Prices) for 120,000 random ticks.

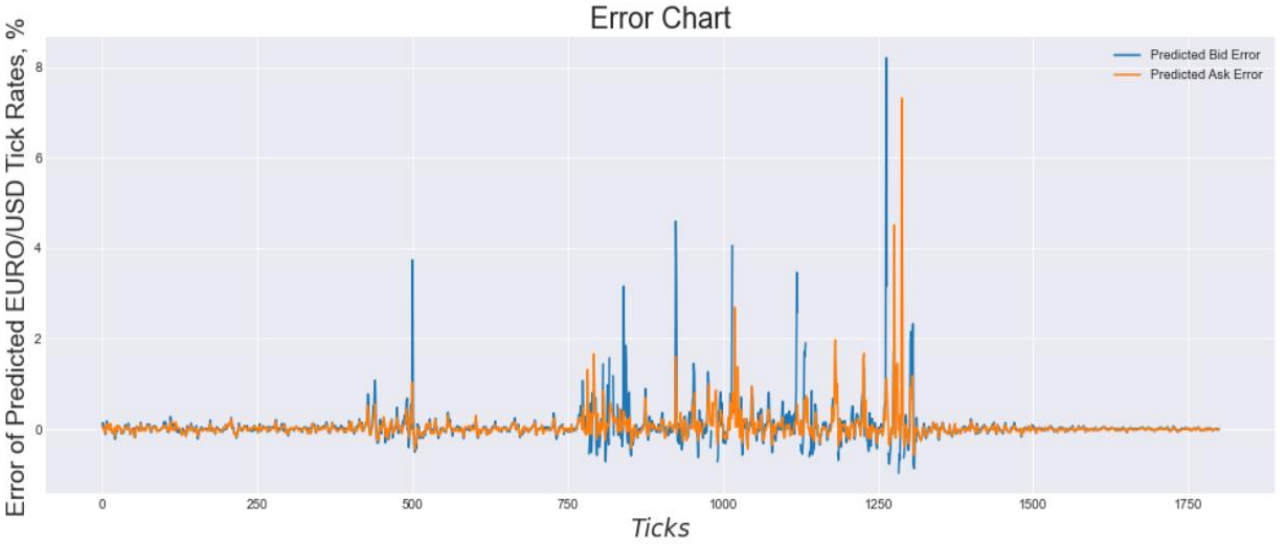


Figure 5.5: Error chart for bid and ask EUR/USD Tick Rates (Prices) for 1,800 random ticks.

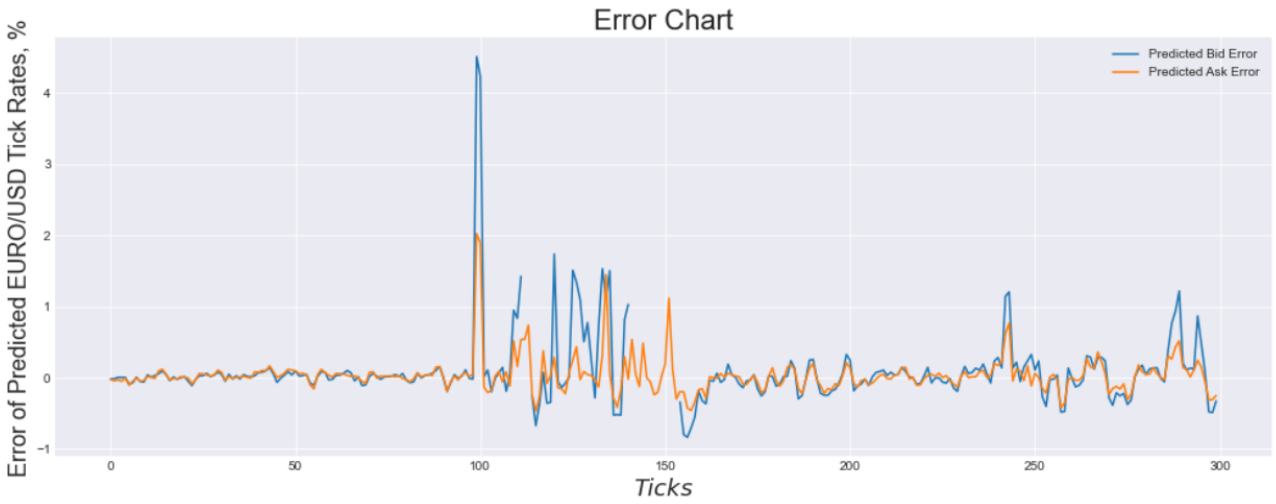


Figure 5.6: Error chart for bid and ask EUR/USD Tick Rates (Prices) for 300 random ticks.

The higher level of error percentage (outliers) for bid prices than for ask prices is explained by the fact that financial markets almost always fall faster than they grow, and new patterns of behaviour of market participants also reflect these nuances of movements in different directions.

5.4. Evaluation

This section describes two types of evaluation: statistical, where various CNN experiments' results are analysed, and comparative, where the CNN is compared with GRU and LSTM. The evaluation results of the next tick price prediction experiment are also compared against baseline state-of-the-art techniques.

5.4.1. Statistical Evaluation

The integrated parameters and error metrics results are shown in Table 5.1.

Batch size	Epochs number before overfitting (range)	time per Epoch, (seconds) (Average)	Validation Mean Absolute Error, MAE (Gl. Average)	Testing Mean Absolute Error, MAE (Gl. Average)	Testing Mean Squared Error, MSE (Gl. Average)
15	[5, 7]	618	0,0216	0,0307	0,00194
30	[7, 9]	424	0,0211	0,0300	0,00188
50	[5, 9]	241	0,0209	0,0327	0,00195
80	[7, 10]	196	0,0208	0,0296	0,00184
100	[11, 14]	186	0,0205	0,0295	0,00184

Table 5.1: Integrated parameters and error metrics results.

For achieving the purpose of predicting the next tick prices, it is essential to reduce the error in predicting a significant price movement (outlier). Training this neural network model is a regression task; consequently, it is necessary to observe the mean squared error (MSE) values more attentively than the mean absolute error (MAE) values. Due to its quadratic function, the MSE parameter is more sensitive to outliers than the MAE. Thus, by tracking the training losses of the neural network by MSE, we will get better models. At the same time, MAE is used when testing this CNN to determine the overfitting start point. Figure 5.7 shows two training plots during each epoch that were randomly selected from 45 experiments.

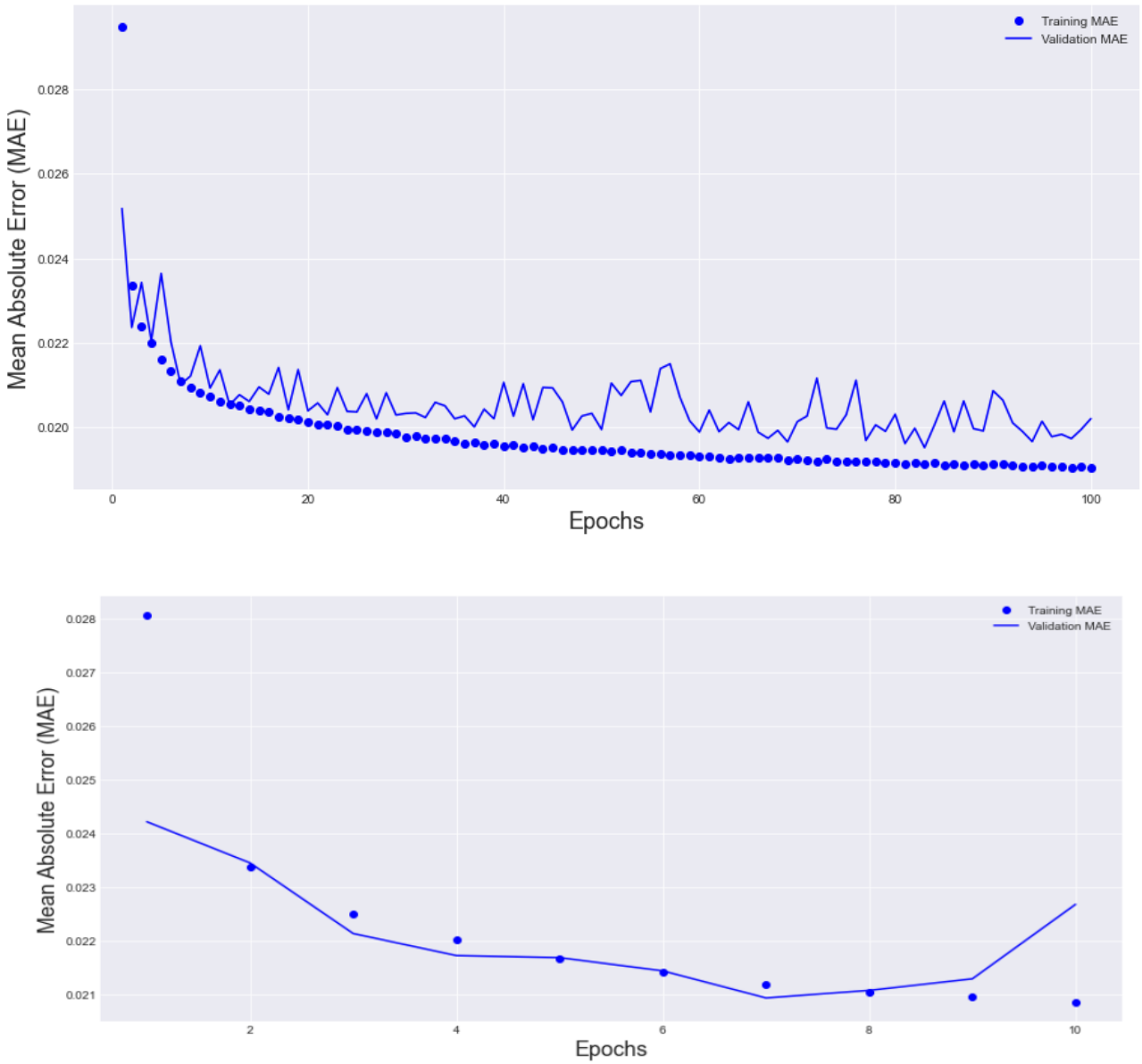


Figure 5.7: Training and validation mean absolute error (MAE) graph.

When selecting the parameters of the neural network model for further use, it is necessary to look at the MSE value on the test sample, which is indicated in Table 5.1 of the integrated results. The minimum MSE value was achieved using the data batch size parameter equal to

80 or 100. Moreover, with a large batch size, the average training time during each epoch decreases. However, batch size = 100 requires more training epochs, and hence longer training time. The training process used Keras 'callback function' for the 'EarlyStopping' with the parameter 'patience' from 5 to 30 and epochs=100, which allows to select the best number of epochs and avoid overfitting automatically (Figure 5.7). However, the loss functions did not improve significantly after 10-15 epochs, especially for the validation and test MAEs (for example, the best improvement was about $5 * 10^{-4}$, which is less than 2% of the best MAE), but the training time increased dozens of times. As a result, the 'patience' parameter was reduced, significantly saving processor time and resources without significantly losing training quality. However, the Keras 'callback function' with 'patience' parameter will be used extensively in binary classifiers in key experiments (Chapters 6 and 7).

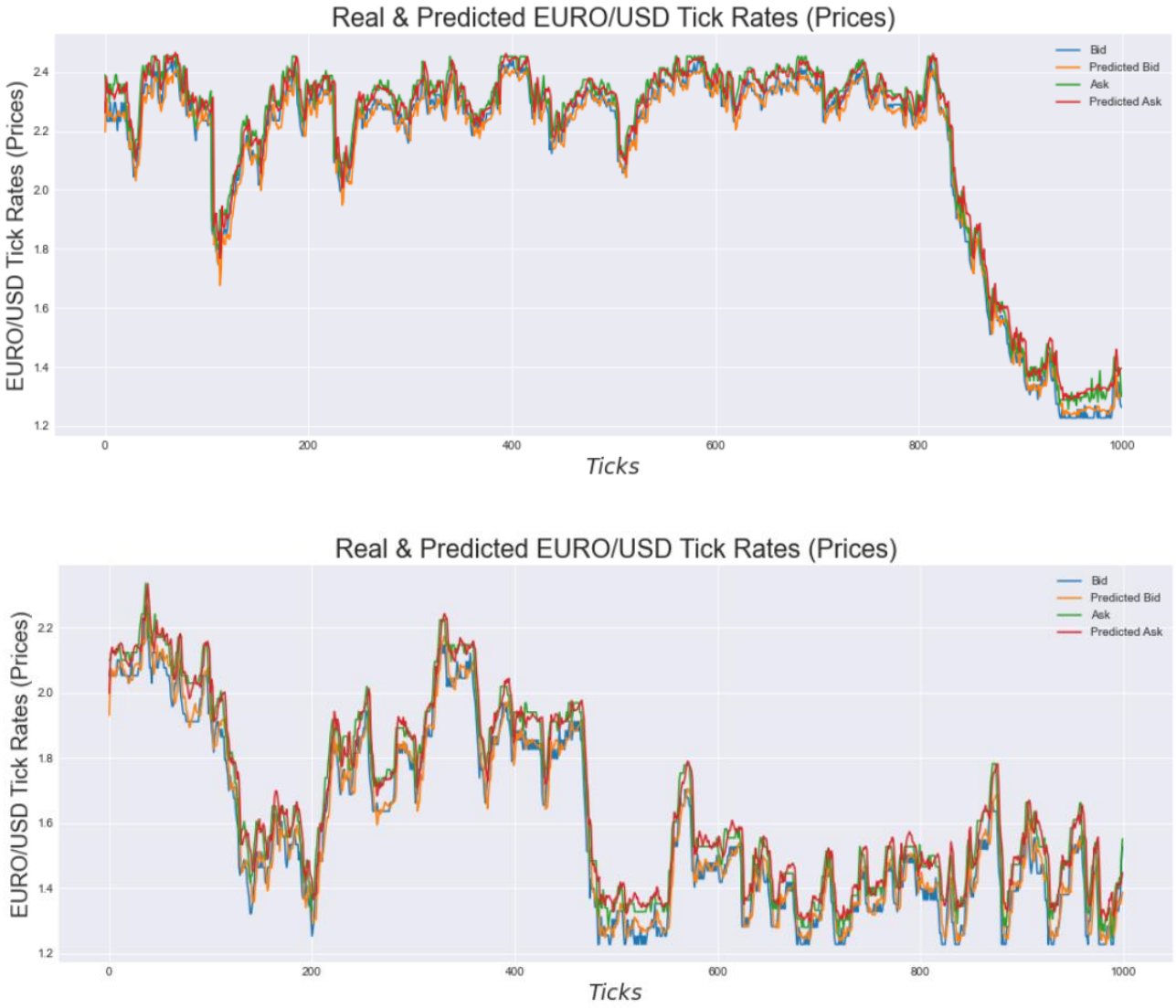


Figure 5.8: Real and predicted bid and ask EUR/USD tick rates (prices) for 1,000 random ticks.

Thus, this CNN model's integrated practically best result is MSE = 0,00184, which was achieved with batch size = 80 and the number of epochs as 10. The MAE value = 0,0208 for these variables is also the second-best (the best one is 0,0205 when batch size = 100 and the number of epochs 10).

A visual evaluation of predictions is also essential in trading. This allows one to immediately find apparent errors in the process of creating and training neural network models. Figure 5.8, and Figure 5.9 demonstrate, at different scales, the graphs of real and predicted bid and ask EUR/USD tick rates (prices). The scales on these graphs are determined automatically based on the total number of ticks in randomly selected intervals, consisting of 1,000 and 300 ticks, respectively. These graphs demonstrate that this CNN is quite accurate in predicting the next ask and bid tick prices. Each pair of graphs demonstrates quality prediction both at the moments of the trend and the flat.

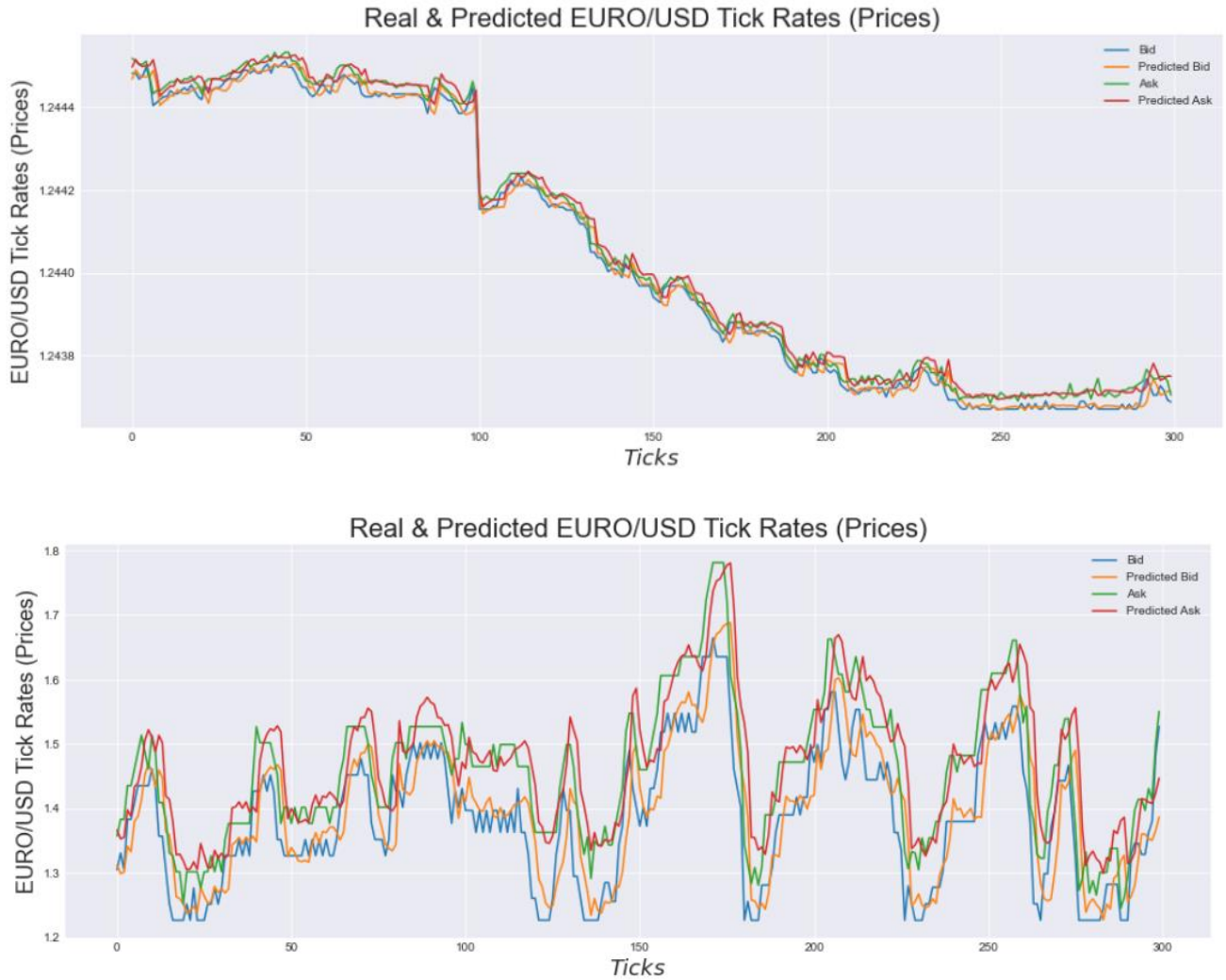


Figure 5.9: Real and predicted bid and ask EUR/USD tick rates (prices) for 300 random ticks.

5.4.2. Comparative Evaluation

For a comparative evaluation, two types of recurrent neural networks were used. The experiments were carried out on a set of three randomly selected consecutive blocks, each of 1 million ticks, which were used to train the CNN in the previous series of experiments. Thus, the total depth of the incoming data was three million ticks. Both recurrent networks used the same batch size = 80 as in the most successful CNN experiments. Python code for training both neural networks is demonstrated in Appendix XI.

The first neural network architecture has three layers, including a Gated Recurrent Unit (GRU) layer described in detail by Chung, et al. (2014). For avoiding overfitting, the dropout technology (described in Section 4.6.1 and Section 5.2) is used. Algorithm 5.3 demonstrates the GRU architecture.

```
model = keras.Sequential([

    keras.layers.GRU(32, dropout=0.1, recurrent_dropout=0.5, return_sequences=True, input_shape=(None, data_a.shape[-1])),

    keras.layers.GRU(64, activation=tf.nn.relu, dropout=0.1, recurrent_dropout=0.5),

    keras.layers.Dense(1)

])

print(model.summary())
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                    (None, None, 32)          3648
_________________________________________________________________
gru_1 (GRU)                  (None, 64)                18816
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 65
=================================================================
Total params: 22,529
Trainable params: 22,529
Non-trainable params: 0
_________________________________________________________________
None
```

Algorithm 5.3: Three layers Gated Recurrent Unit (GRU) neural network for regression.

Figure 5.10 shows the graph of MAE by training epochs. The range of MAE value in the validation process is in the interval between 0.1066 and 0.1981 and is unstable, which is much worse than for the CNN.

As a second architecture for validation, a two-layer LSTM neural network was used. The LSTM architecture is described in Section 3.4.2 and is used most often in the tasks of predicting the prices of financial assets.

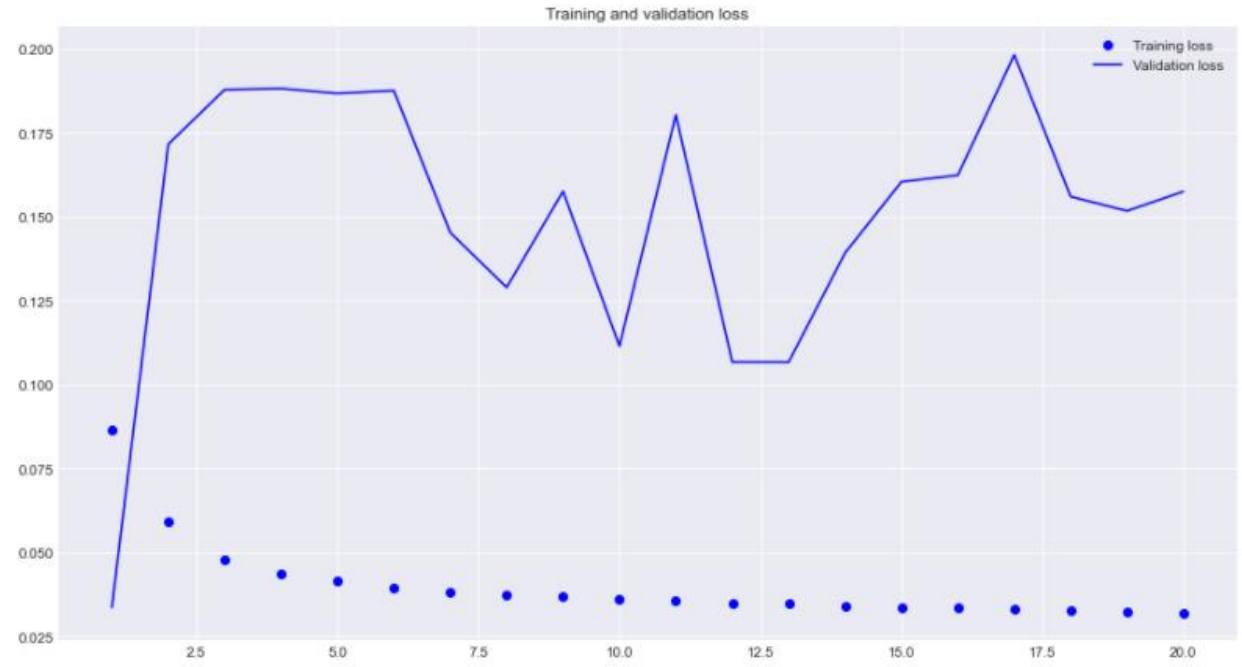


Figure 5.10: GRU Training and validation mean absolute error (MAE) graph.

Figure 5.11 shows the distribution of MAE by training epochs. The MAE achieved during the validation process by the LSTM architecture is equal to 0,0209, which is worse but comparable to the best analogous indicator for the CNN (0,0205).

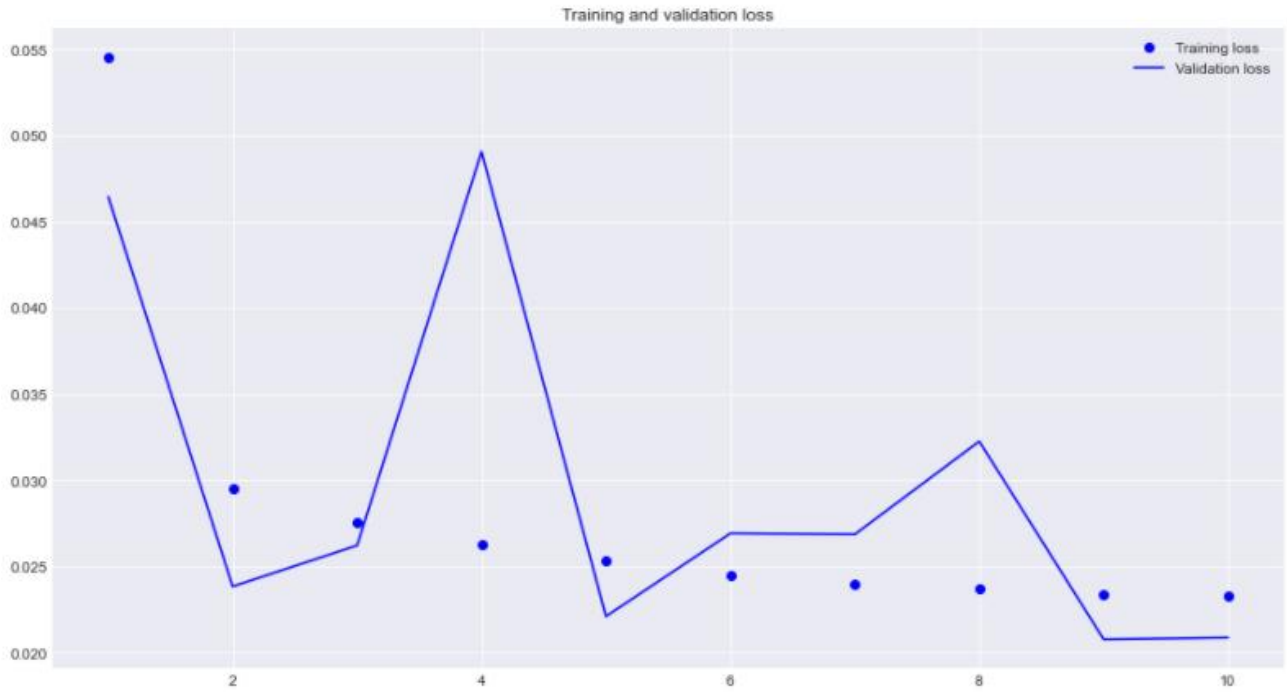


Figure 5.11: LSTM Training and validation mean absolute error (MAE) graph.

Thus, for short-term movements, the indicators of the quality of training of the CNN is slightly better than LSTM, but they are approximately similar. The results of GRU recurrent neural network architectures are worse than in CNN and LSTM. Table 5.2 and Figure 5.12 demonstrate these results.

Neural Network Architecture	Validation Mean Absolute Error, MAE	Batch size	Epochs number before overfitting
CNN	0,0205	100	11
LSTM	0,0209	80	9
GRU	0,1066	80	12

Table 5.2: Comparative evaluation of the quality of training and validation of the neural network architectures (CNN, LSTM, GRU).

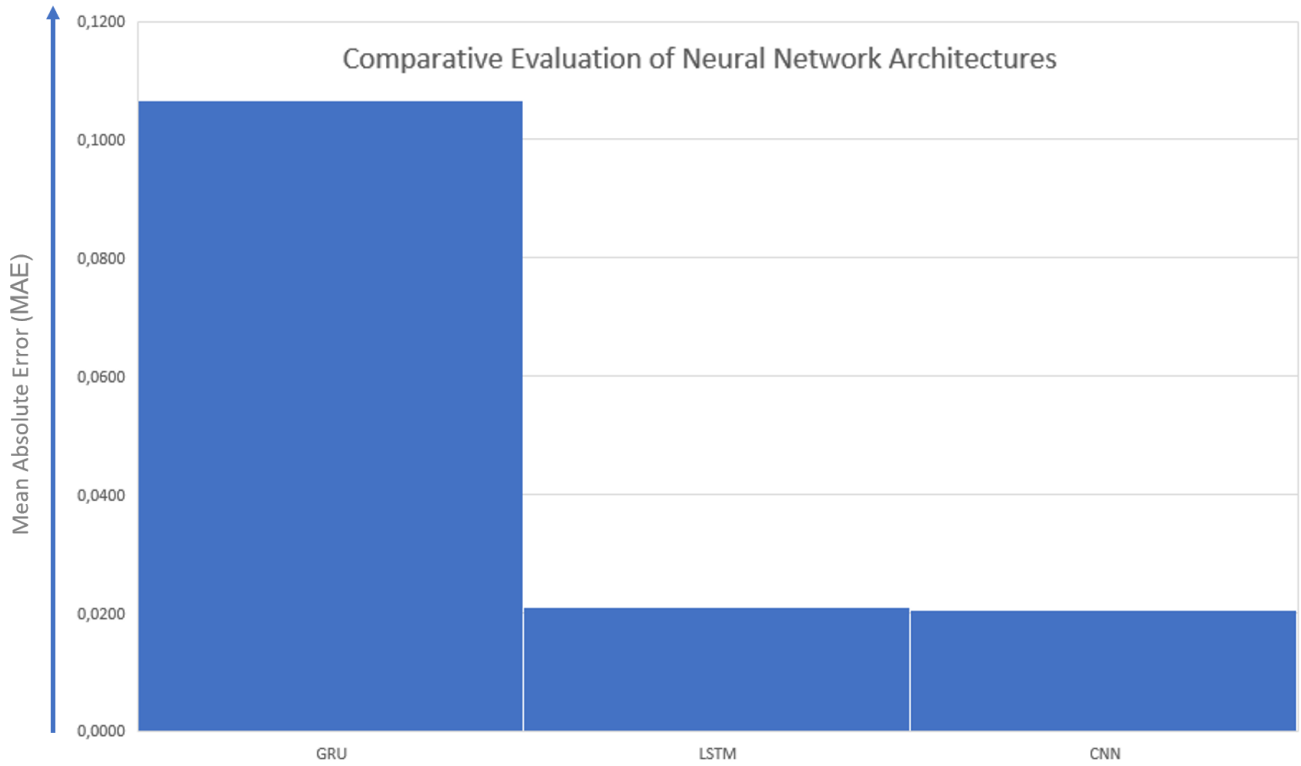


Figure 5.12: Comparative evaluation of the quality of training and validation of the neural network architectures (CNN, LSTM, GRU).

5.4.3. Evaluation against baseline state-of-the-art techniques

This section aims to compare and evaluate the results of the intermediate experiments on predicting the next price of the EUR/USD currency pair against baseline state-of-the-art techniques. As discussed in Sections 3.7 and 5.4.1, the evaluation is based on the standard mean absolute error (MAE) loss function.

Maneejuk and Srichaikul (2021) analysed the historical data of some currency pairs, including EUR/USD, together with data sets for macroeconomic analysis and technical indicators, emulating the most popular methods of Fundamental and Technical analysis. Experiments were carried out using various methods, such as classical ARIMA (Autoregressive Integrated Moving Average), SVM (Support Vector Machines), and various architectures of neural networks: simple ANNs and RNNs, including LSTM. In additional to the databases for Technical analysis, the researchers used several technical indicators, such as Momentum, Bollinger Bands (BB), Moving Average (MA), Rate of Change (ROCh), Commodity Channel Index (CCI), Moving Average Convergence/Divergence (MACD), and the Relative Strength Index (RSI).

Aggarwal and Sahani (2020) compared 22 countries' currency rate, including EUR/USD, prediction results against the United States dollar (USD) using various recurrent neural network architectures, such as simple recurrent neural networks (SRNN), long short-term memory (LSTM) and GRU.

The MAE results of the above experiments are shown in Table 5.3.

Paper	Method	Mean Absolute Error, MAE
(Aggarwal and Sahani, 2020)	LSTM (2020)	0.0475
	SRNN (2020)	0.2685
	GRU (2020)	0.0883
(Maneejuk and Srichaikul, 2021)	ARIMA (2021)	0.0435
	ANN (2021)	0.0319
	RNN (2021)	0.0853
	LTSM (2021)	0.0300
	SVM (2021)	0.0560

Table 5.3: Mean absolute error (MAE) of baseline state-of-the-art techniques.

Figure 5.13 demonstrates the comparison and evaluation of MAE results from this research against those published by Aggarwal and Sahani (2020) and Maneejuk and Srichaikul (2021).

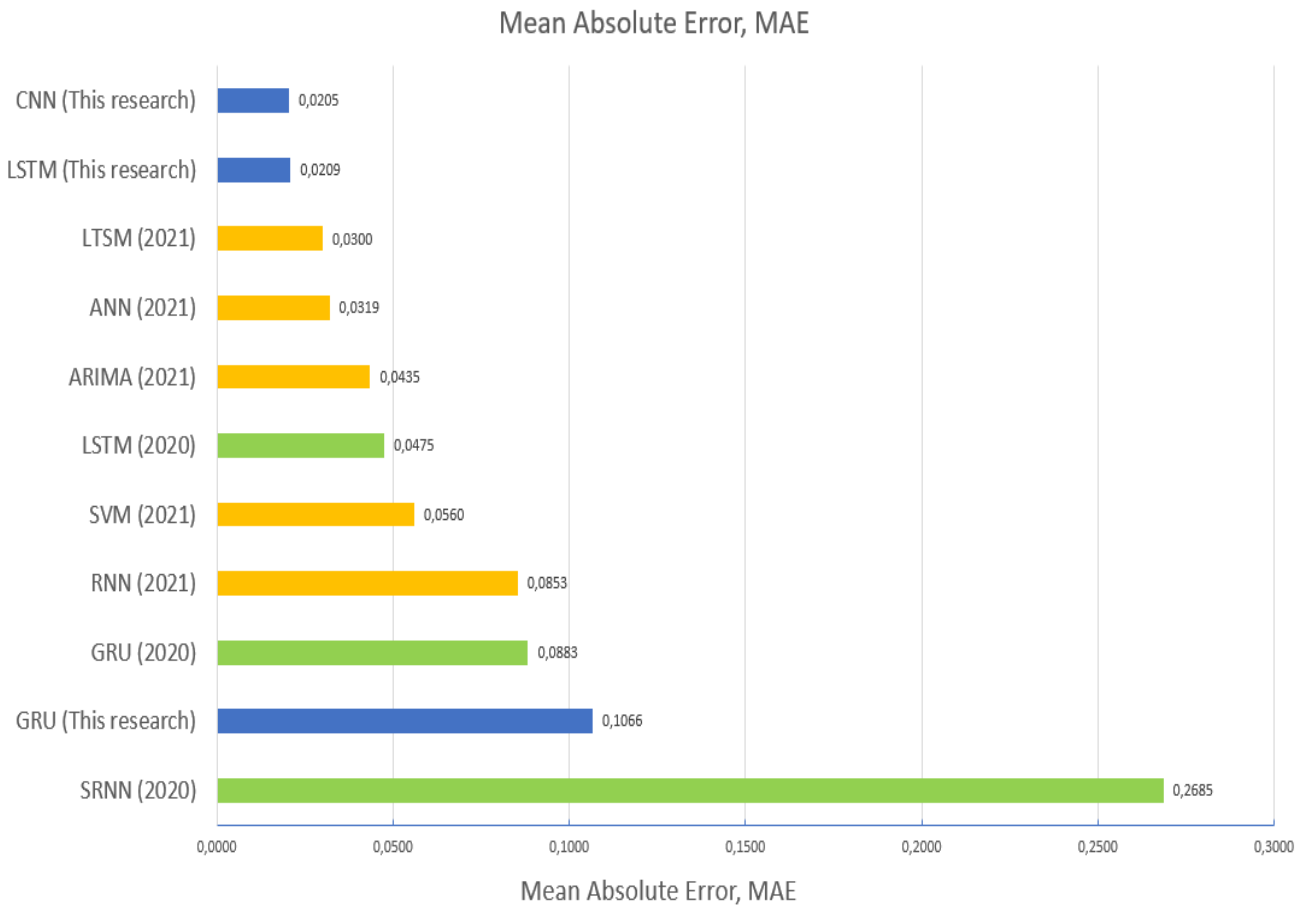


Figure 5.13: Comparison and evaluation of MAE results.

Based on this summary diagram, the following conclusions can be drawn:

- LSTMs are good at predicting the next price in the time series,

- CNNs can generate good results when using raw big financial tick data,

- classical statistical methods of time series analysis, such as ARIMA, also show promising results in predicting subsequent prices in financial data,

- machine-learning methods that are standard for the financial domain, such as SVM, show satisfactory results in this experiment.

5.5. Limitations and Discussion

As it can be seen in Figure 5.13, CNN trained on raw tick data to predict the next tick price based on patterns of behaviour of market participants and showed promising results. However, the neural network training process is limited by the processor power of GPUs widely available on the market or provided free of charge through the Google Colab platform. For maintaining low levels of prediction errors, the neural network model has to be constantly retrained. It is logical to do this retraining with a frequency of once a week on weekends, using the financial markets' data for the previous week (Fang, et al., 2021). However, new behaviour patterns may change due to some financial market participants arriving or departing from a particular financial instrument. This can lead to a decrease in the quality of prediction by the CNN as shown by periodic statistical outliers of absolute prediction errors.

Recurrent neural network models can perform well over short periods (as described in Section 3.4.2). Thus, to predict the patterns of behaviour of market participants, it is necessary to analyse the mutual combination of linearly independent raw data. Correct data interpretation, pre-processing, highlighting more essential features (such as inter-tick intervals) and data cleaning are also crucial.

Predicting the next tick is the basis for predicting the following several ticks, which gradually turns into probabilistic prediction using CNNs. From a practical point of view, when predicting financial instruments' prices, it does not matter what the price will be in 5 or 10 minutes. More important is in which direction and how far the price will change. In a short period, the price can change significantly and then return to the original levels. Nevertheless, this movement is sufficient to make it interesting for analysis. Thus, it is crucial to predict the probability of price movement in a particular direction, the depth of this potential movement with a certain probability and stop-loss level. The next series of experiments will be devoted to this task.

This chapter describes the first series of experiments for predicting the next Forex EUR/USD tick price based on previous tick prices and volumes during the preceding short-term period. Preparation of raw data in the three-dimensional tensor for subsequent experiments is also described. Two types of recurrent neural networks, which are most often used in existing research on the finance market analysis, were used to evaluate the results.

6. Management of the Probability of Significant Movement and Stop-loss Prediction

This chapter describes the next three series of key experiments using the classification task instead of the regression task used in the previous chapter.

The classification task uses an original probabilistic approach to predict future price movements of financial assets for more extended periods during a day trading session. The classification task also allows the flexible use of various error metrics. Furthermore, the experiments are based on a significantly reduced size of raw data, which uses a new method to remove all flat market movements that are not of interest to real trading. For intraday trading, the target strategies are those that recognise the patterns of behaviour of market participants, which will inform predictions of safe points to enter the market with minimal risk of generating losses from stop-losses orders.

This chapter also describes the series of experiments that use big financial tick data to predict the probability of the end of a flat financial market movement and the start of a significant movement and points with minimal risk for entry into the market. It also describes an attempt to determine the depth of a probable future significant movement based on a multi-parameter classification problem. Section 6.7.3 introduces splitting a multivariate classification task into a sequence of binary classifications in order to increase the quality of the prediction. All of the above binary classifications are interconnected and have to be performed sequentially. They correspond to the aim of this research, which is stated in (Section 1.3): to improve state-of-the-art performance for intraday trading using neural networks.

6.1. Introduction

The biggest problem to overcome is the continually changing nature of financial markets. Therefore, it is not enough to just train the CNN model once. This model should have the capability to automatically and permanently retrain.

It is quite simple to solve this problem by organisational methods, namely, by retraining an previously trained neural network using the newly obtained data set. Usually, additional

training of the neural network is performed during the weekend on data sets from the past week (Fang, et al., 2021). Thus, two problems are solved at once:

1) There is no interruption in the analysis of financial markets during trading time since training requires significantly more processing time than for subsequent analysis, and most markets are closed on weekends;

2) It is possible to find geographic regions where electricity is cheaper for industrial enterprises on weekends, which can be important when multiplying the research technology to many financial markets, which will lead to significant demand for electricity. On the other hand, it is possible to find cloud services with GPU processors with lower rental rates on weekends.

Since behaviour patterns depend on market participants, the above weekly retraining will demonstrate a stable result until there are changes involving the most active market participants; for example, they leave the financial markets or transfer capital to operations with another financial instrument. Error metrics and loss functions should be monitored weekly (Fang, et al., 2021). In case of deterioration of these characteristics, it would be desirable to train the neural network from scratch.

This research uses a sequential neural network training method that simulates the subsequent iterative industrial use process described above. The whole three-dimensional financial data tensor was divided into blocks of one and half million tick rows, which is approximately three weeks. In the future, in industrial use, there will be additional training of this neural network every weekend using exactly the same program code since the size of the incoming weekly data block will be less than one million, and therefore will fit into the limitations of Google Colab. Cross-validation techniques will be used to train the CNNs with the ‘rolling training set’ (Section 4.6.1). The majority of these series of experiments are calculated using the computer hardware specified in Section 4.3.5.

An important novelty of this experiment is that it has no purpose to predict the particular price of a financial instrument. Many researchers using machine learning (as described in Section 3.3) try to predict a particular price of a financial instrument over a predetermined time. Next, they determine the quality of training of the neural network using various error metrics or loss functions based on the differences between the predicted prices of the neural network and the actual prices at that particular point in time. From a practical point of view, this task is not a priority since, for actual intraday trading, it does not matter for what time the desired price

will be reached. More importantly is that the price will be reached with a greater than 50% probability before the end of the current trading day. For actual trading, it is also essential to predict that during this period after entering the market, the price does not go in the opposite direction and, as a result, the stop-loss (Section 2.9) is not activated.

Thus, predicting the price is not the primary purpose in predicting the movement of the financial market. A more important task is to calculate the probability of a market-moving in one of two possible directions. It should also predict the depth of this movement. This task is not a regression task, which determines the price of the next tick. It is a classification task of identifying the most likely depth and direction of the financial market's movement in the short-term (until the end of the trading day) movement.

For solving this problem, it is necessary to label the raw data to form a vector of answers for training the neural network so that this problem continues to be supervised learning.

The purpose of the current sub-chapter is to present an intraday trading method that uses a CNN, with raw tick data with variable interval transactions that preserve critical information. This will then predict the probabilities of the directions and depths of the next price movement. As opposed to the previous chapter, this one does not aim to predict a specific price level in the future, but only determines the probabilities of the direction and depth of movement, which is sufficient to open a long or short position in a financial instrument. The approach to replacing the regression problem with the classification problem is found in some scientific papers. However, the classification task in this paper means a simple recognition of the directions of the most probable market movement: up or down (Jiang, 2020), which perhaps is not a correct practical purpose since most of the time, financial markets are flat. Therefore, firstly, it is necessary to separate the flat from the trend and then determine the direction of movement.

As in the first series of experiments, the CNN was chosen due to the search for patterns of behaviour of financial market participants based on data analysis with a sliding window.

Closing a position is carried out by standard methods of money and risk management, such as trailing stop-profit orders, and it is not the goal of the model. Besides, this model proposes a method for reducing the number of calculations by pre-processing the data and considering only patterns ready for significant movement.

6.2. Additional Dataset with Japanese Candlesticks

The dataset, which is used in the experiments is the same EUR/ESD Forex tick data from the Dukascopy Bank (Switzerland) that was used in the previous experiments to predict the next tick rate (Section 4.5.2 and Section 4.8). Additionally, to calculate and estimate volatility data in the form of Japanese candlesticks with periods of one day, one hour, 30 minutes and 15 minutes is used along with tick data. With the use of integrated data in the form of Japanese candlesticks for the periods indicated above instead of big tick data, it is possible to reduce the number of calculations required to estimate the absolute volatility for the same periods. Table 6.1 demonstrates the size (in kB and the number of rows) of the data in the form of Japanese candlesticks for the different periods and compares this data with the size of big tick data.

The main parameters of a typical Japanese candlestick are shown in Figure 6.1, which has an open price, high and low prices, and a closing price. Each set of these Japanese candlesticks has a separate array for the bid and ask prices.

		2018	2019	2020
tick data (Big financial tick data)	**number of rows**	26 048 774	29 186 310	32 758 627
	kB	1 250 383	1 405 434	1 613 168
Japanese candlesticks with periods of (Bid + Ask arrays) one day	**number of rows**	520	518	520
	kB	32	32	34
Japanese candlesticks with periods of (Bid + Ask arrays) one hour	**number of rows**	12 408	12 408	12 456
	kB	740	738	744
Japanese candlesticks with periods of (Bid + Ask arrays) 30 minutes	**number of rows**	24 816	24 816	24 336
	kB	1 466	1 468	1 450
Japanese candlesticks with periods of (Bid + Ask arrays) of 15 minutes	**number of rows**	49 632	49 632	49 824
	kB	2 924	2 920	2 934

Table 6.1: Japanese candlesticks and big financial tick data sizes.

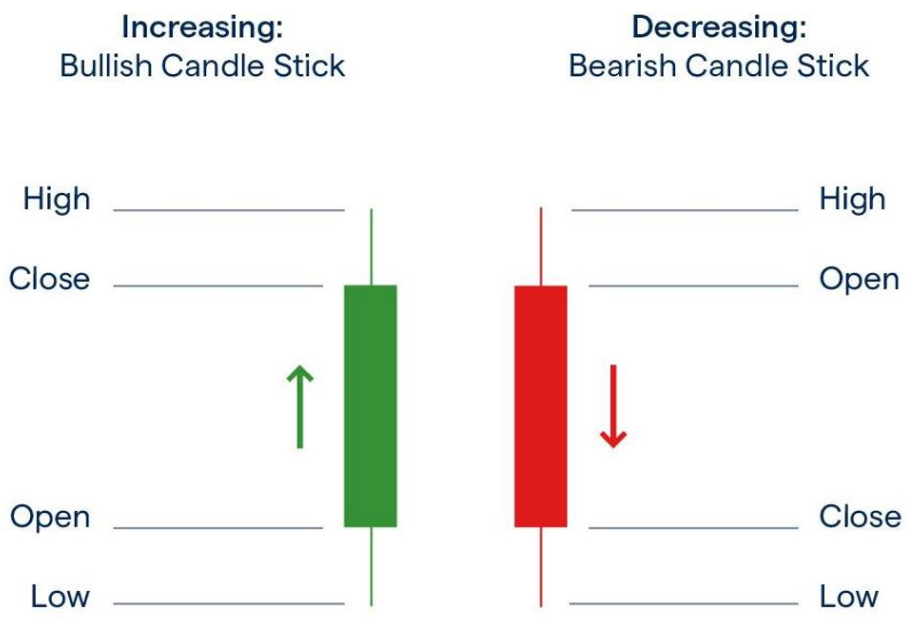


Figure 6.1: Japanese candlestick types.

6.3. Advanced Data Analysis and Volatility

When using supervised learning, it is necessary to mark up the data. It is necessary to find and mark up the set of ticks and patterns of market behaviour that are desirable points for entering the market (for long or short positions). Since this research considers only intraday trading, it is necessary to assess the daily statistical volatility (Section 2.7). Based on the volatility, it will be possible to set trading goals; that is, indicating what sizes of future price movements will be desirable as goals for analyzing and training the neural network.

As a definition of absolute daily volatility, we will use the difference between the maximum daily ask price and the minimum daily bid price (Equation (6.1)).

$$Volatility_{daily} = Ask_{daily}(max) - Bid_{daily}(min) \quad \textbf{(6.1)}$$

Figure 6.2 demonstrates a distribution of the daily volatility of the Forex EUR/USD for 2018 and 2019. It is interesting to note that the volatility in 2019 was noticeably lower than in 2018. Thus, 2018 was more potentially profitable for intraday trading then 2019.

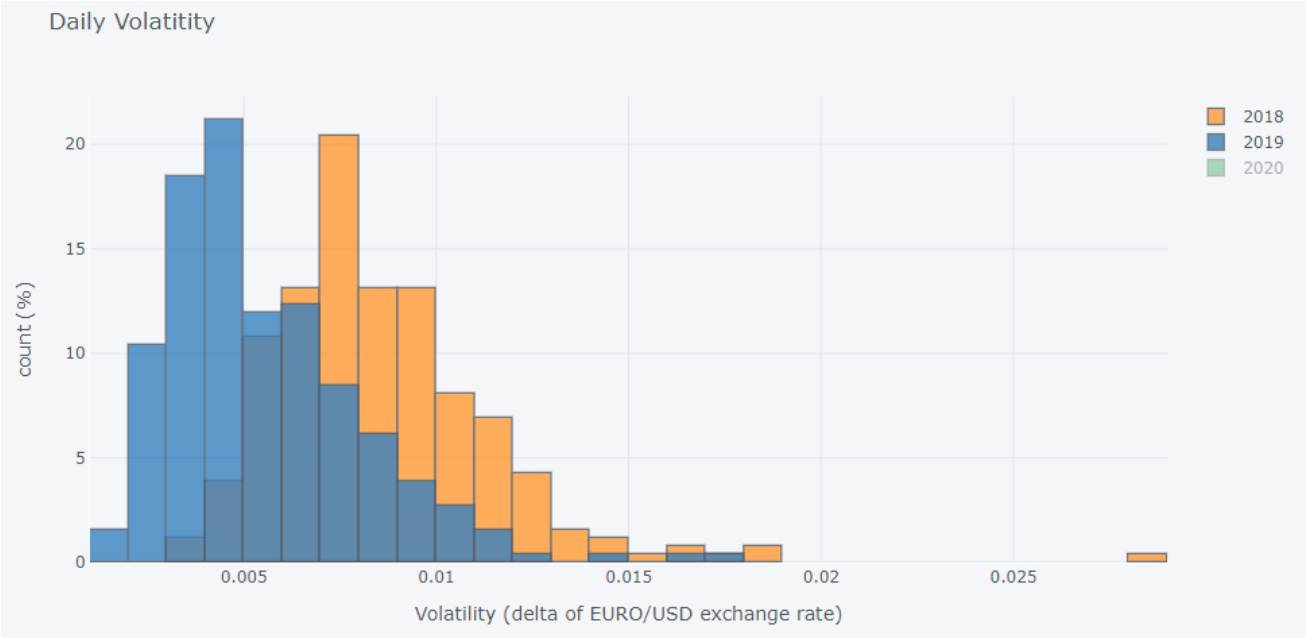


Figure 6.2: Daily volatility of the Forex EUR/USD for 2018 and 2019.

Figure 6.3 added daily volatility for the first six months of 2020, which showed increased volatility due to the COVID-19 pandemic.

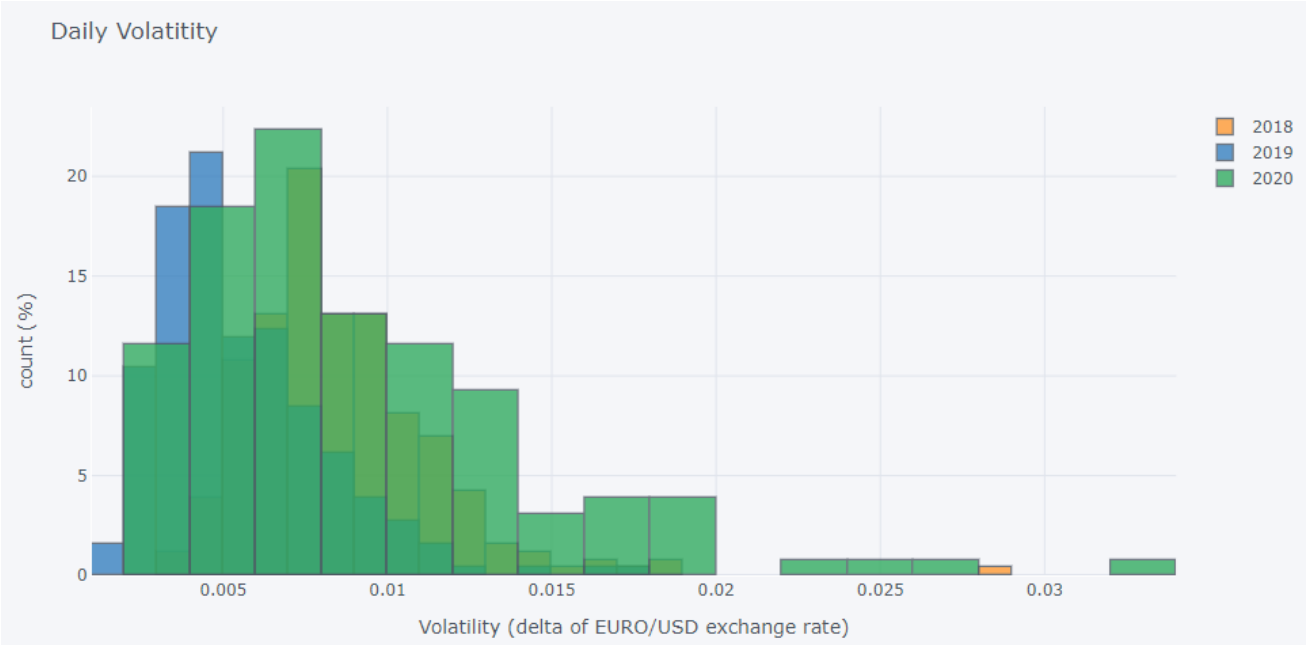


Figure 6.3: Daily volatility of the Forex EUR/USD from 2018 till 2020.

Figure 6.3 also demonstrates that the maximum daily volatility of the Forex EUR/USD varies between 0.018 and 0.035 delta of the EUR/USD exchange rate, depending on the year. It defines the upper target limit for intraday trading.

Due to the margin between the bid and ask prices, as well as the brokerage fees for each trade, movement goals less than 0.001 delta of the EUR/USD exchange rate are virtually unprofitable. It is one of the reasons why deals in the FOREX market are often unprofitable for clients, as it is a ‘negative-sum game’ due to various commissions. The 'negative-sum game' is meant as the opposite concept of a 'zero-sum game' (or antagonistic game) from game theory when the total gain of all players is zero (Neumann and Morgenstern, 2020).

This simple reasoning and calculation define the minimum price goal that needs to be tagged as supervised learning output. Since when entering the market, there are only two possible movement directions: up and down (buying or short selling). The theoretical probability of randomly predicting the market movement coincides with the theoretical probability of a coin toss and is equal to 50%. With this overview, the minimum target for taking profit should be at least twice as high as the stop loss to cover the possible loss of mispredictions. Thus, the minimum price goal that should be labelled in the data as supervised learning output is 0.002 of the delta of the EUR/USD exchange rate.

In order to preliminarily estimate the probability of guessing the minimum target of the movement, it is necessary to analyse the absolute hourly volatility within each trading day, which can be calculated by Equation (6.2):

$$Volatility_{hourly} = Ask_{hourly}(max) - Bid_{hourly}(min) \quad \textbf{(6.2)}$$

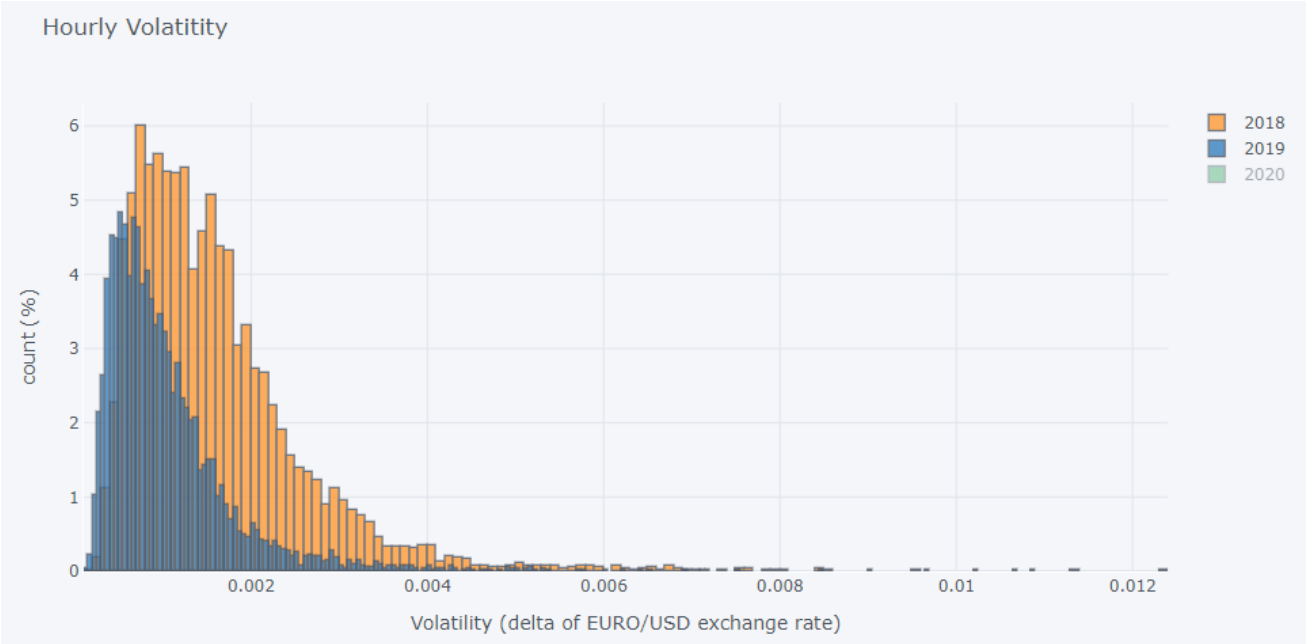


Figure 6.4: Hourly volatility of the Forex EUR/USD for 2018 and 2019.

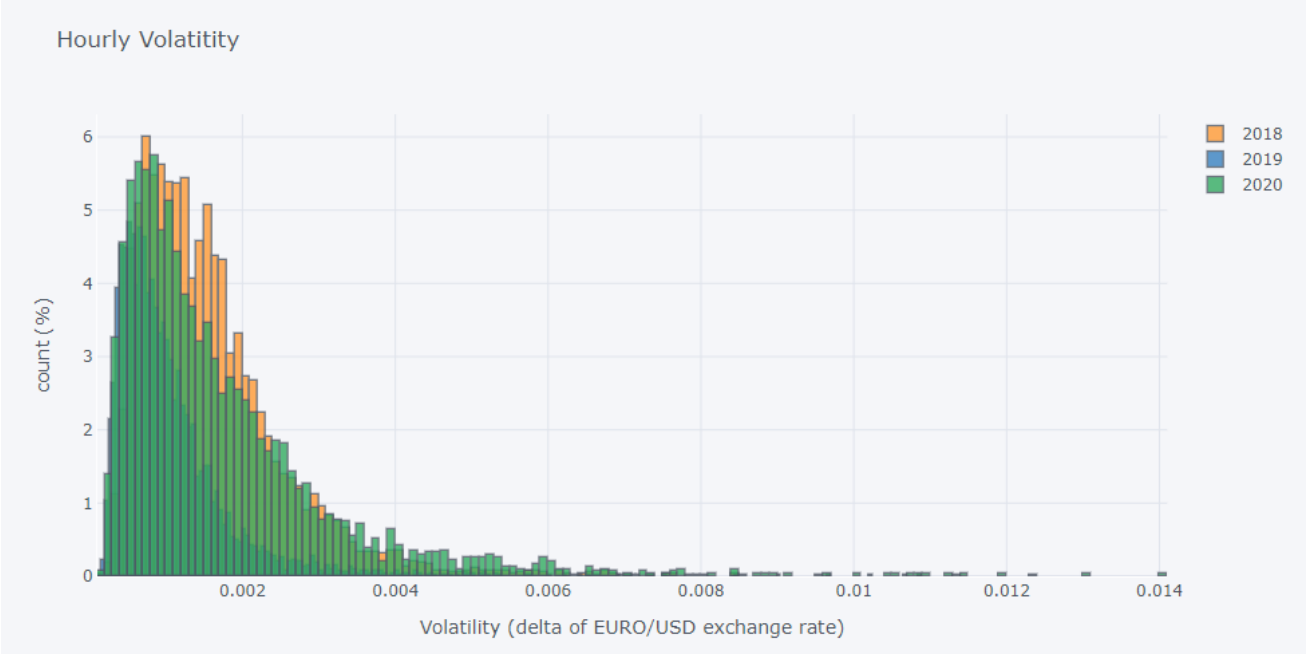


Figure 6.5: Hourly volatility of the Forex EUR/USD from 2018 to 2020.

Figure 6.4 and Figure 6.5 show the distribution of the hourly volatility for 2018-2019 and 2018-2020, respectively. An interesting fact is that hourly volatility (Figure 6.5) is not very high in 2020 (COVID-19 pandemic time), unlike daily volatility (Figure 6.3).

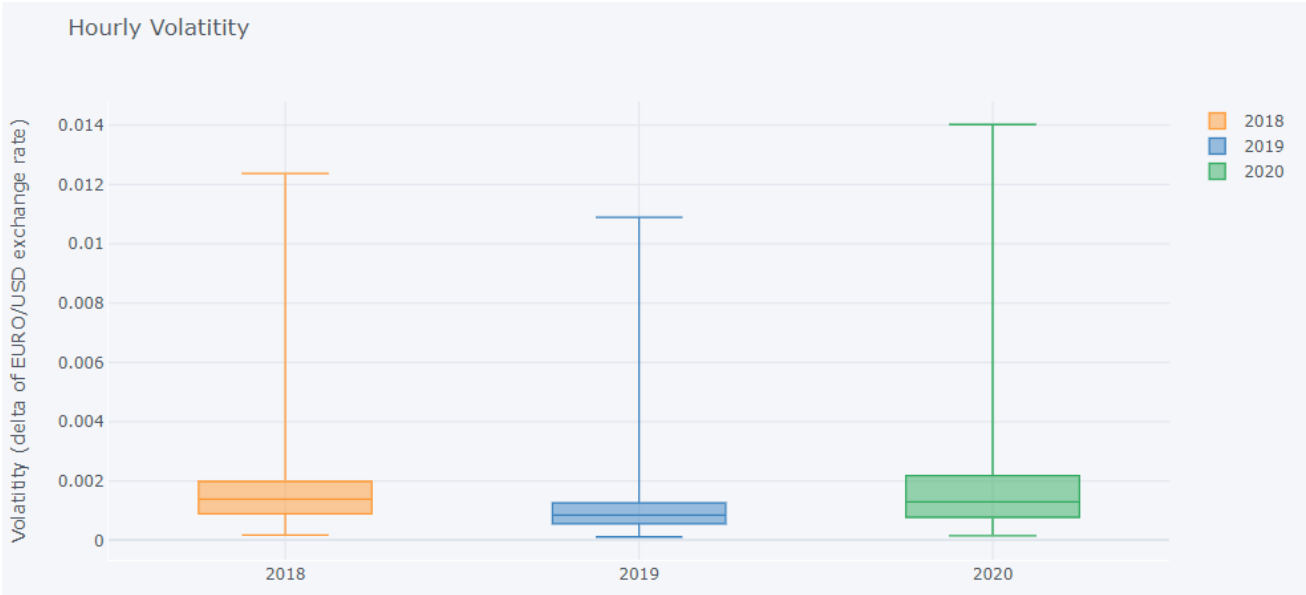


Figure 6.6: Boxplot of the hourly volatility of the Forex EUR/USD from 2018 to 2020.

Figure 6.6 shows the distribution of the same hourly volatilities in the Boxplot. It can be seen that volatility above 0.002, which is the target of the subsequent neural network training, is above the upper quartile. Thus, on average for 2018 and 2020, intraday markets are in motion

no more than 25% of time (above the upper quartile). The rest of the time the price is in flat. For 2019, the flat periods are even longer. Intraday trading in a flat is likely to result in losses. Therefore, it is necessary to recognise the flat periods and avoid trading during this time.

Comparison of the maximum values of volatility in Figure 6.6 and Figure 6.7 confirms that many significant movements (up to 0.012) are achieved for 30 minutes. It also quantitatively confirms the stepped-pattern model of the market movement, which was qualitatively detailed in Section 3.4.4.

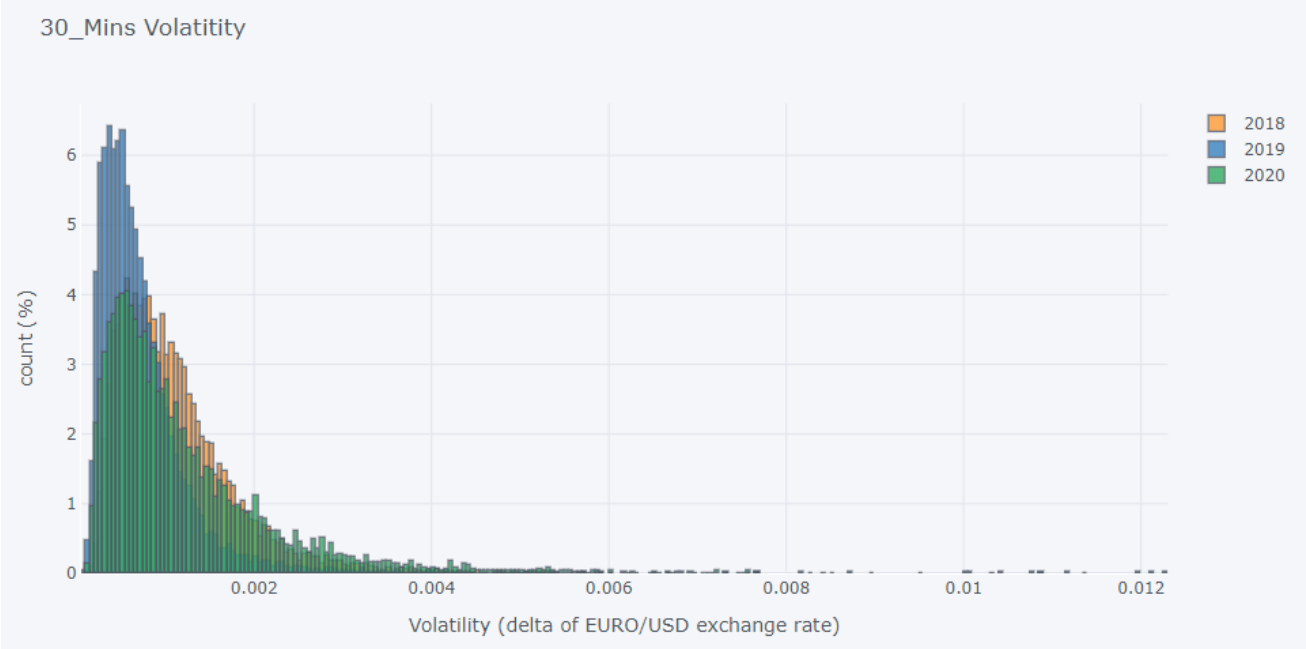


Figure 6.7: 30-minute volatility of the Forex EUR/USD from 2018 to 2020.

6.4. Fractal Structure of Financial Markets and a Fractal Comparison of Volatility.

Since the markets have a fractal structure (Ku, et al., 2020), the transition to a shorter-term trade increases the total length of the fractal line.

Figure 6.8 demonstrates a comparison of the lengths of the fractal lines of the volatility for 10-minutes and 30-minutes time frames. It is evident that the total length of volatility is longer on short time frames. Thus, the total volatility and potential profitability of trading increase for algorithms focused on shorter time frames. For example, for the first half of 2020 in EUR/USD Forex data, the fractal length (potential profit) for trading based on daily time frames was 29 times less than on one-minute time frames, or 6.6 times less than a five-minute time frame. Figure 6.9 demonstrates comparisons of total volatilities at different time frames in absolute

value and percentages. Thus, potential targets in trading on a daily time frame are only 3.4% of one-minute time frame trading. However, it is impossible to infinitely decrease the time frame, since the margin between bid and ask, as well as brokerage fees for a massive number of transactions, will nullify all potential profit. Thus, there is a reasonable frequency of trades and target volatility.

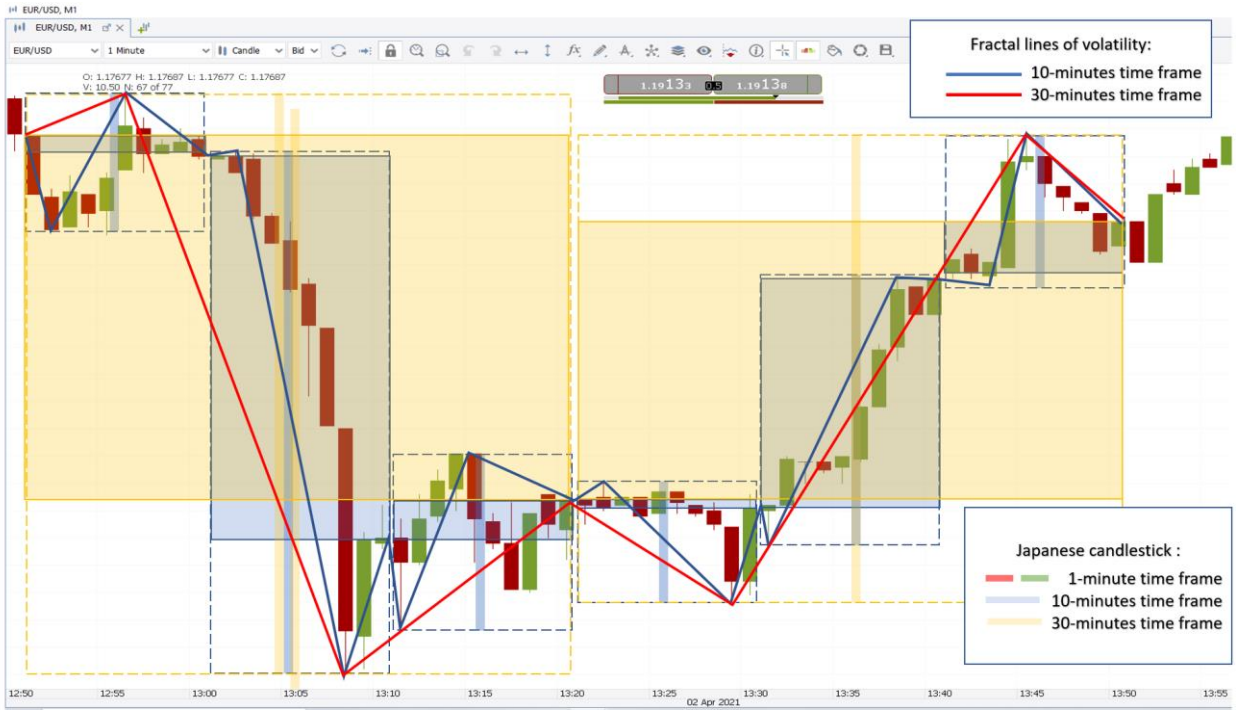


Figure 6.8: Fractal lines of the volatility.

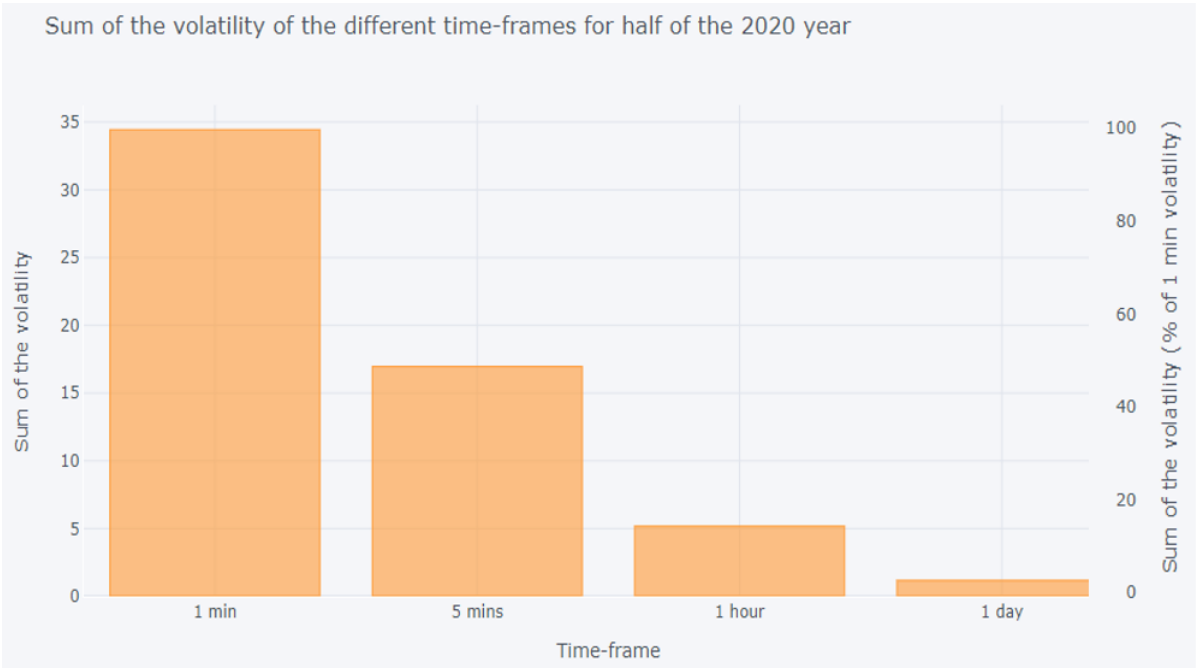


Figure 6.9: Comparisons of total volatilities at different time frames.

The Python language code for analysing volatility and determining future targets for recoilless movement is given in Appendix XII.

6.5. Probabilities Vector

The target volatility discussed in Section 6.4 is essential for constructing the vector of probabilities. The detailed description of the process of creating the volatility vector is also described in detail in (Milke, et al., 2020) and was orally reported in the artificial intelligence section at the 12th International Conference on Agents and Artificial Intelligence (ICAART-2020). The interval (depth) of a possible short-term price movement in both directions was divided into 16 irregular intervals (Figure 6.10), which are described below.

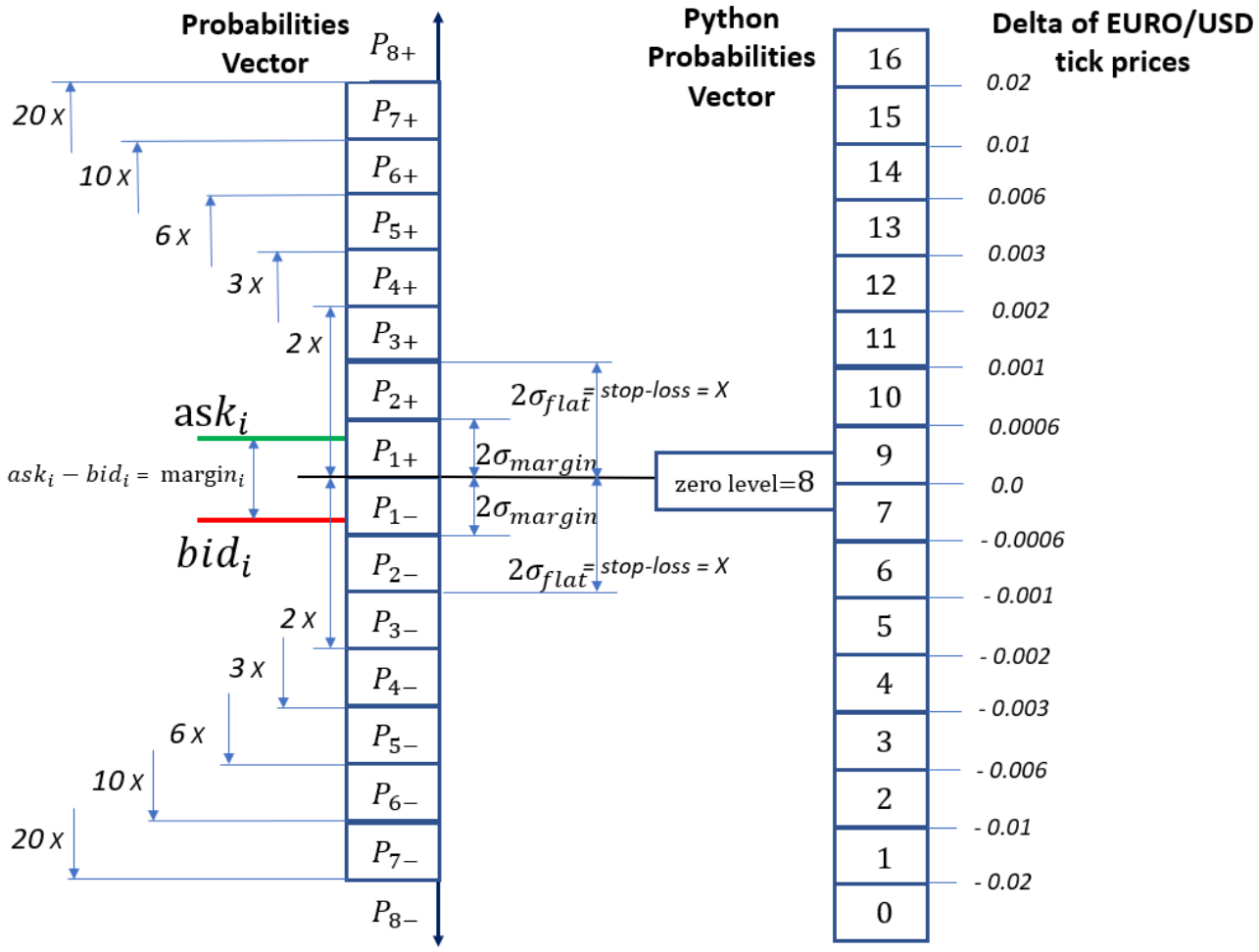


Figure 6.10: Probabilities vector.

The division is based on the statistical distribution of the intraday margin (the difference between ask and bid prices), as well as on the stochastic distribution of flat fluctuations, which are the most likely cause of stop-losses. Each interval has the probability of price movement from the current tick, which has ask_i and bid_i prices. The price movements may go up or down, which correspond to P_+ and to P_- probabilities. Also, for the convenience of Python programming, the whole probability interval is divided into a vector represented by an indexed array with a range [0, 16] (Figure 6.10), which included additional zero level for initial labeling. This dividing will be used in the graphs and charts below.

Figure 6.11 demonstrates, on a randomly selected set of 10,000 lines, that the margin (difference between ask_i and bid_i) has a similar statistical distribution to normal (Gaussian) distribution. Due to the difference between the bid and ask prices, when buying a financial instrument at the ask price, it must rise upper the ask level to be able to sell the position without a loss. Thus, to avoid losses at the time of entering the market, it is necessary to exclude from the targets all flat movements that are less than 0.00006 of the delta of EUR/USD tick prices, which approximately corresponds to a margin two standard deviations. Precisely the same mirror-image situation with short selling. Therefore, the splitting of the probability vector to P_{1+} and P_{1-} sectors is also associated with standard deviations of these distributions.

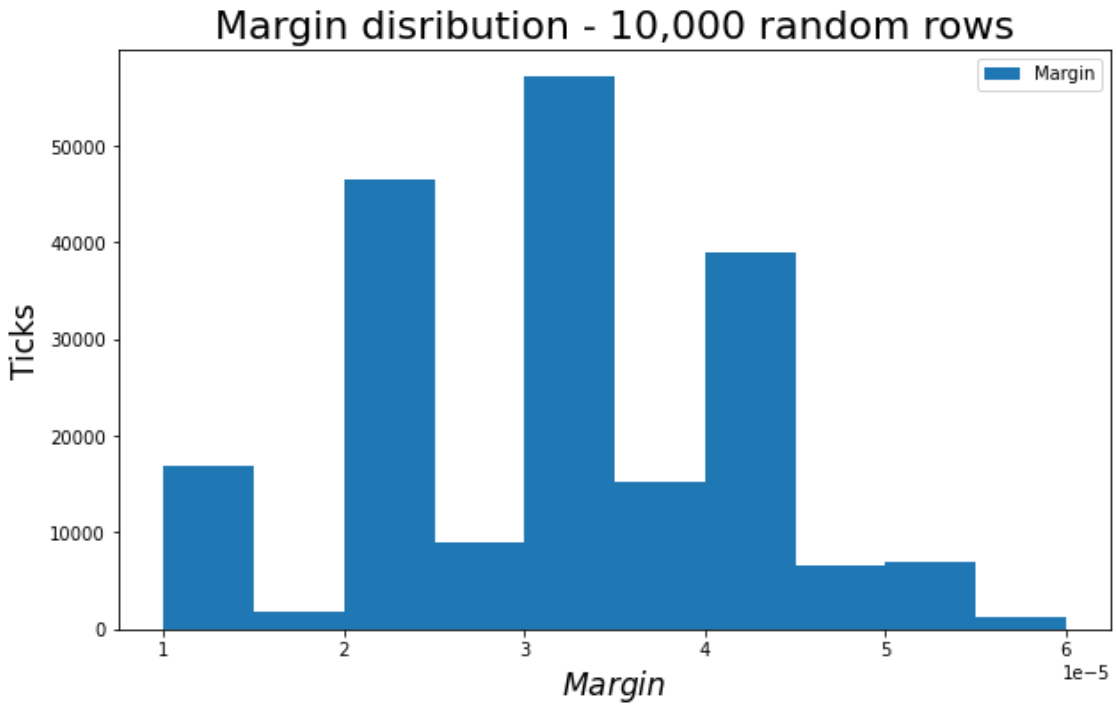


Figure 6.11: Margin distribution.

The flat fluctuations are the most likely cause of stop-losses. For intraday trading, it is vital to reduce the number of stop-losses (Alves, Caarls and Lima, 2018). For indicating the minimum stop-loss (as described in Section 6.3), P_{2+} and P_{2-} boundaries are created, which are also approximately equal to two standard deviations of the flat movement (fluctuation) distributions. Thereby, a reduction in losses in actual trading is achieved.

An additional default field is introduced in the middle of the volatility vector. It also fits all ticks, after which the minimum stop-loss has not been exceeded within ten subsequent ticks. This means that the financial market is not yet ready to move, and this tick does not need to be considered. As a consequence of this approach, a reduction in the size of the data for training and validation is also achieved.

The P_{3+} and P_{3-} boundaries are determined by the minimum target of the expected market movements that the CNN model will look for, which must be at least twice the stop-loss (as described in Section 6.3).

The boundaries of P_{7+} and P_{7-} probability intervals are determined from the maximum average daily volatility (Section 6.3). Since the purpose of this research is intraday trading, precise separation of greater volatility is not necessary; however, probability intervals P_{8+} and P_{8-} are provided for statistical outliers of volatility.

Splitting of the other parts of the probability vector is implemented as the stop-loss level is multiplied by 3, 6, and 10. The multipliers have been chosen to ensure a similar probability of achieving these levels of delta prices. Figure 6.13 demonstrates the initial distribution of the number of values in the probability vector using the example of the first half of 2020. This period was chosen for demonstration because there was increased medium-term volatility of financial markets associated with the COVID-19 pandemic, which is demonstrated by a large daily number of ticks (settlements) in February and March 2020 (Figure 6.12).

Despite the increased medium-term volatility of financial markets associated with the COVID-19 pandemic, the overwhelming majority of the periods the market was flat (zero-level of this probability vector of short-term movement = 8), confirming the concept of stepped model of short-term movement of financial markets (Section 3.4.4). Thus, all significant movements took place quickly and in a short time frame.

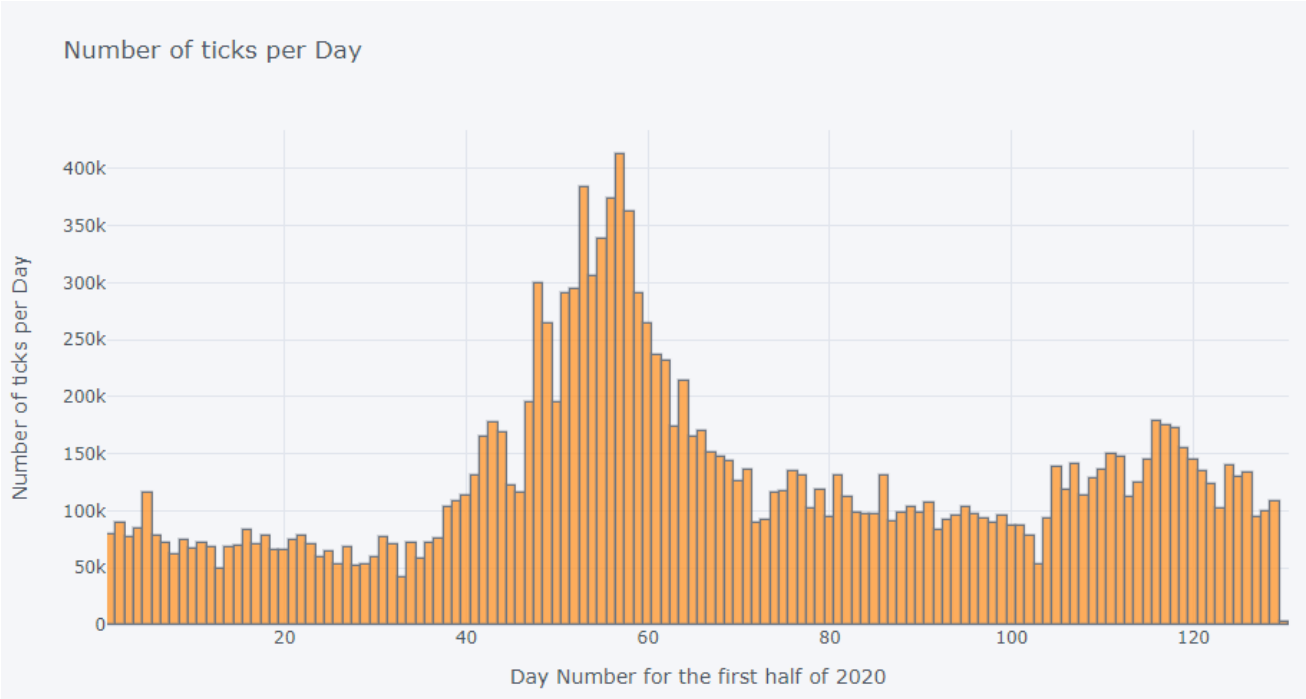


Figure 6.12: Daily number of ticks (settlements) in March 2020 (working days).

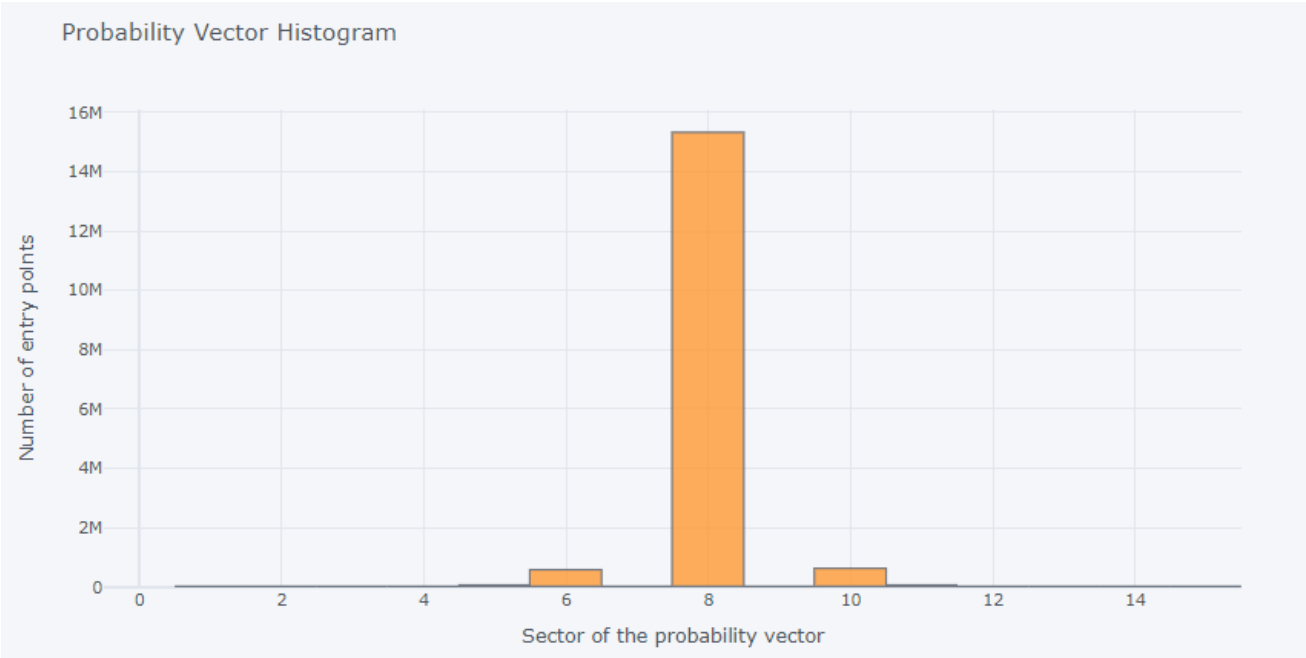


Figure 6.13: Initial distribution of the number of values of this probability vector.

The concept of the above-described probability vector is the central novelty in the second series of experiments. It makes it possible to move from a regression problem to a classification problem, which will allow using other error metrics and a more comprehensive set of machine learning tools.

6.6. Reduction of Calculations

Since GPU processor power and free cloud GPU resources, such as Google Colab, are limited, and the number of analysed tick data (rows) is tens of millions, it is necessary to reduce the number of calculations. For achieving this goal, an additional variable (L) was introduced equal to the number of ticks before the significant movement of the financial market began. If the price does not go beyond the opposite stop loss during the L-ticks after the considered time, then the current tick is considered uninteresting, and the algorithm goes to the next tick. Figure 6.14 shows the distribution of values of this probability vector after removing all non-significant movements (field N-8 of the probability vector) during the next ten ticks for the first half of 2020.

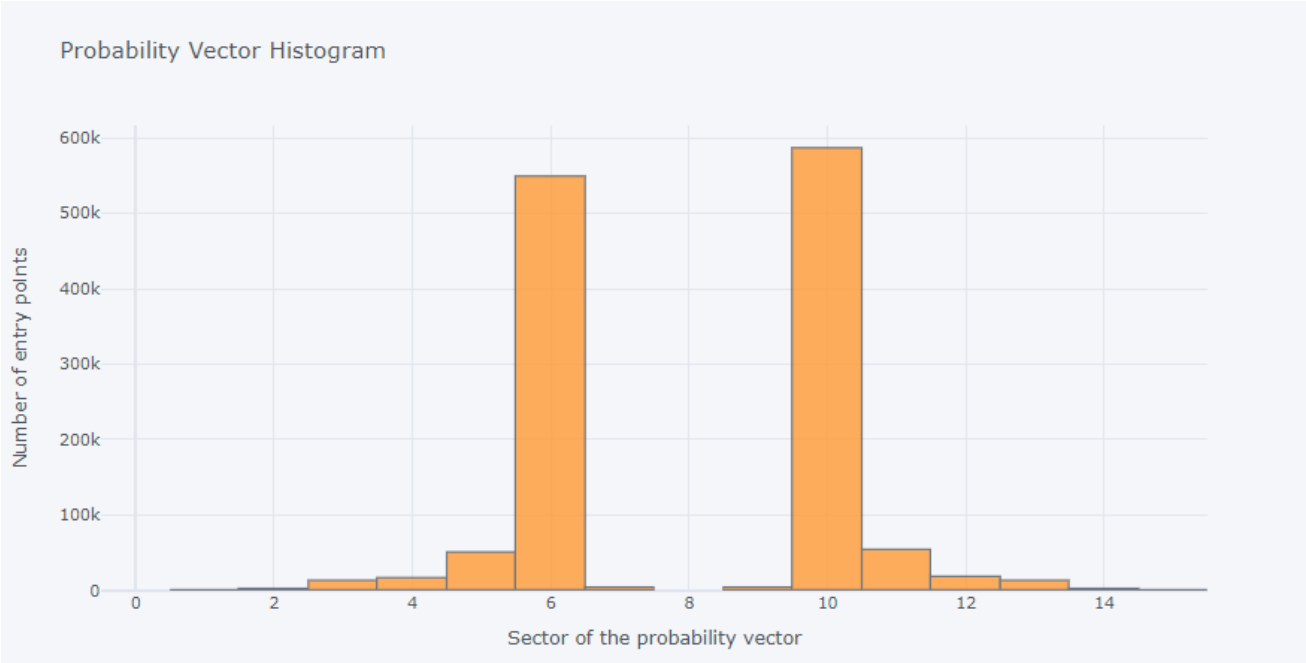


Figure 6.14: Distribution of the number of values of this probability vector without the zero level.

As mentioned above, the stop-loss size was calculated based on the standard deviation of the flat fluctuation distribution. Consequently, a significant size of the useless flat fluctuation was excluded from the training dataset for the neural network. Through a preliminary iteration analysis, the most relevant value L=10 was chosen. On one hand, this allowed the reduction of the size of training and validation datasets significantly. On the other hand, do not overlook the most significant number of long recoilless movements in the training dataset.

Table 6.2 and Figure 6.15 demonstrate this probability vector distribution with only significant financial market movements that are larger than the size of the double stop loss.

Sector of the Probability Vector	Number of rows (points)
0	7
1	250
2	1 213
3	11 765
4	15 949
12	17 373
13	11 899
14	1 314
15	364
Total:	**60 134**

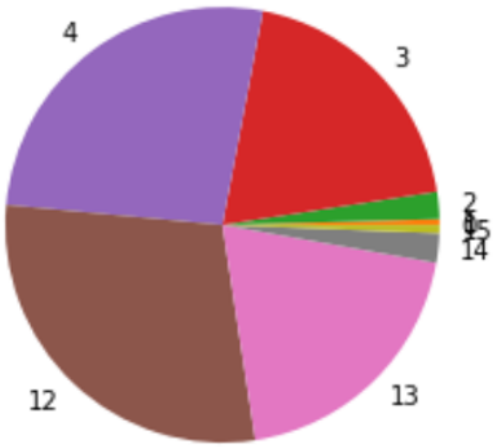


Table 6.2: The probability vector with only significant financial market movements.

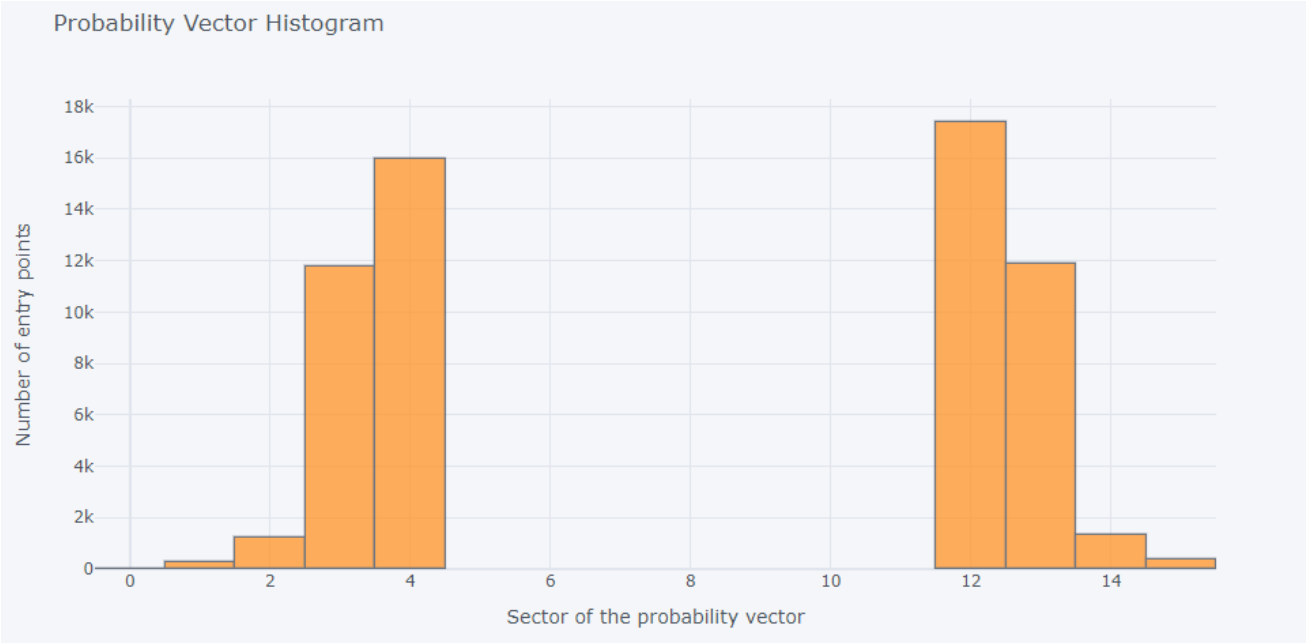


Figure 6.15: Distribution of this probability vector with only significant financial market movements.

Using the example of the most volatile period over the past three years, namely the first half of 2020, this analysis shows that the applied novelty method of reducing the amount of data for the subsequent feeding of the convolutional neural network allows the reduction of the

amount of data by 275 times (from 16.6 million to 60.1 thousand rows). Data with several thousand or tens of thousands of lines is the most convenient size for training neural networks. For other intervals during 2018, 2019, and 2020, the above data reduction ratios are similar. The author understands that entering an additional dropout parameter can also remove some interesting movement data, especially those that start slowly. However, for intraday trading, it is not so essential to catch all the movements. It is more important to recognise the movements with the highest probability of continuation and to not take a stop loss.

6.7. Prediction of the Start of a Significant Movement in Forex EUR/USD Rates by the CNNs

6.7.1. Data Used

As a result of the above analysis, labelling and data reduction, as well as the compilation of the vector of probabilities, tick data was obtained, each line of which preceded a significant movement of the financial market. Figure 6.16 shows the price chart after one of these ticks. It is essential that none of these movements contain significant retracements of the market back, which would lead to the triggering of the stop-loss. Thus, these are the most valuable entry points into the market, including for trading with using financial leverage. The second series of experiments, described below, train a convolutional neural network to recognise similar market entry points using supervised learning.

Most of the time, financial markets are flat but demonstrate an oscillatory (retreating) movement with a small amplitude. For this reason, stop loss often occurs during intraday trading. This feature of financial markets is often overlooked when neural networks solve the regression task of predicting future prices. The probability categorization task described in this chapter has advantages over the indicated regression task since only recoilless movements were taken into account when forming the probability vector. Rollbacks were considered movements which did not exceed half of the main movements.

The data collection for supervised learning was done on a complete three-year dataset. The supervisory signals of the outputs vector of probabilities were formed based on the recoilless movement before the end of the trading day when any open position should be closed. More than 460 hours of net processor time was spent to form this vector of probabilities and reduce

the data's size on the hardware specified in Section 4.3.5. However, if preparing data for training neural networks is done regularly at the end of each week, then this will require about three hours of net processor time, which is acceptable.

Figure 6.16: One of the most valuable points of entry into the market without significant retracements.

This series of experiments uses the same raw financial big data, which was used in the first series of experiments (Sections 4.5.2 and 4.8). Similar 3D tensor and sliding window techniques and procedures, and time format conversions, data cleansing and parameter normalisations are also used (as described in Sections 4.8.1, 4.8.2, 4.8.3 and 5.1).

6.7.2. CNN Architectures

This series of experiments used two CNN architectures with eleven layers (Algorithm 6.1) and thirteen layers (Algorithm 6.2) which contains two additional convolutional layers. The first one contains over 1.5 million parameters, and the second one contains over 4.5 million training parameters for each experiment within this series of experiments. These CNNs use a 15-length one-dimensional convolutional kernel; kernels of length 5 and 9 have also been experimented with. This convolutional kernel was chosen to be long enough to increase the correlation with previous ticks.

```
model = keras.Sequential([
    keras.layers.Conv1D(200, 15, padding='same', input_shape=(200, 5), activation=tf.nn.relu, kernel_initializer="normal"),
    keras.layers.MaxPooling1D(5),
    keras.layers.Dropout(0.25),
    keras.layers.Conv1D(400, 15, padding='same', activation=tf.nn.relu, kernel_initializer="normal"),
    keras.layers.GlobalMaxPooling1D(),
    keras.layers.Dropout(0.25),
    keras.layers.Flatten(),
    keras.layers.Dense(720, activation=tf.nn.relu, kernel_initializer="normal"),
    keras.layers.Dense(72, activation=tf.nn.relu, kernel_initializer="normal"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(1, activation=tf.nn.sigmoid)
])
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_2 (Conv1D)            (None, 200, 200)          15200
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 40, 200)           0
_________________________________________________________________
dropout_3 (Dropout)          (None, 40, 200)           0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 40, 400)           1200400
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 400)               0
_________________________________________________________________
dropout_4 (Dropout)          (None, 400)               0
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_3 (Dense)              (None, 720)               288720
_________________________________________________________________
dense_4 (Dense)              (None, 72)                51912
_________________________________________________________________
dropout_5 (Dropout)          (None, 72)                0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 73
=================================================================
Total params: 1,556,305
Trainable params: 1,556,305
Non-trainable params: 0
_________________________________________________________________
```

Algorithm 6.1: Eleven layer CNN for the probability vector analysis.

These CNNs include both one-dimensional convolutional and fully connected layers for the output classification. For preventing overfitting, ‘dropout layers’ are used, which are described in Section 4.6.1. The ‘Flatten layer’ is used to reformat two-dimensional tables after convolution procedures into a one-dimensional vector for subsequent classification on fully connected layers. Most of these experiments were calculated on the CNN with eleven layers

since the deeper architecture of the neural network did not significantly increase the quality of training but required 2.3 times more time for training: an average of 230 seconds versus 53 seconds for each layer.

```
model = keras.Sequential([
    keras.layers.Conv1D(200, 15, padding='same', input_shape=(200, 5), activation=tf.nn.relu, kernel_initializer="normal"),
    keras.layers.Conv1D(200, 15, padding='same', activation=tf.nn.relu, kernel_initializer="normal"),
    keras.layers.MaxPooling1D(5),
    keras.layers.Dropout(0.25),
    keras.layers.Conv1D(400, 15, padding='same', activation=tf.nn.relu, kernel_initializer="normal"),
    keras.layers.Conv1D(400, 15, padding='same', activation=tf.nn.relu, kernel_initializer="normal"),
    keras.layers.GlobalMaxPooling1D(),
    keras.layers.Dropout(0.25),
    keras.layers.Flatten(),
    keras.layers.Dense(720, activation=tf.nn.relu, kernel_initializer="normal"),
    keras.layers.Dense(72, activation=tf.nn.relu, kernel_initializer="normal"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(1, activation=tf.nn.sigmoid)
])
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_14 (Conv1D)           (None, 200, 200)          15200
_________________________________________________________________
conv1d_15 (Conv1D)           (None, 200, 200)          600200
_________________________________________________________________
max_pooling1d_6 (MaxPooling1 (None, 40, 200)           0
_________________________________________________________________
dropout_18 (Dropout)         (None, 40, 200)           0
_________________________________________________________________
conv1d_16 (Conv1D)           (None, 40, 400)           1200400
_________________________________________________________________
conv1d_17 (Conv1D)           (None, 40, 400)           2400400
_________________________________________________________________
global_max_pooling1d_6 (Glob (None, 400)               0
_________________________________________________________________
dropout_19 (Dropout)         (None, 400)               0
_________________________________________________________________
flatten_6 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_18 (Dense)             (None, 720)               288720
_________________________________________________________________
dense_19 (Dense)             (None, 72)                51912
_________________________________________________________________
dropout_20 (Dropout)         (None, 72)                0
_________________________________________________________________
dense_20 (Dense)             (None, 1)                 73
=================================================================
Total params: 4,556,905
Trainable params: 4,556,905
Non-trainable params: 0
```

Algorithm 6.2: Thirteen layer CNN for the probability vector analysis.

6.7.3. Dynamic Boundaries and Splitting of the Probability Vector

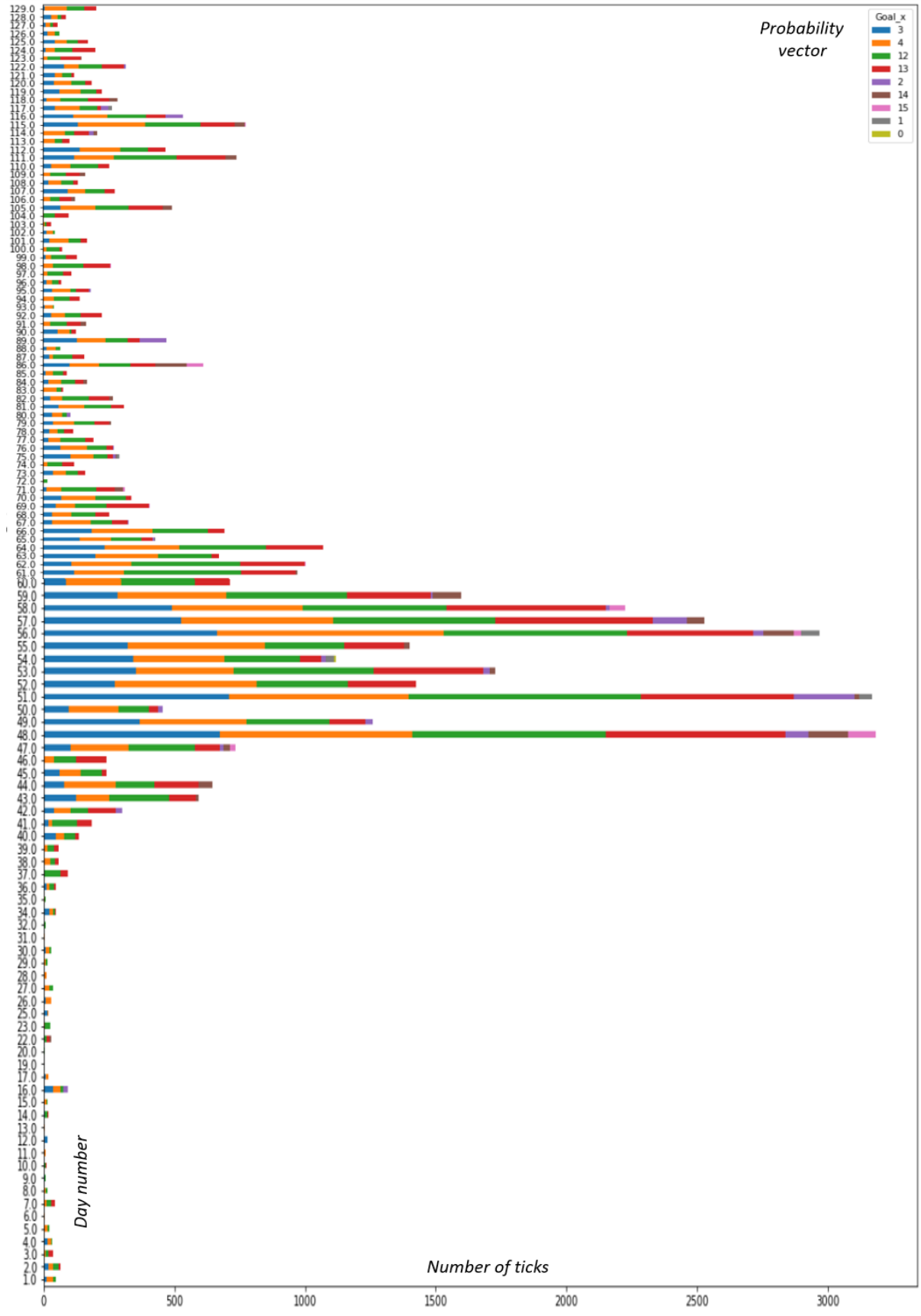


Figure 6.17: Distribution of the probability vector over on each day for the first half of 2020.

Using the example of the first half of 2020, Figure 6.17 shows the distribution of the probability vector over all ticks on each day. Only market trend entry points are shown, without flat ticks. A concentration of profitable entry points into the market can be seen during the start of the COVID-19 pandemic. Following this analysis, it is possible to determine the most favourable days for intraday trading with the most significant number of good entry points into the market and the depth of the expected movement. Thus, the task of determining the probability vector using the CNN is to predict the nature of the next bar on this chart.

After reducing the amount of data for analysis, flat ticks are randomly added to training and testing datasets of good market entries. The number of these added ticks for each field of the probability vector is calculated as the arithmetic mean of the number of good entry points into the market. This artificial selection is made to avoid bias in the training of the neural networks since the number of flat ticks is 275 times more (as explained in Section 6.6), than good entry points into the market.

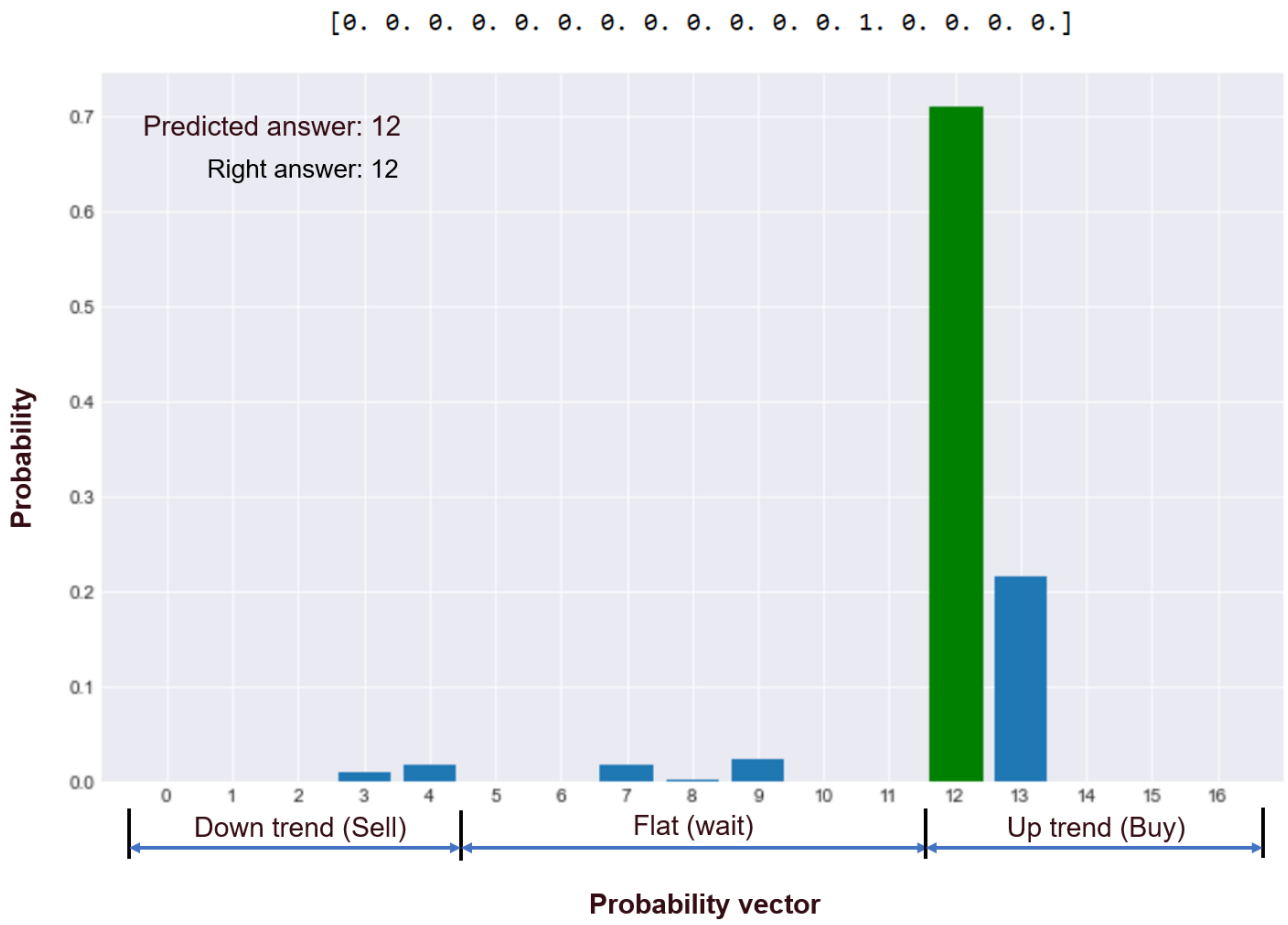


Figure 6.18: A typical probability vector with a correct prediction.

Figure 6.18 demonstrates one of the results of predicting the probability vector at each tick. However, the first tests of predicting the probability vector showed that this neural network determines the directions (Buy or Sell) but mixes neighbouring fields of the probability vector. For example, Figure 6.19 shows a prediction of an uptrend but confuses neighbouring zones 12 and 13, which a summary probability shows is more than 65%. Figure 6.20 and Figure 6.21 demonstrate the same situation with a downtrend and a flat. Thus, the formal distribution of the categorical entropy, used as a loss-function of the CNN, deteriorates. This happens for two reasons:

- due to the almost random creation of the boundaries of the probability vector inside the up (P_{5+} , P_{6+} , P_{7+} , P_{8+}) or down (P_{5-} , P_{6-} , P_{7-} , P_{8-}) trends (Figure 6.10),

- due to the constant change in the value of the probability vector after entering into the financial market.

The author intends to analyse this problem in future research since the dynamic changes of the probability vector boundaries will be required for the calculations at every tick after entering the market. This purpose could only be achieved using a supercomputer.

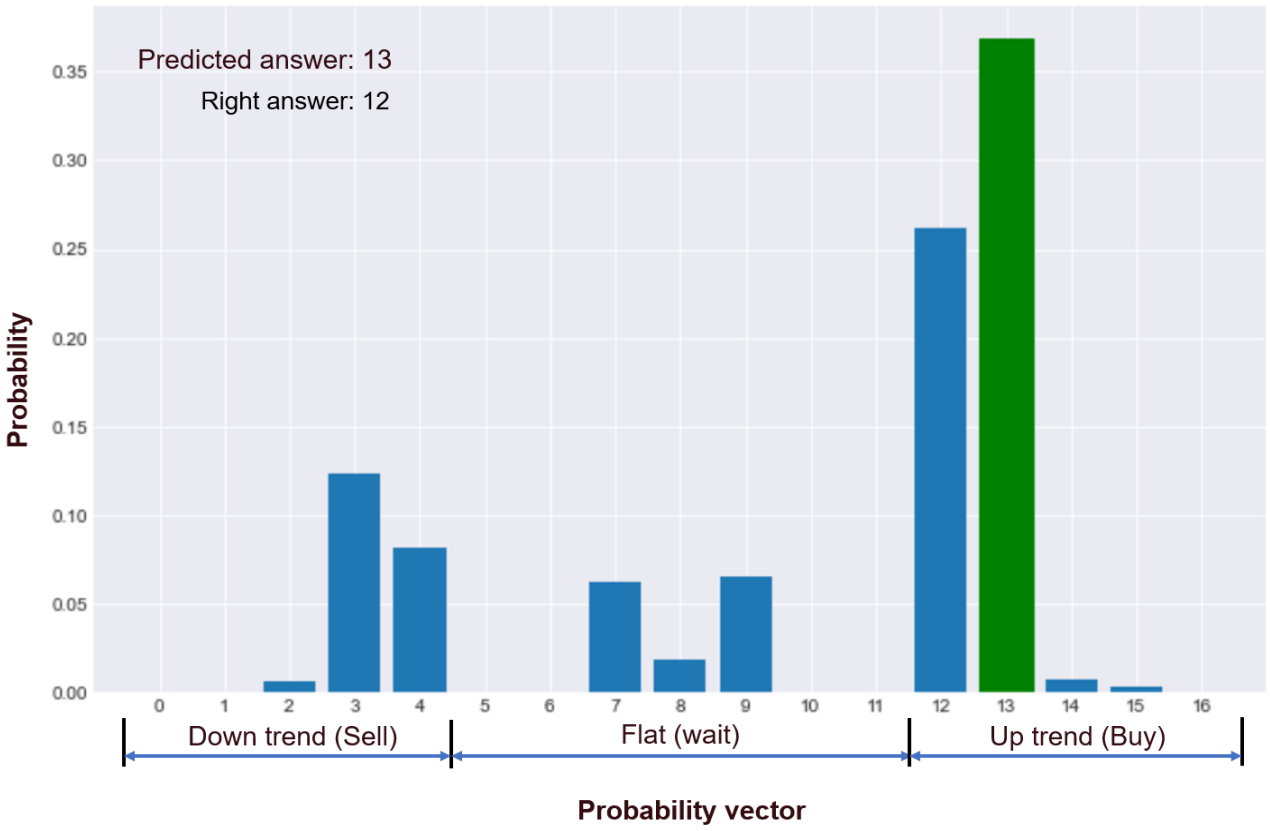


Figure 6.19: A probability vector, which predicted up-trend but mixes neighbouring fields.

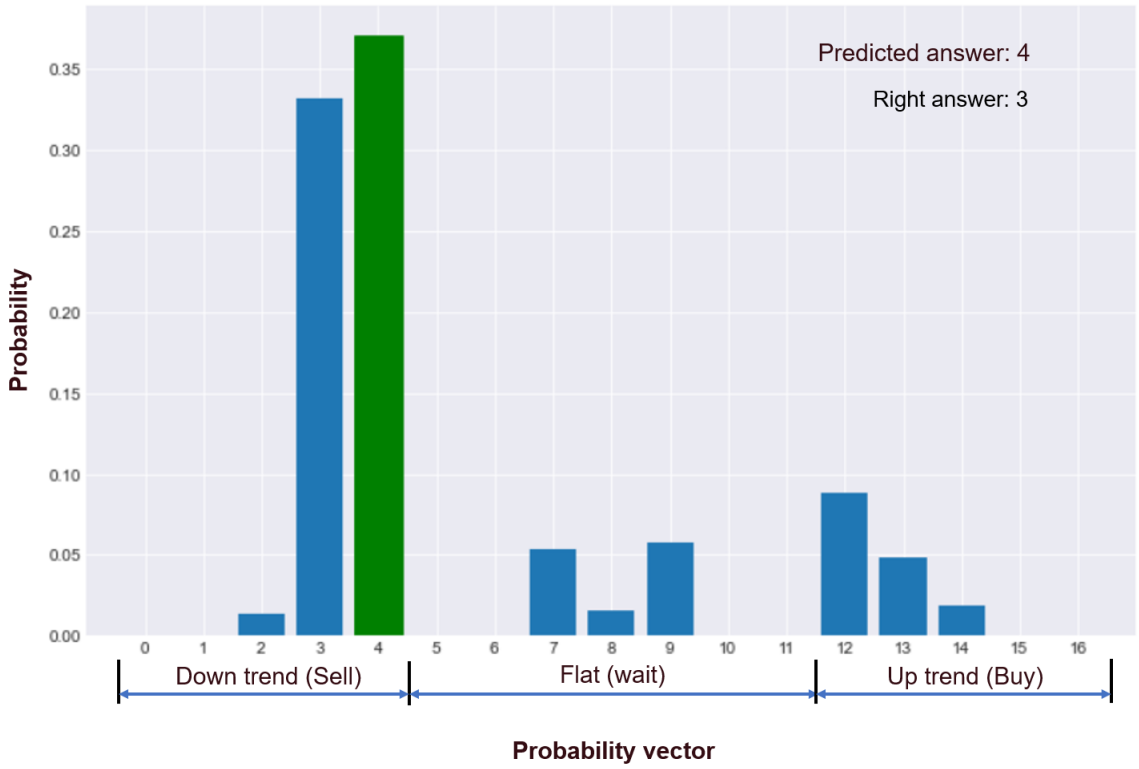


Figure 6.20: A probability vector, which predicted down-trend but mixes neighbouring fields.

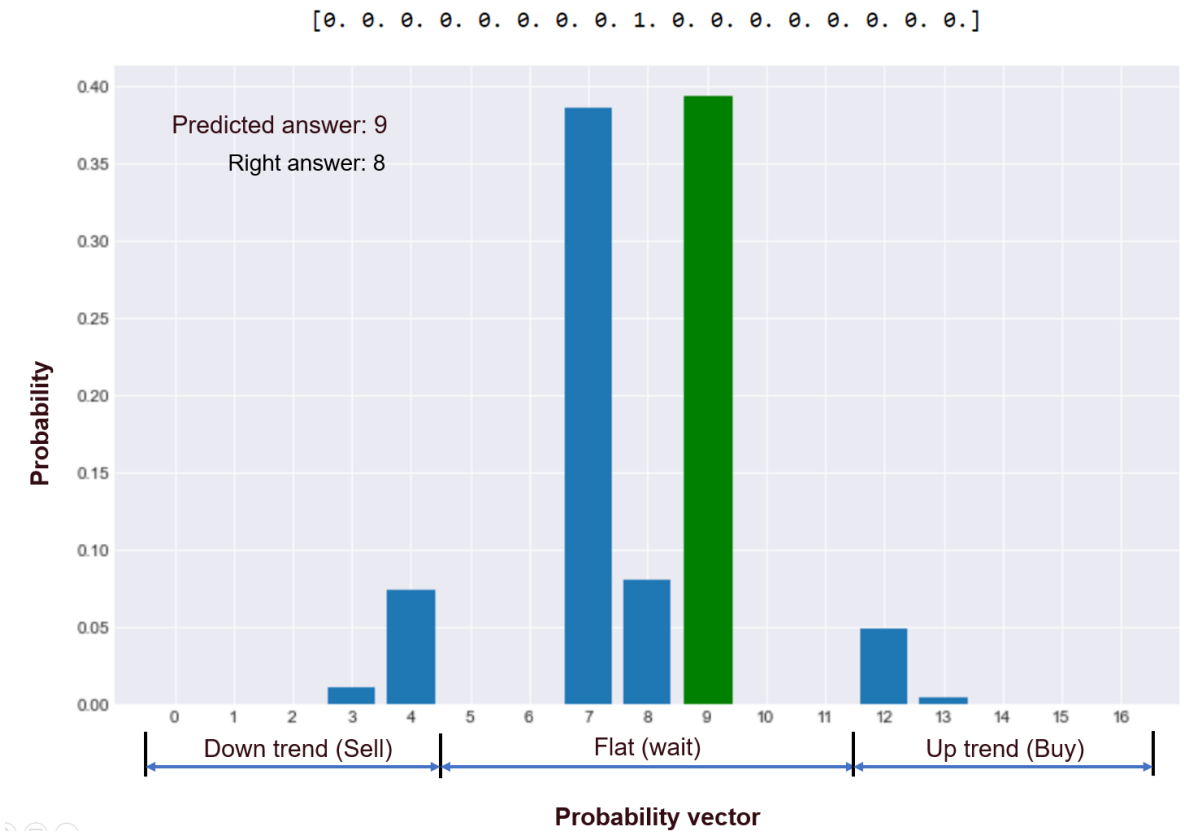


Figure 6.21: A probability vector, which predicted flat but mixes neighbouring fields.

It is possible to simplify and solve the problem using only available processing power by splitting the probability vector and combining essential milestones in decision-making in a sequence, such as determining the start points of a trend and determining the direction of the trend. In other words, a multi-parameter classification task with dynamic boundaries of the probability vector could be split into two sequential binary classifications and an optional third multi-parameter classification with a smaller number of parameters. Figure 6.22 demonstrates this splitting.

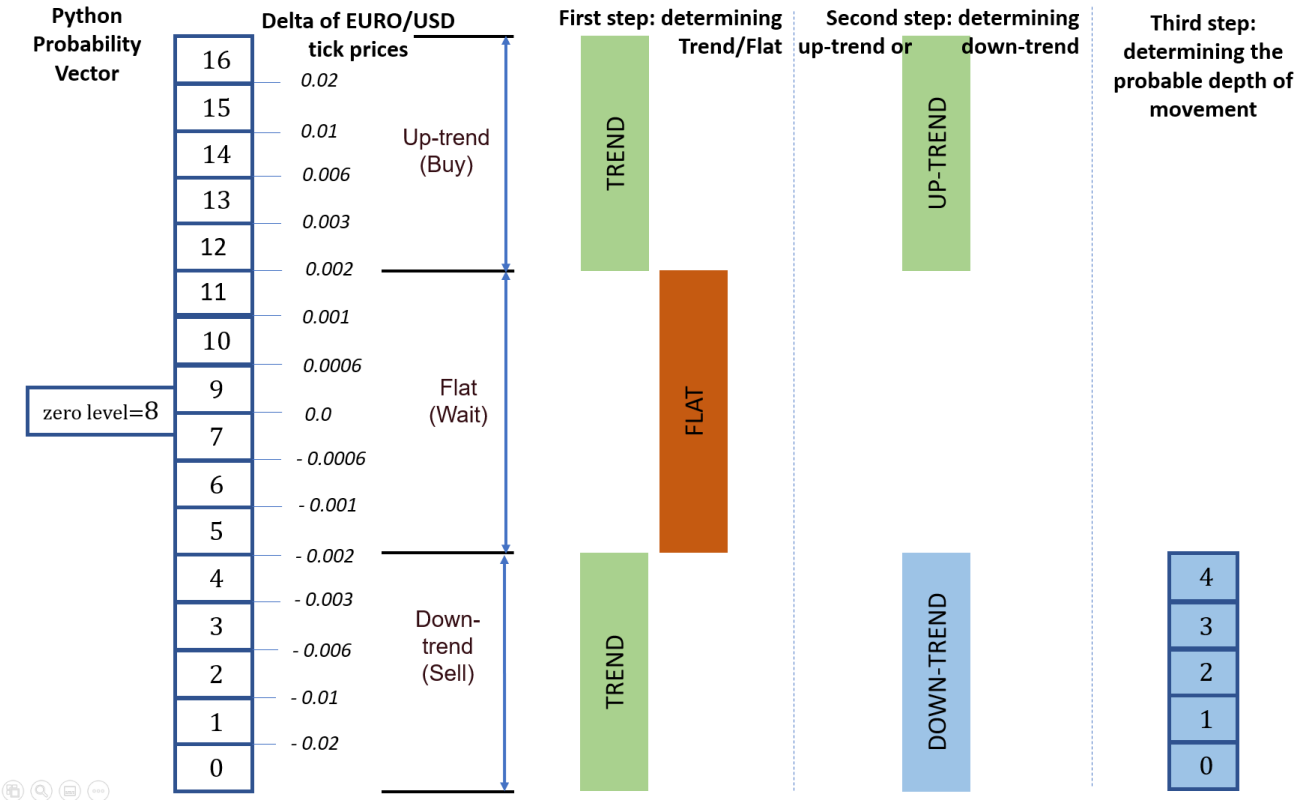


Figure 6.22: Splitting a complex multi-parameter classification task with dynamic bounds of the probability vector into a sequence of simpler tasks.

- Firstly, the classification of the current tick through the probability vector as preceding the trend or located in a flat. If the analysed tick is inside a flat, an ATS has to wait and not enter the financial market. This task is vital for intraday trading, as it avoids losses due to stop-losses. If the current tick is classified as a good entry point into the financial market, the second step should be determining the direction.
- Secondly, the classification of the most probable direction of the proposed movement.
- Thirdly, after determining the direction, it is possible to determine the probable depth of the following movement in that direction, but this is a secondary task for intraday

trading since it could be replaced with a typical trailing stop-loss, which ATSs widely use (Dai, et al., 2020).

Researchers often use neural networks to solve a binary 'Buy or Sell' classification problem (Hilpisch, 2020). This statement of the problem is probably incorrect from a practical view since it involves entering the market in any case. However, most of the time, the financial markets are flat, and the ATSs should wait.

Further, the results of experiments on the first two stages of splitting the probability vector will be presented in detail: using sequential and parallel convolutional neural networks for classification of the current financial market behaviour into trend and flat (Section 6.7.4), and classifications to determine buy and sell signals (Section 7.1).

6.7.4. Binary Classification: (Trend or Flat)

The binary classification experiments on Trend or Flat were carried out with data for 2020. The hyperparameters of the one-dimensional convolutional neural network changed with the size of the convolutional kernel and the addition of two convolutional layers, which did not give significant improvements in the test error metrics. This convolution kernel ranged from 9 to 35 elements, representing a one-dimensional vector. The number of epochs was 100, which provided coverage of the beginning of overfitting. Overfitting is determined based on the analysis of binary cross-entropy as a loss-function and started after 6 to 11 epochs. The time required for each experiment was also measured, and the average time required for each epoch was calculated. One of the variables was the batch size in the direction in which the gradient descent was simultaneously performed. This batch size varied from 50 to 200. The best accuracy was achieved with batch size equals 200, which unexpectedly coincides with the sliding window size (Section 5.1). The training time for each layer and the whole neural network increases with the convolutional kernel size increase. This size is essential for the CNN property to generalise information. In theory, the longer the convolutional kernel is, the further back (within a sliding window of 200 ticks) the CNN 'sees' at each iteration.

The results of the binary classification experiments on Trend or Flat are presented in Appendix II. The summary of these experiments is shown in Table 6.3 and analysed in Section 6.8.1.

Table of the results in Appendix II is divided into blocks, each of which indicates both the synthetic results of training this CNN for 2020 and randomly (using Python random number generator) selected sets of 10 samples of absolute errors in predicting the behaviour of the financial market as a trend and flat. Each of these ten samples was randomly selected from about 20 thousand samples, which are the results of the CNN predictions on the test datasets. In the future, this CNN model can be retrained each weekend using the data for the previous week.

One of the 25 Python codes from these experiments is demonstrated in Appendix XIII. These codes predicting trend or flat periods of Forex EUR/USD are similar to each other but differ in the hyperparameters, training parameters and the results. Each of these codes corresponds to sets of error metrics, such as accuracy, precision and recall.

Thus, the results of these experiments is a dataset of 32 million lines, equal to the number of 2020 ticks, each of which corresponds to a prediction of the market behaviour (Trend or Flat), absolute prediction errors (Figure 6.23) and error metrics.

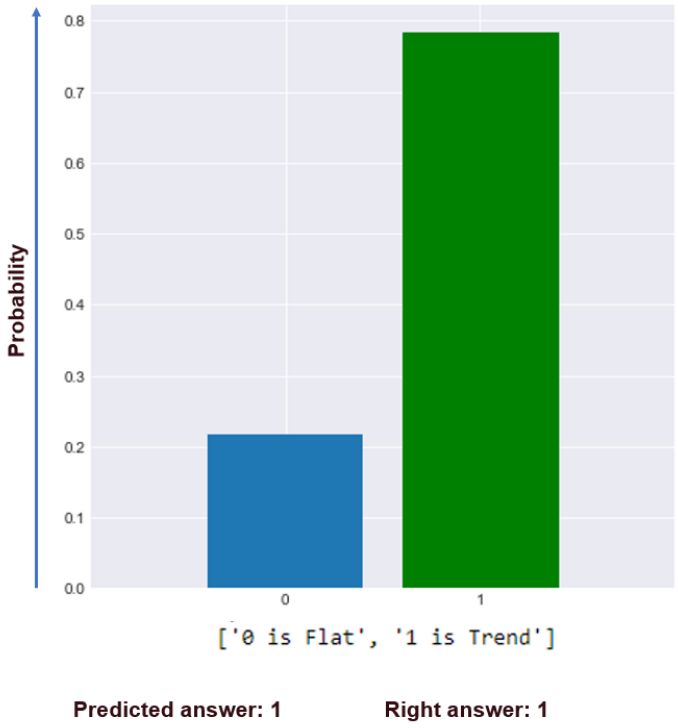


Figure 6.23: Probability (Trend or Flat) and absolute prediction error for each tick.

6.8. Evaluation

6.8.1. Statistical Evaluation

The integrated parameters and error metrics results are of binary classification (Trend or Flat) shown in Table 6.3.

Batch size	Epochs number	Patience	Time per Epoch, (seconds) (Average)	Convolution kernel	Accuracy (Test), %	Precision (Test), %	Recall (Test), %
200	8	30	107	31	84.578	96.084	87.304
150	15	60	115	31	79.647	96.018	81.946
100	7	30	56	15	76.413	96.331	78.116
50	9	30	122	15	81.793	96.185	84.148

Table 6.3: Hyperparameters, training parameters and error metrics results.

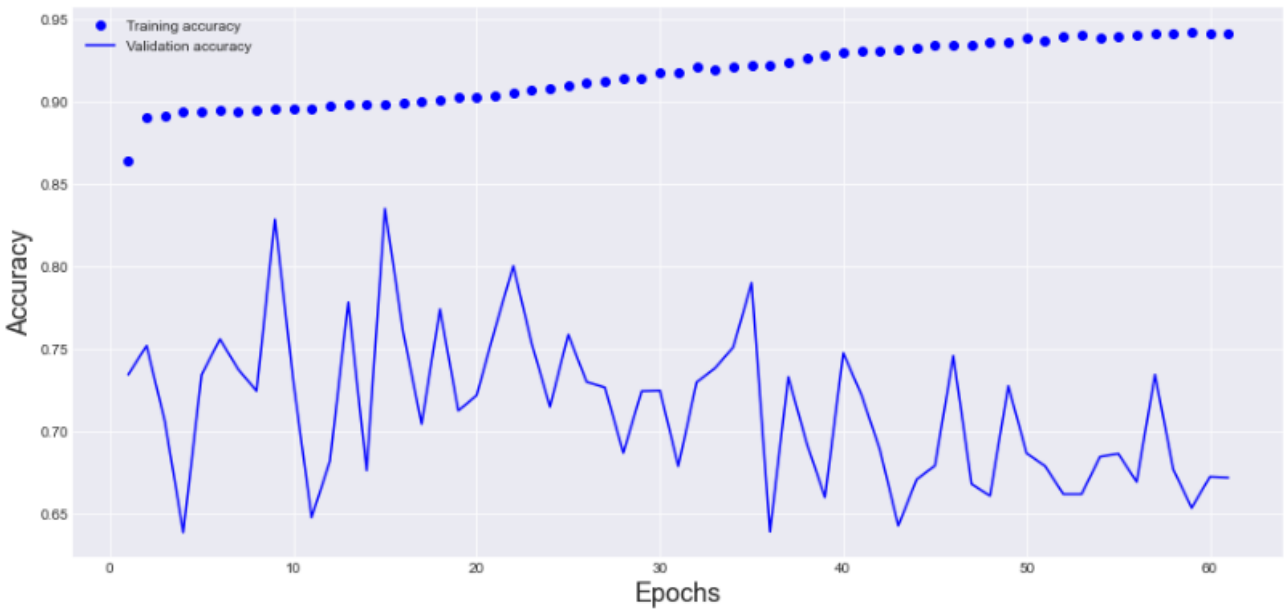


Figure 6.24: One of the accuracy charts for recognising trend or flat.

For achieving the purpose of predicting market behaviour (Trend or Flat), it is essential to improve error metrics such as accuracy, precision and recall. The examples of graphs of these error metrics during the training are shown in Figure 6.24 and Figure 6.25.

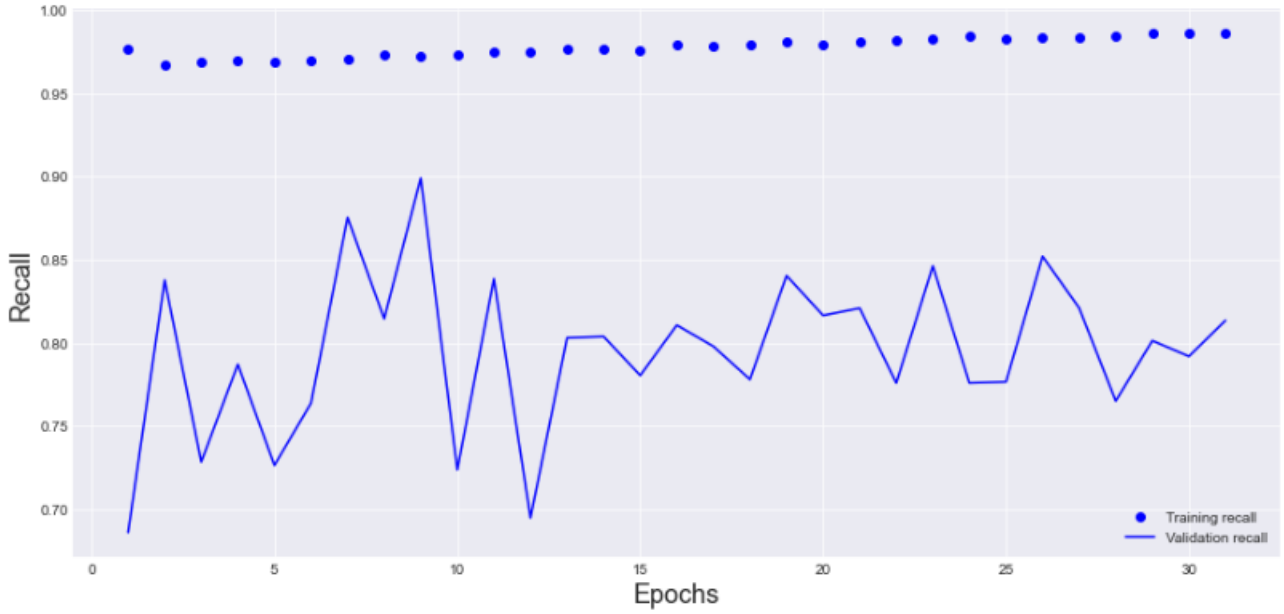


Figure 6.25: One of the recall charts for recognising trend or flat.

Training this CNN model in the second series of experiments is a classification task; consequently, precision is a second important parameter after accuracy for determining the trend, as precision determines how valid the results are. It is critically important not to recognise the flat as a trend, as in this case, there will be a loss from the settlement. Since the patterns of the financial market that precede a trend or flat are very similar to each other, the neural network is very susceptible to overfitting. It is necessary to observe the binary cross-entropy, which is the loss function in the binary classification task. Figure 6.26 shows this loss function where the overfitting start point is visible (as described in Section 4.6.1).

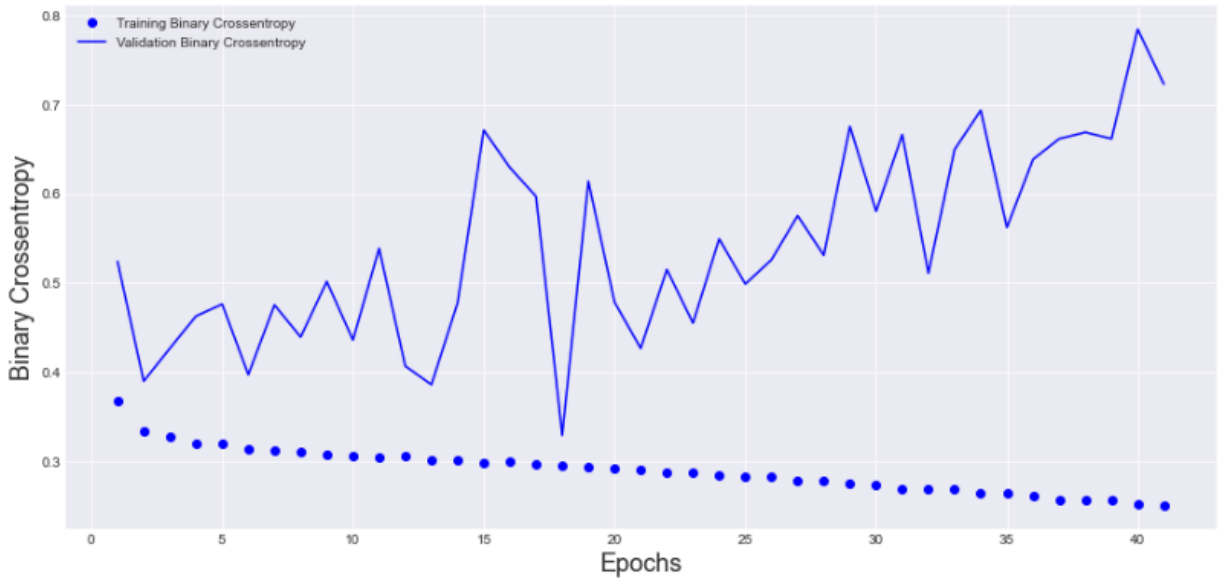


Figure 6.26: Binary cross-entropy loss-function.

The number of epochs was 100; however, the "EarlyStopping" function was applied with the "patience" parameter from 20 to 70. Using the "patience" parameter made it possible to improve the accuracy of the binary classifier (Trend-Flat). The overfitting is determined based on the minimum binary cross-entropy loss function, but maximum accuracy was also analysed. The best accuracy value is achieved when the batch size is 200 on the test sample but the best precision value is achieved with batch size is 100 (Table 6.3). In such ambiguous cases, it is better not to enter the financial market if one of these CNN models shows a flat. It is better to miss one trend than take a stop-loss. The correct solution would be to use an ensemble of these CNN models. Thus, for further practical applications, an ensemble of models with batch sizes of 100 and 200 will be used, with higher accuracy (84.578) and precision (96.331) values on average.

6.8.2. Comparative Evaluation

For a comparative evaluation, as in the first series of experiments, two types of recurrent neural networks were used, which are most often used in the analysis of financial markets and are the baseline in existing research (GRU and LSTM). The results of the experiments are presented in Table 6.4 compared to the results of the main experiments with CNNs.

Neural network binary classifier (Trend or Flat)								
Neural Network Architecture	**Batch size**	**Epochs number**	**Accuracy (Validation), %**	**Precision (Validation), %**	**Recall (Validation), %**	**Accuracy (Test), %**	**Precision (Test), %**	**Recall (Test), %**
CNN	200	8	85.59	92.89	91.47	84.578	96.084	87.304
LSTM	200	7	85.33	92,94	91.10	83.166	96.122	85.715
GRU	200	11	79.76	92.81	84.74	72.227	96.302	73.552

Table 6.4: Comparative evaluation of the quality of training and validation of the neural network architectures (CNN, LSTM, GRU) for binary classifier (Trend or Flat).

The experiments were carried out on the financial tick 2020 dataset. The total number of analysed market entries was more than 90,000 of both potentially profitable entry points and ticks from the flat. The experiments were carried out using the binary classifier (Trend or Flat). For providing a comparative assessment, both recurrent networks used the same batch size=200, which were most successful for the CNN in the main experiments.

LSTM in the classifier (Trend or Flat) shows similar quality metrics with the CNN: accuracy 83.166 and 84.578, respectively.

Thus, according to the binary classifiers (Trend or Flat), the accuracy values using the convolutional neural networks are 84.578, which is slightly better than LSTM, but they are approximately similar. Figure 6.27 demonstrates these results.

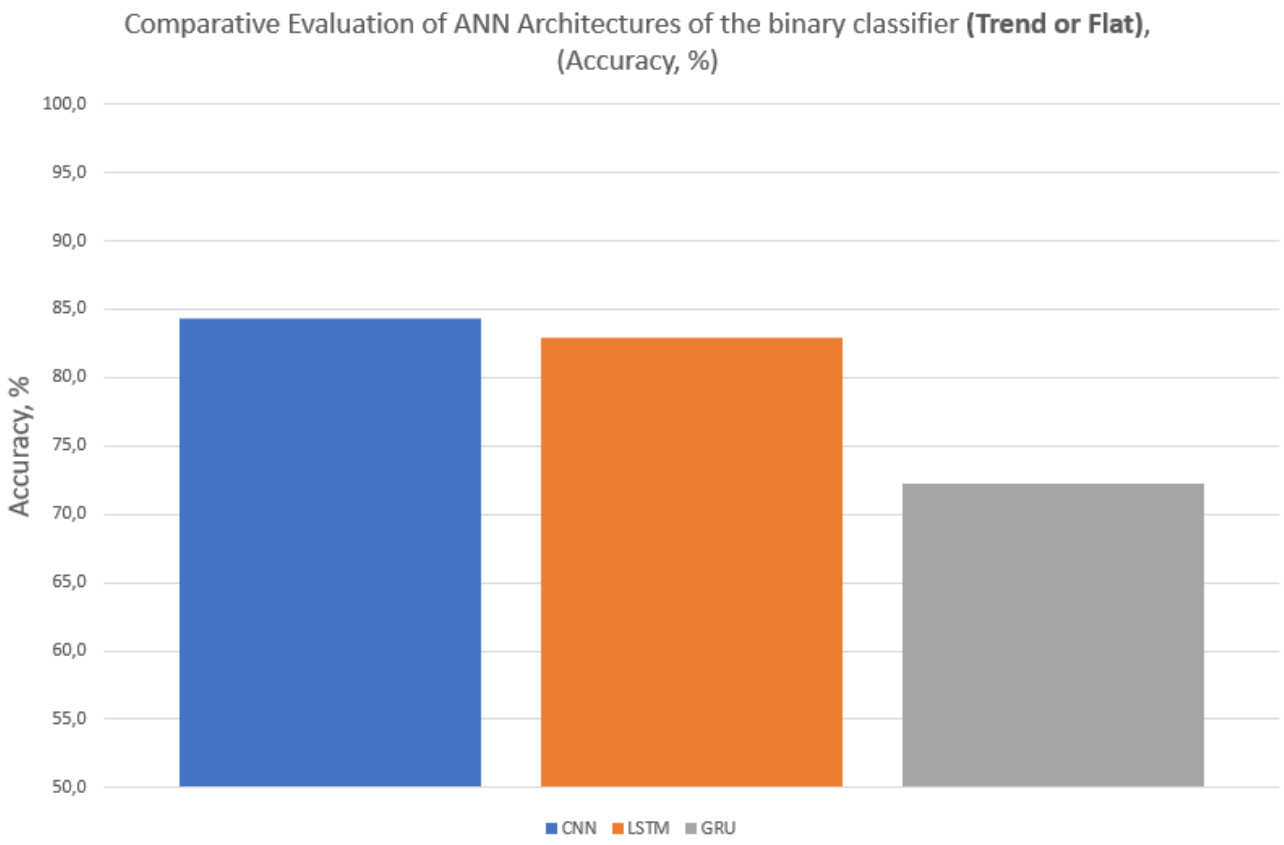


Figure 6.27: Comparative evaluation of the quality of training and validation of the neural network architectures (CNN, LSTM, GRU) for binary classifier (Trend or Flat).

This chapter described the series of experiments used to predict the probability of significant movement in the Forex EUR/USD rate. The chapter also analysed volatility over different time frames, which is necessary to create the probabilities vector. The results of the experiments using the binary classifier (Trend or Flat) are demonstrated and described. This classifier is necessary to significantly reduce trades on the flat market, which are not potentially profitable and instead lead to frequent stop-losses and loss of capital. Two types of recurrent neural networks (GRU and LSTM), which are most often used in existing research on financial market analysis, were used to evaluate the results.

7. Prediction of the Directions of a Significant Movement Based on the Probabilities Vector

This chapter describes two series of key experiments that use big financial tick data to predict the directions of significant financial market movements (up or down) based on the probabilities vector.

7.1. Prediction of the Directions of Significant Movements by the CNNs

These two series of experiments use the same raw financial big data, which was used in the first and second series of experiments (Sections 4.5.2 and 4.8). Similar 3D tensor and sliding window techniques and procedures, and time format conversions, data cleansing and parameter normalisations are also used (as described in Sections 4.8.1, 4.8.2, 4.8.3 and 5.1). The same CNN architectures are used as in the previous series of experiments (Section 6.7.2).

After determining the market behaviour using the binary classification: (Trend or Flat), the next stage is necessary to determine the direction of the market in accordance with the algorithm described above of splitting the probability vector into the parts (Figure 6.22). This second stage uses two independent CNNs with a binary classification, which determine the most probable direction of the future movement of the Forex EUR/USD. These two CNNs determine the following ways: (Sell or Wait) and (Buy or Wait); each of these CNNs trained separately on financial tick data for 2020. Experiment results of the binary classification (Sell or Wait) are presented in Appendix III; experiment results of the binary classification (Buy or Wait) are presented in Appendix IV. The summary of these experiments is shown in Table 7.1 and analysed in Section 7.2.1.

In both experiments, a CNN with eleven layers was used, the same as in the binary classification (Trend or Flat). The CNN (Sell or Wait) used the convolution kernels with lengths of 15 and 31 elements. The CNN (Buy or Wait) used the longer convolution kernel with 31 elements since the financial market patterns with an up-trend are weaker than with a down-trend. This is because financial markets fall faster than they rise, and sell patterns are more distinguishable than buy patterns.

The number of epochs was 100, with the "patience" parameter from 30 to 70. However, the training graphs show that the best parameters of the models occur mainly from epochs 7 to 25. In both CNNs, overfitting is determined based on the maximum accuracy, and the minimum of the binary cross-entropy loss function was also analysed. The time required for each experiment was also measured, and the average time required for each epoch was calculated. One of the variables was the batch size which varied from 30 to 200. At the same time, a larger batch size was used for the classifier (Buy or Wait) for the same reason that uptrend patterns are difficult to distinguish.

Tables in Appendix III and Appendix IV are divided into blocks, each of which indicates both the synthetic results of training these CNNs for 2020 and randomly (using the Python random number generator) selected sets of 10 samples of absolute errors in predicting the most likely solutions: Sell or Wait and Buy or Wait. Each of these ten samples was randomly selected from about 20 thousand samples, which are the results of the CNN predictions on the test datasets. In the future, these CNN models could be retrained each weekend using the data for the previous week.

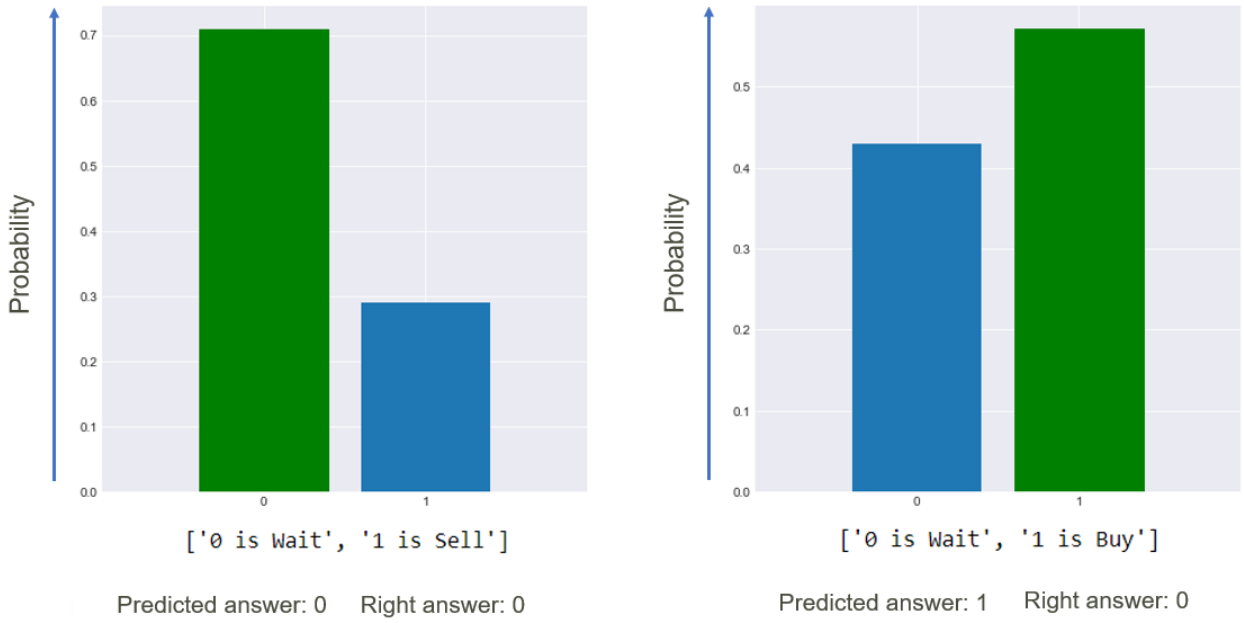


Figure 7.1: Probabilities (Sell or Wait), (Buy or Wait) and absolute prediction error for each tick.

The Python codes used for these experiments are the same as in the classifier (Trend or Flat) and are demonstrated in Appendix XIII. The differences are only in input data, outputs vectors, training parameters and results. For each potential market entry, the probability and absolute

forecast error (Figure 7.1) are calculated, and additional results of these experiments are error metrics, such as accuracy, precision and recall.

7.2. Evaluation

7.2.1. Statistical Evaluation

The training parameters, hyperparameters and error metrics results for both CNN classifiers are shown in Table 7.1.

Batch size	Epochs number	Patience	Time per Epoch, (seconds) (Average)	Convolution kernel	Accuracy (Test), %	Precision (Test), %	Recall (Test), %
Binary classification: Sell or Wait							
150	23	30	55	31	83.039	90.917	89.979
100	9	30	58	31	85.528	91.460	92.413
50	7	30	30	15	84.106	91.504	90.598
30	16	30	33	15	80.721	92.039	85.814
Binary classification: Buy or Wait							
200	14	30	63	31	82.556	92.532	87.914
150	18	30	58	31	81.935	92.285	87.446
100	13	30	61	31	77.167	92.331	81.676
50	15	30	72	31	75.111	92.505	89.259

Table 7.1: Training parameters, hyperparameters and error metrics for binary classifications: (Sell or Wait) and (Buy or Wait).

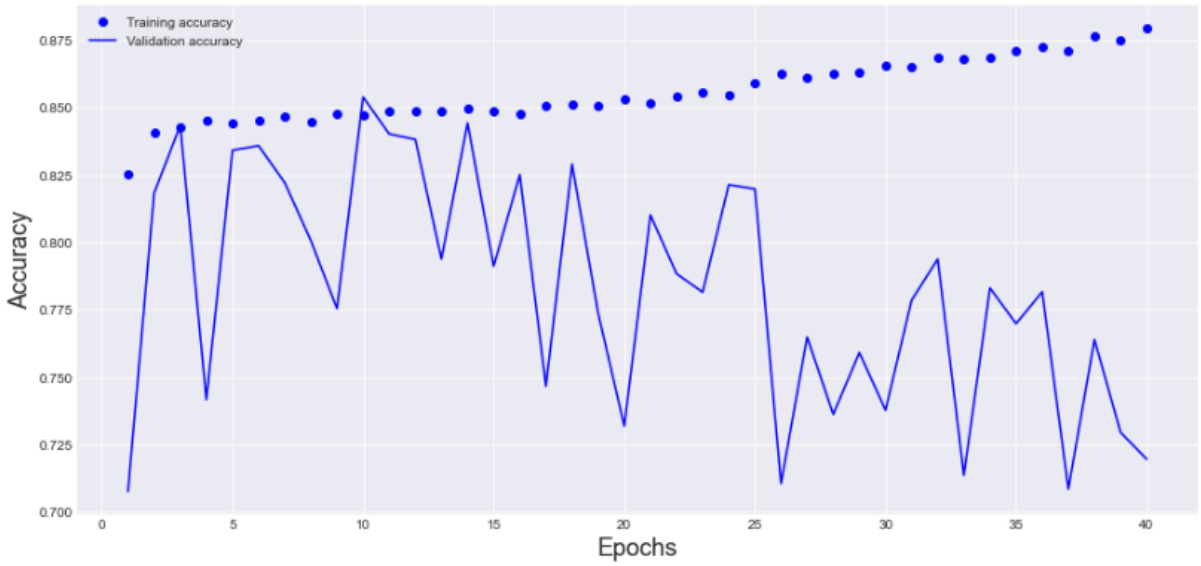


Figure 7.2: One of the accuracy charts for recognising (Sell or Wait) signals.

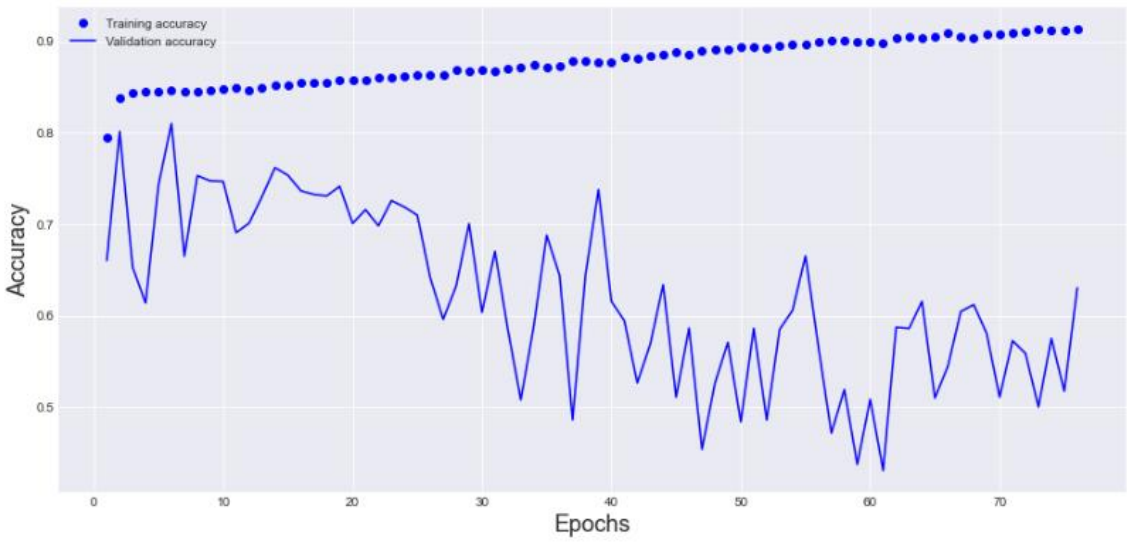
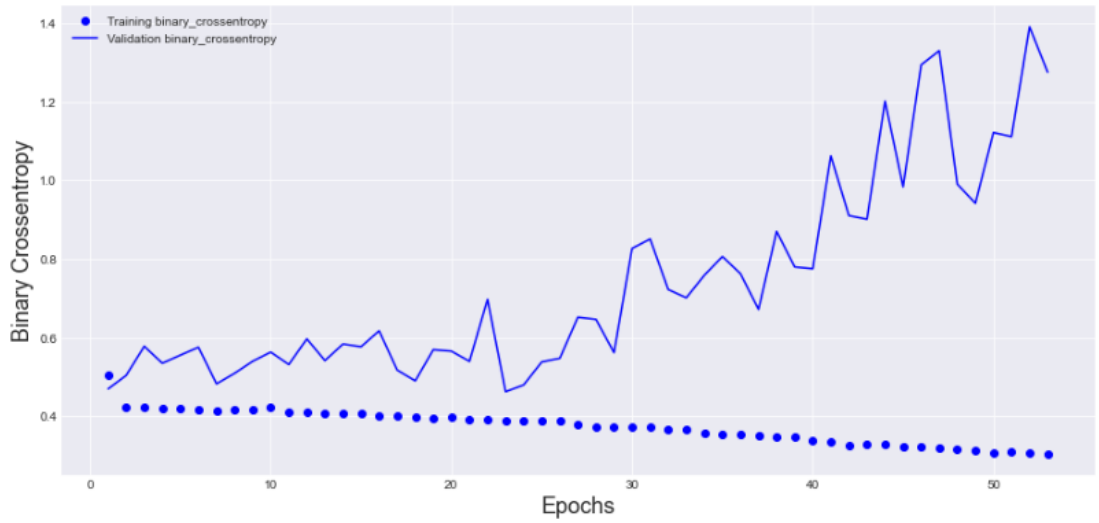


Figure 7.3: One of the accuracy charts for recognising (Buy or Wait) signals.



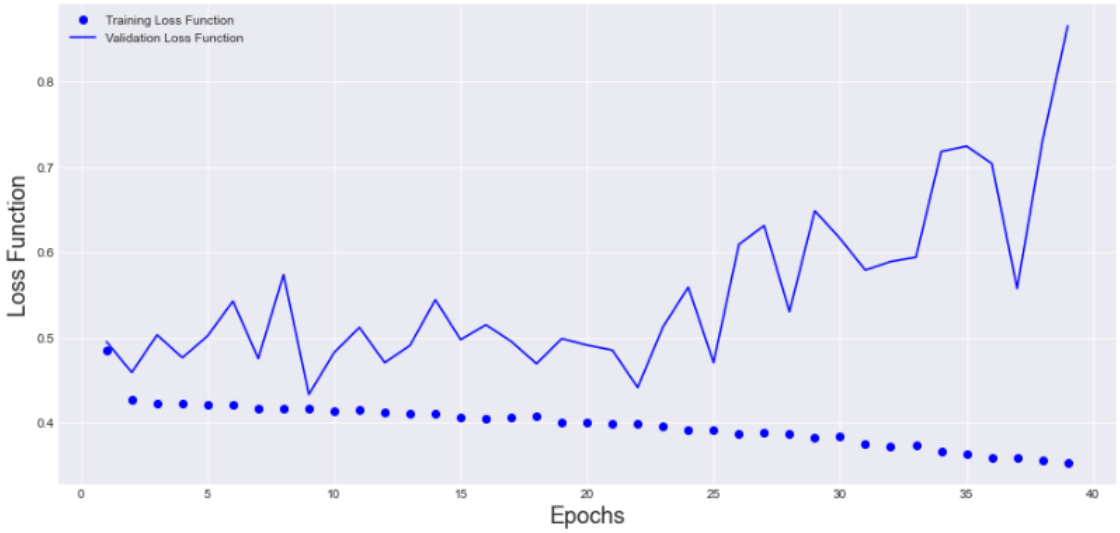


Figure 7.4: Binary cross-entropy loss function for recognising Sell or Wait and Buy or Wait signals.

For binary classifiers based on CNNs (Sell or Wait) and (Buy or Wait), the best accuracy values on the test samples are 85.528 and 82.556 and achieved when the batch sizes are 100 and 200, respectively. Figure 7.2 and Figure 7.3 demonstrate accuracy graphs of (Buy or Wait) and (Sell or Wait) classifiers.

For avoiding overfitting, the binary cross-entropy loss function was also used in these neural networks. These CNNs are very susceptible to overfitting since the Buy or Sell signals are only slightly different from each other, and the instability of accuracy during training also affects the loss function. Figure 7.4 shows this loss function.

Since the size of the incoming data was significantly reduced (Section 6.6) compared to the task of determining the price of the next tick in the first series of experiments, the training time is not very long and is approximately equal to 40…60 minutes per iteration, which allows experimentation with deeper neural network architectures.

For further practical applications, the CNN (Sell or Wait) and (Buy or Wait) classifiers will be used with batch sizes 30 and 200 with a convolutional kernel length of 31.

7.2.2. Comparative Evaluation

For a comparative evaluation, as in the previous binary classification, two types of recurrent neural networks were used (GRU and LSTM). The results of the experiments are presented in Table 7.2 compared to the results of the main experiments with CNNs.

Neural Network Architecture	Batch size	Epochs number	Accuracy (Validation), %	Precision (Validation), %	Recall (Validation), %	Accuracy (Test), %	Precision (Test), %	Recall (Test), %
Neural network binary classifier (Sell or Wait)								
CNN	100	9	84.96	86.31	98.13	85.528	91.460	92.413
LSTM	100	83	56.80	58.50	20.55	56.499	45.996	13.411
GRU	100	11	57.13	52.78	18.97	56.061	45.905	13.256
Neural network binary classifier (Buy or Wait)								
CNN	200	14	81.01	87.53	91.21	82.556	92.532	87.914
LSTM	150	79	52.96	53.33	32.06	49.213	51.510	25.855
GRU	150	67	53.04	57.54	15.97	50.510	57.611	15.258

Table 7.2: Comparative evaluation of the quality of training and validation of the neural network architectures (CNN, LSTM, GRU) for binary classifiers (Sell or Wait) and (Buy or Wait).

The GRU and LSTM architectures of the recurrent neural networks used are shown by Algorithm 7.1 and Algorithm 7.2, respectively. According to the technique of unrolling recurrent neural networks in time (as explained in Section 3.4.2), the LSTM used is the equivalent of a feedforward neural network with 32 consecutive layers, which is about three times deeper than the CNNs used in this research.

```
model = keras.Sequential([

    keras.layers.GRU(32, dropout=0.1, recurrent_dropout=0.5, return_sequences=True, input_shape=(200, 5)),

    keras.layers.GRU(64, activation=tf.nn.relu, dropout=0.1, recurrent_dropout=0.5),

    #keras.layers.Dense(1)
    keras.layers.Dense(1, activation=tf.nn.sigmoid)

])

print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_6 (GRU)                  (None, 200, 32)           3744
_________________________________________________________________
gru_7 (GRU)                  (None, 64)                18816
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 65
=================================================================
Total params: 22,625
Trainable params: 22,625
Non-trainable params: 0
```

Algorithm 7.1: Gated Recurrent Unit (GRU) neural network architecture for classification.

The experiments were carried out on the financial tick 2020 dataset. The total number of analysed market entries was more than 90,000 of both potentially profitable entry points and ticks from the flat. The experiments were carried out using two binary classifiers: (Sell or Wait), (Buy or Wait). Both recurrent networks used the similar batch size from 50 to 200. The number of epochs was 100, with the "patience" parameter from 20 to 70. The Python code for training both neural networks is the same as the classifier code (Trend or Flat) demonstrated in Appendix XIV. It differs only in outputs vector and results.

```
# Create LSTM model
model = keras.Sequential([

    keras.layers.LSTM(32, input_shape=(200, 5)),

    #keras.layers.Dense(1)
    keras.layers.Dense(1, activation=tf.nn.sigmoid)

])

print(model.summary())
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 32)                4864
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33
=================================================================
Total params: 4,897
Trainable params: 4,897
Non-trainable params: 0
_________________________________________________________________
```

Algorithm 7.2: Long short-term memory (LSTM) neural network architecture for classification.

Despite the deeper architecture of recurrent neural networks, all error metrics are better in the CNN. In the binary classifiers (Sell or Wait) and (Buy or Wait), recurrent neural networks show poor results, not comparable to the results of the CNN. For example, the LSTM network in the classifier (Sell or Wait) demonstrates accuracy that is slightly more than 56% (Figure 7.5). This is explained by the fact that CNNs look more effectively for poorly recognisable signals, as it looks at all data (within a 200-tick of sliding window). RNNs, on the other hand, analyse ticks separately, trying to remember the results of the analysis of the previous data.

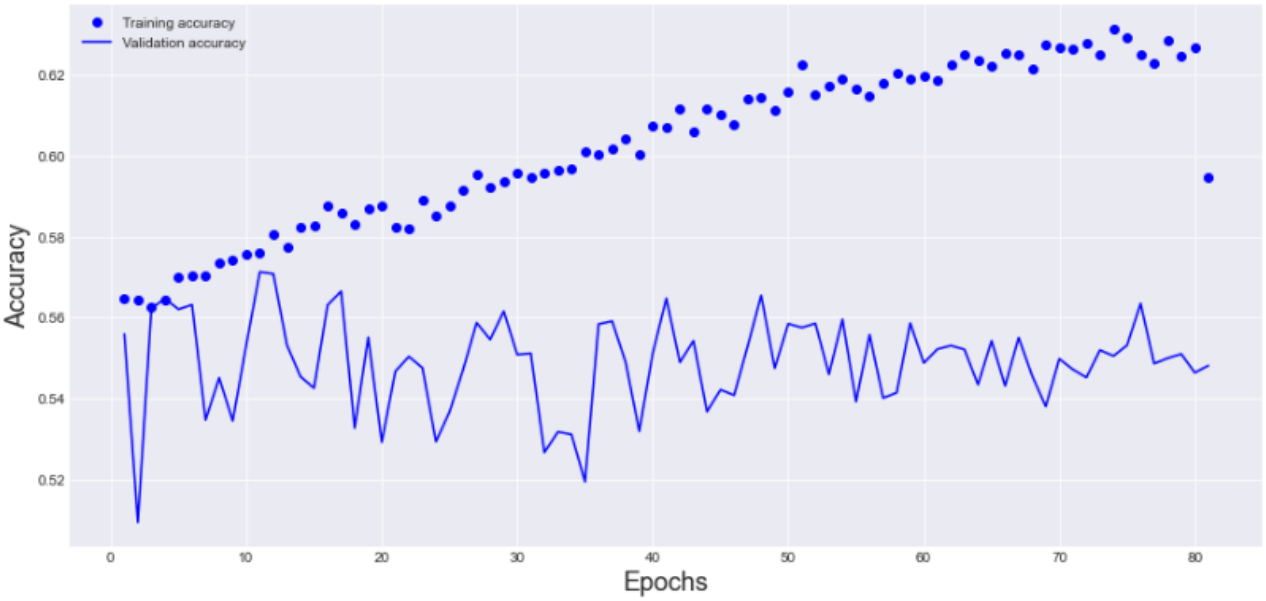


Figure 7.5: Accuracy of the LSTM neural network classifier (Sell or Wait).

Figure 7.6, Figure 7.7 and Figure 7.8 show examples of error metrics, such as accuracy, precision and recall, during the training of these RNN binary classifiers. These graphs demonstrate significant instability in training recurrent neural networks with noisy financial data.

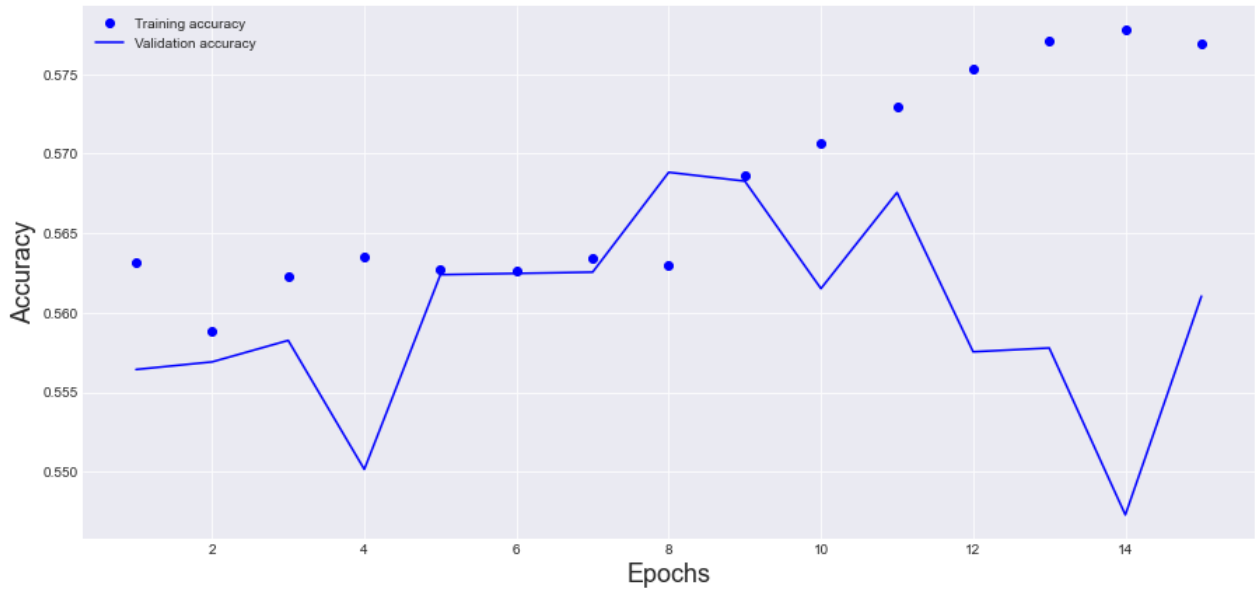


Figure 7.6: Accuracy graph of training the GRU neural network classifier (Sell or Wait).

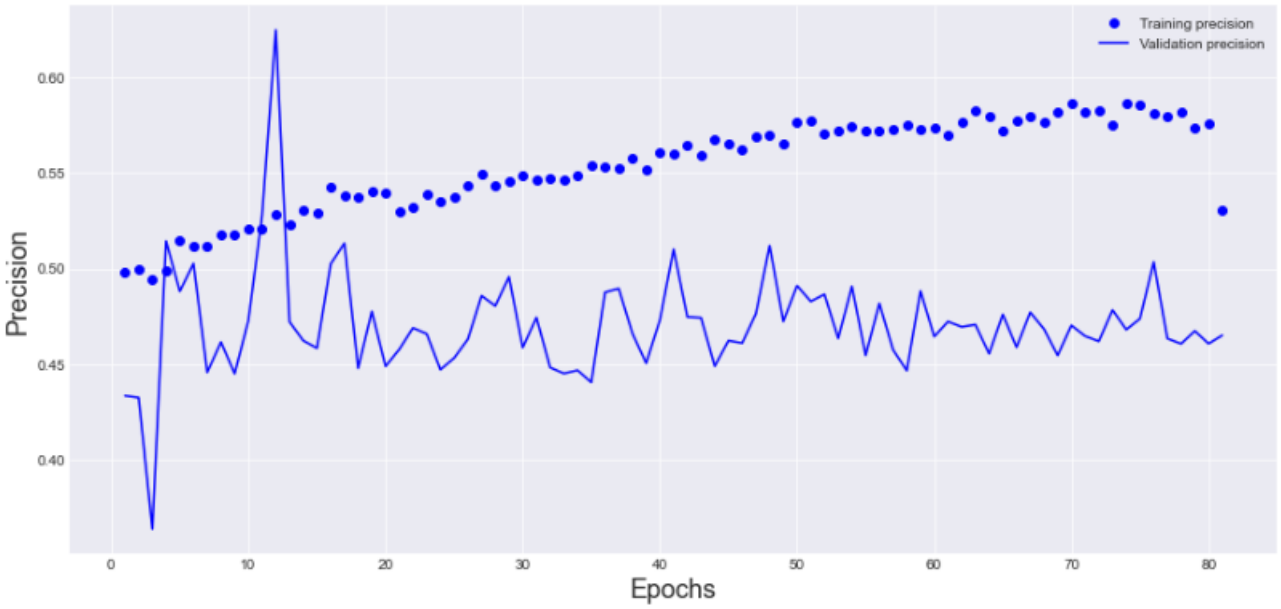


Figure 7.7: Precision graph of training the GPU neural network classifier (Sell or Wait).

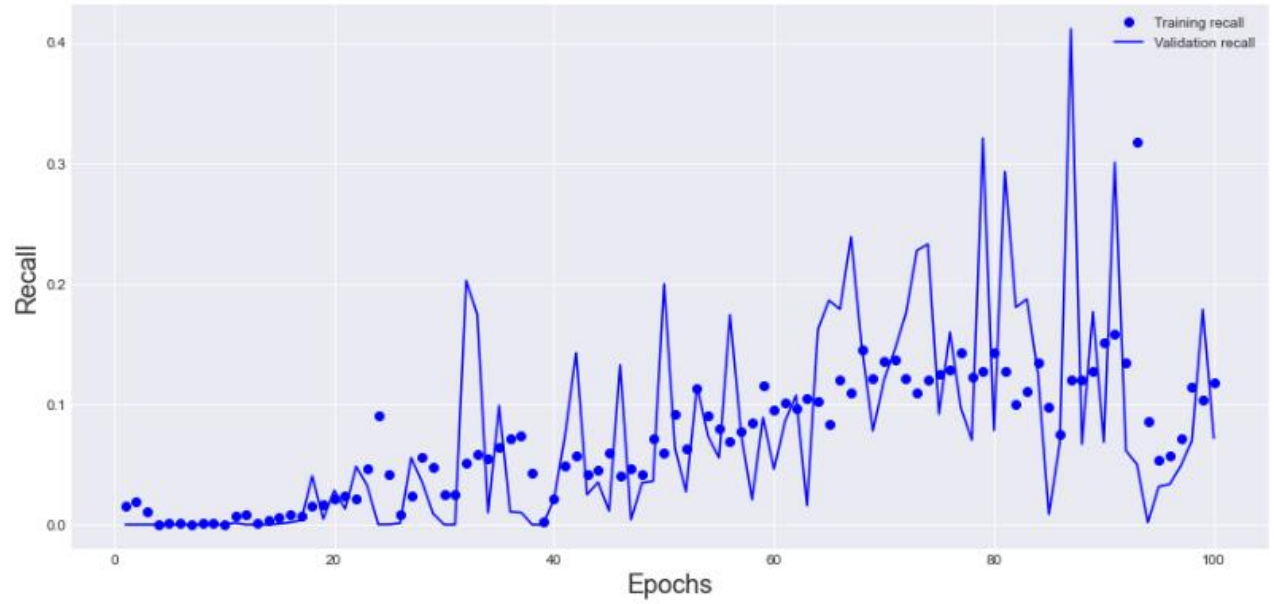


Figure 7.8: Recall graph of training the LSTM neural network classifier (Buy or Wait).

Thus, according to two binary classifiers (Sell or Wait) and (Buy or Wait), the accuracy values using the convolutional neural networks are 85.528 and 82.556, respectively, which is the best value of these experiments. Figure 7.9 and Figure 7.10 demonstrate these results.

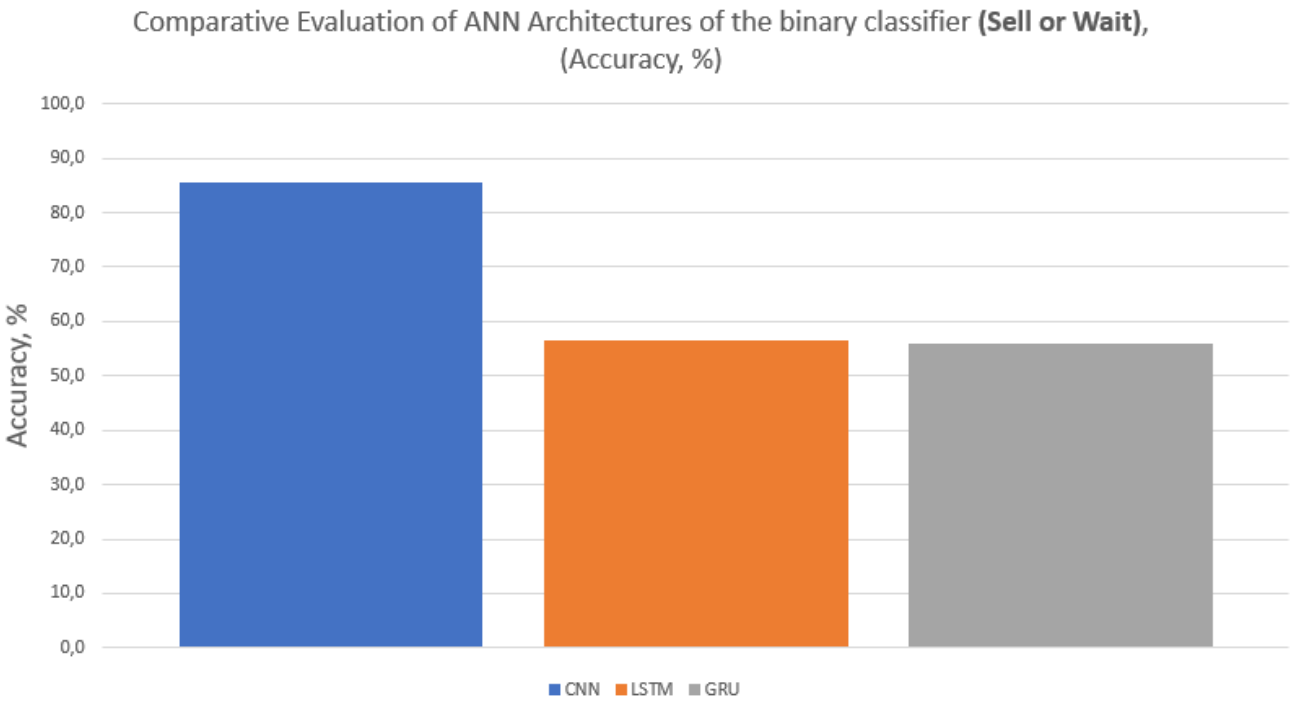


Figure 7.9: Comparative evaluation of the quality of training and validation of the neural network architectures (CNN, LSTM, GRU) for binary classifier (Sell or Wait).

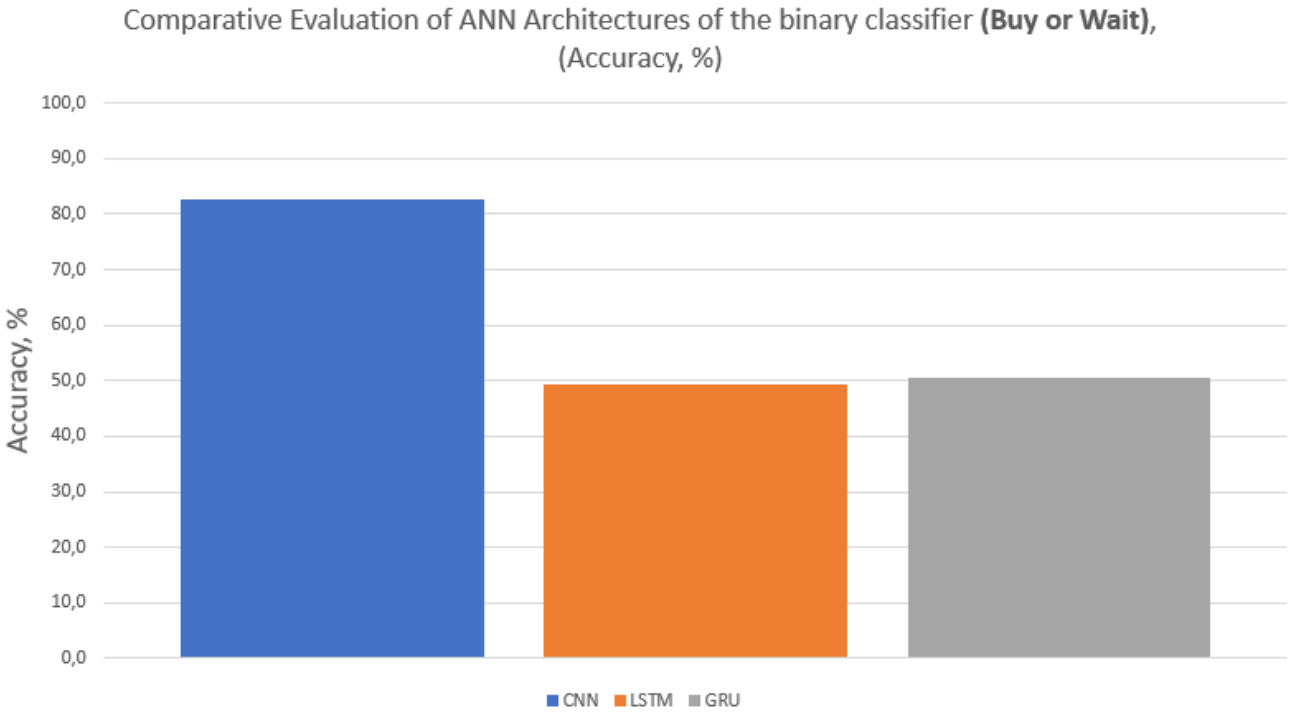


Figure 7.10: Comparative evaluation of the quality of training and validation of the neural network architectures (CNN, LSTM, GRU) for binary classifier (Buy or Wait).

This chapter described the two series of experiments in predicting the directions of the following significant movement in the Forex EUR/USD rate based on the probabilities vector. The results of these experiments using two binary classifiers (Sell or Wait) and (Buy or Wait) were demonstrated and described. These classifiers are the final results of this research and allow entry into positions with probabilities above 80%, which is a good result. Two types of recurrent neural networks (GRU and LSTM) were used for validation.

8. Evaluation and Discussions

8.1. Comparison of the Results of Key Experiments

This section summarises all the results of the key experiments (described in Chapters 6 and 7) in achieving the overall intraday trading task, which consists of three binary classifiers:

- separation of noisy, flat movements from trend ones (Trend or Flat),

- determining the readiness of the EUR/USD currency pair for a significant downward movement (weakening of EUR and strengthening of USD) (Sell or Wait),

- determining the readiness of the EUR/USD currency pair for a significant upward movement (strengthening EUR and weakening USD) (Buy or Wait).

The experimental results are shown in Table 8.1. The comparison is based on Accuracy as the primary error metric, which could then be evaluated against existing research results. Comparison and evaluation are carried out on three architectures of neural networks (CNN, LSTM, GRU), which are the state-of-the-art baseline technique in this domain (Section 3.7).

	Neural Network Binary Classifiers		
Neural Network Architecture	**Trend or Flat**	**Sell or Wait**	**Buy or Wait**
CNN	84.578	85.528	82.556
LSTM	83.166	56.499	49.213
GRU	72.227	56.061	50.510

Table 8.1: Comparative evaluation of the Accuracy of training and validation of the neural network architectures (CNN, LSTM, GRU) for key binary classifiers on the testing datasets.

Figure 8.1 demonstrates the above results of the key experiments of this research.

Thus, according to these three binary classifiers (Trend or Flat), (Sell or Wait) and (Buy or Wait), the accuracy values using the convolutional neural networks are 84.578, 85.528 and 82.556, respectively, which is the best value of these experiments.

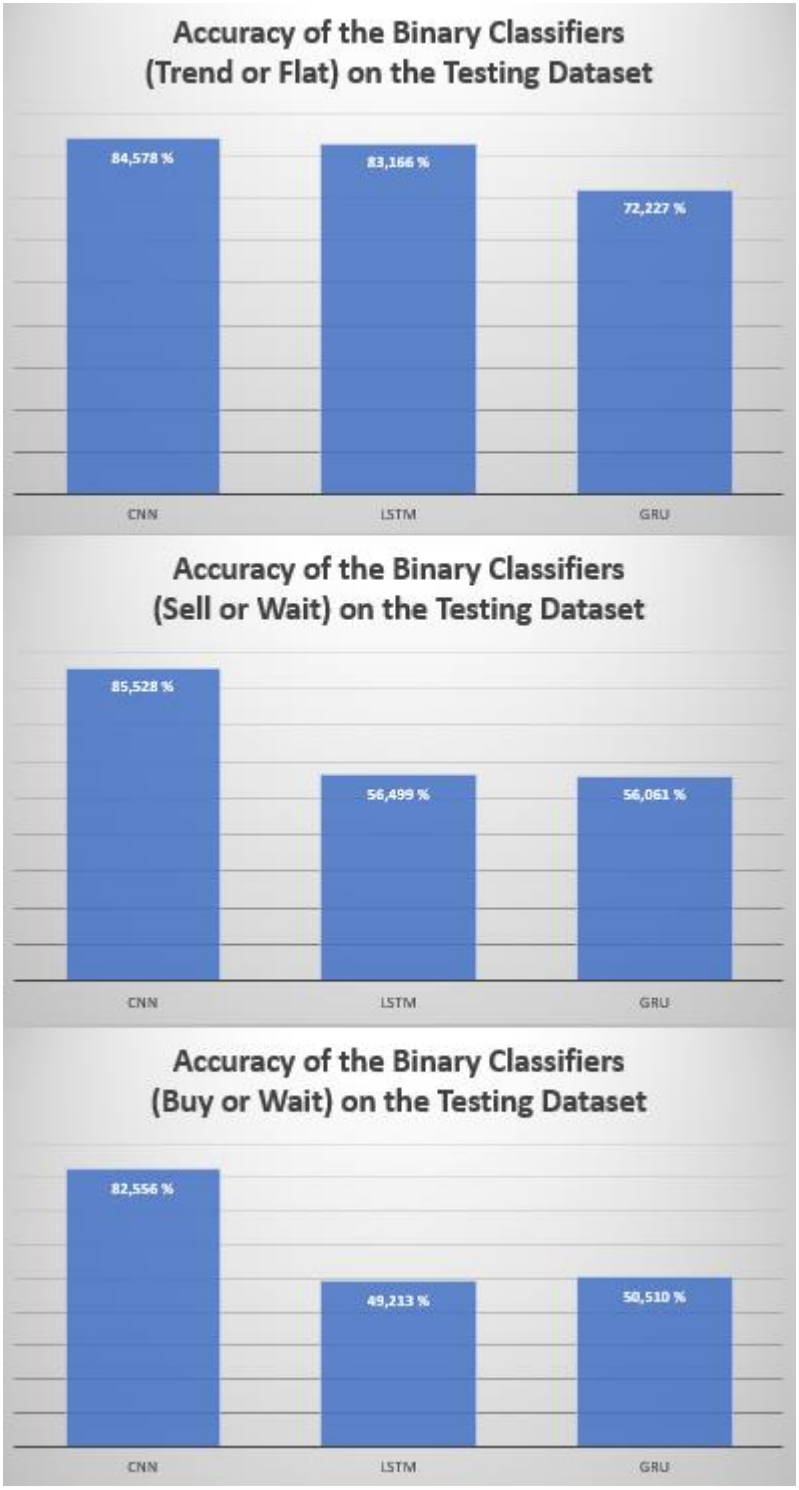


Figure 8.1: Comparative evaluation of the neural network architectures (CNN, LSTM, GRU) by the accuracy on testing data.

This result confirms the theory expressed in Section 3.5 that the analysis of short-term intraday movements using convolutional neural networks is an evolutionary development of Japanese candlestick charting techniques. The aim of this was to determine the significant movement and direction of the financial market within short time frames. If the visual analysis

of Japanese candlestick patterns was mainly limited to five candlesticks, then modern convolutional networks can successfully operate up to 200 or more tick data simultaneously, finding bullish or bearish patterns in their combinations.

8.2. Evaluation of the key experiments

When analysing the Forex market, the majority of researchers focus on predicting the future prices of currency pairs or precious metals. Such an approach is justified if the considered investment horizons are sufficiently long because, in this case, it is necessary to assess the ROC compared to alternative returns. However, there are papers that focus on the needs of short-term intraday trading, the primary task of which is to determine the direction of the most probable next significant movement, which coincides with the purpose of the key experiments in this research.

Galeshchuk and Mukherjee (2017) analyse the movements of the EUR/USD currency pair, focusing on predicting the next movement direction. This paper uses two statistical methods for the analysis of time series: ARIMA and Exponential Smoothing (ETS), in comparison with a machine learning method SVM, and deep neural networks ANN(MLP) and CNN. The authors also trained ANN(MLP) in addition to the currency pair prices on moving averages MLP(MA). Table 8.2 shows the results of the Accuracy of predicting the direction of the next movement.

ARIMA	ETS	MLP	MLP(MA)	SVM	CNN
60.05	52.05	58.33	65.40	65.13	75.28

Table 8.2: Classification accuracy for the EUR/USD currency pair in determining the direction of the next price movement (Galeshchuk and Mukherjee, 2017).

In this paper, the Accuracy of predicting the next movement is calculated separately from the following 30 prices, some of which may be in a flat movement. Then, the average Accuracy is calculated. These results can be compared with the results of this research of both binary classifiers (Trend or Flat) and the binary classifiers for determining the direction of future movement (Sell or Wait) and (Buy or Wait).

Moghaddam and Momtazi (2021) analysed and applied a complex model consisting of CNN and LSTM elements to analyse Japanese candlestick patterns of the EUR/USD Forex

currency pair. They examined a binary classifier (profitable/non-profitable) and ternary classifier (buy/sell/non-profitable). The intersection of moving averages is used to confirm the profit, which can be controversial. The (profitable/non-profitable) classifier is similar to the classifier (Trend-Flat) of this research.

Table 8.3 shows the results of calculating the Accuracy of these binary and ternary classifiers in predicting trends and flats in the paper.

LSTM-CNN (Trend or Flat)	LSTM-CNN (Buy-Sell-Wait)
76.09	63.95

Table 8.3: Classification Accuracy for the EUR/USD currency pair to determine Trend-Flat and Buy-Sell-Wait options of the next price movement (Moghaddam and Momtazi, 2021).

Figure 8.2 shows the comparison of the Accuracies obtained by the author in this research, Galeshchuk and Mukherjee (2017), and Moghaddam and Momtazi (2021).

Based on this summary diagram, the following conclusions could be drawn:

- CNNs are the most accurate in finding the direction of the subsequent significant intraday price movements, as they analyse short-term market participants' behaviour patterns, similar to the principles of analysis using Japanese candlesticks. The data preprocessing technologies proposed in this research contributed significantly to the high accuracy of the predictions.

- LSTMs also show promising results in determining the market behaviour (Trend or Flat). This is due to the high importance of the sequence of data within each slice of the 3D tensor when expected to exit from the flat.

- Binary classifiers perform better than ternary classifiers even when using more complex neural network architectures.

- Simple ANNs, such as MLP, can produce significantly better results when technical indicators are used in addition to price data.

- Recurrent neural networks show medium results in determining the directions of future short-term movements.

- Baseline techniques such as ARIMA and SVM show weaker results than neural networks in determining the patterns of subsequent intraday market movements.

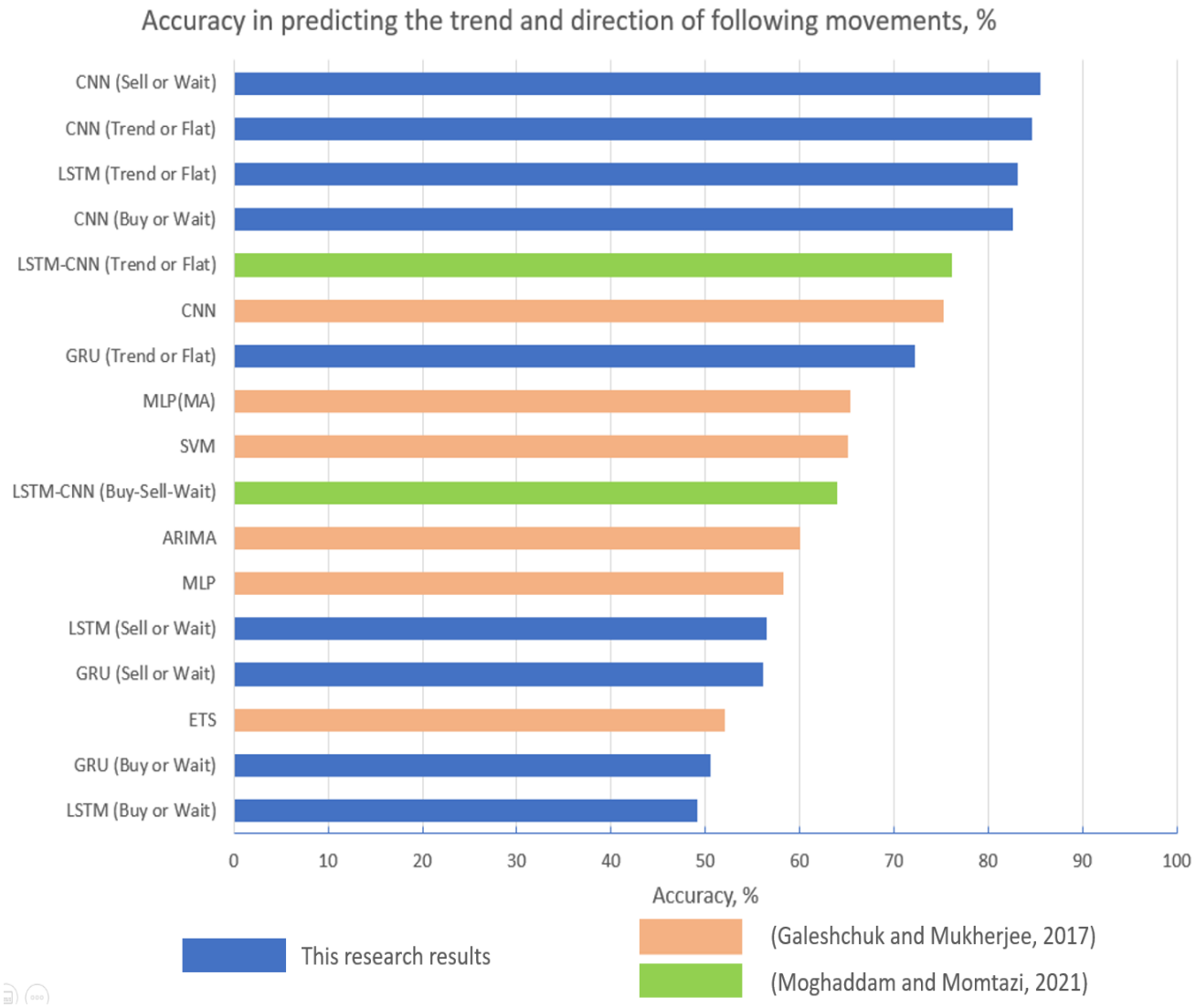


Figure 8.2: Predicting the trend and direction of subsequent movements (Accuracy, %).

8.3. Limitations and Discussion

The complex purpose of predicting the probability vector of significant financial market movement was split into three sub-purposes, two of which are binary classification tasks: predicting the time of the exit from a flat and the beginning of a trend (Section 6.7) (binary classifier Trend or Flat) and determining the most probable direction of the future market movement (Section 7.1) by two binary classifiers (Sell or Wait) and (Buy or Wait). The third sub-purpose is a multidimensional classification task with dynamic range bounds, which requires more processing power than is available for free on the Google Colab platform

(Section 6.7.3). This problem has been solved using standard techniques such as trailing profit orders.

These two binary classification problems could be successfully solved using the one-dimensional architectures of CNNs (Section 5.1). However, this requires advanced preliminary analysis of big financial tick data. The transition from the regression task to the classification task using the analysis of the probability vector made it possible to significantly reduce the amount of analysed data, training neural networks only on potentially profitable points of entry into the financial market.

Recurrent neural networks commonly used to predict prices of financial markets do not always give an acceptable result, especially if the patterns determining the direction of the financial market movement are difficult to distinguish. It happens because, unlike CNNs, RNNs analyse data sequentially. Another essential advantage of CNNs is the ability to parallelise better computations than RNNs, allowing the training of neural networks faster on the same computer hardware. However, further research is needed to improve the stability of these neural network models.

Recognising the starting of a trend and predicting the direction of significant movement in financial markets using neural networks is a completely different task from recognising images or creating winning strategies in games like Go. The main problems are associated with data that is noisy and non-stationary. However, this research demonstrates that in the short term, it is possible to recognise the behaviours of market participants, which are reflected in the patterns of prices and volumes of transactions.

9. Conclusions and Future Work

The primary motivation behind this thesis is to explore the possibility of analysing big financial data and identifying patterns of behaviour of significant financial market participants, such as funds and market-makers, using advanced deep convolutional neural network models, and bridging the research gap described in Section 1.4.

9.1. Review of the Work

The aim of this research was to improve the state-of-the-art performance of intraday trading. The research investigated deep learning algorithms and conducted experiments to achieve increased accuracy in intraday trading predictions, identify market entry points with the highest probability of significant movements and reduce losses from execution stop-loss orders. In order to achieve this aim, four research objectives have been formulated (Section 1.3).

The detailed report on the implementation of each of these research objectives is presented in Section 9.3.

As noted in Section 2.10, any transaction in the financial market consists of three parts:

- the point of entry into the market,
- tracking the potential of the opened position,
- the point of exit from the market.

The tracking and exiting from the open position have been done using standard techniques such as risk and money management and advanced tools such as trailing profit orders.

As in other similar studies, the main focus of this research was on finding the most promising market entry points with minimal risk. This task, in turn, was separated into several subtasks:

- The minimisation of risk was ensured by using short stop-loss orders, which are mandatory for intraday trading, so as not to receive significant losses in case of a mistake in predicting the direction of the transaction (buy or sell).
- In turn, using short stop-loss orders gave rise to another risk - a possible drawdown of investment capital due to the frequent execution of these orders. To minimise this risk,

the problem of determining high-quality entry points was solved, which have a low risk of a reverse movement in the opposite direction of the transaction.

- Unfortunately, the ‘end-to-end’ concept, where raw data is fed to one big neural network as inputs, a method that is well-utilised in other domains, does not work for big financial data due to noise and variability reasons. Therefore, a deeper preliminary analysis of the raw input data is required, which was carried out in this research.
- Several convolutional neural networks were trained to determine these quality entry points, which work sequentially. These CNNs were trained using a supervised learning algorithm on pre-labelled historical data, marking target ticks followed by recoilless significant price movements of the EUR/USD currency pair. Thus, the research objective-3 was achieved in separating the complex task of creating an intraday trading system into its constituent parts, for each of which standard error metrics and loss functions were identified, such as Accuracy and MAE.
- The first CNN (the first binary classifier) determined the behaviour of the market (Trend or Flat). It blocked the signals to open positions if the market was flat to reduce the likelihood of stop-loss orders being executed.
- The following CNNs (binary classifiers) determined the potential direction of the next significant movement (Up or Down). These CNNs only started analysing the data if the previous binary classifier showed the 'Trend' signal. In other words, they recognised tick patterns on each 200-tick slice of the 3D tensor (Sell or Wait) or (Buy or Wait), similarly to traders who had visually recognised bullish or bearish patterns using Japanese candlestick charts. In the case of positive recognition, a deal was made for buy or sale, respectively.
- The research objective-2 of solving the problem of the constant market-changing was solved with the help of organisational actions in the form of additional training of all the above CNNs every week with data for the previous week.

9.2. Summary of Contributions to Knowledge

This thesis created several novelties, among which the most important are:

- replacement of the typical aim of using a regression task to predict the price of a financial instrument, with a classification task which enabled improvement in the ability to evaluate the quality of the results and utilised many advanced machine learning methods, which are already used in other areas, such as image recognition;

- an advanced preliminary analysis of big financial tick data based on intraday volatility and risk assessments of losses;
- the introduction of a probability vector of significant future movements based on searching for patterns in the behaviour of significant market participants, such as funds or market makers.

For applying these essential novelties, new approaches to advanced preliminary data analysis have been used, such as:

- using big financial tick data together with convolutional neural networks without using typical recurrent neural networks,
- non-linear unique normalisation of data regarding the price and volume of each tick (minimum price change during trading);
- non-linear unique normalisation of time intervals between tick data, which includes crucial information about the structure and intensity of the market at each moment. It also allows conversion of irregular time series into regular time series and thus allows the application of analysis methods designed for typical time series;
- using a sliding window that creates a three-dimensional data tensor which generates inputs for feeding into convolutional neural networks;
- reducing the dimension of data by replacing two-dimensional convolutional neural networks with one-dimensional while preserving the three-dimensional structure of the input data tensor;
- using a new approach to significantly reduce data and remove flat market movements.

As a result, it was possible to avoid the need to predict a specific price of a financial instrument after a particular time, which cannot be solved due to the instability and volatility of financial markets. This was replaced with a far more accurate prediction of the direction of recoilless movement and start predicting its depth. This is a vital task for real-world trading.

From a practical point of view, it does not matter what time the desired price will be reached. It is more important that the price will be achieved with a greater than 50% probability before the end of the current trading day. For actual intraday trading, it is also essential to predict that during the period after entering the market, the price does not go in the opposite direction and activate the stop-loss. The proposed novelties made it possible to solve these problems with sufficiently high accuracy at a level around 80%+ for different subtasks.

These experiments have shown that convolutional neural networks can recognise subtle patterns that recurrent neural networks cannot always detect. However, more research is needed to improve the stability of neural network training.

9.3. Results and Discussion

All the research objectives identified in Section 1.3 have been achieved.

Objective-1 has been achieved by undertaking a literature review and analysis of existing machine learning and data collection software in detail in Chapter 3 and Chapter 4. Preliminary research has been done on the most appropriate neural network architectures, such as CNNs, to achieve the following objectives.

Objective-2 of designing and creating an appropriate neural network and target function that can adapt to dynamically changing data has been achieved by proposing the specific convolutional neural network architectures used in Section 5.3 and Section 6.7.2. These minimise stop-loss as the primary target function of training (Section 2.9) using preliminary preparation and deletion of tick data corresponding to the flat periods of the financial markets. All financial ticks, after which follow movements in the opposite direction (Section 6.6), were also removed because they activate stop-loss orders and lead to losses in intraday trading. Adaptation of these neural networks to the changing financial market is provided by weekly retraining of these convolutional neural networks on the data from the previous week.

Objective-3 of separating the complex aim of creating and improving an intraday trading system into its constituent parts by proposing the separation of one large classifier into a sequence of CNNs (binary classifiers) (Section 6.7.3), for each of which is possible to identify error metrics and loss functions (Section 3.7), has been achieved. Also, all these neural networks were independently tested, compared and evaluated to existing state-of-the-art research (Section 8.2).

Objective-4 of testing and evaluating different models, ML frameworks, tools and data collection software and analysing publicly-available financial market data has been completed. This was achieved by processing tests and evaluating models for predicting the next tick prices (Section 5.4), predicting the nature of the market (Trend or Flat) (Section 7.2.2) and predicting the most probable directions of the following market movements and identifying entry points into the financial market with minimal risk (Section 7.2.2 and Section 8.2). The model for predicting the depth of the following, most probably market movement,

requires more processing power than is publicly available using Google Colab or available GPU hardware. However, this module is temporarily replaced with a regular trailing stop-loss order for practical trading.

The experiments were conducted to predict the price of the S&P index (Section 4.9.1), visualise data (Section 4.7) and determine the probability vector for the second series of experiments (Section 6.5).

In conclusion, it should be noted that today there are many different methods and tools for analysing the financial market using machine learning methods and ANNs. However, the challenge lies in the statement of the problem, in the precise definition of the search goals and understanding and correctly preparing the data from financial markets to obtain stable results.

9.4. Future Work and Challenges

Many difficulties and additional ideas have emerged during this research that could form the basis for further research:

- The task of a fully automatic trading system that adapts to a constantly changing market looks very attractive. Such a model should be able to collect new data and retrain and test itself automatically and permanently without the participation of a human.
- Another possible direction for developing this research is training an ensemble of neural networks, each of which is trained on different time frame data. This idea originated from the three-step decision-making process (described in Section-1) based on Fundamental and Technical analysis and pattern searching using Japanese candlesticks. Such a system will decide to enter into the market based on a combination of factors with certain weighting for each time frame. This system may consist of neural networks with different architectures.
- The next important area of research is profit maximisation through neural network analysis of the levels of take-profit orders. This task could be more complicated than determining market entry points since, in this case, it will be necessary to calculate and support the potential of each open position until it is closed.

- Expanding the application of the results of this research to the analysis of other liquid financial instruments in the Futures contracts and Forex markets. It is also possible to apply this approach to commodities and cryptocurrencies.
- One of the problems identified during this research is difficulties in evaluating the results of predicting the depth of future movements. The reason is that standard error metrics, such as multidimensional cross-entropy, work on the presence of fixed boundaries of recognised classes. However, classes in financial markets have dynamic boundaries that need to be redefined for each tick at short intervals using unsupervised learning methods. This goal возможно потребует more GRU или TPU processing power.
- Extension this research by adding limit-orders data from the Order Book as additional information to big financial tick data. This will significantly increase the size of the analysed data; however, it will allow neural network training based on both completed transactions and the potential intentions of the financial market participants.
- Reinforcement learning technology, which is a logical continuation of this research, also provides more excellent opportunities. Reinforcement learning agents based on this research could be created. However, the number of iterations required for training the neural networks will increase by multiple orders of magnitude.

Machine learning methods are developing quite quickly, and the author is confident that many of the current limitations in processing power for solving the above problems will be overcome, which will allow this research to be continued and enhanced.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X., 2016. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv preprint arXiv:1603.04467.*

Aggarwal, P. and Sahani, A.K., 2020, November. Comparison of neural networks for foreign exchange rate prediction. In 2020 IEEE 15th International Conference on Industrial and Information Systems (ICIIS) (pp. 415-419). IEEE.

Aldrich, E.M. and Vargas, K.L., 2020. Experiments in high-frequency trading: comparing two market institutions. *Experimental Economics*, 23(2), pp.322-352.

Almahdi, S. and Yang, S.Y., 2019. A constrained portfolio trading system using particle swarm algorithm and recurrent reinforcement learning. *Expert Systems with Applications*, 130, pp.145-156

Alves, S.A., Caarls, W. and Lima, P.M., 2018. Weightless Neural Network for High Frequency Trading. In: *2018 International Joint Conference on Neural Networks (IJCNN),* Volume IEEE, pp. 1-7.

ARQA Technologies, 2022. *Quik software package.* [Online] Available at: <http://arqatech.com/en/products/quik/> [Accessed 29 March 2022].

Ashangani, K., Wickramasinghe, K.U., De Silva, D.W., Gamwara, V.M., Nugaliyadde, A. and Mallawarachchi, Y., 2016, October. Semantic video search by automatic video annotation using TensorFlow. In: *2016 Manufacturing & Industrial Engineering Symposium (MIES)* (pp. 1-4). IEEE.

Barber, B.M., Lee, Y.T., Liu, Y.J. and Odean, T., 2009. Just how much do individual investors lose by trading?. *The Review of Financial Studies*, 22(2), pp.609-632.

Baron, M., Brogaard, J., Hagströmer, B. and Kirilenko, A., 2019. Risk and return in high-frequency trading. *Journal of Financial and Quantitative Analysis*, 54(3), pp.993-1024

Bauwens, L. and Giot, P., 2013. *Econometric modelling of stock market intraday activity* (Vol. 38). Springer Science & Business Media

Baviera, R. and Baldi, T.S., 2019. Stop-loss and leverage in optimal statistical arbitrage with an application to energy market. *Energy Economics*, 79, pp.130-143.

Baxter, M. and Rennie, A., 1998. Financial Calculus: An Introduction to Derivative Pricing. *Econometric Theory,* 14(3), pp. 365-368.

Bengio, Y., Simard, P. and Frasconi, P., 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2), pp.157-166.

Benos, E., Brugler, J., Hjalmarsson, E. and Zikes, F., 2017. Interactions among high-frequency traders. *Journal of Financial and Quantitative Analysis*, 52(4), pp.1375-1402.

Bergmeir, C., Hyndman, R.J. and Koo, B., 2018. A note on the validity of cross-validation for evaluating autoregressive time series prediction. *Computational Statistics & Data Analysis*, 120, pp.70-83.

Blume, L., Easley, D. and O'hara, M., 1994. Market statistics and technical analysis: The role of volume. *The journal of finance*, 49(1), pp.153-181.

Borovykh, A., Bohte, S. and Oosterlee, C.W., 2017. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*.

Bouchaud, J.P., Bonart, J., Donier, J. and Gould, M., 2018. *Trades, quotes and prices: financial markets under the microscope.* Cambridge University Press.

Brogaard, J., Carrion, A., Moyaert, T., Riordan, R., Shkilko, A. and Sokolov, K., 2018. High frequency trading and extreme price movements. *Journal of Financial Economics*, 128(2), pp.253-265.

Bulkowski, T.N., 2012. *Encyclopedia of candlestick charts*. John Wiley & Sons.

Caporale, G.M. and Plastun, A., 2017. Price gaps: Another market anomaly?. *Investment Analysts Journal*, 46(4), pp.279-293.

Chaboud, A.P., Chiquoine, B., Hjalmarsson, E. and Vega, C., 2014. Rise of the machines: Algorithmic trading in the foreign exchange market. *The Journal of Finance*, 69(5), pp.2045-2084.

Chan, J., 2019. Technical Indicators: Market Technicians Trading Tools. In: *Automation of Trading Machine for Traders* (pp. 19-36). Palgrave Pivot, Singapore.

Chebotarev, Y., 2011. *Trading robots on the Russian Stock Market*. Smart Book.

Chen, C. and Liao, C., 2018, May. Short term intraday trading of futures market analysis. In: *2018 33rd Youth Academic Annual Conference of Chinese Association of Automation (YAC)* (pp. 549-554). IEEE.

Chen, J. and Tsai, Y., 2020. *Encoding candlesticks as images for pattern classification using convolutional neural networks*. Financial Innovation, 6(1), pp.1-19.

Chen, L., 2022. *Keras.js Documentation.* [Online] Available at: <https://transcranial.github.io/keras-js-docs/> [Accessed 27 April 2022].

Chen, S.H., Kaboudan, M. and Du, Y.R. eds., 2018. *The Oxford handbook of computational economics and finance*. Oxford University Press.

Chollet, F., 2022. *Keras: The Python Deep Learning library.* [Online] Available at: <https://keras.io/> [Accessed 29 March 2022].

Chong, E., Han, C. and Park, F.C., 2017. Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies. *Expert Systems with Applications*, 83, pp.187-205.

Cho, T., 2020. Turning alphas into betas: Arbitrage and endogenous risk. *Journal of Financial Economics*, 137(2), pp.550-570.

Chou, J.S., Truong, D.N. and Le, T.L., 2020. Interval forecasting of financial time series by accelerated particle swarm-optimized multi-output machine learning system. *IEEE Access*, *8*, pp.14798-14808.

Chung, J., Gulcehre, C., Cho, K. and Bengio, Y., 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555.*

Carapuco, J., Neves, R. and Horta, N., 2018. Reinforcement learning applied to Forex trading*. Applied Soft Computing*, 73, pp.783-794.

Cournapeau, D., 2022. *Scikit-Learn. Machine Learning in Python.* [Online] Available at: <http://scikit-learn.org/stable/> [Accessed 27 April 2022].

CQG Inc, 2022. *CQG.* [Online] Available at: <http://www.cqg.com/> [Accessed 27 March 2022].

Dai, B., Marshall, B.R., Nguyen, N.H. and Visaltanachoti, N., 2020. Risk reduction using trailing stop-loss rules. *International Review of Finance*.

De Prado, M.L., 2018. *Advances in financial machine learning.* John Wiley & Sons.

Dinh, T.A. and Kwon, Y.K., 2018, September. An empirical study on importance of modeling parameters and trading volume-based features in daily stock trading using neural networks. In *Informatics, Multidisciplinary Digital Publishing Institute*, 5(3), p. 36.

Dixon, M., 2018. Sequence classification of the limit order book using recurrent neural networks. *Journal of computational science*, 24, pp.277-286.

Dixon, M., Klabjan, D. and Bang, J.H., 2017. Classification-based financial markets prediction using deep neural networks. *Algorithmic Finance*, 6(3-4), pp.67-77.

Doering, J., Fairbank, M. and Markose, S., 2017. Convolutional neural networks applied to high-frequency market microstructure forecasting. In: *2017 9th Computer Science and Electronic Engineering (CEEC)* (pp. 31-36). IEEE.

Dukascopy Swiss Banking Group, 2022a. *FIX API Programming Guide.* [Online] Available at: <https://www.dukascopy.com/swiss/english/forex/api/fix-api/> [Accessed 27 March2022].

Dukascopy Swiss Banking Group, 2022b. *Historical Data Feed.* [Online] Available at: <https://www.dukascopy.com/swiss/english/marketwatch/historical/> [Accessed 27 March 2022].

Dwyer, G.P. and Hafer, R.W. eds., 2013. *The stock market: Bubbles, volatility, and chaos*. Springer Science & Business Media.

Dymova, L., Sevastjanov, P. and Kaczmarek, K., 2016. A Forex trading expert system based on a new approach to the rule-base evidential reasoning. *Expert Systems with Application*s, 51, pp.1-13.

Edwards, R. and Magee, J., 1948. *Technical Analysis of Stock Trends*.

Eissa, M., 2016. *Introduction to Machine Learning.* [Online] Available at: <https://www.codeproject.com/Articles/1146582/Introduction-to-Machine-Learning> [Accessed 28 March 2022].

Eraslan, S., 2019. Asymmetric arbitrage trading on offshore and onshore renminbi markets. *Empirical Economics*, 57(5), pp.1653-1675.

Euromoney, 2018. *Euromoney FX Survey 2018.* [Online] Available at: <https://www.euromoney.com/article/b18f5s6kqtcr6m/euromoney-fx-survey-2018-press-release> [Accessed 29 March 2022].

Fafula, A. and Peterson, R., 2015. Predicting Global Economic Activity with Media Analytics. *Available on Research Gate.*

Fang, F., Chung, W., Ventre, C., Basios, M., Kanthan, L., Li, L. and Wu, F., 2021. Ascertaining price formation in cryptocurrency markets with machine learning. *The European Journal of Finance,* pp.1-23.

Fidelity Investments Inc., 2020. *Stay invested: Do not risk missing the market's best day.* [Online] Available at: <https://www.fidelity.com//bin-public/060_www_fidelity_com/documents/dont-miss-best-days.pdf> [Accessed 29 March 2022].

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.

Flach, P., 2012. *Machine learning: the art and science of algorithms that make sense of data.* Cambridge University Press.

Frost, A., 2005. *Elliott Wave principle: Key to Market Behavior.* Delhi: Elliott Wave International.

Galeshchuk, S. and Mukherjee, S., 2017. Deep networks for predicting direction of change in foreign exchange rates. *Intelligent Systems in Accounting, Finance and Management*, 24(4), pp.100-110.

Gerrard, R., Hiabu, M., Nielsen, J.P. and Vodička, P., 2020. Long-term real dynamic investment planning. *Insurance: Mathematics and Economics*, 92, pp.90-103.

Gitman, L.J., Joehnk, M.D., Smart, S. and Juchau, R.H., 2015. *Fundamentals of investing*. Pearson Higher Education AU.

Göçken, M., Özçalıcı, M., Boru, A. and Dosdoğru, A.T., 2016. Integrating metaheuristics and artificial neural networks for improved stock price prediction. *Expert Systems with Applications*, 44, pp.320-331.

Goel, A., Sharma, A. and Mehra, A., 2019. Robust optimization of mixed CVaR STARR ratio using copulas. *Journal of Computational and Applied Mathematics*, 347, pp.62-83.

Goodfellow, I., Bengio, Y., Courville, A. and Bengio, Y., 2016. *Deep learning*, 1(2). Cambridge: MIT press.

Google Colaboratory, 2022. *Colaboratory.* [Online] Available at: <https://colab.research.google.com/notebooks/welcome.ipynb> [Accessed 29 March 2022].

Google LLC, 2022. *TensorFlow. An open-source software library for Machine Intelligence.* [Online] Available at: <https://www.tensorflow.org/> [Accessed 29 March 2022].

Grant, K.L., 2011. *Trading risk: Enhanced profitability through risk control*. John Wiley & Sons.

Gruner, S. and Bartelmann, M., 2020. On Computer Simulation, with Particular Regard to their Application in Contemporary Astrophysics. *Cosmos and History: The Journal of Natural and Social Philosophy*, 16(2), pp.537-581.

Grünke, P., 2019. Chess, Artificial Intelligence, and Epistemic Opacity. *Információs Társadalom*, 19, pp.1-7.

Guasoni, P., Mishura, Y. and Rásonyi, M., 2021. High-frequency trading with fractional Brownian motion. *Finance and Stochastics*, 25(2), pp.277-310.

Gulli, A. and Pal, S., 2017. *Deep learning with Keras.* Packt Publishing Ltd.

Gurav, U. and Sidnal, N., 2018. Predict Stock Market Behavior: Role of Machine Learning Algorithms. In: *Intelligent Computing and Information and Communication* (pp. 383-394). Springer, Singapore.

Hagan, M.T., Howard, B.D., Beale, M.H. and DeJesús, O., 2014. *Neural Network Design.* 2nd ed. ISBN-10, 971732116.

Halls-Moore, M., 2018. *Advanced Algorithmic Trading.* [Online] Available at: <https://www.quantstart.com/advanced-algorithmic-trading-ebook> [Accessed 29 March 2022].

Halls-Moore, M., 2019. *Successful Algorithmic trading.* [Online] Available at: <https://www.quantstart.com/successful-algorithmic-trading-ebook> [Accessed 29 March 2022].

Han, X. and Li, Y., 2017. Can investor sentiment be a momentum time-series predictor? Evidence from China. *Journal of Empirical Finance*, 42, pp.212-239.

Heinz, A., Jamaloodeen, M., Saxena, A. and Pollacia, L., 2021. Bullish and Bearish Engulfing Japanese Candlestick patterns: A statistical analysis on the S&P 500 index. *The Quarterly Review of Economics and Finance*, 79, pp.221-244.

Heinz, S., 2017. *A simple deep learning model for stock price prediction using TensorFlow.* [Online] Available at: <https://medium.com/mlreview/a-simple-deep-learning-model-for-stock-price-prediction-using-tensorflow-30505541d877> [Accessed 29 March 2022].

Hernández, J.A.C., Hoyos-Reyes, L.F. and Martínez-Preece, M.R., 2018. A hybrid alpha-stable model development for high frequency trading markets. *Contaduría y Administración*, 63(4), pp.31-32.

Hilpisch, Y., 2018. *Python for finance: mastering data-driven finance.* O'Reilly Media.

Hilpisch, Y., 2020. *Artificial Intelligence in Finance.* O'Reilly Media.

Hirano, M., Izumi, K., Matsushima, H. and Sakaji, H., 2020. *Comparing Actual and Simulated HFT Traders' Behavior for Agent Design. Journal of Artificial Societies and Social Simulation,* 23(3).

Hochreiter, S., 1991. *Fundamental Deep Learning Problem.* [Online] Available at: <http://people.idsia.ch/~juergen/fundamentaldeeplearningproblem.html> [Accessed 29 March 2022].

Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. *Neural computation*, 9(8), pp.1735-1780.

Hsu, P.H., Taylor, M.P. and Wang, Z., 2016. Technical trading: Is it still beating the foreign exchange market?. *Journal of International Economics*, 102, pp.188-208.

Hull, J., 2014. *Options, futures, and other derivatives.* 9th ed., Pearson Education.

Hu, Y.J. and Lin, S.J., 2019, February. Deep reinforcement learning for optimizing finance portfolio management. In: *2019 Amity International Conference on Artificial Intelligence (AICAI)* (pp. 14-20). IEEE.

Ican, O. and Celik, T.B., 2017. Stock market prediction performance of neural networks: A literature review. *International Journal of Economics and Finance*, 9(11), pp.100-108.

Interactive Brokers Group, 2022. *FX Trader.* [Online] Available at: <https://www.interactivebrokers.com/en/index.php?f=796> [Accessed 28 March 2022].

Interactive Brokers LLC, 2022. *Interactive brokers.* [Online] Available at: <https://www.interactivebrokers.com/en/home.php> [Accessed 27 March 2022].

Interactive Brokers, 2022. *Trader Workstation API.* [Online] Available at: <https://interactivebrokers.github.io/tws-api/index.html> [Accessed 29 March 2022].

Jansen, S., 2020. *Machine Learning for Algorithmic Trading: Predictive Models to Extract Signals from Market and Alternative Data for Systematic Trading Strategies with Python,* 2nd ed. Packt Publishing Limited.

Jarrah, M. and Salim, N., 2019. A recurrent neural network and a discrete wavelet transform to predict the Saudi stock price trends. *International Journal of Advanced Computer Science and Applications*, 10(4), pp.155-162.

Jarrow, R.A. and Chatterjea, A., 2019. Arbitrage and Trading. *World Scientific Book Chapters*, pp.112-128.

Jiang, W., 2020. Applications of deep learning in stock market prediction: recent progress. *arXiv preprint arXiv:2003.01859.*

Jiang, Z. and Liang, J., 2017, September. Cryptocurrency portfolio management with deep reinforcement learning. *In 2017 Intelligent Systems Conference* (pp. 905-913). IEEE.

Jitpakdee, P. and Pravithana, N., 2017. A novel method of trend lines generation using hough transform method. *International Journal of Computing Academic Research*, 6, pp.125-135.

Johnson, T., 2017. The Ascendancy of Financial Economics. In: *Ethics in Quantitative Finance* (pp. 195-220). Palgrave Macmillan, Cham.

Karp, A. and Van Vuuren, G., 2019. Investment implications of the fractal market hypothesis. *Annals of Financial Economics*, 14(01), p.1950001.

Karpathy, A., 2015. *The Unreasonable Effectiveness of Recurrent Neural Networks.* [Online] Available at: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/> [Accessed 29 March 2022].

Kaufman, P.J., 2013. *Trading Systems and Methods* (Vol. 591). John Wiley & Sons.

Khedr, A.E. and Yaseen, N., 2017. Predicting stock market behavior using data mining technique and news sentiment analysis. *International Journal of Intelligent Systems and Applications,* 9(7), p.22.

Kim, Y.S., Rachev, S.T., Bianchi, M.L., Mitov, I. and Fabozzi, F.J., 2011. Time series analysis for financial market meltdowns. *Journal of Banking & Finance,* 35(8), pp.1879-1891.

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.

Ku, S., Lee, C., Chang, W. and Song, J.W., 2020. Fractal structure in the S&P500: A correlation-based threshold network approach. *Chaos, Solitons & Fractals*, 137, p.109848.

Larsen, J., 2010. *Predicting stock prices using technical analysis and machine learning.* [Online] Available at: <https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/252181> [Accessed 29 March 2022].

LeDell, E., 2016. *Deep Learning via Multilayer Perceptrons (MLPs).* [Online] Available at: <https://htmlpreview.github.io/?https://github.com/ledell/sldm4-h2o/blob/master/sldm4-deeplearning-h2o.html> [Accessed 29 March 2022].

Lee, C.Y. and Soo, V.W., 2017, December. Predict stock price with financial news based on recurrent convolutional neural networks. In: *2017 Conference on Technologies and Applications of Artificial Intelligence (TAAI)* (pp. 160-165). IEEE.

Leung, T. and Li, X., 2015. Optimal mean reversion trading with transaction costs and stop-loss exit. *International Journal of Theoretical and Applied Finance*, 18(03), p.1550020.

Lien, K., 2006. *Day trading the currency market: Technical and fundamental strategies to profit from market swings,* (260). John Wiley & Sons.

Lien, K., 2008. *Day trading and swing trading the currency market: technical and fundamental strategies to profit from market moves* (Vol. 431). John Wiley & Sons.

Lin, T.C., 2013. *The new financial industry*. Ala. L. Rev., 65, p.567.

Linh, T., 2019. Applying Volume Spread Analysis To The Analysis Of Stock Price Spread On Ho Chi Minh City Stock Exchange. *Journal of Economic Development*, pp.12-18.

Liu, J., Lu, Z. and Du, W., 2019, January. Combining enterprise knowledge graph and news sentiment analysis for stock price prediction. In: *Proceedings of the 52nd Hawaii International Conference on System Sciences.*

Li, Y. and Dai, W., 2020. Bitcoin price forecasting method based on CNN-LSTM hybrid neural network model. *The Journal of Engineering*, 2020(13), pp.344-347.

Li, Z. and Tam, V., 2018. A machine learning view on momentum and reversal trading. *Algorithms*, 11(11), p.170.

Livermore, J., 1940. *How to trade in stocks: The Livermore formula for combining time element and price.* Laurus-Lexecon Kft.

Maaten, L. and Hinton, G., 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9. pp. 2579-2605.

Mandelbrot, B., 1983. *The fractal geometry of nature.* New York: WH freeman.

Mandelbrot, B.B. and Hudson, R.L., 2010. *The (mis) behaviour of markets: a fractal view of risk, ruin and reward.* Profile books.

Mane, D.T. and Kulkarni, U.V., 2020. A survey on supervised convolutional neural network and its major applications. In: *Deep Learning and Neural Networks: Concepts, Methodologies, Tools, and Applications, IGI Global.* (pp. 1058-1071).

Maneejuk, P. and Srichaikul, W., 2021. Forecasting foreign exchange markets: further evidence using machine learning models. *Soft Computing*, 25(12), pp.7887-7898.

Mashnin, T., 2016. *Programming with MQL-5. Advanced using of Trading Platform Meta-Trader-5.* Publishing Solutions.

Mei, S., Jiang, R., Ji, J., Sun, J., Peng, Y. and Zhang, Y., 2017, November. Invariant feature extraction for image classification via multi-channel convolutional neural network. In *2017 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)* (pp. 491-495). IEEE.

MetaQuotes Software Corp, 2022a. *MetaTrader 5.* [Online] Available at: <http://www.metatrader5.com/> [Accessed 28 March 2022].

MetaQuotes Software Corp, 2022b. *MQL5 reference - Language Basics.* [Online] Available at: <https://www.mql5.com/en/docs/basis> [Accessed 28 March 2022].

Miles, M.B. and Huberman, A.M., 1984. Qualitative data analysis: A sourcebook of new methods. In: *Qualitative data analysis: a sourcebook of new methods* (pp. 263-263).

Milke, V., Luca, C. and Wilson, G., 2017, May. Minimisation of parameters for optimisation of Algorithmic Trading Systems. In: *2017 International Conference on Optimization of Electrical and Electronic Equipment (OPTIM) & 2017 Intl Aegean Conference on Electrical Machines and Power Electronics (ACEMP)* (pp. 1114-1119). IEEE

Milke, V., Luca, C., Wilson, G. and Fatima, A., 2020. Using Convolutional Neural Networks and Raw Data to Model Intraday Trading Market Behaviour. In: *12th International Conference on Agents and Artificial Intelligence (ICAART)* (2), (pp. 224-231).

Mills, T.C. and Markellos, R.N., 2008. *The econometric modelling of financial time series.* Cambridge University Press.

Moghaddam, A.H. and Momtazi, S., 2021. Image processing meets time series analysis: Predicting Forex profitable technical pattern positions. *Applied Soft Computing*, 108, p.107460.

Morris, G.L., 2006. *Candlestick Charting Explained: Timeless Techniques for Trading Stocks and Futures: Timeless Techniques for Trading stocks and Sutures.* McGraw Hill Professional.

Moscow Exchange, 2018. *SPECTRA Plaza-2 gate.* [Online] Available at: <https://fs.moex.com/f/9096/iqs-p2gate-en.pdf> [Accessed 29 March 2022].

Moscow Exchange, 2022. *Moscow Exchange.* [Online] Available at: <http://moex.com/en/> [Accessed 28 March 2022].

Moskowitz, T.J., Ooi, Y.H. and Pedersen, L.H., 2012. Time series momentum. *Journal of financial economics*, 104(2), pp.228-250.

Mueller, A. and Guido, S., 2017. *Introduction to machine learning using Python*. Publishing House Gevissta Publ., Moscow.

Murphy, J.J., 2009. *The visual investor: how to spot market trends* (443). John Wiley &Sons.

Nakajima, T., 2019. Expectations for statistical arbitrage in energy futures markets. *Journal of Risk and Financial Management*, 12(1), p.14.

Namdari, A. and Li, Z.S., 2018, June. Integrating fundamental and technical analysis of stock market through multi-layer perceptron. In: *2018 IEEE Technology and Engineering Management Conference (TEMSCON)* (pp. 1-6). IEEE.

Naranjo, R. and Santos, M., 2016, October. Fuzzy candlesticks forecasting using pattern recognition for stock markets. *In International Joint Conference SOCO'16-CISIS'16-ICEUTE'16* (pp. 323-333). Springer, Cham.

Nazário, R.T.F., e Silva, J.L., Sobreiro, V.A. and Kimura, H., 2017. A literature review of technical analysis on stock markets. *The Quarterly Review of Economics and Finance*, 66, pp.115-126.

Nelson, D.M., Pereira, A.C. and de Oliveira, R.A., 2017, May. Stock market's price movement prediction with LSTM neural networks. In: *2017 International joint conference on neural networks (IJCNN)* (pp. 1419-1426). IEEE.

Neumann, J. and Morgenstern, O., 2020. *Theory of Games and Economic Behavior.* DIANA.

Nison, S., 1994. *Beyond candlesticks: New Japanese charting techniques revealed* (Vol. 56). John Wiley & Sons.

Nison, S., 2004. *The candlestick course*. John Wiley & Sons.

Nison, S., 2011. *Strategies for Profiting with Japanese Candlestick Charts* (Vol. 132). John Wiley & Sons.

Noonan, L., 2018. JPMorgan's requirement for new staff: coding lessons. *Financial Times*, 8 October, p. 29.

O'Neil, W.J, 2005. *How to make money selling stocks short.* John Wiley & Sons.

OANDA Corporation, 2022. *OANDA.* [Online] Available at: <https://www.oanda.com/> [Accessed 29 March 2022].

Olah, C., 2015. *Understanding LSTM Networks.* [Online] Available at: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/> [Accessed 29 March 2022].

Orimoloye, L.O., Sung, M.C., Ma, T. and Johnson, J.E., 2020. Comparing the effectiveness of deep feedforward neural networks and shallow architectures for predicting stock price indices. *Expert Systems with Applications*, 139, p.112828.

Park, Y.S., Konge, L. and Artino, A.R., 2020. The positivism paradigm of research. *Academic Medicine*, 95(5), pp.690-694.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L. and Lerer, A., 2017. Automatic differentiation in pytorch. In *Long Beach, California, USA, Autodiff Workshop.*

Patel, J., Shah, S., Thakkar, P. and Kotecha, K., 2015. Predicting stock market index using fusion of machine learning techniques. *Expert Systems with Applications*, 42(4), pp.2162-2172.

Petukhina, A.A., Reule, R.C. and Härdle, W.K., 2021. Rise of the machines? Intraday high-frequency trading patterns of cryptocurrencies. *The European Journal of Finance,* 27(1-2), pp.8-30.

Pichl, L., Nan, Z. and Kaizoji, T., 2020. Time series analysis of ether cryptocurrency prices: Efficiency, predictability, and arbitrage on exchange rates. In: *Advanced studies of financial technologies and cryptocurrency markets* (pp. 183-196). Springer, Singapore.

Poon, S.H., 2005. *A practical guide to forecasting financial market volatility*. John Wiley & Sons.

Portugal, I., Alencar, P. and Cowan, D., 2016. A survey on domain-specific languages for machine learning in big data. *arXiv preprint arXiv:1602.07637.*

Qiu, M. and Song, Y., 2016. Predicting the direction of stock market index movement using an optimized artificial neural network model. *Public Library of Science,* 19 May, 11(5), p.e0155133.

Rampasek, L. and Goldenberg, A., 2016. TensorFlow: biology's gateway to deep learning?. *Cell systems*, 2(1), pp.12-14.

Renaissance Technologies LLC, 2022. *Renaissance.* [Online] Available at: <https://www.rentec.com/Home.action?index=true> [Accessed 29 March 2022].

Rundo, F., Trenta, F., di Stallo, A.L. and Battiato, S., 2019. Machine learning for quantitative finance applications: A survey. *Applied Sciences*, 9(24), p.5574

S&P Dow Jones Indices, 2022. *S&P 500.* [Online] Available at: <https://us.spindices.com/indices/equity/sp-500> [Accessed 29 March 2022].

Saha, S., 2018. *A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way.* [Online] Available at: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53> [Accessed 29 March 2022].

Sandubete, J. and Escot, L., 2020. *Chaotic signals inside some tick-by-tick financial time series*. Chaos, Solitons & Fractals, 137, p.109852.

Sang, C. and Di Pierro, M., 2019. Improving trading technical analysis with tensorflow long short-term memory (LSTM) neural network. *The Journal of Finance and Data Science*, 5(1), pp.1-11.

Schwager, J.D., 1995. *Technical analysis* (Vol. 43). John Wiley & Sons.

Schwager, J.D., 2017. *A Complete Guide to the Futures Market: Technical Analysis, Trading Systems, Fundamental Analysis, Options, Spreads, and Trading Principles*. John Wiley & Sons.

Sezer, O.B., Gudelek, M.U. and Ozbayoglu, A.M., 2020. Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. *Applied Soft Computing,* 90, p.106181.

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*, 70, pp.525-538.

Sezer, O.B., Ozbayoglu, A.M. and Dogdu, E., 2017. An artificial neural network-based stock trading system using technical analysis and big data framework. In: *proceedings of the southeast conference, April,* (pp. 223-226).

Sharpe, W.F., 1998. The sharpe ratio. *Streetwise–the Best of the Journal of Portfolio Management,* pp.169-185.

Shreve, S., 2005. *Stochastic calculus for finance I: the binomial asset pricing model.* Springer Science & Business Media.

Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556.*

Sirignano, J.A., 2019. Deep learning for limit order books. *Quantitative Finance*, 19(4), pp.549-570.

Solabuto, N., 2011. *Trading systems and methods.* Piter.

Sozikin, A., 2022. *Transfer Learning with Keras.* [Online] Available at: <https://www.asozykin.ru/courses/nnpython> [Accessed 29 March 2022].

Spark, 2017. *A long journey with 1.5m of flat photos.* [Online] Available at: <https://spark-in.me/post/neural-net-used-for-flats> [Accessed 29 March 2021].

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research,* 15(1), pp.1929-1958.

Stasiak, M.D., 2020. Candlestick—The Main Mistake of Economy Research in High Frequency Markets. *International Journal of Financial Studies*, 8(4), p.59.

Stock Sharp, 2020. *Stock Sharp. Free open source algorithmic trading and quantitative trading platrorm.* [Online] Available at: <http://stocksharp.com/products/> [Accessed 29 March 2022].

Stock Sharp, 2021. *S#.Data.* [Online] Available at: <https://stocksharp.com/robot/hydra/> [Accessed 27 March 2021].

Tabar, S., Sharma, S. and Volkman, D., 2020. A new method for predicting stock market crashes using classification and artificial neural networks. *International Journal of Business and Data Analytics*, 1(3), pp.203-217.

Tan, Y., Shi, Y., Buarque, F., Gelbukh, A., Das, S. and Engelbrecht, A., 2015. Advances in swarm and computational intelligence, In: *6th international conferens ICSI.* Beijing, China, Springer.

The Economist, 2019. The stockmarket is now run by computers, algorithms and passive managers. *The Economist*, 5th October 2019. [Online] Available at: <https://www.economist.com/briefing/2019/10/05/the-stockmarket-is-now-run-by-computers-algorithms-and-passive-managers?mod=djemDailyShot> [Accessed 20 July 2022].

Tilen, C.U.K. and Van Waeyenberge, A., 2018. European legal framework for algorithmic and high frequency trading (Mifid 2 and MAR): A global approach to managing the risks of the modern trading paradigm. *European Journal of Risk Regulation*, 9(1), pp.146-153.

TIOBE, 2021. *TIOBE Index for March 2021.* [Online] Available at: <https://www.tiobe.com/tiobe-index/> [Accessed 28 March 2022].

Troiano, L., Villa, E.M. and Loia, V., 2018. Replicating a trading strategy by means of LSTM for financial industry applications. *IEEE transactions on industrial informatics,* 14(7), pp.3226-3234.

Tsay, R., 2005. *Analysis of financial time series.* John wiley & sons.

TSLab, 2022. *TsLab - Visual Trading Environment.* [Online] Available at: <https://www.tslab.pro/en-US/home> [Accessed 29 March 2022].

U.S.Securities and Exchange Commission (SEC), 2022. *Day Trading.* [Online] Available at: <https://www.investor.gov/introduction-investing/investing-basics/glossary/day-trading> [Accessed 29 March 2022].

Udagawa, Y., 2018. Design and Implementation of Candlestick Chart Retrieval Algorithm for Predicting Stock Price Trend. In: *The Fourth International Conference on Big Data, Small Data, Linked Data and Open Data (ALLDATA 2018)* (pp. 19-25).

University of Oxford, 2021. *Very Deep Convolutional Networks for Large-Scale Visual Recognition.* [Online] Available at: <http://www.robots.ox.ac.uk/~vgg/research/very_deep/> [Accessed 29 March 2022].

Van Kervel, V. and Menkveld, A.J., 2019. High-frequency trading around large institutional orders. *The Journal of Finance*, 74(3), pp.1091-1137.

Van, A., 2022. *OS.Engine.* [Online] Available at: <https://o-s-a.net/> [Accessed 29 March 2022].

Vezeris, D., Kyrgos, T. and Schinas, C., 2018. Take Profit and Stop Loss Trading Strategies Comparison in Combination with an MACD Trading System. *Journal of Risk and Financial Management*, 11(3), p.56.

Wang, J., Wang, J., Fang, W. and Niu, H., 2016. Financial time series prediction using elman recurrent random neural networks. *Computational intelligence & neuroscience, 2016.*

Wang, P., Ho, T. and Li, Y., 2020. The Price-Volume Relationship of the Shanghai Stock Index: Structural Change and the Threshold Effect of Volatility. *Sustainability*, 12(8), p.3322.

Wang, X., 2020. Valuation of Asian options with default risk under GARCH models. *International Review of Economics & Finance*, 70, pp.27-40.

Wang, Y., Li, Q., Huang, Z. and Li, J., 2019, June. EAN: Event attention network for stock price trend prediction based on sentimental embedding. In: *Proceedings of the 10th ACM conference on web science* (pp. 311-320).

Weiming, J.M., 2019. *Mastering Python for Finance: Implement advanced state-of-the-art financial statistical applications using Python*. Packt Publishing Ltd.

Weissman, R.L., 2005. *Mechanical trading systems: Pairing trader psychology with technical analysis*. John Wiley & Sons.

Weng, B., Ahmed, M.A. and Megahed, F.M., 2017. Stock market one-day ahead movement prediction using disparate data sources. *Expert Systems with Applications*, 79, pp.153-163.

Wilmott, P., 2013. *Paul wilmott on quantitative finance.* 2 ed. John Wiley & Sons.

Wirsansky, E., 2020. *Hands-On Genetic Algorithms with Python: Applying genetic algorithms to solve real-world deep learning and artificial intelligence problems.* Birmingham-Mumbai: Packt.

Wu, X., Chen, H., Wang, J., Troiano, L., Loia, V. and Fujita, H., 2020a. Adaptive stock trading strategies with deep reinforcement learning methods. *Information Sciences*, 538, pp.142-158.

Wu, J.M.T., Wu, M.E., Hung, P.J., Hassan, M.M. and Fortino, G., 2020b. Convert index trading to option strategies via LSTM architecture. *Neural Computing and Applications*, pp.1-18.

Yan, Y., 2017. *Python for Finance.* 2 ed. Packt Publishing Ltd.

Yıldırım, D.C., Toroslu, I.H. and Fiore, U., 2021. Forecasting directional movement of Forex data using LSTM with technical and macroeconomic indicators. *Financial Innovation*, 7(1), pp.1-36

Zhang, Z., Zohren, S. and Roberts, S., 2019. Deeplob: Deep convolutional neural networks for limit order books. *IEEE Transactions on Signal Processing*, 67(11), pp.3001-3012.

Appendix I. Results of experiments on Prediction of Forex EUR/USD Tick Rates

N	Year	Batch size	Epochs number	Time per Epoch, (seconds) (Average)	Sample Absolute Error Bid, %	Sample Absolute Error Ask, %	Mean Absolute Error, MAE (Validation)	Mean Absolute Error, MAE (Testing, Average)	Mean Squared Error, MSE (Testing, Average)
1	2	3	4	5	6	7	8	9	10
1	2018	15	7	208	6.481	4.382	0.0205	0.0295	0.00181
2	2018	15	7	208	2.256	0.477	0.0205	0.0295	0.00181
3	2018	15	7	208	1.534	0.566	0.0205	0.0295	0.00181
4	2018	15	7	208	0.291	1.281	0.0205	0.0295	0.00181
5	2018	15	7	208	18.712	17.943	0.0205	0.0295	0.00181
6	2018	15	7	208	1.004	4.355	0.0205	0.0295	0.00181
7	2018	15	7	208	0.854	1.246	0.0205	0.0295	0.00181
8	2018	15	7	208	2.894	7.429	0.0205	0.0295	0.00181
9	2018	15	7	208	1.401	3.886	0.0205	0.0295	0.00181
10	2018	15	7	208	1.061	0.275	0.0205	0.0295	0.00181
11	2019	15	7	203	1.761	2.276	0.0214	0.0299	0.00186
12	2019	15	7	203	4.801	1.716	0.0214	0.0299	0.00186
13	2019	15	7	203	0.158	0.168	0.0214	0.0299	0.00186
14	2019	15	7	203	7.032	6.489	0.0214	0.0299	0.00186
15	2019	15	7	203	0.836	0.489	0.0214	0.0299	0.00186
16	2019	15	7	203	1.411	1.839	0.0214	0.0299	0.00186
17	2019	15	7	203	12.208	8.636	0.0214	0.0299	0.00186
18	2019	15	7	203	6.461	8.449	0.0214	0.0299	0.00186
19	2019	15	7	203	2.824	1.501	0.0214	0.0299	0.00186
20	2019	15	7	203	2.823	1.714	0.0214	0.0299	0.00186
21	2020	15	5	207	3.267	2.184	0.0228	0.0326	0.00214
22	2020	15	5	207	2.105	6.419	0.0228	0.0326	0.00214
23	2020	15	5	207	0.789	6.087	0.0228	0.0326	0.00214
24	2020	15	5	207	8.238	5.491	0.0228	0.0326	0.00214
25	2020	15	5	207	2.806	0.807	0.0228	0.0326	0.00214
26	2020	15	5	207	0.716	2.299	0.0228	0.0326	0.00214
27	2020	15	5	207	2.308	1.894	0.0228	0.0326	0.00214
28	2020	15	5	207	0.487	1.221	0.0228	0.0326	0.00214
29	2020	15	5	207	12.889	8.035	0.0228	0.0326	0.00214
30	2020	15	5	207	1.591	2.521	0.0228	0.0326	0.00214
31	2018	30	9	156	1.491	2.406	0.0203	0.0289	0.00176
32	2018	30	9	156	4.782	6.904	0.0203	0.0289	0.00176
33	2018	30	9	156	0.451	0.315	0.0203	0.0289	0.00176
34	2018	30	9	156	4.091	0.394	0.0203	0.0289	0.00176
35	2018	30	9	156	5.164	7.337	0.0203	0.0289	0.00176
36	2018	30	9	156	15.954	16.412	0.0203	0.0289	0.00176

37	2018	30	9	156	1.224	1.971	0.0203	0.0289	0.00176
38	2018	30	9	156	2.203	0.389	0.0203	0.0289	0.00176
39	2018	30	9	156	2.538	5.049	0.0203	0.0289	0.00176
40	2018	30	9	156	4.804	6.144	0.0203	0.0289	0.00176
41	2019	30	9	132	0.534	2.182	0.0208	0.0297	0.00185
42	2019	30	9	132	0.156	1.162	0.0208	0.0297	0.00185
43	2019	30	9	132	0.923	1.597	0.0208	0.0297	0.00185
44	2019	30	9	132	1.771	3.195	0.0208	0.0297	0.00185
45	2019	30	9	132	5.051	4.639	0.0208	0.0297	0.00185
46	2019	30	9	132	0.873	1.499	0.0208	0.0297	0.00185
47	2019	30	9	132	16.575	16.048	0.0208	0.0297	0.00185
48	2019	30	9	132	0.598	1.432	0.0208	0.0297	0.00185
49	2019	30	9	132	7.288	7.007	0.0208	0.0297	0.00185
50	2019	30	9	132	3.892	7.191	0.0208	0.0297	0.00185
51	2020	30	7	136	0.829	0.133	0.0221	0.0313	0.00203
52	2020	30	7	136	5.931	9.201	0.0221	0.0313	0.00203
53	2020	30	7	136	0.138	0.581	0.0221	0.0313	0.00203
54	2020	30	7	136	1.971	7.231	0.0221	0.0313	0.00203
55	2020	30	7	136	2.743	2.831	0.0221	0.0313	0.00203
56	2020	30	7	136	1.457	4.782	0.0221	0.0313	0.00203
57	2020	30	7	136	3.874	3.281	0.0221	0.0313	0.00203
58	2020	30	7	136	5.534	8.397	0.0221	0.0313	0.00203
59	2020	30	7	136	0.244	0.189	0.0221	0.0313	0.00203
60	2020	30	7	136	1.545	2.418	0.0221	0.0313	0.00203
61	2018	50	5	80	2.876	4.733	0.0202	0.0361	0.00188
62	2018	50	5	80	3.441	8.608	0.0202	0.0361	0.00188
63	2018	50	5	80	0.725	0.531	0.0202	0.0361	0.00188
64	2018	50	5	80	5.392	5.492	0.0202	0.0361	0.00188
65	2018	50	5	80	2.959	0.409	0.0202	0.0361	0.00188
66	2018	50	5	80	11.836	8.909	0.0202	0.0361	0.00188
67	2018	50	5	80	1.862	0.981	0.0202	0.0361	0.00188
68	2018	50	5	80	0.919	0.109	0.0202	0.0361	0.00188
69	2018	50	5	80	0.399	2.799	0.0202	0.0361	0.00188
70	2018	50	5	80	0.745	4.965	0.0202	0.0361	0.00188
71	2019	50	7	80	0.423	2.924	0.0209	0.0307	0.00193
72	2019	50	7	80	6.366	10.494	0.0209	0.0307	0.00193
73	2019	50	7	80	0.411	4.509	0.0209	0.0307	0.00193
74	2019	50	7	80	6.308	2.632	0.0209	0.0307	0.00193
75	2019	50	7	80	6.251	6.526	0.0209	0.0307	0.00193
76	2019	50	7	80	1.489	0.701	0.0209	0.0307	0.00193
77	2019	50	7	80	1.476	4.044	0.0209	0.0307	0.00193
78	2019	50	7	80	6.218	3.298	0.0209	0.0307	0.00193
79	2019	50	7	80	0.645	3.909	0.0209	0.0307	0.00193
80	2019	50	7	80	2.896	0.851	0.0209	0.0307	0.00193
81	2020	50	9	81	2.983	2.795	0.0216	0.0314	0.00203
82	2020	50	9	81	10.878	7.464	0.0216	0.0314	0.00203

83	2020	50	9	81	0.152	0.689	0.0216	0.0314	0.00203
84	2020	50	9	81	2.499	3.821	0.0216	0.0314	0.00203
85	2020	50	9	81	0.502	0.604	0.0216	0.0314	0.00203
86	2020	50	9	81	4.456	4.203	0.0216	0.0314	0.00203
87	2020	50	9	81	0.178	1.008	0.0216	0.0314	0.00203
88	2020	50	9	81	1.255	1.607	0.0216	0.0314	0.00203
89	2020	50	9	81	1.751	1.324	0.0216	0.0314	0.00203
90	2020	50	9	81	4.533	3.574	0.0216	0.0314	0.00203
91	2018	80	10	66	1.543	2.024	0.0197	0.0283	0.00169
92	2018	80	10	66	5.376	0.379	0.0197	0.0283	0.00169
93	2018	80	10	66	1.389	0.515	0.0197	0.0283	0.00169
94	2018	80	10	66	8.168	7.333	0.0197	0.0283	0.00169
95	2018	80	10	66	1.046	0.779	0.0197	0.0283	0.00169
96	2018	80	10	66	4.298	2.296	0.0197	0.0283	0.00169
97	2018	80	10	66	11.564	8.917	0.0197	0.0283	0.00169
98	2018	80	10	66	1.816	1.144	0.0197	0.0283	0.00169
99	2018	80	10	66	0.745	1.685	0.0197	0.0283	0.00169
100	2018	80	10	66	1.481	1.685	0.0197	0.0283	0.00169
101	2019	80	7	65	5.702	3.646	0.0213	0.0297	0.00184
102	2019	80	7	65	0.542	0.228	0.0213	0.0297	0.00184
103	2019	80	7	65	2.312	1.864	0.0213	0.0297	0.00184
104	2019	80	7	65	7.477	5.639	0.0213	0.0297	0.00184
105	2019	80	7	65	2.758	0.912	0.0213	0.0297	0.00184
106	2019	80	7	65	0.946	2.269	0.0213	0.0297	0.00184
107	2019	80	7	65	12.622	11.017	0.0213	0.0297	0.00184
108	2019	80	7	65	6.825	5.556	0.0213	0.0297	0.00184
109	2019	80	7	65	5.718	1.412	0.0213	0.0297	0.00184
110	2019	80	7	65	0.248	0.781	0.0213	0.0297	0.00184
111	2020	80	10	65	4.814	5.558	0.0214	0.0308	0.00198
112	2020	80	10	65	2.173	1.206	0.0214	0.0308	0.00198
113	2020	80	10	65	0.231	1.667	0.0214	0.0308	0.00198
114	2020	80	10	65	1.694	1.235	0.0214	0.0308	0.00198
115	2020	80	10	65	2.363	5.711	0.0214	0.0308	0.00198
116	2020	80	10	65	3.491	4.523	0.0214	0.0308	0.00198
117	2020	80	10	65	8.412	7.982	0.0214	0.0308	0.00198
118	2020	80	10	65	1.761	1.371	0.0214	0.0308	0.00198
119	2020	80	10	65	0.743	0.504	0.0214	0.0308	0.00198
120	2020	80	10	65	1.852	2.309	0.0214	0.0308	0.00198
121	2018	100	11	62	2.602	4.945	0.0194	0.0284	0.00171
122	2018	100	11	62	4.131	1.835	0.0194	0.0284	0.00171
123	2018	100	11	62	0.328	0.385	0.0194	0.0284	0.00171
124	2018	100	11	62	9.667	5.331	0.0194	0.0284	0.00171
125	2018	100	11	62	0.379	2.406	0.0194	0.0284	0.00171
126	2018	100	11	62	3.149	3.388	0.0194	0.0284	0.00171
127	2018	100	11	62	0.809	0.946	0.0194	0.0284	0.00171
128	2018	100	11	62	2.142	2.871	0.0194	0.0284	0.00171

129	2018	100	11	62	2.155	1.501	0.0194	0.0284	0.00171
130	2018	100	11	62	5.331	5.223	0.0194	0.0284	0.00171
131	2019	100	14	61	2.312	2.486	0.0206	0.0291	0.00181
132	2019	100	14	61	8.358	4.614	0.0206	0.0291	0.00181
133	2019	100	14	61	2.196	1.509	0.0206	0.0291	0.00181
134	2019	100	14	61	1.755	1.747	0.0206	0.0291	0.00181
135	2019	100	14	61	10.103	3.461	0.0206	0.0291	0.00181
136	2019	100	14	61	2.368	2.559	0.0206	0.0291	0.00181
137	2019	100	14	61	2.844	3.596	0.0206	0.0291	0.00181
138	2019	100	14	61	0.554	0.668	0.0206	0.0291	0.00181
139	2019	100	14	61	0.372	0.684	0.0206	0.0291	0.00181
140	2019	100	14	61	0.489	1.487	0.0206	0.0291	0.00181
141	2020	100	14	63	3.027	1.306	0.0214	0.0309	0.00199
142	2020	100	14	63	1.172	2.708	0.0214	0.0309	0.00199
143	2020	100	14	63	10.143	10.866	0.0214	0.0309	0.00199
144	2020	100	14	63	3.463	2.552	0.0214	0.0309	0.00199
145	2020	100	14	63	2.085	0.549	0.0214	0.0309	0.00199
146	2020	100	14	63	6.799	7.546	0.0214	0.0309	0.00199
147	2020	100	14	63	2.709	3.612	0.0214	0.0309	0.00199
148	2020	100	14	63	2.394	3.992	0.0214	0.0309	0.00199
149	2020	100	14	63	0.774	0.181	0.0214	0.0309	0.00199
150	2020	100	14	63	4.934	5.476	0.0214	0.0309	0.00199

Table 1: Results of the first series of experiments
(Prediction of Forex EUR/USD Tick Rates).

Appendix II. Results of the binary classification experiments on Trend or Flat

N	Batch size	Epochs number	Patience	Time per Epoch, (seconds) (Average)	Convolution kernel	Prediction, Probability, %	Prediction: Trend/Flat	Accuracy (Train), %	Accuracy (Validation), %	Precision (Train), %	Precision (Validation), %	Recall (Train),%	Recall (Validation), %	Loss Function (Validation)	Accuracy (Test), %	Precision (Test), %	Recall (Test), %
1	200	8	30	107	31	73.848	Flat	89.52	85.59	91.43	92.89	96.74	91.47	0.3621	84.578	96.084	87.304
2	200	8	30	107	31	96.901	Trend	89.52	85.59	91.43	92.89	96.74	91.47	0.3621	84.578	96.084	87.304
3	200	8	30	107	31	92.355	Flat	89.52	85.59	91.43	92.89	96.74	91.47	0.3621	84.578	96.084	87.304
4	200	8	30	107	31	89.601	Trend	89.52	85.59	91.43	92.89	96.74	91.47	0.3621	84.578	96.084	87.304
5	200	8	30	107	31	53.851	Trend	89.52	85.59	91.43	92.89	96.74	91.47	0.3621	84.578	96.084	87.304
6	200	8	30	107	31	89.236	Flat	89.52	85.59	91.43	92.89	96.74	91.47	0.3621	84.578	96.084	87.304
7	200	8	30	107	31	51.246	Trend	89.52	85.59	91.43	92.89	96.74	91.47	0.3621	84.578	96.084	87.304
8	200	8	30	107	31	94.417	Trend	89.52	85.59	91.43	92.89	96.74	91.47	0.3621	84.578	96.084	87.304
9	200	8	30	107	31	59.673	Flat	89.52	85.59	91.43	92.89	96.74	91.47	0.3621	84.578	96.084	87.304
10	200	8	30	107	31	77.764	Trend	89.52	85.59	91.43	92.89	96.74	91.47	0.3621	84.578	96.084	87.304
11	150	15	60	115	31	90.934	Flat	89.84	83.52	91.23	92.90	97.42	89.03	0.4034	79.647	96.018	81.946
12	150	15	60	115	31	66.919	Flat	89.84	83.52	91.23	92.90	97.42	89.03	0.4034	79.647	96.018	81.946
13	150	15	60	115	31	90.489	Trend	89.84	83.52	91.23	92.90	97.42	89.03	0.4034	79.647	96.018	81.946
14	150	15	60	115	31	85.748	Trend	89.84	83.52	91.23	92.90	97.42	89.03	0.4034	79.647	96.018	81.946
15	150	15	60	115	31	76.315	Trend	89.84	83.52	91.23	92.90	97.42	89.03	0.4034	79.647	96.018	81.946
16	150	15	60	115	31	79.389	Flat	89.84	83.52	91.23	92.90	97.42	89.03	0.4034	79.647	96.018	81.946
17	150	15	60	115	31	87.332	Flat	89.84	83.52	91.23	92.90	97.42	89.03	0.4034	79.647	96.018	81.946
18	150	15	60	115	31	85.374	Trend	89.84	83.52	91.23	92.90	97.42	89.03	0.4034	79.647	96.018	81.946
19	150	15	60	115	31	75.171	Flat	89.84	83.52	91.23	92.90	97.42	89.03	0.4034	79.647	96.018	81.946
20	150	15	60	115	31	61.575	Trend	89.84	83.52	91.23	92.90	97.42	89.03	0.4034	79.647	96.018	81.946

Continued on next page

N	Batch size	Epochs number	Patience	Time per Epoch, (seconds) (Average)	Convolution kernel	Prediction, Probability, %	Prediction: Trend/Flat	Accuracy (Train), %	Accuracy (Validation), %	Precision (Train), %	Precision (Validation), %	Recall (Train),%	Recall (Validation), %	Loss Function (Validation)	Accuracy (Test), %	Precision (Test), %	Recall (Test), %
21	100	7	30	56	15	87.012	Flat	85.28	80.47	91.29	92.75	97.04	85.64	0.4178	76.413	96.331	78.116
22	100	7	30	56	15	94.133	Trend	89.62	80.47	91.29	92.75	97.04	85.64	0.4178	76.413	96.331	78.116
23	100	7	30	56	15	87.889	Flat	89.62	80.47	91.29	92.75	97.04	85.64	0.4178	76.413	96.331	78.116
24	100	7	30	56	15	88.982	Trend	89.62	80.47	91.29	92.75	97.04	85.64	0.4178	76.413	96.331	78.116
25	100	7	30	56	15	57.920	Flat	85.28	80.47	91.29	92.75	97.04	85.64	0.4178	76.413	96.331	78.116
26	100	7	30	56	15	74.867	Trend	85.28	80.47	91.29	92.75	97.04	85.64	0.4178	76.413	96.331	78.116
27	100	7	30	56	15	70.062	Trend	89.62	80.47	91.29	92.75	97.04	85.64	0.4178	76.413	96.331	78.116
28	100	7	30	56	15	41.282	Flat	89.62	80.47	91.29	92.75	97.04	85.64	0.4178	76.413	96.331	78.116
29	100	7	30	56	15	91.963	Flat	89.62	80.47	91.29	92.75	97.04	85.64	0.4178	76.413	96.331	78.116
30	100	7	30	56	15	92.193	Trend	85.28	80.47	91.29	92.75	97.04	85.64	0.4178	76.413	96.331	78.116
31	50	9	30	122	15	90.322	Trend	89.51	84.24	91.06	92.89	97.20	89.89	0.4012	81.793	96.185	84.148
32	50	9	30	122	15	60.347	Flat	89.51	84.24	91.06	92.89	97.20	89.89	0.4012	81.793	96.185	84.148
33	50	9	30	122	15	81.279	Flat	89.51	84.24	91.06	92.89	97.20	89.89	0.4012	81.793	96.185	84.148
34	50	9	30	122	15	92.735	Flat	89.51	84.24	91.06	92.89	97.20	89.89	0.4012	81.793	96.185	84.148
35	50	9	30	122	15	58.156	Trend	89.51	84.24	91.06	92.89	97.20	89.89	0.4012	81.793	96.185	84.148
36	50	9	30	122	15	88.985	Trend	89.51	84.24	91.06	92.89	97.20	89.89	0.4012	81.793	96.185	84.148
37	50	9	30	122	15	85.255	Flat	89.51	84.24	91.06	92.89	97.20	89.89	0.4012	81.793	96.185	84.148
38	50	9	30	122	15	88.183	Trend	89.51	84.24	91.06	92.89	97.20	89.89	0.4012	81.793	96.185	84.148
39	50	9	30	122	15	90.152	Trend	89.51	84.24	91.06	92.89	97.20	89.89	0.4012	81.793	96.185	84.148
40	50	9	30	122	15	58.416	Flat	89.51	84.24	91.06	92.89	97.20	89.89	0.4012	81.793	96.185	84.148

Table 1: Results of the second series of experiments (Binary classification experiments on Trend or Flat).

Appendix III. Results of the binary classification experiments on Sell or Wait

N	Batch size	Epochs number	Patience	Time per Epoch, (seconds) (Average)	Convolution kernel	Prediction Probability, %	Prediction Sell/Wait	Accuracy (Train), %	Accuracy (Validation), %	Precision (Train), %	Precision (Validation), %	Recall (Train), %	Recall (Validation), %	Loss Function (Validation)	Accuracy (Test), %	Precision (Test), %	Recall (Test), %
1	150	23	30	55	31	88.754	Sell	85.74	82.43	86.63	86.38	97.48	94.48	0.4625	83.039	90.917	89.979
2	150	23	30	55	31	75.055	Sell	85.74	82.43	86.63	86.38	97.48	94.48	0.4625	83.039	90.917	89.979
3	150	23	30	55	31	82.499	Wait	85.74	82.43	86.63	86.38	97.48	94.48	0.4625	83.039	90.917	89.979
4	150	23	30	55	31	89.952	Sell	85.74	82.43	86.63	86.38	97.48	94.48	0.4625	83.039	90.917	89.979
5	150	23	30	55	31	86.767	Wait	85.74	82.43	86.63	86.38	97.48	94.48	0.4625	83.039	90.917	89.979
6	150	23	30	55	31	76.469	Sell	85.74	82.43	86.63	86.38	97.48	94.48	0.4625	83.039	90.917	89.979
7	150	23	30	55	31	75.814	Sell	85.74	82.43	86.63	86.38	97.48	94.48	0.4625	83.039	90.917	89.979
8	150	23	30	55	31	73.164	Wait	85.74	82.43	86.63	86.38	97.48	94.48	0.4625	83.039	90.917	89.979
9	150	23	30	55	31	50.672	Sell	85.74	82.43	86.63	86.38	97.48	94.48	0.4625	83.039	90.917	89.979
10	150	23	30	55	31	48.032	Wait	85.74	82.43	86.63	86.38	97.48	94.48	0.4625	83.039	90.917	89.979
11	100	9	30	58	31	81.422	Sell	84.41	84.96	85.16	86.31	98.30	98.13	0.4329	85.528	91.460	92.413
12	100	9	30	58	31	78.733	Sell	84.41	84.96	85.16	86.31	98.30	98.13	0.4329	85.528	91.460	92.413
13	100	9	30	58	31	81.559	Wait	84.41	84.96	85.16	86.31	98.30	98.13	0.4329	85.528	91.460	92.413
14	100	9	30	58	31	79.635	Sell	84.41	84.96	85.16	86.31	98.30	98.13	0.4329	85.528	91.460	92.413
15	100	9	30	58	31	81.363	Sell	84.41	84.96	85.16	86.31	98.30	98.13	0.4329	85.528	91.460	92.413
16	100	9	30	58	31	59.206	Wait	84.41	84.96	85.16	86.31	98.30	98.13	0.4329	85.528	91.460	92.413
17	100	9	30	58	31	71.568	Wait	84.41	84.96	85.16	86.31	98.30	98.13	0.4329	85.528	91.460	92.413
18	100	9	30	58	31	82.585	Sell	84.41	84.96	85.16	86.31	98.30	98.13	0.4329	85.528	91.460	92.413
19	100	9	30	58	31	76.995	Wait	84.41	84.96	85.16	86.31	98.30	98.13	0.4329	85.528	91.460	92.413
20	100	9	30	58	31	87.538	Sell	84.41	84.96	85.16	86.31	98.30	98.13	0.4329	85.528	91.460	92.413

Continued on next page

N	Batch size	Epochs number	Patience	Time per Epoch, (seconds) (Average)	Convolution kernel	Prediction Probability, %	Prediction Sell/Wait	Accuracy (Train), %	Accuracy (Validation), %	Precision (Train), %	Precision (Validation), %	Recall (Train), %	Recall (Validation), %	Loss Function (Validation)	Accuracy (Test), %	Precision (Test), %	Recall (Test), %
21	50	7	30	30	15	94.083	Sell	84.50	82.37	85.78	86.38	97.00	94.41	0.4645	84.106	91.504	90.598
22	50	7	30	30	15	79.282	Sell	84.50	82.37	85.78	86.38	97.00	94.41	0.4645	84.106	91.504	90.598
23	50	7	30	30	15	86.347	Wait	84.50	82.37	85.78	86.38	97.00	94.41	0.4645	84.106	91.504	90.598
24	50	7	30	30	15	80.571	Wait	84.50	82.37	85.78	86.38	97.00	94.41	0.4645	84.106	91.504	90.598
25	50	7	30	30	15	78.684	Sell	84.50	82.37	85.78	86.38	97.00	94.41	0.4645	84.106	91.504	90.598
26	50	7	30	30	15	82.639	Sell	84.50	82.37	85.78	86.38	97.00	94.41	0.4645	84.106	91.504	90.598
27	50	7	30	30	15	58.096	Wait	84.50	82.37	85.78	86.38	97.00	94.41	0.4645	84.106	91.504	90.598
28	50	7	30	30	15	82.527	Wait	84.50	82.37	85.78	86.38	97.00	94.41	0.4645	84.106	91.504	90.598
29	50	7	30	30	15	89.329	Sell	84.50	82.37	85.78	86.38	97.00	94.41	0.4645	84.106	91.504	90.598
30	50	7	30	30	15	85.836	Sell	84.50	82.37	85.78	86.38	97.00	94.41	0.4645	84.106	91.504	90.598
31	30	16	30	33	15	80.715	Sell	85.25	80.41	86.09	86.34	97.61	91.75	0.4737	80.721	92.039	85.814
32	30	16	30	33	15	51.426	Sell	85.25	80.41	86.09	86.34	97.61	91.75	0.4737	80.721	92.039	85.814
33	30	16	30	33	15	86.395	Wait	85.25	80.41	86.09	86.34	97.61	91.75	0.4737	80.721	92.039	85.814
34	30	16	30	33	15	76.969	Sell	85.25	80.41	86.09	86.34	97.61	91.75	0.4737	80.721	92.039	85.814
35	30	16	30	33	15	46.657	Wait	85.25	80.41	86.09	86.34	97.61	91.75	0.4737	80.721	92.039	85.814
36	30	16	30	33	15	82.434	Sell	85.25	80.41	86.09	86.34	97.61	91.75	0.4737	80.721	92.039	85.814
37	30	16	30	33	15	80.026	Wait	85.25	80.41	86.09	86.34	97.61	91.75	0.4737	80.721	92.039	85.814
38	30	16	30	33	15	75.501	Wait	85.25	80.41	86.09	86.34	97.61	91.75	0.4737	80.721	92.039	85.814
39	30	16	30	33	15	69.579	Sell	85.25	80.41	86.09	86.34	97.61	91.75	0.4737	80.721	92.039	85.814
40	30	16	30	33	15	64.984	Sell	85.25	80.41	86.09	86.34	97.61	91.75	0.4737	80.721	92.039	85.814

Table 1: Results of the Binary classification experiments (Sell or Wait).

Appendix IV. Results of the binary classification experiments on Buy or Wait

N	Batch size	Epochs number	Patience	Time per Epoch, (seconds) (Average)	Convolution kernel	Prediction, Probability, %	Prediction Sell/Wait	Accuracy (Train), %	Accuracy (Validation), %	Precision (Train), %	Precision (Validation), %	Recall (Train), %	Recall (Validation), %	Loss Function (Validation)	Accuracy (Test), %	Precision (Test), %	Recall (Test), %
1	200	14	30	63	31	87.242	Buy	84.64	81.00	85.46	87.53	97.51	91.21	0.4854	82.556	92.532	87.914
2	200	14	30	63	31	63.411	Wait	84.64	81.00	85.46	87.53	97.51	91.21	0.4854	82.556	92.532	87.914
3	200	14	30	63	31	86.020	Wait	84.64	81.00	85.46	87.53	97.51	91.21	0.4854	82.556	92.532	87.914
4	200	14	30	63	31	58.934	Buy	84.64	81.00	85.46	87.53	97.51	91.21	0.4854	82.556	92.532	87.914
5	200	14	30	63	31	92.344	Buy	84.64	81.00	85.46	87.53	97.51	91.21	0.4854	82.556	92.532	87.914
6	200	14	30	63	31	82.977	Wait	84.64	81.00	85.46	87.53	97.51	91.21	0.4854	82.556	92.532	87.914
7	200	14	30	63	31	81.942	Wait	84.64	81.00	85.46	87.53	97.51	91.21	0.4854	82.556	92.532	87.914
8	200	14	30	63	31	85.725	Buy	84.64	81.00	85.46	87.53	97.51	91.21	0.4854	82.556	92.532	87.914
9	200	14	30	63	31	79.952	Buy	84.64	81.00	85.46	87.53	97.51	91.21	0.4854	82.556	92.532	87.914
10	200	14	30	63	31	85.081	Buy	84.64	81.00	85.46	87.53	97.51	91.21	0.4854	82.556	92.532	87.914
11	150	18	30	58	31	69.953	Buy	85.42	81.09	85.55	87.53	98.52	91.33	0.5148	81.935	92.285	87.446
12	150	18	30	58	31	85.627	Buy	85.42	81.09	85.55	87.53	98.52	91.33	0.5148	81.935	92.285	87.446
13	150	18	30	58	31	92.946	Wait	85.42	81.09	85.55	87.53	98.52	91.33	0.5148	81.935	92.285	87.446
14	150	18	30	58	31	72.729	Buy	85.42	81.09	85.55	87.53	98.52	91.33	0.5148	81.935	92.285	87.446
15	150	18	30	58	31	85.292	Wait	85.42	81.09	85.55	87.53	98.52	91.33	0.5148	81.935	92.285	87.446
16	150	18	30	58	31	67.667	Buy	85.42	81.09	85.55	87.53	98.52	91.33	0.5148	81.935	92.285	87.446
17	150	18	30	58	31	75.998	Buy	85.42	81.09	85.55	87.53	98.52	91.33	0.5148	81.935	92.285	87.446
18	150	18	30	58	31	90.670	Buy	85.42	81.09	85.55	87.53	98.52	91.33	0.5148	81.935	92.285	87.446
19	150	18	30	58	31	79.818	Wait	85.42	81.09	85.55	87.53	98.52	91.33	0.5148	81.935	92.285	87.446
20	150	18	30	58	31	91.523	Wait	85.42	81.09	85.55	87.53	98.52	91.33	0.5148	81.935	92.285	87.446

Continued on next page

N	Batch size	Epochs number	Patience	Time per Epoch, (seconds) (Average)	Convolution kernel	Prediction, Probability, %	Prediction 'Buy/Wait	Accuracy (Train), %	Accuracy (Validation), %	Precision (Train), %	Precision (Validation), %	Recall (Train), %	Recall (Validation), %	Loss Function (Validation)	Accuracy (Test), %	Precision (Test), %	Recall (Test), %
21	100	13	30	61	31	86.647	Wait	84.99	82.52	85.34	87.57	98.23	93.19	0.4854	77.167	92.331	81.676
22	100	13	30	61	31	68.982	Buy	84.99	82.52	85.34	87.57	98.23	93.19	0.4854	77.167	92.331	81.676
23	100	13	30	61	31	59.495	Wait	84.99	82.52	85.34	87.57	98.23	93.19	0.4854	77.167	92.331	81.676
24	100	13	30	61	31	73.359	Buy	84.99	82.52	85.34	87.57	98.23	93.19	0.4854	77.167	92.331	81.676
25	100	13	30	61	31	85.411	Buy	84.99	82.52	85.34	87.57	98.23	93.19	0.4854	77.167	92.331	81.676
26	100	13	30	61	31	71.198	Wait	84.99	82.52	85.34	87.57	98.23	93.19	0.4854	77.167	92.331	81.676
27	100	13	30	61	31	80.855	Wait	84.99	82.52	85.34	87.57	98.23	93.19	0.4854	77.167	92.331	81.676
28	100	13	30	61	31	74.846	Wait	84.99	82.52	85.34	87.57	98.23	93.19	0.4854	77.167	92.331	81.676
29	100	13	30	61	31	84.543	Buy	84.99	82.52	85.34	87.57	98.23	93.19	0.4854	77.167	92.331	81.676
30	100	13	30	61	31	66.044	Buy	84.99	82.52	85.34	87.57	98.23	93.19	0.4854	77.167	92.331	81.676
31	50	15	30	72	31	66.525	Wait	85.55	81.42	85.14	87.51	97.58	91.96	0.3272	75.111	92.505	89.259
32	50	15	30	72	31	81.131	Wait	85.55	81.42	85.14	87.51	97.58	91.96	0.3272	75.111	92.505	89.259
33	50	15	30	72	31	60.957	Buy	85.55	81.42	85.14	87.51	97.58	91.96	0.3272	75.111	92.505	89.259
34	50	15	30	72	31	89.701	Buy	85.55	81.42	85.14	87.51	97.58	91.96	0.3272	75.111	92.505	89.259
35	50	15	30	72	31	87.328	Buy	85.55	81.42	85.14	87.51	97.58	91.96	0.3272	75.111	92.505	89.259
36	50	15	30	72	31	70.659	Wait	85.55	81.42	85.14	87.51	97.58	91.96	0.3272	75.111	92.505	89.259
37	50	15	30	72	31	60.062	Buy	85.55	81.42	85.14	87.51	97.58	91.96	0.3272	75.111	92.505	89.259
38	50	15	30	72	31	85.072	Buy	85.55	81.42	85.14	87.51	97.58	91.96	0.3272	75.111	92.505	89.259
39	50	15	30	72	31	76.557	Wait	85.55	81.42	85.14	87.51	97.58	91.96	0.3272	75.111	92.505	89.259
40	50	15	30	72	31	72.974	Buy	85.55	81.42	85.14	87.51	97.58	91.96	0.3272	75.111	92.505	89.259

Table 1: Results of the Binary classification experiments (Buy or Wait).

Appendix V.

Machine Learning and Deep Learning Methods Abbreviations

Abbreviation	Full Name
Deep Learning Models	
AE	Autoencoder
ANN	Artificial Neural Network
BGRU	Bidirectional GRU
BiLSTM	bi-directional long short-term memory
BPNN	Backpropagation Neural Network
CapTE	Capsule network based Transformer Encoder
CF-DA-RNN	DA-RNN with cycle value
CH-RNN	Cross-modal Attention based Hybrid Recurrent Neural Network
CNN	Convolutional Neural Network
DA-RNN	Dual-Stage Attention-Based RNN
DBN	Deep Belief Network
DEM	Dilation-erosion Model
DGM	Deep Neural Generative Model
DIDLNN	Deep Increasingdecreasing-linear Neural Network
DMN	Dendrite Morphological Neuron
DNN	Deep Neural Network
ELM	Extreme Learning Machines
eLSTM	Tensor-based Event-LSTM
EMD2FNN	Empirical Mode Decomposition and Factorization Machine based Neural Network
EMD2NN	Empirical Mode Decomposition based Neural Network
FDNN	Fuzzy Deep Neural Network
FFNN	Feedforward Neural Network
FNN	Factorization Machine based Neural Network
GAN	Generative Adversarial Network
GCNN	Graph Convolutional Neural network

Table 1. Deep Learning Models. Part 1. (Jiang, 2020)

Abbreviation	Full Name
GNN	Graph Neural Network
GRU	Gated Recurrent Unit
HAN	Discriminative Deep Neural Network with Hierarchical Attention
HCAN	Hierarchical Complementary Attention Network
HGAN	Hierarchical Graph Attention Network
KDTCN	Knowledge-Driven Temporal Convolutional Network
LNNN	Linear and Nonlinear Neural Network
LSTM	Long Short Term Memory
MAFN	Multi-head Attention Fusion Network
MGU	Minimal Gated Unit
MLP	Multilayer Perceptron
MS-CNN	Multi-Scale CNN
MSTD-RCNN	Multi-Scale Temporal Dependent Recurrent Convolutional Neural Network
NARXT	Nonlinear Autoregressive Neural Network with Exogenous Takens Inputs
NN	Neural Network
PELMNN	Prediction Evolutionary Levenberg-marquardt Neural Networks
RBFNN	Radial Basis Function Neural Network
RBM	Restricted Boltzmann Machines
RNN	Recurrent Neural Network
SAEs	Stacked Autoencoders
SFM	State Frequency Memory
SRCGUs	Selective Recurrent Neural Networks with Random Connectivity Gated Unit
STNN	Stochastic Time Effective Function Neural Network

Table 1. Deep Learning Models. Part 2. (Jiang, 2020)

Abbreviation	Full Name
TABL	Temporal Attention Augmented Bilinear Layer
TGC	Temporal Graph Convolution
TSLDA	Generative Topic Model Jointly Learning Topics and Sentiments
WMN	Wavelet Neural Network
WDBPNN	Wavelet De-noising based BPNN
1D CNN	One-dimensional Convolutional Neural Networks

Table 1. Deep Learning Models. Part 3. (Jiang, 2020)

Abbreviation	Full Name
Machine Learning Models	
AB	AdaBoost
ANFIS	Adaptive Neuro-fuzzy Inference System
kNN	k-Nearest neighbor
Lasso	Lasso Regression
Logit	Logistics Regression
MKL	Multiple Kernel Learning
MRFs	Markov Random Fields
RF	Random Forest
Ridge	Ridge Regression
SVM	Support Vector Machine
SVR	Support Vector Regression

Table 2. Machine Learning Models (Jiang, 2020).

Abbreviation	Full Name
Linear Models	
AR	Autoregressive
ARMA	Autoregressive Moving Average
ARIMA	Autoregressive Integrated Moving Average
EMA	Estimated Moving Average
GARCH	Generalized Autoregressive Conditional Heteroskedasticity
LDA	Linear Discriminant Analysis
LR	Linear Regression
PLR	Piecewise Linear Regression
MA	Moving Average
MR	Mean Reversion
MCSDA	Multilinear Class-specific Discriminant Analysis
MDA	Multilinear Discriminant Analysis
MTR	Multilinear Time-series Regression
WMTR	Weighted Multilinear Time-series Regression

Table 2. Linear Models (Jiang, 2020).

Abbreviation	Full Name
Optimization Algorithms	
HM	Harmony Memory
GA	Genetic Algorithm
ISCA	Improved Sine Cosine Algorithm
GWO	Grey Wolf Optimizer
PSO	particle swarm optimization
WOA	Whale Optimization Algorithm
SCA	Sine Cosine Algorithm

Table 3. Optimisation Algorithms (Jiang, 2020).

Abbreviation	Full Name
Preprocessing Techniques	
BoF	Bag-of-feature
BoW	Bag-of-words
CEAM	Cycle Embeddings with Attention Mechanism
CEEMDAN	Complete Ensemble Empirical Mode Decomposition with Adaptive Noise
EMD	Empirical Mode Decomposition
MACD	Moving Average Convergence / Divergence
MOM	Momentum
N-BoF	Neural Bag-of-Feature
PCA	Principal Components Analysis
RS	Rough Set
RSI	Relative Strength Index
SMA	Simple Moving Average
SMC	Sub-mode Coordinate Algorithm
WT	Wavelet Transform
(2D)2PCA	2-Directional 2- Dimensional Principal Component Analysis

Table 4. Preprocessing Techniques (Jiang, 2020).

Jiang, W., 2020. Applications of deep learning in stock market prediction: recent progress. *arXiv preprint arXiv:2003.01859.*

Appendix VI.

The connector-1 to Meta Trader-5.

```
//+------------------------------------------------------------------------------------------+
//|                                TempletBook MQL5-2.mq5                                    |
//|   Based checking system and Sending buying and selling Orders                            |
//|   flag TradeSygnalBuy & TradeSygnalSell  shoud be from ML model                          |
//|         Futures contract:  ED (EURO/USD)   and others                                    |
//+------------------------------------------------------------------------------------------+
//--- input parameters
input double Lot = 1;
input int EA_Magic = 20016;
input double spreadLevel = 5.0;
input double StopLoss = 0.01;
input double Profit = 0.01;
input ulong Deviation = 10;   // In points
input ENUM_TIMEFRAMES PPeriod = PERIOD_H1;

input int numberBarOpenPosition = 5;
input int numberBarStopPosition = 5;
bool flagStopLoss = false;
// Handles
int handleIMA18;
int handleIMA28;
int handleIWMA5;
int handleIWMA8;
double MA18Buffer[];
double MA28Buffer[];
double WMA5Buffer[];
double WMA8Buffer[];
//+------------------------------------------------------------------+
//| OnCheckTradeInit – first checking                                |
//+------------------------------------------------------------------+
int OnCheckTradeInit()
 {
  if( (ENUM_ACCOUNT_TRADE_MODE)AccountInfoInteger(ACCOUNT_TRADE_MODE) == ACCOUNT_TRADE_MODE_REAL )
  {
```

```
  handleIWMA8 = iMA( _Symbol, PPeriod, 8, 0, MODE_LWMA, PRICE_CLOSE);


  return( OnCheckTradeInit() );
//--- create timer
//   EventSetTimer(20);
//   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- destroy timer
//   EventKillTimer();
  }
//+------------------------------------------------------------------+
//| OnCheckTradeTick – Second checking                               |
//+------------------------------------------------------------------+
int OnCheckTradeTick()
  {
    if( !TerminalInfoInteger(TERMINAL_CONNECTED) )
    {
      Alert("No connection to the trade server");
      return(0);
    }
    if( !TerminalInfoInteger(TERMINAL_TRADE_ALLOWED) )
    {
      Alert("Authomatic trading is turn off ");
      return(0);
    }
    if( !MQLInfoInteger(MQL_TRADE_ALLOWED) )
    {
      Alert("It is turn off", __FILE__);
      return(0);
    }
    if( (ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_MARGIN_SO_MODE) == ACCOUNT_STOPOUT_MODE_PERCENT )
// ACCOUNT_MARGIN_SO_MODE - равен либо ACCOUNT_STOPOUT_MODE_PERCENT либо ACCOUNT_STOPOUT_MODE_MONEY
    {
```

```
   int mb = MessageBox("Start it?", "Message Box", MB_YESNO|MB_ICONQUESTION);
   if( mb == IDNO )  return(0);  // "No" button
  }
  if( !TerminalInfoInteger(TERMINAL_CONNECTED) )
  {
   Alert("No connection to the trade server");
   return(0);
  }
  else
  {
   if( !AccountInfoInteger(ACCOUNT_TRADE_ALLOWED) )
   {
    Alert("Trade for this account is prohibited");
    return(0);
   }
  }
  if( !AccountInfoInteger(ACCOUNT_TRADE_EXPERT) )
  {
   Alert("Trade with the help of experts for the account is prohibited");
   return(0);
  }
  if( Lot < SymbolInfoDouble(Symbol(), SYMBOL_VOLUME_MIN) ||       // Lot < Min сделки
     Lot > SymbolInfoDouble(Symbol(), SYMBOL_VOLUME_MAX)         //  Lot > MAX
   )
  {
   Alert("Lot is not correct!");
   return(0);
  }
  return(INIT_SUCCEEDED);
 }
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
 {
  handleIMA18 = iMA( _Symbol, PPeriod, 18, 0, MODE_EMA, PRICE_CLOSE);
  handleIMA28 = iMA( _Symbol, PPeriod, 28, 0, MODE_EMA, PRICE_CLOSE);
  handleIWMA5 = iMA( _Symbol, PPeriod, 5, 0, MODE_LWMA, PRICE_CLOSE);
```

```
  if( AccountInfoDouble(ACCOUNT_MARGIN_LEVEL) != 0   &&
      AccountInfoDouble(ACCOUNT_MARGIN_LEVEL) <= AccountInfoDouble(ACCOUNT_MARGIN_SO_CALL)
      // (Margin Call)
    )
  {
    Alert("Margin Call !!");
    return(0);
  }
}
if( (ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_MARGIN_SO_MODE) == ACCOUNT_STOPOUT_MODE_MONEY )
{
  if( AccountInfoDouble(ACCOUNT_EQUITY) <= AccountInfoDouble(ACCOUNT_MARGIN_SO_CALL) )
      // (Margin Call)
  {
    Alert("Margin Call !!");
    return(0);
  }
}
if( (ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_MARGIN_SO_MODE) == ACCOUNT_STOPOUT_MODE_PERCENT )
{
  if( AccountInfoDouble(ACCOUNT_MARGIN_LEVEL) != 0   &&
      AccountInfoDouble(ACCOUNT_MARGIN_LEVEL) <= AccountInfoDouble(ACCOUNT_MARGIN_SO_SO)
    )
  {
    Alert("Stop Out !!");
    return(0);
  }
}
if( (ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_MARGIN_SO_MODE) == ACCOUNT_STOPOUT_MODE_MONEY )
{
  if( AccountInfoDouble(ACCOUNT_EQUITY) <= AccountInfoDouble(ACCOUNT_MARGIN_SO_SO) )
  {
    Alert("Stop Out !!");
    return(0);
  }
}
// checking bouble marging
double margin;
```

```
    MqlTick last_tick;
    ResetLastError();
    if( SymbolInfoTick(Symbol(), last_tick) )
    {
      if( OrderCalcMargin(ORDER_TYPE_BUY, Symbol(), Lot, last_tick.ask, margin) )
      {
        if( margin > AccountInfoDouble(ACCOUNT_MARGIN_FREE) )
        {
          Alert("Not enough money in the account");
          return(0);
        }
      }
    }
    else
    {
      Print( GetLastError()
    }
//     double _spread = SymbolInfoInteger(Symbol(), SYMBOL_SPREAD) * MathPow(10, -SymbolInfoInteger(Symbol(), SYMBOL_DIGITS) ) /
MathPow(10, -4);
    if( (ENUM_SYMBOL_TRADE_MODE)SymbolInfoInteger(Symbol(), SYMBOL_TRADE_MODE) != SYMBOL_TRADE_MODE_FULL )
    {
      Alert("We have limits for trading operation");
      return(0);
    }
    if( Bars(Symbol(), 0) < 100 )
    {
      Alert("In the chart little bars, Expert will not work!");
      return(0);
    }
    return(1);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
 {
    // Second checking go on
    if( !OnCheckTradeTick() )
```

```
{
  return;
}
static datetime last_time;
datetime last_bar_time = (datetime)SeriesInfoInteger( Symbol(), Period(), SERIES_LASTBAR_DATE );
if( last_time != last_bar_time )
{
  last_time = last_bar_time;
}
else
{
  return;
}
static datetime last_time_pause;
datetime last_bar_time_pause = (datetime)SeriesInfoInteger( Symbol(), PERIOD_H1, SERIES_LASTBAR_DATE );
if( last_time_pause != last_bar_time_pause )
{
  last_time_pause = last_bar_time_pause;
  flagStopLoss = false;
}
if( flagStopLoss == true ) return;
bool BuyOpened  = false;
bool SellOpened = false;
if( PositionSelect(_Symbol) == true )
{
  if( PositionGetInteger(POSITION_TYPE) == POSITION_TYPE_BUY )
  {
    BuyOpened = true;
  }
  else if( PositionGetInteger(POSITION_TYPE) == POSITION_TYPE_SELL )
  {
    SellOpened = true;
  }
}
MqlRates mrate[];
ResetLastError();
if( CopyRates(Symbol(), Period(), 0, numberBarStopPosition, mrate) < 0 )
{
```

```
    Print(GetLastError());
    return;
  }
  ArraySetAsSeries(mrate, true);
  // Start the Trading
     bool TradeSignalBuy  = false;
  bool TradeSignalSell = false;
  TradeSignalBuy  = OnTradeSignalBuy();
  TradeSignalSell = OnTradeSignalSell();
  bool TradeSignalBuyStop    = false;
  bool TradeSignalSellStop   = false;
  TradeSignalBuyStop   = OnTradeSignalBuyStop(mrate);
  TradeSignalSellStop  = OnTradeSignalSellStop(mrate);

  MqlTradeRequest mrequest;
  MqlTradeCheckResult check_result;
  MqlTradeResult mresult;
  MqlTick latest_price;
  if( !SymbolInfoTick(_Symbol, latest_price) )
  {
    Alert("Have not received last data: ", GetLastError());
    return;
  }
  // ----------------------------------
  //  BUYING
  if(   TradeSignalBuy == true    &&
        BuyOpened == false
   )
  {
    //---------------
    if( ((ENUM_SYMBOL_TRADE_EXECUTION)SymbolInfoInteger(Symbol(), SYMBOL_TRADE_EXEMODE)) ==
SYMBOL_TRADE_EXECUTION_INSTANT )
    {
      ZeroMemory( mrequest );
      mrequest.action   = TRADE_ACTION_DEAL;
      mrequest.symbol   = _Symbol;
      mrequest.volume   = Lot;
      mrequest.price    = NormalizeDouble(latest_price.ask, _Digits);
```

```
mrequest.sl      = NormalizeDouble(latest_price.bid - StopLoss, _Digits);
mrequest.tp      = NormalizeDouble(latest_price.ask + Profit, _Digits);
mrequest.deviation= Deviation;
mrequest.type     = ORDER_TYPE_BUY;
mrequest.type_filling = ORDER_FILLING_FOK
ZeroMemory(check_result);
ZeroMemory(mresult);
if( !OrderCheck(mrequest, check_result) )
{
  if( check_result.retcode == 10014 ) Alert("Wrong volume");
  if( check_result.retcode == 10015 ) Alert("Wrong price");
  if( check_result.retcode == 10016 ) Alert("Wrong stop-loss");
  if( check_result.retcode == 10019 ) Alert("No money");
  return;
}
else    // everything is OK
{
  if( OrderSend(mrequest, mresult) )       // send order
  {
    if( mresult.retcode == 10009  ||
        mresult.retcode == 10008)
    {
      Print("Price=", mresult.price);
    }
    else
    {
      if( mresult.retcode == 1004 )    // Reqouta
      {
        Print("Requota bid ", mresult.bid);
        Print("Requota ask ", mresult.ask);
      }
      else
      {
        Print("Retcode ", mresult.retcode);
      }
    }
  }
  else  // the order havn't sent
```

```
      {
        Print("Retcode ", mresult.retcode);
      }
     }
    }
    //---------------
    if( ((ENUM_SYMBOL_TRADE_EXECUTION)SymbolInfoInteger(Symbol(), SYMBOL_TRADE_EXEMODE)) ==
SYMBOL_TRADE_EXECUTION_EXCHANGE )
     {
     ZeroMemory(mrequest);
     mrequest.action   = TRADE_ACTION_DEAL;
     mrequest.symbol   = _Symbol;
     mrequest.volume   = Lot;
     mrequest.type     = ORDER_TYPE_BUY;
     mrequest.type_filling = ORDER_FILLING_FOK;
     ZeroMemory(check_result);
     ZeroMemory(mresult);
     if( !OrderCheck(mrequest, check_result)          {
       if( check_result.retcode == 10014 ) Alert("Wrong volume");
       if( check_result.retcode == 10019 ) Alert("No money");
       return;
     }
     else    // everything is ok
     {
       if( OrderSend(mrequest, mresult) )       // send the order
       {
         if( mresult.retcode == 10009  ||
             mresult.retcode == 10008)
         {
           //----------------
           ZeroMemory(mrequest);
           mrequest.action = TRADE_ACTION_SLTP;
           mrequest.symbol   = _Symbol;
           mrequest.sl      = NormalizeDouble(mresult.price - StopLoss, _Digits);
           mrequest.tp      = NormalizeDouble(mresult.price + Profit, _Digits);
           ZeroMemory(check_result);
           ZeroMemory(mresult);
           if( !OrderCheck(mrequest, check_result) )
```

```
            {
              if( check_result.retcode == 10015 ) Alert("Wrong price");
              if( check_result.retcode == 10016 ) Alert("Wrong stop-loss");
              return;
            }
            else
            {
              if( OrderSend(mrequest, mresult) )
              {
                if( mresult.retcode == 10009  ||
                    mresult.retcode == 10008)
                {
                  Print("SL ", mrequest.sl, "TP ", mrequest.tp);
                }
                else
                {
                  Print("Retcode ", mresult.retcode);
                }
              }
              else
              {
                Print("Retcode ", mresult.retcode);
              }
            }
          }
          else
          {
            Print("Retcode ", mresult.retcode);
          }
        }
        else
        {
          Print("Retcode ", mresult.retcode);
        }
      }
    }
  }
}
```

```
// SELL-STOP order
    if(   TradeSignalSellStop == true    &&
        SellOpened == true
     )
    {
      if( ((ENUM_SYMBOL_TRADE_EXECUTION)SymbolInfoInteger(Symbol(), SYMBOL_TRADE_EXEMODE)) ==
SYMBOL_TRADE_EXECUTION_INSTANT )
      {
        ZeroMemory( mrequest );
        mrequest.action   = TRADE_ACTION_DEAL;
        mrequest.symbol   = _Symbol;
        mrequest.volume   = Lot;
        mrequest.price    = NormalizeDouble(latest_price.ask, _Digits);          // buy by ask
        mrequest.sl       = NormalizeDouble(0.0, _Digits);
        mrequest.tp       = NormalizeDouble(0.0, _Digits);
        mrequest.deviation= Deviation;
        mrequest.type     = ORDER_TYPE_BUY;
        mrequest.type_filling = ORDER_FILLING_FOK;
        ZeroMemory(check_result);
        ZeroMemory(mresult);
        if( !OrderCheck(mrequest, check_result) )
        {
          if( check_result.retcode == 10014 ) Alert("Wrong volume");
          if( check_result.retcode == 10015 ) Alert("Wrone price");
          if( check_result.retcode == 10016 ) Alert("Wrong stop-loss");
          if( check_result.retcode == 10019 ) Alert("No money");
          return;
        }
        else
        {
          if( OrderSend(mrequest, mresult) )
          {
            if( mresult.retcode == 10009  ||
                mresult.retcode == 10008)
            {
              Print("Price=", mresult.price);
            }
            else
```

```
      {
        if( mresult.retcode == 1004 )
        {
          Print("Requota bid ", mresult.bid);
          Print("Requota ask ", mresult.ask);
        }
        else
        {
          Print("Retcode ", mresult.retcode);
        }
      }
    }
    else
    {
      Print("Retcode ", mresult.retcode);
    }
  }
}
//---------------
if( ((ENUM_SYMBOL_TRADE_EXECUTION)SymbolInfoInteger(Symbol(), SYMBOL_TRADE_EXEMODE)) ==
SYMBOL_TRADE_EXECUTION_EXCHANGE )
{
  ZeroMemory(mrequest);
  mrequest.action   = TRADE_ACTION_DEAL;
  mrequest.symbol   = _Symbol;
  mrequest.volume   = Lot;
  mrequest.type     = ORDER_TYPE_BUY;
  mrequest.type_filling = ORDER_FILLING_FOK;
  ZeroMemory(check_result);
  ZeroMemory(mresult);
  if( !OrderCheck(mrequest, check_result) )             {
    if( check_result.retcode == 10014 ) Alert("Wrong volume");
    if( check_result.retcode == 10019 ) Alert("No money");
    return;
  }
  else    // ok
  {
    if( OrderSend(mrequest, mresult) )
```

```
          {
            if( mresult.retcode == 10009  ||
                mresult.retcode == 10008)
            {
              Print("Price", mrequest.price);
            }
            else
            {
              Print("Retcode ", mresult.retcode);
            }
          }
          else
          {
            Print("Retcode ", mresult.retcode);
          }
        }
      }
      //----------------
    }
    //------------------------------------------------------


   // SELLING
    if(   TradeSignalSell == true    &&
          SellOpened == false
      )
    {
      //---------------
      if( ((ENUM_SYMBOL_TRADE_EXECUTION)SymbolInfoInteger(Symbol(), SYMBOL_TRADE_EXEMODE)) ==
SYMBOL_TRADE_EXECUTION_INSTANT )
      {
        ZeroMemory( mrequest );
        mrequest.action   = TRADE_ACTION_DEAL;
        mrequest.symbol   = _Symbol;
        mrequest.volume   = Lot;
        mrequest.price    = NormalizeDouble(latest_price.bid, _Digits);          // sell by bid
        mrequest.sl       = NormalizeDouble(latest_price.ask + StopLoss, _Digits);
        mrequest.tp       = NormalizeDouble(latest_price.bid - Profit, _Digits);
        mrequest.deviation= Deviation;
```

```
mrequest.type     = ORDER_TYPE_SELL;
mrequest.type_filling = ORDER_FILLING_FOK;
ZeroMemory(check_result);
ZeroMemory(mresult);
if( !OrderCheck(mrequest, check_result) )
{
  if( check_result.retcode == 10014 ) Alert("Wrong volume");
  if( check_result.retcode == 10015 ) Alert("Wrong price");
  if( check_result.retcode == 10016 ) Alert("Wrong stop-loss");
  if( check_result.retcode == 10019 ) Alert("No money");
  return;
}
else    // ok
{
  if( OrderSend(mrequest, mresult) )
  {
    if( mresult.retcode == 10009  ||
        mresult.retcode == 10008)
    {
      Print("Price=", mresult.price);
    }
    else
    {
      if( mresult.retcode == 1004 )
      {
        Print("Requota bid ", mresult.bid);
        Print("Requota ask ", mresult.ask);
      }
      else
      {
        Print("Retcode ", mresult.retcode);
      }
    }
  }
  else
  {
    Print("Retcode ", mresult.retcode);
  }
```

```
   }
  }
  //---------------
  if( ((ENUM_SYMBOL_TRADE_EXECUTION)SymbolInfoInteger(Symbol(), SYMBOL_TRADE_EXEMODE)) ==
SYMBOL_TRADE_EXECUTION_EXCHANGE )
  {
   ZeroMemory(mrequest);
   mrequest.action   = TRADE_ACTION_DEAL;
   mrequest.symbol   = _Symbol;
   mrequest.volume   = Lot;
   mrequest.type     = ORDER_TYPE_SELL;
   mrequest.type_filling = ORDER_FILLING_FOK;
   ZeroMemory(check_result);
   ZeroMemory(mresult);
   if( !OrderCheck(mrequest, check_result) )
   {
    if( check_result.retcode == 10014 ) Alert("Wrong volume");
    if( check_result.retcode == 10019 ) Alert("No money");
    return;
   }
   else
   {
    if( OrderSend(mrequest, mresult) )
    {
     if( mresult.retcode == 10009  ||
         mresult.retcode == 10008)
     {
      //----------------
      ZeroMemory(mrequest);
      mrequest.action = TRADE_ACTION_SLTP;
      mrequest.symbol   = _Symbol;
      mrequest.sl      = NormalizeDouble(mresult.price + StopLoss, _Digits); // changing stop-loss
      mrequest.tp      = NormalizeDouble(mresult.price - Profit, _Digits);   // changing take-profit
      ZeroMemory(check_result);
      ZeroMemory(mresult);
      if( !OrderCheck(mrequest, check_result) )
      {
       if( check_result.retcode == 10015 ) Alert("Wrong price");
```

```
            if( check_result.retcode == 10016 ) Alert("Wrong stop-loss");
            return;
          }
          else
          {
            if( OrderSend(mrequest, mresult) )
            {
              if( mresult.retcode == 10009  ||
                  mresult.retcode == 10008)
              {
                Print("SL ", mrequest.sl, "TP ", mrequest.tp);
              }
              else
              {
                Print("Retcode ", mresult.retcode);
              }
            }
            else
            {
              Print("Retcode ", mresult.retcode);
            }
          }
          //-------------------
        }
        else
        {
          Print("Retcode ", mresult.retcode);
        }
      }
      else
      {
        Print("Retcode ", mresult.retcode);
      }
    }
  }
}
//---------------------------------------------------------
```

```
  // BUY-STOP
   if(   TradeSignalBuyStop == true    &&
       BuyOpened == true
     )
   {
     //---------------
     if( ((ENUM_SYMBOL_TRADE_EXECUTION)SymbolInfoInteger(Symbol(), SYMBOL_TRADE_EXEMODE)) ==
SYMBOL_TRADE_EXECUTION_INSTANT )
     {
       ZeroMemory( mrequest );
       mrequest.action   = TRADE_ACTION_DEAL;
       mrequest.symbol   = _Symbol;
       mrequest.volume   = Lot;
       mrequest.price    = NormalizeDouble(latest_price.bid, _Digits);          // sell by bid
       mrequest.sl       = NormalizeDouble(0.0, _Digits);
       mrequest.tp       = NormalizeDouble(0.0, _Digits);
       mrequest.deviation= Deviation;
       mrequest.type     = ORDER_TYPE_SELL;
       mrequest.type_filling = ORDER_FILLING_FOK;
       ZeroMemory(check_result);
       ZeroMemory(mresult);
       if( !OrderCheck(mrequest, check_result) )
       {
         if( check_result.retcode == 10014 ) Alert("Wrong volume");
         if( check_result.retcode == 10015 ) Alert("Wrong price");
         if( check_result.retcode == 10016 ) Alert("Wrong stop-loss");
         if( check_result.retcode == 10019 ) Alert("No money");
         return;
       }
       else
       {
         if( OrderSend(mrequest, mresult) )
         {
           if( mresult.retcode == 10009  ||
               mresult.retcode == 10008)
           {
             Print("Price=", mresult.price);
           }
```

```
      else
      {
        if( mresult.retcode == 1004 )                    {
          Print("Requota bid ", mresult.bid);
          Print("Requota ask ", mresult.ask);
        }
        else
        {
          Print("Retcode ", mresult.retcode);
        }
      }
    }
    else
    {
      Print("Retcode ", mresult.retcode);
    }
  }
}
//---------------
if( ((ENUM_SYMBOL_TRADE_EXECUTION)SymbolInfoInteger(Symbol(), SYMBOL_TRADE_EXEMODE)) ==
SYMBOL_TRADE_EXECUTION_EXCHANGE )
{
  ZeroMemory(mrequest);
  mrequest.action   = TRADE_ACTION_DEAL;
  mrequest.symbol   = _Symbol;
  mrequest.volume   = Lot;
  mrequest.type     = ORDER_TYPE_SELL;
  mrequest.type_filling = ORDER_FILLING_FOK;
  ZeroMemory(check_result);
  ZeroMemory(mresult);
  if( !OrderCheck(mrequest, check_result) )
  {
    if( check_result.retcode == 10014 ) Alert("wrong volume");
    if( check_result.retcode == 10019 ) Alert("No money");
    return;
  }
  else
  {
```

```
            if( OrderSend(mrequest, mresult) )
            {
              if( mresult.retcode == 10009  ||
                  mresult.retcode == 10008)
              {
                Print("Price ", mresult.price);
              }
              else
              {
                Print("Retcode ", mresult.retcode);
              }
            }
            else
            {
              Print("Retcode ", mresult.retcode);
            }
          }
        }
      }
    }

//+------------------------------------------------------+
//| Trade function                                       |
//|                                                      |
//|       creating flagStopLoss                          |
//+------------------------------------------------------+
void OnTrade()
  {
   static int _deals;
   ulong _ticket = 0;

   if( HistorySelect(0, TimeCurrent()) )
   {
     int i = HistoryDealsTotal() - 1;
     if( _deals != i )
     {
       _deals = i;
```

```
    }
    else
    {
      return;
    }
    if( ( _ticket = HistoryDealGetTicket(i)) > 0 )
    {
      string _comment = HistoryDealGetString( _ticket, DEAL_COMMENT);   // comments
      if( StringFind( _comment, "sl", 0) != -1 )      // StringFind "sl" in the comments
      {
        flagStopLoss = true;
      }
    }
  }
 }

//+------------------------------------------+
//| Trade system part-1                      |
//|   OnTradeSignalBuy – buy signal  |
//|   It will be changed !!!!                |
//+------------------------------------------+
bool OnTradeSignalBuy()
 {
  bool flagBuy = false;
  if( CopyBuffer(handleIMA18, 0, 0, numberBarOpenPosition, MA18Buffer) < 0 )    return false;
  if( CopyBuffer(handleIMA28, 0, 0, numberBarOpenPosition, MA28Buffer) < 0 )    return false;
  if( CopyBuffer(handleIWMA5, 0, 0, numberBarOpenPosition, WMA5Buffer) < 0 )    return false;
  if( CopyBuffer(handleIWMA8, 0, 0, numberBarOpenPosition, WMA8Buffer) < 0 )    return false;
  ArraySetAsSeries( MA18Buffer, true );
  ArraySetAsSeries( MA28Buffer, true );
  ArraySetAsSeries( WMA5Buffer, true );
  ArraySetAsSeries( WMA8Buffer, true );
  bool flagCross1 = false;
  bool flagCross2 = false;
  bool flagCross  = false;
  if(   WMA5Buffer[1] > MA18Buffer[1]          &&
        WMA5Buffer[1] > MA28Buffer[1]          &&
        WMA8Buffer[1] > MA18Buffer[1]          &&
```

```
      WMA8Buffer[1] > MA28Buffer[1]
    )
  {
    for( int i=2; i < numberBarOpenPosition; i++)
    {
      if(   WMA5Buffer[i] < MA18Buffer[i]          &&
            WMA5Buffer[i] < MA28Buffer[i]
       )
      {
        flagCross1 = true;
      }
      if(
            WMA8Buffer[i] < MA18Buffer[i]          &&
            WMA8Buffer[i] < MA28Buffer[i]
       )
      {
        flagCross2 = true;
      }
    }
      if( flagCross1 == true    &&   flagCross2 == true )
      {
        flagCross = true;
      }
  }
  flagBuy = flagCross;
  return flagBuy;
 }

//+------------------------------------------------------------------+
//| Trade system part-2                                              |
//| OnTradeSignalBuyStop – signal  Buy-stop-loss  |
//|   it will be changed                                             |
//+------------------------------------------------------------------+
bool OnTradeSignalBuyStop(MqlRates& mrate[])
 {
  bool flagBuyStop = false;
  if( CopyBuffer(handleIWMA5, 0, 0, numberBarStopPosition, WMA5Buffer) < 0 )   return false;
  if( CopyBuffer(handleIWMA8, 0, 0, numberBarStopPosition, WMA8Buffer) < 0 )   return false;
```

```
 ArraySetAsSeries( WMA5Buffer, true );
 ArraySetAsSeries( WMA8Buffer, true );
   bool flagCross  = false;
   if(   WMA5Buffer[1] < WMA8Buffer[1] )
 {
   for( int i=2; i < numberBarStopPosition; i++)
   {
     if(   WMA5Buffer[i] > WMA8Buffer[i] )
     {
       flagCross = true;
     }
   }
 }
 double max = mrate[1].high;
 for( int i=1; i < numberBarStopPosition; i++)
 {
   if( mrate[i].high > max )     max = mrate[i].high;
 }
 if(   flagCross == true           &&
       mrate[1].high <= max        &&
       mrate[numberBarStopPosition-1].high <= max
   )
   {
    return flagBuyStop = true;
   }
 return flagBuyStop;
 }

//+------------------------------------------+
//| Trade system part-3                      |
//|   OnTradeSignalSel – sell signal    |
//|   it will be changed                     |
//+------------------------------------------+
bool OnTradeSignalSell()
 {
  bool flagSell = false;
  if( CopyBuffer(handleIMA18, 0, 0, numberBarOpenPosition, MA18Buffer) < 0 )    return false;
  if( CopyBuffer(handleIMA28, 0, 0, numberBarOpenPosition, MA28Buffer) < 0 )    return false;
```

```
if( CopyBuffer(handleIWMA5, 0, 0, numberBarOpenPosition, WMA5Buffer) < 0 )    return false;
if( CopyBuffer(handleIWMA8, 0, 0, numberBarOpenPosition, WMA8Buffer) < 0 )    return false;
ArraySetAsSeries( MA18Buffer, true );
ArraySetAsSeries( MA28Buffer, true );
ArraySetAsSeries( WMA5Buffer, true );
ArraySetAsSeries( WMA8Buffer, true );
bool flagCross1 = false;
bool flagCross2 = false;
bool flagCross  = false;
  if(   WMA5Buffer[1] < MA18Buffer[1]          &&
     WMA5Buffer[1] < MA28Buffer[1]          &&
     WMA8Buffer[1] < MA18Buffer[1]          &&
     WMA8Buffer[1] < MA28Buffer[1]
  )
{
  for( int i=2; i < numberBarOpenPosition; i++)
  {
     if(   WMA5Buffer[i] > MA18Buffer[i]          &&
          WMA5Buffer[i] > MA28Buffer[i]
       )
     {
       flagCross1 = true;
     }
     if(
          WMA8Buffer[i] > MA18Buffer[i]          &&
          WMA8Buffer[i] > MA28Buffer[i]
       )
     {
       flagCross2 = true;
     }
  }
     if( flagCross1 == true     &&    flagCross2 == true )
     {
       flagCross = true;
     }
}
flagSell = flagCross;
return flagSell;
```

```
 }

//+------------------------------------------------------------------+
//| Trade system part-4                                              |
//|                                                                  |
//| OnTradeSignalSellStop - Sell-stop-loss signal                    |
//|   It will be changed                                             |
//+------------------------------------------------------------------+
bool OnTradeSignalSellStop(MqlRates& mrate[])
 {
  bool flagSellStop = false;
  if( CopyBuffer(handleIWMA5, 0, 0, numberBarStopPosition, WMA5Buffer) < 0 )    return false;
  if( CopyBuffer(handleIWMA8, 0, 0, numberBarStopPosition, WMA8Buffer) < 0 )    return false;
    ArraySetAsSeries( WMA5Buffer, true );
  ArraySetAsSeries( WMA8Buffer, true );
  bool flagCross  = false;
    if(   WMA5Buffer[1] > WMA8Buffer[1] )
  {
    for( int i=2; i < numberBarStopPosition; i++)
    {
      if(   WMA5Buffer[i] < WMA8Buffer[i] )
      {
        flagCross = true;
      }
    }
  }
  double min = mrate[1].low;
  for( int i=1; i < numberBarStopPosition; i++)
  {
    if( mrate[i].low < min )     min = mrate[i].low;
  }
  if(   flagCross == true           &&
        mrate[1].low  >= min        &&
        mrate[numberBarStopPosition-1].low  >= min
    )
    {
     return flagSellStop = true;
    }
```

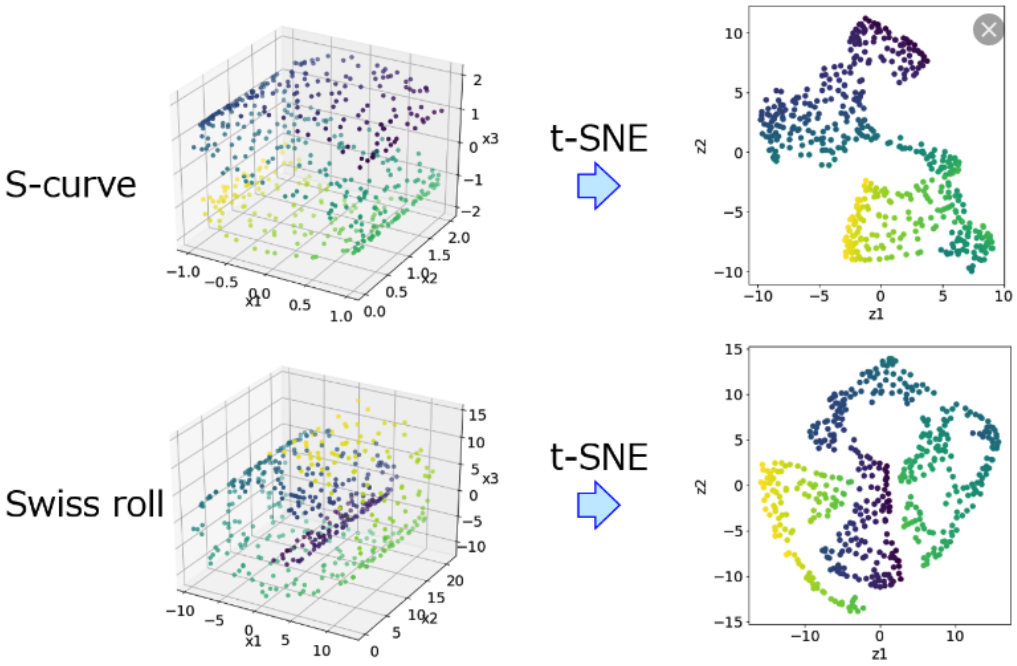
```
 return flagSellStop;
 }

//+------------------------------------------------------------------+
//| Timer function                                                   |
//+------------------------------------------------------------------+
void OnTimer()

//+------------------------------------------------------------------+
//| TradeTransaction function                                        |
//+------------------------------------------------------------------+
void OnTradeTransaction(const MqlTradeTransaction& trans,
                        const MqlTradeRequest& request,
                        const MqlTradeResult& result)
{}
//+------------------------------------------------------------------+
//| ChartEvent function                                              |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
 {}
//+------------------------------------------------------------------+
//| BookEvent function                                               |
//+------------------------------------------------------------------+
void OnBookEvent(const string &symbol)
 { }
//+------------------------------------------------------------------+
```

Appendix VII.

t-SNE Algorithm for Raw Data EURO/USD

Visualisation of Multidimensional Data

t-SNE algorithm (t-distributed Stochastic Neighbor Embedding)



In [1]:
```
# Uses the compactness hypothesis: similar points of d-dimensional space are stacked together
# We transforme the image(data) from the d-dimensional space into 2-dimensional (less often into 3-dimensional) space
# It means that we transforme the matrix [number of rows]*d[number of columns] to the matrix [number of rows]*[2]
```

Dataset

In [2]:
```
import numpy as np
import pandas as pd
```

In [3]:
```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [4]:
```
# FOREX EURUSD: c 01.01.2019 no 30.06.2019

data = pd.read_csv('/Users/vital/Python_Code/PhD_Code/2019 July/Ticks/Dukastcopy/EURUSD_ticks_2019-6.csv', sep=';')

print(data.head(n=10))
print(data[-5:])
```

```
                Time (UTC)      Ask      Bid  AskVolume  BidVolume
0  2019.01.01 22:02:37.254  1.14682  1.14598       0.75       3.75
1  2019.01.01 22:02:38.590  1.14682  1.14599       0.75       2.25
2  2019.01.01 22:02:39.138  1.14684  1.14599       0.75       3.75
3  2019.01.01 22:02:55.787  1.14684  1.14598       0.75       3.75
4  2019.01.01 22:03:02.060  1.14684  1.14598       0.75       3.75
5  2019.01.01 22:03:12.290  1.14684  1.14599       0.75       3.00
6  2019.01.01 22:03:16.253  1.14684  1.14599       0.75       3.00
7  2019.01.01 22:03:58.115  1.14691  1.14607       1.12       0.75
8  2019.01.01 22:03:59.146  1.14678  1.14607       0.75       0.75
9  2019.01.01 22:04:00.208  1.14684  1.14607       0.75       0.75
                       Time (UTC)      Ask      Bid  AskVolume  BidVolume
16680590  2019.06.30 23:58:52.782  1.13606  1.13603       2.25       1.25
16680591  2019.06.30 23:58:53.478  1.13605  1.13604       1.00       1.00
16680592  2019.06.30 23:58:56.478  1.13606  1.13603       1.50       1.25
16680593  2019.06.30 23:58:58.372  1.13605  1.13603       1.00       1.25
16680594  2019.06.30 23:58:59.343  1.13605  1.13602       1.50       1.50
```

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16680595 entries, 0 to 16680594
Data columns (total 5 columns):
Time (UTC)    object
Ask           float64
Bid           float64
AskVolume     float64
BidVolume     float64
dtypes: float64(4), object(1)
memory usage: 636.3+ MB
```

In [6]:
```
#  NaN ?
data.index[data.isnull().all(1)]
```

Out[6]:
```
Int64Index([], dtype='int64')
```

In [7]:
```
# no NaN
```

In [8]:
```
# time transforming
data['DateT'] = pd.to_datetime(data['Time (UTC)'])

data = data.drop(['Time (UTC)'],axis=1)

print(data.dtypes)
print(data.head(n=5))
```

```
Ask                float64
Bid                float64
AskVolume          float64
BidVolume          float64
DateT        datetime64[ns]
dtype: object
       Ask      Bid  AskVolume  BidVolume                  DateT
0  1.14682  1.14598       0.75       3.75 2019-01-01 22:02:37.254
1  1.14682  1.14599       0.75       2.25 2019-01-01 22:02:38.590
2  1.14684  1.14599       0.75       3.75 2019-01-01 22:02:39.138
3  1.14684  1.14598       0.75       3.75 2019-01-01 22:02:55.787
4  1.14684  1.14598       0.75       3.75 2019-01-01 22:03:02.060
```

In [9]:
```
# reducing size
data['Ask']       = pd.to_numeric(data['Ask'],downcast='float')
data['Bid']       = pd.to_numeric(data['Bid'],downcast='float')
data['AskVolume'] = pd.to_numeric(data['AskVolume'],downcast='float')
data['BidVolume'] = pd.to_numeric(data['BidVolume'],downcast='float')

data['DateT'] = pd.to_numeric(data['DateT'],downcast='integer')

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16680595 entries, 0 to 16680594
Data columns (total 5 columns):
Ask          float32
Bid          float32
AskVolume    float32
BidVolume    float32
DateT        int64
dtypes: float32(4), int64(1)
memory usage: 381.8 MB
```

In [10]:
```
print(data.head(n=6))
print(data.tail(n=6))
```

```
       Ask      Bid  AskVolume  BidVolume               DateT
0  1.14682  1.14598       0.75       3.75  1546380157254000000
1  1.14682  1.14599       0.75       2.25  1546380158590000000
2  1.14684  1.14599       0.75       3.75  1546380159138000000
3  1.14684  1.14598       0.75       3.75  1546380175787000000
4  1.14684  1.14598       0.75       3.75  1546380182060000000
5  1.14684  1.14599       0.75       3.00  1546380192290000000
              Ask      Bid  AskVolume  BidVolume               DateT
16680589  1.13605  1.13603       1.00       3.75  1561939131293000000
16680590  1.13606  1.13603       2.25       1.25  1561939132782000000
16680591  1.13605  1.13604       1.00       1.00  1561939133478000000
16680592  1.13606  1.13603       1.50       1.25  1561939136478000000
16680593  1.13605  1.13603       1.00       1.25  1561939138372000000
16680594  1.13605  1.13602       1.50       1.50  1561939139343000000
```

t-SNE

Raw data: Price, Volume, Time

In [11]:
```
from sklearn.manifold import TSNE
```

In [12]:
```
# First 1000 rows
N1 = 1000
N2 = len(data)
indexes_to_drop = np.array(range(N1,N2))
indexes_to_keep = set(range(data.shape[0])) - set(indexes_to_drop)

data2 = data.take(list(indexes_to_keep))

data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 0 to 999
Data columns (total 5 columns):
Ask          1000 non-null float32
Bid          1000 non-null float32
AskVolume    1000 non-null float32
BidVolume    1000 non-null float32
DateT        1000 non-null int64
dtypes: float32(4), int64(1)
memory usage: 31.2 KB
```

In [13]:
```
tsne_1000 = TSNE(random_state=0)
# random_state - randomness control parameter, depending on it there will be different pictures
# default output dimension n_components = 2(plane)
```

In [14]:
```
tsne_1000?
```

In [15]:
```
# training
```

In [16]:
```
%%time
tsne_1000.fit(data2)
```

```
Wall time: 7.81 s
```

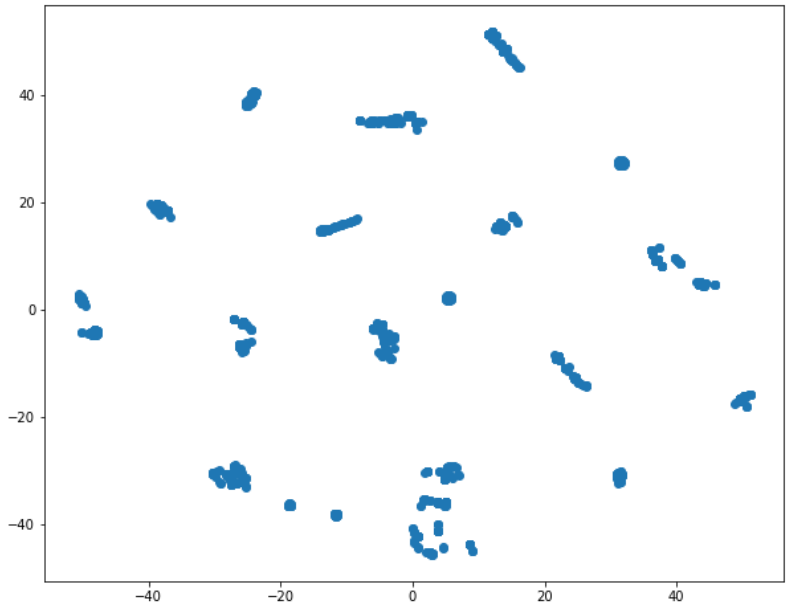
Out[16]:
```
TSNE(angle=0.5, early_exaggeration=12.0, init='random', learning_rate=200.0,
     method='barnes_hut', metric='euclidean', min_grad_norm=1e-07,
     n_components=2, n_iter=1000, n_iter_without_progress=300, perplexity=30.0,
     random_state=0, verbose=0)
```

In [17]:
```
tsne_1000.embedding_.shape  # dimentions are 2 !
```

Out[17]:
```
(1000, 2)
```

In [18]:
```
# this is a matrix with TWO features, which is the result of converting multidimensional space into two-dimensional
```

In [19]:
```
# print the matrix by features
plt.figure(num=1, figsize=(10,8))

plt.scatter(tsne_1000.embedding_[:, 0], tsne_1000.embedding_[:, 1]);
```



In [20]:
```
# First 10000 rows

N1 = 10000
N2 = len(data)
indexes_to_drop = np.array(range(N1,N2))
indexes_to_keep = set(range(data.shape[0])) - set(indexes_to_drop)

data3 = data.take(list(indexes_to_keep))

data3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 0 to 9999
Data columns (total 5 columns):
Ask          10000 non-null float32
Bid          10000 non-null float32
AskVolume    10000 non-null float32
BidVolume    10000 non-null float32
DateT        10000 non-null int64
dtypes: float32(4), int64(1)
memory usage: 312.5 KB
```

In [21]:
```
tsne_10000 = TSNE(random_state=0)
# random_state - randomness control parameter, depending on it there will be different pictures
# default output dimension n_components = 2(plane)
```

In [22]:
```
%%time
tsne_10000.fit(data3)
```

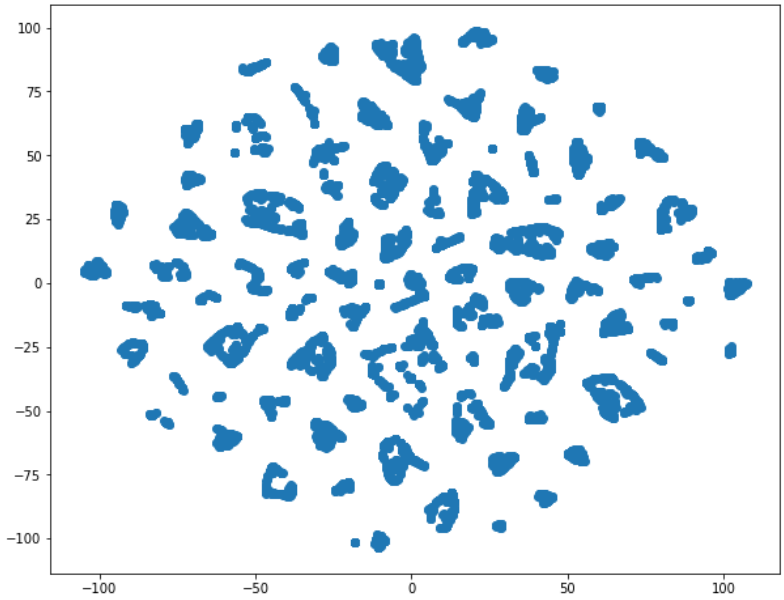
```
Wall time: 1min 19s
```

Out[22]:
```
TSNE(angle=0.5, early_exaggeration=12.0, init='random', learning_rate=200.0,
     method='barnes_hut', metric='euclidean', min_grad_norm=1e-07,
     n_components=2, n_iter=1000, n_iter_without_progress=300, perplexity=30.0,
     random_state=0, verbose=0)
```

In [23]:
```
tsne_10000.embedding_.shape  # dimentions are 2 !
```

Out[23]:
```
(10000, 2)
```

In [24]:
```
# print the matrix by features

plt.figure(num=1, figsize=(10,8))

plt.scatter(tsne_10000.embedding_[:, 0], tsne_10000.embedding_[:, 1]);
```



In [ ]:
```
```

In [25]:
```
# First 50000 rows

N1 = 50000
N2 = len(data)
indexes_to_drop = np.array(range(N1,N2))
indexes_to_keep = set(range(data.shape[0])) - set(indexes_to_drop)

data4 = data.take(list(indexes_to_keep))

data4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50000 entries, 0 to 49999
Data columns (total 5 columns):
Ask          50000 non-null float32
Bid          50000 non-null float32
AskVolume    50000 non-null float32
BidVolume    50000 non-null float32
DateT        50000 non-null int64
dtypes: float32(4), int64(1)
memory usage: 1.5 MB
```

In [26]:
```
tsne_50000 = TSNE(random_state=0)
# random_state - randomness control parameter, depending on it there will be different pictures
# default output dimension n_components = 2(plane)
```

In [27]:
```
%%time
tsne_50000.fit(data4)
```

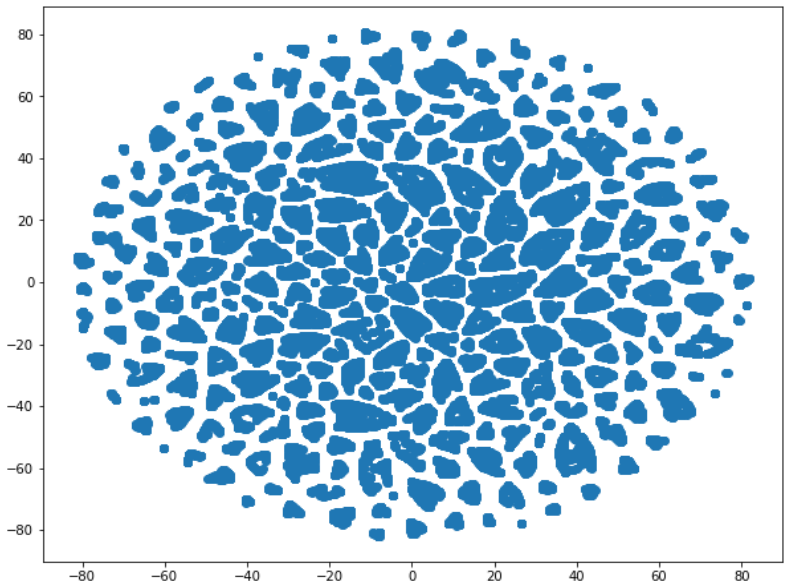
```
Wall time: 5min 39s
```

Out[27]:
```
TSNE(angle=0.5, early_exaggeration=12.0, init='random', learning_rate=200.0,
     method='barnes_hut', metric='euclidean', min_grad_norm=1e-07,
     n_components=2, n_iter=1000, n_iter_without_progress=300, perplexity=30.0,
     random_state=0, verbose=0)
```

In [28]:
```
tsne_50000.embedding_.shape  # dimentions are 2 !
```

Out[28]:
```
(50000, 2)
```

In [29]:
```
# print the matrix by features

plt.figure(num=1, figsize=(10,8))

plt.scatter(tsne_50000.embedding_[:, 0], tsne_50000.embedding_[:, 1]);
```



Relative Data

Deltas at prices, interval with previous tick and margin

In [31]:
```
# FOREX EURUSD: c 01.01.2019 no 30.06.2019

data = pd.read_csv('/Users/vital/Python_Code/PhD_Code/2019 July/Ticks/Dukastcopy/EURUSD_ticks_2019-6.csv', sep=';')

print(data.dtypes)
print(data.head(n=10))
print(data[-5:])
```

```
Time (UTC)     object
Ask           float64
Bid           float64
AskVolume     float64
BidVolume     float64
dtype: object
               Time (UTC)      Ask      Bid  AskVolume  BidVolume
0  2019.01.01 22:02:37.254  1.14682  1.14598       0.75       3.75
1  2019.01.01 22:02:38.590  1.14682  1.14599       0.75       2.25
2  2019.01.01 22:02:39.138  1.14684  1.14599       0.75       3.75
3  2019.01.01 22:02:55.787  1.14684  1.14598       0.75       3.75
4  2019.01.01 22:03:02.060  1.14684  1.14598       0.75       3.75
5  2019.01.01 22:03:12.290  1.14684  1.14599       0.75       3.00
6  2019.01.01 22:03:16.253  1.14684  1.14599       0.75       3.00
7  2019.01.01 22:03:58.115  1.14691  1.14607       1.12       0.75
8  2019.01.01 22:03:59.146  1.14678  1.14607       0.75       0.75
9  2019.01.01 22:04:00.208  1.14684  1.14607       0.75       0.75
                     Time (UTC)      Ask      Bid  AskVolume  BidVolume
16680590  2019.06.30 23:58:52.782  1.13606  1.13603       2.25       1.25
16680591  2019.06.30 23:58:53.478  1.13605  1.13604       1.00       1.00
16680592  2019.06.30 23:58:56.478  1.13606  1.13603       1.50       1.25
16680593  2019.06.30 23:58:58.372  1.13605  1.13603       1.00       1.25
16680594  2019.06.30 23:58:59.343  1.13605  1.13602       1.50       1.50
```

In [32]:
```
data['DateT'] = pd.to_datetime(data['Time (UTC)'])

data = data.drop(['Time (UTC)'],axis=1)

print(data.dtypes)
print(data.head(n=5))
```

```
Ask                float64
Bid                float64
AskVolume          float64
BidVolume          float64
DateT        datetime64[ns]
dtype: object
       Ask      Bid  AskVolume  BidVolume                  DateT
0  1.14682  1.14598       0.75       3.75 2019-01-01 22:02:37.254
1  1.14682  1.14599       0.75       2.25 2019-01-01 22:02:38.590
2  1.14684  1.14599       0.75       3.75 2019-01-01 22:02:39.138
3  1.14684  1.14598       0.75       3.75 2019-01-01 22:02:55.787
4  1.14684  1.14598       0.75       3.75 2019-01-01 22:03:02.060
```

In [33]:
```
# time intervals
data['DateDelta_2'] = data['DateT'].diff()

print(data.dtypes)
print(data.head(n=6))
```

```
Ask                   float64
Bid                   float64
AskVolume             float64
BidVolume             float64
DateT           datetime64[ns]
DateDelta_2    timedelta64[ns]
dtype: object
       Ask      Bid  AskVolume  BidVolume                  DateT  \
0  1.14682  1.14598       0.75       3.75 2019-01-01 22:02:37.254
1  1.14682  1.14599       0.75       2.25 2019-01-01 22:02:38.590
2  1.14684  1.14599       0.75       3.75 2019-01-01 22:02:39.138
3  1.14684  1.14598       0.75       3.75 2019-01-01 22:02:55.787
4  1.14684  1.14598       0.75       3.75 2019-01-01 22:03:02.060
5  1.14684  1.14599       0.75       3.00 2019-01-01 22:03:12.290

      DateDelta_2
0             NaT
1 00:00:01.336000
2 00:00:00.548000
3 00:00:16.649000
4 00:00:06.273000
5 00:00:10.230000
```

In [34]:
```
# intervals to integer
data['DateDelta'] = pd.to_numeric(data['DateDelta_2'],downcast='integer')

print(data.dtypes)
print(data.head(n=6))
```

```
Ask                   float64
Bid                   float64
AskVolume             float64
BidVolume             float64
DateT           datetime64[ns]
DateDelta_2    timedelta64[ns]
DateDelta                int64
dtype: object
       Ask      Bid  AskVolume  BidVolume                  DateT  \
0  1.14682  1.14598       0.75       3.75 2019-01-01 22:02:37.254
1  1.14682  1.14599       0.75       2.25 2019-01-01 22:02:38.590
2  1.14684  1.14599       0.75       3.75 2019-01-01 22:02:39.138
3  1.14684  1.14598       0.75       3.75 2019-01-01 22:02:55.787
4  1.14684  1.14598       0.75       3.75 2019-01-01 22:03:02.060
5  1.14684  1.14599       0.75       3.00 2019-01-01 22:03:12.290

      DateDelta_2            DateDelta
0             NaT -9223372036854775808
1 00:00:01.336000           1336000000
2 00:00:00.548000            548000000
3 00:00:16.649000          16649000000
4 00:00:06.273000           6273000000
5 00:00:10.230000          10230000000
```

In [35]:
```
data['DateDelta'].loc[0]   = 0
```

In [36]:
```
data = data.drop(['DateT', 'DateDelta_2'],axis=1)

print(data.dtypes)
print(data.head(n=6))
print(data.tail(n=6))
```

```
Ask          float64
Bid          float64
AskVolume    float64
BidVolume    float64
DateDelta      int64
dtype: object
       Ask      Bid  AskVolume  BidVolume    DateDelta
0  1.14682  1.14598       0.75       3.75            0
1  1.14682  1.14599       0.75       2.25   1336000000
2  1.14684  1.14599       0.75       3.75    548000000
3  1.14684  1.14598       0.75       3.75  16649000000
4  1.14684  1.14598       0.75       3.75   6273000000
5  1.14684  1.14599       0.75       3.00  10230000000
              Ask      Bid  AskVolume  BidVolume   DateDelta
16680589  1.13605  1.13603       1.00       3.75  2691000000
16680590  1.13606  1.13603       2.25       1.25  1489000000
16680591  1.13605  1.13604       1.00       1.00   696000000
16680592  1.13606  1.13603       1.50       1.25  3000000000
16680593  1.13605  1.13603       1.00       1.25  1894000000
16680594  1.13605  1.13602       1.50       1.50   971000000
```

In [37]:
```
#data['DateDelta']      = data['DateDelta'].div(1000000 , axis=0)  # it is float
data['DateDelta']   = data['DateDelta']/1000000    # it is float
print(data.dtypes)
print(data.head(n=3))
```

```
Ask          float64
Bid          float64
AskVolume    float64
BidVolume    float64
DateDelta    float64
dtype: object
       Ask      Bid  AskVolume  BidVolume  DateDelta
0  1.14682  1.14598       0.75       3.75        0.0
1  1.14682  1.14599       0.75       2.25     1336.0
2  1.14684  1.14599       0.75       3.75      548.0
```

In [38]:
```
# to integer
data['DateDelta'] = data['DateDelta'].astype(int)

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16680595 entries, 0 to 16680594
Data columns (total 5 columns):
Ask          float64
Bid          float64
AskVolume    float64
BidVolume    float64
DateDelta    int32
dtypes: float64(4), int32(1)
memory usage: 572.7 MB
```

In [39]:
```
# reducing size
data['Ask']       = pd.to_numeric(data['Ask'],downcast='float')
data['Bid']       = pd.to_numeric(data['Bid'],downcast='float')
data['AskVolume'] = pd.to_numeric(data['AskVolume'],downcast='float')
data['BidVolume'] = pd.to_numeric(data['BidVolume'],downcast='float')

data['DateDelta'] = pd.to_numeric(data['DateDelta'],downcast='integer')

data.info()
print(data.head(n=5))
print(data[-5:])
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16680595 entries, 0 to 16680594
Data columns (total 5 columns):
Ask          float32
Bid          float32
AskVolume    float32
BidVolume    float32
DateDelta    int32
dtypes: float32(4), int32(1)
memory usage: 318.2 MB
```

```
        Ask      Bid  AskVolume  BidVolume  DateDelta
0   1.14682  1.14598       0.75       3.75          0
1   1.14682  1.14599       0.75       2.25       1336
2   1.14684  1.14599       0.75       3.75        548
3   1.14684  1.14598       0.75       3.75      16649
4   1.14684  1.14598       0.75       3.75       6273
                Ask      Bid  AskVolume  BidVolume  DateDelta
16680590  1.13606  1.13603       2.25       1.25       1489
16680591  1.13605  1.13604       1.00       1.00        696
16680592  1.13606  1.13603       1.50       1.25       3000
16680593  1.13605  1.13603       1.00       1.25       1894
16680594  1.13605  1.13602       1.50       1.50        971
```

In [40]:
```
# Deltas between neighboring prices Ask u Bid
```

In [41]:
```
# numpy Ask
Ask_vals = data['Ask'].values
Ask_vals
```

Out[41]:
```
array([1.14682, 1.14682, 1.14684, ..., 1.13606, 1.13605, 1.13605],
      dtype=float32)
```

In [42]:
```
# numpy Bid
Bid_vals = data['Bid'].values
Bid_vals
```

Out[42]:
```
array([1.14598, 1.14599, 1.14599, ..., 1.13603, 1.13603, 1.13602],
      dtype=float32)
```

In [43]:
```
diffs_Ask = Ask_vals[1:] - Ask_vals[:-1]
diffs_Bid = Bid_vals[1:] - Bid_vals[:-1]
```

In [44]:
```
data['Ask_delta'] = pd.DataFrame(diffs_Ask)
data['Bid_delta'] = pd.DataFrame(diffs_Bid)
data.info()
print(data.head(n=10))
print(data.tail(n=10))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16680595 entries, 0 to 16680594
Data columns (total 7 columns):
Ask          float32
Bid          float32
AskVolume    float32
BidVolume    float32
DateDelta    int32
Ask_delta    float32
Bid_delta    float32
dtypes: float32(6), int32(1)
memory usage: 445.4 MB
       Ask      Bid  AskVolume  BidVolume  DateDelta  Ask_delta  Bid_delta
0  1.14682  1.14598       0.75       3.75          0    0.00000    0.00001
1  1.14682  1.14599       0.75       2.25       1336    0.00002    0.00000
2  1.14684  1.14599       0.75       3.75        548    0.00000   -0.00001
3  1.14684  1.14598       0.75       3.75      16649    0.00000    0.00000
4  1.14684  1.14598       0.75       3.75       6273    0.00000    0.00001
5  1.14684  1.14599       0.75       3.00      10230    0.00000    0.00000
6  1.14684  1.14599       0.75       3.00       3963    0.00007    0.00008
7  1.14691  1.14607       1.12       0.75      41862   -0.00013    0.00000
8  1.14678  1.14607       0.75       0.75       1031    0.00006    0.00000
9  1.14684  1.14607       0.75       0.75       1062   -0.00015    0.00000
                Ask      Bid  AskVolume  BidVolume  DateDelta  Ask_delta  \
16680585  1.13606  1.13604       2.25       1.87        251    0.00000
16680586  1.13606  1.13603       2.25       3.56        754   -0.00001
16680587  1.13605  1.13603       1.00       3.19       1631    0.00001
16680588  1.13606  1.13603       2.25       2.44        205   -0.00001
16680589  1.13605  1.13603       1.00       3.75       2691    0.00001
16680590  1.13606  1.13603       2.25       1.25       1489   -0.00001
16680591  1.13605  1.13604       1.00       1.00        696    0.00001
16680592  1.13606  1.13603       1.50       1.25       3000   -0.00001
16680593  1.13605  1.13603       1.00       1.25       1894    0.00000
16680594  1.13605  1.13602       1.50       1.50        971        NaN
```

```
          Bid_delta
16680585   -0.00001
16680586    0.00000
16680587    0.00000
16680588    0.00000
16680589    0.00000
16680590    0.00001
16680591   -0.00001
16680592    0.00000
16680593   -0.00001
16680594        NaN
```

In [45]:
```
# Move the Delt columns down one line
data['Ask_delta'] = data['Ask_delta'].shift(1)
data['Bid_delta'] = data['Bid_delta'].shift(1)
data.info()
print(data.head(n=10))
print(data.tail(n=10))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16680595 entries, 0 to 16680594
Data columns (total 7 columns):
Ask          float32
Bid          float32
AskVolume    float32
BidVolume    float32
DateDelta    int32
Ask_delta    float32
Bid_delta    float32
dtypes: float32(6), int32(1)
memory usage: 445.4 MB
       Ask      Bid  AskVolume  BidVolume  DateDelta  Ask_delta  Bid_delta
0  1.14682  1.14598       0.75       3.75          0        NaN        NaN
1  1.14682  1.14599       0.75       2.25       1336    0.00000    0.00001
2  1.14684  1.14599       0.75       3.75        548    0.00002    0.00000
3  1.14684  1.14598       0.75       3.75      16649    0.00000   -0.00001
4  1.14684  1.14598       0.75       3.75       6273    0.00000    0.00000
5  1.14684  1.14599       0.75       3.00      10230    0.00000    0.00001
6  1.14684  1.14599       0.75       3.00       3963    0.00000    0.00000
7  1.14691  1.14607       1.12       0.75      41862    0.00007    0.00008
8  1.14678  1.14607       0.75       0.75       1031   -0.00013    0.00000
9  1.14684  1.14607       0.75       0.75       1062    0.00006    0.00000
              Ask      Bid  AskVolume  BidVolume  DateDelta  Ask_delta  \
16680585  1.13606  1.13604       2.25       1.87        251    0.00001
16680586  1.13606  1.13603       2.25       3.56        754    0.00000
16680587  1.13605  1.13603       1.00       3.19       1631   -0.00001
16680588  1.13606  1.13603       2.25       2.44        205    0.00001
16680589  1.13605  1.13603       1.00       3.75       2691   -0.00001
16680590  1.13606  1.13603       2.25       1.25       1489    0.00001
16680591  1.13605  1.13604       1.00       1.00        696   -0.00001
16680592  1.13606  1.13603       1.50       1.25       3000    0.00001
16680593  1.13605  1.13603       1.00       1.25       1894   -0.00001
16680594  1.13605  1.13602       1.50       1.50        971    0.00000

          Bid_delta
16680585    0.00000
16680586   -0.00001
16680587    0.00000
16680588    0.00000
16680589    0.00000
16680590    0.00000
16680591    0.00001
16680592   -0.00001
16680593    0.00000
16680594   -0.00001
```

In [46]:
```
# Deltas to integer

data['Ask_delta'].loc[0]   = 0
data['Bid_delta'].loc[0]   = 0
# * 100.000
data['Ask_delta'] = data['Ask_delta'] * 100000
data['Bid_delta'] = data['Bid_delta'] * 100000

print(data.head(n=10))
print(data.tail(n=10))
```

```
       Ask      Bid  AskVolume  BidVolume  DateDelta  Ask_delta  Bid_delta
0  1.14682  1.14598       0.75       3.75          0   0.000000   0.000000
1  1.14682  1.14599       0.75       2.25       1336   0.000000   1.001358
2  1.14684  1.14599       0.75       3.75        548   2.002716   0.000000
3  1.14684  1.14598       0.75       3.75      16649   0.000000  -1.001358
4  1.14684  1.14598       0.75       3.75       6273   0.000000   0.000000
5  1.14684  1.14599       0.75       3.00      10230   0.000000   1.001358
6  1.14684  1.14599       0.75       3.00       3963   0.000000   0.000000
7  1.14691  1.14607       1.12       0.75      41862   6.997585   7.998943
8  1.14678  1.14607       0.75       0.75       1031 -12.993813   0.000000
9  1.14684  1.14607       0.75       0.75       1062   5.996227   0.000000
              Ask      Bid  AskVolume  BidVolume  DateDelta  Ask_delta  \
16680585  1.13606  1.13604       2.25       1.87        251   1.001358
16680586  1.13606  1.13603       2.25       3.56        754   0.000000
16680587  1.13605  1.13603       1.00       3.19       1631  -1.001358
16680588  1.13606  1.13603       2.25       2.44        205   1.001358
16680589  1.13605  1.13603       1.00       3.75       2691  -1.001358
16680590  1.13606  1.13603       2.25       1.25       1489   1.001358
16680591  1.13605  1.13604       1.00       1.00        696  -1.001358
16680592  1.13606  1.13603       1.50       1.25       3000   1.001358
16680593  1.13605  1.13603       1.00       1.25       1894  -1.001358
16680594  1.13605  1.13602       1.50       1.50        971   0.000000

          Bid_delta
16680585   0.000000
16680586  -1.001358
16680587   0.000000
16680588   0.000000
16680589   0.000000
16680590   0.000000
16680591   1.001358
16680592  -1.001358
16680593   0.000000
16680594  -1.001358
```

In [47]:
```
# round up according to the rules of mathematics to the whole
data['Ask_delta'] = data['Ask_delta'].round()
data['Bid_delta'] = data['Bid_delta'].round()

print(data.head(n=10))
print(data.tail(n=10))
```

```
       Ask      Bid  AskVolume  BidVolume  DateDelta  Ask_delta  Bid_delta
0  1.14682  1.14598       0.75       3.75          0        0.0        0.0
1  1.14682  1.14599       0.75       2.25       1336        0.0        1.0
2  1.14684  1.14599       0.75       3.75        548        2.0        0.0
3  1.14684  1.14598       0.75       3.75      16649        0.0       -1.0
4  1.14684  1.14598       0.75       3.75       6273        0.0        0.0
5  1.14684  1.14599       0.75       3.00      10230        0.0        1.0
6  1.14684  1.14599       0.75       3.00       3963        0.0        0.0
7  1.14691  1.14607       1.12       0.75      41862        7.0        8.0
8  1.14678  1.14607       0.75       0.75       1031      -13.0        0.0
9  1.14684  1.14607       0.75       0.75       1062        6.0        0.0
              Ask      Bid  AskVolume  BidVolume  DateDelta  Ask_delta  \
16680585  1.13606  1.13604       2.25       1.87        251        1.0
16680586  1.13606  1.13603       2.25       3.56        754        0.0
16680587  1.13605  1.13603       1.00       3.19       1631       -1.0
16680588  1.13606  1.13603       2.25       2.44        205        1.0
16680589  1.13605  1.13603       1.00       3.75       2691       -1.0
16680590  1.13606  1.13603       2.25       1.25       1489        1.0
16680591  1.13605  1.13604       1.00       1.00        696       -1.0
16680592  1.13606  1.13603       1.50       1.25       3000        1.0
16680593  1.13605  1.13603       1.00       1.25       1894       -1.0
16680594  1.13605  1.13602       1.50       1.50        971        0.0

          Bid_delta
16680585        0.0
16680586       -1.0
16680587        0.0
16680588        0.0
16680589        0.0
16680590        0.0
16680591        1.0
16680592       -1.0
16680593        0.0
16680594       -1.0
```

In [48]:
```
# to integer!
data['Ask_delta'] = data['Ask_delta'].astype(int)
data['Bid_delta'] = data['Bid_delta'].astype(int)
data.info()
print(data.head(n=10))
print(data.tail(n=10))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16680595 entries, 0 to 16680594
Data columns (total 7 columns):
Ask          float32
Bid          float32
AskVolume    float32
BidVolume    float32
DateDelta    int32
Ask_delta    int32
Bid_delta    int32
dtypes: float32(4), int32(3)
memory usage: 445.4 MB
       Ask      Bid  AskVolume  BidVolume  DateDelta  Ask_delta  Bid_delta
0  1.14682  1.14598       0.75       3.75          0          0          0
1  1.14682  1.14599       0.75       2.25       1336          0          1
2  1.14684  1.14599       0.75       3.75        548          2          0
3  1.14684  1.14598       0.75       3.75      16649          0         -1
4  1.14684  1.14598       0.75       3.75       6273          0          0
5  1.14684  1.14599       0.75       3.00      10230          0          1
6  1.14684  1.14599       0.75       3.00       3963          0          0
7  1.14691  1.14607       1.12       0.75      41862          7          8
8  1.14678  1.14607       0.75       0.75       1031        -13          0
9  1.14684  1.14607       0.75       0.75       1062          6          0
             Ask      Bid  AskVolume  BidVolume  DateDelta  Ask_delta  \
16680585  1.13606  1.13604       2.25       1.87        251          1
16680586  1.13606  1.13603       2.25       3.56        754          0
16680587  1.13605  1.13603       1.00       3.19       1631         -1
16680588  1.13606  1.13603       2.25       2.44        205          1
16680589  1.13605  1.13603       1.00       3.75       2691         -1
16680590  1.13606  1.13603       2.25       1.25       1489          1
16680591  1.13605  1.13604       1.00       1.00        696         -1
16680592  1.13606  1.13603       1.50       1.25       3000          1
16680593  1.13605  1.13603       1.00       1.25       1894         -1
16680594  1.13605  1.13602       1.50       1.50        971          0


          Bid_delta
16680585          0
16680586         -1
16680587          0
16680588          0
16680589          0
16680590          0
16680591          1
16680592         -1
16680593          0
16680594         -1
```

In [49]:
```
# margin (Ask - Bid)

data['Margin'] = data['Ask'] - data['Bid']
print(data.dtypes)
print(data.head(n=10))
print(data.tail(n=10))
```

```
Ask          float32
Bid          float32
AskVolume    float32
BidVolume    float32
DateDelta      int32
Ask_delta      int32
Bid_delta      int32
Margin       float32
dtype: object
       Ask      Bid  AskVolume  BidVolume  DateDelta  Ask_delta  Bid_delta  \
0  1.14682  1.14598       0.75       3.75          0          0          0
1  1.14682  1.14599       0.75       2.25       1336          0          1
2  1.14684  1.14599       0.75       3.75        548          2          0
3  1.14684  1.14598       0.75       3.75      16649          0         -1
4  1.14684  1.14598       0.75       3.75       6273          0          0
5  1.14684  1.14599       0.75       3.00      10230          0          1
6  1.14684  1.14599       0.75       3.00       3963          0          0
7  1.14691  1.14607       1.12       0.75      41862          7          8
8  1.14678  1.14607       0.75       0.75       1031        -13          0
9  1.14684  1.14607       0.75       0.75       1062          6          0
```

```
             Ask      Bid  AskVolume  BidVolume  DateDelta  Ask_delta  \
16680585  1.13606  1.13604       2.25       1.87        251          1
16680586  1.13606  1.13603       2.25       3.56        754          0
16680587  1.13605  1.13603       1.00       3.19       1631         -1
16680588  1.13606  1.13603       2.25       2.44        205          1
16680589  1.13605  1.13603       1.00       3.75       2691         -1
16680590  1.13606  1.13603       2.25       1.25       1489          1
16680591  1.13605  1.13604       1.00       1.00        696         -1
16680592  1.13606  1.13603       1.50       1.25       3000          1
16680593  1.13605  1.13603       1.00       1.25       1894         -1
16680594  1.13605  1.13602       1.50       1.50        971          0

          Bid_delta   Margin
16680585          0  0.00002
16680586         -1  0.00003
16680587          0  0.00002
16680588          0  0.00003
16680589          0  0.00002
16680590          0  0.00003
16680591          1  0.00001
16680592         -1  0.00003
16680593          0  0.00002
16680594         -1  0.00003
```

In [50]:
```
# margin ew integer
data['Margin'] = data['Margin'] * 100000

print(data.head(n=10))
print(data.tail(n=10))
```

```
       Ask      Bid  AskVolume  BidVolume  DateDelta  Ask_delta  Bid_delta  \
0  1.14682  1.14598       0.75       3.75          0          0          0
1  1.14682  1.14599       0.75       2.25       1336          0          1
2  1.14684  1.14599       0.75       3.75        548          2          0
3  1.14684  1.14598       0.75       3.75      16649          0         -1
4  1.14684  1.14598       0.75       3.75       6273          0          0
5  1.14684  1.14599       0.75       3.00      10230          0          1
6  1.14684  1.14599       0.75       3.00       3963          0          0
7  1.14691  1.14607       1.12       0.75      41862          7          8
8  1.14678  1.14607       0.75       0.75       1031        -13          0
9  1.14684  1.14607       0.75       0.75       1062          6          0

      Margin
0  83.994865
1  82.993507
2  84.996223
3  85.997581
4  85.997581
5  84.996223
6  84.996223
7  83.994865
8  71.001053
9  76.997284
             Ask      Bid  AskVolume  BidVolume  DateDelta  Ask_delta  \
16680585  1.13606  1.13604       2.25       1.87        251          1
16680586  1.13606  1.13603       2.25       3.56        754          0
16680587  1.13605  1.13603       1.00       3.19       1631         -1
16680588  1.13606  1.13603       2.25       2.44        205          1
16680589  1.13605  1.13603       1.00       3.75       2691         -1
16680590  1.13606  1.13603       2.25       1.25       1489          1
16680591  1.13605  1.13604       1.00       1.00        696         -1
16680592  1.13606  1.13603       1.50       1.25       3000          1
16680593  1.13605  1.13603       1.00       1.25       1894         -1
16680594  1.13605  1.13602       1.50       1.50        971          0

          Bid_delta    Margin
16680585          0  2.002716
16680586         -1  3.004074
16680587          0  2.002716
16680588          0  3.004074
16680589          0  2.002716
16680590          0  3.004074
16680591          1  1.001358
16680592         -1  3.004074
16680593          0  2.002716
16680594         -1  3.004074
```

In [51]:
```
# round up according to the rules of mathematics to the integer
data['Margin'] = data['Margin'].round()

print(data.head(n=10))
print(data.tail(n=10))
```

```
       Ask      Bid  AskVolume  BidVolume  DateDelta  Ask_delta  Bid_delta  \
0  1.14682  1.14598       0.75       3.75          0          0          0
1  1.14682  1.14599       0.75       2.25       1336          0          1
2  1.14684  1.14599       0.75       3.75        548          2          0
3  1.14684  1.14598       0.75       3.75      16649          0         -1
4  1.14684  1.14598       0.75       3.75       6273          0          0
5  1.14684  1.14599       0.75       3.00      10230          0          1
6  1.14684  1.14599       0.75       3.00       3963          0          0
7  1.14691  1.14607       1.12       0.75      41862          7          8
8  1.14678  1.14607       0.75       0.75       1031        -13          0
9  1.14684  1.14607       0.75       0.75       1062          6          0

   Margin
0    84.0
1    83.0
2    85.0
3    86.0
4    86.0
5    85.0
6    85.0
7    84.0
8    71.0
9    77.0
             Ask      Bid  AskVolume  BidVolume  DateDelta  Ask_delta  \
16680585  1.13606  1.13604       2.25       1.87        251          1
16680586  1.13606  1.13603       2.25       3.56        754          0
16680587  1.13605  1.13603       1.00       3.19       1631         -1
16680588  1.13606  1.13603       2.25       2.44        205          1
16680589  1.13605  1.13603       1.00       3.75       2691         -1
16680590  1.13606  1.13603       2.25       1.25       1489          1
16680591  1.13605  1.13604       1.00       1.00        696         -1
16680592  1.13606  1.13603       1.50       1.25       3000          1
16680593  1.13605  1.13603       1.00       1.25       1894         -1
16680594  1.13605  1.13602       1.50       1.50        971          0

          Bid_delta  Margin
16680585          0     2.0
16680586         -1     3.0
16680587          0     2.0
16680588          0     3.0
16680589          0     2.0
16680590          0     3.0
16680591          1     1.0
16680592         -1     3.0
16680593          0     2.0
16680594         -1     3.0
```

In [52]:
```
# to integer!
data['Margin'] = data['Margin'].astype(int)

data.info()
print(data.head(n=10))
print(data.tail(n=10))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16680595 entries, 0 to 16680594
Data columns (total 8 columns):
Ask          float32
Bid          float32
AskVolume    float32
BidVolume    float32
DateDelta    int32
Ask_delta    int32
Bid_delta    int32
Margin       int32
dtypes: float32(4), int32(4)
memory usage: 509.1 MB
```

```
      Ask      Bid  AskVolume  BidVolume  DateDelta  Ask_delta  Bid_delta  \
0  1.14682  1.14598       0.75       3.75          0          0          0
1  1.14682  1.14599       0.75       2.25       1336          0          1
2  1.14684  1.14599       0.75       3.75        548          2          0
3  1.14684  1.14598       0.75       3.75      16649          0         -1
4  1.14684  1.14598       0.75       3.75       6273          0          0
5  1.14684  1.14599       0.75       3.00      10230          0          1
6  1.14684  1.14599       0.75       3.00       3963          0          0
7  1.14691  1.14607       1.12       0.75      41862          7          8
8  1.14678  1.14607       0.75       0.75       1031        -13          0
9  1.14684  1.14607       0.75       0.75       1062          6          0

   Margin
0      84
1      83
2      85
3      86
4      86
5      85
6      85
7      84
8      71
9      77
              Ask      Bid  AskVolume  BidVolume  DateDelta  Ask_delta  \
16680585  1.13606  1.13604       2.25       1.87        251          1
16680586  1.13606  1.13603       2.25       3.56        754          0
16680587  1.13605  1.13603       1.00       3.19       1631         -1
16680588  1.13606  1.13603       2.25       2.44        205          1
16680589  1.13605  1.13603       1.00       3.75       2691         -1
16680590  1.13606  1.13603       2.25       1.25       1489          1
16680591  1.13605  1.13604       1.00       1.00        696         -1
16680592  1.13606  1.13603       1.50       1.25       3000          1
16680593  1.13605  1.13603       1.00       1.25       1894         -1
16680594  1.13605  1.13602       1.50       1.50        971          0

          Bid_delta  Margin
16680585          0       2
16680586         -1       3
16680587          0       2
16680588          0       3
16680589          0       2
16680590          0       3
16680591          1       1
16680592         -1       3
16680593          0       2
16680594         -1       3
```

In [53]:
```
data = data.drop(['Ask','Bid'],axis=1)

data.info()
print(data.head(n=10))
print(data.tail(n=10))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16680595 entries, 0 to 16680594
Data columns (total 6 columns):
AskVolume    float32
BidVolume    float32
DateDelta    int32
Ask_delta    int32
Bid_delta    int32
Margin       int32
dtypes: float32(2), int32(4)
memory usage: 381.8 MB
   AskVolume  BidVolume  DateDelta  Ask_delta  Bid_delta  Margin
0       0.75       3.75          0          0          0      84
1       0.75       2.25       1336          0          1      83
2       0.75       3.75        548          2          0      85
3       0.75       3.75      16649          0         -1      86
4       0.75       3.75       6273          0          0      86
5       0.75       3.00      10230          0          1      85
6       0.75       3.00       3963          0          0      85
7       1.12       0.75      41862          7          8      84
8       0.75       0.75       1031        -13          0      71
9       0.75       0.75       1062          6          0      77
```

	AskVolume	BidVolume	DateDelta	Ask_delta	Bid_delta	Margin
16680585	2.25	1.87	251	1	0	2
16680586	2.25	3.56	754	0	-1	3
16680587	1.00	3.19	1631	-1	0	2
16680588	2.25	2.44	205	1	0	3
16680589	1.00	3.75	2691	-1	0	2
16680590	2.25	1.25	1489	1	0	3
16680591	1.00	1.00	696	-1	1	1
16680592	1.50	1.25	3000	1	-1	3
16680593	1.00	1.25	1894	-1	0	2
16680594	1.50	1.50	971	0	-1	3

t-SNE

relative data

In [54]:
```
# First 1000 rows

N1 = 1000
N2 = len(data)
indexes_to_drop = np.array(range(N1,N2))
indexes_to_keep = set(range(data.shape[0])) - set(indexes_to_drop)

data2 = data.take(list(indexes_to_keep))

data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 0 to 999
Data columns (total 6 columns):
AskVolume    1000 non-null float32
BidVolume    1000 non-null float32
DateDelta    1000 non-null int32
Ask_delta    1000 non-null int32
Bid_delta    1000 non-null int32
Margin       1000 non-null int32
dtypes: float32(2), int32(4)
memory usage: 31.2 KB
```

In [55]:
```
tsne_1000 = TSNE(random_state=0)
# random_state - randomness control parameter, depending on it there will be different pictures
# default output dimension n_components = 2(plane)
```

In [56]:
```
%%time
tsne_1000.fit(data2)
```

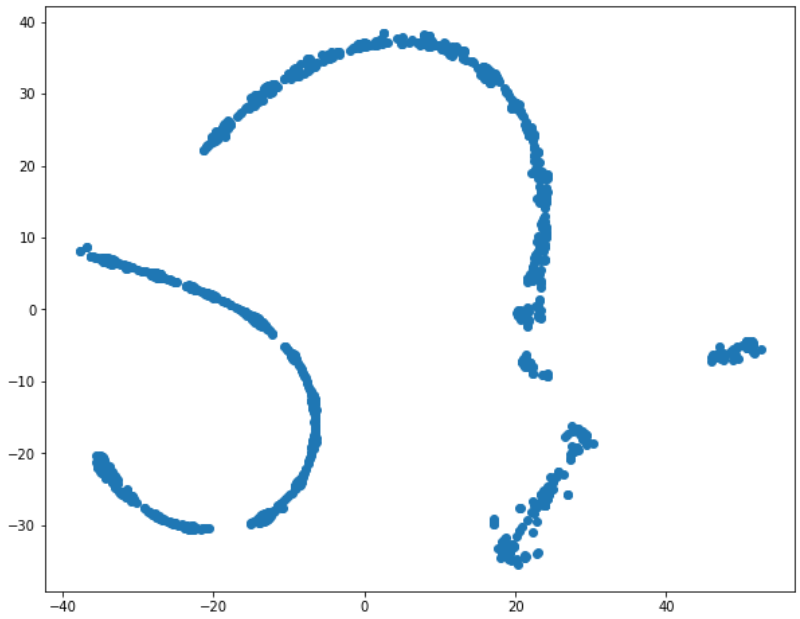
```
Wall time: 3.62 s
```

Out[56]:
```
TSNE(angle=0.5, early_exaggeration=12.0, init='random', learning_rate=200.0,
     method='barnes_hut', metric='euclidean', min_grad_norm=1e-07,
     n_components=2, n_iter=1000, n_iter_without_progress=300, perplexity=30.0,
     random_state=0, verbose=0)
```

In [57]:
```
tsne_1000.embedding_.shape  # dimentions are 2 !
```

Out[57]:
```
(1000, 2)
```

In [58]:
```
# print the matrix by features
plt.figure(num=1, figsize=(10,8))

plt.scatter(tsne_1000.embedding_[:, 0], tsne_1000.embedding_[:, 1]);
```



In [ ]:

In [60]:
```
# First 10000 rows

N1 = 10000
N2 = len(data)
indexes_to_drop = np.array(range(N1,N2))
indexes_to_keep = set(range(data.shape[0])) - set(indexes_to_drop)

data3 = data.take(list(indexes_to_keep))

data3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 0 to 9999
Data columns (total 6 columns):
AskVolume    10000 non-null float32
BidVolume    10000 non-null float32
DateDelta    10000 non-null int32
Ask_delta    10000 non-null int32
Bid_delta    10000 non-null int32
Margin       10000 non-null int32
dtypes: float32(2), int32(4)
memory usage: 312.5 KB
```

In [61]:
```
tsne_10000 = TSNE(random_state=0)
# random_state - randomness control parameter, depending on it there will be different pictures
# default output dimension n_components = 2(plane)
```

In [62]:
```
%%time
tsne_10000.fit(data3)
```

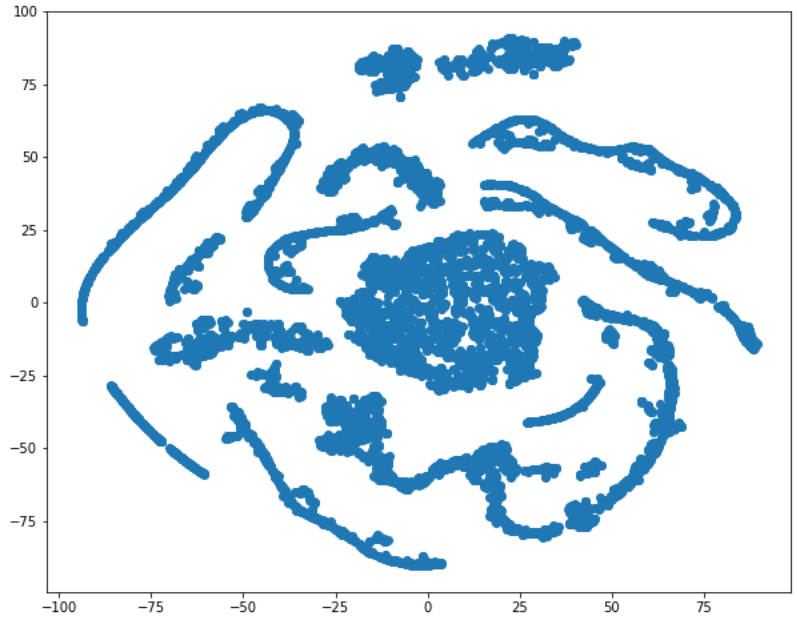
```
Wall time: 1min 6s
```

Out[62]:
```
TSNE(angle=0.5, early_exaggeration=12.0, init='random', learning_rate=200.0,
     method='barnes_hut', metric='euclidean', min_grad_norm=1e-07,
     n_components=2, n_iter=1000, n_iter_without_progress=300, perplexity=30.0,
     random_state=0, verbose=0)
```

In [63]:
```
tsne_10000.embedding_.shape  # dimentions are 2 !
```

Out[63]:
```
(10000, 2)
```

In [64]:
```
# print the matrix by features

plt.figure(num=1, figsize=(10,8))

plt.scatter(tsne_10000.embedding_[:, 0], tsne_10000.embedding_[:, 1]);
```



In [ ]:
```
```

In [65]:
```
# First 50000 rows

N1 = 50000
N2 = len(data)
indexes_to_drop = np.array(range(N1,N2))
indexes_to_keep = set(range(data.shape[0])) - set(indexes_to_drop)

data4 = data.take(list(indexes_to_keep))

data4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50000 entries, 0 to 49999
Data columns (total 6 columns):
AskVolume    50000 non-null float32
BidVolume    50000 non-null float32
DateDelta    50000 non-null int32
Ask_delta    50000 non-null int32
Bid_delta    50000 non-null int32
Margin       50000 non-null int32
dtypes: float32(2), int32(4)
memory usage: 1.5 MB
```

In [66]:
```
tsne_50000 = TSNE(random_state=0)
# random_state - randomness control parameter, depending on it there will be different pictures
# default output dimension n_components = 2(plane)
```

In [67]:
```
%%time
tsne_50000.fit(data4)
```

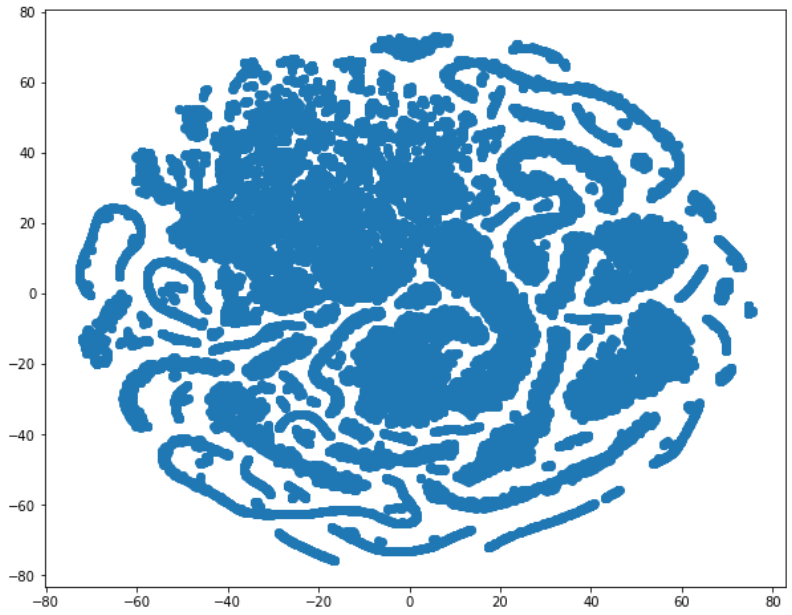
```
Wall time: 8min 23s
```

Out[67]:
```
TSNE(angle=0.5, early_exaggeration=12.0, init='random', learning_rate=200.0,
     method='barnes_hut', metric='euclidean', min_grad_norm=1e-07,
     n_components=2, n_iter=1000, n_iter_without_progress=300, perplexity=30.0,
     random_state=0, verbose=0)
```

In [68]:
```
tsne_50000.embedding_.shape  # dimentions are 2 !
```

Out[68]:
```
(50000, 2)
```

In [69]:
```
# print the matrix by features

plt.figure(num=1, figsize=(10,8))

plt.scatter(tsne_50000.embedding_[:, 0], tsne_50000.embedding_[:, 1]);
```



In [70]:
```
# Random 1000 rows

import random

n_rand = random.randint(0, data.shape[0] - 1000)
NN = 1000

indexes_to_drop1 = np.array(range(0,n_rand))
indexes_to_drop2 = np.array(range(n_rand+NN,data.shape[0]))
indexes_to_keep = set(range(data.shape[0])) - set(indexes_to_drop1) - set(indexes_to_drop2)
data5 = data.take(list(indexes_to_keep))

data5.info()
print(n_rand)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 346112 to 346111
Data columns (total 6 columns):
AskVolume    1000 non-null float32
BidVolume    1000 non-null float32
DateDelta    1000 non-null int32
Ask_delta    1000 non-null int32
Bid_delta    1000 non-null int32
Margin       1000 non-null int32
dtypes: float32(2), int32(4)
memory usage: 31.2 KB
346055
```

In [71]:
```
tsne_1000_Random = TSNE(random_state=0)
# random_state - randomness control parameter, depending on it there will be different pictures
# default output dimension n_components = 2(plane)
```

In [72]:
```
%%time
tsne_1000_Random.fit(data5)
```

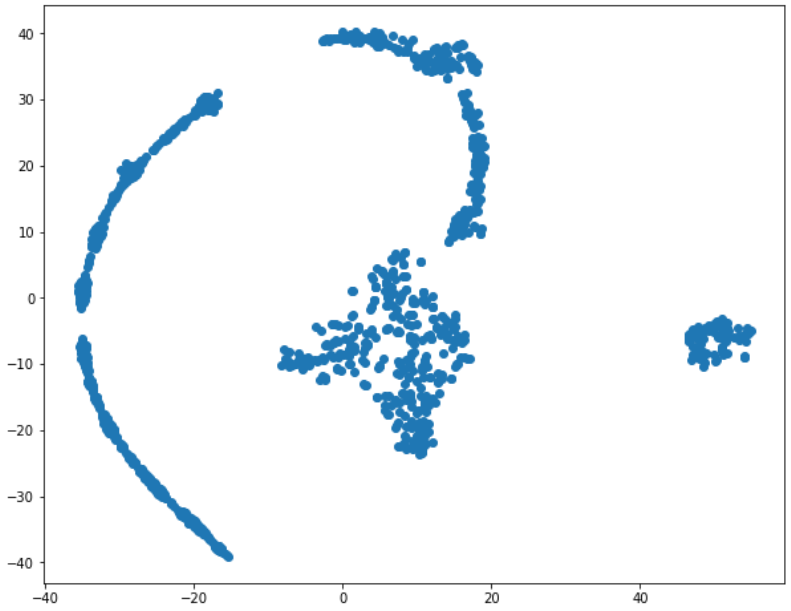
```
Wall time: 4.37 s
```

Out[72]:
```
TSNE(angle=0.5, early_exaggeration=12.0, init='random', learning_rate=200.0,
     method='barnes_hut', metric='euclidean', min_grad_norm=1e-07,
     n_components=2, n_iter=1000, n_iter_without_progress=300, perplexity=30.0,
     random_state=0, verbose=0)
```

In [73]:
```
tsne_1000_Random.embedding_.shape  # dimentions are 2 !
```

Out[73]:
```
(1000, 2)
```

In [74]:
```
# первая рандомная картинка с n_rand = 12.927.804
# print the matrix by features

plt.figure(num=1, figsize=(10,8))

plt.scatter(tsne_1000_Random.embedding_[:, 0], tsne_1000_Random.embedding_[:, 1]);
```



In [75]:
```
# one more random 1000 rows


n_rand = random.randint(0, data.shape[0] - 1000)
NN = 1000

indexes_to_drop1 = np.array(range(0,n_rand))
indexes_to_drop2 = np.array(range(n_rand+NN,data.shape[0]))
indexes_to_keep = set(range(data.shape[0])) - set(indexes_to_drop1) - set(indexes_to_drop2)
data5 = data.take(list(indexes_to_keep))

data5.info()
print(n_rand)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 991590 to 992589
Data columns (total 6 columns):
AskVolume    1000 non-null float32
BidVolume    1000 non-null float32
DateDelta    1000 non-null int32
Ask_delta    1000 non-null int32
Bid_delta    1000 non-null int32
Margin       1000 non-null int32
dtypes: float32(2), int32(4)
memory usage: 31.2 KB
991590
```

In [76]:
```
tsne_1000_Random = TSNE(random_state=0)
# random_state - randomness control parameter, depending on it there will be different pictures
# default output dimension n_components = 2(plane)
```

In [77]:
```
%%time
tsne_1000_Random.fit(data5)
```

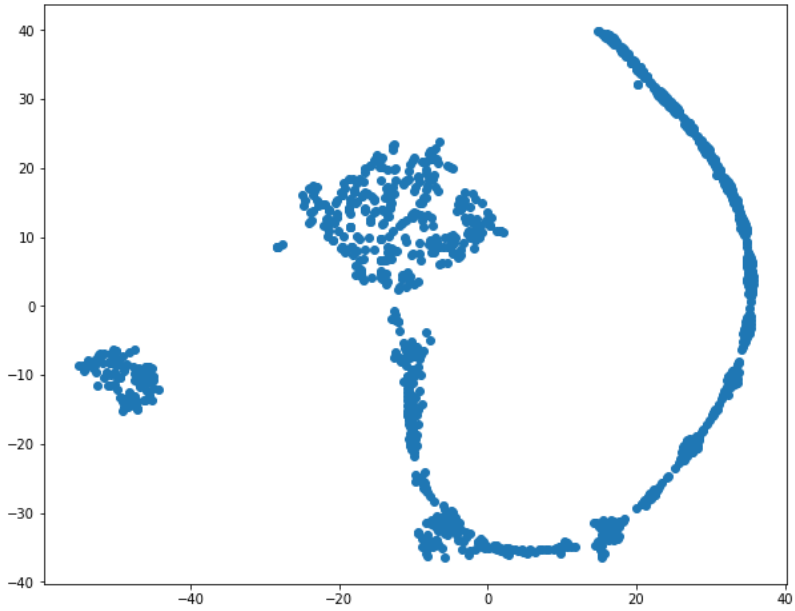
```
Wall time: 4.21 s
```

Out[77]:
```
TSNE(angle=0.5, early_exaggeration=12.0, init='random', learning_rate=200.0,
     method='barnes_hut', metric='euclidean', min_grad_norm=1e-07,
     n_components=2, n_iter=1000, n_iter_without_progress=300, perplexity=30.0,
     random_state=0, verbose=0)
```

In [78]:
```
tsne_1000_Random.embedding_.shape  # dimentions are 2 !
```

Out[78]:
```
(1000, 2)
```

In [79]:
```
# вторая рандомная картинка с n_rand = 9.332.757
# print the matrix by features

plt.figure(num=1, figsize=(10,8))

plt.scatter(tsne_1000_Random.embedding_[:, 0], tsne_1000_Random.embedding_[:, 1]);
```



Let's move the Delta for the price

Move up, those are now on the same line as the volumes, the interval with the previous tick and Delta by Ask and Bid with the FUTURE (!!) tick. We are looking for the dependence of the future price on the previous interval, volume and margin

In [80]:
```
data['Ask_delta'] = data['Ask_delta'].shift(-1)
data['Bid_delta'] = data['Bid_delta'].shift(-1)
data.info()
print(data.head(n=10))
print(data.tail(n=10))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16680595 entries, 0 to 16680594
Data columns (total 6 columns):
AskVolume    float32
BidVolume    float32
DateDelta    int32
Ask_delta    float64
Bid_delta    float64
Margin       int32
dtypes: float32(2), float64(2), int32(2)
memory usage: 509.1 MB
```

```
   AskVolume  BidVolume  DateDelta  Ask_delta  Bid_delta  Margin
0       0.75       3.75          0        0.0        1.0      84
1       0.75       2.25       1336        2.0        0.0      83
2       0.75       3.75        548        0.0       -1.0      85
3       0.75       3.75      16649        0.0        0.0      86
4       0.75       3.75       6273        0.0        1.0      86
5       0.75       3.00      10230        0.0        0.0      85
6       0.75       3.00       3963        7.0        8.0      85
7       1.12       0.75      41862      -13.0        0.0      84
8       0.75       0.75       1031        6.0        0.0      71
9       0.75       0.75       1062      -15.0        0.0      77
          AskVolume  BidVolume  DateDelta  Ask_delta  Bid_delta  Margin
16680585       2.25       1.87        251        0.0       -1.0       2
16680586       2.25       3.56        754       -1.0        0.0       3
16680587       1.00       3.19       1631        1.0        0.0       2
16680588       2.25       2.44        205       -1.0        0.0       3
16680589       1.00       3.75       2691        1.0        0.0       2
16680590       2.25       1.25       1489       -1.0        1.0       3
16680591       1.00       1.00        696        1.0       -1.0       1
16680592       1.50       1.25       3000       -1.0        0.0       3
16680593       1.00       1.25       1894        0.0       -1.0       2
16680594       1.50       1.50        971        NaN        NaN       3
```

In [81]:
```
data['Ask_delta'].loc[len(data)-1] = 0
data['Bid_delta'].loc[len(data)-1] = 0

data.info()
print(data.head(n=10))
print(data.tail(n=10))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16680595 entries, 0 to 16680594
Data columns (total 6 columns):
AskVolume    float32
BidVolume    float32
DateDelta    int32
Ask_delta    float64
Bid_delta    float64
Margin       int32
dtypes: float32(2), float64(2), int32(2)
memory usage: 509.1 MB
   AskVolume  BidVolume  DateDelta  Ask_delta  Bid_delta  Margin
0       0.75       3.75          0        0.0        1.0      84
1       0.75       2.25       1336        2.0        0.0      83
2       0.75       3.75        548        0.0       -1.0      85
3       0.75       3.75      16649        0.0        0.0      86
4       0.75       3.75       6273        0.0        1.0      86
5       0.75       3.00      10230        0.0        0.0      85
6       0.75       3.00       3963        7.0        8.0      85
7       1.12       0.75      41862      -13.0        0.0      84
8       0.75       0.75       1031        6.0        0.0      71
9       0.75       0.75       1062      -15.0        0.0      77

          AskVolume  BidVolume  DateDelta  Ask_delta  Bid_delta  Margin
16680585       2.25       1.87        251        0.0       -1.0       2
16680586       2.25       3.56        754       -1.0        0.0       3
16680587       1.00       3.19       1631        1.0        0.0       2
16680588       2.25       2.44        205       -1.0        0.0       3
16680589       1.00       3.75       2691        1.0        0.0       2
16680590       2.25       1.25       1489       -1.0        1.0       3
16680591       1.00       1.00        696        1.0       -1.0       1
16680592       1.50       1.25       3000       -1.0        0.0       3
16680593       1.00       1.25       1894        0.0       -1.0       2
16680594       1.50       1.50        971        0.0        0.0       3
```

Color active Ticks

In [319]:
```
# Create a new DataFrame - 'responses': Y
# Y = 0 - the next tick did not exceed the margin (the vast majority of ticks)
# Y = 1 - Ask and Bid of the next tick went up - a real break up
# Y = 2 - Ask of the next tick went up, and Bid remained the same (those margins expanded up)
# Y = 3 - Bid of the next tick went down, and Ask remained the same (those margins expanded down)
# Y = 4 - Ask and Bid of the next tick went down - a real break down
# Y = 5 - Margin has diverged: Ask has increased, and Bid has decreased

Y = pd.DataFrame(numpy.zeros(len(data.index)))
Y = pd.to_numeric(Y[0],downcast='integer')
Y
```

Out[319]:
```
0           0
1           0
2           0
3           0
4           0
           ..
16680590    0
16680591    0
16680592    0
16680593    0
16680594    0
Name: 0, Length: 16680595, dtype: int8
```

Create Y

In [ ]:
```
# real break up = 1
```

In [320]:
```
Y1 = ( ((data['Bid_delta'] - data['Margin']) >= 0) & ( (data['Ask_delta'] - data['Margin']) >= 0)).astype(int)
```

In [321]:
```
# count()
Y1.value_counts()
```

Out[321]:
```
0    16381544
1      299051
dtype: int64
```

In [301]:
```
# real break down = 4
```

In [351]:
```
Y4 = ( ((data['Bid_delta'] + data['Margin']) <= 0) & ( (data['Ask_delta'] + data['Margin']) <= 0) ).astype(int)
```

In [352]:
```
Y4[Y4==1] = Y4[Y4==1] + 3
```

In [353]:
```
Y4.value_counts()
```

Out[353]:
```
0    16377248
4      303347
dtype: int64
```

In [261]:
```
# margin expanded greatly up = 2
```

In [355]:
```
Y2 = ((data['Bid_delta'] == 0) & ( (data['Ask_delta'] - data['Margin']) >= 0) ).astype(int)
```

In [356]:
```
Y2[Y2==1] = Y2[Y2==1] + 1
```

In [357]:
```
Y2.value_counts()
```

Out[357]:
```
0    16188347
2      492248
dtype: int64
```

In [358]:
```
# margin expanded down = 3
```

In [359]:
```
Y3 = (((data['Bid_delta'] + data['Margin']) <= 0) & (data['Ask_delta'] == 0) ).astype(int)
```

In [360]:
```
Y3[Y3==1] = Y3[Y3==1] + 2
```

In [361]:
```
Y3.value_counts()
```

Out[361]:
```
0    16192485
3      488110
dtype: int64
```

In [ ]:
```
# margin diverged = 5
```

In [362]:
```
Y5 = ((data['Bid_delta'] < 0) &  (data['Ask_delta'] > 0) ).astype(int)
```

In [363]:
```
Y5[Y5==1] = Y5[Y5==1] + 4
```

In [364]:
```
Y5.value_counts()
```

Out[364]:
```
0    15897823
5      782772
dtype: int64
```

In [365]:
```
Y = Y1 + Y2 + Y3 + Y4 + Y5
```

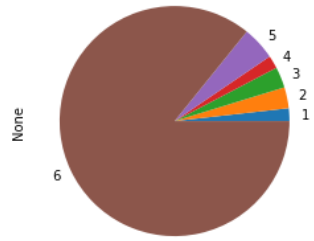
In [444]:
```
Y[Y==0] = Y[Y==0] + 6
```

In [445]:
```
Y.value_counts()
```

Out[445]:
```
6    14315067
5      782772
2      492248
3      488110
4      303347
1      299051
dtype: int64
```

In [ ]:
```
# pie()
```

In [446]:
```
Y.value_counts(sort=False).plot.pie()
plt.show()
```



t-SNE for The MNIST DataBase of handwritten digits

Example

In [97]:
```
# import the DataSet

from sklearn.datasets import load_digits
```

In [98]:
```
digits = load_digits()

X = digits.data
y = digits.target
```

In [99]:
```
X.shape
```

Out[99]:
```
(1797, 64)
```

In [100]:
```
X
```

Out[100]:
```
array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]])
```

In [101]:
```
y.shape
```

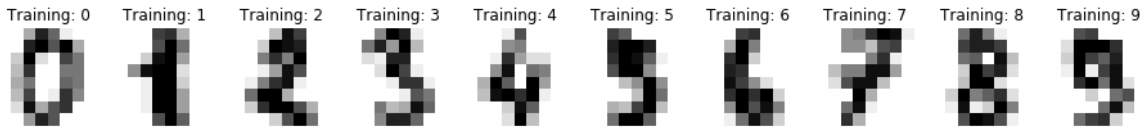
Out[101]:
```
(1797,)
```

In [102]:
```
y
```

Out[102]:
```
array([0, 1, 2, ..., 8, 9, 8])
```

In [103]:
```
# Look at the first ten digits
# These pictures are represented by an 8 x 8 matrix (white intensity for each pixel)
# Further, this matrix is "expanded" into a vector of length 64
# thus, the dimension of the attribute space is 64.
```

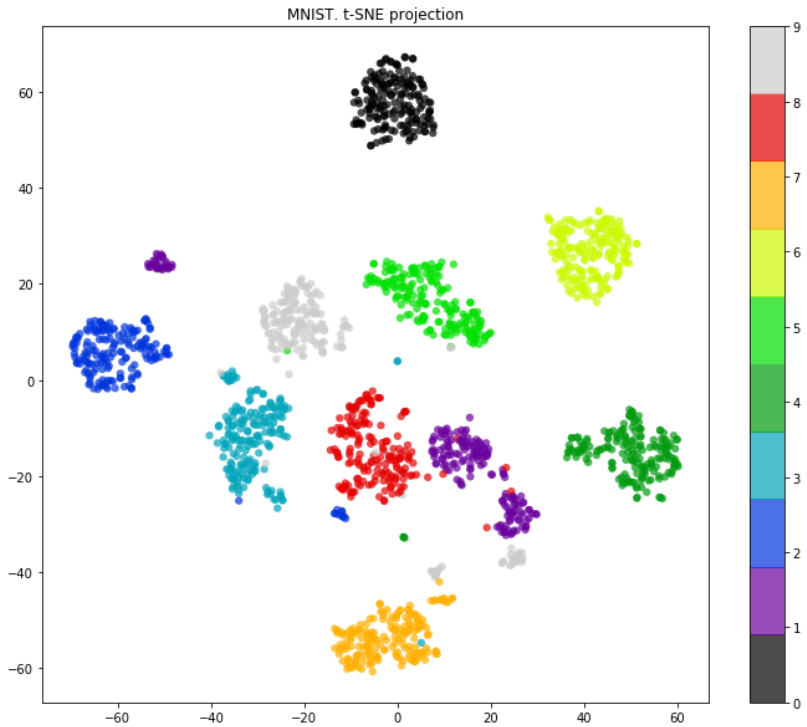
In [104]:
```
images_and_labels = list(zip(digits.images, digits.target))

plt.subplots(figsize=(16,6));

for index, (image, label) in enumerate(images_and_labels[:10]):
    plt.subplot(2, 10, index + 1)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Training: %i' % label)
```



In [105]:
```
%%time

tsne = TSNE(random_state=17)

X_tsne = tsne.fit_transform(X)

plt.figure(figsize=(12,10))
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y,
            edgecolor='none', alpha=0.7, s=40,
            cmap=plt.cm.get_cmap('nipy_spectral', 10))
plt.colorbar()
plt.title('MNIST. t-SNE projection')
```

```
Wall time: 9.03 s
```

Out[105]:
```
Text(0.5, 1.0, 'MNIST. t-SNE projection')
```



In [370]:
```
tsne.fit_transform?
```

In [ ]:
```

```

t-SNE

relative (random) data with deltas known in advance one step ahead at Ask and Bid prices

In [377]:
```
# random 1000 rows

import random

n_rand = random.randint(0, data.shape[0] - 1000)
NN = 1000

indexes_to_drop1 = np.array(range(0,n_rand))
indexes_to_drop2 = np.array(range(n_rand+NN,data.shape[0]))
indexes_to_keep = set(range(data.shape[0])) - set(indexes_to_drop1) - set(indexes_to_drop2)
data6 = data.take(list(indexes_to_keep))

data6.info()
print(n_rand)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 5797888 to 5797887
Data columns (total 6 columns):
AskVolume    1000 non-null float32
BidVolume    1000 non-null float32
DateDelta    1000 non-null int32
Ask_delta    1000 non-null float32
Bid_delta    1000 non-null float32
Margin       1000 non-null int32
dtypes: float32(4), int32(2)
memory usage: 31.2 KB
5797357
```

In [390]:
```
# Y size 1.000 random
Y6 = Y.take(list(indexes_to_keep))

print(Y6.size)
print(n_rand)
```

```
1000
5797357
```

In [434]:
```
Y6[Y6==0] = Y6[Y6==0] + 6
```

In [436]:
```

Y6[Y6==6] = Y6[Y6==6] - 6
```

In [435]:
```
Y6.value_counts()
```

Out[435]:
```
6    848
5     87
3     31
2     23
1      9
4      2
dtype: int64
```

In [384]:
```
tsne_1000_Random = TSNE(random_state=0)
# random_state - randomness control parameter, depending on it there will be different pictures
# default output dimension n_components = 2(plane)
```

In [385]:
```
%%time
tsne_1000_Random.fit(data6)
```

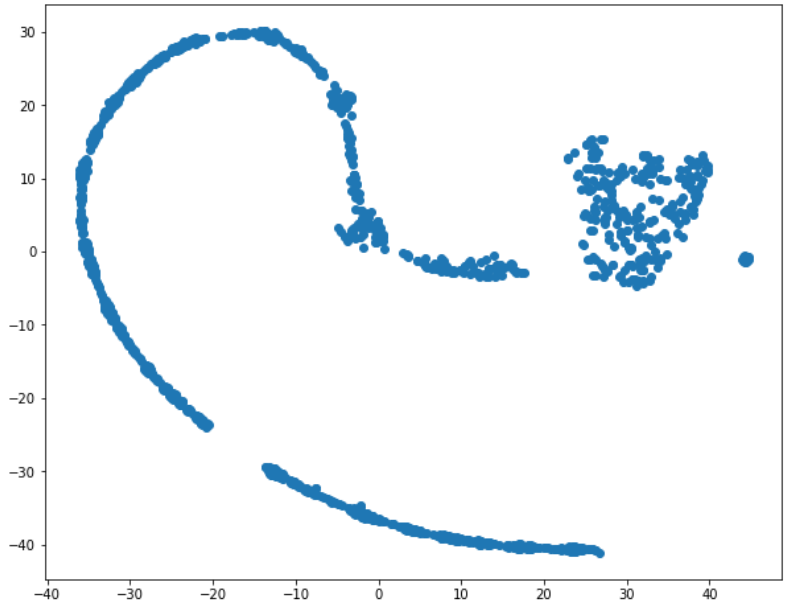
```
Wall time: 3.85 s
```

Out[385]:
```
TSNE(angle=0.5, early_exaggeration=12.0, init='random', learning_rate=200.0,
     method='barnes_hut', metric='euclidean', min_grad_norm=1e-07,
     n_components=2, n_iter=1000, n_iter_without_progress=300, perplexity=30.0,
     random_state=0, verbose=0)
```

In [386]:
```
tsne_1000_Random.embedding_.shape  # dimentions are 2 !
```

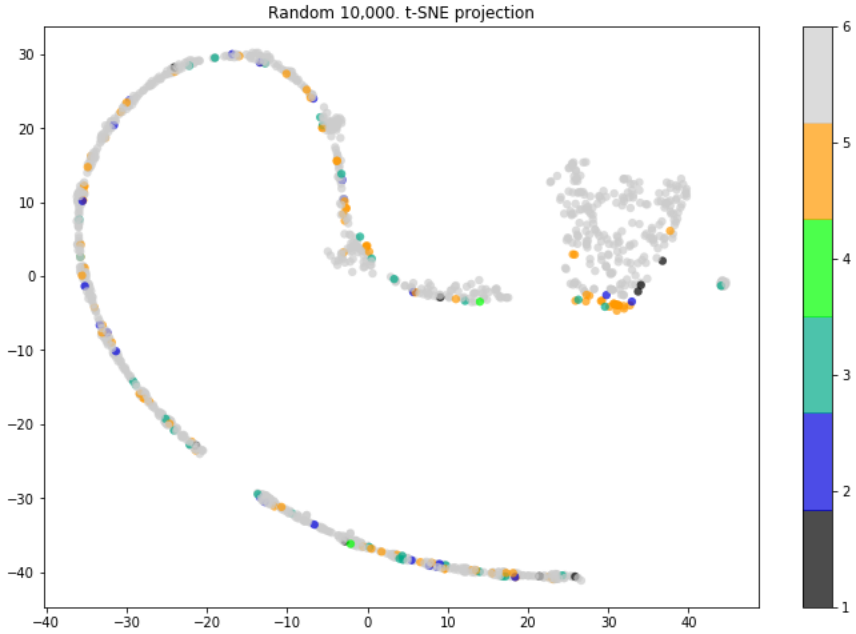
Out[386]:
```
(1000, 2)
```

In [387]:
```
# random with n_rand = 5.797.357 and NN = 1.000
# print the matrix by features

plt.figure(num=1, figsize=(10,8))

plt.scatter(tsne_1000_Random.embedding_[:, 0], tsne_1000_Random.embedding_[:, 1]);
```



In [393]:
```
plt.figure(num=1, figsize=(12,8))
plt.scatter(tsne_1000_Random.embedding_[:, 0], tsne_1000_Random.embedding_[:, 1],
            c=Y6,
            edgecolor='none', alpha=0.7, s=40,
            cmap=plt.cm.get_cmap('nipy_spectral', 6))
plt.colorbar()
plt.title('Random 10,000. t-SNE projection')
```

Out[393]:
```
Text(0.5, 1.0, 'Random 10,000. t-SNE projection')
```



In [394]:
```
# random 10.000 rows

import random

NN = 10000
n_rand = random.randint(0, data.shape[0] - NN)

indexes_to_drop1 = np.array(range(0,n_rand))
indexes_to_drop2 = np.array(range(n_rand+NN,data.shape[0]))
indexes_to_keep = set(range(data.shape[0])) - set(indexes_to_drop1) - set(indexes_to_drop2)
data7 = data.take(list(indexes_to_keep))

data7.info()
print(n_rand)
```

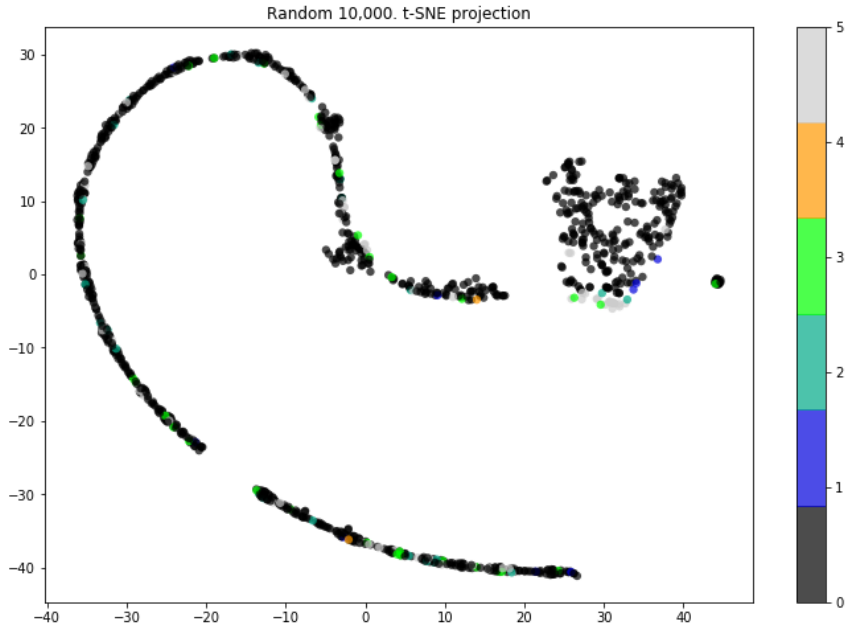
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 13618595 to 13628594
Data columns (total 6 columns):
AskVolume    10000 non-null float32
BidVolume    10000 non-null float32
DateDelta    10000 non-null int32
Ask_delta    10000 non-null float32
Bid_delta    10000 non-null float32
Margin       10000 non-null int32
dtypes: float32(4), int32(2)
memory usage: 312.5 KB
13618595
```

In [395]:
```
#  Y with size 10.000 of random ticks
Y7 = Y.take(list(indexes_to_keep))

print(Y7.size)
print(n_rand)
```

```
10000
13618595
```

In [437]:
```
plt.figure(num=1, figsize=(12,8))
plt.scatter(tsne_1000_Random.embedding_[:, 0], tsne_1000_Random.embedding_[:, 1],
            c=Y6,
            edgecolor='none', alpha=0.7, s=40,
            cmap=plt.cm.get_cmap('nipy_spectral', 6))
plt.colorbar()
plt.title('Random 10,000. t-SNE projection')
```

Out[437]:
```
Text(0.5, 1.0, 'Random 10,000. t-SNE projection')
```



In [394]:
```
# random 10.000 rows

import random

NN = 10000
n_rand = random.randint(0, data.shape[0] - NN)

indexes_to_drop1 = np.array(range(0,n_rand))
indexes_to_drop2 = np.array(range(n_rand+NN,data.shape[0]))
indexes_to_keep = set(range(data.shape[0])) - set(indexes_to_drop1) - set(indexes_to_drop2)
data7 = data.take(list(indexes_to_keep))

data7.info()
print(n_rand)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 13618595 to 13628594
Data columns (total 6 columns):
AskVolume    10000 non-null float32
BidVolume    10000 non-null float32
DateDelta    10000 non-null int32
Ask_delta    10000 non-null float32
Bid_delta    10000 non-null float32
Margin       10000 non-null int32
dtypes: float32(4), int32(2)
memory usage: 312.5 KB
13618595
```

In [395]:
```
#  Y with size 10.000 of random ticks
Y7 = Y.take(list(indexes_to_keep))

print(Y7.size)
print(n_rand)
```

```
10000
13618595
```

In [441]:
```
Y7[Y7==0] = Y7[Y7==0] + 6
```

In [428]:
```
Y7[Y7==6] = Y7[Y7==6] - 6
```

In [442]:
```
Y7.value_counts()
```

Out[442]:
```
6    7984
2     613
5     485
3     460
4     239
1     219
dtype: int64
```

In [400]:
```
tsne_10000_Random = TSNE(random_state=0)
# random_state - randomness control parameter, depending on it there will be different pictures
# default output dimension n_components = 2(plane)
```

In [401]:
```
%%time
tsne_10000_Random.fit(data7)
```

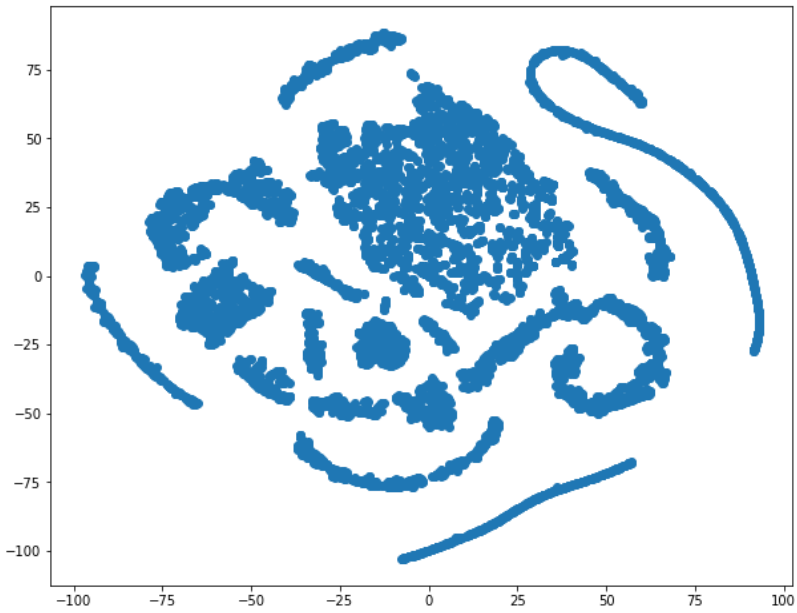
```
Wall time: 1min 11s
```

Out[401]:
```
TSNE(angle=0.5, early_exaggeration=12.0, init='random', learning_rate=200.0,
     method='barnes_hut', metric='euclidean', min_grad_norm=1e-07,
     n_components=2, n_iter=1000, n_iter_without_progress=300, perplexity=30.0,
     random_state=0, verbose=0)
```

In [402]:
```
tsne_10000_Random.embedding_.shape  # dimentions are 2 !
```

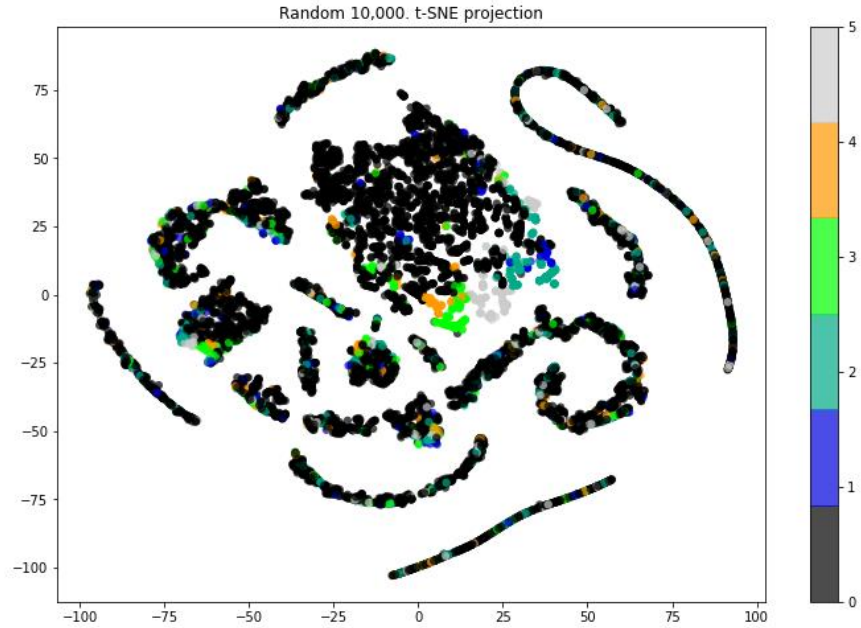
Out[402]:
```
(10000, 2)
```

In [403]:
```
# random with n_rand = 13.618.595 and NN = 10.000
# print the matrix by features

plt.figure(num=1, figsize=(10,8))

plt.scatter(tsne_10000_Random.embedding_[:, 0], tsne_10000_Random.embedding_[:, 1]);
```



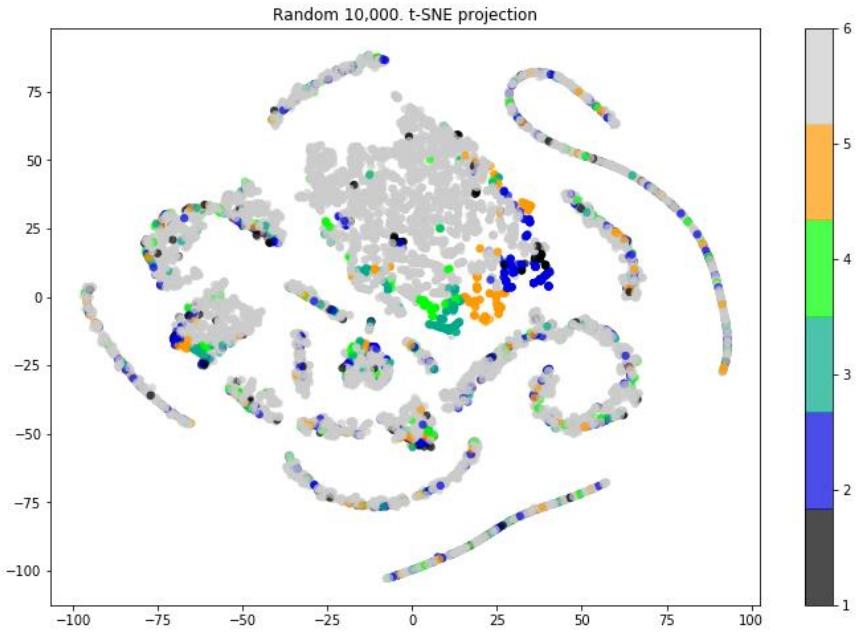
In [430]:
```
plt.figure(num=1, figsize=(12,8))
plt.scatter(tsne_10000_Random.embedding_[:, 0], tsne_10000_Random.embedding_[:, 1],
            c=Y7,
            edgecolor='none', alpha=0.7, s=40,
            cmap=plt.cm.get_cmap('nipy_spectral', 6))
plt.colorbar()
plt.title('Random 10,000. t-SNE projection')
```

Out[430]: Text(0.5, 1.0, 'Random 10,000. t-SNE projection')



In [443]:
```
plt.figure(num=1, figsize=(12,8))
plt.scatter(tsne_10000_Random.embedding_[:, 0], tsne_10000_Random.embedding_[:, 1],
            c=Y7,
            edgecolor='none', alpha=0.7, s=40,
            cmap=plt.cm.get_cmap('nipy_spectral', 6))
plt.colorbar()
plt.title('Random 10,000. t-SNE projection')
```

Out[443]: Text(0.5, 1.0, 'Random 10,000. t-SNE projection')



In [405]:
```
# random 50.000 rows

import random

NN = 50000
n_rand = random.randint(0, data.shape[0] - NN)

indexes_to_drop1 = np.array(range(0,n_rand))
indexes_to_drop2 = np.array(range(n_rand+NN,data.shape[0]))
indexes_to_keep = set(range(data.shape[0])) - set(indexes_to_drop1) - set(indexes_to_drop2)
data8 = data.take(list(indexes_to_keep))

data8.info()
print(n_rand)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50000 entries, 14417920 to 14417919
Data columns (total 6 columns):
AskVolume    50000 non-null float32
BidVolume    50000 non-null float32
DateDelta    50000 non-null int32
Ask_delta    50000 non-null float32
Bid_delta    50000 non-null float32
Margin       50000 non-null int32
dtypes: float32(4), int32(2)
memory usage: 1.5 MB
14398304
```

In [406]:
```
#  Y size 50.000 random ticks
Y8 = Y.take(list(indexes_to_keep))

print(Y8.size)
print(n_rand)
```

```
50000
14398304
```

In [438]:
```
Y8[Y8==0] = Y8[Y8==0] + 6
```

In [431]:
```
Y8[Y8==6] = Y8[Y8==6] - 6
```

In [432]:
```
Y8.value_counts()
```

Out[432]:
```
0    41240
5     2538
3     1936
2     1766
4     1282
1     1238
dtype: int64
```

In [410]:
```
tsne_50000_Random = TSNE(random_state=0)
# random_state - randomness control parameter, depending on it there will be different pictures
# default output dimension n_components = 2(plane)
```

In [411]:
```
%%time
tsne_50000_Random.fit(data8)
```

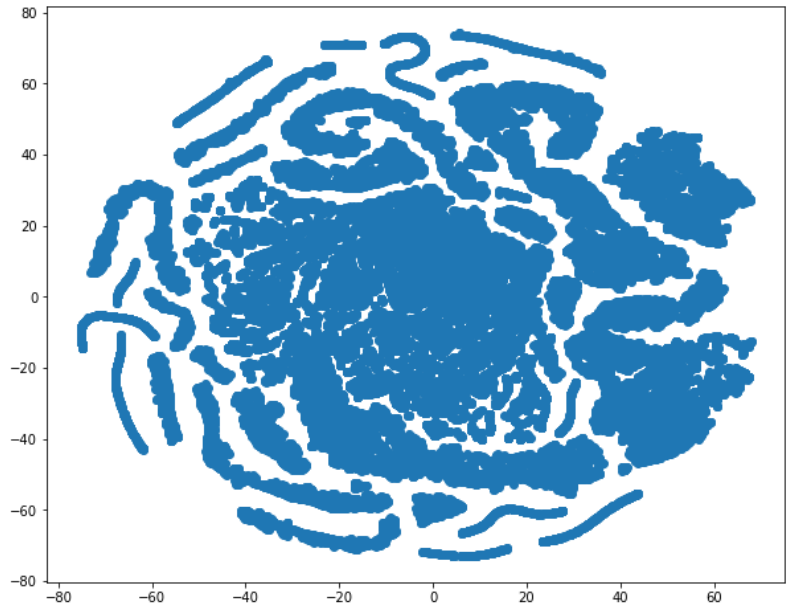
```
Wall time: 8min 40s
```

Out[411]:
```
TSNE(angle=0.5, early_exaggeration=12.0, init='random', learning_rate=200.0,
     method='barnes_hut', metric='euclidean', min_grad_norm=1e-07,
     n_components=2, n_iter=1000, n_iter_without_progress=300, perplexity=30.0,
     random_state=0, verbose=0)
```

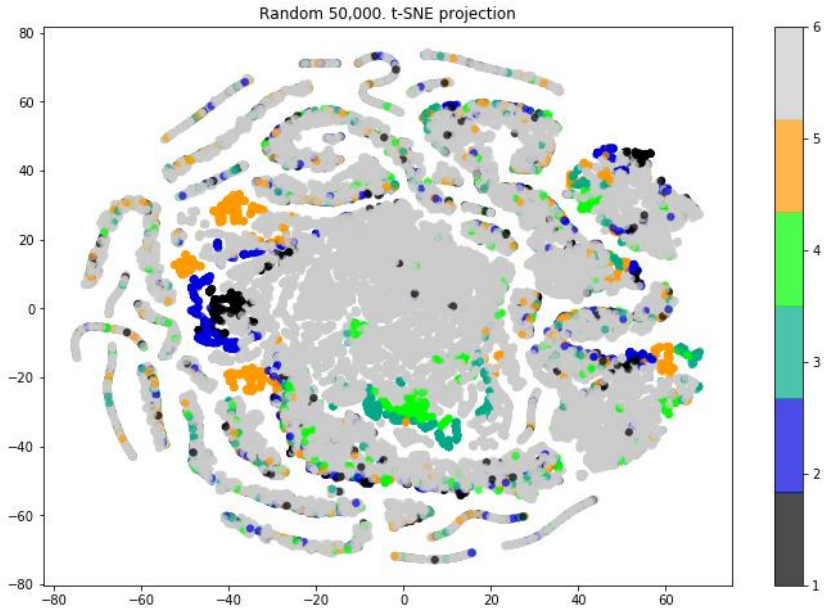
In [412]:
```
tsne_50000_Random.embedding_.shape  # dimentions are 2 !
```

Out[412]:
```
(50000, 2)
```

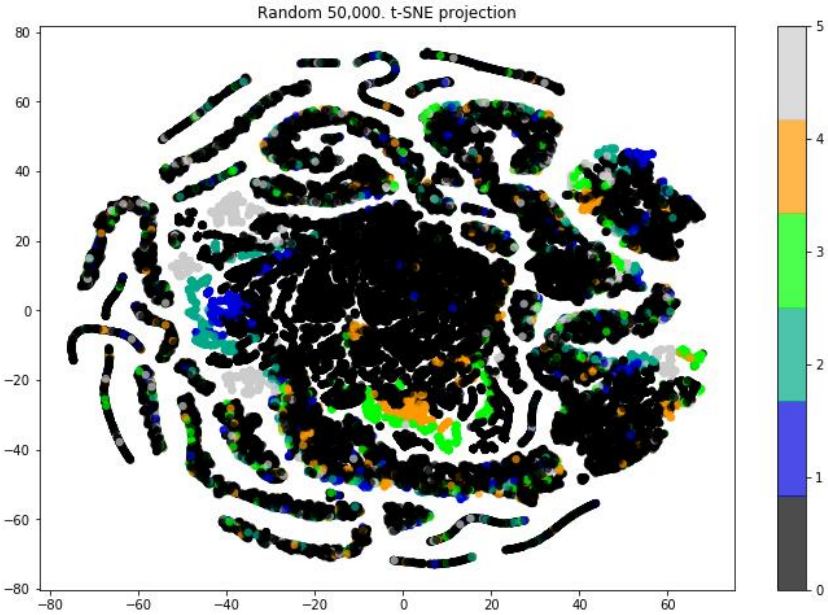
In [413]:
```
# random with c n_rand = 14.398.304 and NN = 50.000
# print the matrix by features

plt.figure(num=1, figsize=(10,8))

plt.scatter(tsne_50000_Random.embedding_[:, 0], tsne_50000_Random.embedding_[:, 1]);
```



In [439]:
```
plt.figure(num=1, figsize=(12,8))
plt.scatter(tsne_50000_Random.embedding_[:, 0], tsne_50000_Random.embedding_[:, 1],
            c=Y8,
            edgecolor='none', alpha=0.7, s=40,
            cmap=plt.cm.get_cmap('nipy_spectral', 6))
plt.colorbar()
plt.title('Random 50,000. t-SNE projection');
```



In [433]:
```
plt.figure(num=1, figsize=(12,8))
plt.scatter(tsne_50000_Random.embedding_[:, 0], tsne_50000_Random.embedding_[:, 1],
            c=Y8,
            edgecolor='none', alpha=0.7, s=40,
            cmap=plt.cm.get_cmap('nipy_spectral', 6))
plt.colorbar()
plt.title('Random 50,000. t-SNE projection');
```



In [415]:
```
# random 100.000 rows

import random

NN = 100000
n_rand = random.randint(0, data.shape[0] - NN)

indexes_to_drop1 = np.array(range(0,n_rand))
indexes_to_drop2 = np.array(range(n_rand+NN,data.shape[0]))
indexes_to_keep = set(range(data.shape[0])) - set(indexes_to_drop1) - set(indexes_to_drop2)
data9 = data.take(list(indexes_to_keep))

data9.info()
print(n_rand)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100000 entries, 14680064 to 14680063
Data columns (total 6 columns):
AskVolume    100000 non-null float32
BidVolume    100000 non-null float32
DateDelta    100000 non-null int32
Ask_delta    100000 non-null float32
Bid_delta    100000 non-null float32
Margin       100000 non-null int32
dtypes: float32(4), int32(2)
memory usage: 3.1 MB
14616272
```

In [416]:
```
Y9 = Y.take(list(indexes_to_keep))

print(Y9.size)
print(n_rand)
```

```
100000
14616272
```

In [417]:
```
Y9[Y9==0] = Y9[Y9==0] + 6
```

In [449]:
```

Y9[Y9==6] = Y9[Y9==6] - 6
```

In [450]:
```
Y9.value_counts()
```

Out[450]:
```
0    83805
5     4973
1     2859
2     2831
3     2788
4     2744
dtype: int64
```

In [420]:
```
tsne_100000_Random = TSNE(random_state=0)
# random_state - randomness control parameter, depending on it there will be different pictures
# default output dimension n_components = 2(plane)
```

In [421]:
```
%%time
tsne_100000_Random.fit(data9)
```

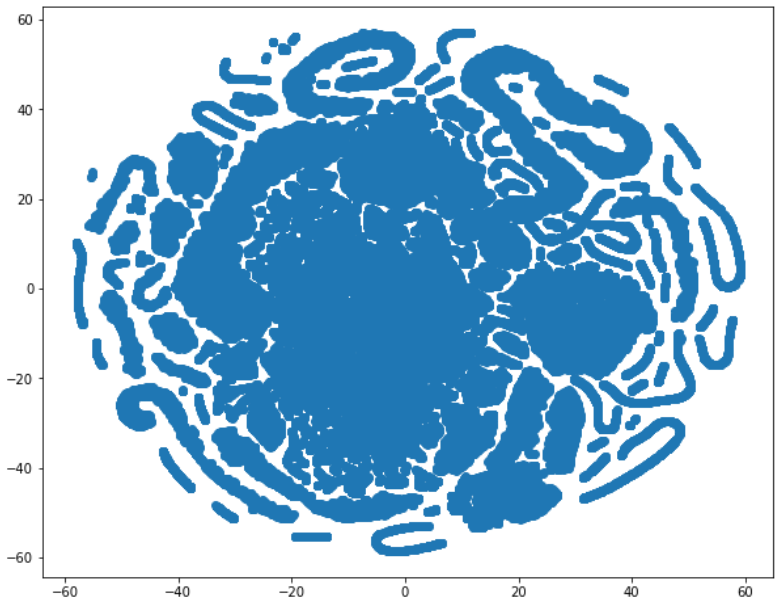
```
Wall time: 19min 25s
```

Out[421]:
```
TSNE(angle=0.5, early_exaggeration=12.0, init='random', learning_rate=200.0,
     method='barnes_hut', metric='euclidean', min_grad_norm=1e-07,
     n_components=2, n_iter=1000, n_iter_without_progress=300, perplexity=30.0,
     random_state=0, verbose=0)
```

In [422]:
```
tsne_100000_Random.embedding_.shape  # dimentions are 2 !
```

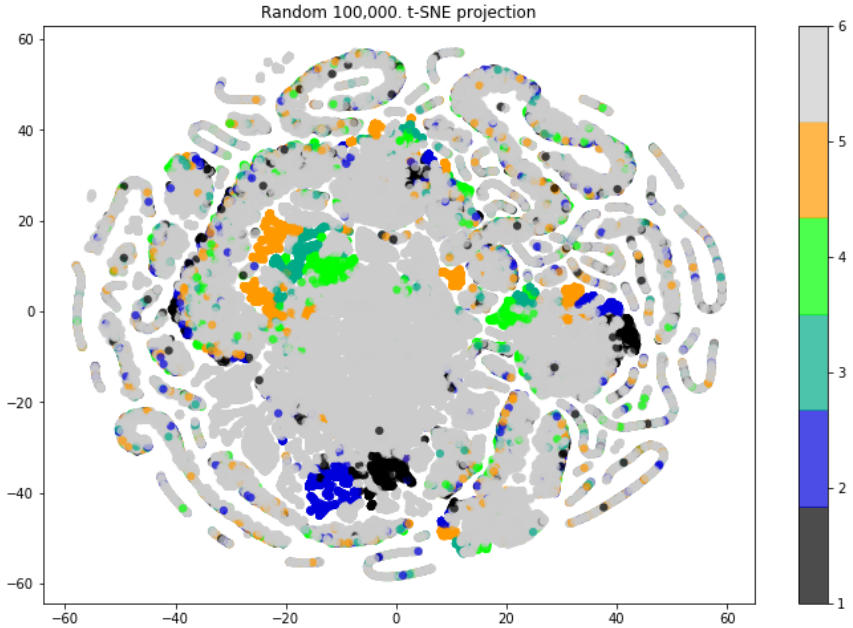
Out[422]:
```
(100000, 2)
```

In [423]:
```
# NN = 100.000
# print the matrix by features

plt.figure(num=1, figsize=(10,8))

plt.scatter(tsne_100000_Random.embedding_[:, 0], tsne_100000_Random.embedding_[:, 1]);
```



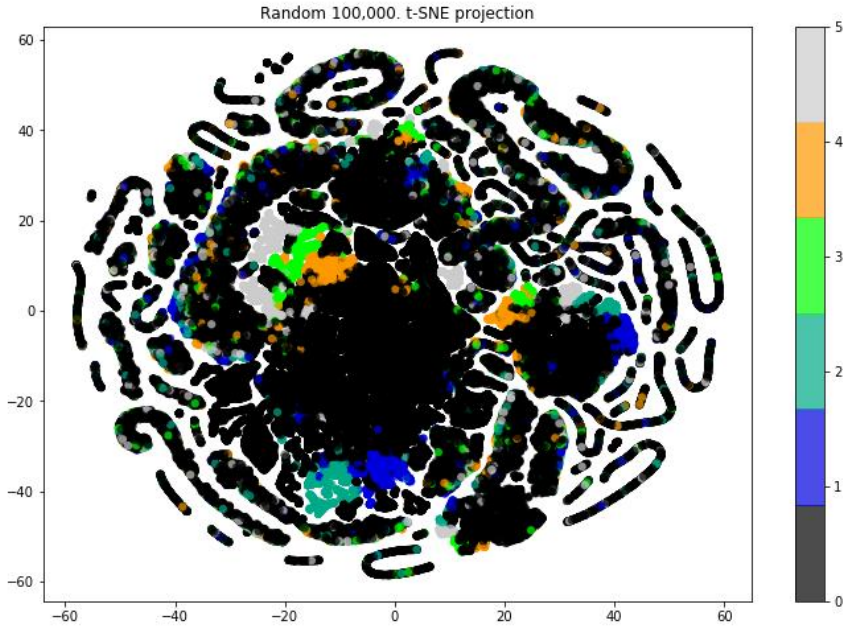
In [425]:
```
plt.figure(num=1, figsize=(12,8))
plt.scatter(tsne_100000_Random.embedding_[:, 0], tsne_100000_Random.embedding_[:, 1],
            c=Y9,
            edgecolor='none', alpha=0.7, s=40,
            cmap=plt.cm.get_cmap('nipy_spectral', 6))
plt.colorbar()
plt.title('Random 100,000. t-SNE projection')
```

Out[425]: Text(0.5, 1.0, 'Random 100,000. t-SNE projection')



In [451]:
```
plt.figure(num=1, figsize=(12,8))
plt.scatter(tsne_100000_Random.embedding_[:, 0], tsne_100000_Random.embedding_[:, 1],
            c=Y9,
            edgecolor='none', alpha=0.7, s=40,
            cmap=plt.cm.get_cmap('nipy_spectral', 6))
plt.colorbar()
plt.title('Random 100,000. t-SNE projection')
```

Out[451]: Text(0.5, 1.0, 'Random 100,000. t-SNE projection')


Random 100,000. t-SNE projection
60
40
20
0
−20
−40
−60
−60
−40
−20
0
20
40
60
5
4
3
2
1
0

Appendix VIII.

Predicting S & P-500 Index Prices by Multilayer Perceptron

A fully connected (multilayer perceptron) neural network with four hidden layers is used.

Purpose: according to the dynamics of prices of 500 stocks, to predict the dynamics of prices of futures S&P

Exchange minute data in csv (41.266 minutes) includes 500 shares + S & P

In [1]:
```
from keras.models import Sequential
from keras.layers import Dense

from keras import utils

import numpy as np
import pandas as pd

from sklearn.preprocessing import MinMaxScaler

import matplotlib.pyplot as plt
%matplotlib inline
```

```
Using TensorFlow backend.
```

In [2]:
```
np.random.seed(42)
```

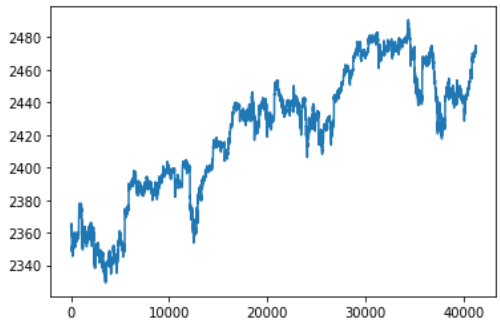
Data

In [3]:
```
data = pd.read_csv('data_stocks.csv')

print(data)
```

```
            DATE      SP500  NASDAQ.AAL  NASDAQ.AAPL  NASDAQ.ADBE  \
0      1491226200  2363.6101     42.3300     143.6800     129.6300
1      1491226260  2364.1001     42.3600     143.7000     130.3200
2      1491226320  2362.6799     42.3100     143.6901     130.2250
3      1491226380  2364.3101     42.3700     143.6400     130.0729
4      1491226440  2364.8501     42.5378     143.6600     129.8800
...           ...        ...         ...          ...          ...
41261  1504209360  2472.2200     44.7200     164.1100     155.0900
41262  1504209420  2471.7700     44.7300     164.1200     155.1600
41263  1504209480  2470.0300     44.7400     164.0100     155.0650
41264  1504209540  2471.4900     44.7100     163.8800     154.9600
41265  1504209600  2471.4900     44.7400     163.9800     155.1600

       NASDAQ.ADI  NASDAQ.ADP  NASDAQ.ADSK  NASDAQ.AKAM  NASDAQ.ALXN  ...  \
0          82.040    102.2300      85.2200       59.760      121.520  ...
1          82.080    102.1400      85.6500       59.840      121.480  ...
2          82.030    102.2125      85.5100       59.795      121.930  ...
3          82.000    102.1400      85.4872       59.620      121.440  ...
4          82.035    102.0600      85.7001       59.620      121.600  ...
...           ...         ...          ...          ...          ...  ...
41261      83.670    106.5650     114.4900       47.150      142.425  ...
41262      83.650    106.5900     114.5200       47.150      142.450  ...
41263      83.620    106.5200     114.4700       47.150      142.330  ...
41264      83.580    106.4000     114.3300       47.135      142.170  ...
41265      83.690    106.4700     114.4600       47.150      142.410  ...
```

```
       NYSE.WYN  NYSE.XEC  NYSE.XEL  NYSE.XL  NYSE.XOM  NYSE.XRX  NYSE.XYL  \
0        84.370   119.035    44.400   39.880    82.030     7.360    50.220
1        84.370   119.035    44.110   39.880    82.030     7.380    50.220
2        84.585   119.260    44.090   39.980    82.020     7.360    50.120
3        84.460   119.260    44.250   39.990    82.020     7.350    50.160
4        84.470   119.610    44.110   39.960    82.030     7.360    50.200
...         ...       ...       ...      ...       ...       ...       ...
41261    99.675    99.530    49.485   40.955    76.360    32.285    62.110
41262    99.730    99.630    49.480   40.960    76.370    32.295    62.100
41263    99.735    99.640    49.495   40.940    76.315    32.290    62.090
41264    99.700    99.630    49.485   40.925    76.300    32.275    62.075
41265    99.670    99.640    49.490   40.940    76.320    32.270    62.070

       NYSE.YUM  NYSE.ZBH  NYSE.ZTS
0         63.86   122.000   53.3500
1         63.74   121.770   53.3500
2         63.75   121.700   53.3650
3         63.88   121.700   53.3800
4         63.91   121.695   53.2400
...         ...       ...       ...
41261     76.88   114.310   62.7250
41262     76.90   114.330   62.7100
41263     76.88   114.310   62.6850
41264     76.83   114.230   62.6301
41265     76.81   114.280   62.6800

[41266 rows x 502 columns]
```

In [4]:
```
plt.plot(data['SP500'])
plt.show()
```



In [5]:
```
data_m = data.drop(['DATE'], 1)
```

In [6]:
```
n = data_m.shape[0]
p = data_m.shape[1]

data_a = np.array(data_m)

print(data_a)

print("data_a:")
print("n = ", n, " p = ", p)
print("type: ", data_a.dtype, type(data_a), "; Size: ", data_a.shape)
```

```
[[2363.6101   42.33    143.68   ...   63.86    122.       53.35  ]
 [2364.1001   42.36    143.7    ...   63.74    121.77     53.35  ]
 [2362.6799   42.31    143.6901 ...   63.75    121.7      53.365 ]
 ...
 [2470.03     44.74    164.01   ...   76.88    114.31     62.685 ]
 [2471.49     44.71    163.88   ...   76.83    114.23     62.6301]
 [2471.49     44.74    163.98   ...   76.81    114.28     62.68  ]]
data_a:
n =  41266  p =  501
type:  float64 <class 'numpy.ndarray'> ; Size:  (41266, 501)
```

target: predicting the value of the S & P-500 index in the near future (in our example, the S & P-500 value in the next minute). To do this, shift the S & P-500 forward one minute

In [7]:
```
# shift the values of the column 'SP500' one line up
for i in range(n-1):
    data_a[i,0] = data_a[i+1,0]

print(data_a)

print("data_a")
print("n = ", n, " p = ", p)
print("type ", data_a.dtype, type(data_a), "; Size: ", data_a.shape)
```

```
[[2364.1001   42.33    143.68   ...   63.86    122.       53.35  ]
 [2362.6799   42.36    143.7    ...   63.74    121.77     53.35  ]
 [2364.3101   42.31    143.6901 ...   63.75    121.7      53.365 ]
 ...
 [2471.49     44.74    164.01   ...   76.88    114.31     62.685 ]
 [2471.49     44.71    163.88   ...   76.83    114.23     62.6301]
 [2471.49     44.74    163.98   ...   76.81    114.28     62.68  ]]
data_a
n =  41266  p =  501
type  float64 <class 'numpy.ndarray'> ; Size:  (41266, 501)
```

In [8]:
```
print(data_a[-5:,:])
```

```
[[2471.77     44.72    164.11   ...   76.88    114.31     62.725 ]
 [2470.03     44.73    164.12   ...   76.9     114.33     62.71  ]
 [2471.49     44.74    164.01   ...   76.88    114.31     62.685 ]
 [2471.49     44.71    163.88   ...   76.83    114.23     62.6301]
 [2471.49     44.74    163.98   ...   76.81    114.28     62.68  ]]
```

In [9]:
```
# delete the last row of the array, because for it we do not have the S & P-500 value in the next minute
data_a = np.delete(data_a,(n-1), axis=0)

print(data_a[-5:,:])

print("data_a")
print("type ", data_a.dtype, type(data_a), "; Size: ", data_a.shape)
```

```
[[2472.22     44.71    164.14   ...   76.885   114.3      62.73  ]
 [2471.77     44.72    164.11   ...   76.88    114.31     62.725 ]
 [2470.03     44.73    164.12   ...   76.9     114.33     62.71  ]
 [2471.49     44.74    164.01   ...   76.88    114.31     62.685 ]
 [2471.49     44.71    163.88   ...   76.83    114.23     62.6301]]
data_a
type  float64 <class 'numpy.ndarray'> ; Size:  (41265, 501)
```

Data for testing

In [10]:
```
# We allocate approximately 80% of the data (rows) for training and approximately 20% of the data (rows) are reserved for testin
g
n = data_a.shape[0]
train_start = 0
train_end = int(np.floor(0.8*n))
print(train_end)
```

```
33012
```

In [11]:
```
train_end -= 12
print(train_end)
```

```
33000
```

In [12]:
```
test_start = train_end + 1
test_end = n

# divide the data array into two unequal parts
data_train = data_a[np.arange(train_start, train_end), :]
data_test = data_a[np.arange(test_start, test_end), :]
```

In [13]:
```
print('Data for training:')
print(data_train.shape)
print('Data for testing:')
print(data_test.shape)
```

```
Data for training:
(33000, 501)
Data for testing:
(8264, 501)
```

Data normalisation

In [14]:
```
scaler = MinMaxScaler()
scaler.fit(data_train)
data_train = scaler.transform(data_train)
data_test = scaler.transform(data_test)

print('Data for training ',"data_train:")
print("Type: ", data_train.dtype, type(data_train), "; Size: ", data_train.shape)

print("\n",'Data or testing ',"data_test:")
print("Type: ", data_test.dtype,  type(data_test), "; Size: ", data_test.shape)
```

```
Data for training  data_train:
Type:  float64 <class 'numpy.ndarray'> ; Size:  (33000, 501)

 Data or testing  data_test:
Type:  float64 <class 'numpy.ndarray'> ; Size:  (8264, 501)
```

create a matrix of input parameters and a response vector for training and test parts

In [16]:
```
# create two data sets: for training: (X_train, y_train)
# and for tests: (X_test, y_test)
# X_train and X_test are tables with minute stock prices
# y_train and y_test are S & P-500 values in the next minute
```

In [15]:
```
X_train = data_train[:, 1:]
y_train = data_train[:, 0]
X_test = data_test[:, 1:]
y_test = data_test[:, 0]
```

In [16]:
```
# print X_train, y_train, X_test and y_test

print('Data for training:')
print("X_train:")
print("Type: ", X_train.dtype, type(X_train), "; Size: ", X_train.shape)

print("\n","y_train:")
print("Type: ", y_train.dtype, type(y_train), "; Size: ", y_train.shape)

print('Data for testing:')
print("\n","X_test:")
print("Type: ", X_test.dtype,type(X_test), "; Size: ", X_test.shape)

print("\n","y_test:")
print("Type: ", y_test.dtype, type(y_test), "; Size: ", y_test.shape)
```

```
Data for training:
X_train:
Type:  float64 <class 'numpy.ndarray'> ; Size:  (33000, 500)

 y_train:
Type:  float64 <class 'numpy.ndarray'> ; Size:  (33000,)
Data for testing:

 X_test:
Type:  float64 <class 'numpy.ndarray'> ; Size:  (8264, 500)

 y_test:
Type:  float64 <class 'numpy.ndarray'> ; Size:  (8264,)
```

create Neural Network

In [17]:
```
model = Sequential()

model.add(Dense(1024, input_dim=500, activation="relu", kernel_initializer="normal"))

model.add(Dense(512, activation="relu", kernel_initializer="normal"))

model.add(Dense(256, activation="relu", kernel_initializer="normal"))

model.add(Dense(128, activation="relu", kernel_initializer="normal"))

model.add(Dense(1))

print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              513024
_________________________________________________________________
dense_2 (Dense)              (None, 512)               524800
_________________________________________________________________
dense_3 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_4 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,202,177
Trainable params: 1,202,177
Non-trainable params: 0
_________________________________________________________________
None
```

Compile the model

In [18]:
```
model.compile(optimizer="adam", loss="mse", metrics=["mae"])
```

Training NN

In [19]:
```
history = model.fit(X_train, y_train, batch_size=2, epochs=15, validation_split=0.2, verbose=1)
```

```
Train on 26400 samples, validate on 6600 samples
Epoch 1/15
26400/26400 [==============================] - 48s 2ms/step - loss: 8.3030e-04 - mae: 0.0184 - val_loss: 0.0071 - val_mae: 0.07
68
Epoch 2/15
26400/26400 [==============================] - 47s 2ms/step - loss: 1.6087e-04 - mae: 0.0096 - val_loss: 0.0032 - val_mae: 0.05
21
Epoch 3/15
26400/26400 [==============================] - 49s 2ms/step - loss: 1.0910e-04 - mae: 0.0079 - val_loss: 0.0017 - val_mae: 0.03
72
Epoch 4/15
26400/26400 [==============================] - 48s 2ms/step - loss: 9.0482e-05 - mae: 0.0072 - val_loss: 0.0020 - val_mae: 0.04
12
Epoch 5/15
26400/26400 [==============================] - 47s 2ms/step - loss: 8.5976e-05 - mae: 0.0070 - val_loss: 7.7292e-04 - val_mae:
0.0241
Epoch 6/15
26400/26400 [==============================] - 47s 2ms/step - loss: 7.8457e-05 - mae: 0.0067 - val_loss: 5.6890e-04 - val_mae:
0.0199
Epoch 7/15
26400/26400 [==============================] - 47s 2ms/step - loss: 7.6018e-05 - mae: 0.0066 - val_loss: 7.4006e-04 - val_mae:
0.0218
Epoch 8/15
26400/26400 [==============================] - 47s 2ms/step - loss: 7.2076e-05 - mae: 0.0064 - val_loss: 4.4217e-04 - val_mae:
0.0167
Epoch 9/15
26400/26400 [==============================] - 48s 2ms/step - loss: 6.9581e-05 - mae: 0.0062 - val_loss: 5.9766e-04 - val_mae:
0.0189
Epoch 10/15
26400/26400 [==============================] - 47s 2ms/step - loss: 6.7049e-05 - mae: 0.0061 - val_loss: 6.1169e-04 - val_mae:
0.0197
```

```
Epoch 11/15
26400/26400 [==============================] - 46s 2ms/step - loss: 6.3644e-05 - mae: 0.0059 - val_loss: 7.3719e-04 - val_mae:
0.0208
Epoch 12/15
26400/26400 [==============================] - 47s 2ms/step - loss: 6.2211e-05 - mae: 0.0059 - val_loss: 7.6822e-04 - val_mae:
0.0210
Epoch 13/15
26400/26400 [==============================] - 47s 2ms/step - loss: 6.2264e-05 - mae: 0.0058 - val_loss: 6.4980e-04 - val_mae:
0.0201
Epoch 14/15
26400/26400 [==============================] - 47s 2ms/step - loss: 5.9963e-05 - mae: 0.0058 - val_loss: 7.7279e-04 - val_mae:
0.0215
Epoch 15/15
26400/26400 [==============================] - 47s 2ms/step - loss: 5.9814e-05 - mae: 0.0058 - val_loss: 0.0013 - val_mae: 0.02
68
```

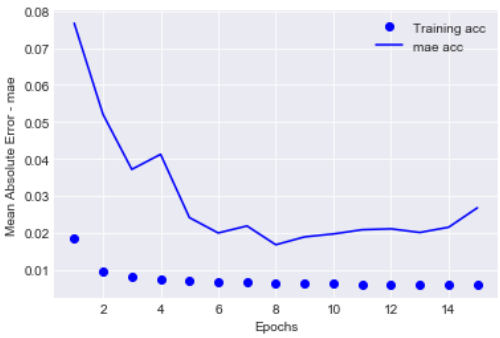
Evaluation

In [21]:
```
# Оцениваем качество обучения сети на тестовых данных (которые сеть НЕ видела)
mse, mae = model.evaluate(X_test, y_test, verbose=0)
print("Mean absolute error: %.2f%%" % (mae*100))
```

```
Mean absolute error: 4.04%
```

Training accuracy graphs

In [25]:
```
history_dict = history.history
mean_absolute_error_values = history_dict['mae']
val_mean_absolute_error_values = history_dict['val_mae']
epochs = range(1, len(mean_absolute_error_values) + 1)
plt.plot(epochs, mean_absolute_error_values, 'bo', label='Training acc')
plt.plot(epochs, val_mean_absolute_error_values, 'b', label='mae acc')
plt.xlabel('Epochs')
plt.ylabel('Mean Absolute Error - mae')
plt.legend()
plt.show()
```



Using neural networks to predict the price of the S & P-500

In [26]:
```
import random

pred = model.predict(X_test)

n_rec = random.randint(0, X_test.shape[0])

print("Predicted price:", pred[n_rec][0], ", the right price:", y_test[n_rec], ", error:", pred[n_rec][0]-y_test[n_rec],
      "%error:", (pred[n_rec][0]-y_test[n_rec])/y_test[n_rec]*100,"%")
```

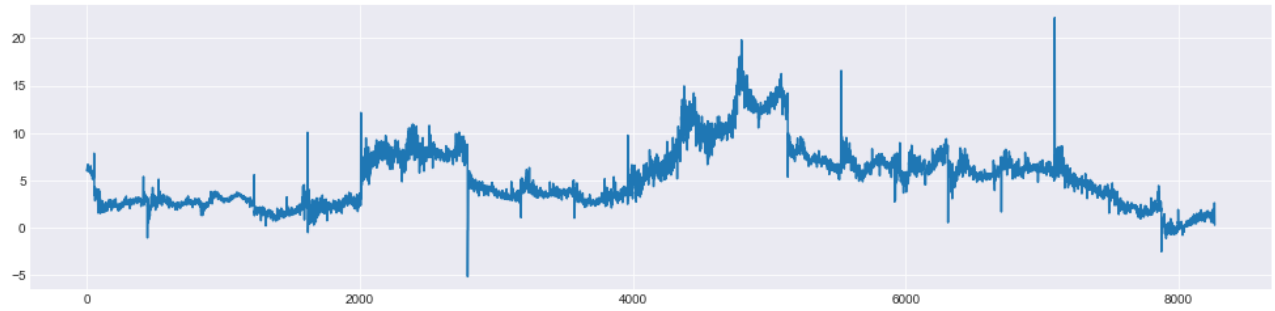
```
Predicted price: 0.9195916 , the right price: 0.8870742378530725 , error: 0.03251736781022707 %error: 3.6656873148437636 %
```

Price chart

In [27]:
```
plt.style.use('seaborn-darkgrid')
fig = plt.figure(figsize=(17,4))
ax1 = fig.add_subplot(111)
line1, = ax1.plot(y_test)
line2, = ax1.plot(pred[:,0])
plt.show()
```

Error Chart

In [28]:
```
plt.style.use('seaborn-darkgrid')
plt.ion()
fig = plt.figure(figsize=(17,4))
ax1 = fig.add_subplot(111)
a = (pred[:,0]-y_test[:])/y_test[:]*100
line1, = ax1.plot(a)
plt.show()
```



Appendix IX.

Cleaning, Normalisation and Preparation Data for the First Series of the Experiments

Data Normalisation and Preparation

Price, Volume, Inter-tick intervals

Forex EUR/USD from Dukastcopy Bank

Price normalisation: subtract the minimum divide by the maximum price among 200 ticks that include this tick

Normalisation of volumes - based on a maximum of 10 lots (the main number of lots). The interval (from 10 to 5000 lots) are equated to 10 lots (those to 1 after normalisation)

We normalise the intervals between ticks - based on a maximum of 20,000 milliseconds (20 seconds) (the main number of intervals). The interval (from 20,000) are equated to 20,000 milliseconds (those to 1 after normalization)

Import

B [1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
#from scipy.misc import toimage
%matplotlib inline
```

```
data = pd.read_csv('/Users/v135790/Documents/Python 3/May 2018/Tick/Dukastcopy/EURUSD_tick.csv')

print(data.head(n=10))
print(data[-5:])
```

```
                     Date      Bid      Ask  Vol_Bid  Vol_Ask
0  2018.01.01 22:00:08.661  1.20102  1.20143     1.50     0.75
1  2018.01.01 22:00:08.895  1.20102  1.20148     1.50     1.50
2  2018.01.01 22:00:10.634  1.20102  1.20147     1.50     0.75
3  2018.01.01 22:00:11.223  1.20102  1.20148     1.50     1.50
4  2018.01.01 22:00:29.530  1.20102  1.20145     1.50     0.75
5  2018.01.01 22:00:52.173  1.20102  1.20145     1.50     0.75
6  2018.01.01 22:00:52.224  1.20102  1.20148     1.50     1.50
7  2018.01.01 22:00:56.269  1.20097  1.20124     0.75     1.50
8  2018.01.01 22:00:56.836  1.20100  1.20115     1.50     1.50
9  2018.01.01 22:01:10.848  1.20099  1.20115     0.75     1.50
                        Date      Bid      Ask  Vol_Bid  Vol_Ask
8543871  2018.04.29 22:59:50.426  1.21214  1.21218     1.00     1.69
8543872  2018.04.29 22:59:51.566  1.21214  1.21217     1.00     1.00
8543873  2018.04.29 22:59:51.669  1.21212  1.21216     2.62     1.00
8543874  2018.04.29 22:59:52.267  1.21213  1.21215     1.00     1.00
8543875  2018.04.29 22:59:58.734  1.21213  1.21216     1.00     1.00
```

```
print(data.dtypes)
```

```
Date        object
Bid        float64
Ask        float64
Vol_Bid    float64
Vol_Ask    float64
dtype: object
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8543876 entries, 0 to 8543875
Data columns (total 5 columns):
Date       object
Bid        float64
Ask        float64
Vol_Bid    float64
Vol_Ask    float64
dtypes: float64(4), object(1)
memory usage: 325.9+ MB
```

MAXs & MINs

```
mxB  = data['Bid'].max()
mnB  = data['Bid'].min()
mxA  = data['Ask'].max()
mnA  = data['Ask'].min()
mxVB = data['Vol_Bid'].max()
mnVB = data['Vol_Bid'].min()
mxVA = data['Vol_Ask'].max()
mnVA = data['Vol_Ask'].min()

maxPrice = mxA
minPrice = mnB

if mxVB > mxVA:
    maxVolume = mxVB
else:
    maxVolume = mxVA

if mnVB < mnVA:
    minVolume = mnVB
else:
    minVolume = mnVA

print('Max Bid = ',mxB)
print('Min Bid = ',mnB)
print('Max Ask = ',mxA)
print('Min Ask = ',mnA)
print('Max Price = ',maxPrice)
print('Min Price = ',minPrice)

maxPriceUse = 1.26
minPriceUse = 1.12
minPriceUse1 = 1.19
print('Max Price Use = ',maxPriceUse)
print('Min Price Use = ',minPriceUse)
print('Min Price Use 1 = ',minPriceUse1)

print('Max Volume Bid = ',mxVB)
print('Min Volume Bid = ',mnVB)
print('Max Volume Ask = ',mxVA)
print('Min Volume Ask = ',mnVA)
print('Max Volume = ',maxVolume)
print('Min Volume = ',minVolume)

maxVolumeUse = 10
print('Max Volume Use = ',maxVolumeUse)
```

```
Max Bid =  1.25553
Min Bid =  1.19156
Max Ask =  1.25556
Min Ask =  1.19158
Max Price =  1.25556
Min Price =  1.19156
Max Price Use =  1.26
Min Price Use =  1.12
Min Price Use 1 =  1.19
Max Volume Bid =  5242.35
Min Volume Bid =  0.1
Max Volume Ask =  5362.95
Min Volume Ask =  0.1
Max Volume =  5362.95
Min Volume =  0.1
Max Volume Use =  10
```

```
pd.to_datetime(['31/07/2017', 'test'], errors='coerce')

data['DateT'] = pd.to_datetime(data['Date'])
print(data.dtypes)
print(data.head(n=5))
```

```
Date               object
Bid               float64
Ask               float64
Vol_Bid           float64
Vol_Ask           float64
DateT      datetime64[ns]
dtype: object
                     Date      Bid      Ask  Vol_Bid  Vol_Ask  \
0  2018.01.01 22:00:08.661  1.20102  1.20143      1.5     0.75
1  2018.01.01 22:00:08.895  1.20102  1.20148      1.5     1.50
2  2018.01.01 22:00:10.634  1.20102  1.20147      1.5     0.75
3  2018.01.01 22:00:11.223  1.20102  1.20148      1.5     1.50
4  2018.01.01 22:00:29.530  1.20102  1.20145      1.5     0.75

                   DateT
0 2018-01-01 22:00:08.661
1 2018-01-01 22:00:08.895
2 2018-01-01 22:00:10.634
3 2018-01-01 22:00:11.223
4 2018-01-01 22:00:29.530
```

```
DeltaNormal = maxPriceUse - minPriceUse
DeltaNormal_1 = maxPriceUse - minPriceUse1

data['Bid_N'] = (data['Bid'] - minPriceUse) / DeltaNormal
data['Ask_N'] = (data['Ask'] - minPriceUse) / DeltaNormal

data['Bid_N_use'] = (data['Bid'] - minPriceUse1) / DeltaNormal_1
data['Ask_N_use'] = (data['Ask'] - minPriceUse1) / DeltaNormal_1

print(data.dtypes)
print(data.head(n=5))
print(data[-5:])
```

```
Date                 object
Bid                 float64
Ask                 float64
Vol_Bid             float64
Vol_Ask             float64
DateT        datetime64[ns]
Bid_N               float64
Ask_N               float64
Bid_N_use           float64
Ask_N_use           float64
dtype: object
                      Date      Bid      Ask  Vol_Bid  Vol_Ask  \
0  2018.01.01 22:00:08.661  1.20102  1.20143      1.5     0.75
1  2018.01.01 22:00:08.895  1.20102  1.20148      1.5     1.50
2  2018.01.01 22:00:10.634  1.20102  1.20147      1.5     0.75
3  2018.01.01 22:00:11.223  1.20102  1.20148      1.5     1.50
4  2018.01.01 22:00:29.530  1.20102  1.20145      1.5     0.75

                   DateT     Bid_N     Ask_N  Bid_N_use  Ask_N_use
0 2018-01-01 22:00:08.661  0.578714  0.581643   0.157429   0.163286
1 2018-01-01 22:00:08.895  0.578714  0.582000   0.157429   0.164000
2 2018-01-01 22:00:10.634  0.578714  0.581929   0.157429   0.163857
3 2018-01-01 22:00:11.223  0.578714  0.582000   0.157429   0.164000
4 2018-01-01 22:00:29.530  0.578714  0.581786   0.157429   0.163571
                           Date      Bid      Ask  Vol_Bid  Vol_Ask  \
8543871  2018.04.29 22:59:50.426  1.21214  1.21218     1.00     1.69
8543872  2018.04.29 22:59:51.566  1.21214  1.21217     1.00     1.00
8543873  2018.04.29 22:59:51.669  1.21212  1.21216     2.62     1.00
8543874  2018.04.29 22:59:52.267  1.21213  1.21215     1.00     1.00
8543875  2018.04.29 22:59:58.734  1.21213  1.21216     1.00     1.00

                         DateT     Bid_N     Ask_N  Bid_N_use  Ask_N_use
8543871 2018-04-29 22:59:50.426  0.658143  0.658429   0.316286   0.316857
8543872 2018-04-29 22:59:51.566  0.658143  0.658357   0.316286   0.316714
8543873 2018-04-29 22:59:51.669  0.658000  0.658286   0.316000   0.316571
8543874 2018-04-29 22:59:52.267  0.658071  0.658214   0.316143   0.316429
8543875 2018-04-29 22:59:58.734  0.658071  0.658286   0.316143   0.316571
```

```
data['Vol_Bid_N'] = data['Vol_Bid'] / maxVolumeUse
data['Vol_Ask_N'] = data['Vol_Ask'] / maxVolumeUse

indexs_Big_Vol_Bid = data.index[data.Vol_Bid_N > 1]
indexs_Big_Vol_Ask = data.index[data.Vol_Ask_N > 1]
print('indexs_Big_Vol_Bid:  ', indexs_Big_Vol_Bid)
print('\n')
print('indexs_Big_Vol_Ask:  ', indexs_Big_Vol_Ask)
```

```
indexs_Big_Vol_Bid:   Int64Index([   5295,    5297,    5298,    5302,    5398,    5399,    5499,
              5528,    5586,    5589,
            ...
            8539716, 8539794, 8539873, 8539875, 8539876, 8539879, 8539921,
            8540082, 8540334, 8542834],
           dtype='int64', length=87662)


indexs_Big_Vol_Ask:   Int64Index([   3933,    3937,    4287,    4288,    4476,    4510,    4778,
              4800,    4863,    4888,
            ...
            8539406, 8539408, 8539495, 8539641, 8539917, 8539952, 8539960,
            8539961, 8540049, 8540436],
           dtype='int64', length=95757)
```

```
data[data.Date == 1]
```

Date	Bid	Ask	Vol_Bid	Vol_Ask	DateT	Bid_N	Ask_N	Bid_N_use	Ask_N_use	Vol_Bid_N	Vol_Ask_N

```
data['Vol_Ask_N'].loc[indexs_Big_Vol_Ask] = 1
data['Vol_Bid_N'].loc[indexs_Big_Vol_Bid] = 1
```

```
data[data.Date == 1]
```

Date	Bid	Ask	Vol_Bid	Vol_Ask	DateT	Bid_N	Ask_N	Bid_N_use	Ask_N_use	Vol_Bid_N	Vol_Ask_N

```
mmx  = data['Vol_Bid_N'].max()
mmn  = data['Vol_Bid_N'].min()
print(mmx, ' ', mmn)

mmx  = data['Vol_Ask_N'].max()
mmn  = data['Vol_Ask_N'].min()
print(mmx, ' ', mmn)
```

```
1.0   0.01
1.0   0.01
```

```
# time
import datetime

n = data.shape[0]

data['DateDelta'] = data['DateT'].astype(int)

data['DateDelta'] = data['DateDelta'].div(1000000 , axis=0)
data['DateDelta'] = data['DateDelta'].astype(int)
data['DateDelta1'] = data['DateDelta'].diff()

data['DateDelta1'].loc[0] = 0
data['DateDelta1'] = data['DateDelta1'].astype(int)

print(data.dtypes)
print(data.head(n=10))
print(data[-10:])
print(data['DateDelta1'].loc[:4])
```

```
Date                  object
Bid                  float64
Ask                  float64
Vol_Bid              float64
Vol_Ask              float64
DateT         datetime64[ns]
Bid_N                float64
Ask_N                float64
Bid_N_use            float64
Ask_N_use            float64
Vol_Bid_N            float64
Vol_Ask_N            float64
DateDelta              int64
DateDelta1             int64
dtype: object
                     Date      Bid      Ask  Vol_Bid  Vol_Ask  \
0  2018.01.01 22:00:08.661  1.20102  1.20143     1.50     0.75
1  2018.01.01 22:00:08.895  1.20102  1.20148     1.50     1.50
2  2018.01.01 22:00:10.634  1.20102  1.20147     1.50     0.75
3  2018.01.01 22:00:11.223  1.20102  1.20148     1.50     1.50
4  2018.01.01 22:00:29.530  1.20102  1.20145     1.50     0.75
5  2018.01.01 22:00:52.173  1.20102  1.20145     1.50     0.75
6  2018.01.01 22:00:52.224  1.20102  1.20148     1.50     1.50
7  2018.01.01 22:00:56.269  1.20097  1.20124     0.75     1.50
8  2018.01.01 22:00:56.836  1.20100  1.20115     1.50     1.50
9  2018.01.01 22:01:10.848  1.20099  1.20115     0.75     1.50

                   DateT     Bid_N     Ask_N  Bid_N_use  Ask_N_use  \
0 2018-01-01 22:00:08.661  0.578714  0.581643   0.157429   0.163286
1 2018-01-01 22:00:08.895  0.578714  0.582000   0.157429   0.164000
2 2018-01-01 22:00:10.634  0.578714  0.581929   0.157429   0.163857
3 2018-01-01 22:00:11.223  0.578714  0.582000   0.157429   0.164000
4 2018-01-01 22:00:29.530  0.578714  0.581786   0.157429   0.163571
5 2018-01-01 22:00:52.173  0.578714  0.581786   0.157429   0.163571
6 2018-01-01 22:00:52.224  0.578714  0.582000   0.157429   0.164000
7 2018-01-01 22:00:56.269  0.578357  0.580286   0.156714   0.160571
8 2018-01-01 22:00:56.836  0.578571  0.579643   0.157143   0.159286
9 2018-01-01 22:01:10.848  0.578500  0.579643   0.157000   0.159286
```

```
   Vol_Bid_N  Vol_Ask_N      DateDelta  DateDelta1
0      0.150      0.075  1514844008661           0
1      0.150      0.150  1514844008895         234
2      0.150      0.075  1514844010633        1738
3      0.150      0.150  1514844011223         590
4      0.150      0.075  1514844029529       18306
5      0.150      0.075  1514844052173       22644
6      0.150      0.150  1514844052224          51
7      0.075      0.150  1514844056269        4045
8      0.150      0.150  1514844056836         567
9      0.075      0.150  1514844070848       14012
                          Date      Bid      Ask  Vol_Bid  Vol_Ask  \
8543866  2018.04.29 22:59:36.550  1.21216  1.21219     1.00     1.19
8543867  2018.04.29 22:59:37.232  1.21215  1.21218     1.50     1.19
8543868  2018.04.29 22:59:41.742  1.21214  1.21218     1.27     1.19
8543869  2018.04.29 22:59:42.289  1.21214  1.21219     1.12     4.12
8543870  2018.04.29 22:59:46.156  1.21214  1.21218     1.00     1.87
8543871  2018.04.29 22:59:50.426  1.21214  1.21218     1.00     1.69
8543872  2018.04.29 22:59:51.566  1.21214  1.21217     1.00     1.00
8543873  2018.04.29 22:59:51.669  1.21212  1.21216     2.62     1.00
8543874  2018.04.29 22:59:52.267  1.21213  1.21215     1.00     1.00
8543875  2018.04.29 22:59:58.734  1.21213  1.21216     1.00     1.00

                         DateT     Bid_N     Ask_N  Bid_N_use  Ask_N_use  \
8543866 2018-04-29 22:59:36.550  0.658286  0.658500   0.316571   0.317000
8543867 2018-04-29 22:59:37.232  0.658214  0.658429   0.316429   0.316857
8543868 2018-04-29 22:59:41.742  0.658143  0.658429   0.316286   0.316857
8543869 2018-04-29 22:59:42.289  0.658143  0.658500   0.316286   0.317000
8543870 2018-04-29 22:59:46.156  0.658143  0.658429   0.316286   0.316857
8543871 2018-04-29 22:59:50.426  0.658143  0.658429   0.316286   0.316857
8543872 2018-04-29 22:59:51.566  0.658143  0.658357   0.316286   0.316714
8543873 2018-04-29 22:59:51.669  0.658000  0.658286   0.316000   0.316571
8543874 2018-04-29 22:59:52.267  0.658071  0.658214   0.316143   0.316429
8543875 2018-04-29 22:59:58.734  0.658071  0.658286   0.316143   0.316571

         Vol_Bid_N  Vol_Ask_N      DateDelta  DateDelta1
8543866      0.100      0.119  1525042776550         104
8543867      0.150      0.119  1525042777232         682
8543868      0.127      0.119  1525042781742        4510
8543869      0.112      0.412  1525042782289         547
8543870      0.100      0.187  1525042786156        3867
8543871      0.100      0.169  1525042790425        4269
8543872      0.100      0.100  1525042791566        1141
8543873      0.262      0.100  1525042791669         103
8543874      0.100      0.100  1525042792267         598
8543875      0.100      0.100  1525042798734        6467
0        0
1      234
2     1738
3      590
4    18306
Name: DateDelta1, dtype: int64
```

```
mmx  = data['DateDelta1'].max()
mmn  = data['DateDelta1'].min()
print(mmx, ' ', mmn)

mmx  = data['DateDelta'].max()
mmn  = data['DateDelta'].min()
print(mmx, ' ', mmn)
```

```
172931549   0
1525042798734   1514844008661
```

```
data[data.DateDelta1 > 100000]
```

	Date	Bid	Ask	Vol_Bid	Vol_Ask	DateT	Bid_N	Ask_N	Bid_N_use	Ask_N_use	Vol_Bid_N	Vol_Ask_N	DateDelta	D
93764	2018.01.03 03:44:03.184	1.20463	1.20467	4.69	2.25	2018-01-03 03:44:03.184	0.604500	0.604786	0.209000	0.209571	0.469	0.225	1514951043184	
322405	2018.01.07 22:00:04.850	1.20265	1.20297	0.75	0.75	2018-01-07 22:00:04.850	0.590357	0.592643	0.180714	0.185286	0.075	0.075	1515362404849	1
781883	2018.01.14 22:00:04.641	1.21933	1.21962	0.75	0.75	2018-01-14 22:00:04.641	0.709500	0.711571	0.419000	0.423143	0.075	0.075	1515967204641	1
1318286	2018.01.21 22:00:14.074	1.22702	1.22708	0.37	0.37	2018-01-21 22:00:14.074	0.764429	0.764857	0.528857	0.529714	0.037	0.037	1516572014073	1
1932893	2018.01.28 22:00:34.413	1.24232	1.24291	0.75	0.75	2018-01-28 22:00:34.413	0.873714	0.877929	0.747429	0.755857	0.075	0.075	1517176834413	1
2560927	2018.02.04 22:00:04.598	1.24320	1.24391	0.75	0.75	2018-02-04 22:00:04.598	0.880000	0.885071	0.760000	0.770143	0.075	0.075	1517781604598	1
3262667	2018.02.11 22:00:14.329	1.22483	1.22501	0.37	0.75	2018-02-11 22:00:14.329	0.748786	0.750071	0.497571	0.500143	0.037	0.075	1518386414329	1
3818770	2018.02.18 22:00:14.462	1.24015	1.24035	0.37	0.37	2018-02-18 22:00:14.462	0.858214	0.859643	0.716429	0.719286	0.037	0.037	1518991214462	1
4316953	2018.02.25 22:00:04.559	1.22928	1.22953	0.75	0.75	2018-02-25 22:00:04.559	0.780571	0.782357	0.561143	0.564714	0.075	0.075	1519596004559	1
4840754	2018.03.04 22:00:08.704	1.23248	1.23296	0.37	0.37	2018-03-04 22:00:08.704	0.803429	0.806857	0.606857	0.613714	0.037	0.037	1520200808704	1
5371812	2018.03.11 21:00:07.329	1.23094	1.23176	0.37	0.37	2018-03-11 21:00:07.329	0.792429	0.798286	0.584857	0.596571	0.037	0.037	1520802007329	1
5804131	2018.03.18 21:02:03.226	1.22817	1.22857	1.12	1.12	2018-03-18 21:02:03.226	0.772643	0.775500	0.545286	0.551000	0.112	0.112	1521406923225	1
6298964	2018.03.25 21:00:10.831	1.23470	1.23551	0.37	0.37	2018-03-25 21:00:10.831	0.819286	0.825071	0.638571	0.650143	0.037	0.037	1522011610831	1
6738174	2018.04.01 21:00:18.456	1.23199	1.23267	0.75	0.75	2018-04-01 21:00:18.456	0.799929	0.804786	0.599857	0.609571	0.075	0.075	1522616418456	1
6738589	2018.04.01 21:55:26.956	1.23229	1.23269	1.00	1.50	2018-04-01 21:55:26.956	0.802071	0.804929	0.604143	0.609857	0.100	0.150	1522619726956	
6902514	2018.04.04 04:43:06.917	1.22822	1.22825	2.36	1.00	2018-04-04 04:43:06.917	0.773000	0.773214	0.546000	0.546429	0.236	0.100	1522816986917	
7169706	2018.04.08 21:00:05.291	1.22824	1.22896	0.75	0.75	2018-04-08 21:00:05.291	0.773143	0.778286	0.546286	0.556571	0.075	0.075	1523221205291	1
7265381	2018.04.09 22:38:45.987	1.23235	1.23241	3.94	2.89	2018-04-09 22:38:45.987	0.802500	0.802929	0.605000	0.605857	0.394	0.289	1523313525987	
7636948	2018.04.15 21:00:12.838	1.23278	1.23360	0.75	0.75	2018-04-15 21:00:12.838	0.805571	0.811429	0.611143	0.622857	0.075	0.075	1523826012838	1
8076797	2018.04.22 21:00:13.972	1.22699	1.22772	0.75	0.52	2018-04-22 21:00:13.972	0.764214	0.769429	0.528429	0.538857	0.075	0.052	1524430813972	1
8273969	2018.04.25 03:17:46.392	1.22262	1.22264	1.00	1.00	2018-04-25 03:17:46.392	0.733000	0.733143	0.466000	0.466286	0.100	0.100	1524626266392	
8541118	2018.04.29 21:00:04.454	1.21224	1.21250	0.75	0.75	2018-04-29 21:00:04.454	0.658857	0.660714	0.317714	0.321429	0.075	0.075	1525035604454	1

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8543876 entries, 0 to 8543875
Data columns (total 14 columns):
Date          object
Bid           float64
Ask           float64
Vol_Bid       float64
Vol_Ask       float64
DateT         datetime64[ns]
Bid_N         float64
Ask_N         float64
Bid_N_use     float64
Ask_N_use     float64
Vol_Bid_N     float64
Vol_Ask_N     float64
DateDelta     int64
DateDelta1    int64
dtypes: datetime64[ns](1), float64(10), int64(2), object(1)
memory usage: 912.6+ MB
```

```
data[data.Date == 1]
```

Date	Bid	Ask	Vol_Bid	Vol_Ask	DateT	Bid_N	Ask_N	Bid_N_use	Ask_N_use	Vol_Bid_N	Vol_Ask_N	DateDelta	DateDelta1	DateDelta1_N

```
data['DateDelta1_N'].loc[indexs_Big_DateDelta1] = 1
```

```
data[data.Date == 1]
```

Date	Bid	Ask	Vol_Bid	Vol_Ask	DateT	Bid_N	Ask_N	Bid_N_use	Ask_N_use	Vol_Bid_N	Vol_Ask_N	DateDelta	DateDelta1	DateDelta1_N

```
print(data.dtypes)
print(data.head(n=10))
print(data[-10:])
```

```
Date                    object
Bid                    float64
Ask                    float64
Vol_Bid                float64
Vol_Ask                float64
DateT           datetime64[ns]
Bid_N                  float64
Ask_N                  float64
Bid_N_use              float64
Ask_N_use              float64
Vol_Bid_N              float64
Vol_Ask_N              float64
DateDelta                int64
DateDelta1               int64
DateDelta1_N           float64
dtype: object
                      Date      Bid      Ask  Vol_Bid  Vol_Ask  \
0  2018.01.01 22:00:08.661  1.20102  1.20143     1.50     0.75
1  2018.01.01 22:00:08.895  1.20102  1.20148     1.50     1.50
2  2018.01.01 22:00:10.634  1.20102  1.20147     1.50     0.75
3  2018.01.01 22:00:11.223  1.20102  1.20148     1.50     1.50
4  2018.01.01 22:00:29.530  1.20102  1.20145     1.50     0.75
5  2018.01.01 22:00:52.173  1.20102  1.20145     1.50     0.75
6  2018.01.01 22:00:52.224  1.20102  1.20148     1.50     1.50
7  2018.01.01 22:00:56.269  1.20097  1.20124     0.75     1.50
8  2018.01.01 22:00:56.836  1.20100  1.20115     1.50     1.50
9  2018.01.01 22:01:10.848  1.20099  1.20115     0.75     1.50
```

```
                   DateT     Bid_N     Ask_N  Bid_N_use  Ask_N_use  \
0 2018-01-01 22:00:08.661  0.578714  0.581643   0.157429   0.163286
1 2018-01-01 22:00:08.895  0.578714  0.582000   0.157429   0.164000
2 2018-01-01 22:00:10.634  0.578714  0.581929   0.157429   0.163857
3 2018-01-01 22:00:11.223  0.578714  0.582000   0.157429   0.164000
4 2018-01-01 22:00:29.530  0.578714  0.581786   0.157429   0.163571
5 2018-01-01 22:00:52.173  0.578714  0.581786   0.157429   0.163571
6 2018-01-01 22:00:52.224  0.578714  0.582000   0.157429   0.164000
7 2018-01-01 22:00:56.269  0.578357  0.580286   0.156714   0.160571
8 2018-01-01 22:00:56.836  0.578571  0.579643   0.157143   0.159286
9 2018-01-01 22:01:10.848  0.578500  0.579643   0.157000   0.159286

   Vol_Bid_N  Vol_Ask_N      DateDelta  DateDelta1  DateDelta1_N
0      0.150      0.075  1514844008661           0       0.00000
1      0.150      0.150  1514844008895         234       0.01170
2      0.150      0.075  1514844010633        1738       0.08690
3      0.150      0.150  1514844011223         590       0.02950
4      0.150      0.075  1514844029529       18306       0.91530
5      0.150      0.075  1514844052173       22644       1.00000
6      0.150      0.150  1514844052224          51       0.00255
7      0.075      0.150  1514844056269        4045       0.20225
8      0.150      0.150  1514844056836         567       0.02835
9      0.075      0.150  1514844070848       14012       0.70060
```

```
                        Date      Bid      Ask  Vol_Bid  Vol_Ask  \
8543866  2018.04.29 22:59:36.550  1.21216  1.21219     1.00     1.19
8543867  2018.04.29 22:59:37.232  1.21215  1.21218     1.50     1.19
8543868  2018.04.29 22:59:41.742  1.21214  1.21218     1.27     1.19
8543869  2018.04.29 22:59:42.289  1.21214  1.21219     1.12     4.12
8543870  2018.04.29 22:59:46.156  1.21214  1.21218     1.00     1.87
8543871  2018.04.29 22:59:50.426  1.21214  1.21218     1.00     1.69
8543872  2018.04.29 22:59:51.566  1.21214  1.21217     1.00     1.00
8543873  2018.04.29 22:59:51.669  1.21212  1.21216     2.62     1.00
8543874  2018.04.29 22:59:52.267  1.21213  1.21215     1.00     1.00
8543875  2018.04.29 22:59:58.734  1.21213  1.21216     1.00     1.00

                       DateT     Bid_N     Ask_N  Bid_N_use  Ask_N_use  \
8543866 2018-04-29 22:59:36.550  0.658286  0.658500   0.316571   0.317000
8543867 2018-04-29 22:59:37.232  0.658214  0.658429   0.316429   0.316857
8543868 2018-04-29 22:59:41.742  0.658143  0.658429   0.316286   0.316857
8543869 2018-04-29 22:59:42.289  0.658143  0.658500   0.316286   0.317000
8543870 2018-04-29 22:59:46.156  0.658143  0.658429   0.316286   0.316857
8543871 2018-04-29 22:59:50.426  0.658143  0.658429   0.316286   0.316857
8543872 2018-04-29 22:59:51.566  0.658143  0.658357   0.316286   0.316714
8543873 2018-04-29 22:59:51.669  0.658000  0.658286   0.316000   0.316571
8543874 2018-04-29 22:59:52.267  0.658071  0.658214   0.316143   0.316429
8543875 2018-04-29 22:59:58.734  0.658071  0.658286   0.316143   0.316571

         Vol_Bid_N  Vol_Ask_N      DateDelta  DateDelta1  DateDelta1_N
8543866      0.100      0.119  1525042776550         104       0.00520
8543867      0.150      0.119  1525042777232         682       0.03410
8543868      0.127      0.119  1525042781742        4510       0.22550
8543869      0.112      0.412  1525042782289         547       0.02735
8543870      0.100      0.187  1525042786156        3867       0.19335
8543871      0.100      0.169  1525042790425        4269       0.21345
8543872      0.100      0.100  1525042791566        1141       0.05705
8543873      0.262      0.100  1525042791669         103       0.00515
8543874      0.100      0.100  1525042792267         598       0.02990
8543875      0.100      0.100  1525042798734        6467       0.32335
```

```
mmx  = data['DateDelta1_N'].max()
mmn  = data['DateDelta1_N'].min()
print(mmx, ' ', mmn)
```

```
1.0   0.0
```

```
print(data.head(n=0))
```

```
Empty DataFrame
Columns: [Date, Bid, Ask, Vol_Bid, Vol_Ask, DateT, Bid_N, Ask_N, Bid_N_use, Ask_N_use, Vol_Bid_N, Vol_Ask_N, DateDelta, DateDel
ta1, DateDelta1_N]
Index: []
```

```
data = data[['Vol_Ask_N','Ask_N_use','DateDelta1_N','Bid_N_use','Vol_Bid_N',
             'Date','Bid','Ask','Vol_Bid','Vol_Ask','DateT','Bid_N','Ask_N','DateDelta','DateDelta1']]
print(data.head(n=2))
print(data[-2:])
```

```
   Vol_Ask_N  Ask_N_use  DateDelta1_N  Bid_N_use  Vol_Bid_N  \
0      0.075   0.163286        0.0000   0.157429       0.15
1      0.150   0.164000        0.0117   0.157429       0.15

                     Date      Bid      Ask  Vol_Bid  Vol_Ask  \
0  2018.01.01 22:00:08.661  1.20102  1.20143      1.5     0.75
1  2018.01.01 22:00:08.895  1.20102  1.20148      1.5     1.50

                   DateT     Bid_N     Ask_N      DateDelta  DateDelta1
0 2018-01-01 22:00:08.661  0.578714  0.581643  1514844008661           0
1 2018-01-01 22:00:08.895  0.578714  0.582000  1514844008895         234
         Vol_Ask_N  Ask_N_use  DateDelta1_N  Bid_N_use  Vol_Bid_N  \
8543874        0.1   0.316429       0.02990   0.316143        0.1
8543875        0.1   0.316571       0.32335   0.316143        0.1

                          Date      Bid      Ask  Vol_Bid  Vol_Ask  \
8543874  2018.04.29 22:59:52.267  1.21213  1.21215      1.0      1.0
8543875  2018.04.29 22:59:58.734  1.21213  1.21216      1.0      1.0

                         DateT     Bid_N     Ask_N      DateDelta  DateDelta1
8543874 2018-04-29 22:59:52.267  0.658071  0.658214  1525042792267         598
8543875 2018-04-29 22:59:58.734  0.658071  0.658286  1525042798734        6467
```

```
mmx  = data['Vol_Ask_N'].max()
mmn  = data['Vol_Ask_N'].min()
print(mmx,' ', mmn)
mmx  = data['Ask_N_use'].max()
mmn  = data['Ask_N_use'].min()
print(mmx,' ', mmn)
mmx  = data['DateDelta1_N'].max()
mmn  = data['DateDelta1_N'].min()
print(mmx,' ', mmn)
mmx  = data['Bid_N_use'].max()
mmn  = data['Bid_N_use'].min()
print(mmx,' ', mmn)
mmx  = data['Vol_Bid_N'].max()
mmn  = data['Vol_Bid_N'].min()
print(mmx,' ', mmn)
mmx  = data['Bid_N'].max()
mmn  = data['Bid_N'].min()
print(mmx,' ', mmn)
mmx  = data['Ask_N'].max()
mmn  = data['Ask_N'].min()
print(mmx,' ', mmn)
```

```
1.0   0.01
0.936571428571   0.0225714285714
1.0   0.0
0.936142857143   0.0222857142857
1.0   0.01
0.968071428571   0.511142857143
0.968285714286   0.511285714286
```

```
data_4_mth=data.drop(['Date','Bid','Ask','Vol_Bid','Vol_Ask','DateT','Bid_N','Ask_N','DateDelta','DateDelta1'],axis=1)
data_4_mth_All=data.drop(['Date','Bid','Ask','Vol_Bid','Vol_Ask','DateT','Bid_N_use','Ask_N_use','DateDelta','DateDelta1'],axis=1

data_4_mth_All = data_4_mth_All[['Vol_Ask_N','Ask_N','DateDelta1_N','Bid_N','Vol_Bid_N']]

print(data_4_mth.head(n=2))
print(data_4_mth[-2:])
print('\n')
print(data_4_mth_All.head(n=2))
print(data_4_mth_All[-2:])
```

```
   Vol_Ask_N  Ask_N_use  DateDelta1_N  Bid_N_use  Vol_Bid_N
0      0.075   0.163286        0.0000   0.157429       0.15
1      0.150   0.164000        0.0117   0.157429       0.15
         Vol_Ask_N  Ask_N_use  DateDelta1_N  Bid_N_use  Vol_Bid_N
8543874        0.1   0.316429       0.02990   0.316143        0.1
8543875        0.1   0.316571       0.32335   0.316143        0.1


   Vol_Ask_N     Ask_N  DateDelta1_N     Bid_N  Vol_Bid_N
0      0.075  0.581643        0.0000  0.578714       0.15
1      0.150  0.582000        0.0117  0.578714       0.15
         Vol_Ask_N     Ask_N  DateDelta1_N     Bid_N  Vol_Bid_N
8543874        0.1  0.658214       0.02990  0.658071        0.1
8543875        0.1  0.658286       0.32335  0.658071        0.1
```

```
file_obj1 = open('/Users/v135790/Documents/Python 3/December 2018/Tick/EURUSD_tick_4mth.csv', 'w')
data_4_mth.to_csv('/Users/v135790/Documents/Python 3/December 2018/Tick/EURUSD_tick_4mth.csv',
                  encoding='utf-8', index=False)
file_obj1.close()

file_obj2 = open('/Users/v135790/Documents/Python 3/December 2018/Tick/EURUSD_tick_4mth_All.csv', 'w')
data_4_mth_All.to_csv('/Users/v135790/Documents/Python 3/December 2018/Tick/EURUSD_tick_4mth_All.csv',
                  encoding='utf-8', index=False)
file_obj2.close()
```

```
n_start = 0
n_small = 1100000  # 2 weeks
print('n_start = ',n_start, '\n',
      'n_small = ', n_small)
```

```
n_start =  0
 n_small =  1100000
```

```
data_4_mth_1week =  data_4_mth[n_start:n_small]
data_4_mth_All_1week = data_4_mth_All[n_start:n_small]

print(data_4_mth.iloc[[n_start,n_start+1],])
print(data_4_mth.iloc[[n_small-2,n_small-1],])
print('\n')
print(data_4_mth_1week.head(n=2))
print(data_4_mth_1week[-2:])
print('\n')
print(data_4_mth_All.iloc[[n_start,n_start+1],])
print(data_4_mth_All.iloc[[n_small-2,n_small-1],])
print('\n')
print(data_4_mth_All_1week.head(n=2))
print(data_4_mth_All_1week[-2:])
```

```
   Vol_Ask_N  Ask_N_use  DateDelta1_N  Bid_N_use  Vol_Bid_N
0      0.075   0.163286        0.0000   0.157429       0.15
1      0.150   0.164000        0.0117   0.157429       0.15
         Vol_Ask_N  Ask_N_use  DateDelta1_N  Bid_N_use  Vol_Bid_N
1099998      0.150   0.453714       0.00285   0.453286      0.107
1099999      0.244   0.454143       0.01315   0.453571      0.307
```


```
   Vol_Ask_N  Ask_N_use  DateDelta1_N  Bid_N_use  Vol_Bid_N
0      0.075   0.163286        0.0000   0.157429       0.15
1      0.150   0.164000        0.0117   0.157429       0.15
         Vol_Ask_N  Ask_N_use  DateDelta1_N  Bid_N_use  Vol_Bid_N
1099998      0.150   0.453714       0.00285   0.453286      0.107
1099999      0.244   0.454143       0.01315   0.453571      0.307
```


```
   Vol_Ask_N     Ask_N  DateDelta1_N     Bid_N  Vol_Bid_N
0      0.075  0.581643        0.0000  0.578714       0.15
1      0.150  0.582000        0.0117  0.578714       0.15
         Vol_Ask_N     Ask_N  DateDelta1_N     Bid_N  Vol_Bid_N
1099998      0.150  0.726857       0.00285  0.726643      0.107
1099999      0.244  0.727071       0.01315  0.726786      0.307
```

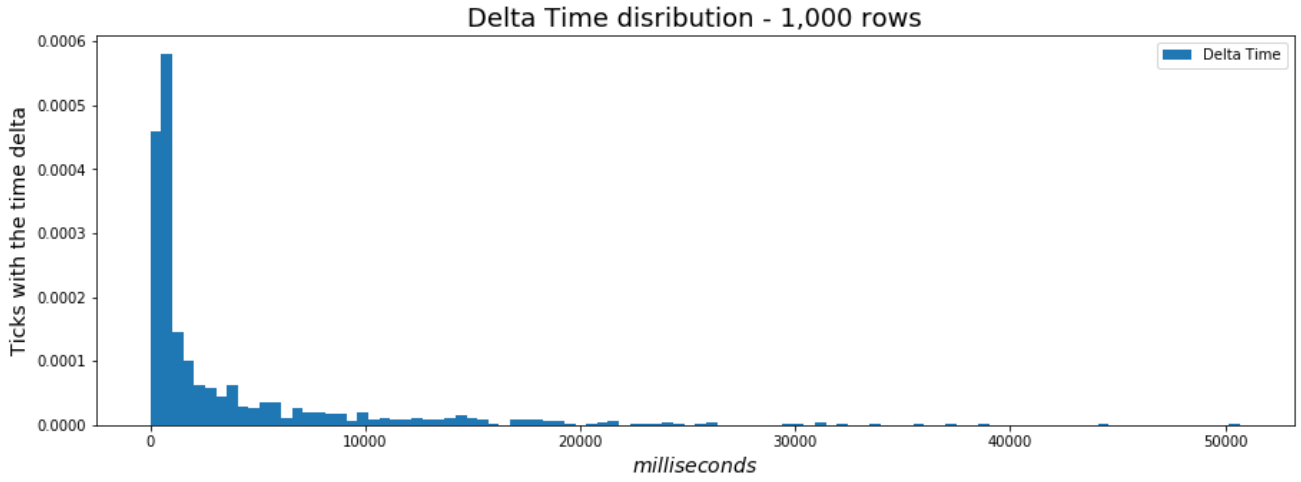

```
   Vol_Ask_N     Ask_N  DateDelta1_N     Bid_N  Vol_Bid_N
0      0.075  0.581643        0.0000  0.578714       0.15
1      0.150  0.582000        0.0117  0.578714       0.15
         Vol_Ask_N     Ask_N  DateDelta1_N     Bid_N  Vol_Bid_N
1099998      0.150  0.726857       0.00285  0.726643      0.107
1099999      0.244  0.727071       0.01315  0.726786      0.307
```


```
file_obj3 = open('/Users/v135790/Documents/Python 3/December 2018/Tick/EURUSD_tick_4mth_1week.csv', 'w')
data_4_mth_1week.to_csv('/Users/v135790/Documents/Python 3/December 2018/Tick/EURUSD_tick_4mth_1week.csv',
                        encoding='utf-8', index=False)
file_obj3.close()

file_obj4 = open('/Users/v135790/Documents/Python 3/December 2018/Tick/EURUSD_tick_4mth_All_1week.csv', 'w')
data_4_mth_All_1week.to_csv('/Users/v135790/Documents/Python 3/December 2018/Tick/EURUSD_tick_4mth_All_1week.csv',
                            encoding='utf-8', index=False)
file_obj4.close()
```

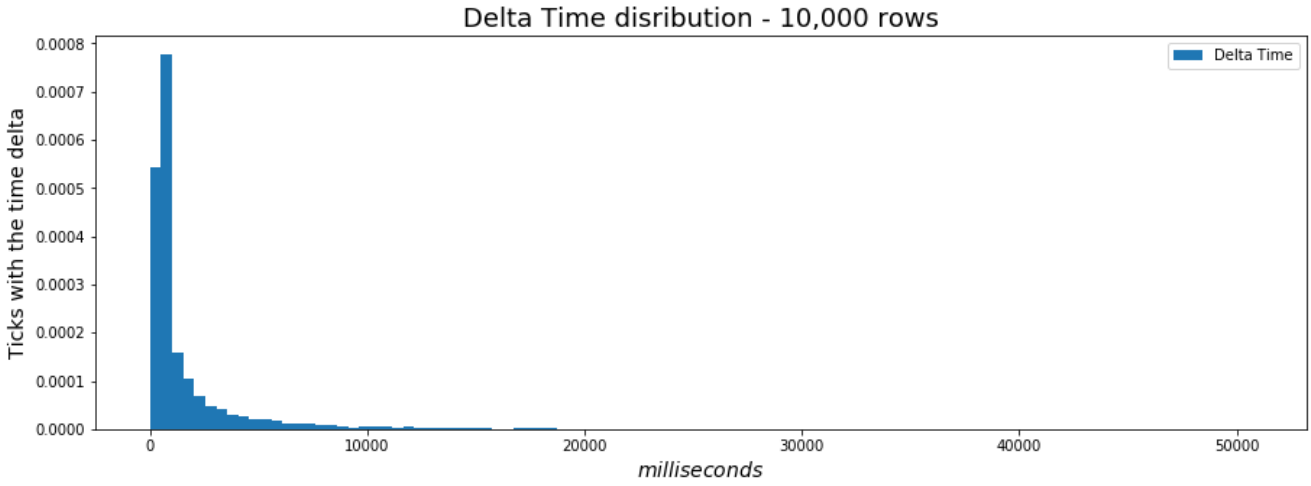
```
# delta ticks
data_Delta_Time  = data['DateDelta1']

print('data_Delta_Time', data_Delta_Time.dtypes)
print(data_Delta_Time[:5])
print(data_Delta_Time[-5:])
```

```
data_Delta_Time int64
0        0
1      234
2     1738
3      590
4    18306
Name: DateDelta1, dtype: int64
8543871    4269
8543872    1141
8543873     103
8543874     598
8543875    6467
Name: DateDelta1, dtype: int64
```

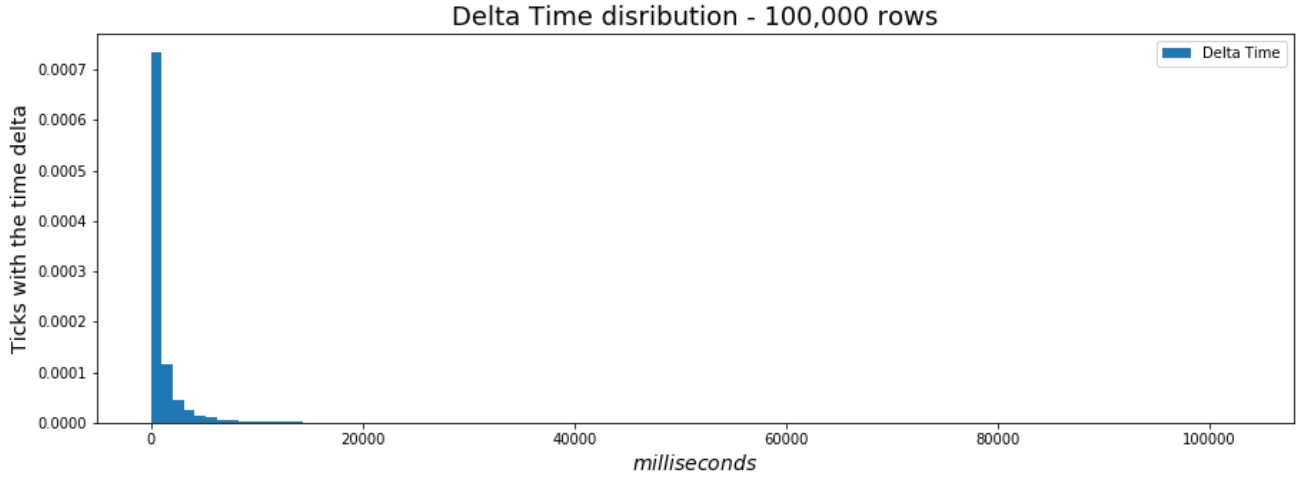
```
# first 1.000 rows

indexes_to_drop = np.array(range(1000,8543876))
indexes_to_keep = set(range(data_Delta_Time.shape[0])) - set(indexes_to_drop)

data_Delta_Time_Test = data_Delta_Time.take(list(indexes_to_keep))

data_Delta_Time_Test_1 = np.array(data_Delta_Time_Test)
data_Delta_Time_Test_1 = data_Delta_Time_Test_1.reshape(1000)

plt.figure(num=1, figsize=(15,5))
plt.hist(data_Delta_Time_Test_1, bins=100, label='Delta Time', normed=True)

plt.title('Delta Time disribution - 1,000 rows', size=18)
plt.ylabel('Ticks with the time delta', size=14)
plt.xlabel('$milliseconds$', size=14)
plt.legend(loc='upper right')
#  PNG
plt.savefig('/Users/v135790/Documents/Python 3/December 2018/Figures/TimeDisribution-1000_rows.png')
```



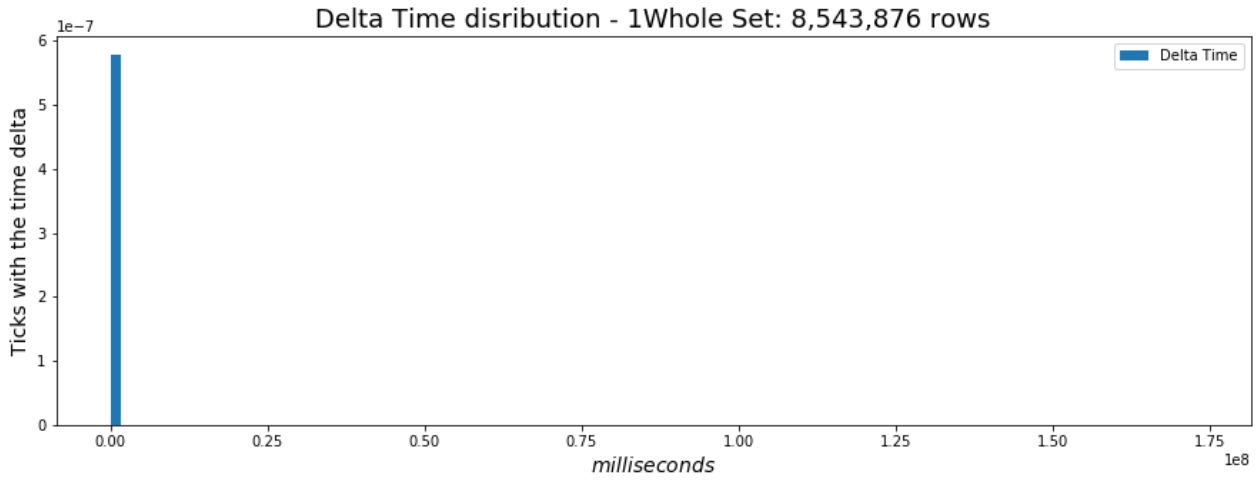

```
# first 10.000 rows

indexes_to_drop = np.array(range(10000,8543876))
indexes_to_keep = set(range(data_Delta_Time.shape[0])) - set(indexes_to_drop)
data_Delta_Time_Test = data_Delta_Time.take(list(indexes_to_keep))

data_Delta_Time_Test_1 = np.array(data_Delta_Time_Test)
data_Delta_Time_Test_1 = data_Delta_Time_Test_1.reshape(10000)

plt.figure(num=1, figsize=(15,5))
plt.hist(data_Delta_Time_Test_1, bins=100, label='Delta Time', normed=True)

plt.title('Delta Time disribution - 10,000 rows', size=18)
plt.ylabel('Ticks with the time delta', size=14)
plt.xlabel('$milliseconds$', size=14)
plt.legend(loc='upper right')
#  PNG
plt.savefig('/Users/v135790/Documents/Python 3/December 2018/Figures/TimeDisribution-10000_rows.png')
```




```
# first 100.000 rows

indexes_to_drop = np.array(range(100000,8543876))
indexes_to_keep = set(range(data_Delta_Time.shape[0])) - set(indexes_to_drop)

data_Delta_Time_Test = data_Delta_Time.take(list(indexes_to_keep))

data_Delta_Time_Test_1 = np.array(data_Delta_Time_Test)
data_Delta_Time_Test_1 = data_Delta_Time_Test_1.reshape(100000)

plt.figure(num=1, figsize=(15,5))
plt.hist(data_Delta_Time_Test_1, bins=100, label='Delta Time', normed=True)

plt.title('Delta Time disribution - 100,000 rows', size=18)
plt.ylabel('Ticks with the time delta', size=14)
plt.xlabel('$milliseconds$', size=14)
plt.legend(loc='upper right')
#  PNG
plt.savefig('/Users/v135790/Documents/Python 3/December 2018/Figures/TimeDisribution-100000_rows.png')
```



```
# full SET  8.543.876 rows

data_Delta_Time_Test_1 = np.array(data_Delta_Time)
data_Delta_Time_Test_1 = data_Delta_Time_Test_1.reshape(8543876)

plt.figure(num=1, figsize=(15,5))
plt.hist(data_Delta_Time_Test_1, bins=100, label='Delta Time', normed=True)

plt.title('Delta Time disribution - 1Whole Set: 8,543,876 rows', size=18)
plt.ylabel('Ticks with the time delta', size=14)
plt.xlabel('$milliseconds$', size=14)
plt.legend(loc='upper right')
#  PNG
plt.savefig('/Users/v135790/Documents/Python 3/December 2018/Figures/TimeDisribution.png')
```




```
data[data.DateDelta1 > 20000].count()
```

```
Date          7374
Bid           7374
Ask           7374
Vol_Bid       7374
Vol_Ask       7374
DateT         7374
Bid_N         7374
Ask_N         7374
Bid_N_use     7374
Ask_N_use     7374
Vol_Bid_N     7374
Vol_Ask_N     7374
DateDelta     7374
DateDelta1    7374
dtype: int64
```

```
data_ = pd.read_csv('/Users/v135790/Documents/Python 3/January 2019/Tick/EURUSD_tick_4mth_4week_200_33th-39th.csv')

print(data_.head(n=5))
print(data_[-5:])
```

```
data_.info()
```

```
data = pd.read_csv('/Users/v135790/Documents/Python 3/January 2019/Tick/EURUSD_tick_4mth_4week_200_.csv')

print(data.head(n=5))
print(data[-5:])
```

```
data.info()
```

```
columns2 = ['Date','Bid','Ask','Vol_Bid','Vol_Ask','DateT','Bid_N','Ask_N','Bid_N_use','Ask_N_use',
           'Vol_Bid_N','Vol_Ask_N','DateDelta','DateDelta1','DateDelta1_N']
data_.drop(columns2, inplace=True, axis=1)
data_.info()
```

```
columns3 = ['Date','Bid','Ask','Vol_Bid','Vol_Ask','DateT',
           'DateDelta','DateDelta1','DateDelta1_N']
data.drop(columns3, inplace=True, axis=1)
data.info()
```

```
data_both = data_.append(data, ignore_index=True)

print(data_both.info())
print(data_both.head(n=600006))
print(data_both[-5:])
```

```
data_both[data_both.Min_200_3 <= 0]
```

```
data_both[data_both.Max_200_3 <= 0]
```

```
data_both[data_both.Delta_200_3 <= 0]
```

```
data_both['Min_200_3'].loc[0:200] = 1.22681
data_both['Max_200_3'].loc[0:200] = 1.22726
data_both['Delta_200_3'].loc[0:200] = data_both['Max_200_3'].loc[0:200] - data_both['Min_200_3'].loc[0:200]
```

```
file_obj5 = open('/Users/v135790/Documents/Python 3/January 2019/Tick/EURUSD_tick_real-4.csv', 'w')
data_both.to_csv('/Users/v135790/Documents/Python 3/January 2019/Tick/EURUSD_tick_real-4.csv',
                 encoding='utf-8', index=False)
file_obj5.close()
```

```
# 4 weeks
data_ = pd.read_csv('/Users/v135790/Documents/Python 3/January 2019/Tick/EURUSD_tick_4mth_4week_200_33th-39th.csv')

print(data_.head(n=5))
print(data_[-5:])
```

```
data = pd.read_csv('/Users/v135790/Documents/Python 3/January 2019/Tick/EURUSD_tick_4mth_4week_200_.csv')
```

```
data_OK1=data_.drop(['Date','Vol_Bid','Vol_Ask','DateT','Bid_N','Ask_N',
                     'Bid_N_use','Ask_N_use','DateDelta','DateDelta1','Min_200_3','Max_200_3','Delta_200_3'],axis=1)

print(data_OK1.info())
print(data_OK1.head(n=5))
print(data_OK1[-5:])
```

```
data_OK2=data.drop(['Date','Vol_Bid','Vol_Ask','DateT',
                     'DateDelta','DateDelta1','Min_200_3','Max_200_3','Delta_200_3'],axis=1)

print(data_OK2.info())
print(data_OK2.head(n=5))
print(data_OK2[-5:])
```

```
data_OK = data_OK1.append(data_OK2, ignore_index=True)

print(data_OK.info())
print(data_OK.head(n=600006))
print(data_OK[-5:])
```

```
data_OK['Bid_N_200_3'] = (data_OK['Bid'] - data_both['Min_200_3']) / data_both['Delta_200_3']
data_OK['Ask_N_200_3'] = (data_OK['Ask'] - data_both['Min_200_3']) / data_both['Delta_200_3']

print(data_OK.info())
print(data_OK.head(n=5))
print(data_OK[-5:])
```

```
print(data_OK['Bid_N_200_3'].min(), '   ', data_OK['Bid_N_200_3'].max())
print(data_OK['Ask_N_200_3'].min(), '   ', data_OK['Ask_N_200_3'].max())
```

```
data_OK = data_OK.drop(['Ask','Bid'],axis=1)

print(data_OK.info())
print(data_OK.head(n=5))
print(data_OK[-5:])
```

```
data_OK = data_OK[['Vol_Ask_N','Ask_N_200_3','DateDelta1_N','Bid_N_200_3','Vol_Bid_N']]

print(data_OK.info())
print(data_OK.head(n=5))
print(data_OK[-5:])
```

```
file_obj3 = open('/Users/v135790/Documents/Python 3/January 2019/Tick/EURUSD_tick_OK-3.csv', 'w')
data_OK.to_csv('/Users/v135790/Documents/Python 3/January 2019/Tick/EURUSD_tick_OK-3.csv',
                  encoding='utf-8', index=False)
file_obj3.close()
```

Appendix X.

Convolutional Neural Network

Forex EURUSD Price Prediction Using Keras

Tick Data With Dukastcopy

Data for 2019 year

We use ONE-DIMENSIONAL convolutional Neural network

(The convolution kernel is one-dimensional = 9). Broken into pieces of 200 ticks, batch_size = 100, epochs = 15, validation_split=0.3 (30% for validation)

(Choosing the best version of training - a variable number of epochs - ModelCheckpoint Callback)

Price normalisation: subtract the minimum divide by the maximum price among 200 ticks that include this tick

Normalisation of volumes - based on a maximum of 10 lots (the main number of lots). The interval (from 10 to 5000 lots) are equated to 10 lots (those to 1 after normalisation)

We normalise the intervals between ticks - based on a maximum of 20,000 milliseconds (20 seconds) (the main number of intervals). The interval (from 20,000) are equated to 20,000 milliseconds (those to 1 after normalization)

The GOAL: According to the dynamics of TICK PRICES and VOLUME Forex EURUSD to predict the price of the next tick

We use a csv file with Dukastcopy and Keras, substantially REFINED and ALREADY normalised

B [2]:
```
# TensorFlow and tf.keras

import tensorflow as tf
from tensorflow import keras

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline
```

B [3]:
```
print(tf.__version__)
```

```
2.4.1
```

B [3]:
```
# COLABORATE
!cat /proc/driver/nvidia/gpus/0000:00:04.0/information
```

```
'cat' is not recognized as an internal or external command,
operable program or batch file.
```

B [4]:
```
# Set seed to repeat results for random number generator
# tk we use stochostic optimization methods
np.random.seed(42)
```

Data

B [ ]:
```
# COLABORATE
from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))
```

B [ ]:
```
from google.colab import drive
drive.mount('/content/grive', force_remount=True)
```

B [3]:
```
# COLABORATE
!ls
```

```
'ls' is not recognized as an internal or external command,
operable program or batch file.
```

B [3]:
```
data = pd.read_csv('/Users/vital/Python_Code/_PhD_Code/2019-01-January/Tick/EURUSD_tick_OK-2.csv')

print(data.head(n=5))
print(data[-5:])
```

```
   Vol_Ask_N  Ask_N_200_3  DateDelta1_N  Bid_N_200_3  Vol_Bid_N
0      0.176     0.636364       0.01655     0.515152      0.100
1      0.364     0.606061       0.02750     0.454545      0.420
2      0.100     0.575758       0.09770     0.454545      0.187
3      0.100     0.151515       0.01590     0.121212      0.100
4      0.270     0.212121       0.01040     0.060606      0.214
         Vol_Ask_N  Ask_N_200_3  DateDelta1_N  Bid_N_200_3  Vol_Bid_N
1099995      0.100     0.705357       0.01060     0.687500      0.195
1099996      0.100     0.758929       0.01630     0.741071      0.244
1099997      0.100     0.776786       0.00260     0.750000      0.382
1099998      0.461     0.794643       0.01045     0.758929      0.393
1099999      0.536     0.794643       0.00260     0.758929      0.664
```

B [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1100000 entries, 0 to 1099999
Data columns (total 5 columns):
 #   Column        Non-Null Count    Dtype
---  ------        --------------    -----
 0   Vol_Ask_N     1100000 non-null  float64
 1   Ask_N_200_3   1100000 non-null  float64
 2   DateDelta1_N  1100000 non-null  float64
 3   Bid_N_200_3   1100000 non-null  float64
 4   Vol_Bid_N     1100000 non-null  float64
dtypes: float64(5)
memory usage: 42.0 MB
```

B [5]:
```
# downsizing

data['Vol_Ask_N']    =pd.to_numeric(data['Vol_Ask_N'],downcast='float')
data['Ask_N_200_3']  =pd.to_numeric(data['Ask_N_200_3'],downcast='float')
data['DateDelta1_N'] =pd.to_numeric(data['DateDelta1_N'],downcast='float')
data['Bid_N_200_3']  =pd.to_numeric(data['Bid_N_200_3'],downcast='float')
data['Vol_Bid_N']    =pd.to_numeric(data['Vol_Bid_N'],downcast='float')

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1100000 entries, 0 to 1099999
Data columns (total 5 columns):
 #   Column        Non-Null Count    Dtype
---  ------        --------------    -----
 0   Vol_Ask_N     1100000 non-null  float32
 1   Ask_N_200_3   1100000 non-null  float32
 2   DateDelta1_N  1100000 non-null  float32
 3   Bid_N_200_3   1100000 non-null  float32
 4   Vol_Bid_N     1100000 non-null  float32
dtypes: float32(5)
memory usage: 21.0 MB
```

B [8]:
```
# n_small = 1100000 # about 4 weeks

# create a three-dimensional data tensor: [(n_small-N + 1), N, 5] and
# two-dimensional array of answers [(n_small-N + 1), N] with a tick price shifted one tick up,
# where N is the number of ticks for analysis at each step (those are looking backward
# then you need to delete the last batch, since there is no price for the subsequent tick for it, BUT we will immediately do
# data array of dimension [(n_small-N + 1-1), N, 5]
```

create numpy

B [6]:
```
n = data.shape[0]
p = data.shape[1]

n_small = 1100000
print('Tick number for treining: ', n_small, ' = size of the numpay')

data_a = np.array(data[:n_small])

print("data_a:")
print(data_a[:4])
print('\n')
print(data_a[-4:])

print('\n',"Tick number in the DataFrame:          n = ", n, '\n',
      "Tick number for training:  n_small= ", n_small, '\n',
      "Column number:                  p = ", p)
print(" numpy type:              ", type(data_a), '\n',

      "numpy size:              ", data_a.shape)
print('Example: data_a[1,0] = ', data_a[1,0])
```

```
Tick number for treining:  1100000  = size of the numpay
data_a:
[[0.176      0.6363636  0.01655    0.5151515  0.1       ]
 [0.364      0.6060606  0.0275     0.45454547 0.42      ]
 [0.1        0.57575756 0.0977     0.45454547 0.187     ]
 [0.1        0.15151516 0.0159     0.12121212 0.1       ]]


[[0.1        0.7589286  0.0163     0.7410714  0.244     ]
 [0.1        0.77678573 0.0026     0.75       0.382     ]
 [0.461      0.79464287 0.01045    0.7589286  0.393     ]
 [0.536      0.79464287 0.0026     0.7589286  0.664     ]]

 Tick number in the DataFrame:          n =  1100000
 Tick number for training:  n_small=  1100000
 Column number:                  p =  5
 numpy type:              <class 'numpy.ndarray'>
 numpy size:              (1100000, 5)
Example: data_a[1,0] =  0.364
```

target: predicting the Bid and Ask values of the next tick in FOREX EURUSD.

we form the response matrix Y with the Bid and Ask values of the next tick in time

B [7]:
```
N = 200   # N - the depth of analysis at each step of the training (how many ticks we look back) - about 10 minutes

# numpy size (L, N, 5), zde L = n_small - N
L = n_small - N
print('Size of a three-dimensional data array for training: ', L, N, 5)
```

```
Size of a three-dimensional data array for training:  1099800 200 5
```

B [8]:
```
data_b = np.zeros( (L, N, 5), dtype= 'float32' )

print('data_b:', '\n', data_b)

print(data_b[L-1, N-1, 0])
print(data_a[n_small-2, 0], '\n')

# create a two-dimensional vector of answers - normalised Ask and Bid
Y = np.zeros((L, 2), dtype= 'float32') # We predict two prices - normalised Ask and Bid

print('Y:', '\n', Y)
```

```
data_b:
 [[[0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  ...
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]]

 [[0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  ...
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]]

 [[0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  ...
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]]

 ...

 [[0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  ...
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]]

 [[0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  ...
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]]

 [[0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  ...
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]]]
0.0
0.461

Y:
 [[0. 0.]
 [0. 0.]
 [0. 0.]
 ...
 [0. 0.]
 [0. 0.]
 [0. 0.]]
```

B [9]:
```
# Progress bar
from tqdm import tqdm
```

B [10]:
```
# control the formation of arrays for training
for s in tqdm(range(N-2,N+5)):
    print(s, data_a[s,:])
print('\n')
for s in tqdm(range(n_small-N-5,n_small-N+1)):
    print(s, data_a[s,:])
```

```
100%|██████████| 7/7 [00:00<00:00, 4507.93it/s]
100%|██████████| 6/6 [00:00<00:00, 1833.44it/s]
```

```
198 [0.701      0.36363637 0.0635     0.21212122 0.214     ]
199 [0.101      0.36363637 0.00705    0.21212122 0.532     ]
200 [0.187      0.36363637 0.1094     0.24242425 0.195     ]
201 [0.157      0.33333334 0.0268     0.21212122 0.401     ]
202 [0.112      0.21212122 0.02875    0.15151516 0.375     ]
203 [0.307      0.24242425 0.03105    0.09090909 0.157     ]
204 [0.101      0.21212122 0.0282     0.09090909 0.169     ]


1099795 [0.195      0.17910448 0.09345    0.10447761 0.15      ]
1099796 [0.225      0.13432837 0.0026     0.08955224 0.115     ]
1099797 [0.1        0.10447761 0.0052     0.04477612 0.187     ]
1099798 [0.195      0.10447761 0.0402     0.04477612 0.1       ]
1099799 [0.195      0.08955224 0.0079     0.         0.316     ]
1099800 [0.195     0.0952381 0.02705   0.        0.1      ]
```

B [11]:
```
# fill the array for training data_b and the array of answers Y from the original data_a array

print('n_small - N = ', n_small-N)
for k in tqdm(range(n_small-N)):
    data_b[k, :, :] = data_a[k:k+N, :]

    Y[k,0] = data_a[k+N,3]   # normalised bid - value for one tick to the future
    Y[k,1] = data_a[k+N,1]   # normalised Ask - value for one tick to the future
print(k) # ast index value (for control)

print('data_b:', '\n', data_b)
print("Numpy size:        ", data_b.shape, '\n')

print( Y[:5,:])
print( Y[-10:,:])
print(      "Numpy size:         ", Y.shape)
```

```
  2%||          | 22387/1099800 [00:00<00:04, 222421.45it/s]

n_small - N =  1099800

100%|██████████| 1099800/1099800 [00:04<00:00, 251122.83it/s]

1099799
data_b:
 [[[0.176      0.6363636  0.01655    0.5151515  0.1       ]
  [0.364      0.6060606  0.0275     0.45454547 0.42      ]
  [0.1        0.57575756 0.0977     0.45454547 0.187     ]
  ...
  [0.225      0.33333334 0.03015    0.21212122 0.12      ]
  [0.701      0.36363637 0.0635     0.21212122 0.214     ]
  [0.101      0.36363637 0.00705    0.21212122 0.532     ]]

 [[0.364      0.6060606  0.0275     0.45454547 0.42      ]
  [0.1        0.57575756 0.0977     0.45454547 0.187     ]
  [0.1        0.15151516 0.0159     0.12121212 0.1       ]
  ...
  [0.701      0.36363637 0.0635     0.21212122 0.214     ]
```

B [12]:
```
# control of the correctness of filling the array Y (should show the normalized prices Bid and Ask one step ahead)
print('data_b:', '           Y:')
print('Bid       Ask       Bid       Ask')
pp = 789564 # (any number before 1099799)

for i in range(5):
    print(data_b[pp+i,N-1,3], ' ', data_b[pp+i,N-1,1], ' ', Y[pp+i,0], ' ', Y[pp+i,1])
```

```
data_b:            Y:
Bid       Ask       Bid       Ask
0.66071427   0.70535713   0.66071427   0.6875
0.66071427   0.6875   0.65178573   0.66071427
0.65178573   0.66071427   0.625   0.65178573
0.625   0.65178573   0.6160714   0.65178573
0.6160714   0.65178573   0.59821427   0.60714287
```

B [13]:
```
# We print Min and Max of all normalised columns of signs - All should be in the range [0; 1]
# Check that the matrices are normalised and there are no values> 1

print('\t','\t Max      Min')
mmx  = data_b[:,:,0].max()
mmn  = data_b[:,:,0].min()
print('Vol_Ask_N', '\t', mmx, '\t', mmn)
mmx  = data_b[:,:,1].max()
mmn  = data_b[:,:,1].min()
print('Ask_N_use', '\t', mmx, '\t', mmn)
mmx  = data_b[:,:,2].max()
mmn  = data_b[:,:,2].min()
print('DateDelta1_N', '\t', mmx, '\t', mmn)
mmx  = data_b[:,:,3].max()
mmn  = data_b[:,:,3].min()
print('Bid_N_use', '\t', mmx, '\t', mmn)
mmx  = data_b[:,:,4].max()
mmn  = data_b[:,:,4].min()
print('Vol_Bid_N', '\t', mmx, '\t', mmn)
```

```
                Max      Min
Vol_Ask_N       1.0     0.01
Ask_N_use       1.0     0.0027027028
DateDelta1_N    1.0     0.0025
Bid_N_use       0.99659866      0.0
Vol_Bid_N       1.0     0.01
```

data for subsequent testing

B [14]:
```
# We allocate about 90% of the data (two-dimensional subarrays) for training and about 10% of the data (two-dimensional subarrays
# n = data_a.shape [0] # new number of lines
train_start = 0
train_end = int(np.floor(0.9*L))
print(L, train_end)
```

```
1099800 989820
```

B [15]:
```
# For this example, let's leave a round number for training: 980,000 two-dimensional subarrays
train_end -= 9820
print(train_end)
```

```
980000
```

B [16]:
```
test_start = train_end + 1
test_end = L
print(test_start, test_end)
```

```
980001 1099800
```

B [17]:
```
print("numpy size:        ", data_b.shape, '\n')
```

```
numpy size:         (1099800, 200, 5)
```

B [18]:
```
print(test_end-train_end)
```

```
119800
```

B [19]:
```
# form X_train and X_test
# Divide data_b into two unequal parts: for training (X_train) and for test (X_test)

X_train = np.zeros( ( train_end, N, 5) , dtype= 'float32')
X_test  = np.zeros( ( test_end-train_end, N, 5) , dtype= 'float32')

for z in tqdm(range(test_end)):
    if z < train_end:
        X_train[z,:,:] = data_b[z,:,:]
    else:
        X_test[z-train_end,:,:]  = data_b[z,:,:]

# print('X_train:', '\n', X_test)
print(" numpy size::        ", X_train.shape, '\n')
print('\n')

# print('X_test:', '\n', X_test)
print(" numpy size::        ", X_test.shape, '\n')
```

```
100%|██████████| 1099800/1099800 [00:03<00:00, 355261.32it/s]
 numpy size::         (980000, 200, 5)


 numpy size::         (119800, 200, 5)
```

B [20]:
```
# Check the correctness of filling
print(X_train[:5,N-1,:])
print(X_train[-5:,N-1,:])
print(X_test[:5,N-1,:])
print(X_test[-5:,N-1,:])
```

```
[[0.101      0.36363637 0.00705    0.21212122 0.532     ]
 [0.187      0.36363637 0.1094     0.24242425 0.195     ]
 [0.157      0.33333334 0.0268     0.21212122 0.401     ]
 [0.112      0.21212122 0.02875    0.15151516 0.375     ]
 [0.307      0.24242425 0.03105    0.09090909 0.157     ]]
[[0.125      0.01754386 0.0449     0.         0.11      ]
 [0.157      0.06779661 0.0079     0.         0.195     ]
 [0.139      0.06779661 0.00255    0.         0.221     ]
 [0.514      0.05084746 0.0156     0.         0.1       ]
 [0.514      0.05084746 0.2208     0.         0.1       ]]
[[0.1        0.05084746 0.0264     0.         0.446     ]
 [0.131      0.04       0.026      0.         0.304     ]
 [0.162      0.06       0.0372     0.         0.409     ]
 [0.131      0.04       0.0274     0.02       0.206     ]
 [0.1        0.05882353 0.1014     0.         0.107     ]]
[[0.15       0.6875     0.0095     0.66071427 0.1       ]
 [0.1        0.70535713 0.0106     0.6875     0.195     ]
 [0.1        0.7589286  0.0163     0.7410714  0.244     ]
 [0.1        0.77678573 0.0026     0.75       0.382     ]
 [0.461      0.79464287 0.01045    0.7589286  0.393     ]]
```

Create an answer matrix for the training (y_train) and test (y_test) parts

B [21]:
```
y_train = Y[np.arange(train_start, train_end), :]
y_test = Y[np.arange(test_start,L), :]
print(train_start, train_end, test_start, L)

print('y_train(first 10):', '\n', y_train[:10,:])
print(" numpy size:         ", y_train.shape ,'\n')

data_add_row = np.array([y_test[L-test_start-1,0], y_test[L-test_start-1,1]])
print(L-test_start-1, data_add_row)
y_test = np.vstack([y_test, data_add_row])


print('y_test(last 10 ):', '\n', y_test[-10:,:])
print(" numpy size:         ", y_test.shape)
```

```
0 980000 980001 1099800
y_train(first 10):
 [[0.24242425 0.36363637]
 [0.21212122 0.33333334]
 [0.15151516 0.21212122]
 [0.09090909 0.24242425]
 [0.09090909 0.21212122]
 [0.18181819 0.24242425]
 [0.21212122 0.33333334]
 [0.24242425 0.36363637]
 [0.24242425 0.36363637]
 [0.21212122 0.33333334]]
 numpy size:          (980000, 2)

119798 [0.7589286  0.79464287]
y_test(last 10 ):
 [[0.65178573 0.6964286 ]
 [0.66071427 0.6964286 ]
 [0.65178573 0.66071427]
 [0.66071427 0.6875    ]
 [0.6875     0.70535713]
 [0.7410714  0.7589286 ]
 [0.75       0.77678573]
 [0.7589286  0.79464287]
 [0.7589286  0.79464287]
 [0.7589286  0.79464287]]
 numpy size:          (119800, 2)
```

Final control

В [22]:
```
# print X_train, y_train, X_test и y_test

print('Data for training:')
print("X_train:")
print("Type: ", X_train.dtype,  type(X_train), "; Size: ", X_train.shape)

print("\n","y_train:")
print("Type: ", y_train.dtype,  type(y_train), "; Size: ", y_train.shape)

print('Data for testing:')
print("\n","X_test:")
print("Type: ", X_test.dtype, type(X_test), "; Size: ", X_test.shape)

print("\n","y_test:")
print("Type: ", y_test.dtype,  type(y_test), "; Size: ", y_test.shape)
```

```
Data for training:
X_train:
Type:  float32 <class 'numpy.ndarray'> ; Size:  (980000, 200, 5)

 y_train:
Type:  float32 <class 'numpy.ndarray'> ; Size:  (980000, 2)
Data for testing:

 X_test:
Type:  float32 <class 'numpy.ndarray'> ; Size:  (119800, 200, 5)

 y_test:
Type:  float32 <class 'numpy.ndarray'> ; Size:  (119800, 2)
```

В [ ]:
```

```

Neural Network

Create NN

В [23]:
```
# Create a consistent model
model = keras.Sequential([

# First convolutional layer
# activation function = "relu"
# initial weights are initialized randomly using the normal distribution - kernel_initializer = "normal"
# Incoming data: matrix with size 200 (time) * 5 (signs). In this model, time is inside in one of the signs
# Take the convolution window 9 (you can and more: 9, 11, etc. - Odd)
    keras.layers.Conv1D(200, 9, padding='same', input_shape=(200, 5), activation=tf.nn.relu, kernel_initializer="normal"),

    keras.layers.MaxPooling1D(7),

    keras.layers.Conv1D(400, 7, padding='same', activation=tf.nn.relu, kernel_initializer="normal"),  # может надо оставить 200

    keras.layers.GlobalMaxPooling1D(),

    keras.layers.Dense(100, activation=tf.nn.relu, kernel_initializer="normal"),

# Слой регуляризации Dropout - ??? - может и не нужен !!
# model.add(Dropout(0.25))

# Output fully connected layer: 2-neuron - response - Bid and Ask price at the next tick
# Layer type = "Dense" - those are all neurons of one level connected to all neurons of the next level
# In our case, at the output of TWO neuron
# On the output layer, we DO NOT use a nonlinear activation function! We use an ordinary linear neuron
    keras.layers.Dense(2)

])

print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 200, 200)          9200
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 28, 200)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 28, 400)           560400
_________________________________________________________________
global_max_pooling1d (Global (None, 400)               0
_________________________________________________________________
dense (Dense)                (None, 100)               40100
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 202
=================================================================
Total params: 609,902
Trainable params: 609,902
Non-trainable params: 0
_________________________________________________________________
None
```

Compile the model

B [24]:

```
# Training method: optimizer - "adam"
# adam (Adaptive Moment Estimation) Optimizer is one of the most common optimizers

# Measure of error in the Backpropagation algorithm: mse (Mean squeared error)

# The metric by which we will perform optimization is mae (Mean absolute error) - those according to it
# stop the process. This is an absolute mistake compared to the correct answer in the test sample.
# You can use mse as a metric, but mae is easier to interpret

model.compile(optimizer="adam", loss="mse", metrics=["mae"])
```

Callback for ModelCheckpoint -memorizing learning outcomes after each epoch

B [25]:

```
from tensorflow.keras.callbacks import ModelCheckpoint


callbacks =[(ModelCheckpoint('convol_200_3_1week-{epoch:02d}-{val_loss:.4f}.hdf5', monitor='val_acc', save_best_only=True))]
```

B [26]:

```
model.fit?
```

B [29]:

```
model.fit_generator?
```

B [26]:

```
model.compile?
```

Trainig NN

B [25]:

```
history = model.fit(X_train, y_train, batch_size=100, epochs=15, validation_split=0.3, shuffle=True, verbose=1)

# use a stochastic gradient descent with a mini-sample size batch_size = 100, we analyze at each step
# 100 lines, we determine the direction of the gradient from them and, in accordance with this direction, determine the size of t
# the number of eras = 15 - those how many times we carry out training with the help of the entire data set, HOWEVER we can choos
# validation_split = 0.3 - the part of the sample that will be used as verification (those 30% of 960000)
# verbose = 2 - printing information in the learning process
```

```
Train on 686000 samples, validate on 294000 samples
Epoch 1/15
686000/686000 [==============================] - 62s 91us/sample - loss: 0.0021 - mae: 0.0304 - val_loss: 0.0014 - val_mae: 0.0
270
Epoch 2/15
686000/686000 [==============================] - 60s 87us/sample - loss: 0.0011 - mae: 0.0234 - val_loss: 0.0010 - val_mae: 0.0
225
Epoch 3/15
686000/686000 [==============================] - 60s 88us/sample - loss: 0.0010 - mae: 0.0224 - val_loss: 0.0010 - val_mae: 0.0
223
Epoch 4/15
686000/686000 [==============================] - 61s 89us/sample - loss: 9.7208e-04 - mae: 0.0219 - val_loss: 9.7933e-04 - val_
mae: 0.0219
Epoch 5/15
686000/686000 [==============================] - 61s 89us/sample - loss: 9.4774e-04 - mae: 0.0216 - val_loss: 9.4641e-04 - val_
mae: 0.0214
```

```
Epoch 6/15
686000/686000 [==============================] - 60s 88us/sample - loss: 9.2625e-04 - mae: 0.0213 - val_loss: 9.2846e-04 - val_
mae: 0.0211
Epoch 7/15
686000/686000 [==============================] - 60s 87us/sample - loss: 9.1360e-04 - mae: 0.0211 - val_loss: 0.0012 - val_mae:
0.0258
Epoch 8/15
686000/686000 [==============================] - 62s 90us/sample - loss: 9.0716e-04 - mae: 0.0210 - val_loss: 9.1828e-04 - val_
mae: 0.0208
Epoch 9/15
686000/686000 [==============================] - 61s 89us/sample - loss: 8.9388e-04 - mae: 0.0208 - val_loss: 9.6964e-04 - val_
mae: 0.0217
Epoch 10/15
686000/686000 [==============================] - 60s 88us/sample - loss: 8.8766e-04 - mae: 0.0207 - val_loss: 8.9967e-04 - val_
mae: 0.0207
Epoch 11/15
686000/686000 [==============================] - 60s 88us/sample - loss: 8.8330e-04 - mae: 0.0207 - val_loss: 0.0012 - val_mae:
0.0250
Epoch 12/15
686000/686000 [==============================] - 60s 87us/sample - loss: 8.7541e-04 - mae: 0.0205 - val_loss: 9.3341e-04 - val_
mae: 0.0212
Epoch 13/15
686000/686000 [==============================] - 61s 89us/sample - loss: 8.6814e-04 - mae: 0.0204 - val_loss: 9.1585e-04 - val_
mae: 0.0209
Epoch 14/15
686000/686000 [==============================] - 60s 87us/sample - loss: 8.6552e-04 - mae: 0.0204 - val_loss: 9.0021e-04 - val_
mae: 0.0206
Epoch 15/15
686000/686000 [==============================] - 60s 87us/sample - loss: 8.6225e-04 - mae: 0.0203 - val_loss: 9.1261e-04 - val_
mae: 0.0209
```

Evaluation

```
[26]: # Evaluate the quality of network training on test data (which the network did NOT see)
mse, mae = model.evaluate(X_test, y_test, verbose=0)
print("Mean absolute error: %.3f%%" % (mae*100))
```

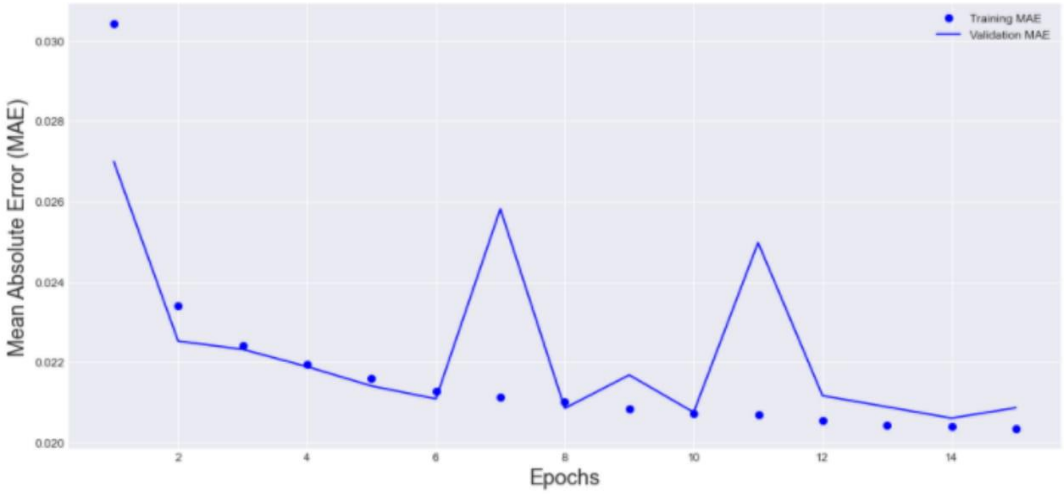
```
Mean absolute error: 2.919%
```

```
[27]: print("Mean squared error: %.3f%%" % (mse*100))
```

```
Mean squared error: 0.181%
```

Mean absolute error (MAE) graphs during training

```
[29]: history_dict = history.history
plt.style.use('seaborn-darkgrid')
mean_absolute_error_values = history_dict['mae']

val_mean_absolute_error_values = history_dict['val_mae']

epochs = range(1, len(mean_absolute_error_values) + 1)
plt.figure(num=1, figsize=(15,7))
plt.plot(epochs, mean_absolute_error_values, 'bo', label='Training MAE')
plt.plot(epochs, val_mean_absolute_error_values, 'b', label='Validation MAE')
plt.xlabel('Epochs', size=18)
plt.ylabel('Mean Absolute Error (MAE)', size=18)
plt.legend()
plt.show()
```



Using a Neural Network to Predict Bid and Ask Prices of the Next Tick - FOREX EURUSD

B [30]:
```
# reverse translation of relative prices to absolute
data_real = pd.read_csv('/Users/vital/Python_Code/_PhD_Code/2019-01-January/Tick/EURUSD_tick_real-1.csv')

print(data_real.head(n=5))
print(data_real[-5:])
```

```
         0        1        2    3    4
0  1.20102  1.20143  0.00041  6.0  0.0
1  1.20102  1.20148  0.00046  6.0  0.0
2  1.20102  1.20147  0.00045  6.0  0.0
3  1.20102  1.20148  0.00046  6.0  0.0
4  1.20102  1.20145  0.00043  6.0  0.0
               0        1        2    3    4
8543871  1.21214  1.21218  0.00004  6.0  0.0
8543872  1.21214  1.21217  0.00003  6.0  0.0
8543873  1.21212  1.21216  0.00004  6.0  0.0
8543874  1.21213  1.21215  0.00002  6.0  0.0
8543875  1.21213  1.21216  0.00003  6.0  0.0
```

B [31]:
```
# create numpay for min and delta for each 200-ticks

data_real = pd.read_csv('/Users/vital/Python_Code//_PhD_Code/2019-01-January/Tick/EURUSD_tick_real-1.csv')

data_real.drop('2', axis=1, inplace=True)
data_real.drop('3', axis=1, inplace=True)
data_real.drop('4', axis=1, inplace=True)

print(data_real.head(n=5))
print(data_real[-5:])

d_real = np.zeros((test_end-train_end, 2), dtype= 'float64')
# create a two-dimensional vector of real min, max and delta prices on each 200 tick slice
d_real[:(test_end-train_end)][:] = np.array(data_real[test_start-1:test_end][:])

print('d_real:', '\n', d_real)
print(" size        ", d_real.shape)
```

```
         0        1
0  1.20102  1.20143
1  1.20102  1.20148
2  1.20102  1.20147
3  1.20102  1.20148
4  1.20102  1.20145
               0        1
8543871  1.21214  1.21218
8543872  1.21214  1.21217
8543873  1.21212  1.21216
8543874  1.21213  1.21215
8543875  1.21213  1.21216
d_real:
 [[1.22699 1.22703]
 [1.22701 1.22703]
 [1.22701 1.22704]
 ...
 [1.22211 1.22212]
 [1.22208 1.22212]
 [1.22207 1.22211]]
 size         (119800, 2)
```

B [49]:
```
import random

pred = model.predict(X_test)

# Check: we take a random element random.randint () and look: what is the difference between absolute and relative

n_rec = random.randint(0, X_test.shape[0])
print(n_rec)


print('Relative parameters:')
print("Predicted price:", pred[n_rec], ", right price:", y_test[n_rec], ", error:", pred[n_rec]-y_test[n_rec],
      "%error:", (pred[n_rec]-y_test[n_rec])/y_test[n_rec]*100,"%")
print('\n')
print('Absolute parameters:')
print("Predicted price:", pred[n_rec]*d_real[n_rec][1] + d_real[n_rec][0], ", right price:",
      y_test[n_rec]*d_real[n_rec][1] + d_real[n_rec][0], ", error:", (pred[n_rec]-y_test[n_rec])*d_real[n_rec][1],
      "%error:", (pred[n_rec]-y_test[n_rec])/y_test[n_rec]*100,"%")
```

```
17965
Relative parameters:
Predicted price: [0.563716   0.64874154] , right price: [0.5609756 0.6585366] , error: [ 0.00274038 -0.00979507] %error: [ 0.48
85031 -1.4873995] %


Absolute parameters:
Predicted price: [1.9219859 2.0264943] , right price: [1.9186175 2.0385337] , error: [ 0.00336831 -0.01203951] %error: [ 0.4885
031 -1.4873995] %
```

Price Chart

B [33]:

```
plt.style.use('seaborn-darkgrid')
plt.ion()
#fig = plt.figure(figsize=(23,8))
fig = plt.figure(figsize=(17,7))

ax1 = fig.add_subplot(111)
line1, = ax1.plot(y_test[:400,0], label='Bid')
line2, = ax1.plot(pred[:400,0], label='Predicted Bid')
line3, = ax1.plot(y_test[0:400,1], label='Ask')
line4, = ax1.plot(pred[:400,1], label='Predicted Ask')

plt.title('Real & Predicted Normalised Prices', size=22)
plt.ylabel('EURO/USD Tick Rates(Prices)', size=18)
plt.xlabel('$Ticks$', size=18)
plt.legend(loc='upper right')

plt.show()
```

B [34]:

```
print(y_test.shape, '  ', pred.shape, '  ', d_real.shape)
```

```
(119800, 2)    (119800, 2)    (119800, 2)
```

Detailed price chart

B [35]:

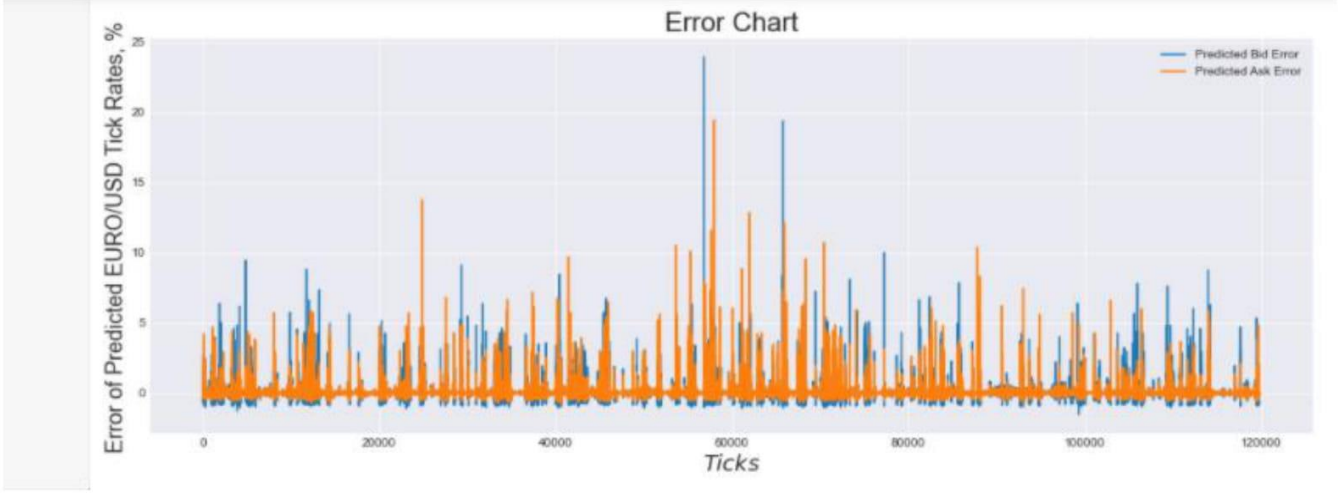
```
# Setting an interactive chart of ABSOLUTE prices
N1=26000
N2=27000
plt.style.use('seaborn-darkgrid')
plt.ion()
fig = plt.figure(figsize=(17,6))
ax1 = fig.add_subplot(111)
line1, = ax1.plot(y_test[N1:N2,0]*d_real[N1:N2,1] + d_real[N1:N2,0], label='Bid')
line2, = ax1.plot(pred[N1:N2,0]*d_real[N1:N2,1] + d_real[N1:N2,0], label='Predicted Bid')
line3, = ax1.plot(y_test[N1:N2,1]*d_real[N1:N2,1] + d_real[N1:N2,0], label='Ask')
line4, = ax1.plot(pred[N1:N2,1]*d_real[N1:N2,1] + d_real[N1:N2,0], label='Predicted Ask')

plt.title('Real & Predicted EURO/USD Tick Rates (Prices)', size=22)
plt.ylabel('EURO/USD Tick Rates (Prices)', size=18)
plt.xlabel('$Ticks$', size=18)
plt.legend(loc='upper right')

plt.show()
```

B [36]:
```
# Setting an interactive chart of ABSOLUTE prices
N1=26700   # start
N2=27000   # end
plt.style.use('seaborn-darkgrid')
plt.ion()
fig = plt.figure(figsize=(17,6))
ax1 = fig.add_subplot(111)
line1, = ax1.plot(y_test[N1:N2,0]*d_real[N1:N2,1] + d_real[N1:N2,0], label='Bid')
line2, = ax1.plot(pred[N1:N2,0]*d_real[N1:N2,1] + d_real[N1:N2,0], label='Predicted Bid')
line3, = ax1.plot(y_test[N1:N2,1]*d_real[N1:N2,1] + d_real[N1:N2,0], label='Ask')
line4, = ax1.plot(pred[N1:N2,1]*d_real[N1:N2,1] + d_real[N1:N2,0], label='Predicted Ask')

plt.title('Real & Predicted EURO/USD Tick Rates (Prices)', size=22)
plt.ylabel('EURO/USD Tick Rates (Prices)', size=18)
plt.xlabel('$Ticks$', size=18)
plt.legend(loc='upper right')

plt.show()
```

Error Chart

B [37]:

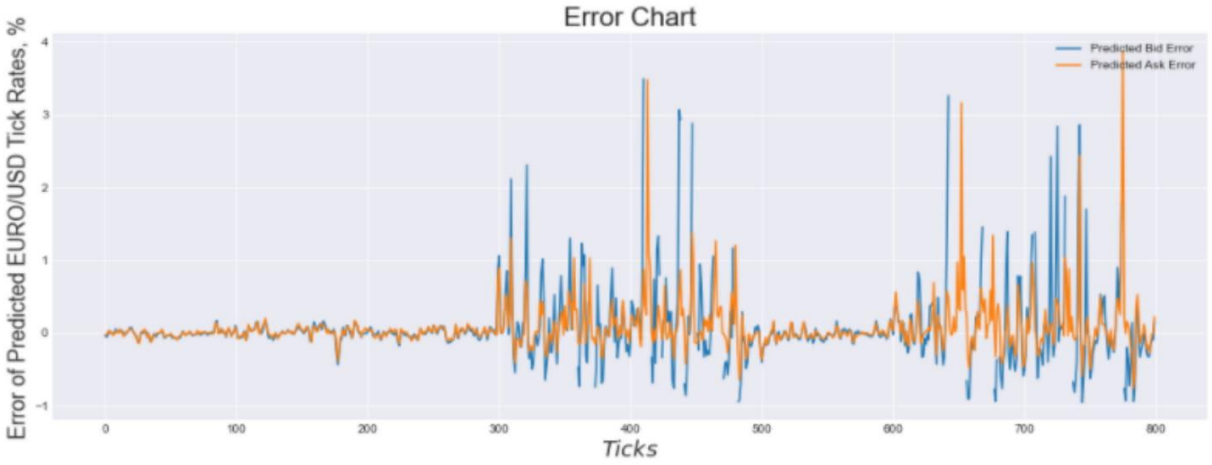
```
plt.style.use('seaborn-darkgrid')
plt.ion()
fig = plt.figure(figsize=(17,6))
ax1 = fig.add_subplot(111)
#a = (pred[:,0]-y_test[:,0])/y_test[:,0]*100
a = (pred[:,0]-y_test[:,0])/y_test[:,0]
#b = (pred[:,1]-y_test[:,1])/y_test[:,1]*100
b = (pred[:,1]-y_test[:,1])/y_test[:,1]
line1, = ax1.plot(a, label='Predicted Bid Error')
line2, = ax1.plot(b, label='Predicted Ask Error')

plt.title('Error Chart', size=22)
plt.ylabel('Error of Predicted EURO/USD Tick Rates, %', size=18)
plt.xlabel('$Ticks$', size=18)
plt.legend(loc='upper right')

plt.show()
```



Detailed graph of Errors

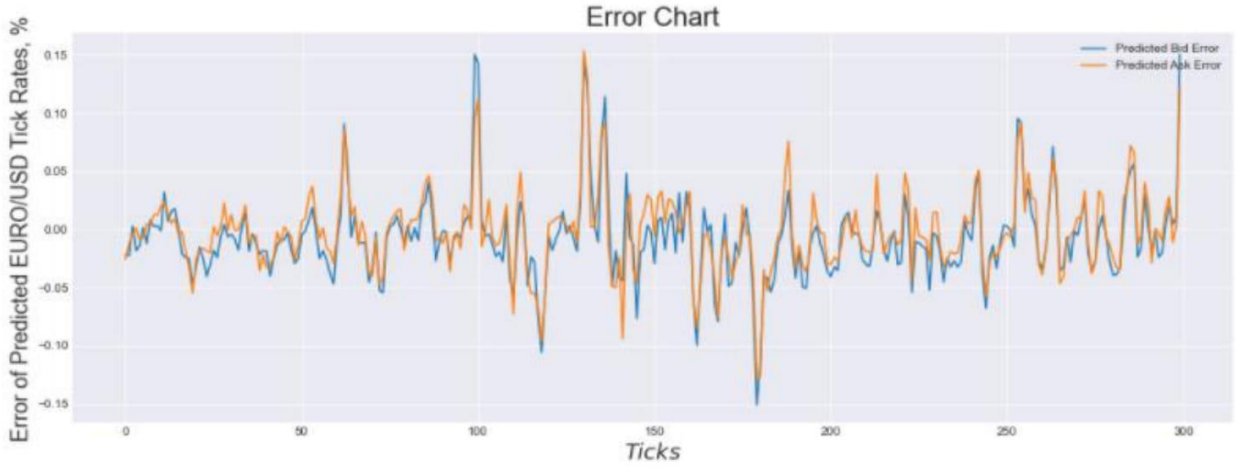
B [38]:

```
N1=32700      # start
N2=33500   # end

plt.style.use('seaborn-darkgrid')
plt.ion()
fig = plt.figure(figsize=(17,6))
ax1 = fig.add_subplot(111)
#a = (pred[N1:N2,0]-y_test[N1:N2,0])/y_test[N1:N2,0]*100
a = (pred[N1:N2,0]-y_test[N1:N2,0])/y_test[N1:N2,0]
#b = (pred[N1:N2,1]-y_test[N1:N2,1])/y_test[N1:N2,1]*100
b = (pred[N1:N2,1]-y_test[N1:N2,1])/y_test[N1:N2,1]
line1, = ax1.plot(a, label='Predicted Bid Error')
line2, = ax1.plot(b, label='Predicted Ask Error')

plt.title('Error Chart', size=22)
plt.ylabel('Error of Predicted EURO/USD Tick Rates, %', size=18)
plt.xlabel('$Ticks$', size=18)
plt.legend(loc='upper right')

plt.show()
```



B [39]:

```
N1=35700      # start
N2=36000   # end

plt.style.use('seaborn-darkgrid')
plt.ion()
fig = plt.figure(figsize=(17,6))
ax1 = fig.add_subplot(111)
#a = (pred[N1:N2,0]-y_test[N1:N2,0])/y_test[N1:N2,0]*100
a = (pred[N1:N2,0]-y_test[N1:N2,0])/y_test[N1:N2,0]
#b = (pred[N1:N2,1]-y_test[N1:N2,1])/y_test[N1:N2,1]*100
b = (pred[N1:N2,1]-y_test[N1:N2,1])/y_test[N1:N2,1]
line1, = ax1.plot(a, label='Predicted Bid Error')
line2, = ax1.plot(b, label='Predicted Ask Error')

plt.title('Error Chart', size=22)
plt.ylabel('Error of Predicted EURO/USD Tick Rates, %', size=18)
plt.xlabel('$Ticks$', size=18)
plt.legend(loc='upper right')

plt.show()
```



B [ ]:

Appendix XI.

Recurrent Neural Networks (GRU and LSTM)

Forex EURUSD Price Prediction using Keras

Tick Data with Dukastcopy

Data for 2020 year

In [1]:
```
# TensorFlow and tf.keras

import tensorflow as tf
from tensorflow import keras

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
print(tf.__version__)
```

```
2.4.1
```

Data

In [3]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline
```

In [4]:
```
data_2 = pd.read_csv('/Users/vital/Python_Code/_PhD_Code/2021-04-April/Data/2020/EURUSD_ticks_2020_1mln.csv', low_memory=False, s

print(data_2.head(n=5))
print(data_2[-5:])
```

```
              Time (UTC)      Ask      Bid  AskVolume  BidVolume
0  2020.03.10 17:18:51.089  1.13075  1.13070       0.94       1.20
1  2020.03.10 17:18:51.646  1.13073  1.13071       0.75       0.37
2  2020.03.10 17:18:51.697  1.13073  1.13069       0.75       0.56
3  2020.03.10 17:18:51.798  1.13074  1.13070       0.94       1.12
4  2020.03.10 17:18:51.899  1.13074  1.13071       0.19       1.12
                    Time (UTC)      Ask      Bid  AskVolume  BidVolume
999995  2020.03.16 08:35:01.232  1.12116  1.12107       1.46       0.94
999996  2020.03.16 08:35:01.283  1.12111  1.12105       0.82       0.19
999997  2020.03.16 08:35:01.334  1.12111  1.12103       0.15       1.00
999998  2020.03.16 08:35:01.435  1.12110  1.12103       0.82       0.19
999999  2020.03.16 08:35:01.537  1.12111  1.12104       0.45       0.19
```

In [5]:
```
data_2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 5 columns):
 #   Column      Non-Null Count    Dtype  
---  ------      --------------    -----  
 0   Time (UTC)  1000000 non-null  object 
 1   Ask         1000000 non-null  float64
 2   Bid         1000000 non-null  float64
 3   AskVolume   1000000 non-null  float64
 4   BidVolume   1000000 non-null  float64
dtypes: float64(4), object(1)
memory usage: 38.1+ MB
```

In [6]:
```
data_2 = data_2.drop(['Time (UTC)'], axis=1)

print(data_2.dtypes)
print(data_2.head(n=3))
print(data_2[-3:])
```

```
Ask          float64
Bid          float64
AskVolume    float64
BidVolume    float64
dtype: object
       Ask      Bid  AskVolume  BidVolume
0  1.13075  1.13070       0.94       1.20
1  1.13073  1.13071       0.75       0.37
2  1.13073  1.13069       0.75       0.56
             Ask      Bid  AskVolume  BidVolume
999997  1.12111  1.12103       0.15       1.00
999998  1.12110  1.12103       0.82       0.19
999999  1.12111  1.12104       0.45       0.19
```

In [7]:
```
data_2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 4 columns):
 #   Column     Non-Null Count    Dtype
---  ------     --------------    -----
 0   Ask        1000000 non-null  float64
 1   Bid        1000000 non-null  float64
 2   AskVolume  1000000 non-null  float64
 3   BidVolume  1000000 non-null  float64
dtypes: float64(4)
memory usage: 30.5 MB
```

In [8]:
```
# create numpy
```

In [9]:
```
n = data_2.shape[0]
p = data_2.shape[1]

print(n, p)
```

```
1000000 4
```

In [10]:
```
data_a = np.array(data_2[:n])

print("data_a:")
print(data_a[:4])
print('\n')
print(data_a[-4:])


print(type(data_a), '\n',
         data_a.shape)
print('Example: data_a[1,0] = ', data_a[1,0])
```


```
print('Example: data_a[1,0] = ', data_a[1,0])
```

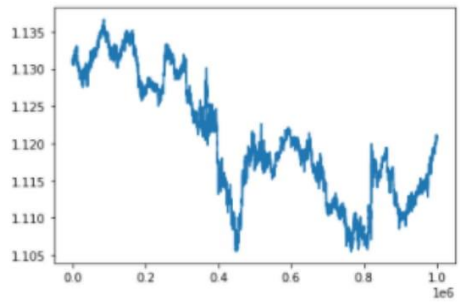

```
data_a:
[[1.13075 1.1307  0.94    1.2    ]
 [1.13073 1.13071 0.75    0.37   ]
 [1.13073 1.13069 0.75    0.56   ]
 [1.13074 1.1307  0.94    1.12   ]]


[[1.12111 1.12105 0.82    0.19   ]
 [1.12111 1.12103 0.15    1.     ]
 [1.1211  1.12103 0.82    0.19   ]
 [1.12111 1.12104 0.45    0.19   ]]
<class 'numpy.ndarray'>
 (1000000, 4)
Example: data_a[1,0] =  1.13073
```

In [11]:
```
data_a.size
```

Out[11]:
```
4000000
```

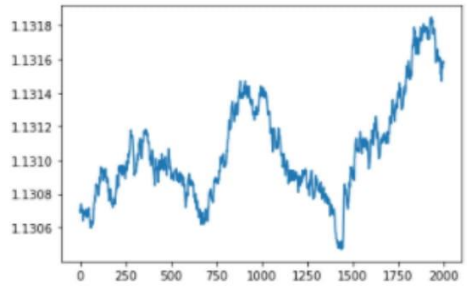
In [12]:
```
ticks_Bid = data_a[:, 1]  # Bid
ticks_Ask = data_a[:, 0]  # Ask
plt.plot(range(len(ticks_Bid)),ticks_Bid)
```

Out[12]:
```
[<matplotlib.lines.Line2D at 0x1a3f8459490>]
```



In [13]:
```
# Bid - the first 2000 ticks

plt.plot(range(2000), ticks_Bid[:2000])
```

Out[13]:
```
[<matplotlib.lines.Line2D at 0x1a384491fa0>]
```



In [14]:
```
step     = 1
lookback = 200 # sliding window
delay    = 1
```

In [15]:
```
# Normalisation

mean1 = data_a[:500000].mean(axis=0)
data_a -= mean1
std1 = data_a[:500000].std(axis=0)
data_a /= std1
```

In [16]:
```
# Generater

def generator(data,
              lookback,
              delay,
              min_index,
              max_index,
              shuffle=False,
              batch_size=128,
              step=6):
    if max_index is None:
        max_index = len(data) - delay - 1
    i = min_index + lookback
    while 1:
        if shuffle:
            rows = np.random.randint(
                min_index + lookback, max_index, size=batch_size)
        else:
            if i + batch_size >= max_index:
                i = min_index + lookback
            rows = np.arange(i, min(i + batch_size, max_index))
            i += len(rows)
        samples = np.zeros((len(rows),
                            lookback // step,
                            data.shape[-1]))
        targets = np.zeros((len(rows),))
        for j, row in enumerate(rows):
            indices = range(rows[j] - lookback, rows[j], step)
            samples[j] = data[indices]
            targets[j] = data[rows[j] + delay][1]
        yield samples, targets
```

B [18]:
```
batch_size = 80

train_gen = generator(data_a,
                      lookback = lookback,
                      delay = delay,
                      min_index = 0,
                      max_index = 500000,
                      shuffle = True,
                      step = step,
                      batch_size = batch_size)

val_gen = generator(data_a,
                      lookback = lookback,
                      delay = delay,
                      min_index = 500001,
                      max_index = 800000,
                      step = step,
                      batch_size = batch_size)

test_gen = generator(data_a,
                      lookback = lookback,
                      delay = delay,
                      min_index = 800001,
                      max_index = None,
                      step = step,
                      batch_size = batch_size)


val_steps = (800000 - 500001 - lookback) // batch_size


test_steps = (len(data_a) - 800001 - lookback) // batch_size
```

Recurrent Neural Network-1

Gated Recurrent Unit (GRU)

B [77]:
```
model = keras.Sequential([

    keras.layers.GRU(32, dropout=0.1, recurrent_dropout=0.5, return_sequences=True, input_shape=(None, data_a.shape[-1])),

    keras.layers.GRU(64, activation=tf.nn.relu, dropout=0.1, recurrent_dropout=0.5),

    keras.layers.Dense(1)

])

print(model.summary())
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_2 (GRU)                  (None, None, 32)          3648
_________________________________________________________________
gru_3 (GRU)                  (None, 64)                18816
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 65
=================================================================
Total params: 22,529
Trainable params: 22,529
Non-trainable params: 0
_________________________________________________________________
None
```

B [78]:
```
model.compile(optimizer=keras.optimizers.RMSprop(), loss="mae")
```

```
B [79]: history = model.fit_generator(train_gen, steps_per_epoch=500, epochs=20, validation_data=val_gen, validation_steps=val_steps)
```

```
C:\Users\vital\anaconda3_\lib\site-packages\tensorflow\python\keras\engine\training.py:1844: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
```

```
Epoch 1/20
500/500 [==============================] - 17s 26ms/step - loss: 0.1306 - val_loss: 0.0336
Epoch 2/20
500/500 [==============================] - 12s 24ms/step - loss: 0.0628 - val_loss: 0.1715
Epoch 3/20
500/500 [==============================] - 12s 25ms/step - loss: 0.0496 - val_loss: 0.1878
Epoch 4/20
500/500 [==============================] - 13s 25ms/step - loss: 0.0448 - val_loss: 0.1881
Epoch 5/20
500/500 [==============================] - 13s 26ms/step - loss: 0.0418 - val_loss: 0.1866
Epoch 6/20
500/500 [==============================] - 13s 26ms/step - loss: 0.0400 - val_loss: 0.1875
Epoch 7/20
500/500 [==============================] - 13s 26ms/step - loss: 0.0383 - val_loss: 0.1452
Epoch 8/20
500/500 [==============================] - 13s 26ms/step - loss: 0.0377 - val_loss: 0.1289
Epoch 9/20
500/500 [==============================] - 13s 26ms/step - loss: 0.0372 - val_loss: 0.1575
Epoch 10/20
500/500 [==============================] - 12s 25ms/step - loss: 0.0360 - val_loss: 0.1114
Epoch 11/20
500/500 [==============================] - 12s 25ms/step - loss: 0.0358 - val_loss: 0.1803
Epoch 12/20
500/500 [==============================] - 13s 25ms/step - loss: 0.0348 - val_loss: 0.1067
Epoch 13/20
500/500 [==============================] - 13s 25ms/step - loss: 0.0347 - val_loss: 0.1066
Epoch 14/20
500/500 [==============================] - 12s 25ms/step - loss: 0.0340 - val_loss: 0.1391
Epoch 15/20
500/500 [==============================] - 12s 25ms/step - loss: 0.0337 - val_loss: 0.1603
Epoch 16/20
500/500 [==============================] - 13s 26ms/step - loss: 0.0338 - val_loss: 0.1623
Epoch 17/20
500/500 [==============================] - 14s 27ms/step - loss: 0.0333 - val_loss: 0.1981
Epoch 18/20
500/500 [==============================] - 13s 26ms/step - loss: 0.0329 - val_loss: 0.1559
Epoch 19/20
500/500 [==============================] - 12s 25ms/step - loss: 0.0325 - val_loss: 0.1517
Epoch 20/20
500/500 [==============================] - 13s 26ms/step - loss: 0.0321 - val_loss: 0.1574
```

B [80]:
```
plt.style.use('seaborn-darkgrid')
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) +1)
plt.figure(num=1, figsize=(15,8))

#plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

B [ ]:

Recurrent Neural Network-2

LSTM

B [68]:
```
model = keras.Sequential([

    keras.layers.LSTM(32, input_shape=(None, data_a.shape[-1])),

    keras.layers.Dense(1)

])

print(model.summary())
```

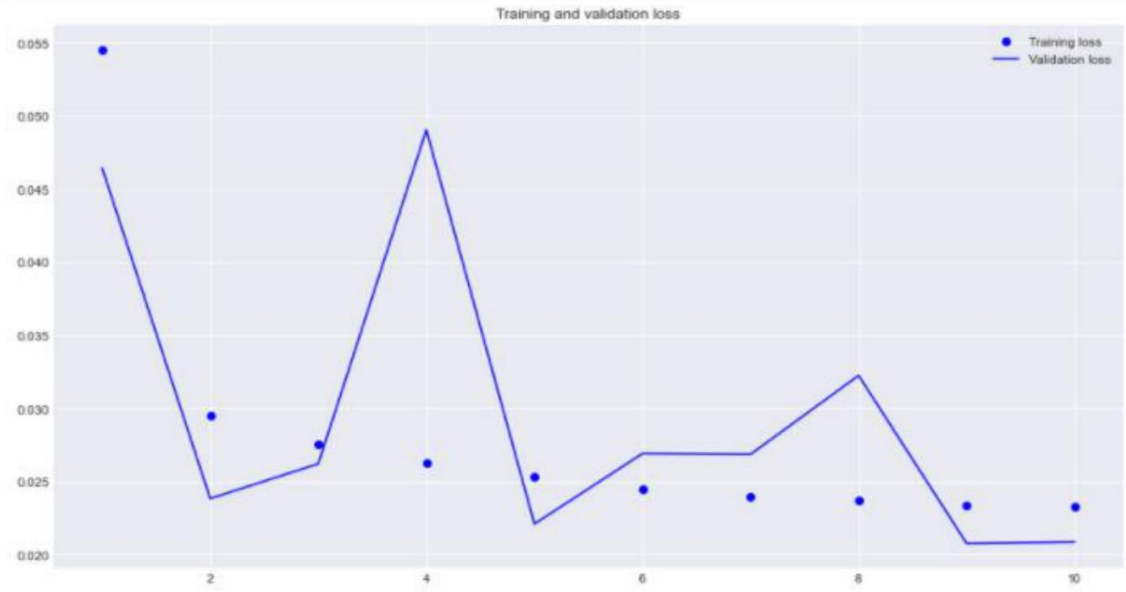
```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_7 (LSTM)                (None, 32)                4736
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 33
=================================================================
Total params: 4,769
Trainable params: 4,769
Non-trainable params: 0
_________________________________________________________________
None
```

B [69]:
```
model.compile(optimizer=keras.optimizers.RMSprop(), loss="mae")
```

In [70]:
```
history = model.fit_generator(train_gen, steps_per_epoch=500, epochs=10, validation_data=val_gen, validation_steps=val_steps)
```

```
C:\Users\vital\anaconda3_\lib\site-packages\tensorflow\python\keras\engine\training.py:1844: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/10
500/500 [==============================] - 10s 16ms/step - loss: 0.1180 - val_loss: 0.0464
Epoch 2/10
500/500 [==============================] - 7s 15ms/step - loss: 0.0303 - val_loss: 0.0238
Epoch 3/10
500/500 [==============================] - 7s 14ms/step - loss: 0.0280 - val_loss: 0.0262
Epoch 4/10
500/500 [==============================] - 7s 15ms/step - loss: 0.0268 - val_loss: 0.0491
Epoch 5/10
500/500 [==============================] - 7s 15ms/step - loss: 0.0255 - val_loss: 0.0221
Epoch 6/10
500/500 [==============================] - 7s 15ms/step - loss: 0.0244 - val_loss: 0.0269
Epoch 7/10
500/500 [==============================] - 7s 15ms/step - loss: 0.0240 - val_loss: 0.0269
Epoch 8/10
500/500 [==============================] - 7s 15ms/step - loss: 0.0238 - val_loss: 0.0322
Epoch 9/10
500/500 [==============================] - 7s 15ms/step - loss: 0.0234 - val_loss: 0.0208
Epoch 10/10
500/500 [==============================] - 7s 15ms/step - loss: 0.0235 - val_loss: 0.0209
```

In [71]:
```
plt.style.use('seaborn-darkgrid')
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) +1)
plt.figure(num=1, figsize=(15,8))


plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```



In [ ]:
```

```

Appendix XII.

Volatility Analysis

Data Dukascopy Bank: Forex EUR/USD - 2020

In [1]:
```
# TensorFlow and tf.keras

import tensorflow as tf
from tensorflow import keras
```

In [2]:
```
print(tf.__version__)
```

```
2.0.0
```

In [3]:
```
# import

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline
```

In [4]:
```
from tqdm import tqdm
```

In [5]:
```
#  plotly

import chart_studio.plotly as py
import plotly.graph_objects as go
from plotly.offline import iplot
import cufflinks as cf
cf.go_offline()
```

In [ ]:
```

```

Preliminary Data Analysis (2020)

In [6]:
```
data_raw = pd.read_csv('Data/EURUSD_Ticks_2020.01.01_2020.06.30.csv', low_memory=False, sep=',')

print(data_raw.head(n=5))
print(data_raw[-5:])
```

```
              Time (UTC)      Ask      Bid  AskVolume  BidVolume
0  2020.01.01 22:01:12.821  1.12160  1.12106     0.0937       0.75
1  2020.01.01 22:01:17.176  1.12160  1.12120     0.0937       0.75
2  2020.01.01 22:01:18.545  1.12160  1.12117     0.0937       0.75
3  2020.01.01 22:01:19.145  1.12161  1.12123     0.8400       0.75
4  2020.01.01 22:01:19.246  1.12161  1.12120     0.8400       0.75
                        Time (UTC)      Ask      Bid  AskVolume  BidVolume
16586025  2020.06.30 23:58:40.816  1.12337  1.12333       0.75       5.06
16586026  2020.06.30 23:58:40.917  1.12338  1.12333       1.57       3.94
16586027  2020.06.30 23:58:47.486  1.12337  1.12333       0.75       3.94
16586028  2020.06.30 23:58:52.038  1.12338  1.12333       2.02       3.94
16586029  2020.06.30 23:58:55.919  1.12338  1.12334       1.50       0.37
```

In [7]:
```
data_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16586030 entries, 0 to 16586029
Data columns (total 5 columns):
 #   Column      Dtype
---  ------      -----
 0   Time (UTC)  object
 1   Ask         float64
 2   Bid         float64
 3   AskVolume   float64
 4   BidVolume   float64
dtypes: float64(4), object(1)
memory usage: 632.7+ MB
```

In [8]:
```
# sample-2

indexes_to_drop = np.array(range(N,data_raw.shape[0]))
indexes_to_keep = set(range(data_raw.shape[0])) - set(indexes_to_drop)
data2 = data_raw.take(list(indexes_to_keep))

print(data2.head(n=5))
print(data2[-5:])
```

```
              Time (UTC)      Ask      Bid  AskVolume  BidVolume
0  2020.01.01 22:01:12.821  1.12160  1.12106     0.0937       0.75
1  2020.01.01 22:01:17.176  1.12160  1.12120     0.0937       0.75
2  2020.01.01 22:01:18.545  1.12160  1.12117     0.0937       0.75
3  2020.01.01 22:01:19.145  1.12161  1.12123     0.8400       0.75
4  2020.01.01 22:01:19.246  1.12161  1.12120     0.8400       0.75
                        Time (UTC)      Ask      Bid  AskVolume  BidVolume
1499995  2020.01.30 07:11:33.290  1.10087  1.10086       0.45       0.75
1499996  2020.01.30 07:11:33.392  1.10086  1.10083       0.75       4.57
1499997  2020.01.30 07:11:37.666  1.10086  1.10084       0.75       0.75
1499998  2020.01.30 07:11:37.768  1.10087  1.10085       2.44       1.50
1499999  2020.01.30 07:11:38.045  1.10086  1.10084       0.75       1.50
```

In [9]:
```
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1500000 entries, 0 to 1499999
Data columns (total 5 columns):
 #   Column      Non-Null Count    Dtype
---  ------      --------------    -----
 0   Time (UTC)  1500000 non-null  object
 1   Ask         1500000 non-null  float64
 2   Bid         1500000 non-null  float64
 3   AskVolume   1500000 non-null  float64
 4   BidVolume   1500000 non-null  float64
dtypes: float64(4), object(1)
memory usage: 68.7+ MB
```

In [17]:
```
#sample-3

N0 = 4500000
N = 1500000 + N0

indexes_to_drop1 = np.array(range(N,data_raw.shape[0]))
indexes_to_drop2 = np.array(range(0,N0))
indexes_to_keep = set(range(data_raw.shape[0])) - set(indexes_to_drop1) - set(indexes_to_drop2)
data3 = data_raw.take(list(indexes_to_keep))

print(data3.head(n=5))
print(data3.tail(5))
print(data3.shape)
```

```
                        Time (UTC)      Ask      Bid  AskVolume  BidVolume
4500000  2020.03.10 17:18:51.089  1.13075  1.13070       0.94       1.20
4500001  2020.03.10 17:18:51.646  1.13073  1.13071       0.75       0.37
4500002  2020.03.10 17:18:51.697  1.13073  1.13069       0.75       0.56
4500003  2020.03.10 17:18:51.798  1.13074  1.13070       0.94       1.12
4500004  2020.03.10 17:18:51.899  1.13074  1.13071       0.19       1.12
                        Time (UTC)      Ask      Bid  AskVolume  BidVolume
5999995  2020.03.17 19:37:31.855  1.10055  1.10050       0.19       0.82
5999996  2020.03.17 19:37:32.216  1.10055  1.10049       0.19       0.82
5999997  2020.03.17 19:37:33.192  1.10052  1.10047       0.19       1.50
5999998  2020.03.17 19:37:33.294  1.10048  1.10042       0.19       0.82
5999999  2020.03.17 19:37:33.426  1.10047  1.10041       0.56       0.75
(1500000, 5)
```

In [18]:
```
data3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1500000 entries, 4500000 to 5999999
Data columns (total 5 columns):
 #   Column      Non-Null Count    Dtype
---  ------      --------------    -----
 0   Time (UTC)  1500000 non-null  object
 1   Ask         1500000 non-null  float64
 2   Bid         1500000 non-null  float64
 3   AskVolume   1500000 non-null  float64
 4   BidVolume   1500000 non-null  float64
dtypes: float64(4), object(1)
memory usage: 68.7+ MB
```

B [19]:
```
# compression

pd.options.mode.chained_assignment = None  # default='warn'

data2['Ask']       = pd.to_numeric(data2['Ask'],downcast='float')
data2['Bid']       = pd.to_numeric(data2['Bid'],downcast='float')
data2['AskVolume'] = pd.to_numeric(data2['AskVolume'],downcast='float')
data2['BidVolume'] = pd.to_numeric(data2['BidVolume'],downcast='float')

data3['Ask']       = pd.to_numeric(data3['Ask'],downcast='float')
data3['Bid']       = pd.to_numeric(data3['Bid'],downcast='float')
data3['AskVolume'] = pd.to_numeric(data3['AskVolume'],downcast='float')
data3['BidVolume'] = pd.to_numeric(data3['BidVolume'],downcast='float')

print(data2.info())

print(data3.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1500000 entries, 0 to 1499999
Data columns (total 5 columns):
 #   Column      Non-Null Count    Dtype
---  ------      --------------    -----
 0   Time (UTC)  1500000 non-null  object
 1   Ask         1500000 non-null  float32
 2   Bid         1500000 non-null  float32
 3   AskVolume   1500000 non-null  float32
 4   BidVolume   1500000 non-null  float32
dtypes: float32(4), object(1)
memory usage: 45.8+ MB
None
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1500000 entries, 4500000 to 5999999
Data columns (total 5 columns):
 #   Column      Non-Null Count    Dtype
---  ------      --------------    -----
 0   Time (UTC)  1500000 non-null  object
 1   Ask         1500000 non-null  float32
 2   Bid         1500000 non-null  float32
 3   AskVolume   1500000 non-null  float32
 4   BidVolume   1500000 non-null  float32
dtypes: float32(4), object(1)
memory usage: 45.8+ MB
None
```

B [20]:
```
data2[:6]
```

Out[20]:

	Time (UTC)	Ask	Bid	AskVolume	BidVolume
0	2020.01.01 22:01:12.821	1.12160	1.12106	0.0937	0.75
1	2020.01.01 22:01:17.176	1.12160	1.12120	0.0937	0.75
2	2020.01.01 22:01:18.545	1.12160	1.12117	0.0937	0.75
3	2020.01.01 22:01:19.145	1.12161	1.12123	0.8400	0.75
4	2020.01.01 22:01:19.246	1.12161	1.12120	0.8400	0.75
5	2020.01.01 22:01:21.124	1.12161	1.12122	0.0937	0.75

B [21]:
```
data3[:6]
```

Out[21]:

	Time (UTC)	Ask	Bid	AskVolume	BidVolume
4500000	2020.03.10 17:18:51.089	1.13075	1.13070	0.94	1.20
4500001	2020.03.10 17:18:51.646	1.13073	1.13071	0.75	0.37
4500002	2020.03.10 17:18:51.697	1.13073	1.13069	0.75	0.56
4500003	2020.03.10 17:18:51.798	1.13074	1.13070	0.94	1.12
4500004	2020.03.10 17:18:51.899	1.13074	1.13071	0.19	1.12
4500005	2020.03.10 17:18:51.950	1.13075	1.13073	0.19	0.37

Date

B [22]:
```
pd.to_datetime(['31/07/2017', 'test'], errors='coerce') # errors='coerce'

data_raw['DateT'] = pd.to_datetime(data_raw['Time (UTC)'])
print(data_raw.dtypes)
print(data_raw.head(n=5))
print(data_raw[-5:])
```

```
Time (UTC)            object
Ask                  float64
Bid                  float64
AskVolume            float64
BidVolume            float64
DateT         datetime64[ns]
dtype: object
                Time (UTC)      Ask      Bid  AskVolume  BidVolume  \
0  2020.01.01 22:01:12.821  1.12160  1.12106     0.0937       0.75
1  2020.01.01 22:01:17.176  1.12160  1.12120     0.0937       0.75
2  2020.01.01 22:01:18.545  1.12160  1.12117     0.0937       0.75
3  2020.01.01 22:01:19.145  1.12161  1.12123     0.8400       0.75
4  2020.01.01 22:01:19.246  1.12161  1.12120     0.8400       0.75

                    DateT
0 2020-01-01 22:01:12.821
1 2020-01-01 22:01:17.176
2 2020-01-01 22:01:18.545
3 2020-01-01 22:01:19.145
4 2020-01-01 22:01:19.246
                      Time (UTC)      Ask      Bid  AskVolume  BidVolume  \
16586025  2020.06.30 23:58:40.816  1.12337  1.12333       0.75       5.06
16586026  2020.06.30 23:58:40.917  1.12338  1.12333       1.57       3.94
16586027  2020.06.30 23:58:47.486  1.12337  1.12333       0.75       3.94
16586028  2020.06.30 23:58:52.038  1.12338  1.12333       2.02       3.94
16586029  2020.06.30 23:58:55.919  1.12338  1.12334       1.50       0.37

                             DateT
16586025 2020-06-30 23:58:40.816
16586026 2020-06-30 23:58:40.917
16586027 2020-06-30 23:58:47.486
16586028 2020-06-30 23:58:52.038
16586029 2020-06-30 23:58:55.919
```

B [23]:
```
pd.to_datetime(['31/07/2017', 'test'], errors='coerce') # errors='coerce' -

# reformate date
data2['DateT'] = pd.to_datetime(data2['Time (UTC)'])

print(data2.dtypes)
print(data2.head(n=5))
print(data2[-5:])
```

```
Time (UTC)            object
Ask                  float32
Bid                  float32
AskVolume            float32
BidVolume            float32
DateT         datetime64[ns]
dtype: object
                Time (UTC)      Ask      Bid  AskVolume  BidVolume  \
0  2020.01.01 22:01:12.821  1.12160  1.12106     0.0937       0.75
1  2020.01.01 22:01:17.176  1.12160  1.12120     0.0937       0.75
2  2020.01.01 22:01:18.545  1.12160  1.12117     0.0937       0.75
3  2020.01.01 22:01:19.145  1.12161  1.12123     0.8400       0.75
4  2020.01.01 22:01:19.246  1.12161  1.12120     0.8400       0.75

                    DateT
0 2020-01-01 22:01:12.821
1 2020-01-01 22:01:17.176
2 2020-01-01 22:01:18.545
3 2020-01-01 22:01:19.145
4 2020-01-01 22:01:19.246
```

```
                    DateT
0 2020-01-01 22:01:12.821
1 2020-01-01 22:01:17.176
2 2020-01-01 22:01:18.545
3 2020-01-01 22:01:19.145
4 2020-01-01 22:01:19.246
                      Time (UTC)      Ask      Bid  AskVolume  BidVolume  \
1499995  2020.01.30 07:11:33.290  1.10087  1.10086       0.45       0.75
1499996  2020.01.30 07:11:33.392  1.10086  1.10083       0.75       4.57
1499997  2020.01.30 07:11:37.666  1.10086  1.10084       0.75       0.75
1499998  2020.01.30 07:11:37.768  1.10087  1.10085       2.44       1.50
1499999  2020.01.30 07:11:38.045  1.10086  1.10084       0.75       1.50

                           DateT
1499995 2020-01-30 07:11:33.290
1499996 2020-01-30 07:11:33.392
1499997 2020-01-30 07:11:37.666
1499998 2020-01-30 07:11:37.768
1499999 2020-01-30 07:11:38.045
```

B [24]:
```
# reformate date
data3['DateT'] = pd.to_datetime(data3['Time (UTC)'])

print(data3.dtypes)
print(data3.head(n=5))
print(data3[-5:])
```

```
Time (UTC)            object
Ask                  float32
Bid                  float32
AskVolume            float32
BidVolume            float32
DateT         datetime64[ns]
dtype: object
                      Time (UTC)      Ask      Bid  AskVolume  BidVolume  \
4500000  2020.03.10 17:18:51.089  1.13075  1.13070       0.94       1.20
4500001  2020.03.10 17:18:51.646  1.13073  1.13071       0.75       0.37
4500002  2020.03.10 17:18:51.697  1.13073  1.13069       0.75       0.56
4500003  2020.03.10 17:18:51.798  1.13074  1.13070       0.94       1.12
4500004  2020.03.10 17:18:51.899  1.13074  1.13071       0.19       1.12

                           DateT
4500000 2020-03-10 17:18:51.089
4500001 2020-03-10 17:18:51.646
4500002 2020-03-10 17:18:51.697
4500003 2020-03-10 17:18:51.798
4500004 2020-03-10 17:18:51.899
                      Time (UTC)      Ask      Bid  AskVolume  BidVolume  \
5999995  2020.03.17 19:37:31.855  1.10055  1.10050       0.19       0.82
5999996  2020.03.17 19:37:32.216  1.10055  1.10049       0.19       0.82
5999997  2020.03.17 19:37:33.192  1.10052  1.10047       0.19       1.50
5999998  2020.03.17 19:37:33.294  1.10048  1.10042       0.19       0.82
5999999  2020.03.17 19:37:33.426  1.10047  1.10041       0.56       0.75

                           DateT
5999995 2020-03-17 19:37:31.855
5999996 2020-03-17 19:37:32.216
5999997 2020-03-17 19:37:33.192
5999998 2020-03-17 19:37:33.294
5999999 2020-03-17 19:37:33.426
```

B [25]:
```
import datetime
```

B [26]:
```
n = data_raw.shape[0]

data_raw['DateDelta'] = data_raw['DateT'].astype('int64')

data_raw['DateDelta'] = data_raw['DateDelta'].div(1000000 , axis=0)
data_raw['DateDelta'] = data_raw['DateDelta'].astype('int64')
data_raw['DateDelta1'] = data_raw['DateDelta'].diff()


#  cleaning 'NaN'
data_raw['DateDelta1'].iloc[0] = 0

data_raw['DateDelta1'] = data_raw['DateDelta1'].astype(int)

print(data_raw.dtypes)
print(data_raw.head(n=5))
print(data_raw[-5:])
print(data_raw['DateDelta1'].iloc[:4])
```

```
Time (UTC)            object
Ask                  float64
Bid                  float64
AskVolume            float64
BidVolume            float64
DateT          datetime64[ns]
DateDelta              int64
DateDelta1             int32
dtype: object
                Time (UTC)      Ask      Bid  AskVolume  BidVolume  \
0  2020.01.01 22:01:12.821  1.12160  1.12106     0.0937       0.75
1  2020.01.01 22:01:17.176  1.12160  1.12120     0.0937       0.75
2  2020.01.01 22:01:18.545  1.12160  1.12117     0.0937       0.75
3  2020.01.01 22:01:19.145  1.12161  1.12123     0.8400       0.75
4  2020.01.01 22:01:19.246  1.12161  1.12120     0.8400       0.75

                    DateT      DateDelta  DateDelta1
0 2020-01-01 22:01:12.821  1577916072821           0
1 2020-01-01 22:01:17.176  1577916077176        4355
2 2020-01-01 22:01:18.545  1577916078545        1369
3 2020-01-01 22:01:19.145  1577916079145         600
4 2020-01-01 22:01:19.246  1577916079246         101
                         Time (UTC)      Ask      Bid  AskVolume  BidVolume  \
16586025  2020.06.30 23:58:40.816  1.12337  1.12333       0.75       5.06
16586026  2020.06.30 23:58:40.917  1.12338  1.12333       1.57       3.94
16586027  2020.06.30 23:58:47.486  1.12337  1.12333       0.75       3.94
16586028  2020.06.30 23:58:52.038  1.12338  1.12333       2.02       3.94
16586029  2020.06.30 23:58:55.919  1.12338  1.12334       1.50       0.37

                           DateT      DateDelta  DateDelta1
16586025 2020-06-30 23:58:40.816  1593561520816         101
16586026 2020-06-30 23:58:40.917  1593561520917         101
16586027 2020-06-30 23:58:47.486  1593561527486        6569
16586028 2020-06-30 23:58:52.038  1593561532038        4552
16586029 2020-06-30 23:58:55.919  1593561535919        3881
0       0
1    4355
2    1369
3     600
Name: DateDelta1, dtype: int32
```

B [27]:

```
n = data2.shape[0]

data2['DateDelta'] = data2['DateT'].astype('int64')

data2['DateDelta'] = data2['DateDelta'].div(1000000 , axis=0)
data2['DateDelta'] = data2['DateDelta'].astype('int64')
data2['DateDelta1'] = data2['DateDelta'].diff()

data2['DateDelta1'].iloc[0] = 0  #
data2['DateDelta1'] = data2['DateDelta1'].astype(int)

print(data2.dtypes)
print(data2.head(n=5))
print(data2[-5:])
print(data2['DateDelta1'].iloc[:4])
```

```
Time (UTC)            object
Ask                  float32
Bid                  float32
AskVolume            float32
BidVolume            float32
DateT          datetime64[ns]
DateDelta              int64
DateDelta1             int32
dtype: object
                Time (UTC)      Ask      Bid  AskVolume  BidVolume  \
0  2020.01.01 22:01:12.821  1.12160  1.12106     0.0937       0.75
1  2020.01.01 22:01:17.176  1.12160  1.12120     0.0937       0.75
2  2020.01.01 22:01:18.545  1.12160  1.12117     0.0937       0.75
3  2020.01.01 22:01:19.145  1.12161  1.12123     0.8400       0.75
4  2020.01.01 22:01:19.246  1.12161  1.12120     0.8400       0.75

                    DateT      DateDelta  DateDelta1
0 2020-01-01 22:01:12.821  1577916072821           0
1 2020-01-01 22:01:17.176  1577916077176        4355
2 2020-01-01 22:01:18.545  1577916078545        1369
3 2020-01-01 22:01:19.145  1577916079145         600
4 2020-01-01 22:01:19.246  1577916079246         101
```

```
                     Time (UTC)      Ask      Bid  AskVolume  BidVolume  \
1499995  2020.01.30 07:11:33.290  1.10087  1.10086       0.45       0.75
1499996  2020.01.30 07:11:33.392  1.10086  1.10083       0.75       4.57
1499997  2020.01.30 07:11:37.666  1.10086  1.10084       0.75       0.75
1499998  2020.01.30 07:11:37.768  1.10087  1.10085       2.44       1.50
1499999  2020.01.30 07:11:38.045  1.10086  1.10084       0.75       1.50

                          DateT      DateDelta  DateDelta1
1499995 2020-01-30 07:11:33.290  1580368293289         101
1499996 2020-01-30 07:11:33.392  1580368293392         103
1499997 2020-01-30 07:11:37.666  1580368297665        4273
1499998 2020-01-30 07:11:37.768  1580368297768         103
1499999 2020-01-30 07:11:38.045  1580368298045         277
0       0
1    4355
2    1369
3     600
Name: DateDelta1, dtype: int32
```

B [28]:

```
n = data3.shape[0]

data3['DateDelta'] = data3['DateT'].astype('int64')

data3['DateDelta'] = data3['DateDelta'].div(1000000 , axis=0)
data3['DateDelta'] = data3['DateDelta'].astype('int64')
data3['DateDelta1'] = data3['DateDelta'].diff()


data3['DateDelta1'].iloc[0] = 0
data3['DateDelta1'] = data3['DateDelta1'].astype(int)

print(data3.dtypes)
print(data3.head(n=5))
print(data3[-5:])
print(data3['DateDelta1'].iloc[:4])
```

```
Time (UTC)            object
Ask                  float32
Bid                  float32
AskVolume            float32
BidVolume            float32
DateT         datetime64[ns]
DateDelta             int64
DateDelta1            int32
dtype: object
                     Time (UTC)      Ask      Bid  AskVolume  BidVolume  \
4500000  2020.03.10 17:18:51.089  1.13075  1.13070       0.94       1.20
4500001  2020.03.10 17:18:51.646  1.13073  1.13071       0.75       0.37
4500002  2020.03.10 17:18:51.697  1.13073  1.13069       0.75       0.56
4500003  2020.03.10 17:18:51.798  1.13074  1.13070       0.94       1.12
4500004  2020.03.10 17:18:51.899  1.13074  1.13071       0.19       1.12

                          DateT      DateDelta  DateDelta1
4500000 2020-03-10 17:18:51.089  1583860731089           0
4500001 2020-03-10 17:18:51.646  1583860731646         557
4500002 2020-03-10 17:18:51.697  1583860731697          51
4500003 2020-03-10 17:18:51.798  1583860731798         101
4500004 2020-03-10 17:18:51.899  1583860731899         101
                     Time (UTC)      Ask      Bid  AskVolume  BidVolume  \
5999995  2020.03.17 19:37:31.855  1.10055  1.10050       0.19       0.82
5999996  2020.03.17 19:37:32.216  1.10055  1.10049       0.19       0.82
5999997  2020.03.17 19:37:33.192  1.10052  1.10047       0.19       1.50
5999998  2020.03.17 19:37:33.294  1.10048  1.10042       0.19       0.82
5999999  2020.03.17 19:37:33.426  1.10047  1.10041       0.56       0.75

                          DateT      DateDelta  DateDelta1
5999995 2020-03-17 19:37:31.855  1584473851855         152
5999996 2020-03-17 19:37:32.216  1584473852216         361
5999997 2020-03-17 19:37:33.192  1584473853192         976
5999998 2020-03-17 19:37:33.294  1584473853294         102
5999999 2020-03-17 19:37:33.426  1584473853425         131
4500000      0
4500001    557
4500002     51
4500003    101
Name: DateDelta1, dtype: int32
```

B [29]:
```
# Millisecond

mmx  = data_raw['DateDelta1'].max()
mmn  = data_raw['DateDelta1'].min()
print(mmx, ' ', mmn)

mmx  = data_raw['DateDelta'].max()
mmn  = data_raw['DateDelta'].min()
print(mmx, ' ', mmn)
```

```
173188984   0
1593561535919   1577916072821
```

B [30]:
```
mmx  = data2['DateDelta1'].max()
mmn  = data2['DateDelta1'].min()
print(mmx, ' ', mmn)

mmx  = data2['DateDelta'].max()
mmn  = data2['DateDelta'].min()
print(mmx, ' ', mmn)
```

```
172860715   0
1580368298045   1577916072821
```

B [31]:
```
mmx  = data3['DateDelta1'].max()
mmn  = data3['DateDelta1'].min()
print(mmx, ' ', mmn)

mmx  = data3['DateDelta'].max()
mmn  = data3['DateDelta'].min()
print(mmx, ' ', mmn)
```

```
172875221   0
1584473853425   1583860731089
```

B [32]:
```
# Find weekends
# 2 days * 24 hours * 60 minutes * 60 sec = 172.800 seconds = 172.800.000 milliseconds

#  DateDelta1 > 100000 milliseconds (pauses)
data2[data2.DateDelta1 > 100000]
```

Out[32]:

	Time (UTC)	Ask	Bid	AskVolume	BidVolume	DateT	DateDelta	DateDelta1
80111	2020.01.02 22:22:47.339	1.11736	1.11710	1.1200	1.92	2020-01-02 22:22:47.339	1578003767339	185349
168840	2020.01.05 22:00:22.039	1.11687	1.11656	0.7500	0.75	2020-01-05 22:00:22.039	1578261622039	172826531
169150	2020.01.05 22:38:29.241	1.11622	1.11605	0.7500	0.75	2020-01-05 22:38:29.241	1578263909241	102757
169251	2020.01.05 22:55:32.854	1.11620	1.11602	0.7500	0.75	2020-01-05 22:55:32.854	1578264932854	132969
246129	2020.01.06 22:26:20.727	1.11962	1.11960	0.1000	0.89	2020-01-06 22:26:20.727	1578349580727	105034
445476	2020.01.08 21:37:24.104	1.11093	1.11089	1.5000	2.70	2020-01-08 21:37:24.104	1578519444104	116648
596235	2020.01.12 22:00:03.473	1.11183	1.11141	0.7500	0.75	2020-01-12 22:00:03.473	1578866403473	172860715
937913	2020.01.19 22:00:39.388	1.10940	1.10910	1.5000	0.75	2020-01-19 22:00:39.388	1579471239388	172846669
938167	2020.01.19 22:17:05.091	1.10920	1.10911	0.0187	0.75	2020-01-19 22:17:05.091	1579472225091	107093
986400	2020.01.20 22:04:04.062	1.10951	1.10947	0.5600	4.00	2020-01-20 22:04:04.062	1579557844062	109938
989182	2020.01.21 00:51:51.657	1.10928	1.10925	3.0000	0.75	2020-01-21 00:51:51.657	1579567911657	111787
1276585	2020.01.26 22:00:06.713	1.10355	1.10318	0.3700	0.75	2020-01-26 22:00:06.713	1580076006713	172810298
1361733	2020.01.28 03:24:02.593	1.10203	1.10199	0.8200	0.75	2020-01-28 03:24:02.593	1580181842593	128380

B [33]:
```
data3[data3.DateDelta1 > 100000]
```

Out[33]:

	Time (UTC)	Ask	Bid	AskVolume	BidVolume	DateT	DateDelta	DateDelta1
5319481	2020.03.15 21:00:37.540	1.10779	1.10705	0.75	0.75	2020-03-15 21:00:37.540	1584306037540	172875221
5703021	2020.03.16 21:10:59.443	1.11764	1.11761	0.75	0.37	2020-03-16 21:10:59.443	1584393059443	102046

B [34]:
```
data_raw[data_raw.DateDelta1 > 160000000]
```

Out[34]:

	Time (UTC)	Ask	Bid	AskVolume	BidVolume	DateT	DateDelta	DateDelta1
168840	2020.01.05 22:00:22.039	1.11687	1.11656	0.75	0.75	2020-01-05 22:00:22.039	1578261622039	172826531
596235	2020.01.12 22:00:03.473	1.11183	1.11141	0.75	0.75	2020-01-12 22:00:03.473	1578866403473	172860715
937913	2020.01.19 22:00:39.388	1.10940	1.10910	1.50	0.75	2020-01-19 22:00:39.388	1579471239388	172846669
1276585	2020.01.26 22:00:06.713	1.10355	1.10318	0.37	0.75	2020-01-26 22:00:06.713	1580076006713	172810298
1637760	2020.02.02 22:00:14.503	1.10957	1.10939	0.75	0.75	2020-02-02 22:00:14.503	1580680814503	172818305
1951341	2020.02.09 22:00:03.235	1.09479	1.09439	1.31	0.75	2020-02-09 22:00:03.235	1581285603235	172806909
2262899	2020.02.16 22:00:01.861	1.08428	1.08406	0.75	0.56	2020-02-16 22:00:01.861	1581890401861	172805780
2581015	2020.02.23 22:00:02.104	1.08163	1.08145	1.31	0.75	2020-02-23 22:00:02.104	1582495202104	172805815
3199857	2020.03.01 22:00:01.729	1.10482	1.10423	0.75	1.31	2020-03-01 22:00:01.729	1583100001729	172805437
3976886	2020.03.08 21:00:04.528	1.13538	1.13473	0.75	0.75	2020-03-08 21:00:04.528	1583701204528	169207847
5319481	2020.03.15 21:00:37.540	1.10779	1.10705	0.75	0.75	2020-03-15 21:00:37.540	1584306037540	172875221
7131362	2020.03.22 21:06:00.261	1.07144	1.07011	0.75	0.75	2020-03-22 21:06:00.261	1584911160261	173188984
8515962	2020.03.29 21:01:27.315	1.11249	1.11172	0.37	0.75	2020-03-29 21:01:27.315	1585515687315	172890941
9389506	2020.04.05 21:00:04.357	1.08168	1.08082	0.75	0.75	2020-04-05 21:00:04.357	1586120404357	172812997
10029255	2020.04.12 21:00:05.792	1.09499	1.09482	0.75	0.75	2020-04-12 21:00:05.792	1586725205792	172897883
10619973	2020.04.19 21:05:25.566	1.08751	1.08748	0.19	0.75	2020-04-19 21:05:25.566	1587330325566	173129722
11177560	2020.04.26 21:00:07.116	1.08242	1.08211	0.75	0.75	2020-04-26 21:00:07.116	1587934807116	172810728
11693053	2020.05.03 21:00:01.382	1.09755	1.09676	0.75	0.75	2020-05-03 21:00:01.382	1588539601382	172805995
12181692	2020.05.10 21:00:02.339	1.08287	1.08265	0.75	0.75	2020-05-10 21:00:02.339	1589144402339	172807778
12661526	2020.05.17 21:02:21.339	1.08155	1.08088	0.75	0.75	2020-05-17 21:02:21.339	1589749341339	172945414
13096424	2020.05.24 21:00:04.066	1.08973	1.08956	0.75	0.75	2020-05-24 21:00:04.066	1590354004065	172807800
13639145	2020.05.31 21:00:04.281	1.11213	1.11129	0.75	0.75	2020-05-31 21:00:04.281	1590958804281	172808520
14312617	2020.06.07 21:00:04.814	1.12961	1.12934	0.75	1.50	2020-06-07 21:00:04.814	1591563604814	172808538
15047417	2020.06.14 21:00:02.193	1.12353	1.12329	0.75	0.75	2020-06-14 21:00:02.193	1592168402193	172805757
15776539	2020.06.21 21:00:01.397	1.11900	1.11835	0.75	0.75	2020-06-21 21:00:01.397	1592773201397	172804944
16376109	2020.06.28 21:01:03.322	1.12221	1.12149	0.75	0.75	2020-06-28 21:01:03.322	1593378063321	172867082

B [36]:

```
data_raw_weekends = data_raw[data_raw.DateDelta1 > 1600000]


columns_1 = ['Time (UTC)', 'DateDelta']
data_raw_weekends.drop(columns_1,inplace=True, axis=1)
data_raw_weekends.info()
data_raw_weekends
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26 entries, 168840 to 16376109
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Ask         26 non-null     float64
 1   Bid         26 non-null     float64
 2   AskVolume   26 non-null     float64
 3   BidVolume   26 non-null     float64
 4   DateT       26 non-null     datetime64[ns]
 5   DateDelta1  26 non-null     int32
dtypes: datetime64[ns](1), float64(4), int32(1)
memory usage: 1.3 KB
```

Out[36]:

	Ask	Bid	AskVolume	BidVolume	DateT	DateDelta1
168840	1.11687	1.11656	0.75	0.75	2020-01-05 22:00:22.039	172826531
596235	1.11183	1.11141	0.75	0.75	2020-01-12 22:00:03.473	172860715
937913	1.10940	1.10910	1.50	0.75	2020-01-19 22:00:39.388	172846669
1276585	1.10355	1.10318	0.37	0.75	2020-01-26 22:00:06.713	172810298
1637760	1.10957	1.10939	0.75	0.75	2020-02-02 22:00:14.503	172818305
1951341	1.09479	1.09439	1.31	0.75	2020-02-09 22:00:03.235	172806909
2262899	1.08428	1.08406	0.75	0.56	2020-02-16 22:00:01.861	172805780
2581015	1.08163	1.08145	1.31	0.75	2020-02-23 22:00:02.104	172805815
3199857	1.10482	1.10423	0.75	1.31	2020-03-01 22:00:01.729	172805437
3976886	1.13538	1.13473	0.75	0.75	2020-03-08 21:00:04.528	169207847
5319481	1.10779	1.10705	0.75	0.75	2020-03-15 21:00:37.540	172875221
7131362	1.07144	1.07011	0.75	0.75	2020-03-22 21:06:00.261	173188984
8515962	1.11249	1.11172	0.37	0.75	2020-03-29 21:01:27.315	172890941
9389506	1.08168	1.08082	0.75	0.75	2020-04-05 21:00:04.357	172812997
10029255	1.09499	1.09482	0.75	0.75	2020-04-12 21:00:05.792	172897883
10619973	1.08751	1.08748	0.19	0.75	2020-04-19 21:05:25.566	173129722
11177560	1.08242	1.08211	0.75	0.75	2020-04-26 21:00:07.116	172810728
11693053	1.09755	1.09676	0.75	0.75	2020-05-03 21:00:01.382	172805995
12181692	1.08287	1.08265	0.75	0.75	2020-05-10 21:00:02.339	172807778
12661526	1.08155	1.08088	0.75	0.75	2020-05-17 21:02:21.339	172945414
13096424	1.08973	1.08956	0.75	0.75	2020-05-24 21:00:04.066	172807800
13639145	1.11213	1.11129	0.75	0.75	2020-05-31 21:00:04.281	172808520
14312617	1.12961	1.12934	0.75	1.50	2020-06-07 21:00:04.814	172808538
15047417	1.12353	1.12329	0.75	0.75	2020-06-14 21:00:02.193	172805757
15776539	1.11900	1.11835	0.75	0.75	2020-06-21 21:00:01.397	172804944
16376109	1.12221	1.12149	0.75	0.75	2020-06-28 21:01:03.322	172867082

B [37]:

```
data2_weekends = data2[data2.DateDelta1 > 1600000]

# dropping
columns_1 = ['Time (UTC)', 'DateDelta']
data2_weekends.drop(columns_1,inplace=True, axis=1)
data2_weekends.info()
data2_weekends
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4 entries, 168840 to 1276585
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Ask         4 non-null      float32
 1   Bid         4 non-null      float32
 2   AskVolume   4 non-null      float32
 3   BidVolume   4 non-null      float32
 4   DateT       4 non-null      datetime64[ns]
 5   DateDelta1  4 non-null      int32
dtypes: datetime64[ns](1), float32(4), int32(1)
memory usage: 144.0 bytes
```

Out[37]:

	Ask	Bid	AskVolume	BidVolume	DateT	DateDelta1
168840	1.11687	1.11656	0.75	0.75	2020-01-05 22:00:22.039	172826531
596235	1.11183	1.11141	0.75	0.75	2020-01-12 22:00:03.473	172860715
937913	1.10940	1.10910	1.50	0.75	2020-01-19 22:00:39.388	172846669
1276585	1.10355	1.10318	0.37	0.75	2020-01-26 22:00:06.713	172810298

B [38]:

```
# new week
data3_weekends = data3[data3.DateDelta1 > 1600000]

columns_1 = ['Time (UTC)', 'DateDelta']
data3_weekends.drop(columns_1,inplace=True, axis=1)
data3_weekends.info()
data3_weekends
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1 entries, 5319481 to 5319481
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Ask         1 non-null      float32
 1   Bid         1 non-null      float32
 2   AskVolume   1 non-null      float32
 3   BidVolume   1 non-null      float32
 4   DateT       1 non-null      datetime64[ns]
 5   DateDelta1  1 non-null      int32
dtypes: datetime64[ns](1), float32(4), int32(1)
memory usage: 36.0 bytes
```

Out[38]:

	Ask	Bid	AskVolume	BidVolume	DateT	DateDelta1
5319481	1.10779	1.10705	0.75	0.75	2020-03-15 21:00:37.540	172875221

B [39]:

```
data_raw_week_start_index = data_raw_weekends.index.tolist()
data_raw_week_start_index = np.array(data_raw_week_start_index)

print("data_raw_week_start_index:")
print(type(data_raw_week_start_index))
print(data_raw_week_start_index.shape)
# print('\n')
print(data_raw_week_start_index)
```

```
data_raw_week_start_index:
<class 'numpy.ndarray'>
(26,)
[  168840   596235   937913  1276585  1637760  1951341  2262899  2581015
  3199857  3976886  5319481  7131362  8515962  9389506 10029255 10619973
 11177560 11693053 12181692 12661526 13096424 13639145 14312617 15047417
 15776539 16376109]
```

B [40]:

```
data2_week_start_index = data2_weekends.index.tolist()
data2_week_start_index = np.array(data2_week_start_index)

print("data2_week_start_index:")
print(type(data2_week_start_index))
print(data2_week_start_index.shape)
# print('\n')
print(data2_week_start_index)
```

```
data2_week_start_index:
<class 'numpy.ndarray'>
(4,)
[ 168840  596235  937913 1276585]
```

B [41]:

```
data3_week_start_index = data3_weekends.index.tolist()
data3_week_start_index = np.array(data3_week_start_index)

print("data3_week_start_index:")
print(type(data3_week_start_index))
print(data3_week_start_index.shape)
print(data3_week_start_index)
```

```
data3_week_start_index:
<class 'numpy.ndarray'>
(1,)
[5319481]
```

B [42]:

```
data_Weekly_Ask = pd.read_csv('Data/EURUSD_Weekly_Ask_2019.12.29_2020.06.28.csv', low_memory=False, sep=',')
data_Weekly_Bid = pd.read_csv('Data/EURUSD_Weekly_Bid_2019.12.29_2020.06.28.csv', low_memory=False, sep=',')

print(data_Weekly_Ask)
print('/n')
print(data_Weekly_Bid)
```

	Time (UTC)	Open	High	Low	Close	Volume
0	2019.12.29 22:00:00	1.11820	1.12393	1.11251	1.11626	7.375537e+05
1	2020.01.05 22:00:00	1.11687	1.12058	1.10852	1.11256	7.927178e+05
2	2020.01.12 22:00:00	1.11183	1.11728	1.10863	1.10920	6.695170e+05
3	2020.01.19 22:00:00	1.10940	1.11181	1.10197	1.10280	6.260430e+05
4	2020.01.26 22:00:00	1.10355	1.10961	1.09924	1.10952	6.641088e+05
5	2020.02.02 22:00:00	1.10957	1.10957	1.09422	1.09476	6.006794e+05
6	2020.02.09 22:00:00	1.09479	1.09578	1.08275	1.08331	1.742815e+06
7	2020.02.16 22:00:00	1.08428	1.08637	1.07782	1.08482	6.944203e+05
8	2020.02.23 22:00:00	1.08163	1.10532	1.08054	1.10289	1.006607e+06
9	2020.03.01 22:00:00	1.10482	1.13549	1.10365	1.12867	1.306598e+06
10	2020.03.08 21:00:00	1.13538	1.14961	1.10551	1.11004	1.248912e+06
11	2020.03.15 21:00:00	1.10779	1.12368	1.06377	1.07018	1.585563e+06
12	2020.03.22 21:00:00	1.07144	1.11479	1.06364	1.11457	1.592586e+06
13	2020.03.29 21:00:00	1.11249	1.11443	1.07729	1.07996	1.278563e+06
14	2020.04.05 21:00:00	1.08168	1.09532	1.07686	1.09419	1.145022e+06
15	2020.04.12 21:00:00	1.09499	1.09920	1.08123	1.08786	1.156205e+06
16	2020.04.19 21:00:00	1.08751	1.08969	1.07273	1.08245	1.174046e+06
17	2020.04.26 21:00:00	1.08242	1.10190	1.08099	1.09829	1.114037e+06
18	2020.05.03 21:00:00	1.09755	1.09759	1.07667	1.08419	1.052173e+06
19	2020.05.10 21:00:00	1.08287	1.08967	1.07751	1.08198	7.713983e+05
20	2020.05.17 21:00:00	1.08155	1.10086	1.08001	1.09063	1.054548e+06
21	2020.05.24 21:00:00	1.08973	1.11452	1.08707	1.11071	1.304692e+06
22	2020.05.31 21:00:00	1.11213	1.13840	1.11007	1.12920	1.394424e+06
23	2020.06.07 21:00:00	1.12961	1.14227	1.12127	1.12564	1.327001e+06
24	2020.06.14 21:00:00	1.12353	1.13536	1.11683	1.11788	1.265643e+06
25	2020.06.21 21:00:00	1.11900	1.13487	1.11687	1.12204	1.048285e+06
26	2020.06.28 21:00:00	1.12221	1.13030	1.11850	1.12510	9.104226e+05

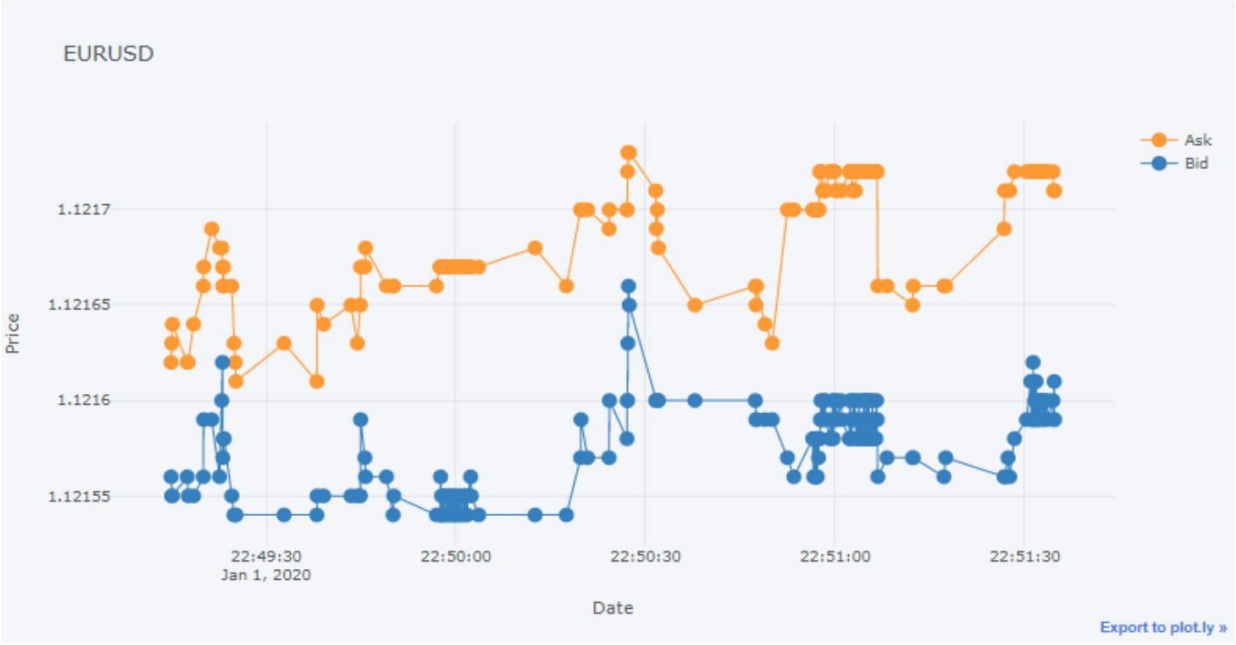
/n

	Time (UTC)	Open	High	Low	Close	Volume
0	2019.12.29 22:00:00	1.11736	1.12391	1.11249	1.11548	9.090149e+05
1	2020.01.05 22:00:00	1.11656	1.12055	1.10850	1.11165	7.648469e+05
2	2020.01.12 22:00:00	1.11141	1.11727	1.10862	1.10890	6.668899e+05
3	2020.01.19 22:00:00	1.10910	1.11179	1.10195	1.10224	6.625243e+05
4	2020.01.26 22:00:00	1.10318	1.10954	1.09923	1.10944	6.678325e+05
5	2020.02.02 22:00:00	1.10939	1.10939	1.09419	1.09438	6.070288e+05
6	2020.02.09 22:00:00	1.09439	1.09574	1.08271	1.08299	9.108856e+05
7	2020.02.16 22:00:00	1.08406	1.08635	1.07778	1.08456	6.567202e+05
8	2020.02.23 22:00:00	1.08145	1.10530	1.08050	1.10251	1.018095e+06
9	2020.03.01 22:00:00	1.10423	1.13546	1.10362	1.12804	1.064452e+06
10	2020.03.08 21:00:00	1.13473	1.14954	1.10550	1.10992	1.263062e+06
11	2020.03.15 21:00:00	1.10705	1.12366	1.06375	1.06929	1.680000e+06
12	2020.03.22 21:00:00	1.07011	1.11473	1.06357	1.11379	1.650858e+06
13	2020.03.29 21:00:00	1.11172	1.11440	1.07727	1.07955	1.350786e+06
14	2020.04.05 21:00:00	1.08082	1.09524	1.07683	1.09326	1.217516e+06
15	2020.04.12 21:00:00	1.09482	1.09907	1.08119	1.08711	1.171811e+06
16	2020.04.19 21:00:00	1.08748	1.08967	1.07271	1.08196	1.207161e+06
17	2020.04.26 21:00:00	1.08211	1.10187	1.08097	1.09777	1.117642e+06
18	2020.05.03 21:00:00	1.09676	1.09736	1.07665	1.08371	1.098800e+06
19	2020.05.10 21:00:00	1.08265	1.08965	1.07748	1.08183	1.070373e+06
20	2020.05.17 21:00:00	1.08088	1.10083	1.07999	1.08980	1.002680e+06
21	2020.05.24 21:00:00	1.08956	1.11451	1.08706	1.11036	1.234105e+06
22	2020.05.31 21:00:00	1.11129	1.13836	1.11004	1.12911	1.307916e+06
23	2020.06.07 21:00:00	1.12934	1.14224	1.12125	1.12550	1.298828e+06
24	2020.06.14 21:00:00	1.12329	1.13532	1.11680	1.11759	1.252222e+06
25	2020.06.21 21:00:00	1.11835	1.13486	1.11684	1.12170	1.059157e+06
26	2020.06.28 21:00:00	1.12149	1.13026	1.11848	1.12464	8.938301e+05

Interactive Graph

B [43]:
```
cf.set_config_file(world_readable=True, theme='pearl', offline=True)
```

B [44]:
```
data_raw.iloc[1000:1200,:][['Ask', 'Bid', 'DateT', 'AskVolume','BidVolume']].iplot(
                                                x='DateT', y=['Ask', 'Bid'],
                                                mode='lines+markers',
                                                # mode='lines+markers+text',
                                                xTitle='Date', yTitle='Price',
                                                text='AskVolume',
                                                #text=['AskVolume','BidVolume'],
                                                title='EURUSD  ')
```



B [ ]:

Margin

B [45]:

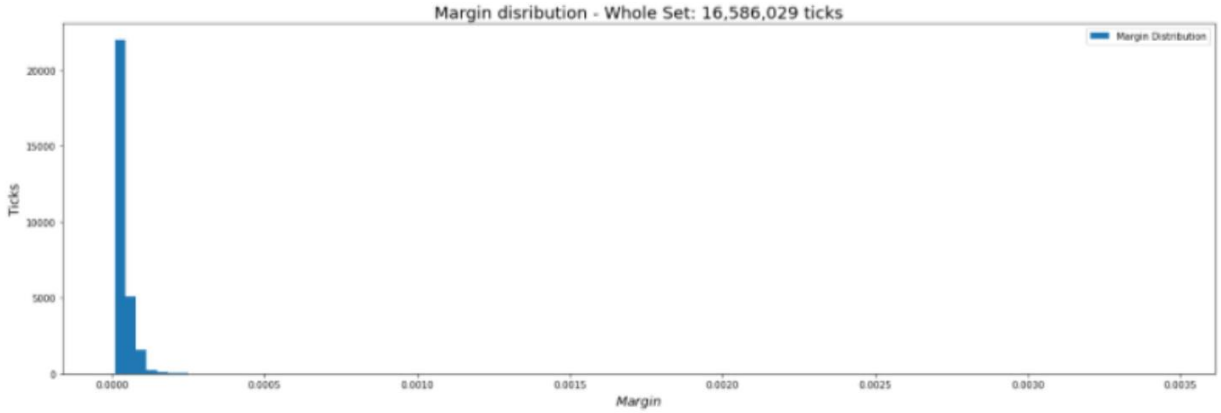
```
data_raw['Margin'] = data_raw['Ask'] - data_raw['Bid']
print(data_raw.dtypes)
print(data_raw.head(n=5))
print(data_raw[-5:])

print(data_raw['Margin'].iloc[:4])
```

```
Time (UTC)            object
Ask                  float64
Bid                  float64
AskVolume            float64
BidVolume            float64
DateT         datetime64[ns]
DateDelta              int64
DateDelta1             int32
Margin               float64
dtype: object
              Time (UTC)      Ask      Bid  AskVolume  BidVolume  \
0  2020.01.01 22:01:12.821  1.12160  1.12106     0.0937       0.75
1  2020.01.01 22:01:17.176  1.12160  1.12120     0.0937       0.75
2  2020.01.01 22:01:18.545  1.12160  1.12117     0.0937       0.75
3  2020.01.01 22:01:19.145  1.12161  1.12123     0.8400       0.75
4  2020.01.01 22:01:19.246  1.12161  1.12120     0.8400       0.75

                   DateT      DateDelta  DateDelta1   Margin
0 2020-01-01 22:01:12.821  1577916072821           0  0.00054
1 2020-01-01 22:01:17.176  1577916077176        4355  0.00040
2 2020-01-01 22:01:18.545  1577916078545        1369  0.00043
3 2020-01-01 22:01:19.145  1577916079145         600  0.00038
4 2020-01-01 22:01:19.246  1577916079246         101  0.00041
                  Time (UTC)      Ask      Bid  AskVolume  BidVolume  \
16586025  2020.06.30 23:58:40.816  1.12337  1.12333       0.75       5.06
16586026  2020.06.30 23:58:40.917  1.12338  1.12333       1.57       3.94
16586027  2020.06.30 23:58:47.486  1.12337  1.12333       0.75       3.94
16586028  2020.06.30 23:58:52.038  1.12338  1.12333       2.02       3.94
16586029  2020.06.30 23:58:55.919  1.12338  1.12334       1.50       0.37

                           DateT      DateDelta  DateDelta1   Margin
16586025 2020-06-30 23:58:40.816  1593561520816         101  0.00004
16586026 2020-06-30 23:58:40.917  1593561520917         101  0.00005
16586027 2020-06-30 23:58:47.486  1593561527486        6569  0.00004
16586028 2020-06-30 23:58:52.038  1593561532038        4552  0.00005
16586029 2020-06-30 23:58:55.919  1593561535919        3881  0.00004
0    0.00054
1    0.00040
2    0.00043
3    0.00038
Name: Margin, dtype: float64
```

B [46]:
```
data_raw_Margin = np.array(data_raw['Margin'])

data_raw_Margin = data_raw_Margin.reshape(data_raw_Margin.shape[0])

print(type(data_raw_Margin[0]))
print(data_raw_Margin[:5])
print(data_raw_Margin[-5:])
print(data_raw_Margin.shape)
```

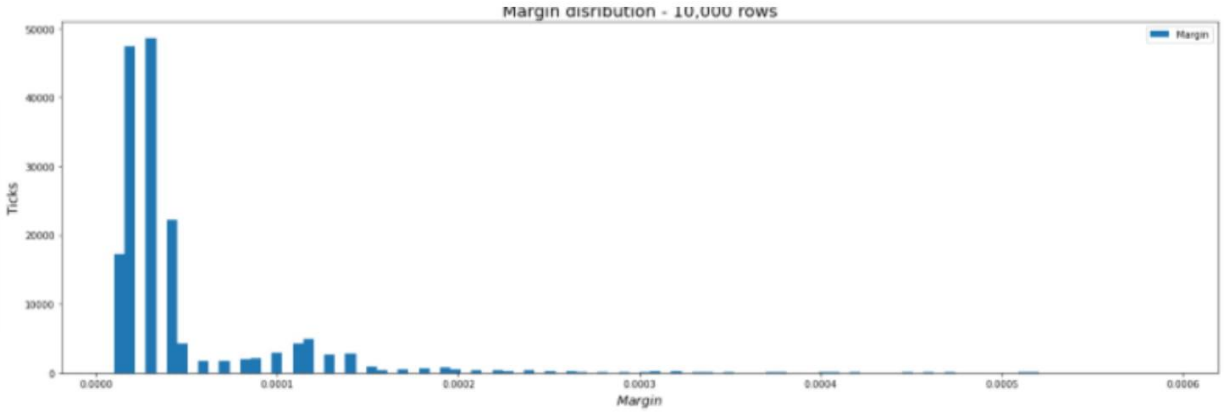
```
<class 'numpy.float64'>
[0.00054 0.0004  0.00043 0.00038 0.00041]
[4.e-05 5.e-05 4.e-05 5.e-05 4.e-05]
(16586030,)
```

B [47]:
```
# Margin distribution

plt.figure(num=1, figsize=(22,7))
plt.hist(data_raw_Margin, bins=100, label='Margin Distribution', density=True)


plt.title('Margin disribution - Whole Set: 16,586,029 ticks', size=18)
plt.ylabel('Ticks', size=14)
plt.xlabel('$Margin$', size=14)
plt.legend(loc='upper right')

plt.savefig('Figures/MarginDistribution-aii_ticks-2020-6-months.png')
```

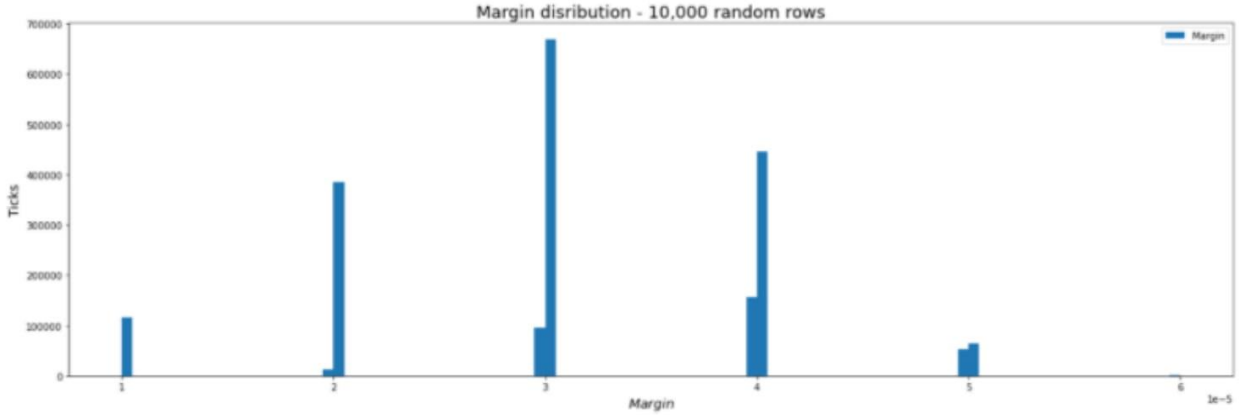


B [ ]:
```
```

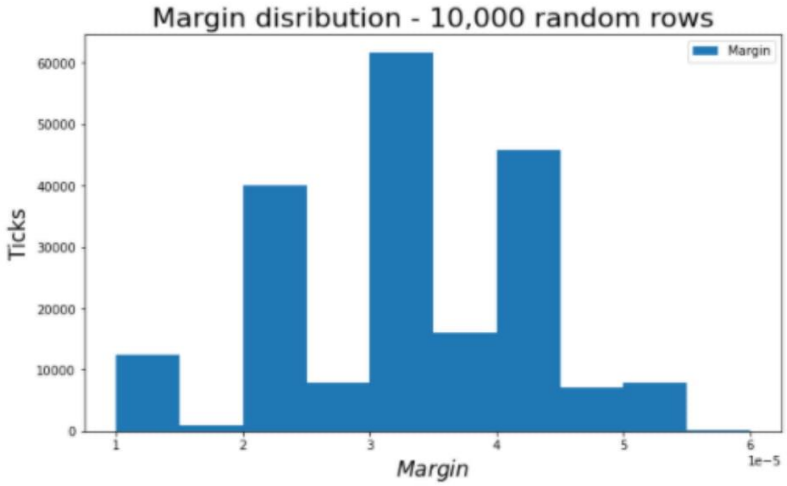
B [48]:
```
# The first 10.000 rows

NN = 10000

indexes_to_drop = np.array(range(NN,data_raw.shape[0]))
indexes_to_keep = set(range(data_raw_Margin.shape[0])) - set(indexes_to_drop)
data_raw_Margin_hist = data_raw['Margin'].take(list(indexes_to_keep))

data_raw_Margin_hist_array = np.array(data_raw_Margin_hist)
data_raw_Margin_hist_array = data_raw_Margin_hist_array.reshape(NN)

plt.figure(num=1, figsize=(22,7))
plt.hist(data_raw_Margin_hist_array, bins=100, label='Margin', density=True)


plt.title('Margin disribution - 10,000 rows', size=18)
plt.ylabel('Ticks', size=14)
plt.xlabel('$Margin$', size=14)
plt.legend(loc='upper right')

plt.savefig('Figures/MarginDistribution-10000_rows-2020.png')
```



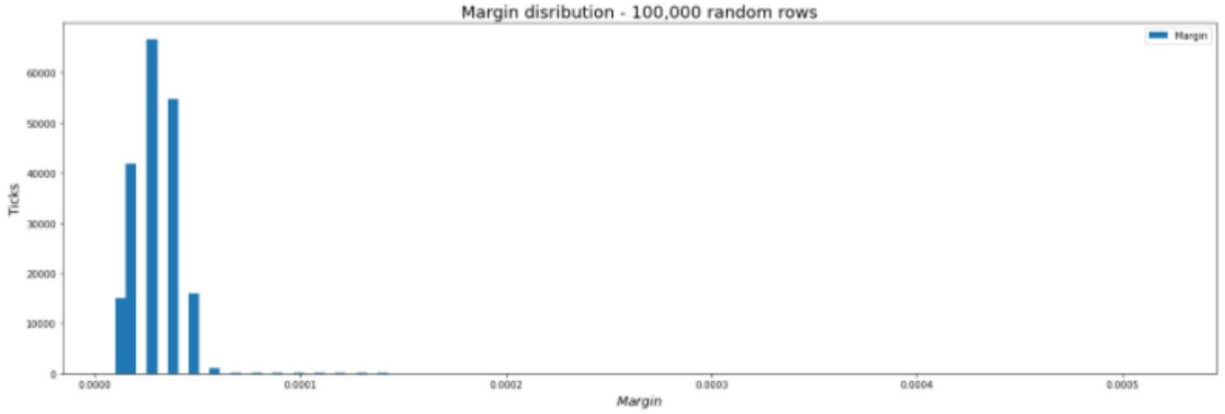
B [49]:
```
# 10.000 rows somewhere in the middle
import random

n_rand = random.randint(0, data_raw_Margin.shape[0] - 10000)

indexes_to_drop1 = np.array(range(0,n_rand))
indexes_to_drop2 = np.array(range(n_rand+10000,data_raw_Margin.shape[0]))
indexes_to_keep = set(range(data_raw_Margin.shape[0])) - set(indexes_to_drop1) - set(indexes_to_drop2)
data_raw_Margin_hist = data_raw['Margin'].take(list(indexes_to_keep))

data_raw_Margin_hist_array = np.array(data_raw_Margin_hist)
data_raw_Margin_hist_array = data_raw_Margin_hist_array.reshape(10000)

plt.figure(num=1, figsize=(22,7))
plt.hist(data_raw_Margin_hist_array, bins=100, label='Margin', density=True)

plt.title('Margin disribution - 10,000 random rows', size=18)
plt.ylabel('Ticks', size=14)
plt.xlabel('$Margin$', size=14)
plt.legend(loc='upper right') #

plt.savefig('Figures/MarginDistribution-10000_random_rows-2020.png')
```



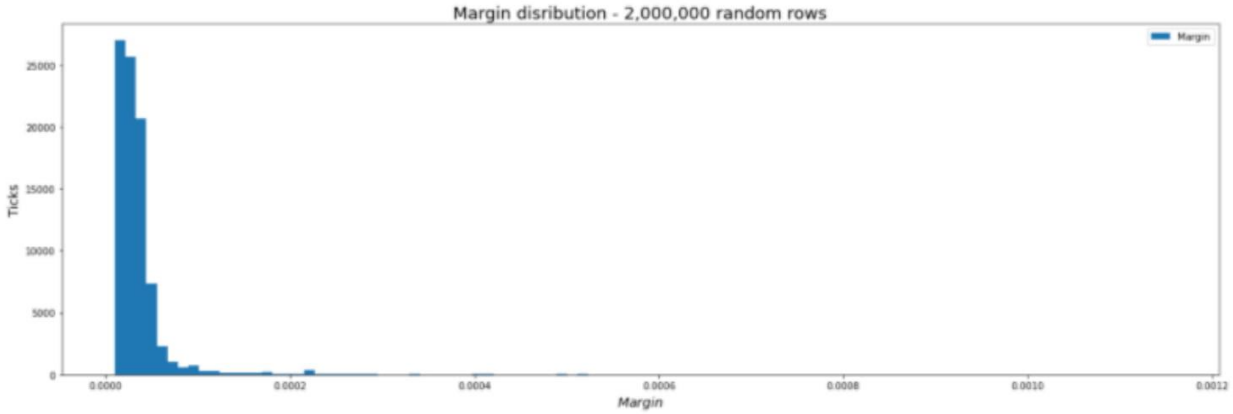
B [50]:
```
# 10.000 rows somewhere in the middle

import random

n_rand = random.randint(0, data_raw_Margin.shape[0] - 10000)

indexes_to_drop1 = np.array(range(0,n_rand))
indexes_to_drop2 = np.array(range(n_rand+10000,data_raw_Margin.shape[0]))
indexes_to_keep = set(range(data_raw_Margin.shape[0])) - set(indexes_to_drop1) - set(indexes_to_drop2)
data_raw_Margin_hist = data_raw['Margin'].take(list(indexes_to_keep))

data_raw_Margin_hist_array = np.array(data_raw_Margin_hist)
data_raw_Margin_hist_array = data_raw_Margin_hist_array.reshape(10000)


plt.figure(num=1, figsize=(10,6))
plt.hist(data_raw_Margin_hist_array, bins=10, label='Margin', density=True)

plt.title('Margin disribution - 10,000 random rows', size=22)
plt.ylabel('Ticks', size=17)
plt.xlabel('$Margin$', size=17)
plt.legend(loc='upper right')

plt.savefig('Figures/MarginDistribution-10000-2_random_rows-2020.png')
```



B [51]:
```
# 100.000 rows somewhere in the middle

import random

n_rand = random.randint(0, data_raw_Margin.shape[0] - 100000)

indexes_to_drop1 = np.array(range(0,n_rand))
indexes_to_drop2 = np.array(range(n_rand+100000,data_raw_Margin.shape[0]))
indexes_to_keep = set(range(data_raw_Margin.shape[0])) - set(indexes_to_drop1) - set(indexes_to_drop2)
data_raw_Margin_hist = data_raw['Margin'].take(list(indexes_to_keep))

data_raw_Margin_hist_array = np.array(data_raw_Margin_hist)
data_raw_Margin_hist_array = data_raw_Margin_hist_array.reshape(100000)

plt.figure(num=1, figsize=(22,7))
plt.hist(data_raw_Margin_hist_array, bins=100, label='Margin', density=True)

plt.title('Margin disribution - 100,000 random rows', size=18)
plt.ylabel('Ticks', size=14)
plt.xlabel('$Margin$', size=14)
plt.legend(loc='upper right')

plt.savefig('Figures/MarginDistribution-100000-1_random_rows-2020.png')
```



B [52]:
```
# 2.000.000 rows somewhere in the middle

import random

n_rand = random.randint(0, data_raw_Margin.shape[0] - 2000000)

indexes_to_drop1 = np.array(range(0,n_rand))
indexes_to_drop2 = np.array(range(n_rand+2000000,data_raw_Margin.shape[0]))
indexes_to_keep = set(range(data_raw_Margin.shape[0])) - set(indexes_to_drop1) - set(indexes_to_drop2)
data_raw_Margin_hist = data_raw['Margin'].take(list(indexes_to_keep))

data_raw_Margin_hist_array = np.array(data_raw_Margin_hist)
data_raw_Margin_hist_array = data_raw_Margin_hist_array.reshape(2000000)

plt.figure(num=1, figsize=(22,7))
plt.hist(data_raw_Margin_hist_array, bins=100, label='Margin', density=True)

plt.title('Margin disribution - 2,000,000 random rows', size=18)
plt.ylabel('Ticks', size=14)
plt.xlabel('$Margin$', size=14)
plt.legend(loc='upper right')

plt.savefig('Figures/MarginDistribution-2000000_random_rows-2020.png')
```



B [ ]:

Days data

B [54]:
```
data_raw_days_Ask = pd.read_csv('Data/EURUSD_Daily_Ask_2019.12.31_2020.06.30.csv', low_memory=False, sep=',')
data_raw_days_Bid = pd.read_csv('Data/EURUSD_Daily_Bid_2019.12.31_2020.06.30.csv', low_memory=False, sep=',')


print(data_raw_days_Ask.head(n=5))
print(data_raw_days_Bid[-5:])
```

```
            Time (UTC)     Open     High      Low    Close      Volume
0  2020.01.01 22:00:00  1.12158  1.12247  1.11638  1.11729  282243.1150
1  2020.01.02 22:00:00  1.11728  1.11800  1.11251  1.11626  224433.1250
2  2020.01.05 22:00:00  1.11687  1.12058  1.11574  1.11977  138759.6694
3  2020.01.06 22:00:00  1.11980  1.11980  1.11337  1.11543  144814.1100
4  2020.01.07 22:00:00  1.11543  1.11685  1.11018  1.11060  198430.8104
              Time (UTC)     Open     High      Low    Close      Volume
125  2020.06.24 21:00:00  1.12503  1.12597  1.11903  1.12170  234684.1200
126  2020.06.25 21:00:00  1.12181  1.12393  1.11951  1.12170  175079.0520
127  2020.06.28 21:00:00  1.12149  1.12879  1.12149  1.12410  179170.7479
128  2020.06.29 21:00:00  1.12412  1.12619  1.11910  1.12336  196959.7740
129  2020.06.30 21:00:00  1.12340  1.12751  1.11848  1.12507  186639.9400
```

B [55]:
```
data_raw_days_Ask_2019 = pd.read_csv('Data/2019/EURUSD_Daily_Ask_2018.12.31_2019.12.31.csv', low_memory=False, sep=',')
data_raw_days_Bid_2019 = pd.read_csv('Data/2019/EURUSD_Daily_Bid_2018.12.31_2019.12.31.csv', low_memory=False, sep=',')


print(data_raw_days_Ask_2019.head(n=5))
print(data_raw_days_Bid_2019[-5:])
```

```
            Time (UTC)     Open     High      Low    Close    Volume
0  2019.01.01 22:00:00  1.14682  1.14971  1.13253  1.13456  447727.82
1  2019.01.02 22:00:00  1.13456  1.14113  1.13098  1.13944  483691.02
2  2019.01.03 22:00:00  1.13946  1.14192  1.13459  1.13975  419587.69
3  2019.01.06 22:00:00  1.13997  1.14828  1.13981  1.14749  369487.32
4  2019.01.07 22:00:00  1.14747  1.14849  1.14224  1.14422  369776.16
              Time (UTC)     Open     High      Low    Close      Volume
255  2019.12.24 22:00:00  1.10876  1.10885  1.10730  1.10821    2317.8500
256  2019.12.25 22:00:00  1.10885  1.11088  1.10821  1.10974   46848.9740
257  2019.12.26 22:00:00  1.10973  1.11883  1.10964  1.11713  133030.7323
258  2019.12.29 22:00:00  1.11736  1.12207  1.11718  1.11985  130942.6444
259  2019.12.30 22:00:00  1.11985  1.12391  1.11974  1.12076  110824.2900
```

B [56]:
```
data_raw_days_Ask_2018 = pd.read_csv('Data/2018/EURUSD_Daily_Ask_2017.12.31_2018.12.30.csv', low_memory=False, sep=',')
data_raw_days_Bid_2018 = pd.read_csv('Data/2018/EURUSD_Daily_Bid_2017.12.31_2018.12.30.csv', low_memory=False, sep=',')


print(data_raw_days_Ask_2018.head(n=5))
print(data_raw_days_Bid_2018.head(n=5))
print(data_raw_days_Bid_2018[-5:])
```

```
            Time (UTC)     Open     High      Low    Close    Volume
0  2018.01.01 22:00:00  1.20143  1.20813  1.20044  1.20590  198843.00
1  2018.01.02 22:00:00  1.20590  1.20664  1.20015  1.20146  202783.87
2  2018.01.03 22:00:00  1.20146  1.20890  1.20046  1.20685  210052.72
3  2018.01.04 22:00:00  1.20684  1.20833  1.20209  1.20314  199758.33
4  2018.01.07 22:00:00  1.20297  1.20525  1.19561  1.19679  192051.28
            Time (UTC)     Open     High      Low    Close    Volume
0  2018.01.01 22:00:00  1.20102  1.20811  1.20007  1.20583  198808.80
1  2018.01.02 22:00:00  1.20581  1.20662  1.20011  1.20141  194342.53
2  2018.01.03 22:00:00  1.20142  1.20889  1.20044  1.20680  211855.85
3  2018.01.04 22:00:00  1.20679  1.20829  1.20205  1.20304  195569.39
4  2018.01.07 22:00:00  1.20265  1.20521  1.19560  1.19666  183816.72
              Time (UTC)     Open     High      Low    Close    Volume
255  2018.12.24 22:00:00  1.13976  1.14170  1.13894  1.13988    8155.28
256  2018.12.25 22:00:00  1.13901  1.14198  1.13428  1.13507  257461.15
257  2018.12.26 22:00:00  1.13507  1.14540  1.13504  1.14285  563907.83
258  2018.12.27 22:00:00  1.14285  1.14727  1.14247  1.14414  453409.89
259  2018.12.30 22:00:00  1.14331  1.14674  1.14213  1.14644  316061.54
```

B [57]:
```
# day volatility
```

B [58]:
```
daily_volatility = pd.DataFrame(columns=['2018', '2019', '2020'], dtype='float')
daily_volatility
daily_volatility.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 0 entries
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   2018    0 non-null      float64
 1   2019    0 non-null      float64
 2   2020    0 non-null      float64
dtypes: float64(3)
memory usage: 0.0+ bytes
```

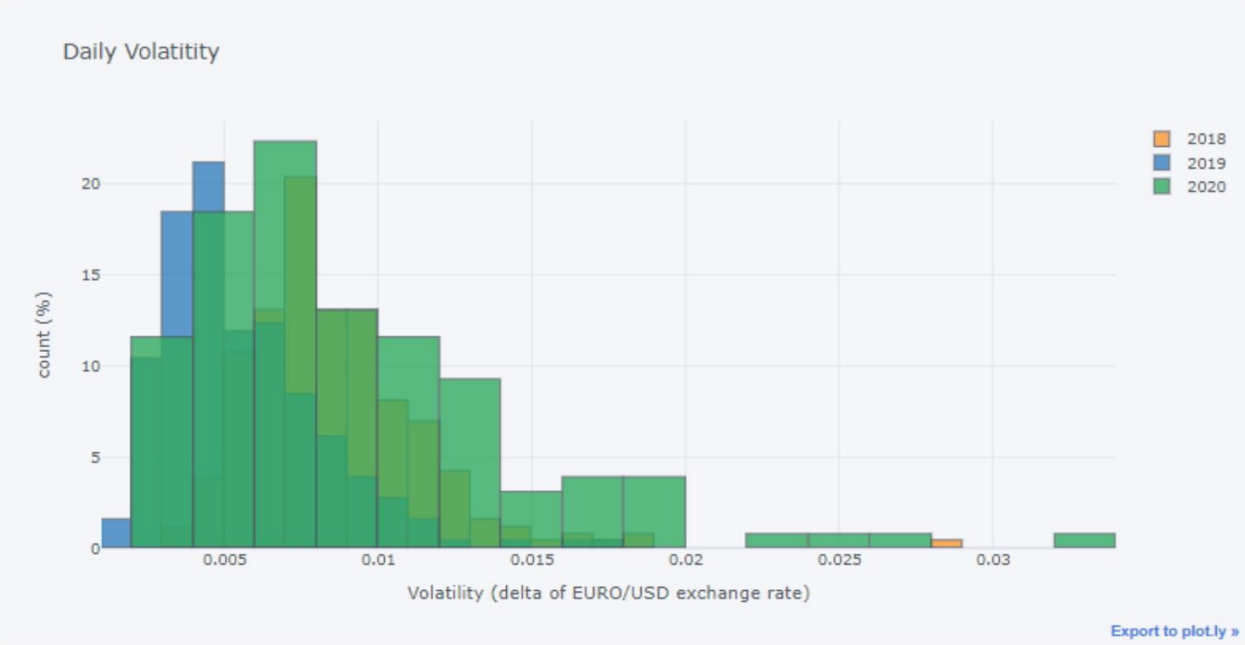
B [59]:

```
daily_volatility[['2018']] = data_raw_days_Ask_2018[['High']] - data_raw_days_Bid_2018[['Low']].values
daily_volatility[['2019']] = data_raw_days_Ask_2019[['High']] - data_raw_days_Bid_2019[['Low']].values
daily_volatility[['2020']] = data_raw_days_Ask[['High']]      - data_raw_days_Bid[['Low']].values

daily_volatility.info()
print(daily_volatility.head(n=5))
print(daily_volatility[-5:])
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 260 entries, 0 to 259
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   2018    260 non-null    float64
 1   2019    260 non-null    float64
 2   2020    130 non-null    float64
dtypes: float64(3)
memory usage: 6.2 KB
      2018     2019     2020
0  0.00806  0.01721  0.00612
1  0.00653  0.01021  0.00551
2  0.00846  0.00736  0.00487
3  0.00628  0.00874  0.00645
4  0.00965  0.00627  0.00669
        2018     2019  2020
255  0.00319  0.00177   NaN
256  0.00777  0.00271   NaN
257  0.01040  0.00921   NaN
258  0.00483  0.00491   NaN
259  0.00477  0.00419   NaN
```

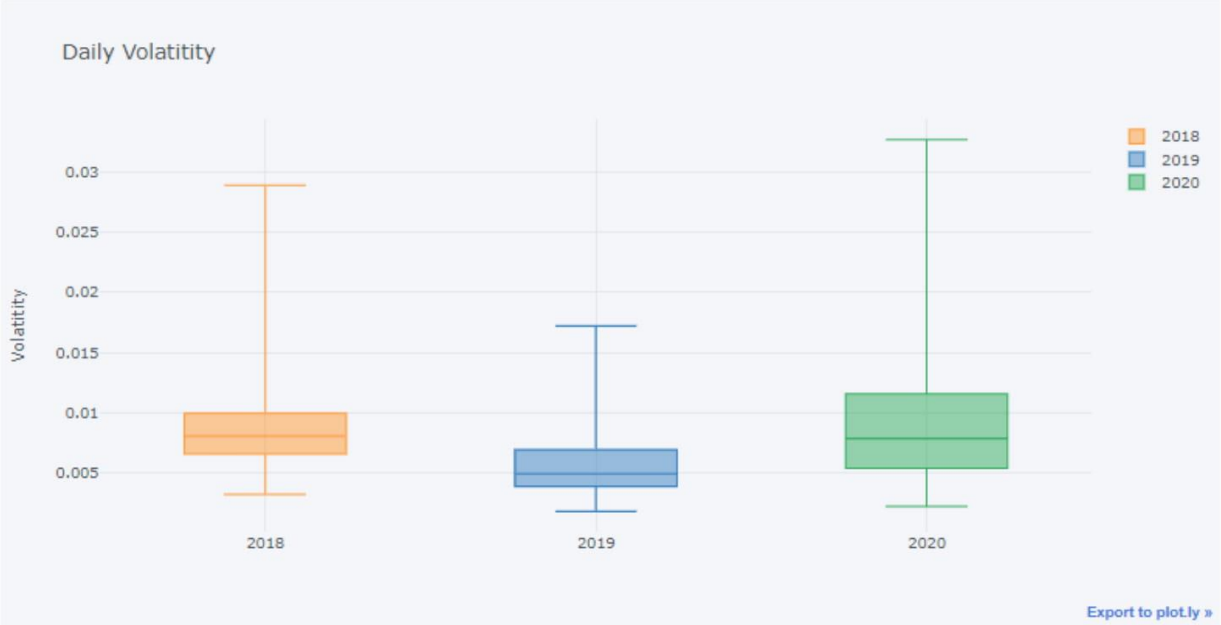
B [60]:

```
daily_volatility[['2018', '2019', '2020']].iplot(
                                        kind='hist',
                                        histnorm='percent',
                                        barmode='overlay',
                                        xTitle='Volatility (delta of EURO/USD exchange rate)',
                                        yTitle='count (%)',
                                        title='Daily Volatitity')
```



B [61]:

```
daily_volatility.iplot(
        kind='box',
        yTitle='Volatitity',
        title='Daily Volatitity')
```



Hour data

B [63]:

```
data_raw_hourly_Ask = pd.read_csv('Data/EURUSD_Hourly_Ask_2020.01.01_2020.06.30.csv', low_memory=False, sep=',')
data_raw_hourly_Bid = pd.read_csv('Data/EURUSD_Hourly_Bid_2020.01.01_2020.06.30.csv', low_memory=False, sep=',')


print(data_raw_hourly_Ask.head(n=5))
print(data_raw_hourly_Bid[-5:])
```

```
            Time (UTC)     Open     High      Low    Close  Volume
0  2020.01.01 22:00:00  1.12154  1.12154  1.12154  1.12154    0.00
1  2020.01.01 23:00:00  1.12158  1.12228  1.12156  1.12191  1560.80
2  2020.01.02 00:00:00  1.12192  1.12193  1.12159  1.12185  1909.81
3  2020.01.02 01:00:00  1.12184  1.12247  1.12182  1.12213  3362.80
4  2020.01.02 02:00:00  1.12212  1.12247  1.12186  1.12226  2867.59
             Time (UTC)     Open     High      Low    Close   Volume
3094  2020.06.30 19:00:00  1.12327  1.12387  1.12263  1.12337  6723.190
3095  2020.06.30 20:00:00  1.12336  1.12356  1.12305  1.12336  3191.170
3096  2020.06.30 21:00:00  1.12340  1.12364  1.12310  1.12314   742.025
3097  2020.06.30 22:00:00  1.12316  1.12362  1.12313  1.12349  1314.545
3098  2020.06.30 23:00:00  1.12348  1.12355  1.12328  1.12335  2553.770
```

B [64]:

```
data_raw_hourly_Ask_2019 = pd.read_csv('Data/2019/EURUSD_Hourly_Ask_2019.01.01_2019.12.31.csv', low_memory=False, sep=',')
data_raw_hourly_Bid_2019 = pd.read_csv('Data/2019/EURUSD_Hourly_Bid_2019.01.01_2019.12.31.csv', low_memory=False, sep=',')

print(data_raw_hourly_Ask_2019.head(n=5))
print(data_raw_hourly_Bid_2019[-5:])
```

```
            Time (UTC)     Open     High      Low    Close   Volume
0  2019.01.01 22:00:00  1.14682  1.14691  1.14626  1.14641   1712.89
1  2019.01.01 23:00:00  1.14643  1.14686  1.14611  1.14616   9370.51
2  2019.01.02 00:00:00  1.14616  1.14630  1.14547  1.14566   8852.19
3  2019.01.02 01:00:00  1.14566  1.14566  1.14462  1.14526  11870.29
4  2019.01.02 02:00:00  1.14525  1.14537  1.14431  1.14459  14304.63
               Time (UTC)     Open     High      Low    Close   Volume
6200  2019.12.30 20:00:00  1.12020  1.12035  1.11963  1.11983  3425.165
6201  2019.12.30 21:00:00  1.11983  1.12027  1.11978  1.11985  1376.280
6202  2019.12.30 22:00:00  1.11985  1.12042  1.11974  1.11995  1475.760
6203  2019.12.30 23:00:00  1.12029  1.12035  1.12012  1.12013  1571.970
6204  2019.12.31 00:00:00  1.12013  1.12050  1.12010  1.12035  1680.400
```

B [65]:

```
data_raw_hourly_Ask_2018 = pd.read_csv('Data/2018/EURUSD_Hourly_Ask_2018.01.01_2018.12.31.csv', low_memory=False, sep=',')
data_raw_hourly_Bid_2018 = pd.read_csv('Data/2018/EURUSD_Hourly_Bid_2018.01.01_2018.12.31.csv', low_memory=False, sep=',')


print(data_raw_hourly_Ask_2018.head(n=5))
print(data_raw_hourly_Bid_2018.head(n=5))
print(data_raw_hourly_Bid_2018[-5:])
```

```
            Time (UTC)     Open     High      Low    Close  Volume
0  2018.01.01 22:00:00  1.20143  1.20173  1.20062  1.20158  1623.51
1  2018.01.01 23:00:00  1.20158  1.20161  1.20044  1.20110  6297.24
2  2018.01.02 00:00:00  1.20109  1.20178  1.20096  1.20162  3564.46
3  2018.01.02 01:00:00  1.20162  1.20232  1.20141  1.20159  6219.69
4  2018.01.02 02:00:00  1.20157  1.20165  1.20081  1.20096  5386.51
            Time (UTC)     Open     High      Low    Close  Volume
0  2018.01.01 22:00:00  1.20102  1.20166  1.20007  1.20136  1687.29
1  2018.01.01 23:00:00  1.20137  1.20158  1.20026  1.20106  6885.93
2  2018.01.02 00:00:00  1.20105  1.20177  1.20092  1.20160  4362.53
3  2018.01.02 01:00:00  1.20159  1.20231  1.20139  1.20154  5860.19
4  2018.01.02 02:00:00  1.20153  1.20163  1.20078  1.20095  6161.26
               Time (UTC)     Open     High      Low    Close   Volume
6200  2018.12.28 20:00:00  1.14412  1.14538  1.14410  1.14419  23544.86
6201  2018.12.28 21:00:00  1.14419  1.14508  1.14398  1.14414   7015.90
6202  2018.12.30 22:00:00  1.14331  1.14397  1.14331  1.14392   1125.38
6203  2018.12.30 23:00:00  1.14393  1.14406  1.14379  1.14389   8728.43
6204  2018.12.31 00:00:00  1.14389  1.14502  1.14381  1.14467  11734.42
```

B [67]:

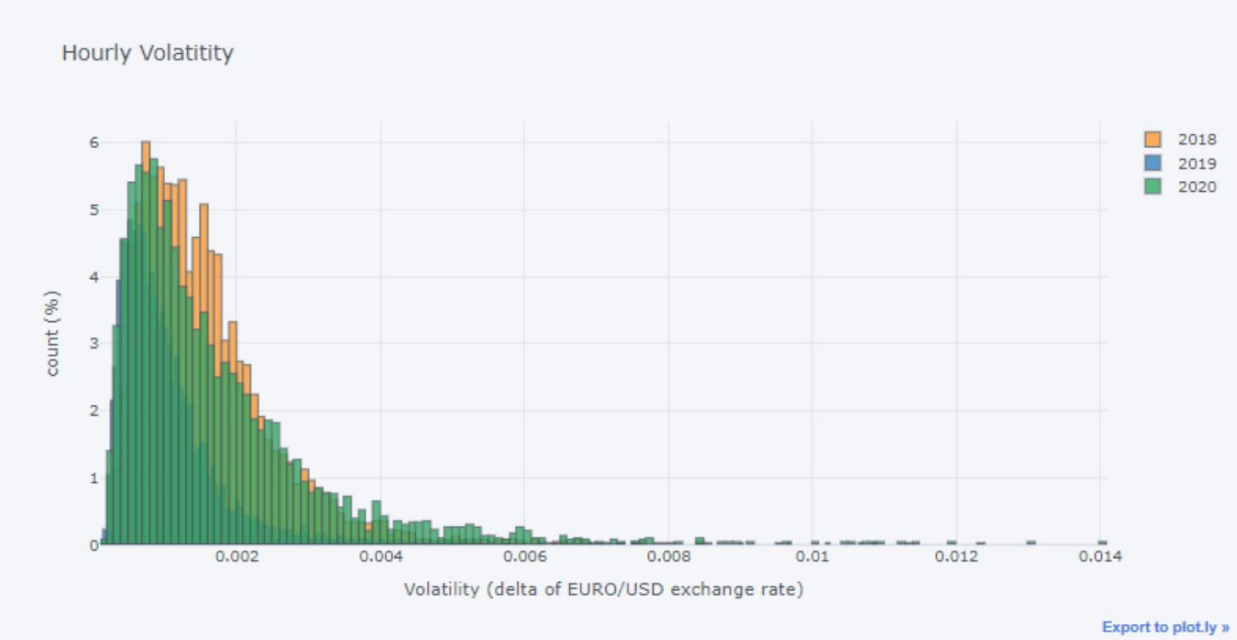
```
hourly_volatility = pd.DataFrame(columns=['2018', '2019', '2020'], dtype='float')
hourly_volatility
hourly_volatility.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 0 entries
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   2018    0 non-null      float64
 1   2019    0 non-null      float64
 2   2020    0 non-null      float64
dtypes: float64(3)
memory usage: 0.0+ bytes
```

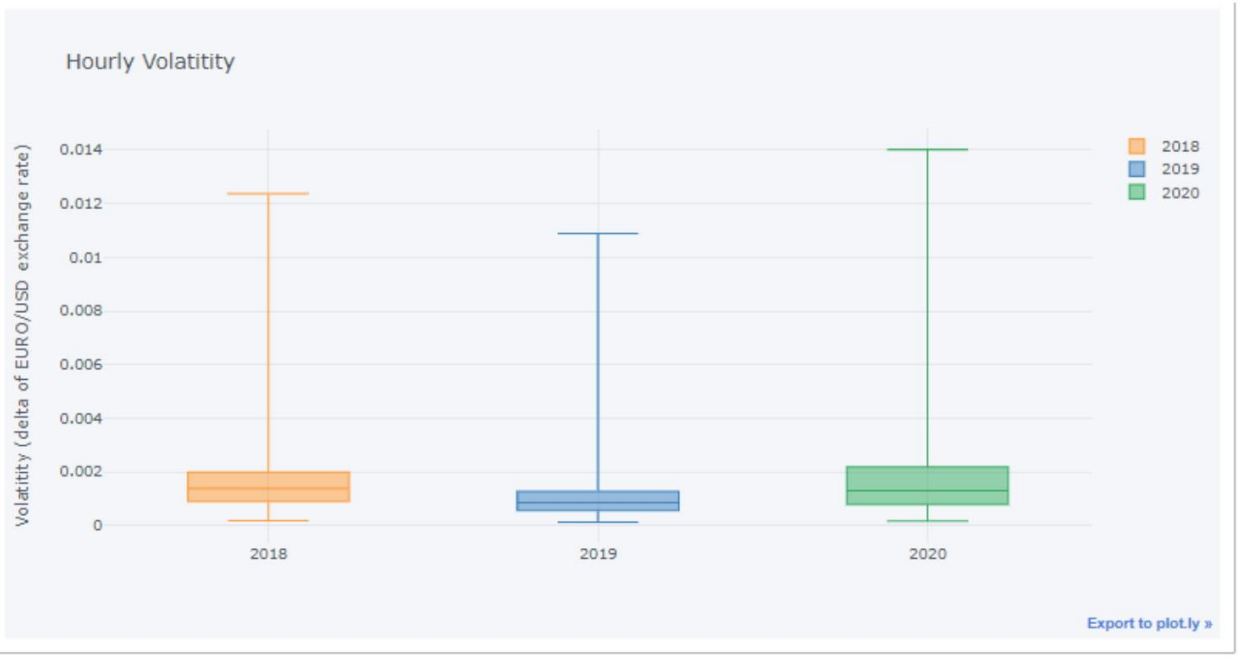
In [68]:
```
hourly_volatility[['2018']] = data_raw_hourly_Ask_2018[['High']] - data_raw_hourly_Bid_2018[['Low']].values
hourly_volatility[['2019']] = data_raw_hourly_Ask_2019[['High']] - data_raw_hourly_Bid_2019[['Low']].values
hourly_volatility[['2020']] = data_raw_hourly_Ask[['High']]      - data_raw_hourly_Bid[['Low']].values

hourly_volatility.info()
print(hourly_volatility.head(n=5))
print(hourly_volatility[-5:])
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6205 entries, 0 to 6204
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   2018    6205 non-null   float64
 1   2019    6205 non-null   float64
 2   2020    3099 non-null   float64
dtypes: float64(3)
memory usage: 145.6 KB
      2018     2019     2020
0  0.00166  0.00093  0.00078
1  0.00135  0.00120  0.00086
2  0.00086  0.00087  0.00036
3  0.00093  0.00109  0.00067
4  0.00087  0.00111  0.00063
         2018     2019  2020
6200  0.00132  0.00075   NaN
6201  0.00112  0.00052   NaN
6202  0.00072  0.00079   NaN
6203  0.00034  0.00025   NaN
6204  0.00124  0.00044   NaN
```

In [69]:
```
hourly_volatility[['2018', '2019', '2020']].iplot(
                                        kind='hist',
                                        histnorm='percent',
                                        barmode='overlay',
                                        xTitle='Volatility (delta of EURO/USD exchange rate)',
                                        yTitle='count (%)',
                                        title='Hourly Volatitity')
```



In [70]:
```
hourly_volatility.iplot(
       kind='box',
       yTitle='Volatitity (delta of EURO/USD exchange rate)',
       title='Hourly Volatitity')
```



In [ ]:

30-minutes volatility

In [72]:
```
data_raw_30_Mins_Ask = pd.read_csv('Data/EURUSD_30 Mins_Ask_2020.01.01_2020.06.30.csv', low_memory=False, sep=',')
data_raw_30_Mins_Bid = pd.read_csv('Data/EURUSD_30 Mins_Bid_2020.01.01_2020.06.30.csv', low_memory=False, sep=',')

print(data_raw_30_Mins_Ask.head(n=5))
print(data_raw_30_Mins_Bid[-5:])
```

```
            Time (UTC)     Open     High      Low    Close   Volume
0  2020.01.01 22:00:00  1.12172  1.12179  1.12124  1.12163   299.764
1  2020.01.01 22:30:00  1.12166  1.12173  1.12156  1.12157   588.540
2  2020.01.01 23:00:00  1.12158  1.12228  1.12156  1.12213   973.810
3  2020.01.01 23:30:00  1.12214  1.12219  1.12180  1.12191   586.990
4  2020.01.02 00:00:00  1.12192  1.12193  1.12159  1.12164  1127.490
              Time (UTC)     Open     High      Low    Close   Volume
6193  2020.06.30 21:30:00  1.12336  1.12337  1.12310  1.12314   574.600
6194  2020.06.30 22:00:00  1.12316  1.12362  1.12313  1.12355   717.875
6195  2020.06.30 22:30:00  1.12356  1.12362  1.12341  1.12349   596.670
6196  2020.06.30 23:00:00  1.12348  1.12355  1.12328  1.12332  1079.430
6197  2020.06.30 23:30:00  1.12332  1.12346  1.12329  1.12335  1474.340
```

In [73]:
```
data_raw_30_Mins_Ask_2019 = pd.read_csv('Data/2019/EURUSD_30 Mins_Ask_2019.01.01_2019.12.31.csv', low_memory=False, sep=',')
data_raw_30_Mins_Bid_2019 = pd.read_csv('Data/2019/EURUSD_30 Mins_Bid_2019.01.01_2019.12.31.csv', low_memory=False, sep=',')

print(data_raw_30_Mins_Ask_2019.head(n=5))
print(data_raw_30_Mins_Bid_2019[-5:])
```

```
            Time (UTC)     Open     High      Low    Close  Volume
0  2019.01.01 22:00:00  1.14682  1.14691  1.14636  1.14660   494.50
1  2019.01.01 22:30:00  1.14662  1.14675  1.14626  1.14641  1218.39
2  2019.01.01 23:00:00  1.14643  1.14686  1.14611  1.14669  5908.96
3  2019.01.01 23:30:00  1.14669  1.14673  1.14614  1.14616  3461.55
4  2019.01.02 00:00:00  1.14616  1.14630  1.14578  1.14583  5144.20
              Time (UTC)     Open     High      Low    Close  Volume
12404  2019.12.30 22:00:00  1.11985  1.12042  1.11985  1.11997   442.32
12405  2019.12.30 22:30:00  1.11997  1.12036  1.11974  1.11995  1033.44
12406  2019.12.30 23:00:00  1.12029  1.12035  1.12016  1.12028   712.73
12407  2019.12.30 23:30:00  1.12029  1.12033  1.12012  1.12013   859.24
12408  2019.12.31 00:00:00  1.12013  1.12036  1.12010  1.12036   916.51
```

B [74]:

```
data_raw_30_Mins_Ask_2018 = pd.read_csv('Data/2018/EURUSD_30 Mins_Ask_2018.01.01_2018.12.31.csv', low_memory=False, sep=',')
data_raw_30_Mins_Bid_2018 = pd.read_csv('Data/2018/EURUSD_30 Mins_Bid_2018.01.01_2018.12.31.csv', low_memory=False, sep=',')


print(data_raw_30_Mins_Ask_2018.head(n=5))
print(data_raw_30_Mins_Bid_2018.head(n=5))
print(data_raw_30_Mins_Bid_2018[-5:])
```

```
            Time (UTC)     Open     High      Low    Close  Volume
0  2018.01.01 22:00:00  1.20143  1.20148  1.20062  1.20085   920.58
1  2018.01.01 22:30:00  1.20083  1.20173  1.20083  1.20158   702.93
2  2018.01.01 23:00:00  1.20158  1.20161  1.20044  1.20141  4018.41
3  2018.01.01 23:30:00  1.20143  1.20155  1.20098  1.20110  2278.83
4  2018.01.02 00:00:00  1.20109  1.20178  1.20096  1.20148  2178.16
            Time (UTC)     Open     High      Low    Close  Volume
0  2018.01.01 22:00:00  1.20102  1.20106  1.20007  1.20064   853.06
1  2018.01.01 22:30:00  1.20064  1.20166  1.20062  1.20136   834.23
2  2018.01.01 23:00:00  1.20137  1.20158  1.20026  1.20135  4130.51
3  2018.01.01 23:30:00  1.20136  1.20151  1.20093  1.20106  2755.42
4  2018.01.02 00:00:00  1.20105  1.20177  1.20092  1.20143  1874.49
                Time (UTC)     Open     High      Low    Close  Volume
12404  2018.12.30 22:00:00  1.14331  1.14368  1.14331  1.14364   349.41
12405  2018.12.30 22:30:00  1.14365  1.14397  1.14361  1.14392   775.97
12406  2018.12.30 23:00:00  1.14393  1.14406  1.14386  1.14393  4116.24
12407  2018.12.30 23:30:00  1.14392  1.14400  1.14379  1.14389  4612.19
12408  2018.12.31 00:00:00  1.14389  1.14477  1.14381  1.14477  5525.80
```

B [76]:

```
volatility_30_Mins = pd.DataFrame(columns=['2018', '2019', '2020'], dtype='float')
volatility_30_Mins
volatility_30_Mins.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 0 entries
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   2018    0 non-null      float64
 1   2019    0 non-null      float64
 2   2020    0 non-null      float64
dtypes: float64(3)
memory usage: 0.0+ bytes
```

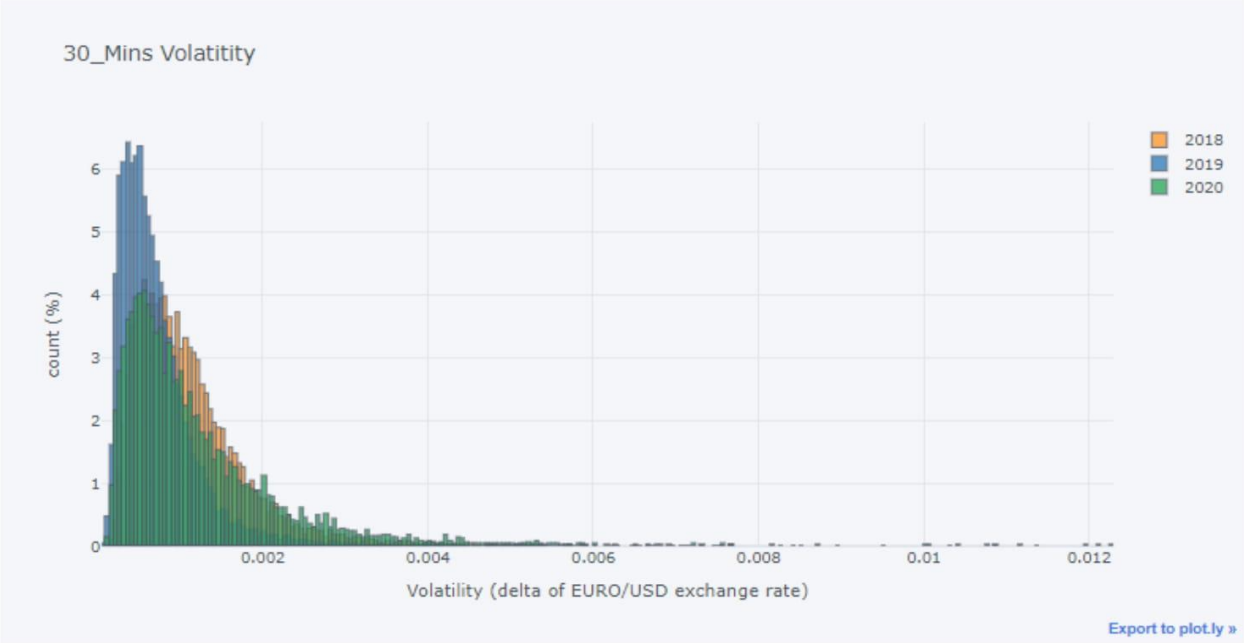
B [77]:

```
volatility_30_Mins[['2018']] = data_raw_30_Mins_Ask_2018[['High']] - data_raw_30_Mins_Bid_2018[['Low']].values
volatility_30_Mins[['2019']] = data_raw_30_Mins_Ask_2019[['High']] - data_raw_30_Mins_Bid_2019[['Low']].values
volatility_30_Mins[['2020']] = data_raw_30_Mins_Ask[['High']]      - data_raw_30_Mins_Bid[['Low']].values

volatility_30_Mins.info()
print(volatility_30_Mins.head(n=5))
print(volatility_30_Mins[-5:])
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12409 entries, 0 to 12408
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   2018    12409 non-null  float64
 1   2019    12409 non-null  float64
 2   2020    6198 non-null   float64
dtypes: float64(3)
memory usage: 291.0 KB
      2018     2019     2020
0  0.00141  0.00093  0.00059
1  0.00111  0.00073  0.00035
2  0.00135  0.00120  0.00086
3  0.00062  0.00063  0.00043
4  0.00086  0.00057  0.00036
          2018     2019  2020
12404  0.00072  0.00068   NaN
12405  0.00042  0.00072   NaN
12406  0.00027  0.00021   NaN
12407  0.00023  0.00024   NaN
12408  0.00101  0.00029   NaN
```

B [78]:

```
volatility_30_Mins[['2018', '2019', '2020']].iplot(
                                        kind='hist',
                                        histnorm='percent',
                                        barmode='overlay',
                                        xTitle='Volatility (delta of EURO/USD exchange rate)',
                                        yTitle='count (%)',
                                        title='30_Mins Volatitity')
```



[79]:
```
volatility_30_Mins.iplot(
        kind='box',
        yTitle='Volatitity (delta of EURO/USD exchange rate)',
        title='30_Mins Volatitity')
```

15-minutes data

B [82]:
```
data_raw_15_Mins_Ask = pd.read_csv('Data/EURUSD_15 Mins_Ask_2020.01.01_2020.06.30.csv', low_memory=False, sep=',')
data_raw_15_Mins_Bid = pd.read_csv('Data/EURUSD_15 Mins_Bid_2020.01.01_2020.06.30.csv', low_memory=False, sep=',')


print(data_raw_15_Mins_Ask.head(n=5))
print(data_raw_15_Mins_Bid[-5:])
```

```
            Time (UTC)     Open     High      Low    Close  Volume
0  2020.01.01 22:00:00  1.12172  1.12179  1.12124  1.12152   63.874
1  2020.01.01 22:15:00  1.12150  1.12169  1.12143  1.12163  235.890
2  2020.01.01 22:30:00  1.12166  1.12169  1.12157  1.12162   80.600
3  2020.01.01 22:45:00  1.12166  1.12173  1.12156  1.12157  507.940
4  2020.01.01 23:00:00  1.12158  1.12228  1.12156  1.12203  741.000
                Time (UTC)     Open     High      Low    Close  Volume
12391  2020.06.30 22:45:00  1.12348  1.12356  1.12341  1.12349   352.65
12392  2020.06.30 23:00:00  1.12348  1.12355  1.12341  1.12341   391.95
12393  2020.06.30 23:15:00  1.12342  1.12344  1.12328  1.12332   687.48
12394  2020.06.30 23:30:00  1.12332  1.12344  1.12329  1.12339   964.76
12395  2020.06.30 23:45:00  1.12338  1.12346  1.12333  1.12335   509.58
```

B [83]:

```
data_raw_15_Mins_Ask_2019 = pd.read_csv('Data/2019/EURUSD_15 Mins_Ask_2019.01.01_2019.12.31.csv', low_memory=False, sep=',')
data_raw_15_Mins_Bid_2019 = pd.read_csv('Data/2019/EURUSD_15 Mins_Bid_2019.01.01_2019.12.31.csv', low_memory=False, sep=',')

print(data_raw_15_Mins_Ask_2019.head(n=5))
print(data_raw_15_Mins_Bid_2019[-5:])
```

```
            Time (UTC)     Open     High      Low    Close  Volume
0  2019.01.01 22:00:00  1.14682  1.14691  1.14636  1.14667   283.78
1  2019.01.01 22:15:00  1.14669  1.14675  1.14636  1.14660   210.72
2  2019.01.01 22:30:00  1.14662  1.14674  1.14626  1.14643   306.78
3  2019.01.01 22:45:00  1.14643  1.14675  1.14630  1.14641   911.61
4  2019.01.01 23:00:00  1.14643  1.14686  1.14611  1.14662  2306.68
                Time (UTC)     Open     High      Low    Close  Volume
24812  2019.12.30 23:00:00  1.12029  1.12032  1.12016  1.12028   470.31
24813  2019.12.30 23:15:00  1.12028  1.12035  1.12027  1.12028   242.42
24814  2019.12.30 23:30:00  1.12029  1.12033  1.12014  1.12018   504.31
24815  2019.12.30 23:45:00  1.12019  1.12031  1.12012  1.12013   354.93
24816  2019.12.31 00:00:00  1.12013  1.12035  1.12010  1.12028   558.66
```

B [84]:

```
data_raw_15_Mins_Ask_2018 = pd.read_csv('Data/2018/EURUSD_15 Mins_Ask_2018.01.01_2018.12.31.csv', low_memory=False, sep=',')
data_raw_15_Mins_Bid_2018 = pd.read_csv('Data/2018/EURUSD_15 Mins_Bid_2018.01.01_2018.12.31.csv', low_memory=False, sep=',')


print(data_raw_15_Mins_Ask_2018.head(n=5))
print(data_raw_15_Mins_Bid_2018.head(n=5))
print(data_raw_15_Mins_Bid_2018[-5:])
```

```
            Time (UTC)     Open     High      Low    Close  Volume
0  2018.01.01 22:00:00  1.20143  1.20148  1.20062  1.20105   229.03
1  2018.01.01 22:15:00  1.20104  1.20131  1.20076  1.20085   691.55
2  2018.01.01 22:30:00  1.20083  1.20173  1.20083  1.20162   423.00
3  2018.01.01 22:45:00  1.20160  1.20165  1.20145  1.20158   279.93
4  2018.01.01 23:00:00  1.20158  1.20158  1.20044  1.20154  2216.51
            Time (UTC)     Open     High      Low    Close  Volume
0  2018.01.01 22:00:00  1.20102  1.20103  1.20007  1.20093   259.03
1  2018.01.01 22:15:00  1.20094  1.20106  1.20054  1.20064   594.03
2  2018.01.01 22:30:00  1.20064  1.20166  1.20062  1.20140   392.86
3  2018.01.01 22:45:00  1.20139  1.20150  1.20131  1.20136   441.37
4  2018.01.01 23:00:00  1.20137  1.20148  1.20026  1.20147  2528.19
                Time (UTC)     Open     High      Low    Close  Volume
24812  2018.12.30 23:00:00  1.14393  1.14406  1.14391  1.14392  1941.02
24813  2018.12.30 23:15:00  1.14393  1.14403  1.14386  1.14393  2175.22
24814  2018.12.30 23:30:00  1.14392  1.14400  1.14379  1.14391  1660.45
24815  2018.12.30 23:45:00  1.14391  1.14397  1.14388  1.14389  2951.74
24816  2018.12.31 00:00:00  1.14389  1.14429  1.14381  1.14422  2903.29
```

B [86]:

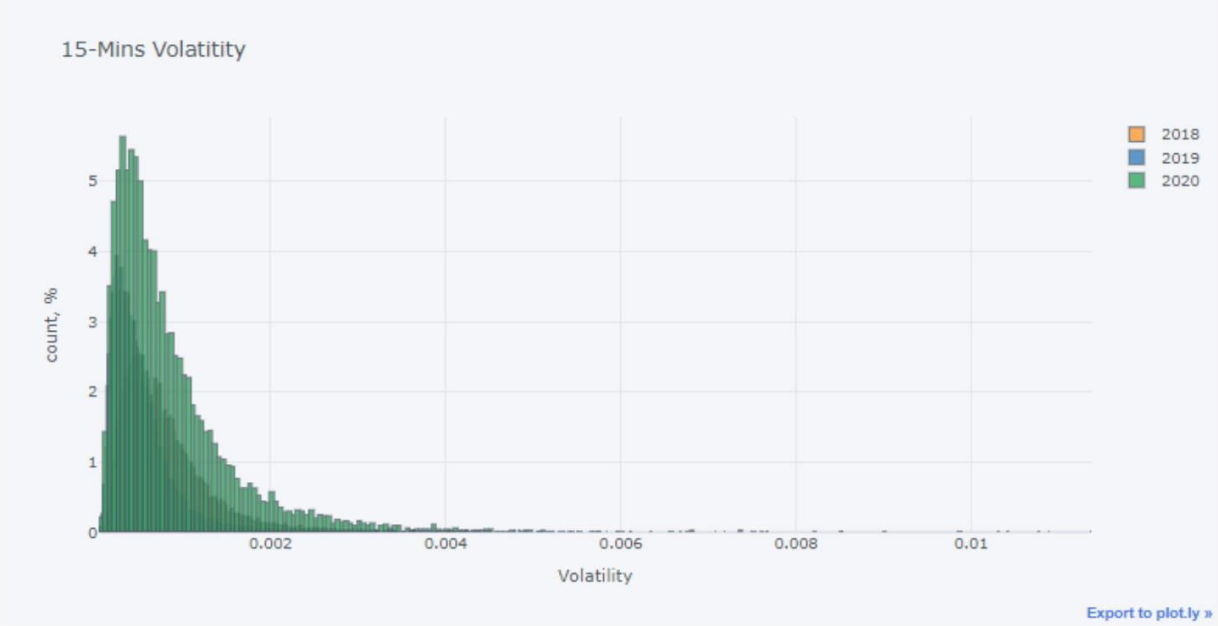
```
volatility_15_Mins = pd.DataFrame(columns=['2018', '2019', '2020'], dtype='float')
volatility_15_Mins
volatility_15_Mins.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 0 entries
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   2018    0 non-null      float64
 1   2019    0 non-null      float64
 2   2020    0 non-null      float64
dtypes: float64(3)
memory usage: 0.0+ bytes
```

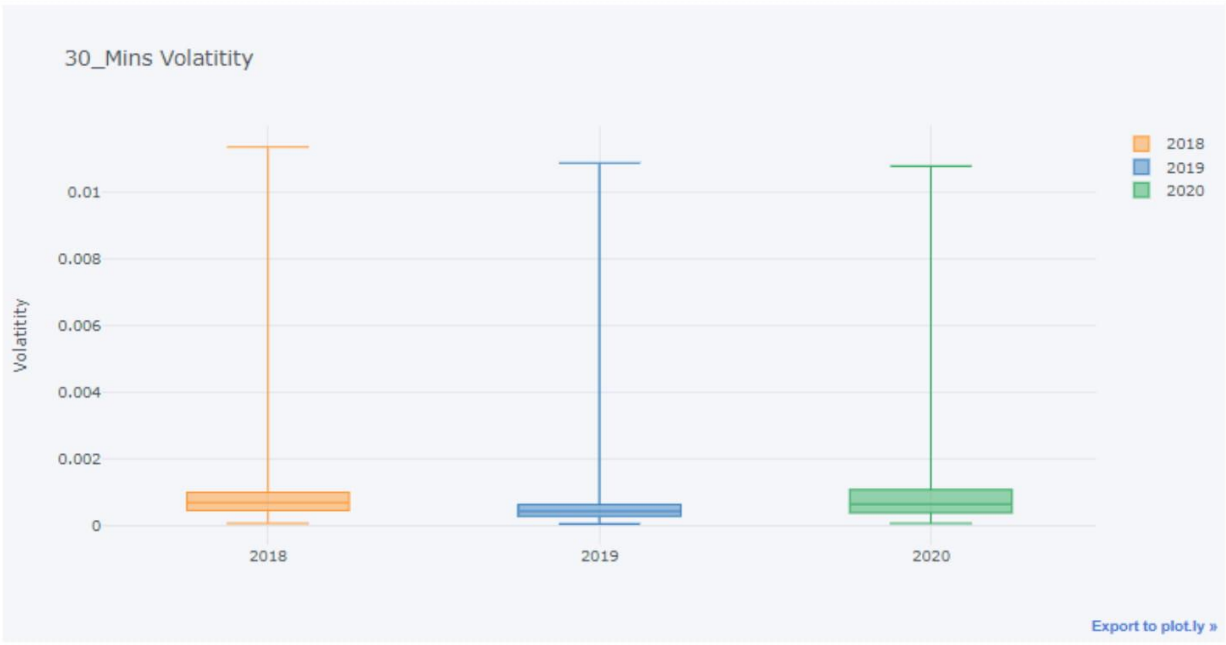
In [87]:
```
volatility_15_Mins[['2018']] = data_raw_15_Mins_Ask_2018[['High']] - data_raw_15_Mins_Bid_2018[['Low']].values
volatility_15_Mins[['2019']] = data_raw_15_Mins_Ask_2019[['High']] - data_raw_15_Mins_Bid_2019[['Low']].values
volatility_15_Mins[['2020']] = data_raw_15_Mins_Ask[['High']]      - data_raw_15_Mins_Bid[['Low']].values

volatility_15_Mins.info()
print(volatility_15_Mins.head(n=5))
print(volatility_15_Mins[-5:])
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24817 entries, 0 to 24816
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   2018    24817 non-null  float64
 1   2019    24817 non-null  float64
 2   2020    12396 non-null  float64
dtypes: float64(3)
memory usage: 581.8 KB
      2018     2019     2020
0  0.00141  0.00093  0.00059
1  0.00077  0.00071  0.00042
2  0.00111  0.00071  0.00031
3  0.00034  0.00073  0.00030
4  0.00132  0.00120  0.00086
          2018     2019  2020
24812  0.00022  0.00020   NaN
24813  0.00021  0.00010   NaN
24814  0.00023  0.00022   NaN
24815  0.00013  0.00020   NaN
24816  0.00052  0.00028   NaN
```

In [88]:
```
volatility_15_Mins[['2018', '2019', '2020']].iplot(
                                    kind='hist',
                                    histnorm='percent',
                                    barmode='overlay',
                                    xTitle='volatility',
                                    yTitle='count, %',
                                    title='15-Mins Volatitity')
```



In [89]:
```
volatility_15_Mins.iplot(
       kind='box',
       yTitle='Volatitity',
       title='30_Mins Volatitity')
```



In [ ]:

Intday trading advantages due to reduced fractal dimension and volatility

In [90]:
```
print(data_raw_hourly_Ask.head(n=5))
print(data_raw_hourly_Ask[-5:])
```

```
            Time (UTC)     Open     High      Low    Close  Volume
0  2020.01.01 22:00:00  1.12154  1.12154  1.12154  1.12154    0.00
1  2020.01.01 23:00:00  1.12158  1.12228  1.12156  1.12191  1560.80
2  2020.01.02 00:00:00  1.12192  1.12193  1.12159  1.12185  1909.81
3  2020.01.02 01:00:00  1.12184  1.12247  1.12182  1.12213  3362.80
4  2020.01.02 02:00:00  1.12212  1.12247  1.12186  1.12226  2867.59
                Time (UTC)     Open     High      Low    Close   Volume
3094  2020.06.30 19:00:00  1.12328  1.12388  1.12264  1.12338  6442.970
3095  2020.06.30 20:00:00  1.12338  1.12359  1.12307  1.12341  2947.780
3096  2020.06.30 21:00:00  1.12347  1.12373  1.12321  1.12325  1590.380
3097  2020.06.30 22:00:00  1.12325  1.12367  1.12323  1.12353  1775.355
3098  2020.06.30 23:00:00  1.12352  1.12360  1.12332  1.12338  2710.450
```

In [91]:
```
# Daily_Volatility
daily_volatility2 = pd.DataFrame(columns=['2020'], dtype='float')
daily_volatility2[['2020']] = data_raw_days_Ask[['High']] - data_raw_days_Ask[['Low']].values

daily_volatility2.info()
print(daily_volatility2['2020'].head(n=5))
print(daily_volatility2['2020'].tail(5))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 130 entries, 0 to 129
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   2020    130 non-null    float64
dtypes: float64(1)
memory usage: 1.1 KB
0    0.00609
1    0.00549
2    0.00484
3    0.00643
4    0.00667
Name: 2020, dtype: float64
125    0.00693
126    0.00441
127    0.00706
128    0.00709
129    0.00903
Name: 2020, dtype: float64
```

In [92]:
```
Daily_Volatility2 = daily_volatility2['2020'].sum()
Daily_Volatility2
```

Out[92]: 1.1714299999999995

In [93]:
```
# Hourly_Volatility
hourly_volatility2 = pd.DataFrame(columns=['2020'], dtype='float')
hourly_volatility2[['2020']] = data_raw_hourly_Ask[['High']] - data_raw_hourly_Ask[['Low']].values

hourly_volatility2.info()
print(hourly_volatility2['2020'].head(n=5))
print(hourly_volatility2['2020'].tail(5))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3099 entries, 0 to 3098
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   2020    3099 non-null   float64
dtypes: float64(1)
memory usage: 24.3 KB
0    0.00000
1    0.00072
2    0.00034
3    0.00065
4    0.00061
Name: 2020, dtype: float64
3094    0.00124
3095    0.00052
3096    0.00052
3097    0.00044
3098    0.00028
Name: 2020, dtype: float64
```

In [94]:
```
Hourly_Volatility = hourly_volatility2['2020'].sum()
Hourly_Volatility
```

Out[94]: 5.173710000000006

In [95]:
```
# volatility_5_Mins

data_raw_5_Mins_Ask = pd.read_csv('Data/EURUSD_5 Mins_Ask_2020.01.01_2020.06.30.csv', low_memory=False, sep=',')

print(data_raw_5_Mins_Ask.head(n=5))
print(data_raw_5_Mins_Ask[-5:])
```

```
            Time (UTC)     Open     High      Low    Close  Volume
0  2020.01.01 22:00:00  1.12172  1.12179  1.12151  1.12151   15.604
1  2020.01.01 22:05:00  1.12151  1.12175  1.12133  1.12159   14.180
2  2020.01.01 22:10:00  1.12158  1.12162  1.12124  1.12152   34.090
3  2020.01.01 22:15:00  1.12150  1.12169  1.12143  1.12151   85.170
4  2020.01.01 22:20:00  1.12151  1.12168  1.12150  1.12164  131.060
                Time (UTC)     Open     High      Low    Close  Volume
37183  2020.06.30 23:35:00  1.12344  1.12346  1.12337  1.12338   286.17
37184  2020.06.30 23:40:00  1.12340  1.12343  1.12335  1.12343   303.22
37185  2020.06.30 23:45:00  1.12343  1.12349  1.12339  1.12340   206.24
37186  2020.06.30 23:50:00  1.12339  1.12349  1.12337  1.12342   173.28
37187  2020.06.30 23:55:00  1.12343  1.12343  1.12337  1.12338   209.57
```

In [96]:
```
volatility_5_Mins = pd.DataFrame(columns=['2020'], dtype='float')
volatility_5_Mins[['2020']] = data_raw_5_Mins_Ask[['High']] - data_raw_5_Mins_Ask[['Low']].values

volatility_5_Mins.info()
print(volatility_5_Mins['2020'].head(n=5))
print(volatility_5_Mins['2020'].tail(5))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 37188 entries, 0 to 37187
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   2020    37188 non-null  float64
dtypes: float64(1)
memory usage: 290.7 KB
0    0.00028
1    0.00042
2    0.00038
3    0.00026
4    0.00018
Name: 2020, dtype: float64
37183    0.00009
37184    0.00008
37185    0.00010
37186    0.00012
37187    0.00006
Name: 2020, dtype: float64
```

In [97]:
```
Volatility_5_Mins = volatility_5_Mins['2020'].sum()
Volatility_5_Mins
```

Out[97]:
```
16.959519999999994
```

In [98]:
```
# volatility_1_Min

data_raw_1_Min_Ask = pd.read_csv('Data/EURUSD_1 Min_Ask_2020.01.01_2020.06.30.csv', low_memory=False, sep=',')

print(data_raw_1_Min_Ask.head(n=5))
print(data_raw_1_Min_Ask[-5:])
```

```
            Time (UTC)     Open     High      Low    Close  Volume
0  2020.01.01 22:00:00  1.12154  1.12154  1.12154  1.12154    0.000
1  2020.01.01 22:01:00  1.12154  1.12154  1.12154  1.12154    0.000
2  2020.01.01 22:02:00  1.12172  1.12179  1.12171  1.12172    3.644
3  2020.01.01 22:03:00  1.12171  1.12172  1.12171  1.12172    9.520
4  2020.01.01 22:04:00  1.12171  1.12172  1.12151  1.12151    2.440
                Time (UTC)     Open     High      Low    Close  Volume
185935  2020.06.30 23:55:00  1.12343  1.12343  1.12338  1.12341   82.03
185936  2020.06.30 23:56:00  1.12340  1.12341  1.12338  1.12338   45.86
185937  2020.06.30 23:57:00  1.12339  1.12339  1.12337  1.12339   16.40
185938  2020.06.30 23:58:00  1.12338  1.12340  1.12337  1.12338   29.41
185939  2020.06.30 23:59:00  1.12339  1.12343  1.12338  1.12338   35.87
```

In [99]:
```

volatility_1_Min = pd.DataFrame(columns=['2020'], dtype='float')
volatility_1_Min[['2020']] = data_raw_1_Min_Ask[['High']] - data_raw_1_Min_Ask[['Low']].values

volatility_1_Min.info()
print(volatility_1_Min['2020'].head(n=5))
print(volatility_1_Min['2020'].tail(5))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 185940 entries, 0 to 185939
Data columns (total 1 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   2020    185940 non-null  float64
dtypes: float64(1)
memory usage: 1.4 MB
0    0.00000
1    0.00000
2    0.00008
3    0.00001
4    0.00021
Name: 2020, dtype: float64
185935    0.00005
185936    0.00003
185937    0.00002
185938    0.00003
185939    0.00005
Name: 2020, dtype: float64
```

In [100]:
```
Volatility_1_Min = volatility_1_Min['2020'].sum()
Volatility_1_Min
```

Out[100]:
```
34.43332999999998
```

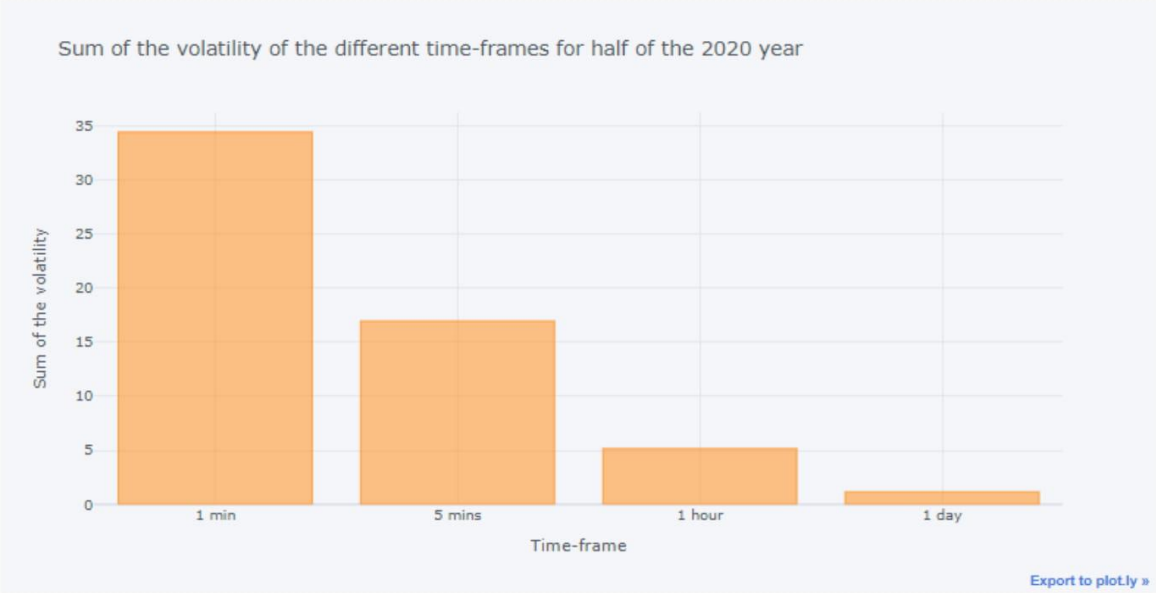
In [101]:
```
# summary volatility DataFrame
Volatility_compare = pd.DataFrame(columns=['2020'], dtype='float')
Volatility_compare.loc[0] = Volatility_1_Min
Volatility_compare.loc[1] = Volatility_5_Mins
Volatility_compare.loc[2] = Hourly_Volatility
Volatility_compare.loc[3] = Daily_Volatility2
Volatility_compare['Term'] =['1 min', '5 mins', '1 hour', '1 day']

Volatility_compare['%'] = Volatility_compare['2020'] / Volatility_1_Min * 100

Volatility_compare.info()
Volatility_compare
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4 entries, 0 to 3
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   2020    4 non-null      float64
 1   Term    4 non-null      object
 2   %       4 non-null      float64
dtypes: float64(2), object(1)
memory usage: 128.0+ bytes
```

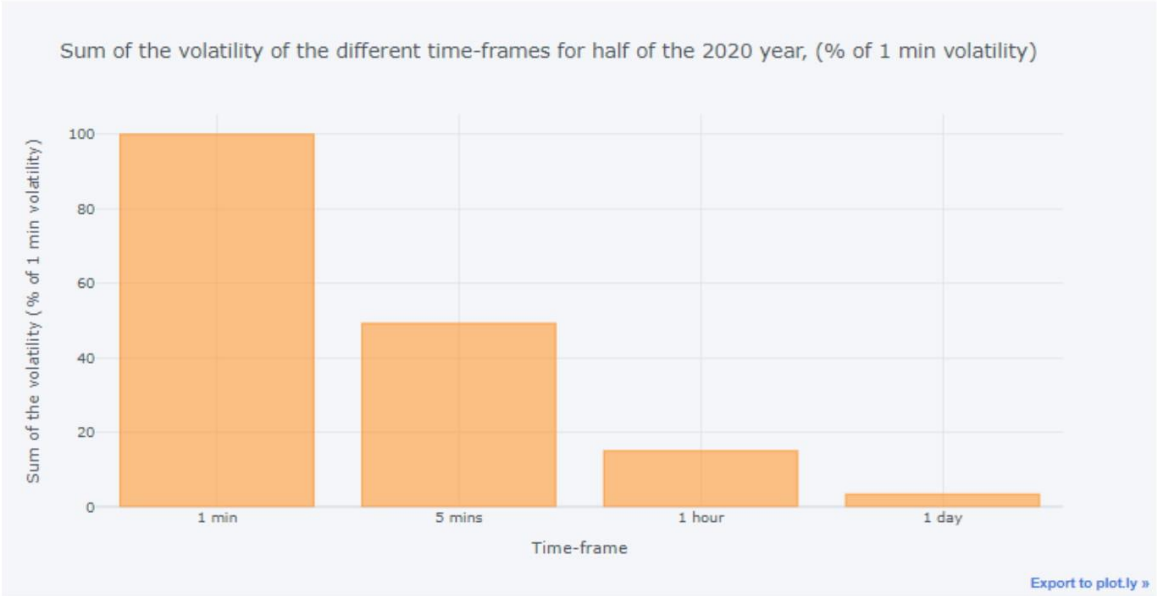
Out[101]:

	2020	Term	%
0	34.43333	1 min	100.000000
1	16.95952	5 mins	49.253209
2	5.17371	1 hour	15.025297
3	1.17143	1 day	3.402024

B [102]:
```
Volatility_compare.iplot(kind='bar', x='Term', y='2020', xTitle='Time-frame', yTitle='Sum of the volatility',
    title='Sum of the volatility of the different time-frames for half of the 2020 year')
```



B [103]:
```
Volatility_compare.iplot(kind='bar', x='Term', y='%', xTitle='Time-frame',
    yTitle='Sum of the volatility (% of 1 min volatility)',
    title='Sum of the volatility of the different time-frames for half of the 2020 year, (% of 1 min volatility)')
```



Appendix XIII.

CNN binary classification on Trend or Flat

Probability prediction

Convolutional Neural Network

Forex EURUSD

Tick Data With Dukastcopy

Data for 2020 year

We use ONE-DIMENSIONAL convolutional Neural network

(Choosing the best version of training - a variable number of epochs - ModelCheckpoint Callback)

Normalisation of volumes - based on a maximum of 10 lots (the main number of lots). The interval (from 10 to 5000 lots) are equated to 10 lots (those to 1 after normalisation)

We normalise the intervals between ticks - based on a maximum of 20,000 milliseconds (20 seconds) (the main number of intervals). The interval (from 20,000) are equated to 20,000 milliseconds (those to 1 after normalization)

Broken into pieces of 200 ticks

The GOAL: Prediction of the Vector of Probabilities

We use a csv file with Dukastcopy and Keras, substantially REFINED and ALREADY normalised

B [1]:
```
# Tensorflow and tf.keras

import tensorflow as tf
from tensorflow import keras

import numpy as np
import pandas as pd

import random

import matplotlib.pyplot as plt
%matplotlib inline
```

B [2]:
```
print(tf.__version__)
```

```
2.4.1
```

B [3]:
```
# Progress bar
from tqdm import tqdm
```

B [ ]:
```
# COLABORATE
!cat /proc/driver/nvidia/gpus/0000:00:04.0/information
```

B [4]:
```
# Set seed to repeat results for random number generator
# tk we use stochostic optimization methods
np.random.seed(42)
```

Data

B [ ]:
```
# COLABORATE
from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))
```

B [ ]:
```
from google.colab import drive
drive.mount('/content/grive', force_remount=True)
```

B [ ]:
```
# COLABORATE
!ls
```

B [5]:
```
data = pd.read_csv('/Users/vital/Python_Code/_PhD_Code/2021-04-April/Experiment-2/Results3/EURO_data_raw_common_norm.csv')

print(data.head(n=5))
print(data[-5:])
```

```
       Ask      Bid  Vol_Bid_N  Vol_Ask_N  DateDelta1_N  Goal  Point  Point1
0  1.12160  1.12106      0.075    0.00937       0.00000     8      1       8
1  1.12160  1.12120      0.075    0.00937       0.21775     8      0       0
2  1.12160  1.12117      0.075    0.00937       0.06845     8      0       0
3  1.12161  1.12123      0.075    0.08400       0.03000     8      0       0
4  1.12161  1.12120      0.075    0.08400       0.00505     8      0       0
              Ask      Bid  Vol_Bid_N  Vol_Ask_N  DateDelta1_N  Goal  Point  \
16586025  1.12337  1.12333      0.506      0.075       0.00505     8      0
16586026  1.12338  1.12333      0.394      0.157       0.00505     8      0
16586027  1.12337  1.12333      0.394      0.075       0.32845     8      0
16586028  1.12338  1.12333      0.394      0.202       0.22760     8      0
16586029  1.12338  1.12334      0.037      0.150       0.19405     8      0

          Point1
16586025       0
16586026       0
16586027       0
16586028       0
16586029       0
```

B [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16586030 entries, 0 to 16586029
Data columns (total 8 columns):
 #   Column        Dtype
---  ------        -----
 0   Ask           float64
 1   Bid           float64
 2   Vol_Bid_N     float64
 3   Vol_Ask_N     float64
 4   DateDelta1_N  float64
 5   Goal          int64
 6   Point         int64
 7   Point1        int64
dtypes: float64(5), int64(3)
memory usage: 1012.3 MB
```

B [7]:
```
# downsising

data['Ask']          =pd.to_numeric(data['Ask'],downcast='float')
data['Bid']          =pd.to_numeric(data['Bid'],downcast='float')
data['Vol_Bid_N']    =pd.to_numeric(data['Vol_Bid_N'],downcast='float')
data['Vol_Ask_N']    =pd.to_numeric(data['Vol_Ask_N'],downcast='float')
data['DateDelta1_N'] =pd.to_numeric(data['DateDelta1_N'],downcast='float')

data['Goal']         = pd.to_numeric(data['Goal'],downcast='integer')
data['Point']        = pd.to_numeric(data['Point'],downcast='integer')
data['Point1']       = pd.to_numeric(data['Point1'],downcast='integer')

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16586030 entries, 0 to 16586029
Data columns (total 8 columns):
 #   Column        Dtype
---  ------        -----
 0   Ask           float32
 1   Bid           float32
 2   Vol_Bid_N     float32
 3   Vol_Ask_N     float32
 4   DateDelta1_N  float32
 5   Goal          int8
 6   Point         int8
 7   Point1        int8
dtypes: float32(5), int8(3)
memory usage: 363.8 MB
```

B [7]:
```
data['Point'].value_counts()
```

Out[7]:
```
0    16533604
1       52426
Name: Point, dtype: int64
```

B [8]:
```
number1 = data['Point'].value_counts()
number1
```

Out[8]:
```
0    16533604
1       52426
Name: Point, dtype: int64
```

B [9]:
```
number1.iloc[1]
```

Out[9]:
```
52426
```

B [10]:
```
# n_small - size of the numpy
n_small = number1.iloc[1]
n_small
```

Out[10]:
```
52426
```

B [11]:
```
type(n_small)
```

Out[11]:
```
numpy.int64
```

B [12]:
```
n_small = np.int32(n_small)
```

B [13]:
```
type(n_small)
```

Out[13]:
```
numpy.int32
```

B [14]:
```
data.shape[0]
```

Out[14]:
```
16586030
```

B [15]:
```
type(data.shape[0])
```

Out[15]:
```
int
```

B [16]:
```
n_small
```

Out[16]:
```
52426
```

B [17]:
```
data.head()
```

Out[17]:

	Ask	Bid	Vol_Bid_N	Vol_Ask_N	DateDelta1_N	Goal	Point	Point1
0	1.12160	1.12106	0.075	0.00937	0.00000	8	1	8
1	1.12160	1.12120	0.075	0.00937	0.21775	8	0	0
2	1.12160	1.12117	0.075	0.00937	0.06845	8	0	0
3	1.12161	1.12123	0.075	0.08400	0.03000	8	0	0
4	1.12161	1.12120	0.075	0.08400	0.00505	8	0	0

B [18]:
```
# The index of the first enter should be up than 200


data['Point'].loc[0]  = 0
data['Point1'].loc[0] = 0

data.head()
```

Out[18]:

	Ask	Bid	Vol_Bid_N	Vol_Ask_N	DateDelta1_N	Goal	Point	Point1
0	1.12160	1.12106	0.075	0.00937	0.00000	8	0	0
1	1.12160	1.12120	0.075	0.00937	0.21775	8	0	0
2	1.12160	1.12117	0.075	0.00937	0.06845	8	0	0
3	1.12161	1.12123	0.075	0.08400	0.03000	8	0	0
4	1.12161	1.12120	0.075	0.08400	0.00505	8	0	0

B [19]:
```
# cheking data

data['Point'].value_counts()
```

Out[19]:
```
0    16533605
1       52425
Name: Point, dtype: int64
```

B [20]:
```
# checking the first enter

for i in tqdm(range(0,data.shape[0])):
    if (data['Point'].iloc[i] == 1):
        print(i)
     #   print(data['Point'].index[i])
        break
    continue
```

```
  0%|          | 76/16586030 [00:00<03:37, 76278.32it/s]
```

```
76
```

B [21]:
```
# Delete all indexes which are less than 200

for i in tqdm(range(0,data.shape[0])):
    if (i > 200):
        break
    else:
        if (data['Point'].iloc[i] == 1):
            print(i)
         #   print(data['Point'].index[i])
    continue
```

```
  0%|          | 201/16586030 [00:00<02:47, 99008.23it/s]
```

```
76
108
109
```

B [22]:
```
print(data['Point1'].iloc[76], '\n')
print(data['Point1'].iloc[108], '\n')
print(data['Point1'].iloc[109])
```

```
10

5

11
```

B [23]:
```
data['Point'].loc[76]  = 0
data['Point1'].loc[76] = 0

data['Point'].loc[108]  = 0
data['Point1'].loc[108] = 0

data['Point'].loc[109]  = 0
data['Point1'].loc[109] = 0
```

B [24]:
```
# double check

for i in tqdm(range(0,data.shape[0])):
    if (data['Point'].iloc[i] == 1):
        print(i)
        break
    continue
```

```
  0%|          | 222/16586030 [00:00<04:57, 55663.29it/s]
```

```
222
```

B [25]:
```
number1 = data['Point'].value_counts()
number1
```

Out[25]:
```
0    16533608
1       52422
Name: Point, dtype: int64
```

B [26]:
```
n_small = number1.iloc[1]
n_small = np.int32(n_small)
n_small
```

Out[26]:
```
52422
```

Create numpy

B [27]:
```
# Delete 'Goal' column
columns1 = ['Goal']
data_for_array = data.drop(columns1, inplace=False, axis=1)

data_for_array.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16586030 entries, 0 to 16586029
Data columns (total 7 columns):
 #   Column        Dtype
---  ------        -----
 0   Ask           float64
 1   Bid           float64
 2   Vol_Bid_N     float64
 3   Vol_Ask_N     float64
 4   DateDelta1_N  float64
 5   Point         int64
 6   Point1        int64
dtypes: float64(5), int64(2)
memory usage: 885.8 MB
```

B [31]:
```
# create a three-dimensional data tensor: [n_small, N, 5] and
# one-dimensional array(vector) of answers [n_small, 1] with a value of the Probability vector for these planes,
# where N is the number of ticks for analysis at each step (those are looking backward)
```

B [28]:
```
n = data_for_array.shape[0]
p = data_for_array.shape[1]


data_a = np.array(data_for_array[:])

print("data_a:")
print(data_a[:4])
print('\n')
print(data_a[-4:])

print('\n', "Row number for training:  n = ", n, '\n',
      "Column number:                  p = ", p)
print(" numpy type:              ", type(data_a), '\n',

      "numpy size:              ", data_a.shape)
print('Example: data_a[1,0] = ', data_a[1,0])
```

```
data_a:
[[1.1216  1.12106 0.075   0.00937 0.      0.      0.     ]
 [1.1216  1.1212  0.075   0.00937 0.21775 0.      0.     ]
 [1.1216  1.12117 0.075   0.00937 0.06845 0.      0.     ]
 [1.12161 1.12123 0.075   0.084   0.03    0.      0.     ]]


[[1.12338    1.12333    0.394      0.157      0.00505    0.
  0.        ]
 [1.12337    1.12333    0.394      0.075      0.32845    0.
  0.        ]
 [1.12338    1.12333    0.394      0.20199999 0.2276     0.
  0.        ]
 [1.12338    1.12334    0.037      0.15       0.19405    0.
  0.        ]]

 Row number for training:  n =  16586030
 Column number:                 p =  7
 numpy type:               <class 'numpy.ndarray'>
 numpy size:               (16586030, 7)
Example: data_a[1,0] =  1.1216
```

we form the output matrix Y with the value of Probability vector

and imputs matrix data_b

B [29]:
```
N = 200   # N - the depth of analysis at each step of the training (how many ticks we look back) - about 10 minutes

# numpy size (L, N, 5)
L = n_small
print('Size of a three-dimensional data array for training: ', L, N, 5)
```

```
Size of a three-dimensional data array for training:  52422 200 5
```

B [30]:
```
data_b = np.zeros( (L, N, 5), dtype= 'float32' )

print('data_b:', '\n', data_b)
print('data_b shape:', '\n', data_b.shape)

# create a one-dimensional vector of answers - Probability vector
Y = np.zeros((L, 1), dtype= 'float32') # We predict valure of Probability vector

print('Y:', '\n', Y)
print('Y shape:', '\n', Y.shape)
```

```
data_b:
 [[[0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  ...
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]]

 [[0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  ...
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]]

 [[0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  ...
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]]
```

```
 ...

 [[0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  ...
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]]

 [[0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  ...
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]]

 [[0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  ...
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]]]
data_b shape:
 (52422, 200, 5)
Y:
 [[0.]
 [0.]
 [0.]
 ...
 [0.]
 [0.]
 [0.]]
Y shape:
 (52422, 1)
```

B [31]:
```
print('n = ', n, '\n')
print('n_small = ', n_small, '\n')
print('N = ', N, '\n')
print('L = ', L, '\n')
```

```
n =  16586030

n_small =  52422

N =  200

L =  52422
```

B [32]:
```
# fill the array for training data_b and
# the array of answers Y from the data_for_array

q=0

for k in tqdm(range(n)):
    if data_a[k,5] == 1:

        data_b[q, :, :] = data_a[k-N+1:k+1, 0:5]

        Y[q,0] = data_a[k,6]   # value of the Probability vector
        q = q +1
    else:
        continue

print(k) # end index value (for checking)

print('Счетчик плоскостей q= ', q, '\n')
print('data_b:', '\n', data_b)
print("Numpy size:        ", data_b.shape, '\n')

print( Y[:,:])

print(      "Numpy size:         ", Y.shape)
```

```
100%|██████████| 16586030/16586030 [00:10<00:00, 1649881.92it/s]
```

```
data_b:
 [[[1.12172    1.12125    0.15       0.056      0.15005   ]
  [1.12172    1.12134    0.075      0.056      0.2372    ]
  [1.12172    1.12135    0.075      0.056      0.00505   ]
  ...
  [1.12154    1.12151    0.075      0.075      1.        ]
  [1.12169    1.12151    0.075      0.075      0.2472    ]
  [1.12154    1.12151    0.075      0.075      0.0026    ]]

 [[1.12212    1.1221     0.075      0.15       0.0025    ]
  [1.12212    1.12211    0.075      0.15       0.01015   ]
  [1.12213    1.12211    0.075      0.15       0.0067    ]
  ...
  [1.12205    1.12199    0.075      0.075      0.00255   ]
  [1.12201    1.12198    0.075      0.075      0.4355    ]
  [1.12199    1.12197    0.082      0.075      0.0025    ]]

 [[1.12206    1.12203    0.382      0.075      0.0026    ]
  [1.12208    1.12203    0.30699998 0.375      0.00255   ]
  [1.12207    1.12203    0.232      0.262      0.1249    ]
  ...
  [1.12208    1.12206    0.075      0.15       0.00765   ]
  [1.1221     1.12208    0.01       0.187      1.        ]
  [1.12212    1.12209    0.157      0.281      0.0051    ]]

 ...

 [[1.12512    1.12508    0.412      0.15       0.0026    ]
  [1.12511    1.12508    0.712      0.075      0.0051    ]
  [1.12509    1.12508    0.15       0.075      0.06205   ]
  ...
  [1.125      1.12496    0.19500001 0.075      0.00515   ]
  [1.125      1.12497    0.01       0.15       0.0223    ]
  [1.125      1.12496    0.27       0.225      0.0115    ]]

 [[1.12301    1.12297    0.15       0.607      0.2692    ]
  [1.12301    1.12296    0.412      0.71999997 0.1116    ]
  [1.12301    1.12297    0.075      0.345      0.051     ]
  ...
  [1.12287    1.12283    0.225      0.892      0.02965   ]
  [1.12286    1.12283    0.225      0.022      0.03155   ]
  [1.12286    1.12284    0.075      0.06       0.0645    ]]

 [[1.12375    1.12373    0.075      0.019      0.01545   ]
  [1.12376    1.12373    0.075      0.056      0.02615   ]
  [1.12374    1.12373    0.097      0.026      0.1048    ]
  ...
  [1.12374    1.12372    0.21       0.075      0.0106    ]
  [1.12373    1.12371    0.131      0.082      0.06035   ]
  [1.12373    1.12372    0.075      0.082      0.16545   ]]]
Numpy size:         (52422, 200, 5)
```

```
[[ 6.]
 [10.]
 [10.]
 ...
 [ 4.]
 [ 7.]
 [ 9.]]
Numpy size:          (52422, 1)
```

B [39]:
```
# double check
```

B [33]:
```
np.set_printoptions(suppress=True)
data_a[16580890,:]
```

Out[33]:
```
array([1.12373, 1.12372, 0.075  , 0.082  , 0.16545, 1.     , 9.     ])
```

B [34]:
```
data_b[n_small-1,N-1,:]
```

Out[34]:
```
array([1.12373, 1.12372, 0.075  , 0.082  , 0.16545], dtype=float32)
```

B [42]:
```
# The upper two raws Should be the same
```

B [37]:
```
# index array
for i in tqdm(range(n)):
    if (data['Point'].iloc[i] == 1):
        print(i)
        continue

    continue
```

```
16557125
16557126
16557127
16557128
16557284
16557492
16557494
16557495
16557496
100%|██████████| 16586030/16586030 [02:51<00:00, 96987.49it/s]
16570373
16570479
16570481
16570482
16579159
16580890
```

B [36]:
```
z = random.randint(0, 199)

print('z= ',z)


print(data_a[16580890-z, :], '\n')


print(data_b[n_small-1, N-1-z, :], '\n')
```

```
z=  125
[1.12369 1.12367 0.15    0.01    0.01755 0.      0.     ]

[1.12369 1.12367 0.15    0.01    0.01755]
```

B [37]:
```
print(data_a[16580886, :], '\n')


print(data_b[n_small-1, N-1-4, :], '\n')
```

```
[1.12375 1.12373 0.075   0.01    0.00765 0.      0.     ]

[1.12375 1.12373 0.075   0.01    0.00765]
```

B [38]:
```
data.tail(-16580886)
```

Out[38]:

	Ask	Bid	Vol_Bid_N	Vol_Ask_N	DateDelta1_N	Goal	Point	Point1
16580886	1.12375	1.12373	0.075	0.010	0.00765	8	0	0
16580887	1.12376	1.12372	0.135	0.195	0.00500	8	0	0
16580888	1.12374	1.12372	0.210	0.075	0.01060	8	0	0
16580889	1.12373	1.12371	0.131	0.082	0.06035	8	0	0
16580890	1.12373	1.12372	0.075	0.082	0.16545	9	1	9
...	...	...	...	...	...	...	...	...
16586025	1.12337	1.12333	0.506	0.075	0.00505	8	0	0
16586026	1.12338	1.12333	0.394	0.157	0.00505	8	0	0
16586027	1.12337	1.12333	0.394	0.075	0.32845	8	0	0
16586028	1.12338	1.12333	0.394	0.202	0.22760	8	0	0
16586029	1.12338	1.12334	0.037	0.150	0.19405	8	0	0

B [39]:
```
# We print Min and Max of all normalised columns of signs - All should be in the range [0; 1]
# Check that the matrices are normalised and there are no values> 1

mmx  = data_b[:,:,2].max()
mmn  = data_b[:,:,2].min()
print('Vol_Bid_N', '\t', mmx, '\t', mmn)

mmx  = data_b[:,:,3].max()
mmn  = data_b[:,:,3].min()
print('Vol_Ask_N', '\t', mmx, '\t', mmn)

mmx  = data_b[:,:,4].max()
mmn  = data_b[:,:,4].min()
print('DateDelta1_N', '\t', mmx, '\t', mmn)
```

```
Vol_Bid_N       1.0     0.0009999999
Vol_Ask_N       1.0     0.0009999999
DateDelta1_N    1.0     0.00245
```

B [84]:
```
# It's OK!
```

B [40]:
```
L
```

Out[40]: 52422

creation datasets for training and testing

B [41]:
```
# We allocate about 80% of the data (two-dimensional subarrays) for training and about 20% of
# the data (two-dimensional subarrays) for testing
# n = data_a.shape [0] # new number of lines
train_start = 0
train_end = int(np.floor(0.8*L))
print(L, train_end)
```

```
52422 41937
```

B [42]:
```
test_start = train_end + 1
test_end = L
print(test_start, test_end)
```

```
41938 52422
```

B [43]:
```
print("numpy size:        ", data_b.shape, '\n')
```

```
numpy size:         (52422, 200, 5)
```

B [44]:
```
print(test_end-train_end)   # dataset size for te для testing
```

```
10485
```

B [45]:
```
# form X_train and X_test
# Divide data_b into two unequal parts: for training (X_train) and for test (X_test)

X_train = np.zeros( ( train_end, N, 5) , dtype= 'float32')
X_test  = np.zeros( ( test_end-train_end, N, 5) , dtype= 'float32')

for zz in tqdm(range(test_end)):
    if zz < train_end:
        X_train[zz,:,:] = data_b[zz,:,:]
    else:
        X_test[zz-train_end,:,:]  = data_b[zz,:,:]

# print('X_train:', '\n', X_test)
print(" Train numpy size::        ", X_train.shape, '\n')
print('\n')

# print('X_test:', '\n', X_test)
print(" Test numpy size::        ", X_test.shape, '\n')
```

```
100%|██████████| 52422/52422 [00:00<00:00, 317334.08it/s]
 Train numpy size::         (41937, 200, 5)


 Test numpy size::         (10485, 200, 5)
```

B [46]:
```
# Visual checking the correctness of filling
print(X_train[:5,N-1,:], '\n')
print(X_train[-5:,N-1,:], '\n')
print(X_test[:5,N-1,:], '\n')
print(X_test[-5:,N-1,:])
```

```
[[1.12154 1.12151 0.075   0.075   0.0026 ]
 [1.12199 1.12197 0.082   0.075   0.0025 ]
 [1.12212 1.12209 0.157   0.281   0.0051 ]
 [1.12194 1.12191 0.075   0.075   0.03285]
 [1.12238 1.12236 0.094   0.075   0.005  ]]

[[1.08625    1.08621    0.45       0.62600005 0.1378    ]
 [1.08623    1.08621    0.075      0.075      0.18385   ]
 [1.08614    1.08612    0.019      0.15       0.01565   ]
 [1.08614    1.08612    0.019      0.525      0.0026    ]
 [1.0861     1.08607    0.01       0.15       0.01905   ]]

[[1.08609 1.08607 0.375   0.075   0.25215]
 [1.08573 1.0857  0.3     0.026   0.0051 ]
 [1.08572 1.0857  0.3     0.019   0.00505]
 [1.08559 1.08556 0.075   0.326   0.0544 ]
 [1.08539 1.08536 0.319   0.075   0.00505]]

[[1.125   1.12497 0.01    0.075   0.03065]
 [1.125   1.12497 0.01    0.15    0.0223 ]
 [1.125   1.12496 0.27    0.225   0.0115 ]
 [1.12286 1.12284 0.075   0.06    0.0645 ]
 [1.12373 1.12372 0.075   0.082   0.16545]]
```

Creation an answer matrix for the training (y_train) and test (y_test) parts

B [48]:
```
# form y_train and y_test
# Divide Y into two unequal parts: for training (y_train) and for test (y_test)

y_train = np.zeros( ( train_end, 1) , dtype= 'float32')
y_test  = np.zeros( ( test_end-train_end, 1) , dtype= 'float32')

for zz in tqdm(range(test_end)):
    if zz < train_end:
        y_train[zz,0] = Y[zz,0]
    else:
        y_test[zz-train_end,0]  = Y[zz,0]

# print('y_train:', '\n', y_test)
print(" Outputs train numpy size::        ", y_train.shape, '\n')
print('\n')

# print('X_test:', '\n', X_test)
print(" Outputs test numpy size::        ", y_test.shape, '\n')
```

```
100%|██████████| 52422/52422 [00:00<00:00, 1376715.18it/s]
 Outputs train numpy size::         (41937, 1)


 Outputs test numpy size::         (10485, 1)
```

B [49]:
```
# Visual checking the correctness of filling

print(y_train[:5,0], '\n')
print(y_train[-5:,0], '\n')
print(y_test[:5,0], '\n')
print(y_test[-5:,0])

print(" Outputs train numpy size:         ", y_train.shape ,'\n')
print(" Outputs test numpy size:          ", y_test.shape ,'\n')
```

```
[ 6. 10. 10. 10.  4.]

[4. 4. 4. 4. 4.]

[4. 3. 3. 3. 4.]

[4. 4. 4. 7. 9.]
 Outputs train numpy size:          (41937, 1)

 Outputs test numpy size:          (10485, 1)
```

B [50]:
```
data[data['Point'] == 1 ]
```

Out[50]:

	Ask	Bid	Vol_Bid_N	Vol_Ask_N	DateDelta1_N	Goal	Point	Point1
222	1.12154	1.12151	0.075	0.075	0.00260	6	1	6
2078	1.12199	1.12197	0.082	0.075	0.00250	10	1	10
4469	1.12212	1.12209	0.157	0.281	0.00510	10	1	10
6894	1.12194	1.12191	0.075	0.075	0.03285	10	1	10
7652	1.12238	1.12236	0.094	0.075	0.00500	4	1	4
...	...	...	...	...	...	...	...	...
16570479	1.12500	1.12497	0.010	0.075	0.03065	4	1	4
16570481	1.12500	1.12497	0.010	0.150	0.02230	4	1	4
16570482	1.12500	1.12496	0.270	0.225	0.01150	4	1	4
16579159	1.12286	1.12284	0.075	0.060	0.06450	7	1	7
16580890	1.12373	1.12372	0.075	0.082	0.16545	9	1	9

52422 rows × 8 columns

B [51]:
```
# print X_train, y_train, X_test u y_test

print('Data for training:')
print("X_train:")
print("Type: ", X_train.dtype,  type(X_train), "; Size: ", X_train.shape)

print("\n","y_train:")
print("Type: ", y_train.dtype,  type(y_train), "; Size: ", y_train.shape)

print('Data for testing:')
print("\n","X_test:")
print("Type: ", X_test.dtype, type(X_test), "; Size: ", X_test.shape)

print("\n","y_test:")
print("Type: ", y_test.dtype,  type(y_test), "; Size: ", y_test.shape)
```

```
Data for training:
X_train:
Type:  float32 <class 'numpy.ndarray'> ; Size:  (41937, 200, 5)

 y_train:
Type:  float32 <class 'numpy.ndarray'> ; Size:  (41937, 1)
Data for testing:

 X_test:
Type:  float32 <class 'numpy.ndarray'> ; Size:  (10485, 200, 5)

 y_test:
Type:  float32 <class 'numpy.ndarray'> ; Size:  (10485, 1)
```

Creation of the Probability vector for train and test sets

B [52]:
```
data_for_array_short = data_for_array[data_for_array['Point'] == 1]

data_for_array_short.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 52422 entries, 222 to 16580890
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Ask           52422 non-null  float64
 1   Bid           52422 non-null  float64
 2   Vol_Bid_N     52422 non-null  float64
 3   Vol_Ask_N     52422 non-null  float64
 4   DateDelta1_N  52422 non-null  float64
 5   Point         52422 non-null  int64
 6   Point1        52422 non-null  int64
dtypes: float64(5), int64(2)
memory usage: 3.2 MB
```

B [53]:
```
indexes_to_keep = set(np.array(range(0,train_end+1)))

data_for_array_short_train  = data_for_array_short.take(list(indexes_to_keep))

print('array_short_train.dtypes', data_for_array_short_train.dtypes)

data_for_array_short_train.info()
```

```
array_short_train.dtypes Ask             float64
Bid             float64
Vol_Bid_N       float64
Vol_Ask_N       float64
DateDelta1_N    float64
Point             int64
Point1            int64
dtype: object
<class 'pandas.core.frame.DataFrame'>
Int64Index: 41938 entries, 222 to 11341974
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Ask           41938 non-null  float64
 1   Bid           41938 non-null  float64
 2   Vol_Bid_N     41938 non-null  float64
 3   Vol_Ask_N     41938 non-null  float64
 4   DateDelta1_N  41938 non-null  float64
 5   Point         41938 non-null  int64
 6   Point1        41938 non-null  int64
dtypes: float64(5), int64(2)
memory usage: 2.6 MB
```

B [54]:
```
indexes_to_keep = set(np.array(range(test_start,L)))

data_for_array_short_test  = data_for_array_short.take(list(indexes_to_keep))

print('array_short_test.dtypes', data_for_array_short_test.dtypes)

data_for_array_short_test.info()
```

```
array_short_test.dtypes Ask             float64
Bid             float64
Vol_Bid_N       float64
Vol_Ask_N       float64
DateDelta1_N    float64
Point             int64
Point1            int64
dtype: object
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10484 entries, 11350534 to 16580890
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Ask           10484 non-null  float64
 1   Bid           10484 non-null  float64
 2   Vol_Bid_N     10484 non-null  float64
 3   Vol_Ask_N     10484 non-null  float64
 4   DateDelta1_N  10484 non-null  float64
 5   Point         10484 non-null  int64
 6   Point1        10484 non-null  int64
dtypes: float64(5), int64(2)
memory usage: 655.2 KB
```

B [55]:
```
# unique value in 'Point1' train set
data_for_array_short_train['Point1'].value_counts()
```

Out[55]:
```
12    10478
4      9942
3      7469
13     6958
6      1000
11      999
10      999
5       999
8       756
2       698
14      675
9       291
7       269
15      231
1       169
0         5
Name: Point1, dtype: int64
```

B [56]:
```
# unique value in 'Point1' in test set:

data_for_array_short_test['Point1'].value_counts()
```

Out[56]:
```
12    2871
4     2641
13    2124
3     1651
14     343
8      243
2      234
9      148
7      146
15      69
1       14
Name: Point1, dtype: int64
```

Probability vector на (Trend/Flat)

binary classification

0, 1, 2, 3, 4, 12, 13, 14, 15, 16 = Trend = (1)

5, 6, 7, 8, 9, 10, 11 = Flat = (0)

B [57]:
```
for i in tqdm(range(train_end)):
    if ((y_train[i,0] <= 4) | (y_train[i,0] >= 12)):
        y_train[i,0] = 1

    if ((y_train[i,0] >= 5) & (y_train[i,0] <= 11)):
        y_train[i,0] = 0
    else:
        continue

for i in tqdm(range(0, test_end-test_start+1)):
    if ((y_test[i,0] <= 4) | (y_test[i,0] >= 12)):
        y_test[i,0] = 1

    if ((y_test[i,0] >= 5) & (y_test[i,0] <= 11)):
        y_test[i,0] = 0
    else:
        continue
```

```
100%|██████████| 41937/41937 [00:00<00:00, 129858.55it/s]
100%|██████████| 10485/10485 [00:00<00:00, 138254.20it/s]
```

B [58]:
```
# checking

for i in tqdm(range(train_end)):
    if (y_train[i,0] >= 2):
        print('Allarm')
        print(i)
    else:
        continue

for i in tqdm(range(0, test_end-test_start+1)):
    if (y_test[i,0] >= 2):
        print('Allarm')
        print(i)
    else:
        continue
```

```
100%|██████████| 41937/41937 [00:00<00:00, 412248.47it/s]
100%|██████████| 10485/10485 [00:00<00:00, 498766.93it/s]
```

B [69]:
```
# No Allarms
```

B [69]:
```
# We have two categories:
# Trend: (1)
# Flat : (0)
```

B [71]:
```
classes=['0 is Flat', '1 is Trend']
```

B [ ]:

Convolutional Neural Network

B [72]:
```
# Create a consistent model
model = keras.Sequential([

# First convolutional layer
# activation function = "relu"
# initial weights are initialized randomly using the normal distribution - kernel_initializer = "normal"
# Incoming data: matrix with size 200 (time) * 5 (signs). In this model, time is inside in one of the signs
# Take the convolution window 15 (it is possible odd: 9, 11, etc. - Odd)

    keras.layers.Conv1D(200, 15, padding='same', input_shape=(200, 5), activation=tf.nn.relu, kernel_initializer="normal"),

    keras.layers.MaxPooling1D(5),

    keras.layers.Dropout(0.25),


    keras.layers.Conv1D(400, 15, padding='same', activation=tf.nn.relu, kernel_initializer="normal"),

    keras.layers.GlobalMaxPooling1D(),

    keras.layers.Dropout(0.25),

    keras.layers.Flatten(),

    keras.layers.Dense(720, activation=tf.nn.relu, kernel_initializer="normal"),

    keras.layers.Dense(72, activation=tf.nn.relu, kernel_initializer="normal"),

    keras.layers.Dropout(0.5),

    keras.layers.Dense(1, activation=tf.nn.sigmoid)


])

print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 200, 200)          15200
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 40, 200)           0
_________________________________________________________________
dropout (Dropout)            (None, 40, 200)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 40, 400)           1200400
_________________________________________________________________
global_max_pooling1d (Global (None, 400)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 400)               0
_________________________________________________________________
flatten (Flatten)            (None, 400)               0
_________________________________________________________________
dense (Dense)                (None, 720)               288720
_________________________________________________________________
dense_1 (Dense)              (None, 72)                51912
_________________________________________________________________
dropout_2 (Dropout)          (None, 72)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 73
=================================================================
Total params: 1,556,305
Trainable params: 1,556,305
Non-trainable params: 0
_________________________________________________________________
None
```

Compile the model

B [73]:
```
# Training method: optimizer - "adam"
# adam (Adaptive Moment Estimation) Optimizer


# The metric by which we will perform optimization (Loss function) is binary_crossentropy


model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['accuracy','Precision','Recall','mse','mae','binary_crossentropy'])
```

Callback for ModelCheckpoint -memorising learning outcomes after each epoch

B [74]:
```
from tensorflow.keras.callbacks import ModelCheckpoint


callbacks =[(ModelCheckpoint('convol_200_3_1week-{epoch:02d}-{val_loss:.4f}.hdf5', monitor='val_acc', save_best_only=True))]
```

B [26]:
```
model.fit?
```

B [26]:
```
model.compile?
```

Trainig NN

B [75]:
```
history = model.fit(X_train, y_train, batch_size=30, epochs=15, validation_split=0.3, shuffle=True, verbose=1)

# size batch_size = 30,
# the number of epochs = 15
# validation_split = 0.3 (30%)- the part of the sample that will be used as verification
# verbose = 2 - printing information in the learning process
```

```
Epoch 1/15
979/979 [==============================] - 57s 58ms/step - loss: 0.3787 - accuracy: 0.8675 - precision: 0.8791 - recall: 0.9791
- mse: 0.1079 - mae: 0.2174 - binary_crossentropy: 0.3787 - val_loss: 0.4789 - val_accuracy: 0.7299 - val_precision: 0.9280 - v
al_recall: 0.7684 - val_mse: 0.1546 - val_mae: 0.3429 - val_binary_crossentropy: 0.4789
Epoch 2/15
979/979 [==============================] - 60s 62ms/step - loss: 0.3229 - accuracy: 0.8912 - precision: 0.9090 - recall: 0.9692
- mse: 0.0902 - mae: 0.1857 - binary_crossentropy: 0.3229 - val_loss: 0.4689 - val_accuracy: 0.7216 - val_precision: 0.9275 - v
al_recall: 0.7592 - val_mse: 0.1544 - val_mae: 0.2943 - val_binary_crossentropy: 0.4689
Epoch 3/15
979/979 [==============================] - 59s 60ms/step - loss: 0.3240 - accuracy: 0.8911 - precision: 0.9086 - recall: 0.9696
- mse: 0.0898 - mae: 0.1844 - binary_crossentropy: 0.3240 - val_loss: 0.4341 - val_accuracy: 0.7674 - val_precision: 0.9270 - v
al_recall: 0.8133 - val_mse: 0.1389 - val_mae: 0.2723 - val_binary_crossentropy: 0.4341
Epoch 4/15
979/979 [==============================] - 58s 59ms/step - loss: 0.3155 - accuracy: 0.8944 - precision: 0.9107 - recall: 0.9713
- mse: 0.0876 - mae: 0.1810 - binary_crossentropy: 0.3155 - val_loss: 0.4368 - val_accuracy: 0.7668 - val_precision: 0.9273 - v
al_recall: 0.8123 - val_mse: 0.1398 - val_mae: 0.2812 - val_binary_crossentropy: 0.4368
Epoch 5/15
979/979 [==============================] - 58s 59ms/step - loss: 0.3122 - accuracy: 0.8955 - precision: 0.9111 - recall: 0.9723
- mse: 0.0863 - mae: 0.1772 - binary_crossentropy: 0.3122 - val_loss: 0.4900 - val_accuracy: 0.7332 - val_precision: 0.9277 - v
al_recall: 0.7726 - val_mse: 0.1609 - val_mae: 0.3101 - val_binary_crossentropy: 0.4900
Epoch 6/15
979/979 [==============================] - 58s 59ms/step - loss: 0.3118 - accuracy: 0.8946 - precision: 0.9104 - recall: 0.9715
- mse: 0.0868 - mae: 0.1781 - binary_crossentropy: 0.3118 - val_loss: 0.4125 - val_accuracy: 0.8153 - val_precision: 0.9278 - v
al_recall: 0.8685 - val_mse: 0.1288 - val_mae: 0.2624 - val_binary_crossentropy: 0.4125
Epoch 7/15
979/979 [==============================] - 58s 59ms/step - loss: 0.3055 - accuracy: 0.8953 - precision: 0.9103 - recall: 0.9726
- mse: 0.0851 - mae: 0.1731 - binary_crossentropy: 0.3055 - val_loss: 0.4016 - val_accuracy: 0.8134 - val_precision: 0.9278 - v
al_recall: 0.8662 - val_mse: 0.1252 - val_mae: 0.2418 - val_binary_crossentropy: 0.4016
Epoch 8/15
979/979 [==============================] - 62s 63ms/step - loss: 0.3095 - accuracy: 0.8915 - precision: 0.9065 - recall: 0.9723
- mse: 0.0865 - mae: 0.1758 - binary_crossentropy: 0.3095 - val_loss: 0.4232 - val_accuracy: 0.8108 - val_precision: 0.9283 - v
al_recall: 0.8626 - val_mse: 0.1318 - val_mae: 0.2698 - val_binary_crossentropy: 0.4232
Epoch 9/15
979/979 [==============================] - 59s 60ms/step - loss: 0.3112 - accuracy: 0.8917 - precision: 0.9070 - recall: 0.9714
- mse: 0.0872 - mae: 0.1785 - binary_crossentropy: 0.3112 - val_loss: 0.5877 - val_accuracy: 0.6966 - val_precision: 0.9276 - v
al_recall: 0.7299 - val_mse: 0.1991 - val_mae: 0.3364 - val_binary_crossentropy: 0.5877
```

```
Epoch 10/15
979/979 [==============================] - 59s 60ms/step - loss: 0.3035 - accuracy: 0.8962 - precision: 0.9107 - recall: 0.9732
- mse: 0.0850 - mae: 0.1719 - binary_crossentropy: 0.3035 - val_loss: 0.4762 - val_accuracy: 0.7670 - val_precision: 0.9275 - v
al_recall: 0.8123 - val_mse: 0.1534 - val_mae: 0.2821 - val_binary_crossentropy: 0.4762
Epoch 11/15
979/979 [==============================] - 58s 59ms/step - loss: 0.3025 - accuracy: 0.8961 - precision: 0.9083 - recall: 0.9757
- mse: 0.0845 - mae: 0.1712 - binary_crossentropy: 0.3025 - val_loss: 0.4939 - val_accuracy: 0.7713 - val_precision: 0.9274 - v
al_recall: 0.8174 - val_mse: 0.1554 - val_mae: 0.3017 - val_binary_crossentropy: 0.4939
Epoch 12/15
979/979 [==============================] - 58s 60ms/step - loss: 0.3016 - accuracy: 0.8960 - precision: 0.9094 - recall: 0.9750
- mse: 0.0843 - mae: 0.1715 - binary_crossentropy: 0.3016 - val_loss: 0.5422 - val_accuracy: 0.7357 - val_precision: 0.9265 - v
al_recall: 0.7767 - val_mse: 0.1766 - val_mae: 0.2920 - val_binary_crossentropy: 0.5422
Epoch 13/15
979/979 [==============================] - 58s 59ms/step - loss: 0.3006 - accuracy: 0.8965 - precision: 0.9100 - recall: 0.9747
- mse: 0.0838 - mae: 0.1690 - binary_crossentropy: 0.3006 - val_loss: 0.5481 - val_accuracy: 0.7426 - val_precision: 0.9272 - v
al_recall: 0.7841 - val_mse: 0.1723 - val_mae: 0.3014 - val_binary_crossentropy: 0.5481
Epoch 14/15
979/979 [==============================] - 61s 62ms/step - loss: 0.2943 - accuracy: 0.8979 - precision: 0.9118 - recall: 0.9744
- mse: 0.0819 - mae: 0.1668 - binary_crossentropy: 0.2943 - val_loss: 0.7359 - val_accuracy: 0.6840 - val_precision: 0.9267 - v
al_recall: 0.7159 - val_mse: 0.2270 - val_mae: 0.3615 - val_binary_crossentropy: 0.7359
Epoch 15/15
979/979 [==============================] - 58s 59ms/step - loss: 0.2982 - accuracy: 0.8986 - precision: 0.9105 - recall: 0.9763
- mse: 0.0827 - mae: 0.1685 - binary_crossentropy: 0.2982 - val_loss: 0.4776 - val_accuracy: 0.7877 - val_precision: 0.9277 - v
al_recall: 0.8363 - val_mse: 0.1490 - val_mae: 0.2541 - val_binary_crossentropy: 0.4776
```

Evaluation

B [76]:
```
# Evaluate the quality of network training on test data (which the network did NOT see)
scores = model.evaluate(X_test, y_test, verbose=0)
```

B [77]:
```
scores
```

Out[77]:
```
[0.5651160478591919,
 0.70701003074646,
 0.9628432989120483,
 0.7189384698867798,
 0.18841664493083954,
 0.33946356177330017,
 0.5651160478591919]
```

B [78]:
```
print("Accuracy: %.3f%%" % (scores[1]*100))
```

```
Accuracy: 70.701%
```

B [79]:
```
print("Precision: %.3f%%" % (scores[2]*100))
```

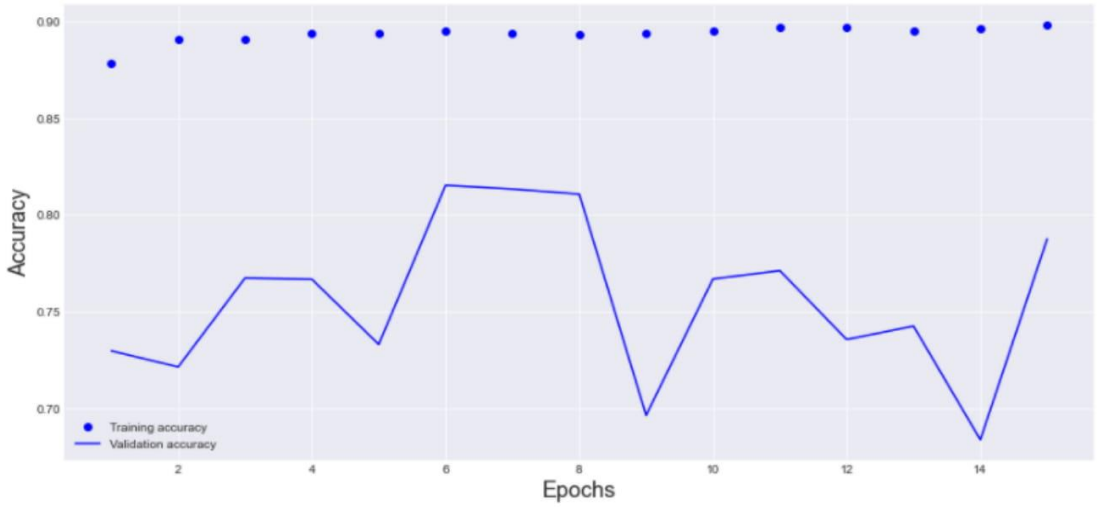
```
Precision: 96.284%
```

B [80]:
```
print("Recall: %.3f%%" % (scores[3]*100))
```

```
Recall: 71.894%
```

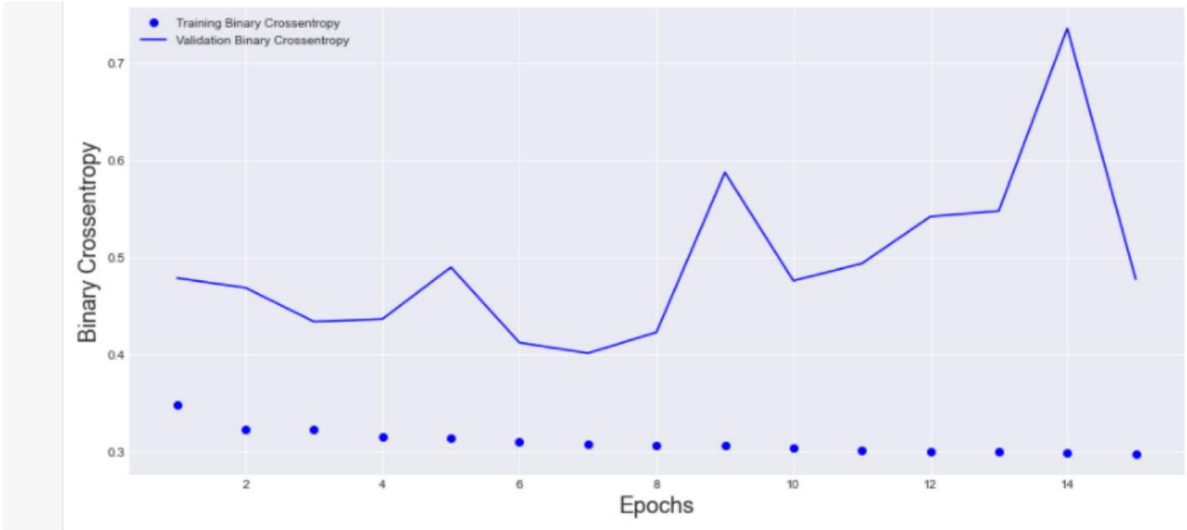
Accuracy graphs during training

B [81]:
```
history_dict = history.history
plt.style.use('seaborn-darkgrid')
accuracy_values = history_dict['accuracy']

val_accuracy_values = history_dict['val_accuracy']

epochs = range(1, len(accuracy_values) + 1)
plt.figure(num=1, figsize=(15,7))
plt.plot(epochs, accuracy_values, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy_values, 'b', label='Validation accuracy')
plt.xlabel('Epochs', size=18)
plt.ylabel('Accuracy', size=18)
plt.legend()
plt.show()
```

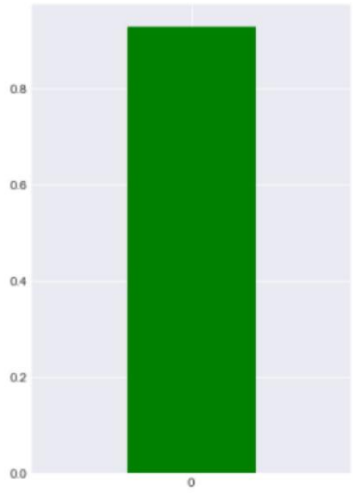


Binary Crossentropy graphs during training

B [87]:
```
history_dict = history.history
plt.style.use('seaborn-darkgrid')
accuracy_values = history_dict['binary_crossentropy']

val_accuracy_values = history_dict['val_binary_crossentropy']

epochs = range(1, len(accuracy_values) + 1)
plt.figure(num=1, figsize=(15,7))
plt.plot(epochs, accuracy_values, 'bo', label='Training Binary Crossentropy')
plt.plot(epochs, val_accuracy_values, 'b', label='Validation Binary Crossentropy')
plt.xlabel('Epochs', size=18)
plt.ylabel('Binary Crossentropy', size=18)
plt.legend()
plt.show()
```



In [89]:
```
index = random.randint(0, Y_test.shape[0])
print('Right answer: ', Y_test[index])
#plt.imshow(toimage(X_test[index]))

x = X_test[index]
x = np.expand_dims(x, axis=0)

prediction = model.predict(x)
sample = x

ans = np.argmax(prediction)

fig = plt.figure(figsize=(10,7))

ax = fig.add_subplot(1, 2, 2)
bar_list = ax.bar(np.arange(1), prediction[0], align='center')
bar_list[ans].set_color('g')
ax.set_xticks(np.arange(1))
ax.set_xlim([-1, 1])
ax.grid('on')

plt.show()
print('Predicted answer/Предсказанный ответ: {}'.format(classes[ans]), "\n ")
print('Right answer/Правильный ответ: {}'.format(classes[Y_test[index][0].astype(int)]) )
print(classes)
```




```
Предсказанный ответ: 0 is Flat

Правильный ответ: 0 is Flat
['0 is Flat', '1 is Trend']
```

Appendix XIV.

Evaluation of the classification task by

Recurrent Neural Networks (GRU and LSTM)

Tick Data with Dukastcopy

Data for 2020 year

Trend(1) or Flat(0)

Normalisation of volumes - based on a maximum of 10 lots (the main number of lots). The interval (from 10 to 5000 lots) are equated to 10 lots (those to 1 after normalisation)

We normalise the intervals between ticks - based on a maximum of 20,000 milliseconds (20 seconds) (the main number of intervals). The interval (from 20,000) are equated to 20,000 milliseconds (those to 1 after normalisation)

```
# TensorFlow and tf.keras

import tensorflow as tf
from tensorflow import keras

import numpy as np
import pandas as pd

import random

import matplotlib.pyplot as plt
%matplotlib inline
```

```
print(tf.__version__)
```

```
# Progress bar
from tqdm import tqdm
```

```
# COLABORATE
!cat /proc/driver/nvidia/gpus/0000:00:04.0/information
```

```
# Set seed to repeat results for random number generator
# tk we use stochastic optimization methods
np.random.seed(42)
```

Data

```
# COLABORATE
from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))
```

```
from google.colab import drive
drive.mount('/content/grive', force_remount=True)
```

```
# COLABORATE
!ls
```

```
data = pd.read_csv('/Users/vital/Python_Code/_PhD_Code/2021-04-April/Experiment-2/Results3/EURO_data_raw_common_norm.csv')

print(data.head(n=5))
print(data[-5:])
```

B [ ]:
```
data.info()
```

B [ ]:
```
# downsising

data['Ask']          =pd.to_numeric(data['Ask'],downcast='float')
data['Bid']          =pd.to_numeric(data['Bid'],downcast='float')
data['Vol_Bid_N']    =pd.to_numeric(data['Vol_Bid_N'],downcast='float')
data['Vol_Ask_N']    =pd.to_numeric(data['Vol_Ask_N'],downcast='float')
data['DateDelta1_N'] =pd.to_numeric(data['DateDelta1_N'],downcast='float')

data['Goal']         = pd.to_numeric(data['Goal'],downcast='integer')
data['Point']        = pd.to_numeric(data['Point'],downcast='integer')
data['Point1']       = pd.to_numeric(data['Point1'],downcast='integer')

data.info()
```

B [ ]:
```
data['Point'].value_counts()
```

B [ ]:
```
number1 = data['Point'].value_counts()
number1
```

B [ ]:
```
number1.iloc[1]
```

B [ ]:
```
n_small = number1.iloc[1]
n_small
```

B [ ]:
```
type(n_small)
```

B [ ]:
```
n_small = np.int32(n_small)
```

B [ ]:
```
type(n_small)
```

B [ ]:
```
data.shape[0]
```

B [ ]:
```
type(data.shape[0])
```

B [ ]:
```
n_small
```

B [ ]:
```
data.head()
```

B [ ]:
```
data['Point'].loc[0]  = 0
data['Point1'].loc[0] = 0

data.head()
```

B [ ]:
```
data['Point'].value_counts()
```

B [ ]:
```
for i in tqdm(range(0,data.shape[0])):
    if (data['Point'].iloc[i] == 1):
        print(i)
     #   print(data['Point'].index[i])
        break
    continue
```

B [ ]:
```
for i in tqdm(range(0,data.shape[0])):
    if (i > 200):
        break
    else:
        if (data['Point'].iloc[i] == 1):
            print(i)
         #   print(data['Point'].index[i])
    continue
```

B [ ]:
```
print(data['Point1'].iloc[76], '\n')
print(data['Point1'].iloc[108], '\n')
print(data['Point1'].iloc[109])
```

```
data['Point'].loc[76]  = 0
data['Point1'].loc[76] = 0

data['Point'].loc[108]  = 0
data['Point1'].loc[108] = 0

data['Point'].loc[109]  = 0
data['Point1'].loc[109] = 0
```

```
for i in tqdm(range(0,data.shape[0])):
    if (data['Point'].iloc[i] == 1):
        print(i)
        break
    continue
```

```
number1 = data['Point'].value_counts()
number1
```

```
n_small = number1.iloc[1]
n_small = np.int32(n_small)
n_small
```

Create numpy

```
# del 'Goal'
columns1 = ['Goal']
data_for_array = data.drop(columns1, inplace=False, axis=1)

data_for_array.info()
```

```
n = data_for_array.shape[0]
p = data_for_array.shape[1]

data_a = np.array(data_for_array[:])

print("data_a:")
print(data_a[:4])
print('\n')
print(data_a[-4:])

print('\n', "Row number for training:  n = ", n, '\n',
      "Column number:                 p = ", p)
print(" numpy type:              ", type(data_a), '\n',

      "numpy size:              ", data_a.shape)
print('Example: data_a[1,0] = ', data_a[1,0])
```

we form the output matrix Y with the value of Probability vector

and imputs matrix data_b

```
N = 200
L = n_small
print('Size of a three-dimensional data array for training: ', L, N, 5)
```

```
data_b = np.zeros( (L, N, 5), dtype= 'float32' )

print('data_b:', '\n', data_b)
print('data_b shape:', '\n', data_b.shape)

# create a one-dimensional vector of answers - Probability vector
Y = np.zeros((L, 1), dtype= 'float32') # We predict value of Probability vector

print('Y:', '\n', Y)
print('Y shape:', '\n', Y.shape)
```

```
print('n = ', n, '\n')
print('n_small = ', n_small, '\n')
print('N = ', N, '\n')
print('L = ', L, '\n')
```

```
# fill the array for training data_b and
# the array of answers Y from the data_for_array

q=0

for k in tqdm(range(n)):
    if data_a[k,5] == 1:
        data_b[q, :, :] = data_a[k-N+1:k+1, 0:5]

        Y[q,0] = data_a[k,6]   # value of the Probability vector
        q = q +1
    else:
        continue

print(k) # end index value (for control)

print('number of plates q= ', q, '\n')
print('data_b:', '\n', data_b)
print("Numpy size:        ", data_b.shape, '\n')

print( Y[:,:])

print(      "Numpy size:         ", Y.shape)
```

```
# checking
```

```
np.set_printoptions(suppress=True)
data_a[16580890,:]        # the last row in data_a
```

```
data_b[n_small-1,N-1,:]
```

```
for i in tqdm(range(n)):
    if (data['Point'].iloc[i] == 1):
        print(i)
        continue

    continue
```

```
# double ckeck of Probabilities vector

print(data_a[16580890,6],'\n')

print(data_for_array['Point1'].iloc[16580890])

print('the numbers must be the same')
```

```
z = random.randint(0, 199)

print('z= ',z)


print(data_a[16580890-z, :], '\n')


print(data_b[n_small-1, N-1-z, :], '\n')
```

```
print(data_a[16580886, :], '\n')


print(data_b[n_small-1, N-1-4, :], '\n')
```

B [ ]:
```
# We print Min and Max of all normalised columns of signs - All should be in the range [0; 1]
# Check that the matrices are normalised and there are no values> 1

mmx  = data_b[:,:,2].max()
mmn  = data_b[:,:,2].min()
print('Vol_Bid_N', '\t', mmx, '\t', mmn)

mmx  = data_b[:,:,3].max()
mmn  = data_b[:,:,3].min()
print('Vol_Ask_N', '\t', mmx, '\t', mmn)

mmx  = data_b[:,:,4].max()
mmn  = data_b[:,:,4].min()
print('DateDelta1_N', '\t', mmx, '\t', mmn)
```

creation datasets for training and testing

B [ ]:
```
# We allocate about 80% of the data (two-dimensional subarrays) for training and about 20% of
# the data (two-dimensional subarrays) for testing
# n = data_a.shape [0] # new number of lines
train_start = 0
train_end = int(np.floor(0.8*L))
print(L, train_end)
```

B [ ]:
```
test_start = train_end + 1
test_end = L
print(test_start, test_end)
```

B [ ]:
```
print("numpy size:        ", data_b.shape, '\n')
```

B [ ]:
```
print(test_end-train_end)   # size for testing
```

B [ ]:
```
# form X_train and X_test
# Divide data_b into two unequal parts: for training (X_train) and for test (X_test)

X_train = np.zeros( ( train_end, N, 5) , dtype= 'float32')
X_test  = np.zeros( ( test_end-train_end, N, 5) , dtype= 'float32')

for zz in tqdm(range(test_end)):
    if zz < train_end:
        X_train[zz,:,:] = data_b[zz,:,:]
    else:
        X_test[zz-train_end,:,:]  = data_b[zz,:,:]

# print('X_train:', '\n', X_test)
print(" Train numpy size::        ", X_train.shape, '\n')
print('\n')

# print('X_test:', '\n', X_test)
print(" Test numpy size::        ", X_test.shape, '\n')
```

B [ ]:
```
# Visual checking the correctness of filling
print(X_train[:5,N-1,:], '\n')
print(X_train[-5:,N-1,:], '\n')
print(X_test[:5,N-1,:], '\n')
print(X_test[-5:,N-1,:])
```

Creation an answer matrix for the training (y_train) and test (y_test) parts

B [ ]:
```
# form y_train and y_test
# Divide Y into two unequal parts: for training (y_train) and for test (y_test)

y_train = np.zeros( ( train_end, 1) , dtype= 'float32')
y_test  = np.zeros( ( test_end-train_end, 1) , dtype= 'float32')

for zz in tqdm(range(test_end)):
    if zz < train_end:
        y_train[zz,0] = Y[zz,0]
    else:
        y_test[zz-train_end,0]  = Y[zz,0]

# print('y_train:', '\n', y_test)
print(" Outputs train numpy size::        ", y_train.shape, '\n')
print('\n')

# print('X_test:', '\n', X_test)
print(" Outputs test numpy size::        ", y_test.shape, '\n')
```

B [ ]:
```
# Visual checking the correctness of filling

print(y_train[:5,0], '\n')
print(y_train[-5:,0], '\n')
print(y_test[:5,0], '\n')
print(y_test[-5:,0])

print(" Outputs train numpy size:         ", y_train.shape ,'\n')
print(" Outputs test numpy size:          ", y_test.shape ,'\n')
```

B [ ]:
```
data[data['Point'] == 1 ]
```

Final size control

B [ ]:
```
# print X_train, y_train, X_test u y_test

print('Data for training:')
print("X_train:")
print("Type: ", X_train.dtype,  type(X_train), "; Size: ", X_train.shape)

print("\n","y_train:")
print("Type: ", y_train.dtype,  type(y_train), "; Size: ", y_train.shape)

print('Data for testing:')
print("\n","X_test:")
print("Type: ", X_test.dtype, type(X_test), "; Size: ", X_test.shape)

print("\n","y_test:")
print("Type: ", y_test.dtype,  type(y_test), "; Size: ", y_test.shape)
```

B [ ]:
```
data_for_array_short = data_for_array[data_for_array['Point'] == 1]

data_for_array_short.info()
```

B [ ]:
```

indexes_to_keep = set(np.array(range(0,train_end+1)))

data_for_array_short_train  = data_for_array_short.take(list(indexes_to_keep))

print('array_short_train.dtypes', data_for_array_short_train.dtypes)

data_for_array_short_train.info()
```

B [ ]:
```

indexes_to_keep = set(np.array(range(test_start,L)))

data_for_array_short_test  = data_for_array_short.take(list(indexes_to_keep))

print('array_short_test.dtypes', data_for_array_short_test.dtypes)

data_for_array_short_test.info()
```

B [ ]:
```
data_for_array_short_train['Point1'].value_counts()
```

B [ ]:
```
data_for_array_short_test['Point1'].value_counts()
```

numpay for Trend / Flat classification

short Probability vector (Trend/Flat)

0, 1, 2, 3, 4, 12, 13, 14, 15, 16 = Trend = (1)

5, 6, 7, 8, 9, 10, 11 = Flat = (0)

B [ ]:
```
for i in tqdm(range(train_end)):
    if ((y_train[i,0] <= 4) | (y_train[i,0] >= 12)):
        y_train[i,0] = 1

    if ((y_train[i,0] >= 5) & (y_train[i,0] <= 11)):
        y_train[i,0] = 0
    else:
        continue

for i in tqdm(range(0, test_end-test_start+1)):
    if ((y_test[i,0] <= 4) | (y_test[i,0] >= 12)):
        y_test[i,0] = 1

    if ((y_test[i,0] >= 5) & (y_test[i,0] <= 11)):
        y_test[i,0] = 0
    else:
        continue
```

B [ ]:
```
for i in tqdm(range(train_end)):
    if (y_train[i,0] >= 2):
        print('Allarm')
        print(i)
    else:
        continue

for i in tqdm(range(0, test_end-test_start+1)):
    if (y_test[i,0] >= 2):
        print('Allarm')
        print(i)
    else:
        continue
```

B [ ]:
```
classes=['0 is Flat', '1 is Trend']
```

Gated Recurrent Unit

B [ ]:
```
# Create GRU model

model = keras.Sequential([

    keras.layers.GRU(32, dropout=0.1, recurrent_dropout=0.5, return_sequences=True, input_shape=(200, 5)),

    keras.layers.GRU(64, activation=tf.nn.relu, dropout=0.1, recurrent_dropout=0.5),

    #keras.layers.Dense(1)
    keras.layers.Dense(1, activation=tf.nn.sigmoid)

])

print(model.summary())
```

B [ ]:
```
model.compile(optimizer=keras.optimizers.RMSprop(), loss='binary_crossentropy',
              metrics=['accuracy','Precision','Recall','binary_crossentropy'])
```

B [ ]:
```
history = model.fit(X_train, y_train, batch_size=300, epochs=15, validation_split=0.3, shuffle=True, verbose=1)
```

```
B [ ]: # Evaluation
scores = model.evaluate(X_test, y_test, verbose=0)
scores
```

```
B [ ]: # Accuracy graphs during training

history_dict = history.history
plt.style.use('seaborn-darkgrid')
accuracy_values = history_dict['accuracy']

val_accuracy_values = history_dict['val_accuracy']

epochs = range(1, len(accuracy_values) + 1)
plt.figure(num=1, figsize=(15,7))
plt.plot(epochs, accuracy_values, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy_values, 'b', label='Validation accuracy')
plt.xlabel('Epochs', size=18)
plt.ylabel('Accuracy', size=18)
plt.legend()
plt.show()
```

```
B [ ]: # Precision graphs during training

history_dict = history.history
plt.style.use('seaborn-darkgrid')
accuracy_values = history_dict['precision']

val_accuracy_values = history_dict['val_precision']

epochs = range(1, len(accuracy_values) + 1)
plt.figure(num=1, figsize=(15,7))
plt.plot(epochs, accuracy_values, 'bo', label='Training precision')
plt.plot(epochs, val_accuracy_values, 'b', label='Validation precision')
plt.xlabel('Epochs', size=18)
plt.ylabel('Precision', size=18)
plt.legend()
plt.show()
```

```
B [ ]: # Recall graphs during training

history_dict = history.history
plt.style.use('seaborn-darkgrid')
accuracy_values = history_dict['recall']

val_accuracy_values = history_dict['val_recall']

epochs = range(1, len(accuracy_values) + 1)
plt.figure(num=1, figsize=(15,7))
plt.plot(epochs, accuracy_values, 'bo', label='Training recall')
plt.plot(epochs, val_accuracy_values, 'b', label='Validation recall')
plt.xlabel('Epochs', size=18)
plt.ylabel('Recall', size=18)
plt.legend()
plt.show()
```

```
B [ ]: # Loss-function graphs during training (здесь: binary_crossentropy)

history_dict = history.history
plt.style.use('seaborn-darkgrid')
accuracy_values = history_dict['loss']

val_accuracy_values = history_dict['val_loss']

epochs = range(1, len(accuracy_values) + 1)
plt.figure(num=1, figsize=(15,7))
plt.plot(epochs, accuracy_values, 'bo', label='Training Loss Function')
plt.plot(epochs, val_accuracy_values, 'b', label='Validation Loss Function')
plt.xlabel('Epochs', size=18)
plt.ylabel('Loss Function', size=18)
plt.legend()
plt.show()
```

```
B [ ]:
```

LSTM

```
# Create GRU model
model = keras.Sequential([

    keras.layers.LSTM(32, input_shape=(200, 5)),

    #keras.layers.Dense(1)
    keras.layers.Dense(1, activation=tf.nn.sigmoid)

])

print(model.summary())
```

```
model.compile(optimizer=keras.optimizers.RMSprop(), loss='binary_crossentropy',
              metrics=['accuracy','Precision','Recall','mse','mae','binary_crossentropy'])
```

```
history = model.fit(X_train, y_train, batch_size=300, epochs=15, validation_split=0.3, shuffle=True, verbose=1)
```

```
# Evaluation
scores = model.evaluate(X_test, y_test, verbose=0)
scores
```

```
# Accuracy graphs during training

history_dict = history.history
plt.style.use('seaborn-darkgrid')
accuracy_values = history_dict['accuracy']

val_accuracy_values = history_dict['val_accuracy']

epochs = range(1, len(accuracy_values) + 1)
plt.figure(num=1, figsize=(15,7))
plt.plot(epochs, accuracy_values, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy_values, 'b', label='Validation accuracy')
plt.xlabel('Epochs', size=18)
plt.ylabel('Accuracy', size=18)
plt.legend()
plt.show()
```

```
# Precision graphs during training

history_dict = history.history
plt.style.use('seaborn-darkgrid')
accuracy_values = history_dict['precision']

val_accuracy_values = history_dict['val_precision']

epochs = range(1, len(accuracy_values) + 1)
plt.figure(num=1, figsize=(15,7))
plt.plot(epochs, accuracy_values, 'bo', label='Training precision')
plt.plot(epochs, val_accuracy_values, 'b', label='Validation precision')
plt.xlabel('Epochs', size=18)
plt.ylabel('Precision', size=18)
plt.legend()
plt.show()
```

```
# Recall graphs during training

history_dict = history.history
plt.style.use('seaborn-darkgrid')
accuracy_values = history_dict['recall']

val_accuracy_values = history_dict['val_recall']

epochs = range(1, len(accuracy_values) + 1)
plt.figure(num=1, figsize=(15,7))
plt.plot(epochs, accuracy_values, 'bo', label='Training recall')
plt.plot(epochs, val_accuracy_values, 'b', label='Validation recall')
plt.xlabel('Epochs', size=18)
plt.ylabel('Recall', size=18)
plt.legend()
plt.show()
```

B [ ]:
```
# Loss-function graphs during training (здесь: binary_crossentropy)

history_dict = history.history
plt.style.use('seaborn-darkgrid')
accuracy_values = history_dict['loss']

val_accuracy_values = history_dict['val_loss']

epochs = range(1, len(accuracy_values) + 1)
plt.figure(num=1, figsize=(15,7))
plt.plot(epochs, accuracy_values, 'bo', label='Training Loss Function')
plt.plot(epochs, val_accuracy_values, 'b', label='Validation Loss Function')
plt.xlabel('Epochs', size=18)
plt.ylabel('Loss Function', size=18)
plt.legend()
plt.show()
```