UNIVERSITÀ
DI PAVIA

DEPARTMENT OF ECONOMICS AND MANAGEMENT

MASTER PROGRAMME IN

ECONOMICS, FINANCE AND INTERNATIONAL INTEGRATION

APPLICATION OF MACHINE LEARNING TO QUANTITATIVE TRADING

Supervisor:

Prof. GIUDICI PAOLO STEFANO

Student:
PHAN TIEN DUNG

Matr. n. 472423

Anno Accademico 2023-2024

Dichiarazione di originalità: Con la presente dichiaro di aver redatto autonomamente la presente tesi di laurea magistrale utilizzando solo gli strumenti indicati e che tutte le parti che, nel testo o nel contenuto, sono tratte da altre opere sono state chiaramente identificate come citazioni, con l'indicazione delle relative fonti. Questa tesi di laurea magistrale non è stata presentata in alcun altro corso di studio come prova d'esame, né in forma identica né simile.

Declaration of Originality: I hereby declare that I have independently written this master's thesis using only the tools indicated, and that all parts that, in text or content, are derived from other works have been clearly identified as quotations, with the relevant sources cited. This master's thesis has not been submitted in any other course of study as an examination work, neither in identical nor similar form.

Phan Tien Dung (Matriculation Number 472423), September 7, 2024

Abstract

This thesis explores the application of Machine Learning in Quantitative Trading. It primarily focuses on how Machine Learning techniques can be used to develop algorithmic trading strategies within financial markets. The Machine Learning models employed range from classical approaches like Logistic Regression and Support Vector Machines to more advanced methods such as Recurrent Neural Networks, Long Short-Term Memory networks, and Gated Recurrent Units.

The models were tested using approximately 20 years of stock price data from International Business Machines Corporation (IBM), spanning from January 2000 to March 2021. This period was divided separately into two segments: a training phase from January 2000 to February2019, used to build the Machine Learning models, and a testing phase from February 2019 to March 2021, used to validate them.

The thesis highlights two popular Machine Learning tasks: regression and classification. Logistic Regression and Support Vector Machines are presented as classification models, where the output serves as a trading signal for the stock. In contrast, Recurrent Neural Networks, Long Short-Term Memory networks, and Gated Recurrent Units are used as regression models, where the output predicts the next price of the stock. An additional step is required to interpret this predicted price as a trading signal.

Furthermore, all model-based strategies are evaluated for effectiveness using an intraday vectorized backtest, which provides visual evidence of each strategy's profitability and risk management through equity curves and maximum drawdown metrics. The thesis also considers metrics such as hit rate and the rate of false positions, combining them with the equity curves to offer a comprehensive assessment of each strategy's overall efficacy. Particularly, the research paid attention to the safety AI solutions in different aspect like: Accuracy, Robustness, Fairness and Explainability.

Keywords: Machine Learning, Deep Learning, Quantitative Trading, Algorithmic Strategies, Backtesting, Maximum Drawdown, Risk Management, Trustworthiness AI

Astratto

Questa tesi esplora l'applicazione del Machine Learning nel Trading Quantitativo. Si concentra principalmente su come le tecniche di Machine Learning possano essere utilizzate per sviluppare strategie di trading algoritmico nei mercati finanziari. I modelli di Machine Learning impiegati spaziano da approcci classici come la Regressione Logistica e le Macchine a Vettori di Supporto a metodi più avanzati come le Reti Neurali Ricorrenti, le Reti Neurali Long Short-Term Memory e le Gated Recurrent Units.

I modelli sono stati testati utilizzando circa 20 anni di dati sui prezzi delle azioni della International Business Machines Corporation (IBM), che vanno da gennaio 2000 a marzo 2021. Questo periodo è stato suddiviso in due segmenti: una fase di addestramento, da gennaio 2000 a febbraio 2019, utilizzata per costruire i modelli di Machine Learning, e una fase di test, da febbraio 2019 a marzo 2021, utilizzata per convalidarli.

La tesi evidenzia due compiti popolari del Machine Learning: la regressione e la classificazione. La Regressione Logistica e le Macchine a Vettori di Supporto sono presentate come modelli di classificazione, in cui l'output serve come segnale di trading per il titolo azionario. Al contrario, le Reti Neurali Ricorrenti, le Reti Neurali Long Short-Term Memory e le Gated Recurrent Units sono utilizzate come modelli di regressione, in cui l'output prevede il prezzo futuro del titolo. È necessario un passaggio aggiuntivo per interpretare questo prezzo previsto come segnale di trading.

Inoltre, tutte le strategie basate sui modelli sono valutate per efficacia tramite un backtest vettorizzato intraday, che fornisce prove visive della redditività e della gestione del rischio di ciascuna strategia attraverso curve di equità e metriche di drawdown massimo. La tesi considera anche metriche come il tasso di successo e il tasso di posizioni errate, combinandole con le curve di equità per offrire una valutazione complessiva dell'efficacia di ciascuna strategia. Particolarmente, la ricerca ha posto attenzione alle soluzioni di sicurezza nell'IA sotto diversi aspetti, come: Accuratezza, Robustezza, Equità e Spiegabilità.

Parole chiave: Machine Learning, Deep Learning, Trading Quantitativo, Strategie Algoritmiche, Backtesting, Maximum Drawdown, Risk Management, Trustworthiness AI

Acknowledgments

This thesis represents the culmination of years of study, research, and personal growth, all of which would not have been possible without the guidance, instruction, support, and opportunities afforded to me by several individuals and institutions.

First and foremost, I would like to extend my deepest gratitude to Prof.Dr. Paolo Giudici, whom I first encountered during my time in the Data Science course in 2018 in my Master's Degree at the University of Pavia. That was the very first time in my life I had known that, students in Economics or Finance can themselves code programms. From the very beginning, his passion for quantitative analysis and its practical applications left a profound impression on me. Through his teachings, I gained a solid understanding of mathematics, statistics, and their critical roles in the fields of economics and finance. Prof. Giudici 's ability to bridge complex theoretical concepts with real-world scenarios was nothing short of inspiring, and it is no exaggeration to say that he was instrumental in laying the foundation for my career in Quantitative Finance. His mentorship not only deepened my knowledge but also sparked my interest in pursuing a career in this field. The insights he shared with me about the world of Quantitative Finance have guided me to where I stand today in my professional journey als a Quant Analyst. For this, I am forever grateful.

Additionally, I would also like to express my sincere appreciation to the Double Degree program, which provided me with an exceptional opportunity to broaden my horizons by living and studying in two different countries: Italy and Germany. This unique experience enriched my academic and personal life in ways that I could never have imagined. The chance to immerse myself in the diverse cultures, educational systems, and perspectives of these two countries has been truly transformative. I owe a great deal of thanks to the supporters, administrators and particularly, Mr. Federico Franceschini, Double Degree coordinator, who facilitated this program, making it possible for students like me to benefit from such an invaluable international experience.

During my time at the University of Tübingen, I was fortunate to have been under the guidance of Prof.Dr. Martin Biewen, whose supervision was crucial to the successful completion of this thesis. Prof Biewen's expertise, patience, and thoughtful feedback were instrumental in shaping the direction and quality of my research. His encouragement and advice helped me navigate through the various challenges of my thesis, and for this, I am deeply thankful.

While my original plan was to graduate earlier, unforeseen personal circumstances required me to extend my studies. Although this delay was not part of my initial plan, it provided me with the opportunity to further refine my research and gain additional insights that I might not have otherwise encountered. In this regard, I would like to express my heartfelt thanks to the University of Pavia for their understanding and support during this time. The city of Pavia, with its rich history and welcoming community, became a second home to me, offering me not just an academic environment but also a nurturing space for personal growth. My time at Pavia marks a significant

chapter in my life, one filled with learning, self-discovery, and cherished memories. I am truly grateful to both the university and the city for everything they have given me.

In closing, this thesis is a testament not only to my hard work but also to the support, guidance, and opportunities provided by many individuals and institutions along the way. To everyone who has been a part of this journey, I extend my deepest gratitude. Thank you for helping me reach this important milestone in my life.

Contents

List of Figures

List of Tables

Part 1: Introduction and Trading Motivations

Chapter 1: Introduction

1.1 Why do people love trading?

Why do people love trading? The main reason was absolutely the profit. Sebastien and Sourva (2019) have agreed that once upon a time, trading is an indispensable part of humankind, people often bought raw materials at a low price and attempted to sell at a higher price to earn a profit. Later, the rich Romans used the Roman forum to exchange currencies, bonds, and their investment. The earliest form of the stock exchange was created in Antwerp, Belgium in 1531. In there, traders frequently met and exchanged promissory notes or bonds. However, the tale of the stock exchange only began when a group of 24 brokers signed the Buttonwood Agreement in 1790, bounding the group to trade with each other under specific rules. After that, the stock exchange kept evolving significantly and continuously in complexity and scope. Nonetheless, it still stayed unchanged conceptually. At that time, the stock exchange was still a trading venue where buyers or sellers screamed, yelled, and used hand signals to place positions of traded products. In detail, the hand signal as palm facing out and hands away or palm facing in and hands holding up used as an indicator of a long or short position respectively. Later, thanks to telegraph's technology, the financial market had begun a new chapter, putting an end to the physical venue. Changes only came in 1971 when the NASDAQ stock exchange launched a completely electronic system. It means the whole trading process can be handled systematically in the blink of an eye. And this, undoubtedly, is the main driver leading to the regime of algorithmic trading, which has been widely implemented nowadays.

Robert (2021) and Raja, Maxence, and Daniel (2020) have defined that algorithmic or quantitative trading as the computerized execution of traded financial products as stocks, bonds, funds, or a plethora of derivatives, following predefined instructions. Compared to human-driven trading, this style is more consistent and efficient in execution while enjoying lower transaction costs. Commonly, quantitative trading falls into two main categories, say rule-based and predictive modeling approach. While the investor needs to determine exactly the rule of trade in the rule-based approach, for instance, if the short-term moving average of a given traded product goes over the long-term moving average then place a long position, the latter employs mathematically sophisticated models to figure out the hidden patterns from historical data. A predictive model emphasizes historical data and attempted to discover the hidden patterns that tend to repeat frequently. Consequently, the model can exploit the valuable signals, which will be converted into long or short positions scientifically. That is the reason why Machine Learning jumps into, and in this framework, only the predictive modeling approach will be discussed.

According to Aurélien (2019), Machine Learning is the field that utilizes statistical analysis, probabilistic programming, mathematics, and computer science to exploit

hidden patterns in the huge amount of data. This is one of the fastest-growing areas recently and has a profound application in practices, ranging across fields in business and technology as image detection or autonomous car. Quantitative Finance or Trading is not an exception. In quantitative finance, Matthew, Igor, and Paul (2020) have pointed out that machine learning offers various applications, such as price prediction by deep learning, optimal execution by reinforcement learning, optimizing trading strategy, risk management, or signal detection amongst noisy datasets. Machine learning employs algorithms that can learn how to perform tasks like classification or regression without explicitly too much being programmed. These can be various, including from classical statistical models such as linear regression to more complex as a fully–connected neural network. Machine learning, nowadays, is ever-improving, and gain popularity in quantitative finance. It attracted the huge attention of large quantitative funds or other financial institutions like World Quant, Renaissance Capital Management, Citadel, or Two Sigma.

Aurélien (2019) has proposed a couple of reasons why machine learning is gaining popularity in the recent regime. Firstly, thanks to its simplicity, programming is increasingly easily accessible to a wider public compared with the past. For instance, in the spam email detection problem, machine learning automatically can learn which words will be contained in the spam email given a labeled data, therefore, this program is much shorter and less complicated than traditional methods. Also, with the trend towards increasingly computational resources and free large datasets, accessing, wrangling, and manipulating the data is easier than ever, therefore, it has been a crucial force behind the dramatic performance improvement of machine learning that is driving innovation across many industries, particularly in finance which already has a long history of using by-far sophisticated models in analyzing and making decisions. Nonetheless, while enjoying many advantages, machine learning has faced a couple of disadvantages. Matthew et al (2020) have advocated that one of the most noticeable things is that machine learning is always considered as a black box, which only ingests input data and produces output without explicit explanation. Matthew et al (2020) have also argued that the absence of well-established theories and concepts in more foundational scientific fields as financial econometrics or financial time series sometimes makes practitioners difficult to understand or seek the optimal answers.

To this end, the thesis will present machine learning as a non-linear extension of various topics in quantitative trading with an emphasis on how to design an automated algorithm for financial traded products. The workflow demystifies computation, explains its intellectual underpinnings, and covers the essential elements of machine learning in quantitative trading. The thesis constitutes five chapters. The first chapter includes an introduction and explanation of why machine learning gained popularity in the finance and business industry. The second part mainly emphasized machine learning theory and its application in designing models in quantitative trading as cross-validation, regularization, stability, hypothesis class. In this section, the thesis will point out the complex mathematics underpinning machine learning, and its contribution to the success of machine learning algorithms. This could help the practitioners get rid of the "black box" problem in machine learning. The next one will elaborate on the practical model in

machine learning and its application in quantitative trading as logistic, support vector machine, recurrent neural network, the long-short term, and gated unit model. Here, each model will be crafted from scratch, then put into an algorithmic trading context. Also, the backtesting mechanism is discussed in this section. It will compare the effectiveness of machine learning-based strategies and the "buy and hold" strategy.

1.2 Research aim and objective

The primary aim of this research is to investigate the effectiveness of integrating advanced machine learning techniques, particularly neural networks, with foundational mathematical and statistical skills to develop sophisticated trading strategies in financial markets. With the rapid growth of computational power and the availability of large datasets, neural networks have emerged as powerful tools capable of capturing intricate patterns within market data that traditional statistical methods might overlook. The research aims to harness these capabilities to enhance the accuracy of market predictions and trading decisions. Specifically, the objectives include developing a framework that combines neural networks with statistical methods like time series analysis and regression models to forecast asset prices and identify profitable trading opportunities.

Moreover, this research will explore the adaptability of these machine learning models to different market conditions, such as varying levels of volatility, and their ability to generalize across different asset classes, including stocks, commodities, and cryptocurrencies. Another key objective is to assess the model's performance in both short-term trading (high-frequency trading) and long-term investment scenarios, providing a comprehensive evaluation of its practical applications. The study will also focus on the interpretability of these models, ensuring that the predictions made by the neural networks can be understood and trusted by human traders and analysts.

The integration of neural networks with traditional mathematical and statistical approaches is expected to provide a dual benefit: the ability of neural networks to process and learn from large, unstructured datasets, and the mathematical models' grounding in established financial theory. Previous research has demonstrated that such hybrid approaches can lead to improved predictive performance and more robust trading strategies accrording to Hu et al (2020) and Hiransha et al (2018).

Additionally, the study aims to contribute to the existing body of knowledge by addressing the challenges of overfitting, model interpretability, and computational efficiency, which are critical for the successful deployment of machine learning models in real-world trading environments mentioned in Heaton et al (2017) and Bao et al (2017)

In summary, the research seeks to advance the field of algorithmic trading by developing and testing a novel approach that combines neural networks with traditional statistical methods. This approach aims to improve the precision and reliability of trading models, ultimately contributing to more effective and profitable trading strategies in the financial

markets.

1.3 Thesis structure

This thesis is organized clearly into three comprehensive sections, each contributing to a thorough journey of the research topic. Part 1: Introduction and Trading Motivations opens with a detailed introduction that delves into the reasons why trading is compelling to individuals, providing a context for the study. This section not only outlines the research aim but also specifies the objectives that guide the entire thesis. It explains the history of trading, earning money and the transformation of the trade work. The introduction sets the stage for the rest of the work by offering a clear roadmap of the thesis structure, helping readers understand the flow of the research from the outset.

Part 2: Literature Review and Methodologies Development is an in-depth examination of the theoretical and methodological underpinnings of the research. It begins with a discussion on the theoretical foundations of machine learning methodologies, emphasizing the relevance of time series data and supervised learning frameworks in the context of trading. This section thoroughly reviews various machine learning algorithms, including logistic regression, support vector machines, and advanced deep learning techniques such as recurrent neural networks (RNNs), long short-term memory networks (LSTMs), and gated recurrent units (GRUs). In addition to algorithmic exploration, this part also focuses on the development of methodologies, detailing the processes involved in model estimations and offering an overview of the data sources utilized in the study. Statistical descriptions of the data are provided to give a clear understanding of the dataset's characteristics, ensuring a solid foundation for the empirical work that follows.

Part 3: Empirical Results, Trustworthiness of AI solutions and Conclusions presents the core findings of the research, focusing on the empirical analysis of both deep learning-based and machine learning-based algorithms. This section provides a detailed account of the results, offering insights into the performance and applicability of the various models discussed in Part 2. The empirical findings are critically analyzed, with a particular focus on their implications for the field of trading and machine learning. The final chapter in this section synthesizes the research findings, drawing conclusions that address the initial research aims and objectives. In this part, various aspects of the safe AI solutions have been proposed and statistically tested. This part will give a comprehensive overview of how reliable the complex black box models are. This will lay a concrete foundation to develop AI solution in the future.

The thesis is further enhanced by an appendix that includes mathematical proofs and code examples, providing additional technical details that support the main text. This supplementary material is designed to give readers deeper insight into the methodologies used and to facilitate replication or further exploration of the study's findings. The thesis concludes with a comprehensive list of references, ensuring that all sources of information are properly credited and that readers have access to the full range of materials that informed the research.

Part 2: Literature Review and Methodologies Development

Chapter 2: Theory and Literature Review

In the following literature review, I focus on the most fundamental underpinnings in Machine Learning and its rigorous mathematical treatment in Finance's application. Firstly, I will present studies that convert a given time series data into a supervised learning form. It is of paramount importance as the time-series data, per se, is not in supervised learning form, while most of the data in finance are time series. Hence, it will run into trouble in implementing the supervised learning algorithms in financial data. Secondly, I will connect the supervised learning converted previously and the cross-validation which is the most convenient approach to model selection in Machine Learning. Also, I notice the main difference in implementing cross-validation techniques in chronological and non-chronological data. In the next part, I will present the rigorous treatment of a stable Machine Learning model with some special conditions of the loss function. The literature review additionally pointed out that if the loss function is equipped with some particular properties, it will lead to a more stable model in practice. The last part of the literature review mostly puts weight on the theory of the machine learning models, ranging from classical models like Logistic Regression to Deep Learning models as Recurrent Neural Network. This part will go through the pure theory and explain how it works in practice.

In this section, I will attempt to use "mathematical language" to describe and explain the algorithm. According to Shalev and Ben (2014), the best way to describe and understand a machine learning algorithm is to characterize a hypothesis class. In other words, a hypothesis class is a scientific extrapolation of an algorithm in which the algorithm is determined mathematically. Also, the hypothesis is integrated seamlessly with several relevant information as the loss function. For example, the hypothesis class of Logistic Regression can be written as:

$$\mathcal{H}_{lr} = \sigma \bullet \mathcal{Q}_d(\theta, x, b) \tag{1}$$

$$\mathcal{Q}_d(\theta, x, b) = \{h_\theta(x) = \sum_{i=1}^{d} x_i\theta_i + b, \forall x, \theta \in \mathbb{R}^d, b \in \mathbb{R}\} \tag{2}$$

$$\sigma(u) = \frac{1}{1+e^{-u}}, \forall u \in \mathbb{R} \tag{3}$$

$$\ell(\theta, x, y) = e^{y(x'\theta+b)}) \tag{4}$$

Where the $\mathcal{H}_{lr}$ is the hypothesis class of Logistic Regression, the mathematical notation $\bullet$ could be considered as a composite operand between two or more functions. $\mathcal{Q}_d(\theta, x, b)$ is indeed a linear combination created from two d-dimension vector x, θ and the real intercept b. $\sigma(u)$ is an activation function, and $\ell(\theta, x, y)$ is a loss function.

The hypothesis class of a Logistic Regression algorithm could be interpreted as, firstly, a linear combination of the instance x and a random parameter θ is created, then that linear combination is fed to the activation function $\sigma(u)$. Ultimately, the trainable parameter θ

is trained during the optimization process of the loss function $\ell(\theta, x, y)$. It is worth noticing that every algorithm has its hypothesis class and the loss function. For instance, in the Support Vector Machine algorithm, the loss function is hinge loss, which is completely different from the 01-loss, commonly used in the classification algorithm. In this sense, the hypothesis class crafts the whole story of an algorithm ranging from algorithm creation to the optimization process of the loss function. The target is to optimize the generalized loss function (objective function) by a couple of conditions, namely empirical risk minimization, structural risk minimization, or regularized risk minimization. The main difference between the three conditions is the objective function and its additional components. In this section, I will demonstrate some fundamental issues and theories in designing machine learning algorithms.

2.1 The theory of machine learning methodologies

2.1.1 From time series to supervised learning

The univariate time-series data, per se, does not have a supervised learning form, therefore, it is impracticable to directly deploy supervised machine learning algorithms. Nevertheless, the time series can be converted from a univariate or multivariate time series into a supervised learning form. For the sake of concreteness, assume that a univariate time-series data is indexed from 1 to 10, with a lag equal to 3, it can be framed in a supervised learning form by the logic: the observation indexed by 1, 2, and 3 will be used to predict the one indexed by 4, similarly, the one indexed by $2, 3$, and 4 will be used to predict the one indexed by 5, and so on. Mathematically speaking, univariate time-series data could be finally re-framed in a matrix $\mathbb{X}$ and a vector y, corresponding to the input and output variables in a supervised learning form. The summary of the pseudo input $\mathbb{X}$ and output y can be described as below:

	Input		Output
1	2	3	4
2	3	4	5
...	...	...	...
7	8	9	10

Source: Author

Table 1: The pseudo supervised learing from converted from a time series data

Ultimately, the supervised learning form of a time series is available, and from now, any supervised machine learning algorithm could be applied for the time-series data. In the quantitative trading context, pseudo data in table 1 could be thought as, for the observations indexed by $1, 2, 3$, the situation of the stock price is indexed by 4, and so on. Thus, if I have all information up to now, says the observations indexed by $8, 9, 10$, what is the next situation or direction of the stock price?. There is a distinct point in the re-framed input $\mathbb{X}$ and the output y. Matthew et al (2020) have warned that fact that $\mathbb{X}$ and y are ordered sequentially, hence, it is impossible to change shuffle the data or use a technique like bootstrap to increase the diversity of the data. It is one of the most

extraordinary of machine learning in financial and non-financial fields.

Additionally, in quantitative trading, the output y could be more flexible. It could be the numerical number or categorical form, corresponding to price or the increase or decrease of the price. Consequently, the classification or regression algorithms could be applied accordingly. In the frame of the thesis, the deep learning model is regression, meaning that I train the model from numerical variable $\mathbb{X}$ to a numerical variable y. The numerical output y is a forecasted price of the given data. Consequently, the trading strategy is established by comparing the predicted and the previous price of the given data. However, in classical machine learning, the model is indeed a classification. The output y of these models is nothing but a pair of $(1, -1)$ or (up, down). It corresponds to the directional movement of the given time-series data. In these models, it is unnecessary to interpret in "trading language" as the output, per se, is indeed trading signal.

2.1.2 Model selection by cross-validation

Getting back to the hypothesis class mentioned previously, each combination of trainable parameter θ and a hyperparameter represents a supervised model. Therefore, if those parameters are not constrained, there will be indefinitely many models. Consequently, the problem boils down to tweaking trainable parameters and hyperparameters. Shalev and Ben (2014) also pointed out that there are two commonly used approaches, namely Structural Risk Minimization (SRM), and cross-validation. Shalev and Ben (2014) have already suggested the SRM approach is particularly useful when the hypothesis class can be partitioned by countable sub-hypothesis classes $\mathcal{H}_1, \mathcal{H}_2 \ldots, \mathcal{H}_n$, then the optimal model, h^*, will be described as follow:

$$h^* \quad \in \quad argmin_{\forall h \in \mathcal{H}}[\mathcal{L}_S(\theta) + \epsilon_{n(h)}(m, w_{n(h)}\zeta)] \tag{5}$$

Where ζ is a confidence parameter. The weights w_i, $\forall i = 1, \ldots, m$, corresponds to the sub-hypothesis $\mathcal{H}_i$. The weights need to satisfy $\sum_{i=1}^{m} w_i \leq 1$, and $\epsilon_n(m, w_n\zeta) = sup_{h,\theta \in \mathcal{H}_n}|\mathcal{L}_D(\theta) - \mathcal{L}_S(\theta)|$. It is well-noted that the value of parameter $\epsilon_n(m, w_n\zeta)$ is the supreme value of the sequence of the difference between the true loss and the training loss for every function h, or more precisely θ, resides in sub-hypothesis $\mathcal{H}_n$.

Shalev and Ben (2014) have given a prominent application of SRM, says the Minumum Description Length (MDL). MDL has many applications, particularly in the Decision and Tree algorithm. By a wise selection of the full set of weights w_i, the right-hand side of the equation (5) could be simplified. Jonh, Peter, Robert, and Martin (1996) have given some theories and examples of SRM, however, they are quite difficult to deploy due to the mathematical complexity. While the SRM approach is quite useful in some cases, the right-hand side of equation (5) always finds too much pessimism in practice. Thus, the optimal models should be sought in a more practical method, the cross-validation.

Matthew et al (2020) have also defined that cross-validation is a method of hyperparameters fine-tuning by rotating k-folds of the training datasets. The models will be trained in the $k-1$ folds then validated in the last fold accordingly. The advantage of

this method is pretty straightforward. Nevertheless, it should be cautious because the converted data, mentioned previously, contained chronological order, consequently, some widely used techniques in cross-validation like shuffling can not be applicable. Hence, the k-folds principle also needs to follow the sequential order of the data. Figure 1 has presented a pseudo approach of cross-validation in sequential data. It is completely different in the cross-validation in other non-sequential data. Firstly, the k^{th} fold validation always follows the $k-1$ training part as the data is in sequential order. Secondly, it is impossible to shuffle or bootstrap the original data. Besides, it is worth noticing that there are two kinds of cross-validation in the time-series supervised learning form. On the one hand, I can fix the beginning point of $k-1$ training part then move forwards only the k^{th} fold validation. On the other hand, I can move forwards with both the training and validation. Both approaches guarantee that the validation part is always independent of the training part and follows the sequential order of the original data.

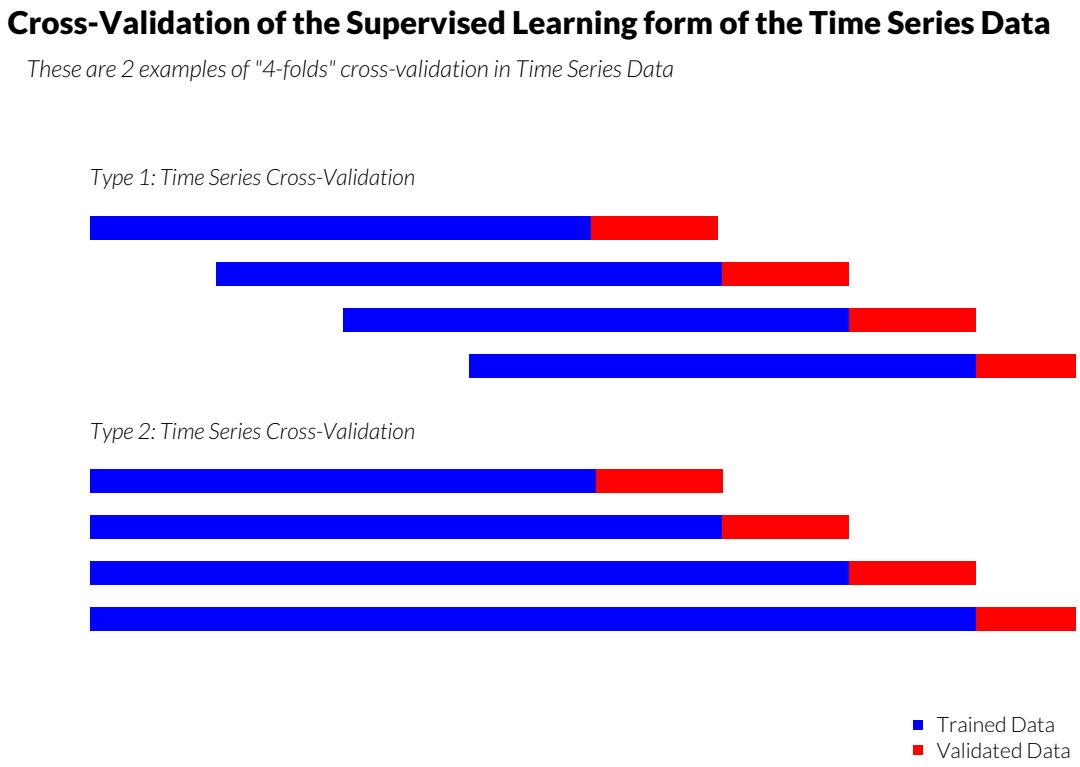

Figure 1: The pseudo cross-validation of a chronological supervised learning form

Figure 1 has explicitly demonstrated two ways of cross-validation of the chronological data. The choice of cross-validation is flexible. However, each way, to some extent, has an impact on the run time of the machine learning algorithm.

2.1.3 Stable models with ℓ_2 regularization, Lipschitz or Smoothed loss function

In machine learning, the centerpiece is the loss function $\ell(\theta, x, y), \forall x \in \mathbb{R}^d, y \in \mathbb{R}$ and the ability to generalize in the testing environment. The loss function $\ell(\theta, x, y)$ could be explained as a penalty function in which its value receive a higher value if the algorithm classifies inaccurately in the classification model or its breadth between in output and y is wide in the regression model, otherwise, its value is small. Formally, the ability to generalize a model in a new environment, which has not been exposed to before, is the true loss. Mathematically speaking, the ability of the generalization is the magnitude of $\mathbb{E}(\mathcal{L}_D(\theta, x, y))$.

The ability to generalize in an unseen environment needs to be bounded tightly. It will help the model prevent the overfitting phenomenon as much as possible, leading to a better and stable model in practice. Intuitively, there are a couple of definitions of the stability of a machine learning model. Shalev and Ben (2014) have illustrated that a model is stable if a small change in the input to the algorithm does not lead to a big change in the output. Formally, let $\mathcal{A}$ is an algorithm and an m-tuple training set $\mathcal{S} = \{z_i = (x_i, y_i)\}\ \forall i = 1, \ldots, m$, and $\mathcal{A}(\mathcal{S})$ denotes the outcome of an algorithm $\mathcal{A}$ with the instances in the training set $\mathcal{S}$ as input. Given a training set $\mathcal{S}$ and an additional observation z^*, $\mathcal{S}_i$ denotes the training set with a replacement z^* for z_i. Therefore, the stability of a model will be measured by comparing the loss value of the outcome $\mathcal{A}(\mathcal{S})$ on z_i to the loss value of $\mathcal{A}(\mathcal{S}_i)$ on z_i. For sake of clarity, the stability is measured by: $\ell(\mathcal{A}(\mathcal{S}_i, z_i)) - \ell(\mathcal{A}(\mathcal{S}), z_i)$.

Shalev and Ben (2014) has proposed that the stability of the model has an intricate relationship with the overfitting phenomenon, which is described as below:

$$\mathbb{E}(\mathcal{L}_{\mathcal{D}}(\mathcal{A}(\mathcal{S})) - \mathcal{L}_{\mathcal{S}}(\mathcal{A}(\mathcal{S}))) \quad = \quad \mathbb{E}(\ell(\mathcal{A}(\mathcal{S}_i, z_i) - \ell(\mathcal{A}(\mathcal{S})), z_i)) \tag{6}$$

Where $\mathcal{L}_{\mathcal{D}}(\mathcal{A}(\mathcal{S}))$ and $\mathcal{L}_{\mathcal{S}}(\mathcal{A}(\mathcal{S}))$ are the true and training loss of algorithm $\mathcal{A}$. The proof of the above equation is straightforward and is given in the appendix.

Hence, the more stable the model is, the better in preventing overfitting the model is. Consequently, the problem of stability boils down to bounding the $\mathbb{E}(\ell(\mathcal{A}(\mathcal{S}_i, z_i) - \ell(\mathcal{A}(\mathcal{S})), z_i))$. Additionally, if the model is stable, it can generalize better in an unseen environment. The stability is inherited from the properties of the loss function, that is why in this section, the loss function will be equipped with a strong property of Lipschitz or Smoothed function and will be trained with a ℓ_2 norm constraint. Thus, the objective function, now, is the train loss function combined with a part of the ℓ_2 norm of the variable θ. Formally, ones will optimize the below function:

$$\mathcal{L}_{\mathcal{S}}(\theta) + \frac{\gamma}{2}\|\theta\|^2 = \frac{1}{m}\sum_{i=1}^{m} \ell(\theta, x_i, y_i) + \frac{\gamma}{2}\|\theta\|^2 \tag{7}$$

Where γ is a regularization hyperparameter, controlling the level of regularization in the training loss.

Shalev and Ben (2014) have given two very important lemmas directly linked between the model and the loss function.

Lemma 1:
Given any algorithm $\mathcal{A}$, an m-tuple (x_i, y_i) training data set $\mathcal{S}$, the loss function $\boldsymbol{\ell}(\theta, x, y)$ has $\rho-$Lipschitz properties, in other words, $\boldsymbol{\ell}(\theta, x, y)$ satisfies: , then ultimately, we have:

$$\mathbb{E}(\mathcal{L}_{\mathcal{D}}(\mathcal{A}(\mathcal{S}))-\mathcal{L}_{\mathcal{S}}(\mathcal{A}(\mathcal{S}))) \leq \frac{2\rho^2}{\gamma m} \tag{8}$$

Where γ is a regularization parameter.

Lemma 2:
Similar notation with **Lemma 1**, and $f(\theta)$ is a differentiable function, the loss function is non-negative, then we ultimately have:

$$\mathbb{E}(\mathcal{L}_{\mathcal{D}}(\mathcal{A}(\mathcal{S}))-\mathcal{L}_{\mathcal{S}}(\mathcal{A}(\mathcal{S}))) \leq \frac{48\beta}{\gamma m}\mathbb{E}(\mathcal{L}_{\mathcal{S}}(\mathcal{A}(\mathcal{S}))) \tag{9}$$

Where γ is a regularization parameter.

Two prominent points coming out from the above lemmas. From one side, if the value of variable γ is increased, the bound of difference in the true and train loss function $\mathcal{L}_{\mathcal{D}}(\mathcal{A}(\mathcal{S}))-\mathcal{L}_{\mathcal{S}}(\mathcal{A}(\mathcal{S})))$ will be more tightened. Undoubtedly, the model will be more stable and better in preventing the overfitting phenomenon. This is a very useful property in reality when practitioners can fine-tune the value of γ to find the optimal value. From the other side, a higher value of γ will lead to the higher value of the training loss. Ultimately, a proper loss function combined with an optimal regularization could lead to an optimal model. This technique is quite useful in the cross-validation mentioned in the last section.

2.2 The theory of machine learning algorithms

2.2.1 Logistic Regression

Logistic Regression is a machine learning algorithm learning from the business domain $\mathbb{X}$ to the interval $[0, 1]$. The outcome of the model could be considered as the probability of the events as the probability of the event $\{1, -1\}$ or $\{$ up, down$\}$. For instance, it could be convenient in predicting the trend of the stock price in the future: increase or decrease. Mathematically speaking, the hypothesis class, including mathematical expression of the algorithm and the loss function, of the logistic regression can be defined as below:

$$\mathcal{H}_{lr} = \sigma \bullet \mathcal{Q}_d(\theta, x, b) \tag{10}$$

$$\mathcal{Q}_d(\theta, x, b) = \{h_\theta(x) = \sum_{i=1}^{d} x_i\theta_i + b, \forall x, \theta \in \mathbb{R}^d, b \in \mathbb{R}\} \tag{11}$$

$$\sigma(u) = \frac{1}{1+e^{-u}}, \forall u \in \mathbb{R} \tag{12}$$

$$\ell_{lr}(\theta, x, y) = e^{y(x'\theta+b)}) \tag{13}$$

Where $\ell_{lr}(\theta, x, y)$ is the loss function, $\sigma(u)$ is a sigma activation function and the mathematical notation $\bullet$ is a composite operand between two or more functions.

It is straightforward to verify that the loss function, $\ell_{lr}(\theta, x, y)$ in equation (13), is a convex function with respect to θ, hence the optimization of the objective function is quite effective. From the previous section, to be more stable and avoid overfitting at all costs, I will introduce the regularization part in the traditional objective function:

$$\mathcal{L}_{\mathcal{S}}(\theta) + \frac{\gamma}{2}\|\theta\|^2 = \frac{1}{m}\sum_{i=1}^{m} e^{y_i(x_i'\theta+b)} + \frac{\gamma}{2}\|\theta\|^2 \tag{14}$$

Where γ is a regularization hyperparameter.

The trainable parameter θ is sought via the optimization process of the equation (14). Thanks to the convexity of the loss function, $\mathcal{L}_{\mathcal{S}}(\theta) + \frac{\gamma}{2}\|\theta\|^2$ is a convex function as well. Hence, the optimization process of the equation can be done easily via the numerical or traditional method. The objective function, $\mathcal{L}_{\mathcal{S}}(\theta) + \frac{\gamma}{2}\|\theta\|^2$, could be optimized numerically via several optimization algorithms like stochastic gradient descent or gradient descent.

2.2.2 Support Vector Machine

Support vector machine (SVM) is a machine learning algorithm that utilizing a linear or non-linear hyperplane to classify observations. The goal of SVM is to train a model that assigns unseen objects into a particular category. It is achieved by creating a hyperplane separating the business domain space, $\mathbb{X}$, into two subspaces. Based on the features in a new sample, the algorithm will place the sample into a predefined side of the hyperplane. Roughly speaking, a half-space or hyperplane will separate a training set with a large margin, meaning that all the observations are not only on the correct side but also far away from the perceptron. Like a logistic regression algorithm, a support vector machine could be used in predicting the trend of the stock price.

There are some misunderstandings in SVM and other linear classified algorithms. At a glance, one always conceived SVM like other linear family algorithms as an individual perceptron classification, however, there are a couple of different factors in those

algorithms. Firstly, SVM no longer uses the "01" loss function in constructing the hypothesis class, instead, SVM will utilize the hinge loss function. Secondly, SVM relies on a powerful assumption that the linear or non-linear hyperplane had a large margin with the trained observations as much as possible. In practice, especially in quantitative finance, the observations are very often not linearly separable, leading to the violation of the linear separator. This leads to the soft and hard margins in the algorithm. Mathematically speaking, the hypothesis class of the support vector machine can be described as follow:

$$\mathcal{H}_{svm} = \sigma \bullet \mathcal{Q}_d(\theta, x, b) \tag{15}$$

$$\mathcal{Q}_d(\theta, x, b) = \{h_\theta(x) = \sum_{i=1}^{d} x_i\theta_i + b, \forall x, \theta \in \mathbb{R}^d, b \in \mathbb{R}\} \tag{16}$$

$$\ell_{svm}(\theta, x, y) = \max\{0, 1 - y(x'\theta + b)\} \tag{17}$$

Where $\ell_{hinge}(\theta, x, y) = \max\{0, 1 - y(x'\theta + b)$ is the hinge loss function and the mathematical notation $\bullet$ is a composite operand between two or more functions. From the expression of hypothesis class of Logistic Regression and SVM, it is not difficult to recognize the main difference between both algorithms. Firstly, SVM does not apply the sigma activation function. Secondly, while SVM attempts to utilize the hinge loss, Logistic Regression uses another loss function. However, there is one thing in common between the two algorithms. That is the loss function of both algorithms is convex. Similar to Logistic Regression, the optimization process to seek the trainable hyperparameter θ is quite straightforward.

In order to be more stable and prevent overfitting at all costs, the objective function can be implemented as below:

$$\mathcal{L}_{\mathcal{S}}(\theta) + \frac{\gamma}{2}\|\theta\|^2 = \frac{1}{m}\sum_{i=1}^{m} \max\{0, 1 - y_i(x_i'\theta + b)\} + \frac{\gamma}{2}\|\theta\|^2 \tag{18}$$

Where γ is a regularization hyperparameter.

The trainable parameter θ is sought via the optimization process of the equation (18). Thanks to the convexity of the loss function, $\mathcal{L}_{\mathcal{S}}(\theta) + \frac{\gamma}{2}\|\theta\|^2$ is a convex function as well. Hence, the optimization process of the equation can be done easily via the numerical or traditional method. The objective function, $\mathcal{L}_{\mathcal{S}}(\theta) + \frac{\gamma}{2}\|\theta\|^2$, could be optimized numerically via several optimization algorithms like stochastic gradient descent or gradient descent.

2.2.3 Deep Learning

Deep learning is a sub-branch of Machine learning utilizing a wide range architectures of neural networks and cutting edge optimization techniques. Deep learning has power, flexibility and is believed to be implemented effectively across many disciplines ranging from social science to finance due to the ability to capture the non-linearity relationship

between variables, which classical linear-based models are not able to do that.

Deep learning has become increasingly important in the analysis of financial data due to its ability to model complex, non-linear relationships and uncover hidden patterns within large datasets. Financial data, characterized by high volatility and noise, presents unique challenges that traditional statistical methods may not effectively address. Deep learning models, particularly those based on neural networks such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, are well-suited for time-series forecasting, which is critical in predicting stock prices, market trends, and economic indicators. These models can capture dependencies across time, enabling them to make more accurate predictions by learning from past data. Additionally, deep learning models benefit from the integration of large volumes of diverse data sources, including historical prices, trading volumes, and even non-traditional data like news sentiment and social media trends, to improve prediction accuracy. However, the application of deep learning in finance also requires careful consideration of model interpretability and the risk of overfitting, given the potential impact on trading strategies and investment decisions as Fischer and Krauss (2018). To maximize effectiveness, deep learning models should be combined with domain knowledge, rigorous validation, and robust regularization techniques to ensure they generalize well to unseen data and remain adaptable to the ever-changing financial markets.

There are various files could be utilised by deep learning, such as Computer Vision, Medicine, Recommendation systems, or Quantitative Finance. Its application could vary from human speech recognition, image detection to stock price prediction. The universal of deep learning is increasingly numerous, and in the scope of the thesis, it is impossible to integrate all of them. In this section, the general architecture of "Recurrent Neural Network" (RNN) and its variants will be presented. Those all revolve around analyzing financial time series, proving ideas of how they relate to the well-known technique in financial econometrics. Deep learning has gained popularity in time-series model, for instance, Paul (1988), Zaiyong, Chrys, and Paul (1991), Julian and Crhis (1998), Zhang and Min (2005), Sepp and Jurgen (1997), or Guoqiang, Patuwo, and Michael (1998) have proposed neural network time-series models in the financial or commodity market. In this thesis, the recurrent neural network models are presented as a non-linear generalization of the classical financial econometrics models as ARIMA(p, d, q), or AR(p). More concretely, the RNNs will provide a picture of the non-linear relationship between variables through time, which could be better to reflex the complexity in the financial market.

2.2.3.1 The Recurrent Neural Network algrorithm

Generally, a simple RNN describes a function between two sequences x_t, y_t, and the solution is to find a function satisfying the equation:

$$y_{t+h} = f(\mathbb{X}_{t,t-j}), h \geq 1, j \geq 1 \quad (19)$$

$$\mathbb{X}_{t,t-j} = \{\mathbb{X}_t, \mathbb{X}_{t-1}, \ldots, \mathbb{X}_{t-j}\} \quad (20)$$

Where y_{t+h} is the output of the model, h $((h \geq 1))$ implies the number of steps ahead forecasted in the future. $\mathbb{X}_{t,t-j}$ is the collection of lagged values of the input. It is worth

noticing that $\mathbb{X}_{t,t-j}$ could be multivariate time-series data. Hence, the RNN-like model is very flexible and efficient in explaining the impact of multivariate time-series data on different time-series data.
At a glance, people may argue that the RNN is identical to other multivariate structural autoregressive model as AR, ARIMA, or VAR. Nevertheless, the elegant beauty lies in the function $f(x)$ in the equation (19). In RNN, it is more flexible than a simple linear function. Its form could be tanh, sigmoid, or other non-linear functions. Matthew et al (2020) have pointed out that RNNs are indeed a non-linear extension of the classical financial econometrics model, which makes RNNs more versatile to capture the dynamic movement of price in the captial market. Mathematically speaking, the general formula of RNN could be presented as below:

$$y_{t+h} = \sigma^{(2)}(W_z^{(2)} z_t + b^{(2)}) + \epsilon_t \tag{21}$$
$$z_{t-j} = \sigma^{(1)}(W_x^{(1)} x_{t-j} + W_z^{(1)} z_{t-j-1}) + b^{(1)}) \tag{22}$$

Where z_{t-j}, x_{t-j}, and y_{t+h} are hidden state variables, input, and output respectively. h indicates the number of steps ahead forecasted in the future. While $W_x^{(1)}$ is a connection matrix between the input x_{t-j} and the hidden state z_{t-j}, $W_z^{(1)}$ connects the hidden state z_{t-j} and its lagged value, and $W_z^{(2)}$ bridges the hidden state z_t and the output y_{t+h}. $W_x^{(1)}, W_z^{(1)}$, and $W_z^{(2)}$ are time-invariant. $\sigma^{(1)}, \sigma^{(2)}$ are non-linear activation functions. The activation function $\sigma^{(1)} and \sigma^{(2)}$ create the elegant beauty and complexity for the RNN model, making the RNN model outstanding with other linear econometrics models as AR(p), MA(q), or ARIMA(p, d, q). Finally, ϵ_t plays a role as a noise.

Figure 2 has demonstrated the architecture of an RNN model. There most interesting part of an RNN model compared with other fully-connected neural networks is the sharing trainable parameters mechanism. It is straightforward to verify that, in figure 2, they all use the same matrix of coefficients from the input x_{t-i} to the hidden states z_{t-i} and another same matrix of coefficients for the hidden states and its lagged values. For the sake of clarity, a simple recurrent neural network is assumed to have a lag equal to 5, 2 hidden layers, 2 hidden unit state in each layer, and predict an individual value in the future. Thanks to the sharing trainable parameters mechanism, the RNN model only needs $2 \cdot 1 + 2 \cdot 2 + 2 \cdot 1$ trainable parameters for the first hidden layer and the input, $2 \cdot 2 + 2 \cdot 2 + 2 \cdot 1$ trainable parameters between the 2^{nd} and 1^{st} hidden layer, and $2 \cdot 1 + 1$ trainable parameters for the last layer and output. In total, the RNN needs 21 trainable paramters. In the case of the fully-connected neural network, the model will need $5 \cdot 5 + 1 + 5 \cdot 5 + 1 + 1 \cdot 5 + 1$, which is equal to 58 trainable parameters. Therefore, while the fully-connected neural network needs 58 trainable parameters, the RNN only needs 21 ones. The parsimony of trainable coefficients is always a motivation to alter the structure of the classical fully-connected neural network. Also, less trainable parameters are likely to have less propensity to overfit and reduce the training time.

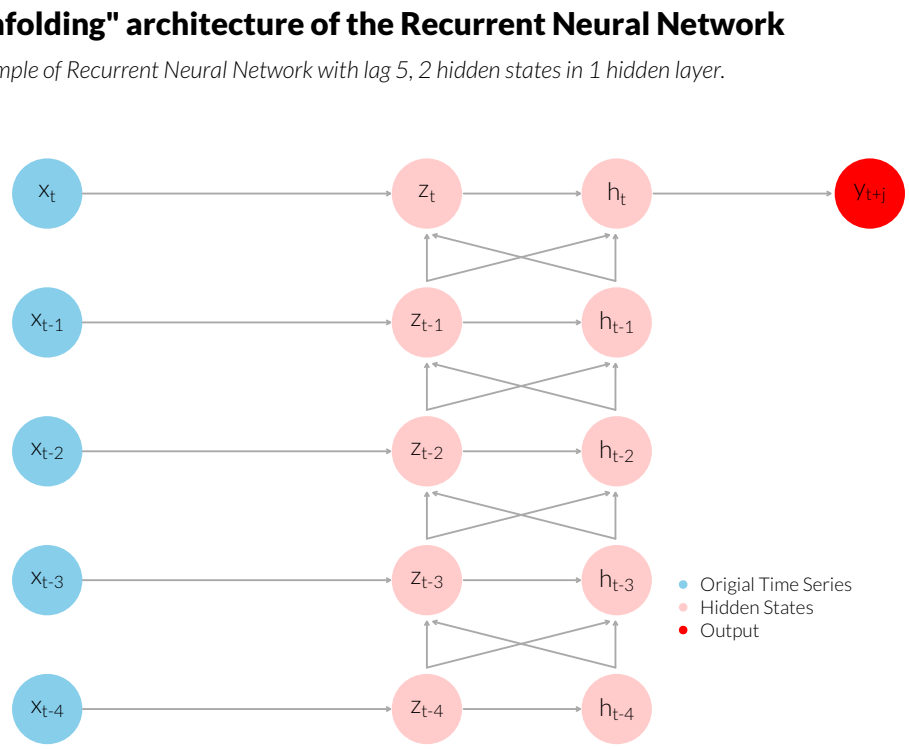

Figure 2: The unfolding architecture of the Recurrent Neural Network

2.2.3.2 The Long Short Term Memory algorithm

The Long Short Term Memory (LSTM) is a variant of the recurrent neural network model. While it inherited some terminologies from simple RNN, it enjoyed a new concept of a longer memory cell c_t. In addition to the long-term memory cell c_t, LSTM introduced four gates, namely main, input, output, and forget gate. Without such gated information, the LSTM is indeed a simple RNN. Accroding to Aurélien (2019), the architecture of LSTM can be depicted in figure 3. Mathematically speaking, the architecture of LSTM can be explained as below:

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) && (23)\\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) && (24)\\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) && (25)\\
g_t &= tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g) && (26)\\
c_t &= f_t \otimes c_{t-1} + i_t \otimes g_t && (27)\\
y_t &= h_t = o_t \otimes tanh(c_t) && (28)
\end{aligned}
$$

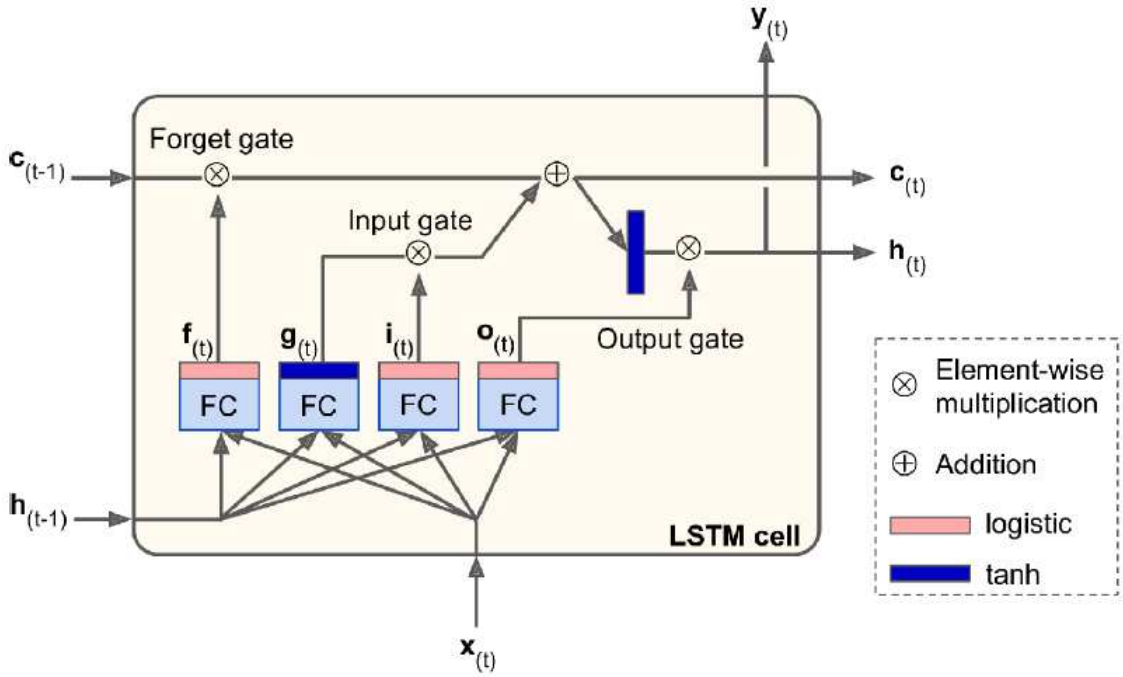

Figure 3: The unfolding architecture of the Long Short Term Memory

Where W_{xi} and W_{hi} are the connection matrices between the input x_t, the previous hidden state h_{t-1} and the input gate i_t respectively. The coefficient b_i could be considered as the intercept of the input gate. Similarly, matrices W_{xf} and W_{hf} bridge the gap between the input information x_t, the previous hidden state h_{t-1}, and the forget gate f_t . The coefficient b_f could be considered as an intercept of this gate. W_{xo} and W_{ho} connect the input information x_t, the previous hidden state h_{t-1} and the output gate o_t with the intercept b_o. W_{xg}, W_{hg} are the connection matrices between input information x_t, the previous hidden state h_{t-1} and the main gate g_t with the intercept b_g. $W_{xi}, W_{hi}, W_{xf}, W_{hf}, W_{xo}, W_{ho}, W_{xg}, W_{hg}, b_i, b_f, b_o$, and b_g are all time-invariant. Finally, $\sigma(\theta)$ and $tanh(\theta)$ is a sigma and tanh activation function, and the mathematical notation $\otimes$ is an element-wise product.

In figure 3, the information of the long-term memory cell c_t is connected with the previous lag c_{t-1} through the forget gate f_t and the product of the information of the main gate g_t and the input gate i_t. More precisely, the information of the long-term memory cell c_t is filtered from the last one, dropping some unnecessary information from the past, then added some more meaningful information from the gate i_t and g_t. Informally, the long-term memory cell c_t is wise enough to take the salient information from the past, then combined it with reliable information from the input x_t and the previous hidden state h_{t-1}. Finally, the cell c_t is filtered by a tanh activation function, then controlled by the output gate o_t before passing to the output y_t.

More precisely, the workflow of the LSTM model can be followed below steps:

- Step 1: A linear combination of the input x_t and the previous hidden state h_{t-1} is fed to the tanh activation function, producing the value of main gate g_t;

- Step 2: A linear combination of the input x_t and the previous hidden state h_{t-1} is fed to the logistic activation function, producing the value of forget gate f_t;
- Step 3: Repeat step 2, but the result leads to the value of the input gate i_t;
- Step 4: Similarly step 3, but the result lead to the value of the output gate o_t;
- Step 5: Compute the long-term memory cell c_t by its past value and the information of i_t, g_t, and f_t by the formula in (25) equation;
- Step 6: Compute the value of output y_t by the value of c_t with the tanh activation and the information of o_t by the formula in (26) equation.

2.2.3.3 The Gated Recurrent Unit algorithm

The Gated Recurrent Unit (GRU) is a simplified version of the LSTM. The common things between LSTM and GRU are forgetting and controlling mechanisms where the model dropped or added some information from the historical data. However, in GRU, there is no extra long memory cell c_t as in LSTM. Similarly, the GRU model makes use of the gate mechanism, says controller z_t, reset r_t, and main gate g_t. According to Aurélien (2019), the structure of GRU can be explained as below:

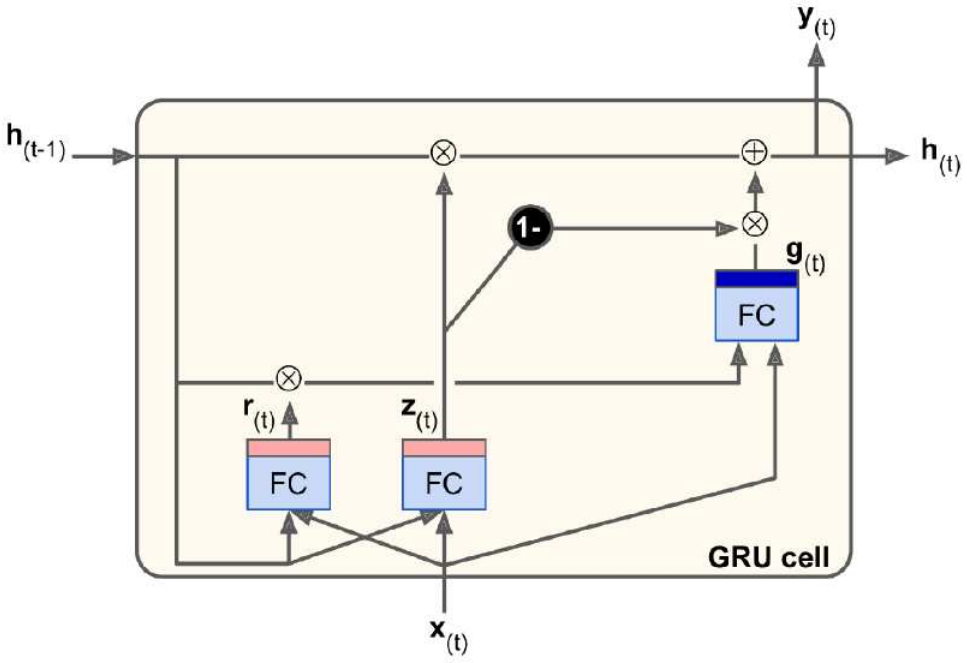

Figure 4: The unfolding architecture of the Gated Recurrent Unit

Aurélien (2019) also proposed the mathematical underpinnings of GRU model as below:

$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \quad (29)$$

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \quad (30)$$

$$g_t = tanh(W_{xg}x_t + W_{hg}(r_t \otimes h_{t-1}) + b_g) \quad (31)$$

$$h_t = z_t \otimes h_{t-1} + (1 - z_t) \otimes g_t \quad (32)$$

Where W_{xz} and W_{hz} are the connection matrices between the input information x_t and the previous hidden state h_{t-1}. The coefficient b_z could be considered as an intercept of this gate. Similarly, the matrices W_{xr} and W_{hr} bridge the input information x_t and the previous hidden state h_{t-1} with an intercept b_r. W_{xg} and W_{hg} are the connection matrices between the input information x_t and a part of the previous hidden state h_{t-1} filter by the value of reset gate r_t. It is well-noted that all of the matrices and the intercepts, namely $W_{xz}, W_{hz}, W_{xr}, W_{hr}, W_{xg}, W_{hg}, b_z, b_r$, and b_g are time-invariant. $\sigma(\theta)$ and $tanh(\theta)$ are a sigma and tanh activation function. The mathematical notation $\otimes$ is an element-wise product.

The GRU model is indeed a simplified version of the LSTM model as the long-term memory cell c_t is dropped. Also, from figure 4, the information of the input x_t and the previous hidden state h_{t-1} only fed to 3 gates instead of 4. Moreover, the controller gate z_t not only controls the value of the previous hidden state h_{t-1} producing the value of h_t, but also controls the value of main gate g_t through the component $1-z_t$. When the value of z_t is very close to 1, g_t is interrupted and has no impact on the output y_t, and vice versa, when the value of z_t is very close to 0, the previous hidden state h_{t-1} has no impact on the output y_t. More precisely, the workflow of the GRU model can be followed below steps:

- Step 1: A linear combination of the input x_t and the previous hidden state h_{t-1} is fed to the logistic activation function, producing the value of reset gate r_t;
- Step 2: Similarly step 1, but the result leads to the value of controller gate z_t;
- Step 3: A linear combination of the input x_t and a part of previous hidden state h_{t-1} filtered by the reset gate r_t is fed to the tanh activation function, producing the value of main gate g_t.
- Step 4: A linear combination of the previous hidden state h_{t-1} and a part of g_t controlled by the value of $1-z_t$, producing the value of output y_t.

Chapter 3: Methodologies Development and Data Scope

3.1 Details in methodology development

The 8-step procedure is employed in this reseach for any traded financial products. The procedure can be described in details as below:

- Step 1: Split a time-series data into training and testing part;
- Step 2: Scale both training and testing part by min-max or normal measurement;
- Step 3: Convert a time series data in both parts into a supervised learning form;
- Step 4: Cross-validation in training part to seek the optimal hyper-parameters;
- Step 5: Apply ℓ_2 norm regularization to prevent overfitting phenomenon;
- Step 6: Train and validate the model in the training and testing part respectively;
- Step 7: Construct trading strategy from the results from testing part;
- Step 8: Backtesting trading strategy.

Step 1 elaborates the process of separating the whole time-series data into the training and testing part, guaranteeing that the trained model is validated independently and can prevent a look-ahead biased error. Additionally, the accuracy of the model in the testing environment, which has not been exposed to before, can evaluate its ability in practice. It is of paramount importance as it can prevent the overfitting phenomenon. Moreover, the ratio of splitting is quite flexible, and in this scope of the thesis, 90% of the observation for training and the rest 10% of the observation for testing. 90% of the data will be utilized to understand IBM's price movement and the rest 10% could be used to verify the model. Step 2 proposed an approach to scale down the original data. In machine learning, scaling original data play a pivotal role in the convergence of the loss function. Also, the model will run quicker and more stable with the scaled input.

Steps 3 and 4 describe the process of bringing time-series data into the proper format and seeking the optimal hyperparameters. As mentioned previously, the time-series data, per se, does not have a supervised learning form, hence, any supervised learning algorithms ranging from classical machine learning to deep learning can not apply. Fortunately, in the scope of the thesis, I have demonstrated an effective way to convert it. As a result, the following step, cross-validation, has been carried out easily and effectively. Besides, in addition to trainable parameters, machine learning models always contain many hyperparameters, which is not sought via the optimization process. The previous section has also pointed out that cross-validation is a straightforward approach to get an optimal hyperparameters.

According to Shalev and Ben (2014), one of the most noticeable points in designing a machine learning model is an overfitting phenomenon, which predominantly happens in

real world comlex problem. Formally, overfit is the situation in which the true loss, $\mathcal{L}_D(\theta)$ is high while its training loss, $\mathcal{L}_S(\theta)$ is low, where the train and true loss can be expressed mathematically as:

$$\mathcal{L}_D(\theta) = \mathbb{E}(\ell(\theta, x, y)) \tag{33}$$

$$\mathcal{L}_S(\theta) = \frac{1}{m}\sum_{i=1}^{m} \ell(\theta, x_i, y_i) \tag{34}$$

Where $\forall x, x_i \in \mathbb{R}^d$, $\forall y, y_i \in \mathbb{R}$, and $\ell(\theta, x, y)$ is a loss function. It is well-noted that while the variable pair (x, y) could be considered as any possible value of the input and output, a pair (x_i, y_i) means the training points in our data. Thus, step 5 has proposed a regularization technique to prevent an overfitting phenomenon. I also pointed out that the regularization parameter γ is quite sensitive. It is a trade-off factor to balance the training loss, $\mathcal{L}_S(\theta)$ and the true loss, $\mathcal{L}_D(\theta)$.

The model is trained in step 6 could be a regression or a classification. If the model is a regression, implemented in the deep learning models, the outcome needs to be interpreted in "trading language". Fortunately, the trading interpretation is straightforward, and it relies completely on the prediction of a given stock price. In detail, if the predicted price is larger than the actual previous price, the machine will assign 1, corresponding to the buy position, otherwise, the model will assign -1 corresponding to the sell position. If the model is the classification, the direct outcome of the tested data can be considered immediately as a "trading" language. Nonetheless, it is well-noted that "short selling" is prohibited in many stock exchanges, as a result, in the scope of the thesis, I will not place any short position. Hence, if the trading signal is -1, I will do nothing instead of placing a short position. Ultimately, in step 7, I will have a full set of pseudo trading signals. This is taken into consideration as the real decision of the investor in the trading context. This decision is completely decided by the models and does not rely on human intervention.

Lastly, step 8 verifies the efficacy of each strategy by the intraday vectorized backtest. In detail, if the investors go a long position, he or she will receive the total daily return no matter what it is a positive or negative return. Via a vectorized backtest, it is easy to conclude which strategy is better. A vectorized backtest is ubiquitous in practice due to its simplicity and effectiveness. Also, the equity curves are exhibited showing the accumulated profit or loss of each strategy. Additionally, I will also investigate the drawdown indicator of each strategy. This is another important point in quantitative trading. The drawdown implies the maximum loss of the strategy during the testing period. It reflects the effectiveness of the model in terms of risk management. A good drawdown indicator implies that the strategy is wise enough not to place a false position, hence, preventing unexpected loss. Each model-based strategy is also compared closely with the "buy and hold" strategy, which helps me to verify the better strategy between them.

3.2 Models estimations

Model estimation is a critical step in the development of both machine learning and deep learning models, where the goal is to find the best set of parameters that minimize the error in predictions. In machine learning, models such as linear regression, logistic regression, and support vector machines rely on well-established estimation techniques like Maximum Likelihood Estimation (MLE) or Ordinary Least Squares (OLS). These methods involve optimizing a predefined loss function, such as mean squared error for regression tasks, to determine the most appropriate model parameters. On the other hand, deep learning models, such as neural networks, require more complex estimation processes due to their highly non-linear structures and large number of parameters. These models typically use gradient-based optimization techniques, such as Stochastic Gradient Descent (SGD) and its variants (e.g., Adam), to iteratively adjust weights and biases in order to minimize the loss function. The estimation process in deep learning is computationally intensive and often requires large datasets and substantial computational resources. Regularization techniques like $l2$ regularization and dropout are also employed to prevent overfitting during the estimation process, ensuring that the model generalizes well to new, unseen data as Goodfellow et al (2016) and LeCun et al (2015) have responded. The choice of estimation technique and optimization strategy is crucial in determining the accuracy, efficiency, and robustness of both machine learning and deep learning models in real-world applications.

3.3 Data Sources and Data-Fetching Tools.

3.3.1 Python as a versatile tool for Deep Learning

Python has established itself as a premier programming language for deep learning due to its simplicity, versatility, and the extensive ecosystem of libraries and frameworks it supports. One of the key reasons Python excels in deep learning is its ease of use, which allows both beginners and experienced developers to quickly prototype, develop, and deploy models. Python's syntax is clean and readable, making it accessible to a wide range of users, including those without a deep background in programming. Furthermore, Python boasts powerful libraries such as TensorFlow, PyTorch, and Keras, which provide high-level APIs for building and training complex neural networks. These libraries are not only optimized for performance but also offer robust tools for visualization, debugging, and deployment, making the development process more efficient. Python's extensive community support also means that developers have access to a wealth of resources, tutorials, and pre-trained models, accelerating the learning curve and enabling faster innovation. Additionally, Python's integration with other tools for data manipulation and analysis, such as NumPy, pandas, and Matplotlib, allows for seamless data preprocessing and visualization, further enhancing its utility in deep learning projects (Chollet (2018) and Paszke et al (2019)). Consequently, Python has become the go-to language for deep learning, empowering researchers and practitioners to push the boundaries of what is possible in AI.

In this research, Python proved to be an invaluable tool for handling data extraction and

preprocessing tasks. Utilizing Python, I accessed and retrieved data from various public sources, such as Yahoo Financial Data through advanced libraries like pandas and numpy. Once the raw data was obtained, I employed Python's powerful data manipulation libraries to clean and process it. Libraries such as pandas provided efficient data structures and functions to handle missing values, remove duplicates, and perform transformations necessary for ensuring data quality McKinney (2010). Additionally, NumPy was used for numerical operations and array manipulations, allowing for the handling of large-scale data efficiently. The preprocessing involved normalizing the data, encoding categorical variables, and creating new features to enhance the dataset's suitability for analysis and modeling.

By leveraging Python's ecosystem, I was able to streamline these tasks and ensure that the data used for subsequent analysis and modeling was accurate, consistent, and ready for deep learning applications. This meticulous approach to data cleaning and preparation, facilitated by Python's robust tools, significantly improved the performance and reliability of the predictive models built in my research and save much time in those steps. (Van Rossum and Drake (2009))

3.3.2 Public Financial Data Source

Public financial data sources, such as Yahoo Finance, are instrumental in providing accessible and comprehensive financial information for both individual investors and researchers. Yahoo Finance offers a wide array of data, including historical stock prices, real-time market quotes, currency pairs, financial statements, and economic indicators, making it a valuable resource for analyzing market trends and developing trading strategies. The platform's API allows for automated data retrieval, which can be seamlessly integrated into financial models and analytical workflows. By leveraging Python libraries such as yfinance or pandas_datareader, users can efficiently fetch and manipulate data from Yahoo Finance, facilitating tasks such as backtesting trading strategies and performing statistical analysis (Yahoo Finance API (20024)). The availability of this data in a user-friendly format supports diverse applications, from academic research to practical investment analysis, making Yahoo Finance a key player in the financial data ecosystem. The broad scope of data and ease of access provided by Yahoo Finance enable users to stay informed and make data-driven decisions in the dynamic world of finance, Yarovaya (2019)

During my research, the data used is freely published by "Yahoo Finance", and can be easily downloaded from their website or fetched automatically by a simple Python script. Theoretically , the content of the thesis can apply to any financial traded products across multiple asset classes ranging from stock and bond to forex. Nonetheless, each type of asset needs a different rigorous treatment.

The selected sample is the stock price of an International Business Machine company. In the thesis, the period of IBM's stock price is from January 2000 to March 2021. The training phase is from the beginning to February 2020, and the testing phase lasts from February 2020 towards March 2021. The training and testing parts are not overlapped,

ensuring the independence of the validation of the model. Also, all steps to verify the efficacy of the model-based strategy are evaluated only during the testing period. It is worth noticing that the data is already in time-ordered form, hence, the testing period needs to follow the training period chronologically. Besides, it is impossible to shuffle data before splitting it like other machine learning techniques.

3.3.3 Statistics descriptive of data

Tables 2, 3, and 4 have come up with the statistical summaries of the whole, training, and testing period respectively. Tables 2, 3, and 4 have illustrated that the standard deviation (volatility) of the testing period is $\$ 8.05$, which is by far lower than the standard deviation during the training period. It could be a fortunate point for the process of training and validating the machine learning model. The model has more opportunities to understand the behaviors of the price movement during the training phase, enabling it to be wiser to cope with the unseen situation in the testing phase.

Statistics	Value	Statistics	Value
Mean	96.50	1^{st} quantile	62.50
Standard Deviation	34.10	2^{nd} quantile	91.70
Min	35.10	3^{rd} quantile	127.60
Max	159.50	4^{th} quantile	159.50

Source: Author

Table 2: The statistical summary of the IBM's stock price during the whole period

Statistics	Value	Statistics	Value
Mean	93.50	1^{st} quantile	60.75
Standard Deviation	34.70	2^{nd} quantile	84.05
Min	35.10	3^{rd} quantile	128.02
Max	159.50	4^{th} quantile	159.50

Source: Author

Table 3: The statistical summary of the IBM's stock price during the training period

Statistics	Value	Statistics	Value
Mean	112.92	1^{st} quantile	118.95
Standard Deviation	8.05	2^{nd} quantile	123.60
Min	88.78	3^{rd} quantile	127.01
Max	147.15	4^{th} quantile	147.15

Source: Author

Table 4: The statistical summary of the IBM's stock price during the testing period

In figure 5, there are 5322 observations in total, including 4789 observations, equal to 95% of the original data, for training and the last 533 observations, equal to the last 5% of the original data. Figure 5 has shown that, during the training period, the stock has been more volatile than compared with the testing period. IBM's stock price rises dramatically from around $ 35 to the peak of roughly $ 160 during the training period, equal to approximately 357% in price appreciation. However, IBM's stock price only fluctuated around $ 100 to $ 140. Being more volatile during the training period and less volatile in the testing phase is indeed an advantage. The model will have a chance to understand what is the behavior of the surge in the price, particularly during the period that the price skyrocketed from $ 35 to $ 160. On the on hand, the model makes use of that valuable information to train to go long positions. On the other hand, the model uses the information during the price depreciation, for instance around March 2017, to train not to place buy positions. All of these , on the first side, increase the profit as much as possible, on the other side, mitigate the loss from the downturn of the price.

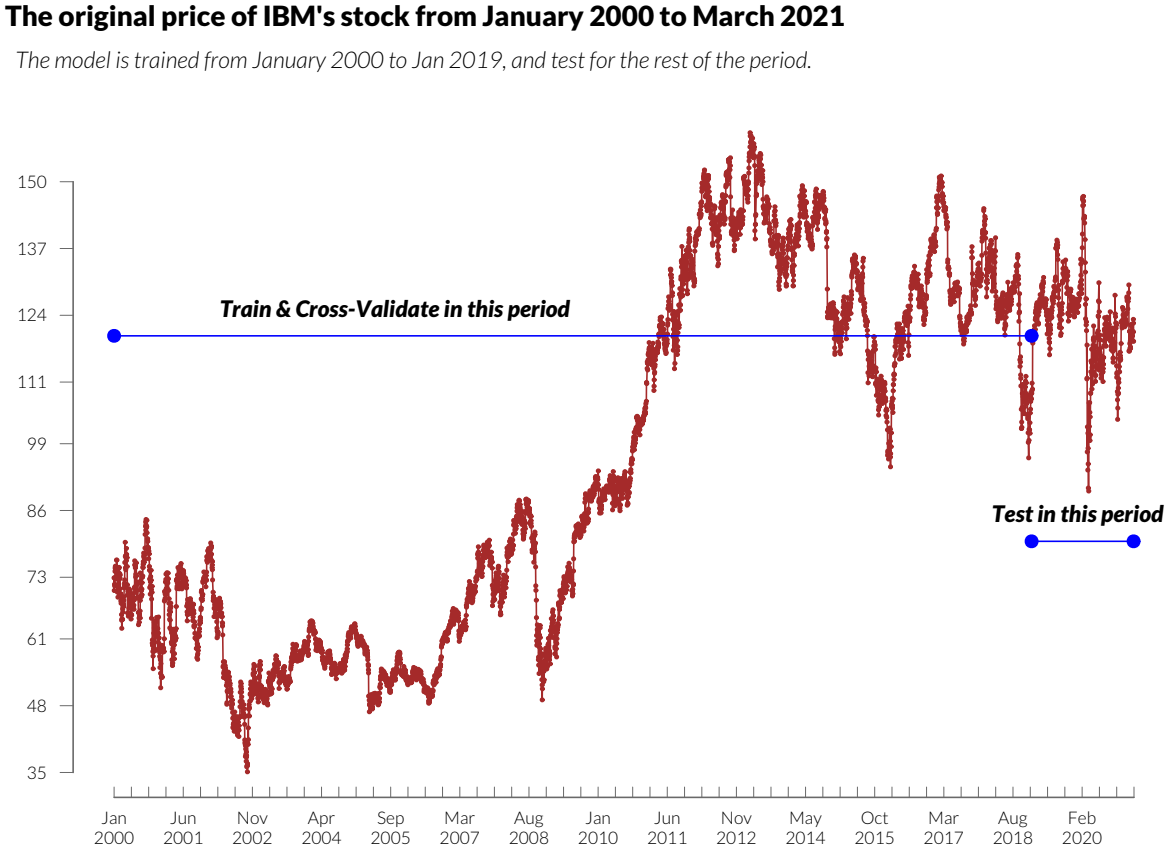

Figure 5: The original stock price of IBM from January 2000 to March 2021

Part 3: Empirical Results, Trustworthiness in AI Finance and Conclusion

Chapter 4: Empirical Results

In this section, I will present the backtest from $02/2019$ to $03/2021$ over the strategies based on machine learning models discussed earlier. I will present two main types of strategies, namely deep learning-based strategy including RNN, LSTM, GRU, and classical machine learning-based strategy including Logistic Regression and SVM.

Theoretically, the algorithms based on machine learning models can be applied for any stocks or other traded financial products as Forex, Commodity, CryptoCurrency, or even contract for difference-formed products. Nonetheless, each asset needs rigorous analysis, and there does not exist a "panacea" for all. Here, the backtest will be performed on the stock of International Business Machine Corporation (IBM). The model will be trained from $01/2000$ to $02/2019$ then validated for the "out of sample" period from $02/2019$ to$03/2021$, which have not been seen during the training period.

The deep learning models are trained from price to price. It means that the outcome of the models is one step ahead forecasted price of IBM's stock. Consequently, in the deep learning-based strategies, I will interpret the predicted price into a trading strategy. The strategy is straightforward. It depended completely on the predicted price of the given stock. On the other hand, in the classical machine learning-based strategy, the outcome is indeed the signal of strategy, for instance, the pair $(1, -1)$ correspond to long, and short positions respectively. In reality, "short-selling" is prohibited in some exchanges, therefore, for the sake of fairness, "short-sell" is restricted. My rule-based strategy contains long-only positions. The position is automatically terminated at the end of the day after placing a long position at the beginning of the traded day. Therefore, every day if I have a long position, I can compute the daily return. The pseudo trading strategy based on models will create long-only positions. I also pointed out the "hit rate" meaning that how many percent of the long position has been placed accurately.

Additionally, the equity curve is presented as evident proof of the profit or loss of each strategy. In the equity curve, the accumulated return of the "buy and hold" and the model-based strategy converted into compound annual growth rate term (CAGR) have been illustrated on the daily basis. All of which are meaningful in comparing the effectiveness of each strategy. While the backtest is quite transparent and straightforward, it did not include slippage fee or the effect of spread, which were likely to have a huge impact on the profit of the investment.

This section will begin with a thorough analysis of the prediction of each method. In each method, I will illustrate the graph of the predicted and original price with the root-mean-square error criterion. The root-mean-square error is effective in recognizing the performance of the prediction. Of course, the smaller the root mean squared error is, the better the prediction is. An excellent prediction also means an excellent strategy.

4.1 Empirical results of deep learning-based algorithms

In this section, I will present the price prediction of the stock IBM by three deep learning models. The price will be predicted in a one-step forecast for every 10 lags during 2 years from Feb 2019 to March 2021. The prediction period will be validated independently with the training period, which ensures that the model-based strategy can prevent look-ahead bias and work in practice. There are some advantages of this approach, say the investors usually have a whole picture of the traded products shortly. Also, the model can be more flexible in that it can predict more than a one-step forecast with a very rigorous treatment, but it is out of the scope of this thesis. The decision can be made quite easily based on the predicted information.

At a glance, from figures 6, 7, and 8, while there are several points that the models can not capture, for instance, during February and March 2020, the model, overall, is quite so good in predicting the trend of the stock, especially from May 2020 towards the end of the period. RNN can describe an overall trend of IBM's stock price. However, the disadvantage of all trained about is that the loss function used is the mean square error, meaning that the model only cares about the penalty if the distance between the forecasted and original price is wide. The model does not pay attention to the direction of the price, which plays an important role in trading.

The workflow of this section can be considerd as a complete 3-step procedure:

- Step 1: Rigorous treatment of IBM's stock price prediction;
- Step 2: Rigorous treatment of pseudo long position of IBM's stock;
- Step 3: Rigorous treatment of equity curves of model-based strategies and the buy and hold strategy.

Firstly, I present a rigorous treatment of IBM's stock price prediction by RNN, LSTM, and GRU model. Figures 6, 7, and 8 have shown the prediction of IBM's stock price. At a glance, all of the graphs posed similar patterns, however, they have different levels of the root-mean-square error, which is a popular criterion in time series forecast. Undoubtedly, the smaller the root-mean-square error is, the better the model is. Although all of the models could not capture 100% of the movement of IBM's stock price, the models could predict very well the main trend of the stock price

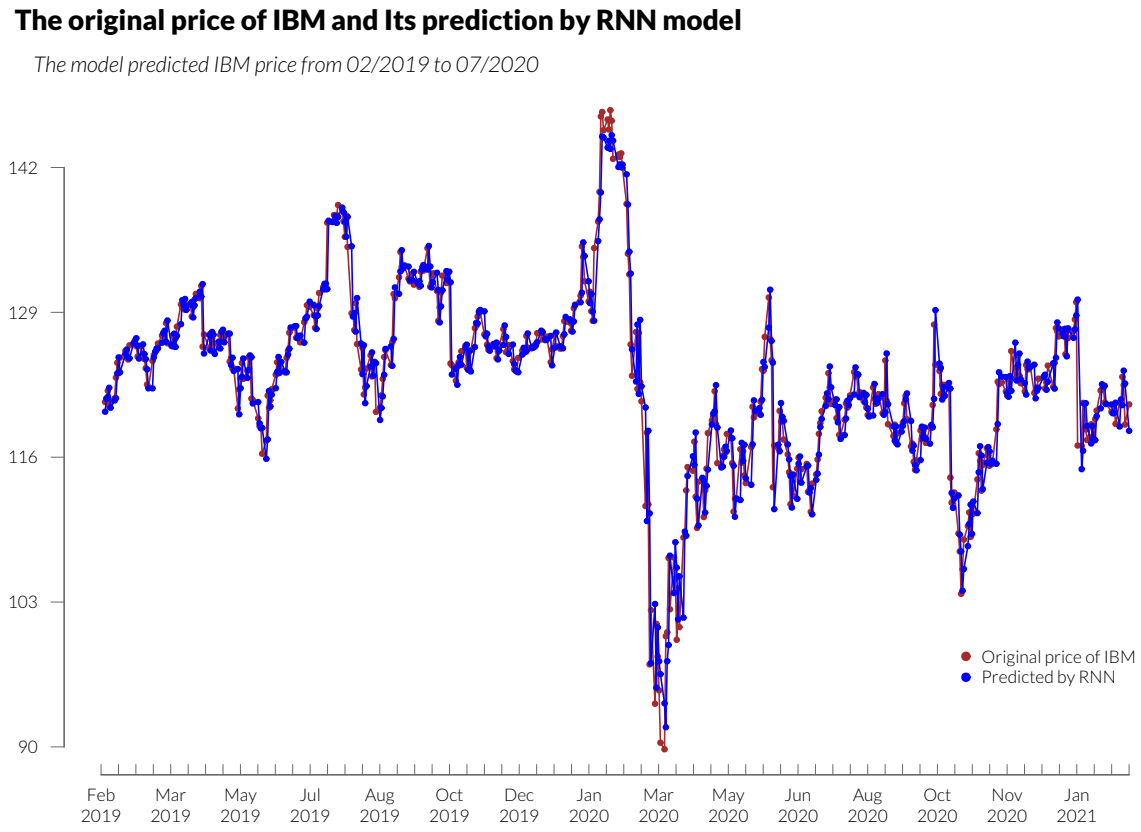

Figure 6: The original stock price of IBM and its predicted value by RNN

In detail, the models could understand the uptrend and downtrend of the price movement, particularly during a plummet in the price during March 2020. This is one of the advantages of the deep learning model in forecasting time series data. As mentioned previously, a recurrent neural network and its variants are indeed a combination of non-linear activation functions, connecting the time-series input, hidden states, and the output. Formally, the model attempts to connect all chronological events by non-linear functions, which enable the models outstanding in understanding the behavior of the time-series data, particularly in the financial market. This mechanism is completely different from the AR(p), MA(q), or ARIMA(p, d, q) models where all chronological events are connected by a linear function.

Figure 6 posed a one-step forecast of IBM's stock price with a lag equal to 10. Formally, during the testing period, for every 10 timestamps, I will produce a one-step forecast. This procedure will be rolled over till the end of the testing period. This technique ensures that I have a full set of predictions from Feb 2019 to March 2021 blundering like look-ahead bias error. The prediction in figures 6, 7, and 8 posed the root mean square error of $ 2.40, $ 2.34, and $ 2.36 of the RNN, LSTM, and GRU models respectively.

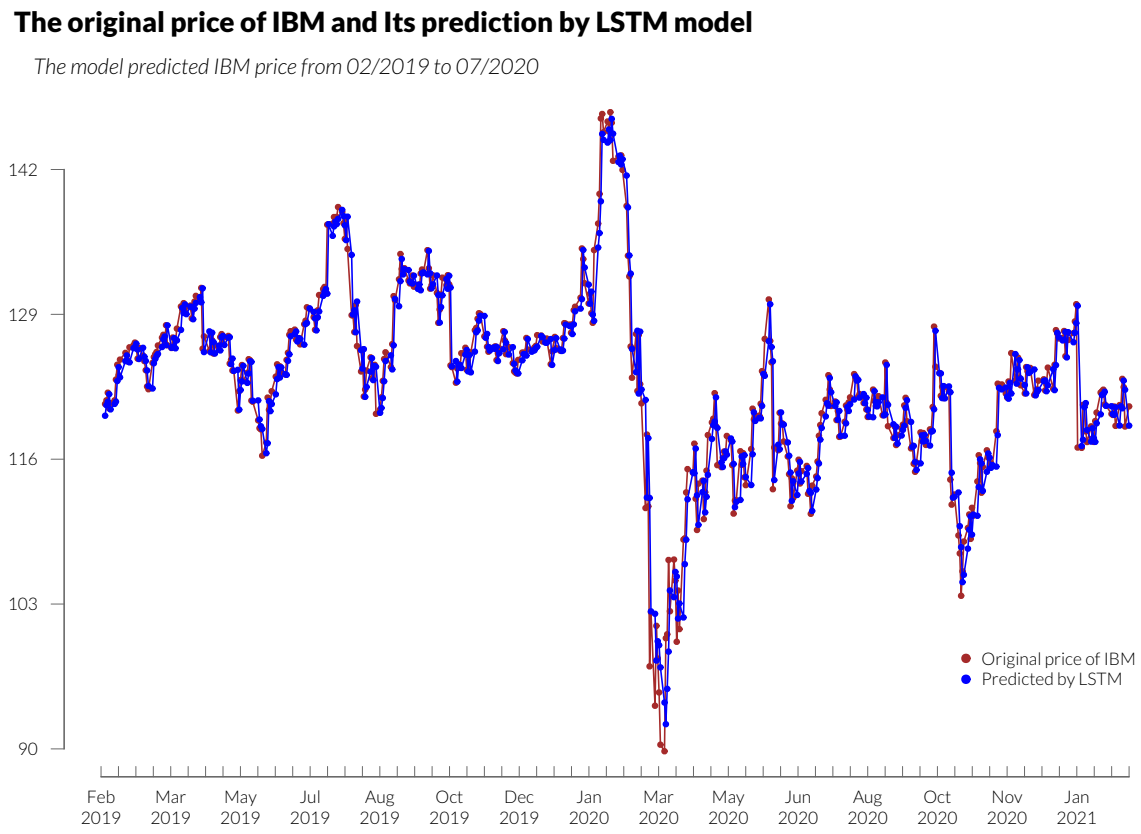

Figure 7: The original stock price of IBM and its prediction by LSTM

The summaries of the root-mean-square error of each model can be described as below:

Model	RMSE
RNN	2.40
LSTM	2.34
GRU	2.36

Source: Author

Table 5: The root-mean-square error of the forecast of IBM's stock price of RNN, LSTM, and GRU model

The root-mean-square error of the LSTM model reached the lowest level, meaning that LSTM model has the highest precision in prediction, followed by GRU and RNN model.

Secondly, upon the completion of the price prediction, the interpreted strategy is straightforward. If the predicted price is greater than the previous price, I will place a long position, otherwise, I would do nothing. It is well-noted that "short sell" is restricted in several stock exchanges in practice.

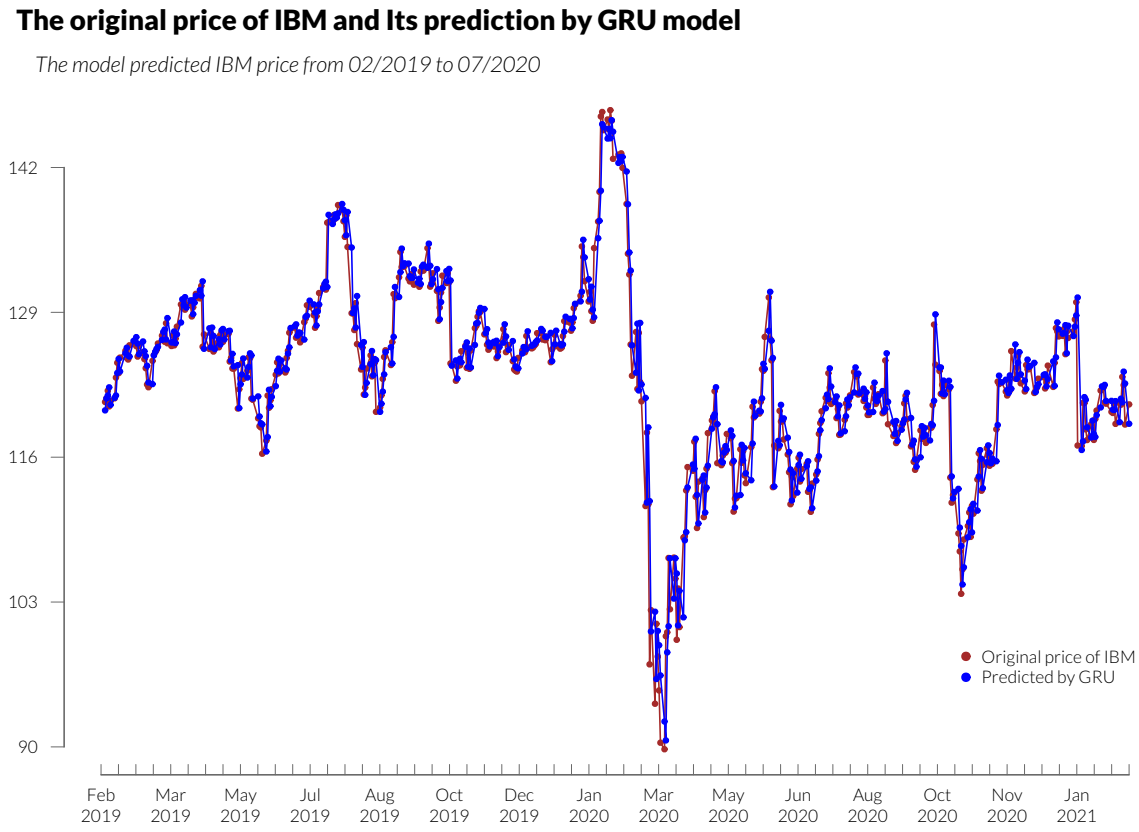

Figure 8: The original stock price of IBM and its prediction by GRU

From figures 9, 10, and 11, there are $373, 363$, and 498 placed long orders out of 523 orders during the testing period from RNN, LSTM, and GRU-based strategy respectively. The rates of placed long order of those models are 71.33%, 69.40%, and 95.20%. The GRU-based strategy has the highest rate of placing position during the testing period, followed by RNN and LSTM-based strategy. Naturally, as the investor only cares about long-only positions, hence, there are only two natural issues posed in the trading context. Firstly, the rate of accurate long-position. And secondly, the rate of the wrong long-position. Informally, if there would be a jump in IBM's stock price, the investor would place a long order, otherwise, the investor would do nothing instead of placing a buy order. Undoubtedly, if the investor can control these two rates, he or she will gain an excellent amount of profit.

Figure 9 has presented that there are 194 accurate long positions out of 279 "ideal" long positions, leading to a hit rate of 70% in the RNN-based strategy. Figures 10 and 11 have also pointed out that there are 191 out of 279 equal to 69% and 269 out of 279 equal to 96.40% accurate long position of the LSTM and GRU-based strategy respectively. The GRU model posed the highest rate of hit rate, followed by RNN and LSTM. The "ideal buy position" could be taken into consideration as the actual increment of the price. If there would have been a jump in a given stock price, the investor should have placed a long order instead of doing nothing.

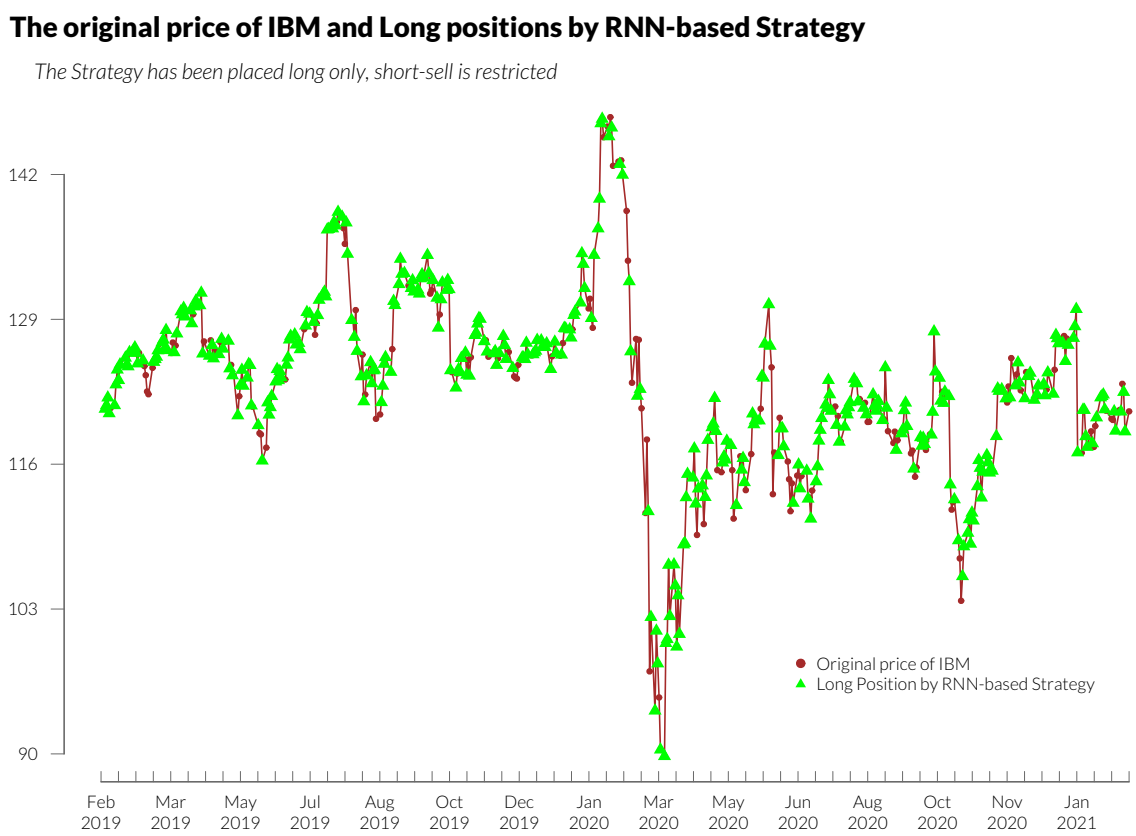

Figure 9: The long position by RNN-based strategy

Strategy	The hit rate
RNN-based strategy	70.0%
LSTM-based strategy	68.5%
GRU-based strategy	96.5%

Source: Author

Table 6: The hit rate of RNN, LSTM, and GRU-based strategy

However, there are still many "false positions" due to the inaccurate predicted prices. From figures 9, 10, and 11, it is easy to detect that there are several downturn periods but the model still placed long positions. These could have not placed any position as the "short-sell" is prohibited. Doing nothing would prevent the investor from losing profit. For instance, around March 2020, there are many long positions issued by RNN, LSTM, and GRU model although IBM's stock price plunges. This could be explained because there exists a false prediction of IBM's stock price, resulting in the false position of the model-based strategy.

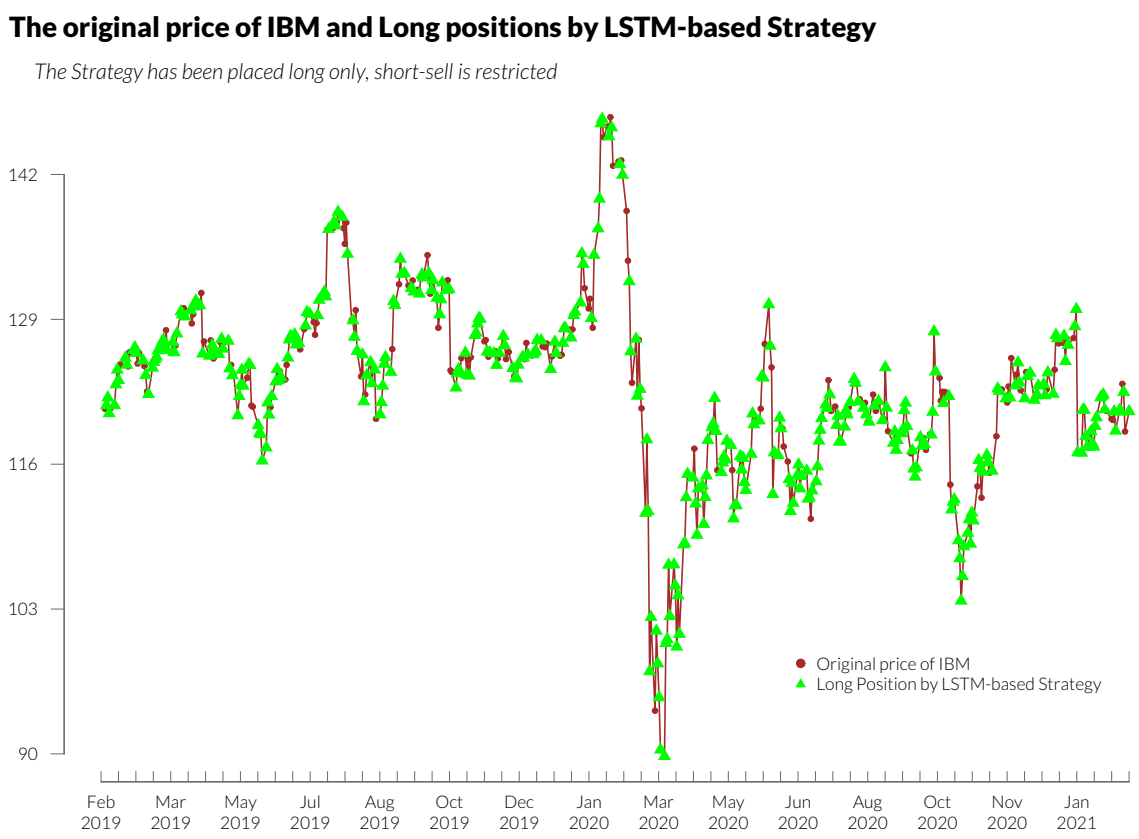

Figure 10: The long position by LSTM-based strategy

This is partly due to the mean squared loss function of the model RNN. While the loss function paid special attention to the breadth of the predicted and true observation, it almost ignored the side of the observation. For example, if the previous price is 120, the actual price today is $\$119.9$, and the predicted price is $\$120.01$, the model will suggest a long position instead of doing nothing. That explains why during the period around August or September 2019, the price of IBM plunged from $\$140$ to $\$89$, but the rule-based strategy always put long positions.

Figures 12, 13, and 14 presented the equity curves of the deep learning model-based strategy and the buy and hold strategy. The equity curves have shown the accumulated profit or loss on the daily basis, which could verify the efficacy of each strategy. It is well-noted that the profit or loss of the equity is computed through the vectorized backtest. In other words, If the investors go a long position, he or she will receive the whole return of that day no matter what it is a positive or negative return. At a glance, all of these graphs have demonstrated that the daily accumulated profit has been divided into three patterns, says the first phase from February 2019 to before March 2020, the second phase around March 2020, and the last phase from March 2020 towards the end. Overall, the first phase explains the outperformance of the buy and hold strategy before a plummet in profit of both strategies during the second phase. The last phase showed a by-far better result of the deep learning model-based strategy over the buy and hold strategy.

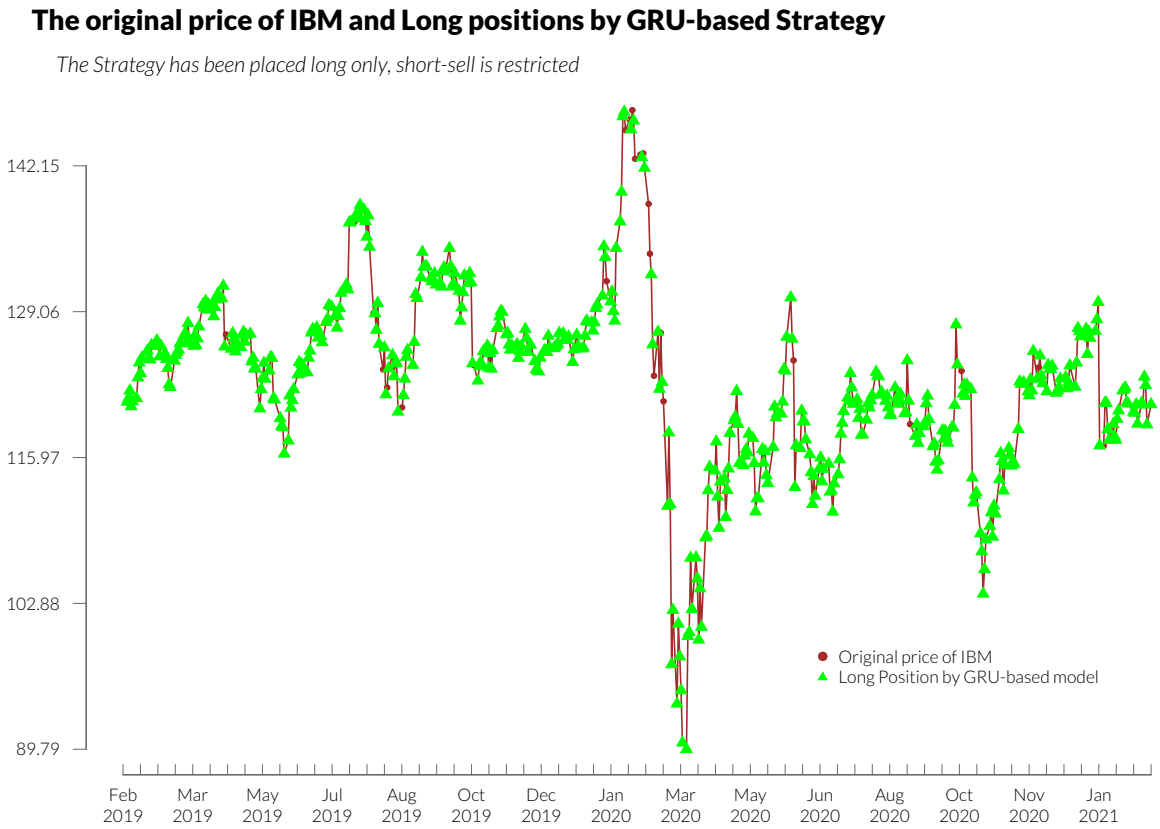

Figure 11: The long position by GRU-based strategy

Table 11 presents the summary of rate of false position in each model.

Strategy	False Position Rate
RNN-based strategy	73.60%
LSTM-based strategy	70.70%
GRU-based strategy	94.20%

Source: Author

Table 7: The rate of false position of RNN, LSTM, GRU-based strategy

From table 11, the GRU-based strategy makes the highest rate of false position, followed by RNN and LSTM-based strategy. This order, also, corresponds to the hit rate of the models. In other words, the more order placed by the model, the higher the accuracy and false rate of the order. Ideally, the more order placed by the model, the higher of accurate rate and the lower the false rate of the order. This is still a huge challenge of the machine learning-based strategy in algorithmic trading.

Finally, the equity curves are investigated to verify the efficacy of each strategy. The equity curve is derived directly from the intraday vectorized backtest. More precisely, the equity curve is computed by the daily return and its pseudo long position instructed

by the model. There are two perspectives taken into consideration from the equity curve, namely the accumulated profit from the effectiveness angle, and the drawdown from the risk management angle.

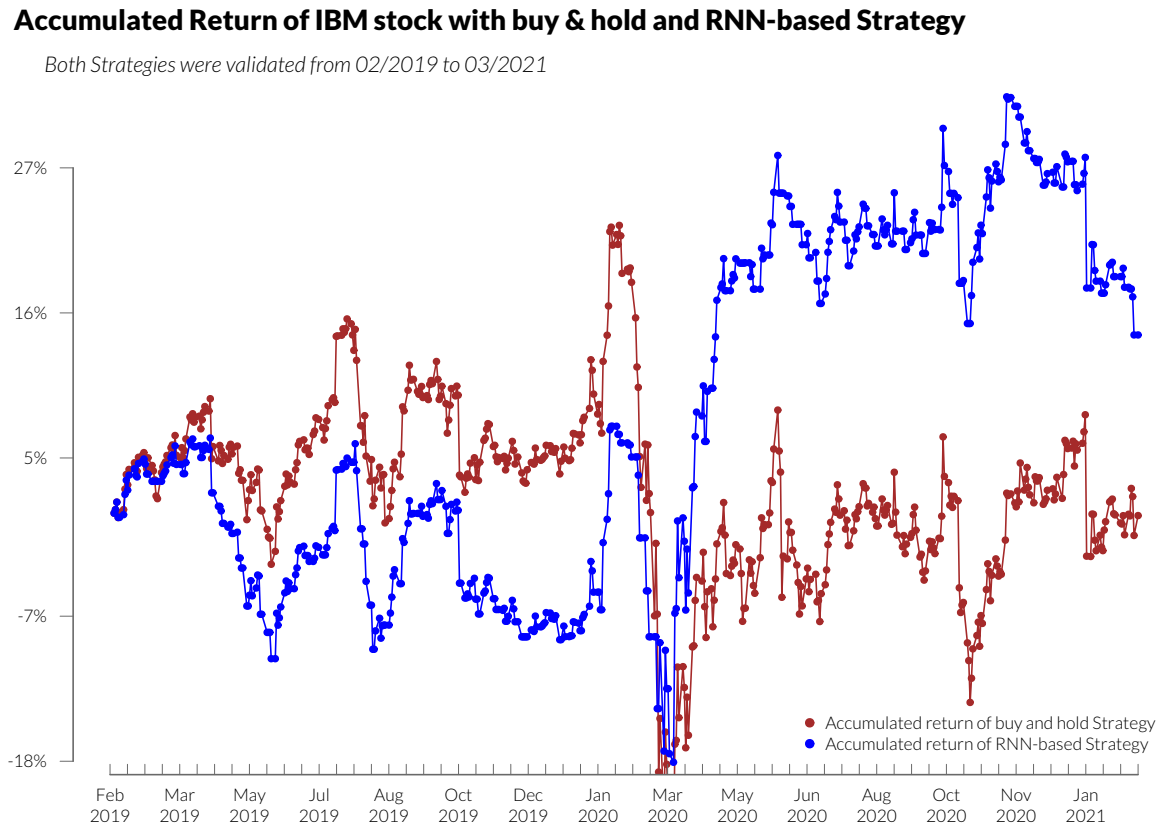

Figure 12: Equity Curve of buy & hold and RNN-based stragtegy

From figure 12, two equity curves, derived from two vectorized intraday backtest from RNN-based strategy and buy & hold strategy, are presented here. Both of which cover trading simulation from February 2019 to February 2021, which is roughly two years. The first equity curve demonstrated the accumulated return of the RNN-based strategy while the second one presented the accumulated profit of the buy and hold strategy. Both of the backtests used the respective "out of sample" observations which have not been exposed to during the training period. It is well-noted that while the backtest is straightforward, it does not contain any extra fee like transaction cost or slippage.

Figure 12 illustrated that, from February 2019 to March 2020, the RNN-based strategy maintained a low level of profit compared with the buy and hold strategy. During this time, the maximum profit of the buy and hold strategy reached around 25%, whereas RNN-based strategy only achieved roughly 5%. However, the situation changed dramatically after a plummet in IBM's stock price around May 2020. Figure 12 pointed out that while the equity curve of the buy and hold strategy dipped below the level of -25%, this figure of RNN-based strategy was only -15%. This could be explained that the RNN model could recognize the downturn of the price movement, hence, it attempted to put a curb on long positions, leading to a limit on loss. The model

outperforms after March 2020 towards the end of the period thanks to the recovery of IBM's stock price. At the end of the period, the RNN-based strategy achieved 14.3% accumulated profit corresponding to a CAGR of 10%, whereas the buy and hold strategy only attained around 0.6% corresponding to a CAGR of 0.4%. The RNN-based strategy outperformed due to the wise decision of buying position after recovery of IBM's stock price after the plunge in May 2019.

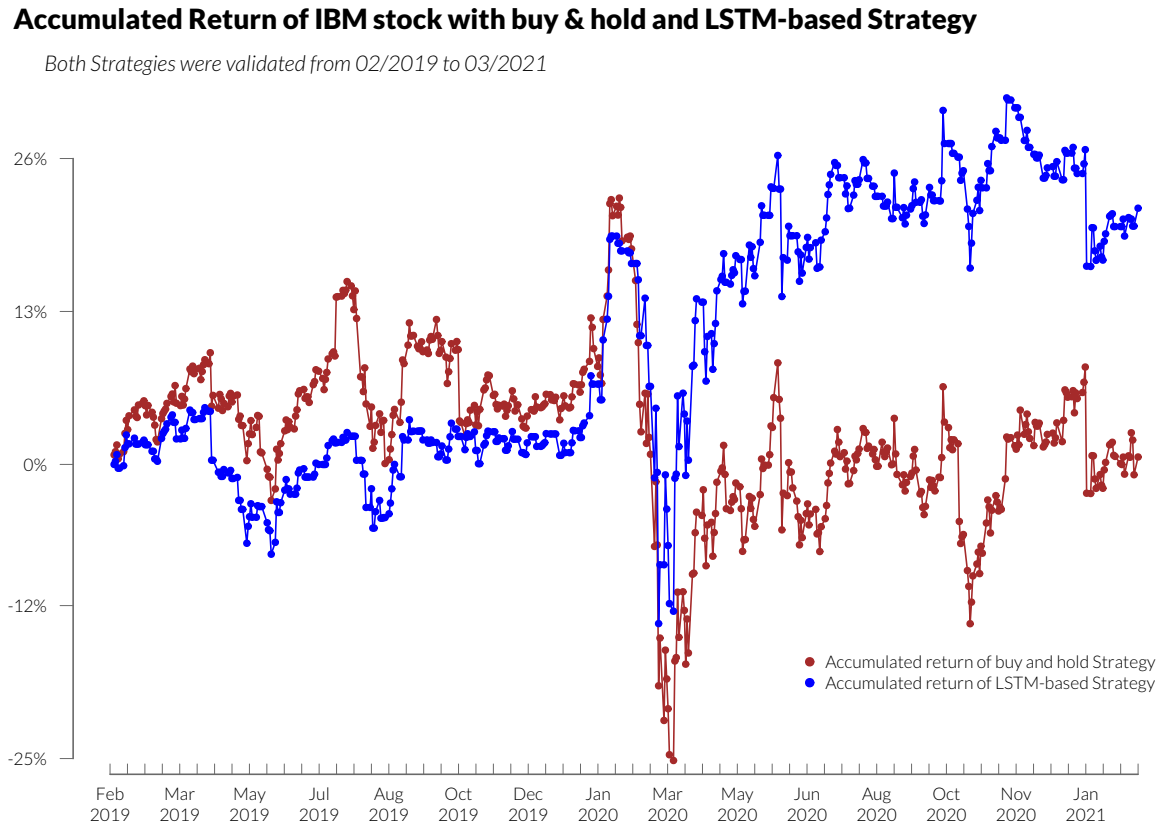

Figure 13: The equity curve of buy & hold and LSTM-based strategy

The key summaries of both strategies can be described as below:

Strategy	Accumulated Profit	Drawdown
RNN-based strategy	$14.30\%(6.00\%)$	-18.05%
Buy & Hold strategy	$0.60\%(0.40\%)$	-25.15%

Source: Author. Note: CAGR in parenthesis

Table 8: Key summaries of RNN-based and the Buy and hold strategy

Similarly, in figure 13, I presented two equity curves, derived from the intraday vectorized backtest of LSTM-based strategy and buy & hold strategy. At a glance, there is a distinctive point between the RNN-based strategy mentioned previously and

LSTM-based strategy. While the buy & hold strategy posted a higher level of profit from February 2019 to March 2020, the breadth between LSTM-based profit and buy & hold profit has been reduced compared with the breadth of RNN-based profit and buy & hold profit. Figure 12 and figure 13 have also demonstrated that the maximum loss of the RNN-based strategy around this time is approximately -10.20%, whereas this figure of LSTM-based strategy is only -7.62%. In other words, the LSTM-based strategy could perform better than the RNN-based strategy in the first phase of the period. Additionally, during the dramatic downturn of IBM's stock price, the maximum loss of LSTM-based strategy only went down a level of -13.5%, whereas this level of the buy and hold strategy and RNN-based strategy dipped around -25.15% and -18.07% respectively. Consequently, in terms of risk management, the LSTM-based strategy could perform better than the RNN-based strategy and buy and hold strategy.

From March 2019 to March 2021, the performance of the LSTM-based strategy is by far better than the buy and hold strategy. Besides the recovery of IBM's stock price from March 2019, the LSTM model is trained to detect the upturn or downturn of the price, hence, it has a good strategy to place positions. This leads to maximizing the accumulated profit. At the end of the period, figure 13 has shown the accumulated profit of LSTM-based strategy attained 21.77% equal to CAGR of 10.50% while the accumulated profit of the buy and hold strategy only achieved 0.6% equal to CAGR of 0.4%. The drawdown of the LSTM-based strategy is also better than the buy and hold strategy. Overall, the LSTM-based strategy has a better achievement than the buy and hold strategy. The key summaries of both strategies can be described as below:

Strategy	Accumulated Profit	Drawdown
LSTM-based strategy	$21.77\%(10.50\%)$	-13.50%
Buy & Hold strategy	$0.60\%(0.40\%)$	-25.15%

Source: Author. Note: CAGR in parenthesis

Table 9: Key summaries of LSTM-based and the Buy and hold strategy

Figure 14 also presented two equity curves derived from the daily vectorized backtest of the GRU-based strategy and the buy and hold strategy. While the overall trend of GRU-based and the buy and hold strategy shapes are similar to RNN-based or LSTM-based strategy, there are a couple of points that enable this pair outstanding. First of all, from February 2019 towards August 2019, the accumulated profit of GRU-based and the buy and hold strategy are almost identical. Then, the equity curve of the buy and hold strategy started increasing and performing better than its competitor. The maximum accumulated profit of the buy and hold strategy achieves 28.50% whereas the figure of the rival only reaches 21.80%. The scenario began changing dramatically since March 2020. Around March 2020, the accumulated profit of both strategies plunged at different levels. While the GRU-based strategy dipped around the level of 12% which is the lowest rate of ever happened strategies, this figure of the buy and hold strategy collapsed below the level of -25%. Clearly, in terms of risk management, the buy and hold

strategy managed to perform better than GRU-based strategy, and even LSTM and RNN-based strategy.

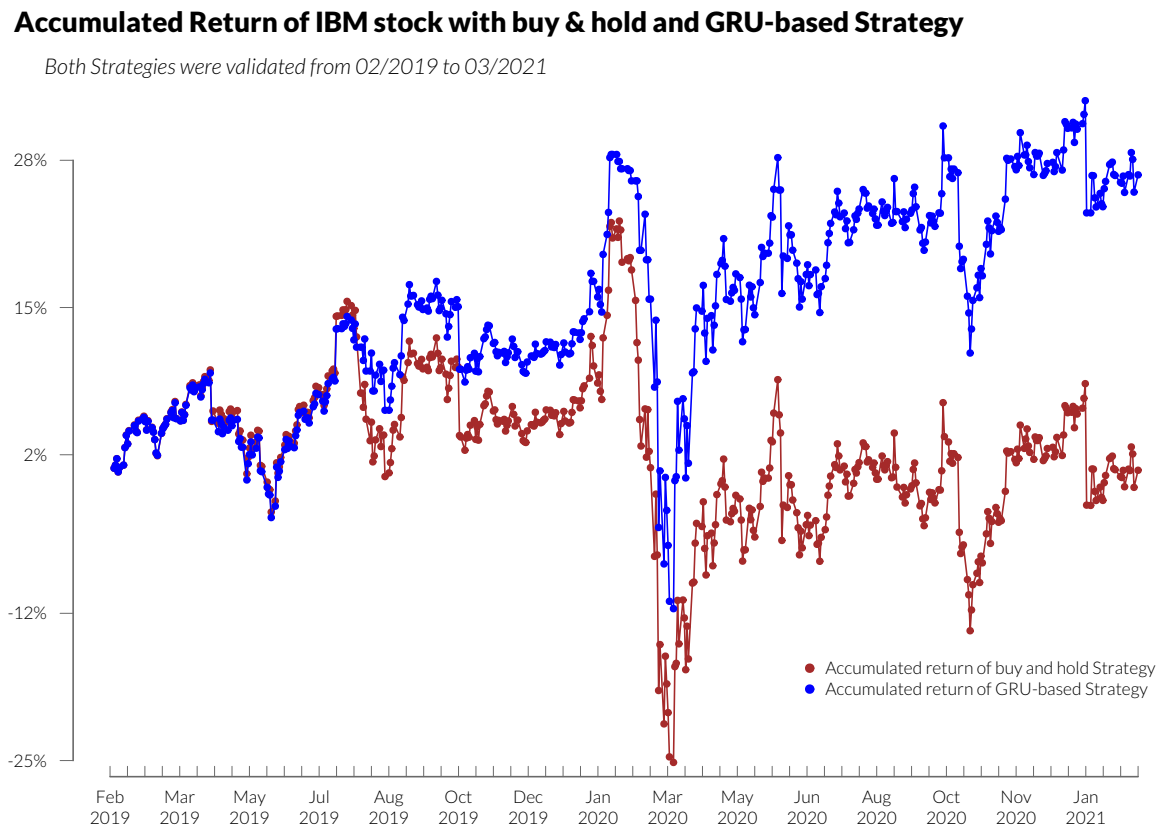

Figure 14: The equity curve of buy & hold and GRU-based strategy

Additionally, after March 2020, the equity curve of the GRU-based strategy started rising significantly. This could be explained by the recovery of IBM's stock price after a plummet. Besides, the GRU model also showed its efficacy in choosing the "wise" long position. At the end of the period, the accumulated profit of the GRU-based strategy attained the level of 26.70% equal to CAGR of 18.65%, which is the highest amongst all strategies, whereas the final accumulated profit of IBM only remained at the level of 0.6% equal to 0.40%. While the GRU-based strategy is not actually effective in risk management, the model has gained by-far better performance in seeking profit for the investors.

The key summaries of both strategies can be described as below:

Strategy	Accumulated Profit	Drawdown
GRU-based strategy	$26.71\%(18.65\%)$	-11.58%
Buy & Hold strategy	$0.60\%(0.40\%)$	-25.15%

Source: Author. Note: CAGR in parenthesis

Table 10: Key summaries GRU-based and the Buy and hold strategy

4.2 Results of two classical Machine Learning models

In this section, I will present the price prediction of the IBM stock by two classical machine learning models, namely Logistic Regression and Support Vector Machine learning. Unlike the model in the previous section, the two classical Machine Learning models will train a classification model. In other words, based on the numerical input, the model will produce the categorical outputs corresponding to the pair $(1, -1)$. The interpretation of a pair $(1, -1)$ is straightforward, says 1 corresponds to the long position and -1 corresponds to the short position. Although I have enough information for both long and short order, only long order will be utilized as "short-selling" is prohibited to ensure the fairness reason. The workflow of this section could be carried out in a complete 2-step procedure:

- Rigorous treatment of the pseudo long position derived from the classical machine learning models;
- Rigorous treatment of the equity curves of the model-based strategy and the buy and hold strategy.

Firstly, I present the rigorous treatment of the long-only position derived from the classical machine learning model. IBM's stock price direction will be predicted one-step forecast for every 10 lag information from Feb 2019 to March 2021. The prediction is, also, validated independently with the training period, which ensures that the model-based strategy can prevent look-ahead bias and work in reality. The classical machine learning-based strategy is simpler than the deep learning-based strategy as I do not need to interpret the strategy. It is well-noted that "short-selling" is not allowed, so, the information of 1 in the outcome is considered as a long position, and -1 means nothing.

Figure 15 has illustrated the long order issued by the LR model. In detail, there are 325 out of 523 orders during the testing period from February 2019 to March 2020. Similarly, from figure 16, there are 324 out of 523 orders of the SVM model during the testing period. The rates of placing long order of Logistic Regression and SVM are 62.15% and 61.95% respectively. Unlike deep learning models, the rate of placing long order of both classical machine learning models is quite similar. There are also two natural problems posed in the context of trading. Firstly, the rate of the accurate long position. And secondly, the rate of the false long position. Figures 15 and 16 have also pointed out that there are 182 and 180 accurate long positions out of 279 "ideal" long positions, leading to the hit rate equal to 65.25% and 64.5% of the LR and SVM-based strategy. Similar to the previous section, the "ideal" long position could be taken into consideration as the buy order when it would have an actual increment of the given stock price. Those figures of classical machine learning model-based strategies are slightly lower than the figures of RNN and LSTM-based strategy, and by far lower than the figure of GRU-based strategy.

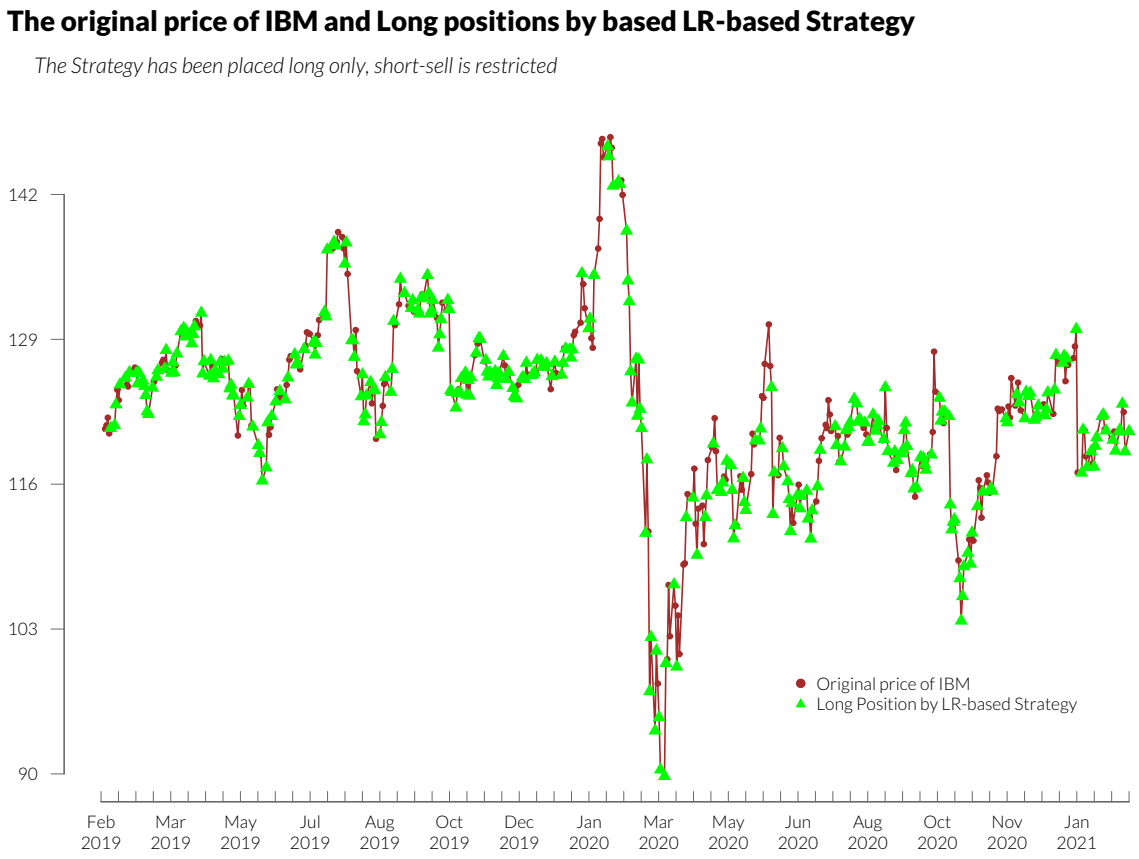

Figure 15: The long position by Logistic Regression-based strategy

The summaries of the hit rate of LR and SVM-based model are described as below:

Strategy	The hit rate
LR-based strategy	62.5%
SVM-based strategy	64.5%

Source: Author

Table 11: The hit rate of LR and SVM-based strategy

Moreover, figures 15 and 16 have also posed the problem of false long order due to inaccurate prediction of the direction. It is not difficult to detect that there are many long positions issued by both classical machine learning models during the downturn of IBM's stock price. As "short-selling" is not allowed, hence, there should have been nothing instead of placing a buy position. Apparently, not placing any order would prevent the investor from a huge loss. For example, during March 2020, there are still many long positions during a plummet of IBM's stock price from around $ 142 to roughly $ 90.

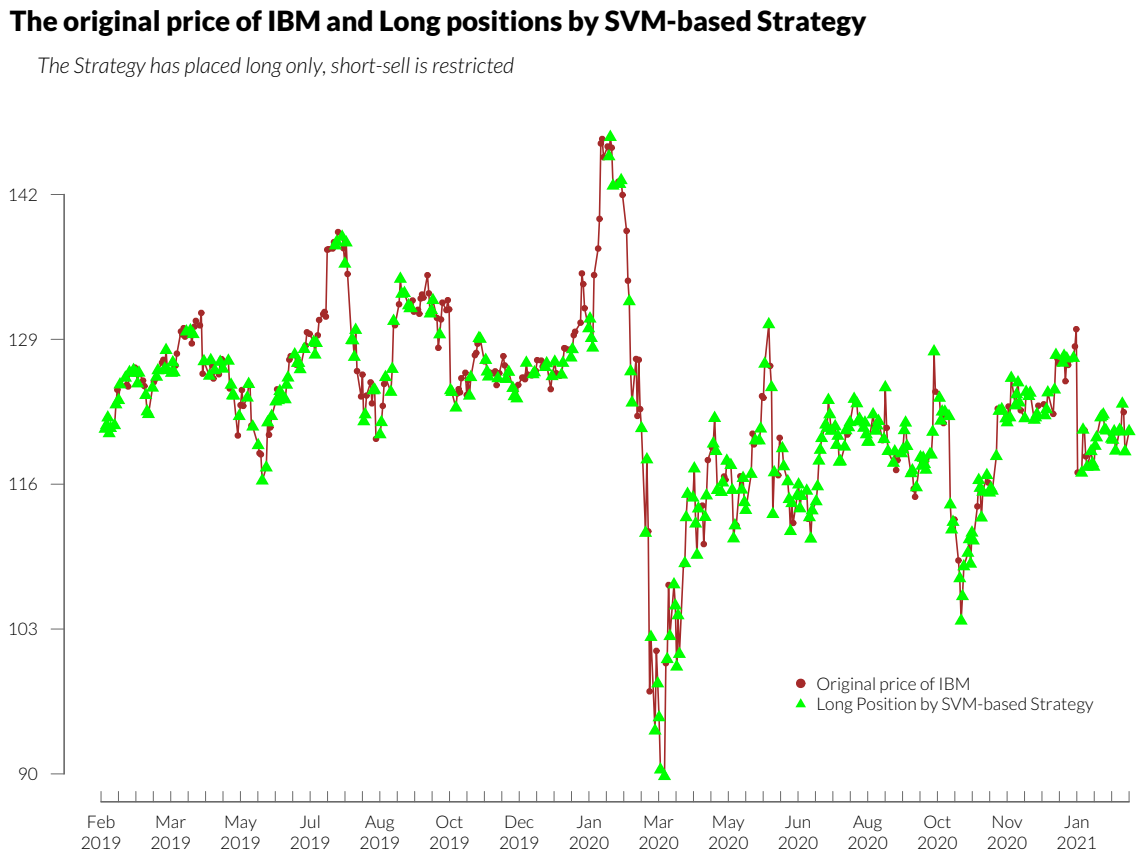

Figure 16: The long position by SVM-based strategy

The summary of rate of false position in each model can be described as below:

Strategy	False Position Rate
LR-based strategy	44.40%
SVM-based strategy	44.40%

Source: Author

Table 12: The rate of false position of LR and SVM-based strategy

Table 12 has shown that the rate of the false long position of each classical machine learning-based strategy has reached roughly 44.4%, which has been overall lower than the figure of deep learning-based models. It is also a good angle from the risk management perspective.

Secondly, I illustrate the equity curves derived from the vectorized backtest of LR and SVM-based strategy. Similar to the previous part, the equity curve is calculated from the daily return and its pseudo long position derived from the model-based strategy. There, also, are two perspectives that appear in the equity curve, namely the accumulated return and the drawdown of the strategy. Both reflect the efficacy and the risk management's angle of the strategy.

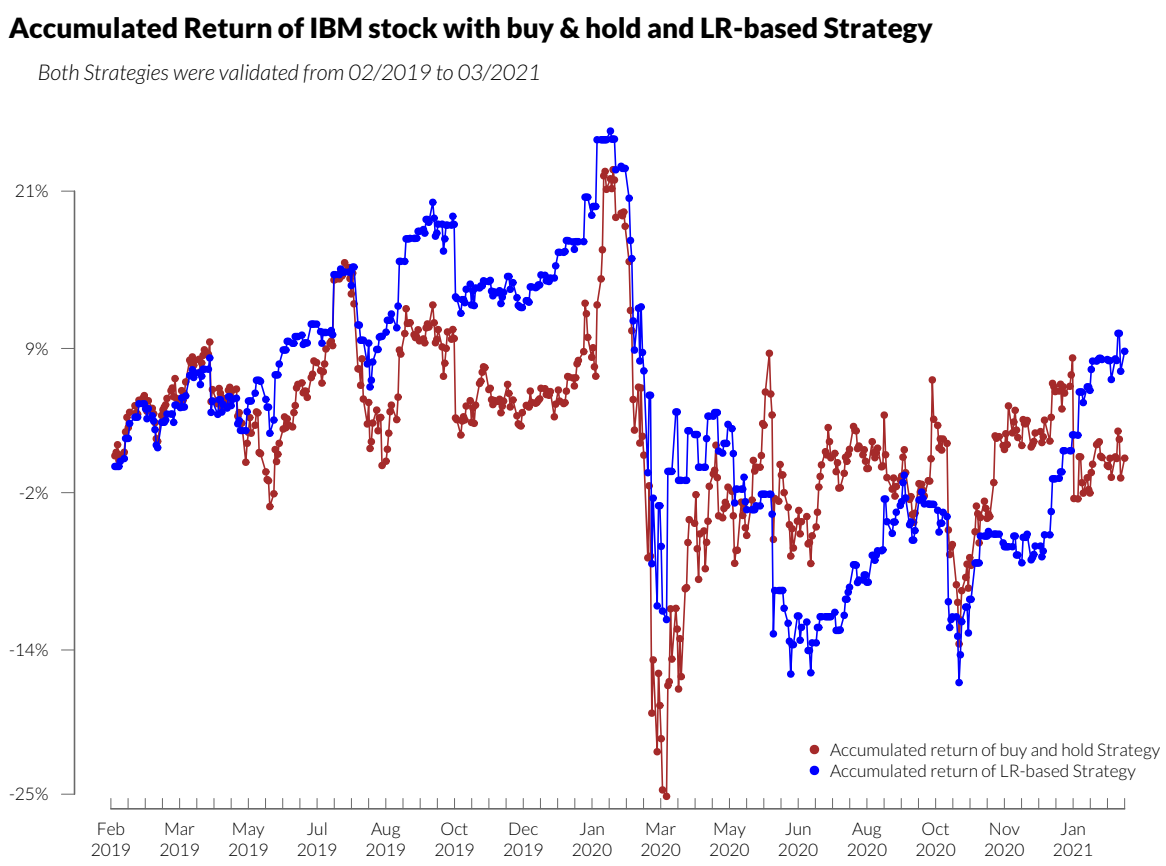

Figure 17: The equity curve of buy & hold and Logistic Regression-based strategy

From figure 17, two equity curves, derived from two vectorized intraday backtest from LR-based strategy and the buy and hold strategy, are presented here. Both of which cover trading simulation from February 2019 to February 2021, which is roughly 523 trading days. The first equity curve demonstrated the accumulated return of the LR-based strategy while the second one presented the accumulated profit of the buy and hold strategy. Both of the backtests used the respective "out of sample" observations which have not been exposed to during the training period. It is well-noted that while the backtest is straightforward, it does not contain any extra fee like transaction cost or slippage.

At a glance, figure 17 has presented that, from February 2019 to March 2020, the accumulated profit of the LR-based strategy maintained a quite similar level with the profit of the buy and hold strategy. From February 2019 to March 2020, the LR-based model often kept a higher level of accumulated profit than the buy and hold strategy, which is completely different from the case of deep learning-based strategy. During that period, the maximum accumulated profit of the LR-based strategy achieved roughly 25%, whereas the maximum profit of the buy and hold only attained around 25%. However, the situation changed dramatically after the plummet of IMB's stock price. During March 2020, the price of IBM stock plunged more than 38%, from \$ 146 to \$ 90. Consequently, the equity curve of the buy and hold strategy dipped below the level of -25%, while this figure of LR-based strategy was only around -12%. Apparently, in terms of risk management, the LR-based strategy has performed better than the strategy

of the rival. Nonetheless, the LR-based strategy could not keep the "wise" decision from March 2020 towards the end of the period. LR model could not be conscientious enough to place accurate orders. There are many buy position instead of doing nothing during this period, leading to a huge drop in the profit. Finally, LR-based strategy ended up earning 8.75% equal to a CAGR of 4.24%, which is much lower than the results of deep learning-based strategy. The key summaries of both strategies can be described as below:

Strategy	Accumulated Profit	Drawdown
LR-based strategy	$8.80\%(6.00\%)$	-18.05%
Buy & Hold strategy	$0.60\%(0.40\%)$	-25.15%

Source: Author. Note: CAGR in parenthesis

Table 13: Key summaries of LR-based and the Buy and hold strategy

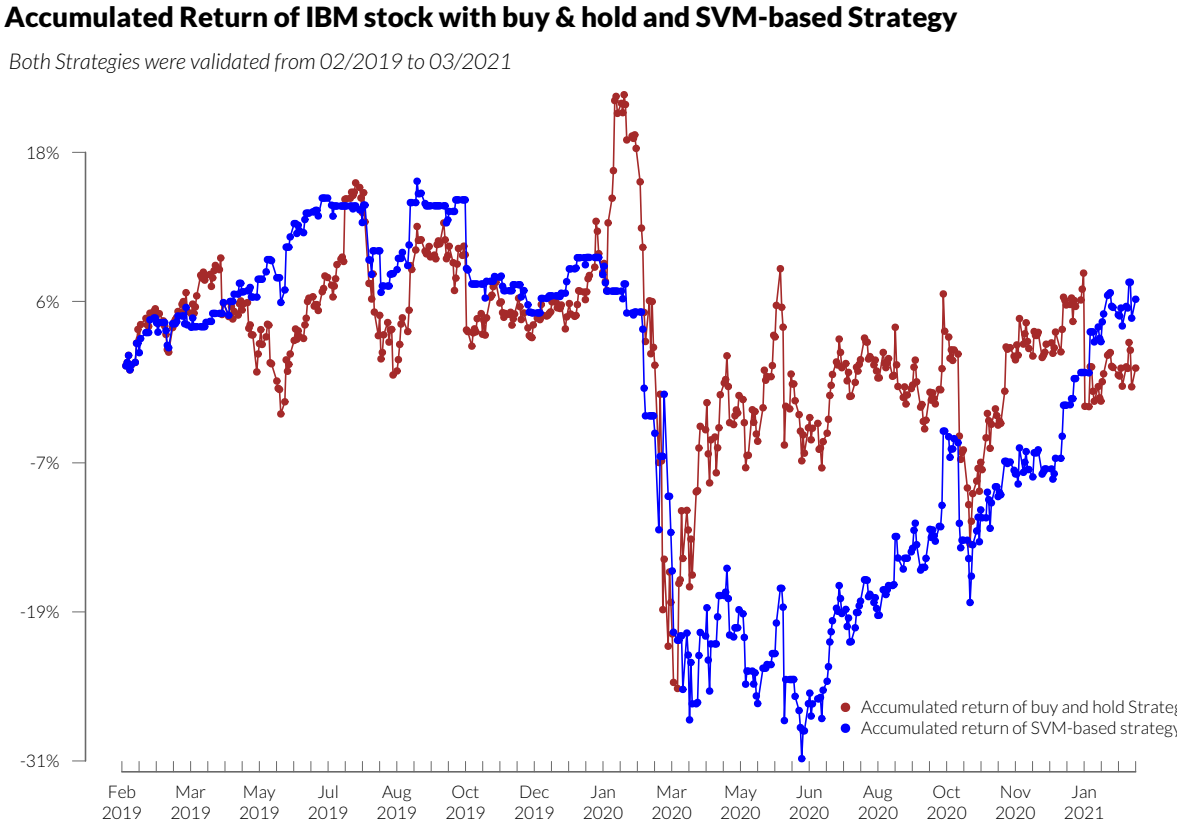

Figure 18: The equity curve of buy & hold and SVM-based strategy

Figure 18 presented two equity curves of SVM-based and the buy and hold strategy. Initially, the buy and hold strategy has a better result than the SVM-based strategy, however, at the end of the period, the SVM-based strategy ended up attaining a higher level of accumulated profit. In detail, from the beginning of the period till February 2020, the SVM-based strategy has undergone a slightly higher level of profit than the buy and hold strategy, but the breadth of these equity curves is not too far. Around

February 2020, there existed a quick-rising in the equity curve of the buy and hold strategy. Formally, reached the maximum accumulated return at the level of 25% whereas the accumulated profit of the SVM-based strategy seemed to level off around 7%. Nonetheless, the situation changed quickly during a gigantic drop in IBM's stock price. In March 2020, IBM's stock price dramatically dropped from $ 146 to $ 90 equal to −38% in price depreciation. As a result, both equity curves of SVM-based and the buy and hold dipped around the level of −30.80% and −25.15%. The fact that SVM-based strategy dipped deeper than the buy and hold strategy is unprecedented. In other model-based strategies, the strategies are always better than the buy and hold strategy. It means that the models are doing good in controlling the downside of the equity curve. Additionally, the level of −30.80% of SVM-based strategy is the worst drawdown ever, implying that SVM-based strategy could not play a role of a guard in terms of risk management.

Strategy	Accumulated Profit	Drawdown
SVM-based strategy	6.15%(2.97%)	−30.08%
Buy & Hold strategy	0.60%(0.40%)	−25.15%

Source: Author. Note: CAGR in parenthesis

Table 14: Key summaries of SVM-based and the Buy and hold strategy

However, thanks to the recovery of IBM's stock price from June 2020 towards the end of the period, the equity curve of the SVM-based strategy keep rising while the equity curve of the buy and hold strategy fluctuated. Finally, the SVM-based strategy's equity curve ended up attaining a level of 6.15% in profit equal to a CAGR of 2.97%. This figure is exiguous compared with the other model-based strategies.

Overall, I presented the overview of algorithmic trading derived from machine learning models. The model-based strategy, to some extent, has shown efficacy over the buy and hold strategy. Deep learning models like RNN, LSTM, and GRU demonstrated a better strategy, resulting in a higher level of profit.

Strategy	Accumulated Profit	Drawdown
SVM-based strategy	6.15%(2.97%)	−30.08%
LR-based strategy	8.80%(6.00%)	−18.05%
RNN-based strategy	14.30%(6.00%)	−18.05%
LSTM-based strategy	21.77%(10.50%)	−13.50%
GRU-based strategy	26.71%(18.65%)	−11.58%
Buy & Hold strategy	0.60%(0.40%)	−25.15%

Source: Author. Note: CAGR in parenthesis

Table 15: Key summaries of all trading strategies

Moreover, deep learning models also play a better role in managing the downside of the equity curve. This success could be explained by the wise decision and better understanding of IBM's stock price of the deep learning models. As a result, those models can "guide" the investor to have a wiser decision in going long position. However, both deep learning and classical machine learning models need more room in improving the detection of false information. The rate of false long position remains at the high level. This is indeed a challenging topic in quantitative trading and machine learning. For the sake of completeness, key summaries of all strategies are given in table 15.

Chapter 5: Trustworthiness in AI Solutions in Finance

5.1 What is safe machine leanring and how trustworthy are AI models?

Machine Learning has revolutionized numerous fields, offering transformative solutions across various applications, yet the safety and reliability of these technologies remain a topic of considerable debate. From healthcare, computer vision, autonomous car driving, where algorithms can predict patient outcomes or recommend treatments, to finance, where they assess credit risks and detect fraud, the potential benefits are vast. In transportation, machine learning enhances autonomous vehicle systems, aiming to improve safety and efficiency. However, the inherent opacity of machine learning model often referred to as the ”black box” problem, raises concerns about their safety and decision-making processes. The lack of transparency makes it challenging to fully understand how these models arrive at their conclusions, leading to potential risks such as unintended biases or erroneous outputs. As machine learning solutions become increasingly integral to critical systems, ensuring their reliability and addressing these safety concerns becomes imperative, requiring ongoing research and robust validation processes to build trust and mitigate potential risks. That is the reason why we need a “safe” AI solution.

Safe machine learning is an evolving framework aimed at mitigating the risks associated with the deployment of AI systems, particularly in sensitive fields like finance. Machine learning models, known for their predictive power, also present challenges such as lack of transparency, fairness, and robustness, often referred to as the “black-box” nature of AI. According to Giudici et al (2024), the S.A.F.E. framework encompassing Sustainability, Accuracy, Fairness, and Explainability, provides essential metrics to assess the trustworthiness of AI systems. This is especially crucial in finance, where models can inadvertently introduce biases, creating unfair advantages or risks. The Rank Graduation Box (RGB) metrics proposed by Babaei et al. (2024) measure these aspects through model-agnostic tools, ensuring that AI systems in finance remain robust and fair under various stress scenarios.

However, the inherent opacity of machine learning systems raises significant concerns in the financial sector. A critical issue is the inability to explain decisions made by complex models such as deep learning or random forests. This lack of explainability can lead to substantial risks in high-stakes environments like banking or insurance, where decisions affect financial stability and individual livelihoods. Regulatory bodies like the European Union have responded by introducing guidelines such as the AI Act (2021), which mandates that AI systems be not only accurate but also interpretable and fair (European Commission, 2021)

The financial sector is particularly vulnerable to these risks, as AI-driven decisions often involve sensitive data and have far-reaching consequences. A failure to address fairness, for instance, can lead to discriminatory practices, especially in areas like lending or

credit scoring. Giudici et al. (2023) underscore that AI models in finance must be robust against extreme events, such as market crashes or cyberattacks, and ensure that biases do not disproportionately affect disadvantaged groups. This is where the Rank Graduation Fairness (RGF) metric comes into play, as it measures how well AI systems perform across different demographic groups, thus mitigating unfair outcomes.

Moreover, Giudici and Raffinetti (2024) highlight that explainability is another critical factor in ensuring the safety of machine learning models. For AI systems to be truly trustworthy, they must offer clear reasoning for their decisions, which is especially difficult for black-box models. The Rank Graduation Explainability (RGE) metric helps by offering a quantitative measure of how understandable a model's decisions are, ensuring transparency without sacrificing accuracy.

Finally, the development of safe machine learning systems is vital for the responsible use of AI in finance. With frameworks like S.A.F.E. and tools such as the Rank Graduation Box, it is possible to measure and manage the risks associated with AI, ensuring that systems remain accurate, fair, and transparent. The stakes are particularly high in finance, where model failures can lead to significant financial and ethical consequences, making it imperative that these systems are rigorously evaluated using comprehensive, model-agnostic metrics. In this thesis, Python will be utilized assess the reliability of models applied in trading systems.

5.2 AI models trustworthiness methodology

The methodology for SAFE AI revolves around a framework for evaluating AI systems across four key principles: Sustainability, Accuracy, Fairness, and Explainability. These dimensions are unified under a statistical framework, grounded in the Lorenz curve and its extensions. This common methodological basis ensures consistency across different metrics of AI risk evaluation.

The core theoretical foundation uses the Lorenz curve, traditionally applied in economics to assess income inequality. The authors, Babaei et al (2024)extend this concept to AI evaluation by introducing two related tools: **Dual Lorenz Curve**: Orders the data inversely, providing an alternative view on data distribution, and seconly **Concordance Curve**: A novel tool comparing the ranks of two distributions, used to measure the degree of similarity between predicted and actual outcomes.

These statistical tools are integrated into the Rank Graduation (RG) measure, which is central to the SAFE framework. The RG measure quantifies the distance between the concordance curve and the Lorenz curves, yielding a value that reflects the alignment between predictions and actual outcomes.

The Rank Graduation Box (RGB) incorporates four metrics, each corresponding to a key principle in the SAFE framework: S for Sustainability, A for Accuracy, F for Fairness and E for Explainability. The details of the definitions are illustrated as below:

- Sustainability: Measured by the Rank Graduation Robustness (RGR), which evaluates the AI model's robustness by comparing predictions on normal versus perturbed data (Babaei et al., (2024));
- Accuracy: The Rank Graduation Accuracy (RGA) generalizes the Area Under the Curve (AUC) to continuous and ordinal predictions, capturing the closeness of predictions to actual outcomes (Giudici and Raffinetti, (2023));
- Fairness: The Rank Graduation Fairness (RGF) metric assesses whether the AI model treats different population groups equally by comparing models with and without group-specific variables (Giudici et al., (2024));
- Explainability: The Rank Graduation Explainability (RGE) metric captures the contribution of each input variable by comparing a full model to models excluding specific predictors (Raffinetti, (2023)).

Additionally, to validate the metrics, Babaei et al (2024)propose statistical tests based on U-statistics. These tests are used to determine whether the differences in metrics between AI models (e.g., accuracy, fairness, robustness) are statistically significant. This ensures that the performance and risk assessments made using the RGB framework are reliable. To this end, Python code snippets are used to perform the statistical tests.

5.3 How reliable and trustworthy are our models ?

The results presented in this thesis were generated through Python code, which has been included in the Appendix for reference. The code follows the SAFE methodology discussed in the previous section, ensuring consistency with the key principles outlined earlier. It's worth highlighting that all models in the thesis, ranging from sophisticated deep learning algorithms to more traditional machine learning approaches, are constructed exclusively from the stock's own time series data. No external economic indicators or variables were incorporated into these models. This choice was deliberate, as the aim was to assess the stock's historical performance to forecast future trends and, based on those predictions, make informed trading decisions.

The focus on time series data means that the models rely solely on past stock information to predict future behavior, emphasizing the intrinsic value of the stock's own historical data. By excluding external economic factors, the analysis isolates the stock's inherent patterns and trends. As a result, the models are more self-contained, relying on the stock's own movements to drive predictions. This approach is particularly relevant in finance, where stock-specific analysis can sometimes offer more precise insights than models that include broader, less directly relevant economic data. Although the overall methodology touches on multiple dimensions of AI model evaluation, including fairness, this section of the thesis places particular emphasis on three main aspects: Accuracy, Sustainability (or Robustness), and Explainability. Accuracy is critical in ensuring that the models make reliable predictions based on the stock's past performance.

Sustainability, or robustness, assesses the models' resilience to changes or disturbances in the input data, ensuring that predictions remain stable under varying conditions. Lastly, Explainability is crucial for understanding how the models arrive at their predictions, offering transparency in decision-making processes.

The empirical results of five models: LSTM, GRU, RNN, SVM, and logistic regression (logit)—in terms of accuracy and robustness reveals clear advantages of deep learning over classical machine learning approaches. The deep learning models (LSTM, GRU, and RNN) significantly outperformed classical machine learning models (SVM and Logit) in terms of accuracy. Specifically, LSTM achieved an accuracy of 0.9998, GRU reached 0.9999, and RNN scored 0.9996, showcasing their ability to make highly precise predictions. In contrast, the classical models demonstrated much lower accuracy, with SVM achieving 0.51 and Logit scoring 0.56. This substantial difference can be attributed to the deep learning models' capability to capture complex, non-linear relationships within the stock's time series data, which classical models like SVM and logit struggle to detect. These results are expected and consistent with those of the backtesting analysis mentioned in the earlier chapter, where deep learning models consistently performed better. The superior accuracy of LSTM, GRU, and RNN highlights their effectiveness in understanding the underlying patterns in stock prices, which often involve non-linear dependencies that classical models fail to capture.

Secondly, robustness index once again proves the models' stability when subjected to variations or perturbations in the data, the deep learning models also outperformed classical machine learning methods. The robustness scores for the deep learning models were relatively high, with both LSTM and RNN scoring 0.87, and GRU scoring 0.86. Meanwhile, the classical models showed much lower robustness, with SVM scoring 0.47 and Logit a t0.53. The similarity in patterns between accuracy and robustness suggests that the deep learning models are not only highly accurate but also maintain their performance under less ideal conditions, further reinforcing their superiority. The robustness of deep learning models can be largely attributed to the use of regularization techniques, which play a critical role in preventing overfitting. Regularization methods, such as $l2$ regularization (weight decay), dropout, and early stopping, work by introducing constraints or penalties on the model's complexity. This helps ensure that the model does not become overly reliant on specific patterns in the training data, making it more generalized and capable of performing well on unseen data. As a result, deep learning models are better equipped to maintain stability and accuracy when faced with variations or noise in the input data. By reducing overfitting, regularization enhances the model's ability to learn underlying, more generalizable patterns, making it more robust and reliable in real-world applications.

To validate these findings, statistical tests were conducted to compare the robustness of each model. At a 95% confidence level, the results confirmed that the differences in robustness across the models are statistically significant. The p-values between models further illustrate these differences: LSTM-GRU (2.67e-226), LSTM-RNN (3.8e-198), LSTM-SVM (0.0009), GRU-RNN (0.015), GRU-SVM (0.0346), RNN-SVM (0.015), and SVM-Logit (2.4e-05). These values indicate that, statistically, the robustness of the

deep learning models is significantly different from that of classical machine learning models, with deep learning showing much higher robustness scores.

In the context of deploying AI models, particularly those characterized by their complexity and opaque nature, often referred to as black-box models, the explainability becomes a fundamental concern. The ability to understand and interpret how these models make decisions is not merely a technical necessity but a crucial factor in gaining and maintaining trust from user, traders and stakeholders. This transparency is essential for fostering confidence in AI systems and encouraging their continued development and effective management.

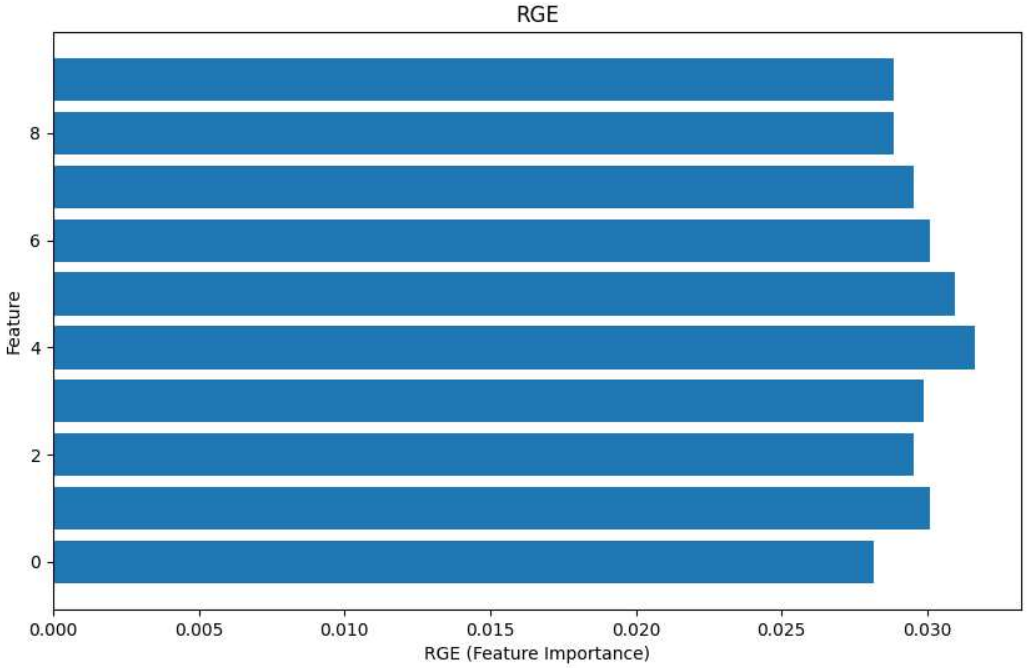

Figure 19: Features Explanation of LSTM model

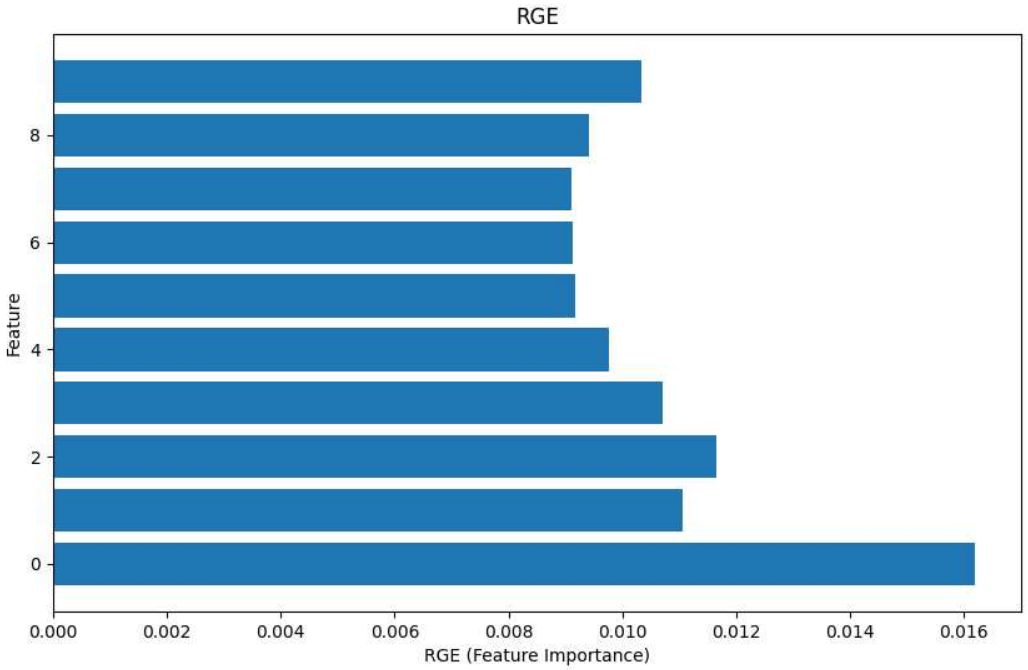

Figure 20: Features Explanation of RNN model

Moreover, methodologies designed to ensure the safe and responsible use of AI play a significant role in demystifying the contributions of various input variables within a model. This is applicable to both regression and classification models. SAFE methodology practices emphasize the importance of understanding each variable's impact on the model's predictions, which in turn helps in improving the model's reliability and accountability.

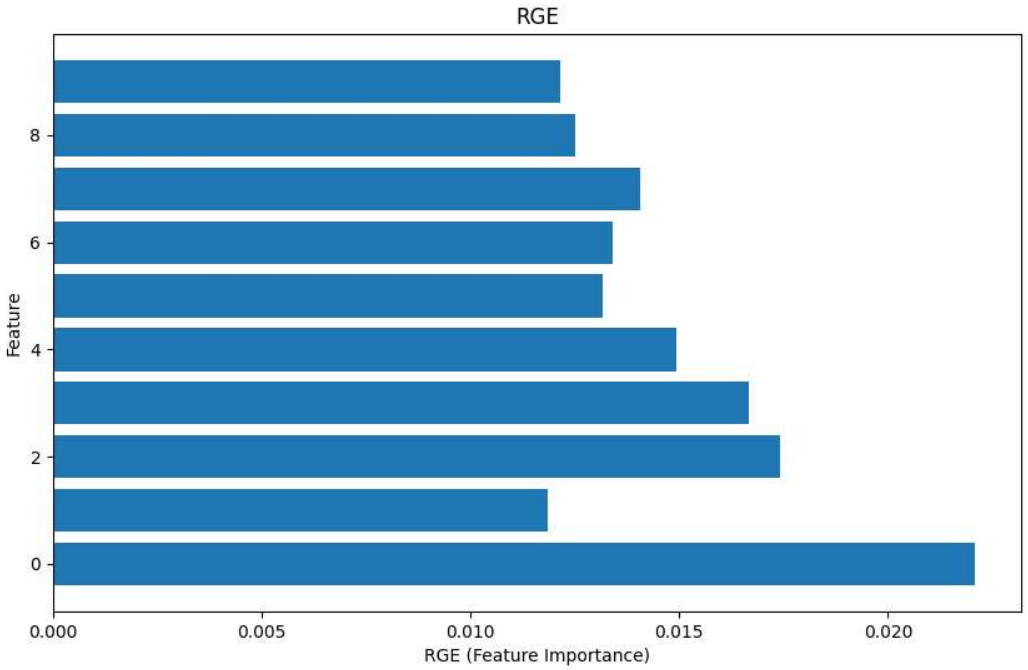

Figure 21: Features Explanation of GRU model

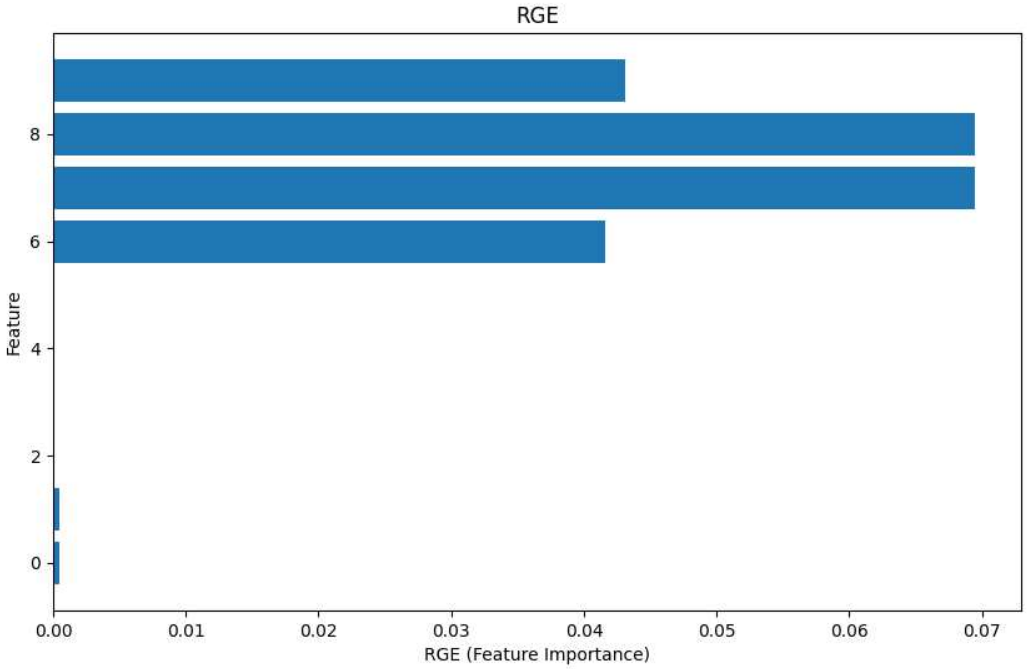

Figure 22: Features Explanation of SVM model

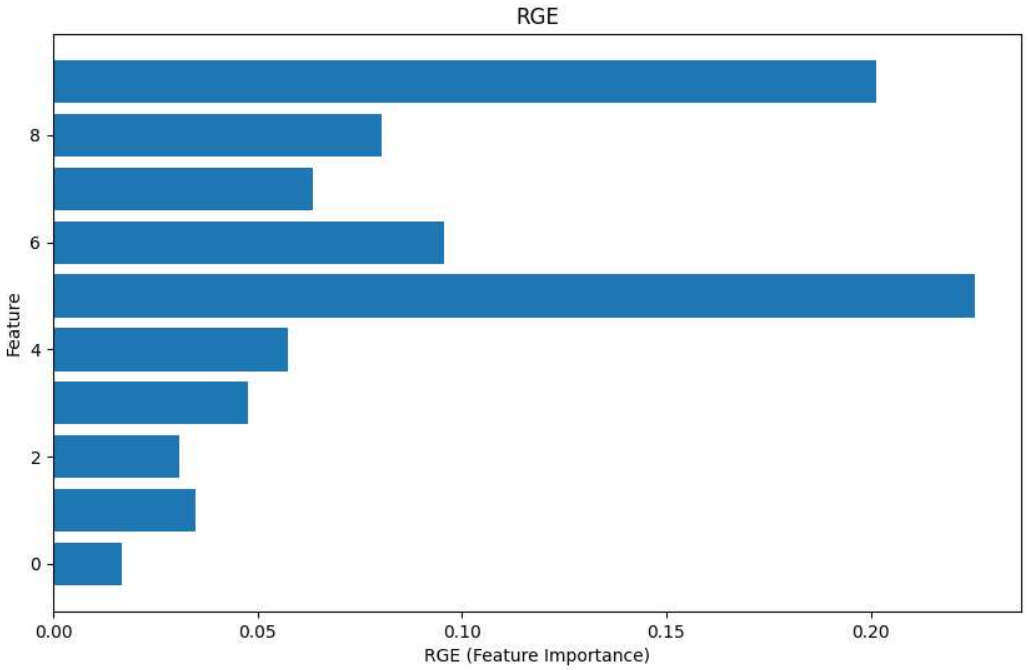

Figure 23: Features Explanation of Logistic model

The significance of these practices is highlighted through demonstrative examples, such as Figures 19, 20, 21, 22, and 23 which depict the behavior of advanced deep learning models. These figures reveal that such models equally prioritize both historical and the most recent price data when making predictions. This balanced approach is particularly valuable in trading environments, where current price information can provide more actionable insights compared to historical data alone. There is no doubt to say that the most recent information is always better in making trading decision

In contrast, classical machine learning models tend to focus more heavily on past data. This methodological difference can account for the observed superiority in accuracy and robustness of advanced deep learning models over traditional machine learning approaches. The advanced models' ability to integrate and weigh both past and present information allows them to adapt more effectively to changing conditions, thereby enhancing their predictive performance and reliability in dynamic environments.

Finally, understanding these differences helps to explain why advanced deep learning techniques often outperform classical models in various applications, especially those involving real-time data and complex decision-making scenarios. By appreciating the role of explainability and the nuances of different AI methodologies, stakeholders can make more informed decisions about the deployment and further development of AI systems.

Chapter 6: Conclusions

6.1 General Conclusions

This thesis analyzed the consequences of quantitative trading based on machine learning algorithms of IBM's stock price from January 2000 to March 2021. In detail, the model is trained from Jan 2000 to February 2019 and evaluated during the rest of the period. The algorithms are mainly classified into two types: classical machine learning and deep learning models. The classical machine learning models include Logistic Regression and Support Vector Machine. Deep learning models contain Recurrent Neural Network, Long Short-Term Memory, and Gated Unit Recurrent. Each model follows strictly the complete 8-step procedure mentioned in the method section, the model-based strategy is validated independently during the testing period, which is absolutely independent of the training period.

On the one hand, the output of the classical machine learning model is indeed the trading signal $(1, -1)$ corresponding to a long and short position. For the sake of fairness, "short-selling" is restricted in this framework, hence, I only make use of the signal of 1. On the other hand, the output of deep learning models is in numerical form, meaning that the model is predicting the one-step-ahead price of IBM's stock. Consequently, I need one more step to convert the output into a trading strategy. The interpretation of trading strategy is quite straightforward and it only relies on the predicted price. If the forecasted price is larger than the actual current price, the trading signal is assigned as a buy position, otherwise, I would do nothing as "short-selling" is prohibited. The model-based strategy, was then verified with the daily vectorized backtest. The daily vectorized backtest is responsible for computing the long-position return of the strategy no matter what it is positive or negative.

The strategy is evaluated through four pillars, namely the efficacy of each strategy by the accumulated profit, the risk management by the drawdown, the hit rate, and the false position rate. When it comes to efficacy and risk management perspective, I presented the equity curve of each strategy. The equity curve elaborates on the accumulated profit of each strategy on the daily basis during the testing period. It also pointed out the maximum drop in the profit curve, which is indeed the drawdown. The drawdown plays a crucial role in risk management, it shows that the largest tolerance of the strategy concerning the movement of the given underlying. On the one side, the accumulated profit shows the effectiveness of the strategy in going long positions. On the other side, the drawdown verifies the effectiveness of not going long position. In other words, from the investor's perspective, the model needs to be wise enough to place a buy position and not to place anything in a particular situation. These also are reflected in the hit rate and the rate of false position.

Overall, the results have demonstrated that the deep learning models have a better performance in generating trading strategies and managing risk than classical machine learning models. In terms of the effectiveness of the strategy, the GRU-based strategy attained an accumulated profit of 26.71% from February 2019 to March 2021, equal to a

CAGR of 18.65%. It is the most outstanding performance among strategies. The other two deep learning-based strategies, namely LSTM-based and RNN-based strategies, also have good results. LSTM-based and RNN-based strategies achieved 21.77% equal to a CAGR of 10.50% and 14.30% equal to a CAGR of 6.00%, whereas the default buy and hold strategy only gained 0.6% equal to a CAGR of 0.4% during the testing period of two years. Additionally, two classical machine learning model-based strategies, namely LR-based and SVM-based strategies ended up earning accumulated profits of 8.80% equal to a CAGR of 6.00% and 6.15% equal to a CAGR of 2.97%, which are better than the default buy and hold strategy but by far moderate than the deep learning model-based strategies. In terms of risk management, the deep learning model-based strategies are always their cutting-edge advantage over the default buy and hold and classical machine learning model-based strategies in whether to place a buy position or not. In detail, while the drawdown of GRU, LSTM, and RNN-based strategy only dipped around a level of $-11.58\%, -13.50\%$, and -18.5% respectively, these figures of two machine learning-based strategies, says LR and SVM-based strategy, and the buy and hold strategy are $-18.05\%, -30.08\%$, and -25.15% respectively. In the worst case, the SVM-based model could lose a maximum of 30.08% of its accumulated profit during the testing period which is fairly larger than the default strategy, whereas the GRU-based strategy could only lose a maximum of 11.58% of its accumulated profit. Besides, the deep learning model-based strategy also conquered a leading position in the hit rate. The hit rate of RNN, LSTM, and GRU-based strategy reached $70.00\%, 68.5\%$, and 96.5% respectively, whereas these figures of LR and SVM-baed strategy only remained around 62.5% and 64.5%. This could partly imply that the deep learning model-based strategies are wise enough to go long positions in the right situation. However, the results have also pointed out that the rate of the false position of deep learning model-based strategies remained higher than their counterpart. In detail, RNN, LSTM, and GRU-based strategy reached $73.6\%, 70.7\%$, and 94.2% respectively, whereas the rate of false position of LR and SVM-based strategy stayed only around 44.4%.

In conclusion, following the contributions such as those by, the results have provided a further empirical example of how to a machine learning model-based strategy in quantitative trading. The results have also shown that while there is still room for improvement, deep learning models paved the way for a fruitful approach in terms of profit and risk management in quantitative investment compared with other model-based strategy or the default buy and hold strategy.

6.2 Research Implications

The research into trading using quantitative techniques, particularly deep learning, has significant implications for both the financial industry and academic fields. By harnessing the power of deep learning algorithms, this research can lead to the development of highly precise models, therefore capable of identifying non-linear, complex patterns and trends in vast amounts of financial data that traditional methods might be overlooked. This could revolutionize trading strategies, enabling investors and institutions to make more informed decisions, minimize risks, and maximize returns in

increasingly volatile markets. Moreover, the successful application of these techniques could spur further innovation in algorithmic trading, leading to the automation of complex trading strategies that adapt in real time to market changes. From an academic point of view, this research contributes to the development of quantitative-based model in Finance, providing a concrete foundation for future studies that explore the intersection of finance, data science, and artificial intelligence. Additionally, according to Heaton et al (2017) and Fischer and Krauss (2018), the findings could have broader implications for regulatory practices, as they highlight the need for updated frameworks that address the challenges and opportunities presented by AI-driven trading systems

6.3 Further research recommendations and suggestions

While no trading strategy is entirely without flaws, the systematic and logical framework provided by quantitative methodologies is crucial in managing the increasing complexity of modern financial markets. This approach offers traders, users a clean, clear and scientific way to analyse and make decision, which will significantly remove the nature of irrationale or senseless of human being. These methodologies guide traders through the often chaotic and unpredictable nature of trading, offering a disciplined approach that helps mitigate the emotional and cognitive biases that can cloud judgment. As the financial landscape continues to evolve, becoming more data-driven and influenced by a multitude of factors, further research should focus on expanding the integration of alternative data sources. This includes not just traditional financial indicators, but also non-conventional data like social media sentiment, real-time news, and even environmental factors, which can provide a more holistic view of market dynamics.

Moreover, the combination of deep learning with reinforcement learning presents a promising frontier for research, where models can be developed not only to forecast market trends but also to adaptively optimize trading decisions in real time. This hybrid approach could lead to the creation of more autonomous trading systems that are capable of learning and improving over time, thereby increasing their effectiveness in various market conditions. Additionally, the reseach also focuss on the ability of interpretability of the model, and how accurate and reasonable the interpretability is. As deep learning models become more complex, understanding how and why a model arrives at certain decisions becomes essential, both for gaining the trust of users and for meeting regulatory standards that require transparency in automated trading systems.

In addition, research should continue to explore ways to enhance the robustness and generalization capabilities of these models, ensuring that they perform reliably across a range of market scenarios, including those that are highly volatile or exhibit unusual patterns. This is critical for ensuring that models are not just effective in specific conditions but are adaptable to the broad spectrum of real-world market behaviors. Finally, there is a growing need for more efficient training algorithms and model architectures that can reduce the computational resources and time required to develop and deploy these sophisticated models. As financial markets become faster and more complex, the ability to quickly iterate and update models in response to new data is

becoming increasingly important. To this end, researchers can help traders stay ahead in the ever-changing landscape of global finance, providing them with the tools they need to make informed, systematic decisions in a logical and consistent manner.

An important yet often overlooked area is the application of AI in risk management. A good profitable trading strategy alway company with exellent risk management. Future research could focus on developing AI-driven methodologies that systematically identify, assess, and mitigate risks within trading strategies, thereby enhancing their overall stability and resilience. By using AI to automate and optimize risk management processes, traders could create more robust strategies that are better equipped to handle market volatility and unexpected events.

APPENDIX

Mathematical Proofs

I would like to prove equation (6) in section $2.1.3$.
Let $\mathcal{A}$ is an algorithm and m-tupe training set $\mathcal{S} = \{z_i = (x_i, y_i)\}\ \forall i = 1, \ldots, m$, and $\mathcal{A}(\mathcal{S})$ denotes the outcome of an algorithm $\mathcal{A}$ with the instances in the training set $\mathcal{S}$ as input. Given a training set $\mathcal{S}$ and an additional observation z^*, $\mathcal{S}_i$ denotes the training set with a replacement z^* for z_i. Therefore, the stability of a model will be measured by comparing the loss value of the outcome $\mathcal{A}(\mathcal{S})$ on z_i to the loss value of $\mathcal{A}(\mathcal{S}_i)$ on z_i. For sake of clarity, the stability is measured by: $\ell(\mathcal{A}(\mathcal{S}_i, z_i))$-$\ell(\mathcal{A}(\mathcal{S}), z_i)$. Prove that:

$$\mathbb{E}(\mathcal{L}_{\mathcal{D}}(\mathcal{A}(\mathcal{S}))\text{-}\mathcal{L}_{\mathcal{S}}(\mathcal{A}(\mathcal{S}))) \quad = \quad \mathbb{E}(\ell(\mathcal{A}(\mathcal{S}_i, z_i)\text{-}\ell(\mathcal{A}(\mathcal{S})), z_i)) \tag{35}$$

Proof: Because, z_i, z^* are independent, so, from the definition of the true loss of an algorithm, I have:

$$\mathcal{L}_{\mathcal{D}}(\mathcal{A}(\mathcal{S})) = \mathbb{E}(\ell(\mathcal{A}(\mathcal{S})), z_i))) = \mathbb{E}(\ell(\mathcal{A}(\mathcal{S}_i, z_i)) \tag{36}$$

Hence,

$$\mathbb{E}(\mathcal{L}_{\mathcal{D}}(\mathcal{A}(\mathcal{S}))) = \mathbb{E}(\ell(\mathcal{A}(\mathcal{S}_i, z_i)) \tag{37}$$

Besides, from the definition of the training loss, I have:

$$\mathcal{L}_S(\theta) \quad = \quad \frac{1}{m}\sum_{i=1}^{m} \ell(\theta, x_i, y_i) \tag{38}$$

So, it is easy to verify that:

$$\mathbb{E}(\mathcal{L}_S(\theta)) = \mathbb{E}(\frac{1}{m}\sum_{i=1}^{m} \ell(\theta, x_i, y_i)) = \mathbb{E}(\ell(\mathcal{A}(\mathcal{S})), z_i))) \tag{39}$$

Equation (37) and (39) conclude the proof.

Manipulation and Computation Code Snipets in Python

Code: Load Historical Data Price of Stock IBM from Yahoo Finance Data Source

```
import pandas as pd
import pandas_datareader as web
from TimeSeries_Preprocess import split_train_val_scale
from Supervised_TimeSeries_Modern import create_X_y

stock = 'IBM'
ibm_df = web.DataReader(stock, start = '2000-01-01', end = '2021-03-01',
    \ data_source = 'yahoo')
ibm_df = ibm_df['Adj Close']
ibm_df = pd.DataFrame(ibm_df)

type_of_input = 'gen'
type_of_output = 'modern'
lag = 10
ahead = 1
ratio = 0.9

input_train_scaled, input_val_scaled, \
input_min_val, input_max_val, input_val \
    = split_train_val_scale(ibm_df, ratio = ratio)


output_train_scaled, output_val_scaled, \
output_min_val, output_max_val, output_val\
    = split_train_val_scale(ibm_df, ratio = ratio)

X_train, y_train = create_X_y(
    input_train_scaled,
    output_train_scaled,
    lag = lag,
    ahead = ahead,
    type_of_input = type_of_input,
    type_of_output = type_of_output
)

X_val, y_val = create_X_y(
    input_val_scaled,
    output_val_scaled,
    lag = lag,
    ahead = ahead,
    type_of_input = type_of_input,
    type_of_output = type_of_output
)
```

Code Exam: Split data into training and testing time series dataset

```
import numpy as np

def split_train_val_scale(df_train_val, ratio = 0.8, scale = 'normal'):
    idx = int(df_train_val.shape[0] * ratio)
    df_train = df_train_val.iloc[: idx, :]
    df_val = df_train_val.iloc[idx :, :]

    if scale == 'minmax':
        min_train = np.min(df_train)
        max_train = np.max(df_train)
        df_train_scaled = (df_train - min_train)/(max_train - min_train)

        if df_val.shape[0] > 1:
            min_val = np.min(df_val)
            max_val = np.max(df_val)
            df_val_scaled = (df_val - min_val)/(max_val - min_val)
        else:
            min_val = min_train
            max_val = max_train
            df_val_scaled = (df_val - min_val) / (max_val - min_val)
        return df_train_scaled, df_val_scaled, min_val.values,
 max_val.values, df_val
    else:
        mu_train = np.mean(df_train)
        sigma_train = np.std(df_train)
        df_train_scaled = (df_train-mu_train)/sigma_train
        mu_val = mu_train
        sigma_val = sigma_train
        df_val_scaled = (df_val - mu_val)/sigma_val
        return df_train_scaled, df_val_scaled, mu_val.values,
    sigma_val.values, df_val

def scale_df(df):
    if df.shape[0] > 1:
        min_value = np.min(df)
        max_value = np.max(df)
        df_scaled = (df - min_value)/(max_value - min_value)

        return min_value, max_value, df_scaled

def reversed_df(df_scaled, min_value, max_value):
    df = df_scaled*(max_value - min_value) + min_value
    return df
```

Code: Split data into training and testing time series dataset

```
import numpy as np

def SupervisedTimeSeries(df, lag, ahead):
    output_ = []
    if ahead > 0:

        for i in range(df.shape[0]):
            if i*ahead + lag > df.shape[0]:
                break
            else:
                output_.append(df[(i * ahead): (i * ahead + lag)].values)
    else:
        for i in range(df.shape[0]):
            if i + lag > df.shape[0]:
                break
            else:
                output_.append(df[i: (i+lag)].values)
    return np.array(output_)

def create_X_y(
        df_input,
        df_output,
        lag,
        ahead,
        type_of_input = 'flex',
        type_of_output = 'classic',
        custom_lag_of_input = [1, 2, 3, 4, 5]

):
    assert df_input.shape[0] == df_output.shape[0]
    input_of_df = SupervisedTimeSeries(df_input, lag, ahead)

    if type_of_input == 'flex':
        assert  df_input.shape[1] == len(custom_lag_of_input)
        assert max(custom_lag_of_input) <= lag

        for i in range(input_of_df.shape[0]):
            for j in range(len(custom_lag_of_input)):
                input_of_df[i, :(input_of_df.shape[1]-custom_lag_of_input[j]), j]
    = 0

    if type_of_output == 'classic':
        output_of_df = []
        if ahead > 0:
            for i in range(df_output.shape[0]):
                if (i + 1)*ahead > df_output[lag:].shape[0]:
                    break
                else:
                    output_of_df.append(df_output[lag:][i*ahead: (i + 1)*ahead])
        else:
            for i in range(df_output.shape[0]):
                if lag + i > df_output.shape[0]:
                    break
```

```
            else:
                output_of_df.append(df_output[(lag+i-1): (lag+i)])

    output_of_df = np.array(output_of_df)
    mutual = min(input_of_df.shape[0], output_of_df.shape[0])
    input_of_df = input_of_df[: mutual, :, :]
    output_of_df = output_of_df[: mutual, :, :]
else:
    k = 0
    for _ in range(input_of_df.shape[0]):
        if ( _*ahead + input_of_df.shape[1]) > (df_input.shape[0] - ahead):
            break
        else:
            k += 1

    if ahead > 0:
        output_of_df = np.ones(shape = (k, input_of_df.shape[1], ahead))

        for i in range(output_of_df.shape[0]):
            for j in range(1, ahead + 1):
                output_of_df[i, :, (j - 1): j] =
  df_output[(ahead*i + j): (ahead *i+j+input_of_df.shape[1])].values
    else:
        output_of_df = np.ones(shape = (k, input_of_df.shape[1], 1))
        for i in range(output_of_df.shape[0]):
            output_of_df[i, :, 0: 1] = df_output[i: (input_of_df.shape[1] + i)]

    mutual = min(input_of_df.shape[0], output_of_df.shape[0])
    input_of_df = input_of_df[: mutual, :, :]
    output_of_df = output_of_df[: mutual, :, :]

return input_of_df, output_of_df
```

Code: GRU Deep Learning Model

```
import tensorflow as tf
from tensorflow import keras
from keras.callbacks import EarlyStopping
from keras.regularizers import l2, l1
from thesis_deeplearning_train_val import X_train, y_train, X_val, y_val, \
    lag, ahead, type_of_output,\
    input_min_val, input_max_val, output_min_val, output_max_val, output_val, \
    stock
import pandas as pd


tf.random.set_seed(5)
max_epoch = 2000
batch_size = 32

if type_of_output == 'modern':
    def last_step_mse(Y_true, Y_pred):
        return keras.metrics.mean_squared_error(Y_true[:, -1], Y_pred[:, -1])

    def rnn(n_units = 32, reg = 0.01, learning_rate = 0.01, seed = 5):
        model = keras.models.Sequential(
            [
                keras.layers.GRU(
                    n_units,
                    kernel_initializer = keras.initializers.glorot_uniform(seed),
                    bias_initializer = keras.initializers.glorot_uniform(seed),
                    recurrent_initializer =\
             keras.initializers.orthogonal(seed = seed),
                    input_shape = (X_train.shape[1], X_train.shape[-1]),
                    kernel_regularizer = l2(reg),
                    return_sequences = True,
                ),
                keras.layers.GRU(
                    n_units,
                    kernel_initializer = keras.initializers.glorot_uniform(seed),
                    bias_initializer = keras.initializers.glorot_uniform(seed),
                    recurrent_initializer = keras.initializers.orthogonal(seed=seed)
                    kernel_regularizer = l2(reg),
                    return_sequences = True,
                ),
                keras.layers.GRU(
                    n_units,
                    kernel_initializer=keras.initializers.glorot_uniform(seed),
                    bias_initializer=keras.initializers.glorot_uniform(seed),
                    recurrent_initializer=keras.initializers.orthogonal(seed=seed),
                    kernel_regularizer=l2(reg),
                    return_sequences=True,
                ),

                keras.layers.TimeDistributed(
                    keras.layers.Dense(
                        ahead,
                        kernel_initializer = keras.initializers.glorot_uniform(seed)
```

```
                    bias_initializer = keras.initializers.glorot_uniform(seed),
                    kernel_regularizer = keras.regularizers.l2(reg)
                )
            )
        ]
    )

    model.compile(
        optimizer = keras.optimizers.RMSprop(learning_rate = learning_rate,
    \ rho = 0.75),
        loss = 'mean_squared_error',
        metrics = [last_step_mse]
    )
    return model

early_stopping = EarlyStopping(
    monitor = 'val_last_step_mse',
    mode = 'min',
    verbose = True,
    patience = 100,
    min_delta = 1e-5,
    restore_best_weights = True
)

"""TRAINING PROCESS"""
retrain_after_crossval = True

if retrain_after_crossval:
    model = rnn(
        n_units = 8,
        reg = 1e-06,
        learning_rate = 1e-03,
        seed = 5
    )

    model.fit(
        X_train,
        y_train,
        epochs = max_epoch,
        batch_size = batch_size,
        validation_data = (X_val, y_val),
        callbacks = [early_stopping]
    )

    print(min(model.history.history['val_last_step_mse']))

    pd.DataFrame(
        {
            'origin_reference': output_val[(lag + ahead - 1):].values.flatten(),
            'actual': y_val[:, -1].flatten()*(output_max_val-input_min_val)
+ input_min_val,
            'pred': model.predict(X_val)[:, -1].flatten()
* (input_max_val - input_min_val)
+ input_min_val,

        },
```

```
            index = output_val[(lag + ahead -1): ].index

        ).to_csv(f'thesis_vis/{stock}_type_{type_of_output}_test_GRU.csv')
else:
    def rnn(n_units = 32, reg = 0.01, learning_rate = 0.01, seed = 5):
        model = keras.models.Sequential(
            [
                keras.layers.SimpleRNN(
                    n_units,
                    kernel_initializer = keras.initializers.glorot_uniform(seed),
                    bias_initializer = keras.initializers.glorot_uniform(seed),
                    recurrent_initializer = keras.initializers.orthogonal(seed=seed),
                    input_shape = (X_train.shape[1], X_train.shape[-1]),
                    kernel_regularizer = l2(reg),
                    return_sequences = False
                ),
                keras.layers.Dense(
                    ahead,
                    kernel_initializer = keras.initializers.glorot_uniform(seed),
                    bias_initializer = keras.initializers.glorot_uniform(seed),
                    kernel_regularizer = keras.regularizers.l2(reg)
                )
            ]
        )
        model.compile(
            optimizer = keras.optimizers.RMSprop(learning_rate = learning_rate,
    rho = 0.9),
            loss = 'mean_squared_error'
        )

        return model

    early_stopping = EarlyStopping(
        monitor = 'val_loss',
        mode = 'min',
        verbose = True,
        patience = 100,
        min_delta = 1e-5,
        restore_best_weights = True
    )

    retrain_after_crossval = True

    if retrain_after_crossval:
        model = rnn(
            n_units = 8,
            reg = 1e-06,
            learning_rate = 1e-04,
            seed = 5
        )
        model.fit(
            X_train,
            y_train,
            epochs = max_epoch,
            batch_size = batch_size,
            validation_data = (X_val, y_val),
```

```
        callbacks = [early_stopping]
    )

    print(min(model.history.history['val_loss']))
    pd.DataFrame(

        {
            'origin_reference': output_val[(lag + ahead - 1):].values.flatten(),
            'actual': y_val[:, -1].flatten()*(output_max_val-input_min_val)
+ input_min_val,
            'pred': model.predict(X_val)[:, -1].flatten()
* (input_max_val - input_min_val)
+ input_min_val,

        },
        index = output_val[(lag + ahead -1): ].index

    ).to_csv(f'thesis_vis/{stock}_type_{type_of_output}_test_GRU.csv')
```

Code: LSTM Deep Learning Model

```
import tensorflow as tf
import pandas as pd
from tensorflow import keras
from keras.callbacks import EarlyStopping
from keras.regularizers import l2, l1
from thesis_deeplearning_train_val import X_train, y_train, X_val, y_val, \
    lag, ahead, type_of_output,\
    input_min_val, input_max_val, output_min_val, output_max_val, output_val, \
    stock

tf.random.set_seed(5)
max_epoch = 2000
batch_size = 32

if type_of_output == 'modern':
    def last_step_mse(Y_true, Y_pred):
        return keras.metrics.mean_squared_error(Y_true[:, -1], Y_pred[:, -1])

    def rnn(n_units = 32, reg = 0.01, learning_rate = 0.01, seed = 5):
        model = keras.models.Sequential(
            [
                keras.layers.LSTM(
                    n_units,
                    kernel_initializer = keras.initializers.glorot_uniform(seed),
                    bias_initializer = keras.initializers.glorot_uniform(seed),
                    recurrent_initializer = \
      keras.initializers.orthogonal(seed=seed),
                    input_shape = (X_train.shape[1], X_train.shape[-1]),
                    kernel_regularizer = l2(reg),
                    return_sequences = True,

                ),
                keras.layers.LSTM(
                    n_units,
                    kernel_initializer = keras.initializers.glorot_uniform(seed),
                    bias_initializer = keras.initializers.glorot_uniform(seed),
                    recurrent_initializer = keras.initializers.orthogonal(seed=seed)
                    kernel_regularizer = l2(reg),
                    return_sequences = True,
                ),
                keras.layers.LSTM(
                    n_units,
                    kernel_initializer=keras.initializers.glorot_uniform(seed),
                    bias_initializer=keras.initializers.glorot_uniform(seed),
                    recurrent_initializer=keras.initializers.orthogonal(seed=seed),
                    kernel_regularizer=l2(reg),
                    return_sequences=True,
                ),

                keras.layers.TimeDistributed(
                    keras.layers.Dense(
                        ahead,
                        kernel_initializer = keras.initializers.glorot_uniform(seed)
```

```
                        bias_initializer = keras.initializers.glorot_uniform(seed),
                        kernel_regularizer = keras.regularizers.l2(reg)
                    )
                )
            ]
        )
        model.compile(
            optimizer = keras.optimizers.RMSprop(learning_rate = learning_rate,
       rho = 0.75),
            loss = 'mean_squared_error',
            metrics = [last_step_mse]
        )
        return model

    early_stopping = EarlyStopping(
        monitor = 'val_last_step_mse',
        mode = 'min',
        verbose = True,
        patience = 100,
        min_delta = 1e-5,
        restore_best_weights = True
    )

    """TRANINING PROCESS"""
    retrain_after_crossval = True

    if retrain_after_crossval:
        model = rnn(
            n_units = 8,
            reg = 1e-06,
            learning_rate = 1e-03,
            seed = 5
        )

        model.fit(
            X_train,
            y_train,
            epochs = max_epoch,
            batch_size = batch_size,
            validation_data = (X_val, y_val),
            callbacks = [early_stopping]
        )
        print(min(model.history.history['val_last_step_mse']))
        pd.DataFrame(
            {
                'origin_reference': output_val[(lag + ahead - 1):].values.flatten(),
                'actual': y_val[:, -1].flatten()*(output_max_val-input_min_val) +
        \ input_min_val,
                'pred': model.predict(X_val)[:, -1].flatten() *
        \ (input_max_val - input_min_val)+input_min_val,
            },
            index = output_val[(lag + ahead -1): ].index
        ).to_csv(f'thesis_vis/{stock}_type_{type_of_output}_test_LSTM.csv')
else:
    def rnn(n_units = 32, reg = 0.01, learning_rate = 0.01, seed = 5):
        model = keras.models.Sequential(
```

```
        [
            keras.layers.SimpleRNN(
                n_units,
                kernel_initializer = keras.initializers.glorot_uniform(seed),
                bias_initializer = keras.initializers.glorot_uniform(seed),
                recurrent_initializer = keras.initializers.orthogonal(seed=seed),
                input_shape = (X_train.shape[1], X_train.shape[-1]),
                kernel_regularizer = l2(reg),
                return_sequences = False
            ),
            keras.layers.Dense(
                ahead,
                kernel_initializer = keras.initializers.glorot_uniform(seed),
                bias_initializer = keras.initializers.glorot_uniform(seed),
                kernel_regularizer = keras.regularizers.l2(reg)
            )
        ]
    )

    model.compile(
        optimizer = keras.optimizers.RMSprop(learning_rate = learning_rate,
  \  rho = 0.9),
        loss = 'mean_squared_error'
    )

    return model

early_stopping = EarlyStopping(
    monitor = 'val_loss',
    mode = 'min',
    verbose = True,
    patience = 100,
    min_delta = 1e-5,
    restore_best_weights = True
)

retrain_after_crossval = True

if retrain_after_crossval:
    model = rnn(
        n_units = 8,
        reg = 1e-06,
        learning_rate = 1e-04,
        seed = 5
    )
    model.fit(
        X_train,
        y_train,
        epochs = max_epoch,
        batch_size = batch_size,
        validation_data = (X_val, y_val),
        callbacks = [early_stopping]
    )
    print(min(model.history.history['val_loss']))

    pd.DataFrame(
```

```
    {
        'origin_reference': output_val[(lag + ahead - 1):].values.flatten(),
        'actual': y_val[:, -1].flatten()*(output_max_val-input_min_val)
\ + input_min_val,
        'pred': model.predict(X_val)[:, -1].flatten() *
\ (input_max_val - input_min_val)
+ input_min_val
    },
    index = output_val[(lag + ahead -1): ].index
).to_csv(f'thesis_vis/{stock}_type_{type_of_output}_test_LSTM.csv')
```

Code: RNN Deep Learning Model

```
import tensorflow as tf
import pandas as pd
from tensorflow import keras
from keras.callbacks import EarlyStopping
from keras.regularizers import l2, l1
from thesis_deeplearning_train_val import X_train, y_train, X_val, y_val, \
    lag, ahead, type_of_output,\
    input_min_val, input_max_val, output_min_val, output_max_val, output_val, \
    stock
tf.random.set_seed(5)
max_epoch = 2000
batch_size = 32

if type_of_output == 'modern':
    def last_step_mse(Y_true, Y_pred):
        return keras.metrics.mean_squared_error(Y_true[:, -1], Y_pred[:, -1])

    def rnn(n_units = 32, reg = 0.01, learning_rate = 0.01, seed = 5):
        model = keras.models.Sequential(
            [
                keras.layers.SimpleRNN(
                    n_units,
                    kernel_initializer = keras.initializers.glorot_uniform(seed),
                    bias_initializer = keras.initializers.glorot_uniform(seed),
                    recurrent_initializer = keras.initializers.orthogonal(seed=seed)
                    input_shape = (X_train.shape[1], X_train.shape[-1]),
                    kernel_regularizer = l2(reg),
                    return_sequences = True,
                ),
                keras.layers.SimpleRNN(
                    n_units,
                    kernel_initializer = keras.initializers.glorot_uniform(seed),
                    bias_initializer = keras.initializers.glorot_uniform(seed),
                    recurrent_initializer = keras.initializers.orthogonal(seed=seed)
                    kernel_regularizer = l2(reg),
                    return_sequences = True,
                ),
                keras.layers.SimpleRNN(
                    n_units,
                    kernel_initializer=keras.initializers.glorot_uniform(seed),
                    bias_initializer=keras.initializers.glorot_uniform(seed),
                    recurrent_initializer=keras.initializers.orthogonal(seed=seed),
                    kernel_regularizer=l2(reg),
                    return_sequences=True,
                ),

                keras.layers.TimeDistributed(
                    keras.layers.Dense(
                        ahead,
                        kernel_initializer = keras.initializers.glorot_uniform(seed)
                        bias_initializer = keras.initializers.glorot_uniform(seed),
                        kernel_regularizer = keras.regularizers.l2(reg)
                    )
```

```
                )
            ]
        )
        model.compile(
            optimizer = keras.optimizers.RMSprop(learning_rate=learning_rate,
                \ rho = 0.75),
            loss = 'mean_squared_error',
            metrics = [last_step_mse]
        )
        return model

    early_stopping = EarlyStopping(
        monitor = 'val_last_step_mse',
        mode = 'min',
        verbose = True,
        patience = 100,
        min_delta = 1e-5,
        restore_best_weights = True
    )

    """TRAINING PROCESS"""
    retrain_after_crossval = True

    if retrain_after_crossval:
        model = rnn(
            n_units = 8,
            reg = 1e-06,
            learning_rate = 1e-03,
            seed = 5
        )

        model.fit(
            X_train,
            y_train,
            epochs = max_epoch,
            batch_size = batch_size,
            validation_data = (X_val, y_val),
            callbacks = [early_stopping]
        )

        print(min(model.history.history['val_last_step_mse']))

        pd.DataFrame(
            {
                'origin_reference': output_val[(lag + ahead - 1):].values.flatten(),
                'actual': y_val[:, -1].flatten()*(output_max_val-input_min_val)
            \ + input_min_val,
                'pred': model.predict(X_val)[:, -1].flatten()
            \ *  (input_max_val - input_min_val)
             \ + input_min_val,
            },
            index = output_val[(lag + ahead -1): ].index
        ).to_csv(f'thesis_vis/{stock}_type_{type_of_output}_test.csv')
else:

    def rnn(n_units = 32, reg = 0.01, learning_rate = 0.01, seed = 5):
```

```
    model = keras.models.Sequential(
        [
            keras.layers.SimpleRNN(
                n_units,
                kernel_initializer = keras.initializers.glorot_uniform(seed),
                bias_initializer = keras.initializers.glorot_uniform(seed),
                recurrent_initializer = keras.initializers.orthogonal(seed=seed)
                input_shape = (X_train.shape[1], X_train.shape[-1]),
                kernel_regularizer = l2(reg),
                return_sequences = False
            ),
            keras.layers.Dense(
                ahead,
                kernel_initializer = keras.initializers.glorot_uniform(seed),
                bias_initializer = keras.initializers.glorot_uniform(seed),
                kernel_regularizer = keras.regularizers.l2(reg)
            )
        ]
    )

    model.compile(
        optimizer = keras.optimizers.RMSprop(learning_rate = learning_rate
    \ , rho = 0.9),
        loss = 'mean_squared_error'
    )

    return model

early_stopping = EarlyStopping(
    monitor = 'val_loss',
    mode = 'min',
    verbose = True,
    patience = 100,
    min_delta = 1e-5,
    restore_best_weights = True
)

retrain_after_crossval = True

if retrain_after_crossval:
    model = rnn(
        n_units = 8,
        reg = 1e-06,
        learning_rate = 1e-04,
        seed = 5
    )
    model.fit(
        X_train,
        y_train,
        epochs = max_epoch,
        batch_size = batch_size,
        validation_data = (X_val, y_val),
        callbacks = [early_stopping]
    )

    print(min(model.history.history['val_loss']))
```

```
pd.DataFrame(
    {
        'origin_reference': output_val[(lag + ahead - 1):].values.flatten(),
        'actual': y_val[:, -1].flatten()*(output_max_val-input_min_val)
\ + input_min_val,
        'pred': model.predict(X_val)[:, -1].flatten()
\ * (input_max_val - input_min_val)
\ + input_min_val,
    },
    index = output_val[(lag + ahead -1): ].index
).to_csv(f'thesis_vis/{stock}_type_{type_of_output}_test.csv')
```

Code: Logit Machine Learning Model

```
import numpy as np
import pandas as pd
from thesis_classicalML_train_val import X_train, y_train, X_val, y_val, \
      index_of_val, ibm_df
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

np.random.seed(5)
logit = LogisticRegression(penalty = 'l2', tol = 1e-4, max_iter = 1000)
params = {
    'C': [1e-02, 1e-01, 1, 10, 100, 1000, 10000, 100000]
}

grid_seach = GridSearchCV(
    estimator = logit,
    param_grid = params,
    return_train_score = True,
    cv = 5
)

grid_results = grid_seach.fit(X_train, y_train)
logit = LogisticRegression(C = 300,penalty =  'l2')
logit.fit(X_train, y_train)
pd.DataFrame(
    {
        'actual': ibm_df[index_of_val],
        'predict': logit.predict(X_val),
    },
    index = index_of_val
).to_csv('thesis_vis/IBM_logistic_test.csv')
```

Code: SAFE Statistical Test for Robustness

```
import tensorflow as tf

from TS_GRU_frame import get_gru_frame
from TS_LSTM_frame import get_lstm_frame
from TS_RNN_frame import get_rnn_frame
from TS_data_deep_learning import x_2d_train, y_2d_train, x_2d_val, y_2d_val
from TS_data_classical_ml import x_2d_train_classic, y_2d_train_classic,
x_2d_val_classic, y_2d_val_classic

from TS_utils import main_folder_fn
from safeaipackage.check_robustness import Robustness, RobustnessForClassicalML

from sklearn.svm import SVC, SVR

from sklearn.linear_model import LogisticRegression

N_UNITS = 8
REG = 1e-06
LEARNING_RATE = 1e-03
SEED = 5

tf.random.set_seed(SEED)

rnn_frame = get_rnn_frame(x_train_2d=x_2d_train, reg=REG,
learning_rate=LEARNING_RATE, n_units=N_UNITS)
lstm_frame = get_lstm_frame(x_train_2d=x_2d_train, reg=REG,
learning_rate=LEARNING_RATE, n_units=N_UNITS)
gru_frame = get_gru_frame(x_train_2d=x_2d_train, reg=REG,
learning_rate=LEARNING_RATE, n_units=N_UNITS)


rnn_fn = f"{main_folder_fn}/RNN_test.h5"
lstm_fn = f"{main_folder_fn}/LSTM_test.h5"
gru_fn = f"{main_folder_fn}/GRU_test.h5"

# Restore all models
lstm_frame.load_weights(lstm_fn)
gru_frame.load_weights(gru_fn)
rnn_frame.load_weights(rnn_fn)
svc = SVC(C=0.1, max_iter=10000)
svm = SVR(C=0.1, max_iter=10000)
logit = LogisticRegression(penalty = 'l2', tol = 1e-4,
max_iter = 10000, C=10000)

lstm_robust = Robustness(xtrain=x_2d_train, ytrain=y_2d_train,
xtest=x_2d_val, ytest=y_2d_val, model=lstm_frame)
gru_robust = Robustness(xtrain=x_2d_train, ytrain=y_2d_train,
xtest=x_2d_val, ytest=y_2d_val, model=gru_frame)
rnn_robust = Robustness(xtrain=x_2d_train, ytrain=y_2d_train,
xtest=x_2d_val, ytest=y_2d_val, model=rnn_frame)

print(f"--- P-VALUE ROBUST LSTM-GRU")
print(lstm_robust.rgr_statistic_test(problemtype="prediction",
```

```
secondmodel=gru_frame))
print("-" * 120)
print(f"--- P-VALUE ROBUST LSTM-RNN")
print(lstm_robust.rgr_statistic_test(problemtype="prediction",
secondmodel=rnn_frame))
print("-" * 120)
print(f"--- P-VALUE ROBUST LSTM-SVM")
print(lstm_robust.rgr_statistic_test(problemtype="prediction",
secondmodel=svm))
print("-" * 120)
print(f"--- P-VALUE ROBUST GRU-RNN")
print(gru_robust.rgr_statistic_test(problemtype="prediction",
secondmodel=rnn_frame))
print(f"--- P-VALUE ROBUST GRU-SVM")
print(gru_robust.rgr_statistic_test(problemtype="prediction",
secondmodel=svm))
print("-" * 120)
print(f"--- P-VALUE ROBUST RNN-SVM")
print(rnn_robust.rgr_statistic_test(problemtype="prediction",
secondmodel=svm))
print(f"--- P-VALUE ROBUST SVC-LOGIT")
svm_robust = RobustnessForClassicalML(xtrain=x_2d_train_classic,
ytrain=y_2d_train_classic,
xtest=x_2d_val_classic, ytest=y_2d_val_classic,
                                      model=svc)
print(svm_robust.rgr_statistic_test(problemtype="classification",
secondmodel=logit, perturbation_percentage=0.45))
```

Code: SAFE AI GRU

```
import tensorflow as tf
from TS_GRU_frame import get_gru_frame
from TS_data_deep_learning import x_2d_train, y_2d_train,
x_2d_val, y_2d_val
from TS_utils import main_folder_fn
from safeaipackage.check_accuracy import Accuracy
from safeaipackage.check_robustness import Robustness
from safeaipackage.check_explainability import Explainability

N_UNITS = 8
REG = 1e-06
LEARNING_RATE = 1e-03
SEED = 5

tf.random.set_seed(SEED)
fn = f"{main_folder_fn}/GRU_test.h5"

gru_frame = get_gru_frame(x_train_2d=x_2d_train, reg=REG,
learning_rate=LEARNING_RATE, n_units=N_UNITS)
# Load Weights to get pre-trained model;
gru_frame.load_weights(fn)

# Safe AI Modeling;
print(f"-- ACCURACY: ")
gru_accuracy = Accuracy(xtrain=x_2d_train, ytrain=y_2d_train,
xtest=x_2d_val, ytest=y_2d_val, model=gru_frame)
print(gru_accuracy.rga())

print("-" * 120)
print(f"-- ROBUSTNESS: ")
gru_robustness = Robustness(xtrain=x_2d_train, ytrain=y_2d_train,
xtest=x_2d_val, ytest=y_2d_val, model=gru_frame)
print(gru_robustness.rgr_all(perturbation_percentage=0.10))

print(f"--- EXPLAIN")
ex = Explainability(xtrain=x_2d_train, ytrain=y_2d_train,
xtest=x_2d_val, ytest=y_2d_val, model=gru_frame)
ex.rge()
```

Code: SAFE AI LSTM

```
import tensorflow as tf
from TS_LSTM_frame import get_lstm_frame
from TS_data_deep_learning import x_2d_train, y_2d_train,
 x_2d_val, y_2d_val
from TS_utils import main_folder_fn
from safeaipackage.check_accuracy import Accuracy
from safeaipackage.check_robustness import Robustness
from safeaipackage.check_explainability import Explainability


N_UNITS = 8
REG = 1e-06
LEARNING_RATE = 1e-03
SEED = 5

tf.random.set_seed(SEED)
fn = f"{main_folder_fn}/LSTM_test.h5"


lstm_frame = get_lstm_frame(x_train_2d=x_2d_train,
reg=REG, learning_rate=LEARNING_RATE, n_units=N_UNITS)
lstm_frame.load_weights(fn)

# Safe AI Modeling;
print(f"-- ACCURACY: ")
gru_accuracy = Accuracy(xtrain=x_2d_train, ytrain=y_2d_train,
xtest=x_2d_val, ytest=y_2d_val, model=lstm_frame)
print(gru_accuracy.rga())

print("-" * 120)
print(f"-- ROBUSTNESS: ")
gru_robustness = Robustness(xtrain=x_2d_train, ytrain=y_2d_train,
 xtest=x_2d_val, ytest=y_2d_val, model=lstm_frame)
print(gru_robustness.rgr_all(perturbation_percentage=0.10))
print(f"--- EXPLAIN")
ex = Explainability(xtrain=x_2d_train, ytrain=y_2d_train,
xtest=x_2d_val, ytest=y_2d_val, model=lstm_frame)
ex.rge()
```

Code: SAFE AI RNN

```
import tensorflow as tf

from TS_RNN_frame import get_rnn_frame


from TS_data_deep_learning import x_2d_train,
y_2d_train, x_2d_val, y_2d_val
from TS_utils import main_folder_fn
from safeaipackage.check_accuracy import Accuracy
from safeaipackage.check_robustness import Robustness
from safeaipackage.check_explainability import Explainability

N_UNITS = 8
REG = 1e-06
LEARNING_RATE = 1e-03
SEED = 5

tf.random.set_seed(SEED)
fn = f"{main_folder_fn}/RNN_test.h5"
rnn_frame = get_rnn_frame(x_train_2d=x_2d_train, reg=REG,
learning_rate=LEARNING_RATE, n_units=N_UNITS)
rnn_frame.load_weights(fn)

# Safe AI Modeling;
print(f"-- ACCURACY: ")
gru_accuracy = Accuracy(xtrain=x_2d_train, ytrain=y_2d_train,
xtest=x_2d_val, ytest=y_2d_val, model=rnn_frame)
print(gru_accuracy.rga())

print("-" * 120)
print(f"-- ROBUSTNESS: ")
gru_robustness = Robustness(xtrain=x_2d_train, ytrain=y_2d_train,
xtest=x_2d_val, ytest=y_2d_val, model=rnn_frame)

print(gru_robustness.rgr_all(perturbation_percentage=0.10))
print("-" * 120)
print(f"--- EXPLAIN")

ex = Explainability(xtrain=x_2d_train, ytrain=y_2d_train,
xtest=x_2d_val, ytest=y_2d_val, model=rnn_frame)
ex.rge()
```

Code: SAFE AI SVM

```
import copy
import numpy as np
from sklearn.svm import SVC
from safeaipackage.check_accuracy import AccuracyForClassicalML
from safeaipackage.check_robustness import RobustnessForClassicalML
from  safeaipackage.check_explainability import ExplainabilityForClassicalML


from TS_data_classical_ml import x_2d_train,
y_2d_train, x_2d_val, y_2d_val

np.random.seed(5)

X_train = copy.deepcopy(x_2d_train)
y_train = copy.deepcopy(y_2d_train)
X_val = copy.deepcopy(x_2d_val)
y_val = copy.deepcopy(y_2d_val)

np.random.seed(5)
svc = SVC(C = 0.1, max_iter = 10000)

svc.fit(X_train, y_train)
svc.score(X_val, y_val)
svm_acc = AccuracyForClassicalML(xtrain=X_train,
ytrain=y_train, xtest=X_val, ytest=y_val, model=svc)

print("--- ACCURACY: ")
print(svm_acc.rga())
print("P-value")
print(svm_acc.rga_statistic_test(problemtype="prediction"))
print("-" * 120)
robustness = RobustnessForClassicalML(xtrain=X_train,
ytrain=y_train, xtest=X_val, ytest=y_val, model=svc)
print("-- ROBUSTNESS")
print(robustness.rgr_all(perturbation_percentage=0.10))

print("--- EXPLAINABLE")
ex = ExplainabilityForClassicalML(xtrain=X_train, ytrain=y_train,
xtest=X_val, ytest=y_val, model=svc)
ex.rge()
```

Code: SAFE AI LOGIT

```
import copy
import numpy as np
from sklearn.linear_model import LogisticRegression

from TS_data_classical_ml import x_2d_train,
y_2d_train, x_2d_val, y_2d_val
from safeaipackage.check_accuracy import AccuracyForClassicalML
from safeaipackage.check_explainability import ExplainabilityForClassicalML
from safeaipackage.check_robustness import RobustnessForClassicalML

np.random.seed(5)
logit = LogisticRegression(penalty = 'l2', tol = 1e-4, max_iter = 10000, C=10000)

X_train = copy.deepcopy(x_2d_train)
y_train = copy.deepcopy(y_2d_train)
X_val = copy.deepcopy(x_2d_val)
y_val = copy.deepcopy(y_2d_val)

logit.fit(X_train, y_train)
accuracy_obj = AccuracyForClassicalML(xtrain=X_train,
ytrain=y_train, xtest=X_val, ytest=y_val, model=logit)
print(f"--- ACCURACY: ")
print(accuracy_obj.rga())
print("-" * 120)
print(f"--- ROBUSTESS: ")
robustness_obj =  RobustnessForClassicalML(xtrain=X_train,
ytrain=y_train, xtest=X_val, ytest=y_val, model=logit)
print(robustness_obj.rgr_all(perturbation_percentage=0.11))

print("-" * 120)
print("--- EXPLAINABILITY")
explain =ExplainabilityForClassicalML(xtrain=X_train, ytrain=y_train,
xtest=X_val, ytest=y_val, model=logit)
explain.rge()
```

Code Exam: SVM Machine Learning Model

```
import numpy as np
from thesis_classicalML_train_val import X_train, y_train,
      \ X_val, y_val, index_of_val, ibm_df
from sklearn.model_selection import GridSearchCV
import pandas as pd
from sklearn.svm import SVC
np.random.seed(5)

svc = SVC(C = 1600, max_iter = 10000)
svc.fit(X_train, y_train)
svc.score(X_val, y_val)

pd.DataFrame(
    {
        'actual': ibm_df[index_of_val],
        'predict': svc.predict(X_val),

    },

    index = index_of_val
).to_csv('thesis_vis/IBM_svc_test.csv')
```

Visualization Snipets of Code in R

Code: Visualization For Predcitions of GRU Model

```
cat('\f')
rm(list = ls())
library(dplyr)
library(tidyverse)
library(lubridate)
library(grDevices)

df <- readxl::read_xlsx('thesis_vis/ibm_gru_strategy.xlsx')
df <- df[-1, ]
% Visual Here:

pdf_file <- 'thesis_vis/ibm_gru_predict.pdf'
cairo_pdf(bg = 'grey98', pdf_file, width = 9.6, height = 5.8)
par(
  omi = c(0.25, 0.5, 0.75, 0.6),
  mai = c(0.25, 0.5, 0.05, 0.8),
  family = 'Lato Light',
  las = 1,
  mgp = c(1, 1, 0)
)
par(cex = 0.85, bg = 'white')
myColor1 <- 'brown'
myColor2 <- 'blue'
myColor3 <- 'red'
myColor4 <- 'green'

min_value <- min(df$actual)
max_value <- max(df$actual)

plot(as.Date(df$Date), df$actual, type = 'n',
axes = FALSE, xlab = "", ylab = '',
     col = rgb(255, 97, 0, 150, maxColorValue = 255),
lwd = 1.5)

points(as.Date(df$Date), df$actual, col = myColor1, lwd = 1, type = 'l')
points(as.Date(df$Date), df$actual, col = myColor1, pch = 19, cex = 0.5)
points(as.Date(df$Date), df$pred, col = myColor2, lwd = 1, type = 'l')
points(as.Date(df$Date), df$pred, col = myColor2, pch = 19, cex = 0.5)
% # Legend,
legend(as.Date('2020-10-20'), 100, c('Original price of IBM', 'Predicted by GRU'),
       border = NA, pch = c(19, 19), col = c(myColor1, myColor2), cex = 0.8,
      \  bty = 'n')
% # Axis;
axis(1, col = rgb(105, 105, 105, maxColorValue = 255), col.
      \ ticks = rgb(105, 105, 105, maxColorValue = 255),
     at = seq(as.Date('2019-02-01'), as.Date('2021-03-01'), length.out = 60),
      cex.axis = 0.8,
     lwd.ticks = 0.15, tck = 0.015, family = 'Lato Light',
     labels = format(seq(as.Date('2019-02-01'), as.Date('2021-03-01'),
        \ length.out = 60), '%b\n%Y'))
```

```
axis(2, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     lwd.ticks = 0.15, cex.axis = 0.8, at =  round(seq(min_value, max_value-5,
      \ length.out = 5), 0))
% # mtext;
mtext('The original price of IBM and Its prediction by GRU model',
      3, line = 1.6, adj = 0.05, cex = 1.09, family = 'Lato Black', outer = TRUE)
mtext('The model predicted IBM price from 02/2019 to 07/2020',
      3, line = -0.0, adj = 0.06, cex = 0.85, family = 'Lato Light', outer = TRUE,
    \ font = 3)
dev.off()
```

Code: Visualization For Predcitions of LSTM Model

```
cat('\f')
rm(list = ls())
library(dplyr)
library(tidyverse)
library(lubridate)
library(grDevices)

df <- readxl::read_xlsx('thesis_vis/ibm_lstm_strategy.xlsx')
df <- df[-1, ]
# Visual here;
pdf_file <- 'thesis_vis/ibm_lstm_predict.pdf'
cairo_pdf(bg = 'grey98', pdf_file, width = 9.6, height = 5.8)
par(
  omi = c(0.25, 0.5, 0.75, 0.8),
  mai = c(0.25, 0.5, 0.05, 0.6),
  family = 'Lato Light',
  las = 1,
  mgp = c(1, 1, 0)
)
par(cex = 0.85, bg = 'white')
myColor1 <- 'brown'
myColor2 <- 'blue'
myColor3 <- 'red'
myColor4 <- 'green'
min_value <- min(df$actual)
max_value <- max(df$actual)

plot(as.Date(df$Date), df$actual, type = 'n', axes = FALSE,
xlab = "", ylab = '',
     col = rgb(255, 97, 0, 150, maxColorValue = 255),
 lwd = 1.5)

points(as.Date(df$Date), df$actual, col = myColor1, lwd = 1, type = 'l')
points(as.Date(df$Date), df$actual, col = myColor1, pch = 19, cex = 0.5)
points(as.Date(df$Date), df$pred, col = myColor2, lwd = 1, type = 'l')
points(as.Date(df$Date), df$pred, col = myColor2, pch = 19, cex = 0.5)
% # Legend,
legend(as.Date('2020-10-20'), 100,
c('Original price of IBM', 'Predicted by LSTM'),
       border = NA, pch = c(19, 19),
col = c(myColor1, myColor2),
  cex = 0.8, bty = 'n')
# Axis;
axis(1, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     at = seq(as.Date('2019-02-01'),
 as.Date('2021-03-01'), length.out = 60),
  cex.axis = 0.8,
     lwd.ticks = 0.15, tck = 0.015, family = 'Lato Light',
     labels = format(seq(as.Date('2019-02-01'), as.Date('2021-03-01'),
length.out = 60), '%b\n%Y'))

axis(2, col = rgb(105, 105, 105, maxColorValue = 255),
```

```
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     lwd.ticks = 0.15, cex.axis = 0.8, at =  round(seq(min_value, max_value-5,
length.out = 5), 0))

%# mtext;
mtext('The original price of IBM and Its prediction by LSTM model',
      3, line = 1.6, adj = 0.05, cex = 1.09,
family = 'Lato Black', outer = TRUE)
mtext('The model predicted IBM price from 02/2019 to 07/2020',
      3, line = -0.0, adj = 0.06, cex = 0.85,
family = 'Lato Light', outer = TRUE,  font = 3)
dev.off()
```

Code Exam: Visualization For Predcitions of RNN Model

```
cat('\f')
rm(list = ls())

library(dplyr)
library(tidyverse)
library(lubridate)
library(grDevices)

df <- readxl::read_xlsx('thesis_vis/ibm_rnn_strategy.xlsx')
df <- df[-1, ]

% # Visual here;
pdf_file <- 'thesis_vis/ibm_rnn_predict.pdf'
cairo_pdf(bg = 'grey98', pdf_file, width = 9.6, height = 5.8)
par(

  omi = c(0.25, 0.5, 0.75, 0.8),
  mai = c(0.25, 0.5, 0.05, 0.6),
  family = 'Lato Light',
  las = 1,
  mgp = c(1, 1, 0)

)
par(cex = 0.85, bg = 'white')
myColor1 <- 'brown'
myColor2 <- 'blue'
myColor3 <- 'red'
myColor4 <- 'green'
min_value <- min(df$actual)
max_value <- max(df$actual)

plot(as.Date(df$Date), df$actual, type = 'n', axes = FALSE, xlab = "",
ylab = '',  col = rgb(255, 97, 0, 150, maxColorValue = 255), lwd = 1.5)

points(as.Date(df$Date), df$actual, col = myColor1, lwd = 1, type = 'l')
points(as.Date(df$Date), df$actual, col = myColor1, pch = 19, cex = 0.5)
points(as.Date(df$Date), df$pred, col = myColor2, lwd = 1, type = 'l')
points(as.Date(df$Date), df$pred, col = myColor2, pch = 19, cex = 0.5)

% # Legend,
legend(as.Date('2020-10-20'), 100, c('Original price of IBM', 'Predicted by RNN'),
       border = NA, pch = c(19, 19), col = c(myColor1, myColor2), cex = 0.8,
bty = 'n')

% # Axis;
axis(1, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     at = seq(as.Date('2019-02-01'), as.Date('2021-03-01'),
length.out = 60),
cex.axis = 0.8,
     lwd.ticks = 0.15, tck = 0.015, family = 'Lato Light',
     labels = format(seq(as.Date('2019-02-01'), as.Date('2021-03-01'),
length.out = 60), '%b\n%Y'))
```

```
axis(2, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     lwd.ticks = 0.15, cex.axis = 0.8, at =  round(seq(min_value, max_value-5,
length.out = 5), 0))

% # mtext;
mtext('The original price of IBM and Its prediction by RNN model',
      3, line = 1.6, adj = 0.05, cex = 1.09, family = 'Lato Black', outer = TRUE)
mtext('The model predicted IBM price from 02/2019 to 07/2020',
      3, line = -0.0, adj = 0.06, cex = 0.85, family = 'Lato Light', outer = TRUE,
font = 3)

dev.off()
```

Code: Visualization For Cross-Validation in Machine Learning Model Selection:

```
cat('\f')
rm(list = ls())
library(dplyr)
library(tidyverse)
library(lubridate)
library(grDevices)

sku <- 'DB6F'
df <- read_csv(paste0('yes4all/topsku/', sku, '.csv'))
names(df) <- c('Time', 'Qty')
df$Time <- format(
  seq(as.POSIXct('2019-01-01'), by = 'week', length.out = dim(df)[1]),
  '%Y-%m-%d'
)

# Visualise here;
font1 <- 'Times New Roman'
font2 <- 'Lato Light'
pdf_file <- 'thesis_vis/cross_val.pdf'
cairo_pdf(bg = 'grey98', pdf_file, width = 9, height = 5.8)
par(
  omi = c(0.05, 0.5, 0.75, 1.0),
  mai = c(0.05, 0.5, 0.05, 0.5),
  family = 'Lato Light',
  las = 1,
  mgp = c(1, 1, 0)
)
par(cex = 0.85, bg = 'white')
#rgb(240, 240, 240, maxColorValue = 255)
myColor1 <- 'brown'
myColor2 <- 'blue'
myColor3 <- 'red'

plot(as.Date(df$Time), df$Qty, type = 'n', axes = FALSE, xlab = "", ylab = "",
     col = rgb(255, 97, 0, 150, maxColorValue = 255), lwd = 1.5)

# rectangular: type 1 cross-validation;
n <- 1250
rect(as.Date('2019-01-01'), n-50, as.Date('2019-01-01')+400, n, col = myColor2,
     border = NA)
rect(as.Date('2020-02-01'), n-50, as.Date('2020-02-01')+100, n, col = myColor3,
     border = NA)
rect(as.Date('2019-01-01')+100, n-150, as.Date('2019-01-01')+500, n-100,
     col = myColor2, border = NA)
rect(as.Date('2019-01-01')+500, n-150, as.Date('2019-01-01')+600, n-100,
     col = myColor3, border = NA)

rect(as.Date('2019-01-01')+200, n-250, as.Date('2019-01-01')+600, n-200,
     col = myColor2, border = NA)
rect(as.Date('2019-01-01')+600, n-250, as.Date('2019-01-01')+700, n-200,
     col = myColor3, border = NA)

rect(as.Date('2019-01-01')+300, n-350, as.Date('2019-01-01')+700, n-300,
```

```
     col = myColor2, border = NA)

rect(as.Date('2019-01-01')+700, n-350, as.Date('2019-01-01')+800, n-300,
     col = myColor3, border = NA)

# Rect: Type2 cross-validation:
m <- 700
rect(as.Date('2019-01-01'), m-50, as.Date('2019-01-01')+400, m,col = myColor2,
     border = NA)
rect(as.Date('2019-01-01')+400, m-50, as.Date('2019-01-01')+500, m, col = myColor3,
     border = NA)
rect(as.Date('2019-01-01'), m-150, as.Date('2019-01-01')+500, m-100,col = myColor2,
     border = NA)
rect(as.Date('2019-01-01')+500, m-150, as.Date('2019-01-01')+600, m-100,
col = myColor3,
     border = NA)
rect(as.Date('2019-01-01'), m-250, as.Date('2019-01-01')+600, m-200,
col = myColor2,
     border = NA)
rect(as.Date('2019-01-01')+600, m-250, as.Date('2019-01-01')+700, m-200,
col = myColor3,
     border = NA)
rect(as.Date('2019-01-01'), m-350, as.Date('2019-01-01')+700, m-300,
col = myColor2,
     border = NA)
rect(as.Date('2019-01-01')+700, m-350, as.Date('2019-01-01')+800, m-300,
col = myColor3,
     border = NA)

# Legend;
p <- 200
legend(as.Date('2020-10-01'), p, c('Trained Data', 'Validated Data'),
       border = NA, pch = 15, col = c(myColor2, myColor3), cex = 0.95,
       bty = 'n')
# Text;
text(as.Date('2019-01-01'), n+80, 'Type 1: Time Series Cross-Validation',
     adj = 0, xpd = TRUE, cex = 1.01, font = 3)
text(as.Date('2019-01-01'), m+80, 'Type 2: Time Series Cross-Validation',
     adj = 0, xpd = TRUE, cex = 1.01, font = 3)

# Margin text;
mtext('Cross-Validation of the Supervised Learning form of the Time Series Data',
      3, line = 0, adj = 0.25, cex = 1.2, family = 'Lato Black',
      outer = TRUE)
mtext('These are 2 examples of "4-folds" cross-validation in Time Series Data',
      3, line = -1.6, adj = 0.1, cex = 0.9, family = 'Lato Light',
      outer = TRUE, font = 3)

dev.off()
```

Code Exam: Visualization For GRU Model Long/Short Positions

```
cat('\f')
rm(list = ls())
library(dplyr)
library(tidyverse)
library(lubridate)
library(grDevices)
df <- readxl::read_xlsx('thesis_vis/ibm_gru_strategy.xlsx')
df <- df[-1, ]

# Visual here;

pdf_file <- 'thesis_vis/ibm_gru_place_position.pdf'
cairo_pdf(bg = 'grey98', pdf_file, width = 9.6, height = 5.8)
par(
  omi = c(0.25, 0.5, 0.75, 0.6),
  mai = c(0.25, 0.5, 0.05, 0.8),
  family = 'Lato Light',
  las = 1,
  mgp = c(1, 1, 0)
)
par(cex = 0.85, bg = 'white')

myColor1 <- 'brown'
myColor2 <- 'blue'
myColor3 <- 'red'
myColor4 <- 'green'

min_value <- min(df$actual)
max_value <- max(df$actual)

plot(as.Date(df$Date), df$actual, type = 'n', axes = FALSE, xlab = "",
 ylab ="",  col = rgb(255, 97, 0, 150, maxColorValue = 255), lwd = 1.5)

points(as.Date(df$Date), df$actual, lwd = 1, type = 'l', col = myColor1)
points(as.Date(df$Date), df$actual, col = myColor1, pch = 19, cex = 0.5)
points(as.Date(df$Date), if_else(df$strategy == "buy", df$actual, NULL),
col = myColor4, pch = 17, cex = 1)

# Legend,

legend(as.Date('2020-06-20'), 100,
c('Original price of IBM', 'Long Position by GRU-based model'),
       border = NA, pch = c(19, 17), col = c(myColor1, myColor4),
cex = 0.8, bty = 'n')
# Axis;
axis(1, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     at = seq(as.Date('2019-02-01'), as.Date('2021-03-01'), length.out = 60),
cex.axis = 0.8,
     lwd.ticks = 0.15, tck = 0.015, family = 'Lato Light',
     labels = format(seq(as.Date('2019-02-01'), as.Date('2021-03-01'),
length.out = 60), '%b\n%Y'))
```

```
axis(2, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     lwd.ticks = 0.15, cex.axis = 0.8,
 at =  round(seq(min_value, max_value-5, length.out = 5), 2))

# mtext;
mtext('The original price of IBM and Long positions by GRU-based Strategy',
      3, line = 1.6, adj = 0.05, cex = 1.09, family = 'Lato Black', outer = TRUE)
mtext('The Strategy has been placed long only, short-sell is restricted',
      3, line = -0.0, adj = 0.06, cex = 0.85, family = 'Lato Light', outer = TRUE,
font = 3)
dev.off()
```

Code: Visualization For Profit Loss of GRU Model Strategies

```
cat('\f')
rm(list = ls())
library(dplyr)
library(tidyverse)
library(lubridate)
library(grDevices)

df <- readxl::read_xlsx('thesis_vis/ibm_gru_strategy.xlsx')
df <- df[-1, ]

# Visual here;
pdf_file <- 'thesis_vis/ibm_gru_strategy.pdf'
cairo_pdf(bg = 'grey98', pdf_file, width = 9.6, height = 5.8)
par(
  omi = c(0.25, 0.5, 0.75, 0.6),
  mai = c(0.25, 0.5, 0.05, 0.8),
  family = 'Lato Light',
  las = 1,
  mgp = c(1, 1, 0)
)
par(cex = 0.85, bg = 'white')

myColor1 <- 'brown'
myColor2 <- 'blue'
myColor3 <- 'red'
myColor4 <- 'green'

min_value <- min(min(df$acc_buyhold_return), min(df$acc_strategy_return))
max_value <- max(max(df$acc_buyhold_return), max(df$acc_strategy_return))

plot(as.Date(df$Date), df$acc_buyhold_return, type = 'n', axes = FALSE,
xlab = "", ylab = '',
     col = rgb(255, 97, 0, 150, maxColorValue = 255), lwd = 1.5,
ylim = c(min_value+0.01, max_value-0.01))

points(as.Date(df$Date), df$acc_buyhold_return,
col = myColor1, lwd = 1, type = 'l')
points(as.Date(df$Date), df$acc_buyhold_return,
col = myColor1, pch = 19, cex = 0.6)
points(as.Date(df$Date), df$acc_strategy_return,
col = myColor2, lwd = 1, type = 'l')
points(as.Date(df$Date), df$acc_strategy_return,
col = myColor2, pch = 19, cex = 0.6)

# Legend;
legend(as.Date('2020-06-20'), -0.15,
c('Accumulated return of buy and hold Strategy',
'Accumulated return of GRU-based Strategy'),
       border = NA, pch = 19, col = c(myColor1, myColor2), cex = 0.8, bty = 'n')

# mtext;
mtext('Accumulated Return of IBM stock with buy & hold and GRU-based Strategy',
      3, line = 1.6, adj = 0.05, cex = 1.09, family = 'Lato Black', outer = TRUE)
```

```
mtext('Both Strategies were validated from 02/2019 to 03/2021',
      3, line = -0, adj = 0.06, cex = 0.85, family = 'Lato Light', outer = TRUE,
font = 3)

# Axis;
axis(1, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     at = seq(as.Date('2019-02-01'), as.Date('2021-03-01'),
length.out = 60), cex.axis = 0.8,
     lwd.ticks = 0.15, tck = 0.015, family = 'Lato Light',
     labels = format(seq(as.Date('2019-02-01'),
as.Date('2021-03-01'), length.out = 60), '%b\n%Y'))

axis(2, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     lwd.ticks = 0.15, cex.axis = 0.8,
at =  round(seq(min_value, max_value-0.05, length.out = 5), 2),
     labels =
paste0(round(seq(min_value, max_value-0.05, length.out = 5), 2)*100, '%'))

dev.off()
```

Code Exam: Visualization For Profit Loss of Logistic Model Strategies

```
cat('\f')
rm(list = ls())
library(dplyr)
library(tidyverse)
library(lubridate)
library(grDevices)

df <- readxl::read_xlsx('thesis_vis/ibm_logistic_strategy.xlsx')
df <- df[-1, ]

# Visual here;
pdf_file <- 'thesis_vis/ibm_logistic_strategy.pdf'
cairo_pdf(bg = 'grey98', pdf_file, width = 9.6, height = 5.8)
par(
  omi = c(0.25, 0.55, 0.75, 0.6),
  mai = c(0.25, 0.55, 0.05, 0.8),
  family = 'Lato Light',
  las = 1,
  mgp = c(1, 1, 0)
)

par(cex = 0.85, bg = 'white')
myColor1 <- 'brown'
myColor2 <- 'blue'
myColor3 <- 'red'
myColor4 <- 'green'

min_value <- min(min(df$acc_buyhold_return), min(df$acc_strategy_return))
max_value <- max(max(df$acc_buyhold_return), max(df$acc_strategy_return))

plot(as.Date(df$Date), df$acc_buyhold_return, type = 'n', axes = FALSE,
xlab = "", ylab = '',
     col = rgb(255, 97, 0, 150, maxColorValue = 255), lwd = 1.5,
ylim = c(min_value+0.01, max_value-0.01))

points(as.Date(df$Date), df$acc_buyhold_return, col = myColor1, lwd = 1,
type = 'l')
points(as.Date(df$Date), df$acc_buyhold_return, col = myColor1, pch = 19,
cex = 0.6)
points(as.Date(df$Date), df$acc_strategy_return, col = myColor2, lwd = 1,
type = 'l')
points(as.Date(df$Date), df$acc_strategy_return, col = myColor2, pch = 19,
cex = 0.6)

# Legend;
legend(as.Date('2020-06-30'), -0.20,
       c('Accumulated return of buy and hold Strategy',
'Accumulated return of LR-based Strategy'),
       border = NA, pch = 19, col = c(myColor1, myColor2), cex = 0.8, bty = 'n')

# mtext;
mtext('Accumulated Return of IBM stock with buy & hold and LR-based Strategy',
      3, line = 2.6, adj = 0.05, cex = 1.09, family = 'Lato Black', outer = TRUE)
```

```
mtext('Both Strategies were validated from 02/2019 to 03/2021',
      3, line = 1, adj = 0.06, cex = 0.85, family = 'Lato Light', outer = TRUE,
font = 3)

# Axis;
axis(1, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     at = seq(as.Date('2019-02-01'), as.Date('2021-03-01'),
length.out = 60),
cex.axis = 0.8,
     lwd.ticks = 0.15, tck = 0.015, family = 'Lato Light',
     labels = format(seq(as.Date('2019-02-01'), as.Date('2021-03-01'),
length.out = 60), '%b\n%Y'))

axis(2, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     lwd.ticks = 0.15, cex.axis = 0.8,
at =  round(seq(min_value, max_value-0.05, length.out = 5), 2),
     labels = paste0(round(seq(min_value, max_value-0.05, length.out = 5), 2)*100,
'%'))
dev.off()
```

Code Exam: Visualization For Long Short Position of Logistic Model

```
cat('\f')
rm(list = ls())

library(dplyr)
library(tidyverse)
library(lubridate)
library(grDevices)

df <- readxl::read_xlsx('thesis_vis/ibm_logistic_strategy.xlsx')
df <- df[-1, ]

# Visual here;
pdf_file <- 'thesis_vis/ibm_logistic_place_position.pdf'
cairo_pdf(bg = 'grey98', pdf_file, width = 9.6, height = 5.8)
par(
  omi = c(0.25, 0.5, 0.75, 0.6),
  mai = c(0.25, 0.5, 0.05, 0.8),
  family = 'Lato Light',
  las = 1,
  mgp = c(1, 1, 0)
)
par(cex = 0.85, bg = 'white')

myColor1 <- 'brown'
myColor2 <- 'blue'
myColor3 <- 'red'
myColor4 <- 'green'

min_value <- min(df$actual)
max_value <- max(df$actual)

plot(as.Date(df$Date), df$actual, type = 'n', axes = FALSE, xlab = "", ylab = '',
     col = rgb(255, 97, 0, 150, maxColorValue = 255), lwd = 1.5)

points(as.Date(df$Date), df$actual, lwd = 1, type = 'l', col = myColor1)
points(as.Date(df$Date), df$actual, col = myColor1, pch = 19, cex = 0.5)
points(as.Date(df$Date), if_else(df$strategy == "buy", df$actual, NULL),
col = myColor4, pch = 17, cex = 1)

# Legend,
legend(as.Date('2020-06-20'), 100,
c('Original price of IBM', 'Long Position by LR-based Strategy'),
       border = NA, pch = c(19, 17),
col = c(myColor1, myColor4), cex = 0.8, bty = 'n')
# Axis;
axis(1, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     at = seq(as.Date('2019-02-01'), as.Date('2021-03-01'), length.out = 60),
cex.axis = 0.8,
     lwd.ticks = 0.15, tck = 0.015, family = 'Lato Light',
     labels = format(seq(as.Date('2019-02-01'), as.Date('2021-03-01'),
length.out = 60), '%b\n%Y'))
```

```
axis(2, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     lwd.ticks = 0.15, cex.axis = 0.8,
at =  round(seq(min_value, max_value-5, length.out = 5), 0))

# mtext;
mtext('The original price of IBM and Long positions by based LR-based Strategy',
      3, line = 2.6, adj = 0.05, cex = 1.09, family = 'Lato Black', outer = TRUE)
mtext('The Strategy has been placed long only, short-sell is restricted',
      3, line = 1, adj = 0.06, cex = 0.85,
family = 'Lato Light', outer = TRUE, font = 3)

dev.off()
```

Code: Visualization For Profit or Loss of LSTM Model

```
cat('\f')
rm(list = ls())
library(dplyr)
library(tidyverse)
library(lubridate)
library(grDevices)

df <- readxl::read_xlsx('thesis_vis/ibm_lstm_strategy.xlsx')
df <- df[-1, ]

# Visual here;
pdf_file <- 'thesis_vis/ibm_lstm_strategy.pdf'
cairo_pdf(bg = 'grey98', pdf_file, width = 9.6, height = 5.8)
par(
  omi = c(0.25, 0.5, 0.75, 0.6),
  mai = c(0.25, 0.5, 0.05, 0.8),
  family = 'Lato Light',
  las = 1,
  mgp = c(1, 1, 0)
)
par(cex = 0.85, bg = 'white')

myColor1 <- 'brown'
myColor2 <- 'blue'
myColor3 <- 'red'
myColor4 <- 'green'

min_value <- min(min(df$acc_buyhold_return), min(df$acc_strategy_return))
max_value <- max(max(df$acc_buyhold_return), max(df$acc_strategy_return))

plot(as.Date(df$Date), df$acc_buyhold_return, type = 'n', axes = FALSE,
xlab = "", ylab = '',
     col = rgb(255, 97, 0, 150, maxColorValue = 255), lwd = 1.5,
ylim = c(min_value+0.01, max_value-0.01))

points(as.Date(df$Date), df$acc_buyhold_return,
col = myColor1, lwd = 1, type = 'l')
points(as.Date(df$Date), df$acc_buyhold_return,
col = myColor1, pch = 19, cex = 0.6)
points(as.Date(df$Date), df$acc_strategy_return,
col = myColor2, lwd = 1, type = 'l')
points(as.Date(df$Date), df$acc_strategy_return,
col = myColor2, pch = 19, cex = 0.6)

# Legend;
legend(as.Date('2020-06-20'), -0.15,
c('Accumulated return of buy and hold Strategy',
 'Accumulated return of LSTM-based Strategy'),
       border = NA, pch = 19, col = c(myColor1, myColor2),
cex = 0.8, bty = 'n')

# mtext;
mtext('Accumulated Return of IBM stock with buy and hold and LSTM-based Strategy',
```

```
      3, line = 1.6, adj = 0.05, cex = 1.09,
family = 'Lato Black', outer = TRUE)
mtext('Both Strategies were validated from 02/2019 to 03/2021',
      3, line = -0, adj = 0.06, cex = 0.85,
family = 'Lato Light', outer = TRUE, font = 3)

# Axis;
axis(1, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     at = seq(as.Date('2019-02-01'),
as.Date('2021-03-01'), length.out = 60), cex.axis = 0.8,
     lwd.ticks = 0.15, tck = 0.015, family = 'Lato Light',
     labels = format(seq(as.Date('2019-02-01'),
as.Date('2021-03-01'), length.out = 60), '%b\n%Y'))

axis(2, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     lwd.ticks = 0.15, cex.axis = 0.8,
at =  round(seq(min_value, max_value-0.05, length.out = 5), 2),
     labels =
paste0(round(seq(min_value, max_value-0.05, length.out = 5), 2)*100, '%'))

dev.off()
```

Code: Visualization Long Short Positions of LSTM Model

```
cat('\f')
rm(list = ls())
library(dplyr)
library(tidyverse)
library(lubridate)
library(grDevices)

df <- readxl::read_xlsx('thesis_vis/ibm_lstm_strategy.xlsx')
df <- df[-1, ]

# Visual here;
pdf_file <- 'thesis_vis/ibm_lstm_place_position.pdf'
cairo_pdf(bg = 'grey98', pdf_file, width = 9.6, height = 5.8)
par(
  omi = c(0.25, 0.5, 0.75, 0.6),
  mai = c(0.25, 0.5, 0.05, 0.8),
  family = 'Lato Light',
  las = 1,
  mgp = c(1, 1, 0)
)
par(cex = 0.85, bg = 'white')

myColor1 <- 'brown'
myColor2 <- 'blue'
myColor3 <- 'red'
myColor4 <- 'green'

min_value <- min(df$actual)
max_value <- max(df$actual)

plot(as.Date(df$Date), df$actual, type = 'n', axes = FALSE, xlab = "", ylab = '',
     col = rgb(255, 97, 0, 150, maxColorValue = 255), lwd = 1.5)

points(as.Date(df$Date), df$actual, lwd = 1, type = 'l', col = myColor1)
points(as.Date(df$Date), df$actual, col = myColor1, pch = 19, cex = 0.5)
points(as.Date(df$Date), if_else(df$strategy == "buy", df$actual, NULL),
col = myColor4, pch = 17, cex = 1)

# Legend,
legend(as.Date('2020-06-20'), 100,
c('Original price of IBM', 'Long Position by LSTM-based Strategy'),
       border = NA, pch = c(19, 17),
col = c(myColor1, myColor4), cex = 0.8, bty = 'n')

# Axis;
axis(1, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     at = seq(as.Date('2019-02-01'), as.Date('2021-03-01'), length.out = 60),
cex.axis = 0.8,
     lwd.ticks = 0.15, tck = 0.015, family = 'Lato Light',
     labels = format(seq(as.Date('2019-02-01'),
as.Date('2021-03-01'), length.out = 60), '%b\n%Y'))
```

```
axis(2, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     lwd.ticks = 0.15, cex.axis = 0.8,
at =  round(seq(min_value, max_value-5, length.out = 5), 0))

# mtext;
mtext('The original price of IBM and Long positions by LSTM-based Strategy',
      3, line = 1.6, adj = 0.05, cex = 1.09,
family = 'Lato Black', outer = TRUE)
mtext('The Strategy has been placed long only, short-sell is restricted',
      3, line = -0.0, adj = 0.06, cex = 0.85,
family = 'Lato Light', outer = TRUE, font = 3)
dev.off()
```

Code: Visualization Long Short Positions of RNN Model

```
cat('\f')
rm(list = ls())

library(dplyr)
library(tidyverse)
library(lubridate)
library(grDevices)

df <- readxl::read_xlsx('thesis_vis/ibm_rnn_strategy.xlsx')
df <- df[-1, ]

# Visual here;
pdf_file <- 'thesis_vis/ibm_rnn_place_position.pdf'
cairo_pdf(bg = 'grey98', pdf_file, width = 9.6, height = 5.8)
par(
  omi = c(0.25, 0.5, 0.75, 0.6),
  mai = c(0.25, 0.5, 0.05, 0.8),
  family = 'Lato Light',
  las = 1,
  mgp = c(1, 1, 0)
)
par(cex = 0.85, bg = 'white')

myColor1 <- 'brown'
myColor2 <- 'blue'
myColor3 <- 'red'
myColor4 <- 'green'

min_value <- min(df$actual)
max_value <- max(df$actual)

plot(as.Date(df$Date), df$actual, type = 'n',
axes = FALSE, xlab = "", ylab = '',
     col = rgb(255, 97, 0, 150, maxColorValue = 255),
lwd = 1.5)

points(as.Date(df$Date), df$actual, lwd = 1, type = 'l', col = myColor1)
points(as.Date(df$Date), df$actual, col = myColor1, pch = 19, cex = 0.5)
points(as.Date(df$Date), if_else(df$strategy == "buy", df$actual, NULL),
col = myColor4, pch = 17, cex = 1)

# Legend,

legend(as.Date('2020-06-20'), 100,
c('Original price of IBM', 'Long Position by RNN-based Strategy'),
       border = NA, pch = c(19, 17), col = c(myColor1, myColor4),
cex = 0.8, bty = 'n')

# Axis;
axis(1, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     at = seq(as.Date('2019-02-01'), as.Date('2021-03-01'), length.out = 60),
cex.axis = 0.8,
```

```
     lwd.ticks = 0.15, tck = 0.015, family = 'Lato Light',
     labels = format(seq(as.Date('2019-02-01'),
as.Date('2021-03-01'), length.out = 60), '%b\n%Y'))

axis(2, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     lwd.ticks = 0.15, cex.axis = 0.8,
at =  round(seq(min_value, max_value-5, length.out = 5), 0))

# mtext;
mtext('The original price of IBM and Long positions by RNN-based Strategy',
      3, line = 1.6, adj = 0.05, cex = 1.09,
family = 'Lato Black', outer = TRUE)
mtext('The Strategy has been placed long only, short-sell is restricted',
      3, line = -0.0, adj = 0.06, cex = 0.85,
family = 'Lato Light', outer = TRUE, font = 3)

dev.off()
```

Code: Visualization Profit or Loss of RNN Model

```
cat('\f')
rm(list = ls())
library(dplyr)
library(tidyverse)
library(lubridate)
library(grDevices)
df <- readxl::read_xlsx('thesis_vis/ibm_rnn_strategy.xlsx')
df <- df[-1, ]

# Visual here;
pdf_file <- 'thesis_vis/ibm_rnn_strategy.pdf'
cairo_pdf(bg = 'grey98', pdf_file, width = 9.6, height = 5.8)
par(
  omi = c(0.25, 0.5, 0.75, 0.6),
  mai = c(0.25, 0.5, 0.05, 0.8),
  family = 'Lato Light',
  las = 1,
  mgp = c(1, 1, 0)
)
par(cex = 0.85, bg = 'white')

myColor1 <- 'brown'
myColor2 <- 'blue'
myColor3 <- 'red'
myColor4 <- 'green'

min_value <- min(min(df$accumulated_buy_hold), min(df$acc_strategy_return))
max_value <- max(max(df$accumulated_buy_hold), max(df$acc_strategy_return))
plot(as.Date(df$Date), df$acc_buyhold_return, type = 'n',
axes = FALSE, xlab = "", ylab = '',
     col = rgb(255, 97, 0, 150, maxColorValue = 255), lwd = 1.5,
ylim = c(min_value+0.01, max_value-0.01))

points(as.Date(df$Date), df$acc_buyhold_return,
col = myColor1, lwd = 1, type = 'l')
points(as.Date(df$Date), df$acc_buyhold_return,
col = myColor1, pch = 19, cex = 0.6)
points(as.Date(df$Date), df$acc_strategy_return,
col = myColor2, lwd = 1, type = 'l')
points(as.Date(df$Date), df$acc_strategy_return,
col = myColor2, pch = 19, cex = 0.6)


# Legend;
legend(as.Date('2020-06-20'), -0.135,
c('Accumulated return of buy and hold Strategy',
'Accumulated return of RNN-based Strategy'),
       border = NA, pch = 19, col = c(myColor1, myColor2),
cex = 0.8, bty = 'n')

# mtext;
mtext('Accumulated Return of IBM stock with buy and hold and RNN-based Strategy',
```

```
      3, line = 1.6, adj = 0.05,
cex = 1.09, family = 'Lato Black', outer = TRUE)
mtext('Both Strategies were validated from 02/2019 to 03/2021',
      3, line = -0.0, adj = 0.06,
cex = 0.85, family = 'Lato Light', outer = TRUE, font = 3)

# Axis;
axis(1, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     at = seq(as.Date('2019-02-01'), as.Date('2021-03-01'), length.out = 60),
cex.axis = 0.8,
     lwd.ticks = 0.15, tck = 0.015, family = 'Lato Light',
     labels = format(seq(as.Date('2019-02-01'),
as.Date('2021-03-01'), length.out = 60), '%b\n%Y'))

axis(2, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     lwd.ticks = 0.15, cex.axis = 0.8,
at =  round(seq(min_value, max_value-0.05, length.out = 5), 2),
     labels =
paste0(round(seq(min_value, max_value-0.05, length.out = 5), 2)*100, '%'))
dev.off()
```

Code: Visualization Long Short Positions of SVM Model

```
cat('\f')
rm(list = ls())
library(dplyr)
library(tidyverse)
library(lubridate)
library(grDevices)

df <- readxl::read_xlsx('thesis_vis/ibm_svc_strategy.xlsx')
df <- df[-1, ]


# Visual here;
pdf_file <- 'thesis_vis/ibm_svm_place_position.pdf'
cairo_pdf(bg = 'grey98', pdf_file, width = 9.6, height = 5.8)
par(
  omi = c(0.25, 0.5, 0.75, 0.6),
  mai = c(0.25, 0.5, 0.05, 0.8),
  family = 'Lato Light',
  las = 1,
  mgp = c(1, 1, 0)
)
par(cex = 0.85, bg = 'white')

myColor1 <- 'brown'
myColor2 <- 'blue'
myColor3 <- 'red'
myColor4 <- 'green'
min_value <- min(df$actual)
max_value <- max(df$actual)

plot(as.Date(df$Date), df$actual, type = 'n',
 axes = FALSE, xlab = "", ylab = '',
     col = rgb(255, 97, 0, 150, maxColorValue = 255),
lwd = 1.5)

points(as.Date(df$Date), df$actual, lwd = 1, type = 'l',
col = myColor1)
points(as.Date(df$Date), df$actual, col = myColor1,
pch = 19, cex = 0.5)
points(as.Date(df$Date), if_else(df$strategy == "buy", df$actual, NULL),
col = myColor4, pch = 17, cex = 1)

# Legend,
legend(as.Date('2020-06-20'), 100,
c('Original price of IBM', 'Long Position by SVM-based Strategy'),
       border = NA, pch = c(19, 17), col = c(myColor1, myColor4),
cex = 0.8, bty = 'n')

# Axis;
axis(1, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     at = seq(as.Date('2019-02-01'), as.Date('2021-03-01'), length.out = 60),
cex.axis = 0.8,
```

```
     lwd.ticks = 0.15, tck = 0.015, family = 'Lato Light',
     labels =
format(seq(as.Date('2019-02-01'), as.Date('2021-03-01'), length.out = 60), '%b\n%Y'))

axis(2, col = rgb(105, 105, 105, maxColorValue = 255),
col.ticks = rgb(105, 105, 105, maxColorValue = 255),
     lwd.ticks = 0.15, cex.axis = 0.8,
at =  round(seq(min_value, max_value-5, length.out = 5), 0))
# mtext;
mtext('The original price of IBM and Long positions by SVM-based Strategy',
      3, line = 2.6, adj = 0.05, cex = 1.09,
family = 'Lato Black', outer = TRUE)
mtext('The Strategy has placed long only, short-sell is restricted',
      3, line = 1, adj = 0.06, cex = 0.85,
family = 'Lato Light', outer = TRUE, font = 3)
dev.off()
```

REFERENCES

[1] Aurélien, G. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow.* Sebastopol, Canada: O'Reilly Media.

[2] Babaei G, Giudici P, Raffinetti E. Safeaipackage: a python package for AI risk measurement. *SSRN; 2024*

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

[4] Chollet, F. (2018). *Deep Learning with Python.* Manning Publications.

[5] European Commission. Proposal for a regulation of the European parliament and of the coun-cil laying down harmonized rules on artificial intelligence (Artificial Intelligence Act) andamending certain (Union Legislative Acts). Brussels: European Commission; 2021 April 21.

[6] 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),* 654-669.

[7] 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)*, 654-669.

[8] Giudici P, Raffinetti E. SAFE artificial intelligence in finance. Finance Res Lett. 2023; 56:104088. doi: 10.1016/j.frl.2023.104088

[9] Giudici, P., Centurelli, M., and Turchetta, S. (2024). Artificial intelligence risk measurement. *Expert Systems with Applications, 235*, Article 121220. http://dx.doi.org/10.1016/j.eswa.2023.121220.

[10] Giudici, P., and Raffinetti, E. (2023). SAFE artificial intelligence in finance. *Finance Research Letters, 13,* Article, 104088. https://doi.org/10.1016/j.eswa.2020.114104

[11] Giudici, P., and Raffinetti, E. (2024). RGA: a unified approach of predictive accuracy. *Advances in Data Analysis and Applications*, http://dx.doi.org/10.1007/s11634-023-00574-2, (in press)

[12] Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep Learning. *MIT Press.*

[13] Guoqiang, Z., Patuwo, B.E., and Michael, Y.H. (1998). Citation: Forecasting with artificial neural networks: the state of the art. *International journal of forecasting, 14(1),* 35-62.

[14] Heaton, J. B., Polson, N. G., and Witte, J. H. (2017). Deep learning for finance: deep portfolios. *Applied Stochastic Models in Business and Industry, 33(1)*, 3-12.

[15] Hiransha, M., Gopalakrishnan, E. A., Menon, V. K., and Soman, K. P. (2018) NSE stock market prediction using deep-learning models. *Procedia Computer Science, 132*, 1351-1362.

[16] Hu, Y., Liu, X., and Zhang, L. (2020). Application of deep learning in financial markets: A review. *Journal of Finance and Data Science, 6(2)* 107-127.

[17] Jonh, S.T., Peter, L.B., Robert, C.W., and Martin, A. (1996). Citation: Structural Risk Minimization over Data-Dependent Hierarchies. *Neural and Computer Learning, 8556,* 18-26.

[18] Julian, F., Chris, C. (1998). Citation: Time series forecasting with neural networks: a comparative study using the air line data. *Journal of the Royal Statistical Society: Series C (Applied Statistics), 47(2),* 231-250.

[19] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. Nature, 521(7553), *436-444.*

[20] Matthew, F.D., Igor, H., Paul, B. (2020). *Machine Learning in Finance From Theory to Practice.* Cham, Switzerland: Springer Nature Switzerland AG.

[21] McKinney, W. (2010). *Data Analysis with Python.* O'Reilly Media.

[22] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... and Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems, 32,* 8026-8037.

[23] Paul, J.W (1988). Citation: Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks, 1(4),* 339-356.

[24] Raffinetti, E. (2023). A rank graduation accuracy measure to mitigate artificial intellgence risks. *Quality and Quantity,* 57(2), 131–150. http://dx.doi.org/10.1007/s11135-023-01613-y

[25] Raja, V., Maxence, H., Daniel, N. (2020). *Algorithmic Trading and Quantitative Strategies.* Boca Raton, Florida, USA: CRC Press.

[26] Richardson, L. (2015). *Python Web Scraping.* Packt Publishing.

[27] Robert, K. (2021). *Algorithmic Trading Methods: Applications using Advanced Statistics, Optimization, and Machine Learning Techniques* (2nd ed.). Oxford, United Kingdom: Academic Press.

[28] Sebastien, D., Sourav, G. (2019). *Learn Algorithmic Trading.* Birmingham, United Kingdom: Packt Publishing.

[29] Sepp, H., Jurgen, S. (1997). Citation: Long Short-Term Memory. *Neural Computation, 9(8),* 1735-1780.

[30] Shalev, S.S., Ben, D.S. (2014). *Understanding Machine Learning From Theory to Algorithms.* New York, USA: Cambridge University Press.

[31] Van Rossum, G., and Drake, F. L. (2009). Python 3 Reference Manual. *CreateSpace Independent Publishing Platform.*

[32] Yahoo Finance API. (2024). *Yahoo Finance API Documentation.* Retrieved from https://www.yahoofinanceapi.com/

[33] Yarovaya, L. (2019). *Financial Markets and Investment Analysis.* Wiley.

[34] Zaiyoung, T., Chrys, D.A, Paul, A.F. (1991). Citation: Time series forecasting using neural networks vs box-jenkins methodology. *Simulation, 57(5),* 303-310.

[35] Zhang, G.P, Min, Q. (2005). Citation: Neural network forecasting for seasonal and trend time series. *European journal of operational research, 160(2),* 501-514.