Leveraging Machine Learning to analyze and find vulnerabilities in the current financial system

Project-IV (IM57004) report submitted to

Indian Institute of Technology Kharagpur

in partial fulfilment for the award of the degree of

Masters of Technology

in

Industrial and Systems Engineering

by

Tushar Mishra

(18MF3IM35)

Under the supervision of

Professor P K Ray

Industrial and Systems Engineering

Indian Institute of Technology Kharagpur

Spring Semester 2022-23

April 28, 2023

DECLARATION

I certify that

(a) The work contained in this report has been done by me under the guidance of my supervisor.

(b) The work has not been submitted to any other Institute for any degree or diploma.

(c) I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

(d) Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Date: April 28, 2023 (Tushar Mishra)
Place: Kharagpur (18MF3IM35)

INDUSTRIAL AND SYSTEMS ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

KHARAGPUR - 721302, INDIA

CERTIFICATE

This is to certify that the project report entitled "**Leveraging Machine Learning to analyze and find vulnerabilities in the current financial system**" submitted by **Tushar Mishra** (Roll No. 18MF3IM35) to Indian Institute of Technology Kharagpur towards partial fulfilment of requirements for the award of degree of Masters of Technology in Industrial and Systems Engineering is a record of bona fide work carried out by him under my supervision and guidance during Spring Semester 2022-23.

Professor P K Ray
Industrial and Systems Engineering
Indian Institute of Technology Kharagpur
Kharagpur - 721302, India

Date: April 28, 2023
Place: Kharagpur

Abstract

Name of the student: **Tushar Mishra** Roll No: **18MF3IM35**

Degree for which submitted: **Masters of Technology**

Department: **Industrial and Systems Engineering**

Thesis title: **Leveraging Machine Learning to analyze and find vulnerabilities in the current financial system**

Thesis supervisor: **Professor P K Ray**

Month and year of thesis submission: **April 28, 2023**

Cutting edge technological applications in our current financial system is the new normal. With so many advances in high performance computing and artificial intelligence, the exploitation of current financial system using these resources is really only a matter of time. Any technological vulnerabilities present in the current financial system can be exploited to cheat people of their hard earned money. We will investigate the Flash Crash, SWIFT international trade network and 1MDB scandal and use Generative Adversarial Networks (GANs) to predict stock prices, generate scarce datasets, optimize several trading strategies and compare its performance with the traditional time series and LSTM Machine Learning models. We will do a comprehensive literature review of High Frequency Trading and try to understand its effects on the global financial markets. Using high performing GANs, we will try to generate execution logs of a trading algorithm and try to mimic a trading algorithm as closely as possible, back-test them and find technical vulnerabilities.

Acknowledgements

I am greatly indebted to my project supervisor Prof. P K Ray for his invaluable guidance and encouragement throughout this project. I am thankful for his aspiring guidance, invaluable constructive criticism and friendly advice during the project work. I would like to take this opportunity to express my sincere and profound gratitude to them. I would also like to gratefully acknowledge the suggestions and discussions received from my fellow project members and students at IIT Kharagpur whenever I had any doubt. I sincerely thank all the faculty of Industrial and Systems Engineering for their kind cooperation in all phases of my project work. Last but not the least, I thank my parents and all my family members for their support and motivation they have provided throughout my project.

Tushar Mishra
18MF3IM35

Contents

List of Figures

Abbreviations

CNN	**C**onvolutional **N**eural **N**etwork
LSTM	**L**ong **S**hort **T**erm **M**emory
GAN	**G**enerative **A**dversarial **N**etwork
HFT	**H**igh **F**requency **T**rading
ETP	**E**lectronic **T**rading **P**latforms

Chapter 1

Introduction

1.1 Introduction

Machine learning has become an essential aspect of financial services and applications. It has revolutionized the way financial institutions manage their assets, evaluate risk, calculate credit scores, and approve loans. The growing use of machine learning in finance has also made it possible to analyze and process vast amounts of financial data, making it easier to spot patterns and insights that can inform investment decisions.

One of the main benefits of machine learning in finance is its ability to learn and improve from experience without the need for explicit programming. Machine learning algorithms use statistical techniques to identify patterns in data and use those patterns to make predictions or decisions. When large volumes of data are fed into the system, machine learning algorithms tend to be more accurate in drawing insights and making predictions.

The financial services industry is one of the industries that generate vast amounts of data relating to daily transactions, bills, payments, vendors, and customers. This data is perfect for machine learning, which can help financial institutions detect fraudulent transactions, optimize their portfolios, and automate trading activities.

Algorithmic trading is one of the primary fields of application for machine learning in finance. It refers to the use of algorithms to make better trade decisions. Traders build mathematical models that monitor business news and trade activities in real-time to detect any factors that can force security prices to rise or fall. These models come with a predetermined set of instructions on various parameters such as timing, price, quantity, and other factors for placing trades without the trader's active involvement.

Unlike human traders, algorithmic trading can simultaneously analyze large volumes of data and make thousands of trades every day. This ability to make fast trading decisions gives human traders an advantage over the market average. Additionally, algorithmic trading does not make trading decisions based on emotions, which is a common limitation among human traders whose judgment may be affected by emotions or personal aspirations. The trading method is mostly employed by hedge fund managers and financial institutions to automate trading activities.

Another important use case for machine learning in finance is fraud detection and prevention. Most systems today leverage machine learning to flag and combat fraudulent financial transactions. Machine learning works by scanning through large data sets to detect unique activities or anomalies and flags them for further investigation by security teams.

In recent years, deep learning has achieved great success in many areas due to its strong capacity in data processing. It has been widely used in financial areas such as stock market prediction, portfolio optimization, financial information processing, and trade execution strategies. Stock market prediction is one of the most popular and valuable areas in finance. It involves using machine learning algorithms to analyze financial data and predict future stock prices.

Our aim is to find vulnerabilities in the current financial system, specifically in automated trading algorithms. To achieve this, we started by addressing the problem of predicting stock price movement using GAN and comparing the performance with multiple Deep Learning models. Despite the extensive exploration with GAN, we found that the relative performance of the GAN model with respect to traditional

deep learning models such as LSTM has not been assessed.
To develop this task, we trained a baseline ARIMA model, a long short-term memory (LSTM) model, a deep LSTM model, and a generative adversarial network (GAN) model. We collected and preprocessed stock price dataset for multiple years for specific companies. The results showed that the deep LSTM model had the highest accuracy rate in predicting future stock prices.
We then moved on to creating a comprehensive trading infrastructure that is all-encompassing and can hold multiple strategies and portfolio calls. This versatile algorithm ensures portfolio diversification and closely resembles the Mid and High frequency trading algorithms in production actually deployed by trading companies and hedge funds.
To test the effectiveness of the algorithm, we created logs for the execution duration of the code and analyzed the possible problems that it can face during a long uninterrupted run, performance on edge cases (Black Swans in finance).

1.2 Literature Review

Financial vulnerability refers to a state of financial instability or a situation that is exposed to financial risks and shocks. It can be caused by various factors such as market volatility, cyber-attacks, fraudulent activities, or even natural disasters. In this context, the vulnerability of the financial system can have serious implications for individuals, organizations, and entire economies.

Over the years, there have been several instances where the financial system has been exploited, leading to significant losses. The three most notable examples of such exploitation in the 21st century are the Flash Crash of 2010, the Bangladesh Central Bank heist in 2016, and the Malaysian state-sponsored 1MDB scandal based on the idea of National Sovereign funds.

The Flash Crash of 2010 was a unique event that happened on May 6th, 2010, in the United States. The US equity markets, comprising of approximately 8,000 tickers, literally collapsed within a few minutes, only to recover by the end of the day. During this short period, the markets were down by 5-15 percent, and some of them lost 60 percent of their value. The markets recovered most of their losses by the end of the day, but still closed 3 percent down from the previous day. The entire market lost about 1 trillion dollars due to the chaotic nature of volatility and the disappearance of liquidity that made limit order books (LOBs) of highly correlated equities extremely unstable and dysfunctional, ultimately leading to the collapse of the system. This event generated serious concerns and highlighted the weaknesses of financial markets and some of the regulations in place.

The Bangladesh Central Bank heist, which occurred in February 2016, was one of the largest cyber heists in history. The Central Bank of Bangladesh (Bangladesh Bank) lost 81 million dollars from its account held in the Federal Reserve Bank of New York. The hackers attempted to steal a whopping 951 million dollars in a well-planned sophisticated attack. They compromised the bank's network and navigated the Society for Worldwide Interbank Financial Telecommunication (SWIFT) gateway to send 35 fraudulent fund transfer requests on behalf of the bank. The SWIFT network is a consortium that operates a trusted and closed computer network for communication between member banks around the world. The Bangladesh Central Bank heist was a clear example of how the cyber-attackers can leverage the vulnerabilities of the financial system for their benefit.

The Malaysian state-sponsored 1MDB scandal based on the idea of National Sovereign funds was yet another example of financial exploitation. Originally, 1MDB was a Malaysian state fund set up in 2009 to promote development through foreign investment and partnerships. However, the fund has since been at the heart of one of the biggest corruption scandals in the world, with more than 4.5 billion dollars being stolen in the resulting scandal. Vast sums were borrowed via government bonds and siphoned into bank accounts, and huge amounts were embezzled. The scandal highlighted the flaws and loopholes in the financial system that can be exploited for personal gains.

Apart from these examples, there are other fraudulent activities that occur in the financial world, such as boiler room scams and front running. Boiler room scams, also known as “pump and dump schemes,” involve artificially inflating the price of a stock that fraudsters own by encouraging others to buy shares. The fraudsters then sell or “dump” their overvalued shares, causing investors to lose money as the price falls. Fraudsters can use disinformation campaigns to perform this type of activity. For high-frequency traders (HFT), compromising and taking over brokerage accounts is one method through which malicious actors could attempt their boiler room scams.

In the financial industry, time series prediction, such as stock price prediction, is a challenging task due to the complexity of the data and unknown inner workings. Many classic algorithms like Long Short-Term Memory (LSTM) and ARIMA are used for time series prediction. However, Generative Adversarial Network (GAN), a powerful Deep Learning model, has not been widely used in this field. GAN is known for its success in image generation and is now being applied to generate time series and other types of financial data. This model has shown considerable progress in finance applications and can serve as an additional tool for data scientists in this field.

Chapter 2

Machine Learning: Theory and Modelling

2.1 Machine Learning Models

2.1.1 Neural Networks

Neural networks are a type of machine learning algorithm inspired by the structure of the human brain. They consist of layers of interconnected nodes, or neurons, that process and transmit information. Neural networks are trained on a set of data, learning to recognize patterns and make predictions by adjusting the strength of connections between neurons. They have been successfully applied to a wide range of tasks, including image and speech recognition, natural language processing, and predictive modeling. The power of neural networks lies in their ability to automatically extract relevant features from raw data, enabling them to make accurate predictions even in the face of complex, nonlinear relationships. However, training a neural network can be a computationally intensive process, and requires careful tuning of hyperparameters to achieve optimal performance.

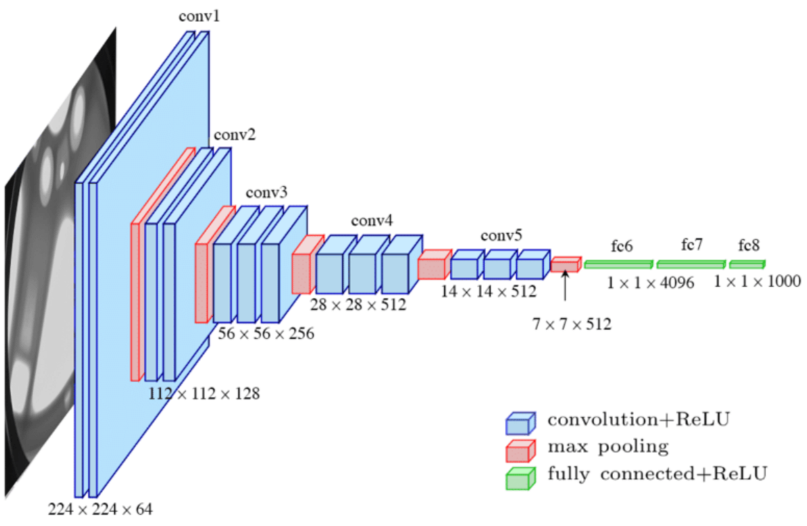


Figure 2.1: CNN Architecture

2.1.2 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNNs) are a type of neural network commonly used in image recognition and computer vision tasks. Unlike traditional neural networks, which process inputs as a single vector, CNNs take inputs as 2D matrices. They use a combination of convolutional layers, pooling layers, and fully connected layers to learn features from images and classify them accurately. In convolutional layers, filters are applied to the input image to extract specific features, and in pooling layers, the output is downsampled to reduce the number of parameters in the model. Fully connected layers then use these features to make predictions. The ability of CNNs to automatically learn and extract features from images makes them a powerful tool in a variety of applications, including object recognition, facial recognition, and medical imaging.

2.1.3 Long Short Term Memory (LSTM)

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) that is designed to handle the problem of vanishing gradients in traditional RNNs.

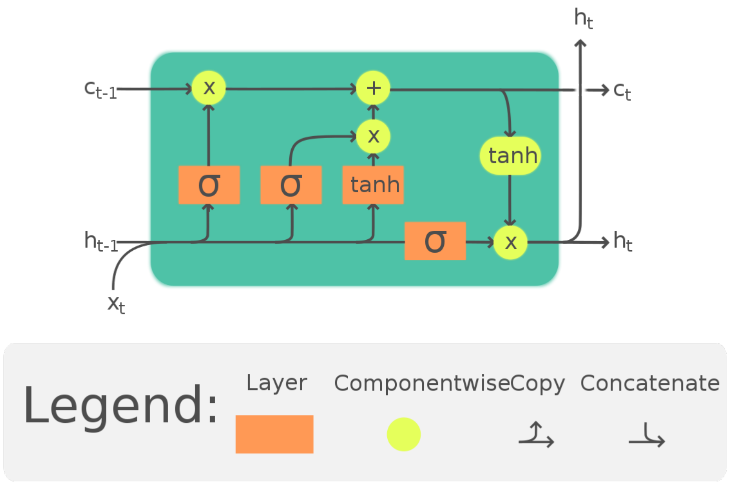


FIGURE 2.2: LSTM Architecture

LSTMs are capable of learning long-term dependencies by selectively retaining or forgetting information at each time step through a series of gates, including an input gate, an output gate, and a forget gate. These gates allow the network to selectively read, write, and forget information from previous time steps, which enables the network to learn long-term patterns in sequential data such as time series or natural language. LSTMs have found wide applications in various fields such as speech recognition, language modeling, and video analysis.

2.1.4 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are a type of deep learning model composed of two neural networks: a generator and a discriminator. The generator produces new data instances that are similar to the training data, while the discriminator evaluates the authenticity of these instances by distinguishing between real and fake data. The two networks are trained together in a competitive process where the generator tries to produce increasingly convincing fake data, while the discriminator tries to become better at detecting it. This adversarial process results in the generator producing high-quality data that is indistinguishable from the real

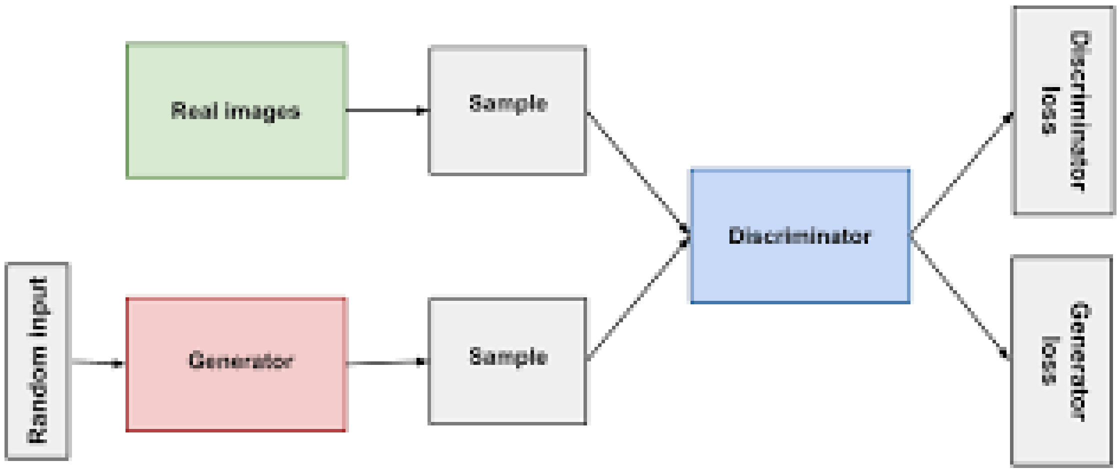


FIGURE 2.3: GAN Architecture

data. GANs have become popular in image generation, but they can also be applied to other types of data, such as time series and financial data.

2.2 Evaluation Metrics

In machine learning, evaluation metrics are used to measure the performance of a model. Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE) are three commonly used evaluation metrics. We have used all three of them to compare and evaluate the performance of our Machine Learning models.

2.2.1 Mean Squared Error (MSE)

MSE is the average squared difference between the predicted values and the true values. It measures the average squared distance between the predicted and actual values, with higher values indicating poorer model performance.

2.2.2 Root Mean Squared Error (RMSE)

RMSE is the square root of the MSE, which gives the same unit of measurement as the original data. It is a popular metric for regression tasks, as it penalizes larger errors more heavily and provides a better understanding of the magnitude of the errors.

2.2.3 Mean Absolute Error (MAE)

MAE is the average absolute difference between the predicted values and the true values. It measures the average distance between the predicted and actual values, regardless of the direction of the difference. MAE is less sensitive to outliers than MSE and RMSE, and is often used when the outliers are important to the analysis.

Chapter 3

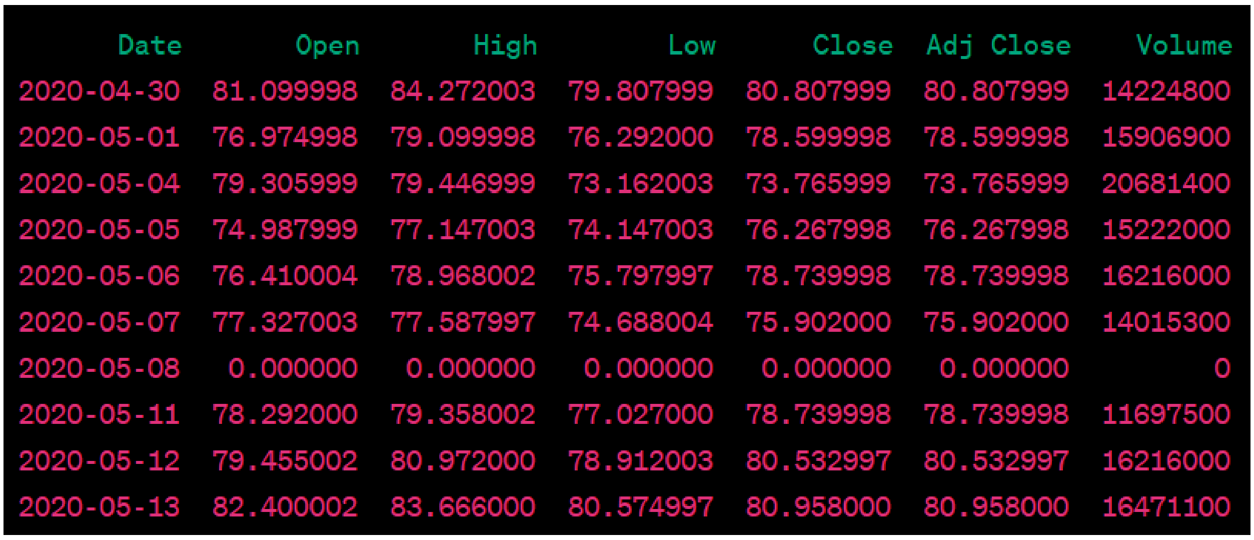
Generative Adversarial Networks: Performance and Comparisons

3.1 Methodology

Here, Our objective is to use multiple Machine Learning models including Linear Regression, ARIMA, LSTM and GANs to predict the stock prices for TESLA company based on historical data. Then, we will do comprehensive comparison and performance evaluation using the three metrics MSE, RMSE and MAE The dataset used in this project is the stock prices for Tesla Inc. (TSLA) from January 2010 to April 2023, obtained from Yahoo Finance.The accompanying images describe the dataset used and the features created in detail.

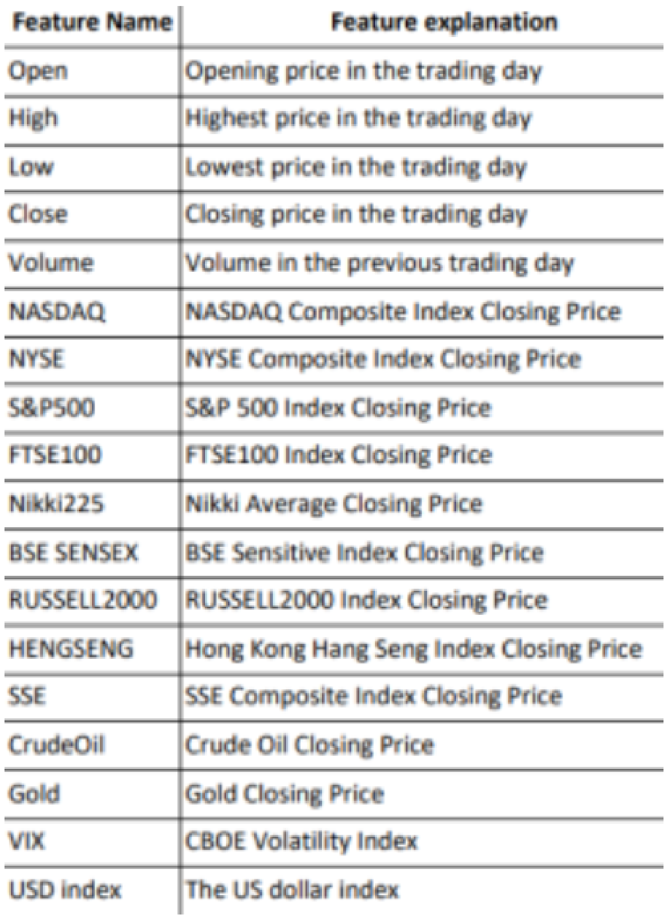
3.1.1 Feature Engineering

The RMSE values are low because the data has been standardized before being used for training the models. Standardization is a common preprocessing step in machine learning where the data is transformed to have zero mean and unit variance. This



```
       Date       Open       High        Low      Close  Adj Close    Volume
2020-04-30  81.099998  84.272003  79.807999  80.807999  80.807999  14224800
2020-05-01  76.974998  79.099998  76.292000  78.599998  78.599998  15906900
2020-05-04  79.305999  79.446999  73.162003  73.765999  73.765999  20681400
2020-05-05  74.987999  77.147003  74.147003  76.267998  76.267998  15222000
2020-05-06  76.410004  78.968002  75.797997  78.739998  78.739998  16216000
2020-05-07  77.327003  77.587997  74.688004  75.902000  75.902000  14015300
2020-05-08   0.000000   0.000000   0.000000   0.000000   0.000000         0
2020-05-11  78.292000  79.358002  77.027000  78.739998  78.739998  11697500
2020-05-12  79.455002  80.972000  78.912003  80.532997  80.532997  16216000
2020-05-13  82.400002  83.666000  80.574997  80.958000  80.958000  16471100
```

FIGURE 3.1: TESLA Stock Price Dataset for Opening, Closing and Trading Prices



Feature Name	Feature explanation
Open	Opening price in the trading day
High	Highest price in the trading day
Low	Lowest price in the trading day
Close	Closing price in the trading day
Volume	Volume in the previous trading day
NASDAQ	NASDAQ Composite Index Closing Price
NYSE	NYSE Composite Index Closing Price
S&P500	S&P 500 Index Closing Price
FTSE100	FTSE100 Index Closing Price
Nikki225	Nikki Average Closing Price
BSE SENSEX	BSE Sensitive Index Closing Price
RUSSELL2000	RUSSELL2000 Index Closing Price
HENGSENG	Hong Kong Hang Seng Index Closing Price
SSE	SSE Composite Index Closing Price
CrudeOil	Crude Oil Closing Price
Gold	Gold Closing Price
VIX	CBOE Volatility Index
USD index	The US dollar index

FIGURE 3.2: Additional Features and Data Points

ensures that the data is in a similar range and is not dominated by a single feature with a large value range.

In this particular case, the data has been standardized using the StandardScaler class from scikit-learn library in Python. The StandardScaler scales the data so that it has a mean of 0 and a standard deviation of 1. This helps the models to learn the patterns in the data more effectively and results in lower RMSE values.

It's worth noting that the RMSE values can vary depending on the specific dataset, features used, and the modeling techniques employed. It's important to evaluate the performance of the models based on multiple metrics and to compare them with appropriate baseline models.

3.2 Performance Comparison

The Architecture and Codes for Linear Regression, ARIMA, LSTM and GANs are attached in the Appendix.

The GAN model consists of two deep neural networks: a generator and a discriminator. The generator takes random noise as input and generates fake stock prices, while the discriminator takes both real and fake stock prices as input and distinguishes between them. The two networks are trained simultaneously, with the generator trying to fool the discriminator and the discriminator trying to distinguish between the real and fake prices. The following is the architecture of the GAN model used in the project:

GAN models have shown promising results in generating realistic and diverse synthetic data samples in various domains, including image, video, and text generation. GANs can be used for time-series prediction tasks by generating realistic future samples that can be used as predictions.

Loss function of Descriminator:

$$-\frac{1}{m}\sum_{i=1}^{m} log\, D(y^i) \;-\; \frac{1}{m}\sum_{i=1}^{m}(1 - log\, D(G(x^i)))$$

Loss function of Generator:

$$-\frac{1}{m}\sum_{i=1}^{m}(\, log\, D(G(x^i)))$$

Figure 3.3: Loss Functions for GAN

Compared to traditional time-series models like ARIMA and Linear Regression, GANs can learn the complex nonlinear relationships between the inputs and outputs without making assumptions about the data distribution. This allows the GAN to generate more accurate and diverse predictions compared to ARIMA and Linear Regression, which can be limited by their assumptions and linearity.

LSTM models are a type of neural network that can also learn complex nonlinear relationships in time-series data. However, GANs have an advantage over LSTM models in that they can generate more diverse and realistic predictions by incorporating noise into the generator's input. In contrast, LSTM models rely solely on the past history of the time-series data to make predictions, which can limit their ability to capture diverse future scenarios.

Looking at the final table of results, we can observe that GAN model outperformed the other three models, LSTM, ARIMA and Linear Regression in terms of RMSE and MAE. GAN model achieved the lowest RMSE and MAE values for TESLA stock price prediction. This indicates that GAN model was able to better capture the patterns and trends in the stock price data and generate more accurate predictions compared to the other models.

LSTM and ARIMA models also performed well with relatively low RMSE and MAE values, but they were outperformed by the GAN model. Linear Regression model performed the worst among the four models, with significantly higher RMSE and

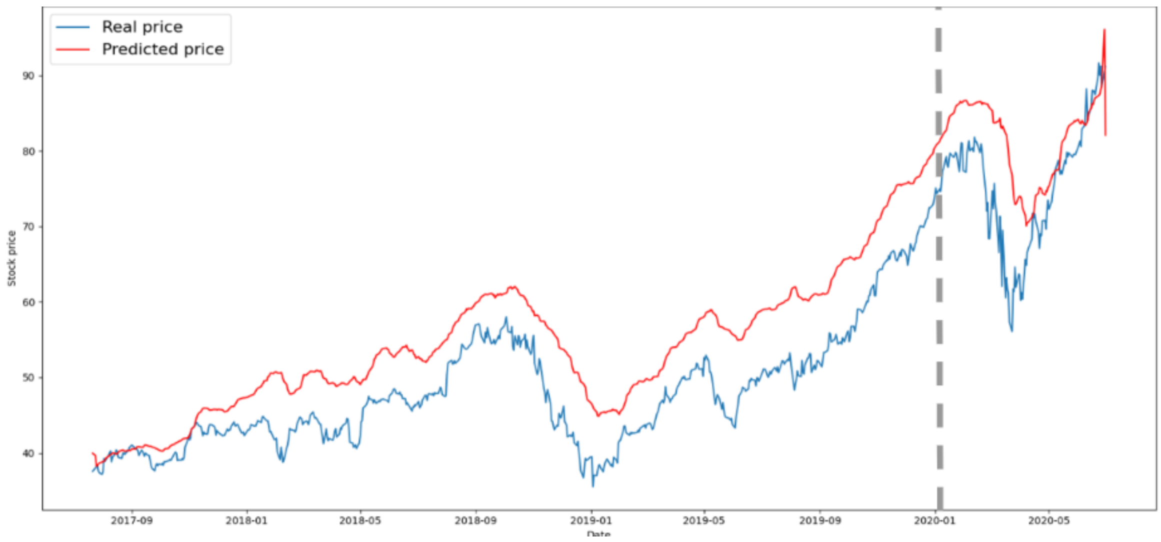


Figure 3.4: Forecasting Results: Baseline LSTM

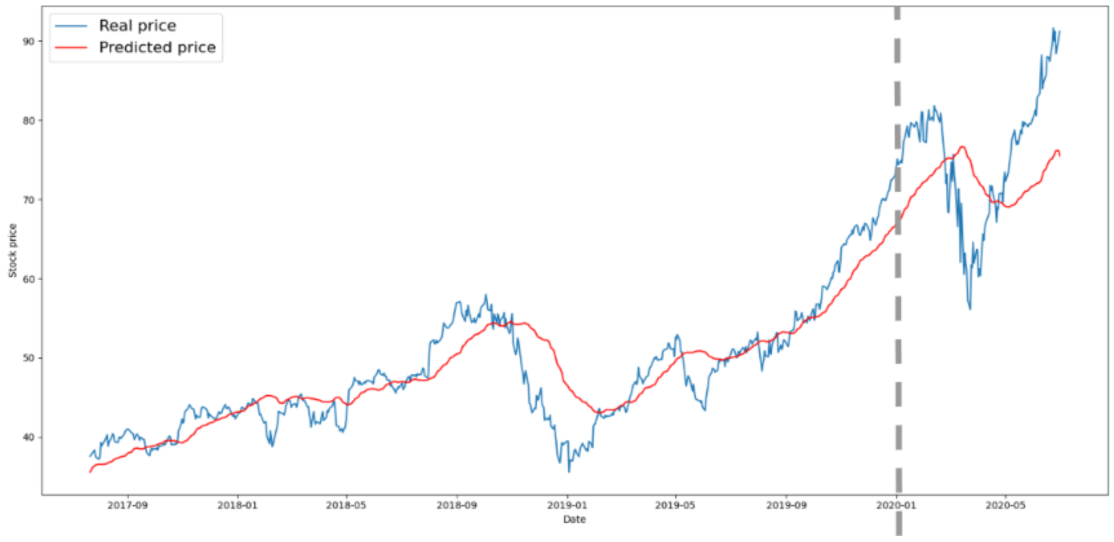
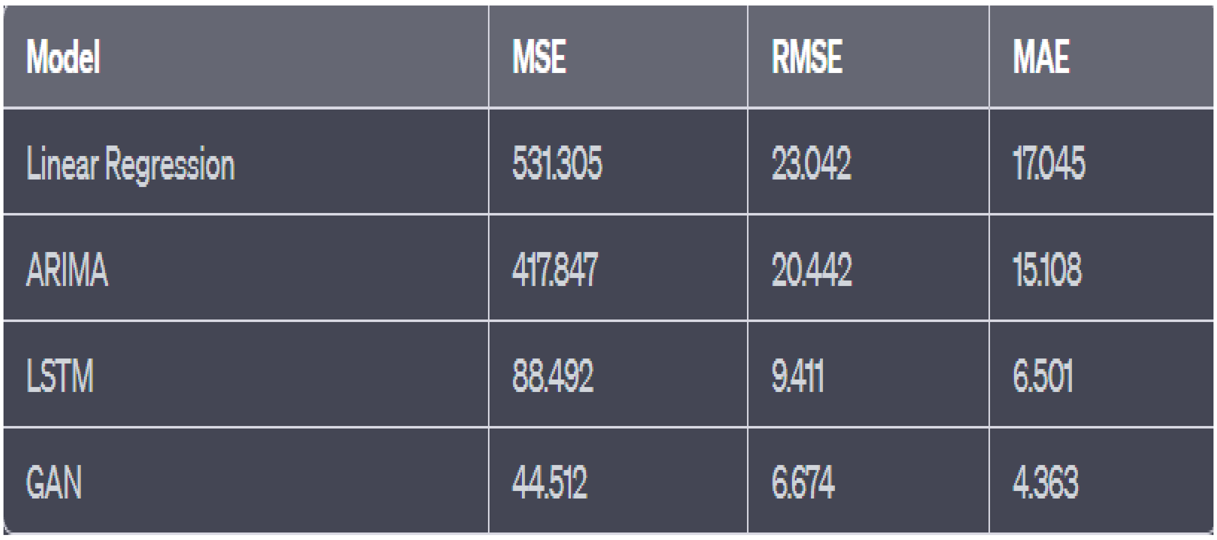


Figure 3.5: Forecasting Results: GAN

MAE values. This indicates that Linear Regression model was not able to effectively capture the complex patterns in the stock price data, and hence resulted in higher prediction errors.

In summary, GAN model proved to be the most effective in predicting the TESLA stock prices, followed by LSTM and ARIMA, while Linear Regression model performed the worst.



Model	MSE	RMSE	MAE
Linear Regression	531.305	23.042	17.045
ARIMA	417.847	20.442	15.108
LSTM	88.492	9.411	6.501
GAN	44.512	6.674	4.363

FIGURE 3.6: Performance Comparison: Stock Price Prediction

Chapter 4

Algorithmic Trading

4.1 Introduction to Algorithmic Trading

Algorithmic trading (AT) is a concept widely explored in recent market microstructure literature. It entails the use of fully automated computer programs that are specifically designed for data analysis, decision-making, and execution, thereby eliminating human involvement in the trading process. The algorithms continuously monitor the market for sudden arbitrage opportunities and react immediately by submitting orders, making AT more efficient than its human counterparts. This efficiency can be attributed to two primary advantages of algorithms. First, AT can assimilate large numbers of signals simultaneously, reducing the uncertainty about the current state of the market. Second, technological advances in AT have improved processing efficiency, leading to an extremely accelerated trading process. As a result, traders can now process vast amounts of information, and relative speeds between traders have become crucial for profitable trading. As a consequence of these developments, latency has become a crucial concept in AT, referring to the time taken to learn about an event, generate a response, and have the exchange act on the response. Therefore, AT's use of algorithms has greatly improved market efficiency by reducing monitoring frictions and providing a quicker response to market signals.

4.1.1 High Frequency Trading

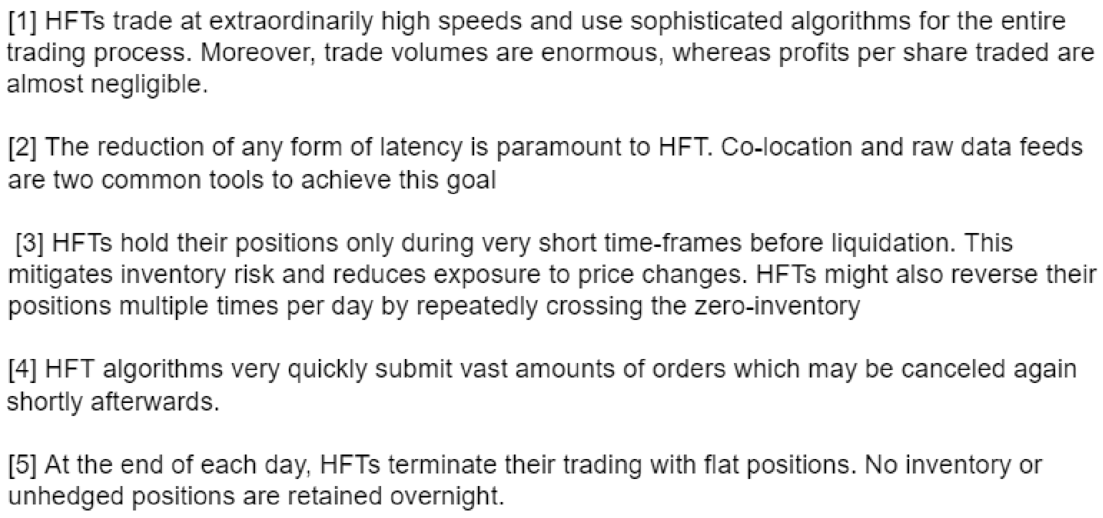
High frequency trading (HFT) is a type of trading that uses complex algorithms and advanced computer technology to buy and sell securities at high speeds and volumes. HFT firms use computer programs to analyze market data and identify profitable trading opportunities in real-time. These programs can execute trades in microseconds, making it possible to profit from even the smallest market fluctuations.

HFT has become a major force in financial markets over the past decade, accounting for a significant portion of trading volume across a range of asset classes. The benefits of HFT include increased market liquidity and improved price discovery, as well as lower transaction costs for investors.

However, HFT has also been criticized for creating market volatility and exacerbating market crashes, as well as for giving an unfair advantage to large HFT firms with the fastest and most powerful trading systems. Regulatory authorities have responded to these concerns with new rules and regulations aimed at ensuring a level playing field for all market participants. The Securities and Exchange Commission (SEC) provided the first quasi-official definition of HFT, which is widely accepted among researchers and practitioners.

4.2 Electronic Trading Platforms and HFTs

Electronic trading platforms refer to computer-based systems that allow traders to buy and sell financial products and securities over an electronic network. These


[1] HFTs trade at extraordinarily high speeds and use sophisticated algorithms for the entire trading process. Moreover, trade volumes are enormous, whereas profits per share traded are almost negligible.

[2] The reduction of any form of latency is paramount to HFT. Co-location and raw data feeds are two common tools to achieve this goal

[3] HFTs hold their positions only during very short time-frames before liquidation. This mitigates inventory risk and reduces exposure to price changes. HFTs might also reverse their positions multiple times per day by repeatedly crossing the zero-inventory

[4] HFT algorithms very quickly submit vast amounts of orders which may be canceled again shortly afterwards.

[5] At the end of each day, HFTs terminate their trading with flat positions. No inventory or unhedged positions are retained overnight.

Figure 4.1: Characteristics of HFT

platforms facilitate automated trading, real-time monitoring of market conditions, and the execution of trades with lightning-fast speed, efficiency, and accuracy.

Electronic trading platforms have revolutionized the financial industry by significantly reducing the cost of trading, increasing liquidity, and improving market efficiency. These platforms have also facilitated the development of new financial instruments and the globalization of financial markets.

Electronic trading platforms are designed to cater to a wide range of traders, from individual investors to institutional investors and large financial firms. They offer a variety of features and tools that enable traders to analyze market trends, monitor price movements, and execute trades with ease and speed.

Examples of popular electronic trading platforms include E-Trade, TD Ameritrade, Interactive Brokers, and Charles Schwab. These platforms offer traders access to a wide range of financial products, including stocks, bonds, options, futures, and foreign currencies.

4.3 Latency Optimisation

Latency optimization refers to the process of minimizing the time delay between sending and receiving data on electronic trading platforms. In electronic trading, every millisecond counts as it can make a difference in the profitability of a trade. Latency optimization can be achieved through various means such as using high-speed networks, locating servers in proximity to the exchange, and utilizing specialized hardware and software.

In the context of high frequency trading, latency optimization is critical as it enables traders to execute trades faster than their competitors, resulting in a significant advantage in the market. High frequency trading strategies rely on the ability to quickly analyze and respond to market data in order to execute trades at lightning-fast speeds. This requires the use of advanced technology and algorithms to reduce latency to the absolute minimum.

To achieve latency optimization in electronic trading platforms and high frequency trading, traders and firms must invest in the latest technology, such as high-speed networks, low-latency hardware and software, and proximity hosting services. They must also continuously monitor and analyze their trading systems to identify areas for improvement and adapt to changes in the market. Overall, latency optimization is crucial for success in electronic trading and high frequency trading, as even the slightest delay can result in missed opportunities and lost profits.

4.4 Latency Arbitrage

The figure "Mechanics of Latency Arbitrage" illustrates the process of how HFTs use latency arbitrage to exploit price differences between different ETPs. To explain this process, let's suppose that an HFT is closely watching the quotes on ETP A. The HFT's co-located servers receive a direct feed that informs them of any quote

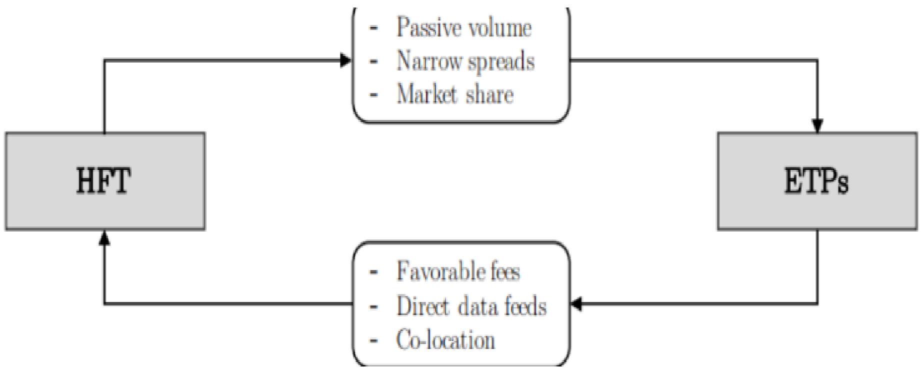


FIGURE 4.2: Mutual dependency and Symbiosis between ETPs and HFTs

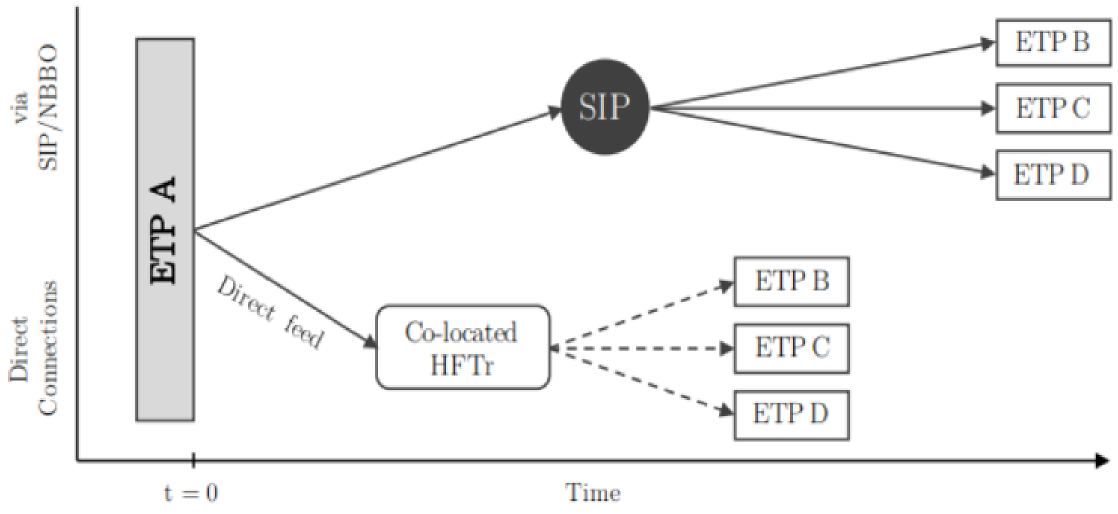


FIGURE 4.3: Mechanics of Latency Arbitrage

changes, which the SIP also receives, but with a delay. During this lag time, the HFT evaluates if the quote adjustment can be exploited on other venues, and if so, sends orders to those platforms, trading in the direction of the incoming change. By the time the rerouted SIP-signal of the new National Best Bid and Offer (NBBO) reaches other markets, the HFT has already taken advantage of the profit opportunity. This effect can accumulate over time, resulting in gains from latency arbitrage, which, though small per trade, can be significant in the long run. Therefore, ETPs offer three types of co-location services, differing in proximity to the servers and subscription fees, which are eagerly subscribed to by some traders, particularly market makers, who are willing to pay a large sum of money to minimize latency relative to other traders. This is akin to a bullet flying at supersonic speeds and hitting its target unexpectedly without the gunshot being heard until it's too late.

Chapter 5

Trading Infrastructure

5.1 Trading Algorithm Framework

Automated pre-programmed trading instructions are utilized in algorithmic trading to execute orders based on variables like time, price, and volume. This approach to trading aims to take advantage of computers' speed and computational capabilities over human traders. A trading algorithm consists of three distinct components.

5.1.1 Predictor

Predictor is the Data Merchant portion of the code. Live feed data is fed to the predictor portion of the code and after exploratory analysis and processing, it returns

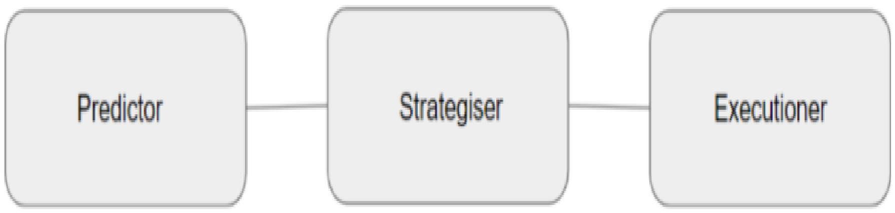
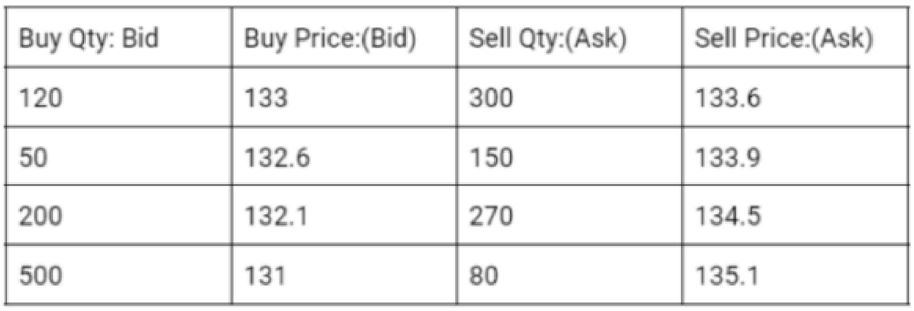


Figure 5.1: Framework of a Trading Algorithm



Buy Qty: Bid	Buy Price:(Bid)	Sell Qty:(Ask)	Sell Price:(Ask)
120	133	300	133.6
50	132.6	150	133.9
200	132.1	270	134.5
500	131	80	135.1

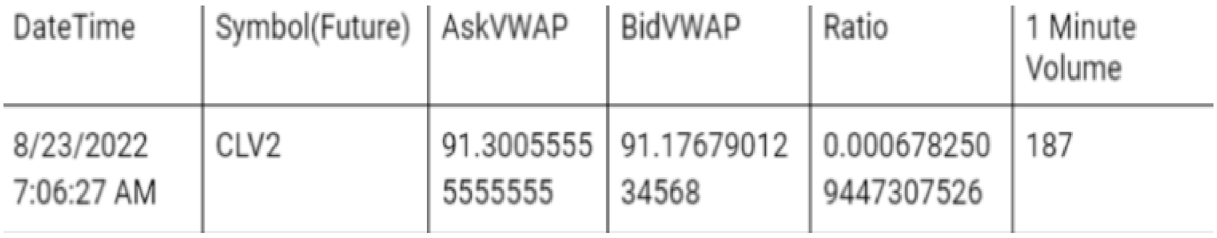
FIGURE 5.2: Order Books

the required data points to the strategiser portion of the code. Here, the live data fed to the Predictor was a basic Order Book of options having 20 levels every 200 ms. The data is fed directly from the stock exchange through the services of a company via an Market Data Manager API. An electronic inventory of buy and sell orders for a security or instrument arranged by price level is called an order book. This organized record of orders enhances market transparency by offering insights into price, availability, trade volume, and the identities of those initiating transactions.

The tables are sorted in decreasing order of Bid Price and ascending order of Ask Price. In order to maximize profit, we will buy from one with lowest ask price and sell to the one with highest bid price.

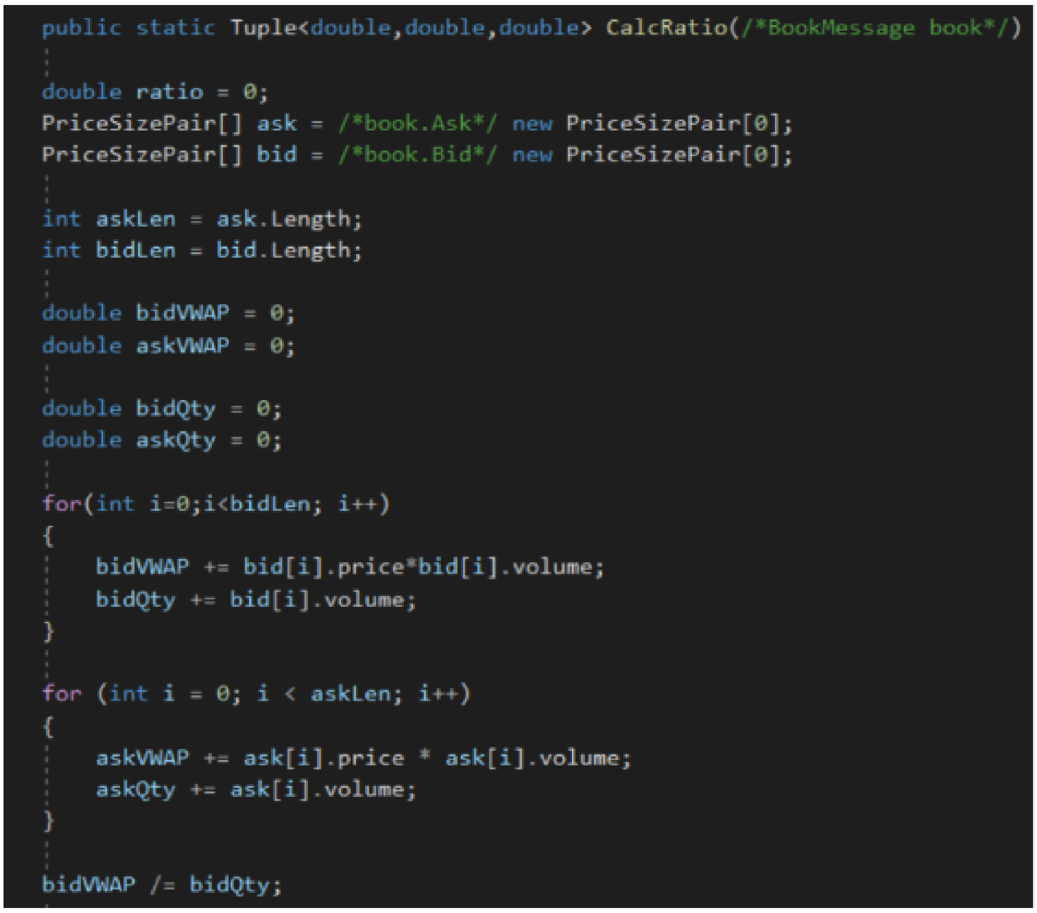
5.1.2 Strategiser

This portion of the code is the brains of the operation. This portion can contain strategies ranging from basic mathematical alphas to complex financial models and State of the art Machine Learning and Deep Learning algorithms. This portion makes sense of the Data provided to it, lets it run through the strategy and then eventually generates a Long/ Short/ Hold signal and sends it to the Executioner part of the code. We need historical data for our futures and options that we created ourselves: We logged data and stored the values for the last 1000 minutes.
VWAP: Volume Weighted Average Price
VWAP Prices are calculated using order book os stocks



DateTime	Symbol(Future)	AskVWAP	BidVWAP	Ratio	1 Minute Volume
8/23/2022 7:06:27 AM	CLV2	91.3005555 5555555	91.17679012 34568	0.000678250 9447307526	187

FIGURE 5.3: Historical Dataset of last 1000 ticks

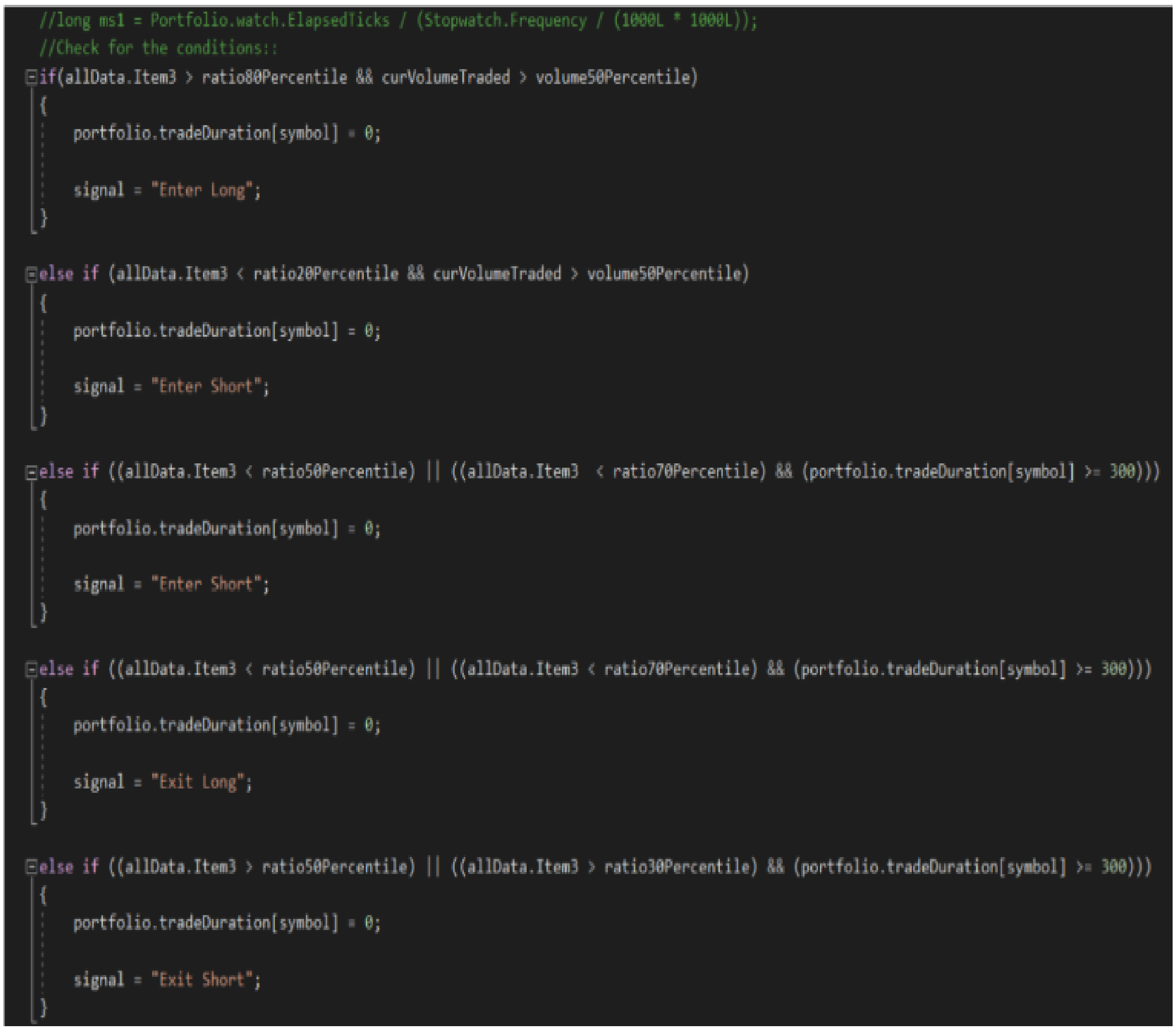


```
public static Tuple<double,double,double> CalcRatio(/*BookMessage book*/)

double ratio = 0;
PriceSizePair[] ask = /*book.Ask*/ new PriceSizePair[0];
PriceSizePair[] bid = /*book.Bid*/ new PriceSizePair[0];

int askLen = ask.Length;
int bidLen = bid.Length;

double bidVWAP = 0;
double askVWAP = 0;

double bidQty = 0;
double askQty = 0;

for(int i=0;i<bidLen; i++)
{
    bidVWAP += bid[i].price*bid[i].volume;
    bidQty += bid[i].volume;
}

for (int i = 0; i < askLen; i++)
{
    askVWAP += ask[i].price * ask[i].volume;
    askQty += ask[i].volume;
}

bidVWAP /= bidQty;
```

FIGURE 5.4: Ratio Calculation using Code

Ratio = (VWAP Ask - VWAP Bid) / (VWAP Ask + VWAP Bid)
1 Minute volume means the quantity of futures and options traded in a minute A hashmap is constructed to maintain symbol wise historical data map is maintained Every minute a new entry takes place and 1 entry is removed from the front.

The Executioner Rival Config needs several information data points and authentication values from the source to connect to their server. Once we send it the required



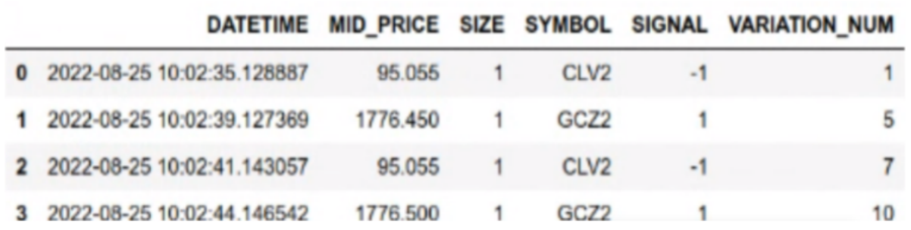
```
//long ms1 = Portfolio.watch.ElapsedTicks / (Stopwatch.Frequency / (1000L * 1000L));
//Check for the conditions::
if(allData.Item3 > ratio80Percentile && curVolumeTraded > volume50Percentile)
{
    portfolio.tradeDuration[symbol] = 0;

    signal = "Enter Long";
}

else if (allData.Item3 < ratio20Percentile && curVolumeTraded > volume50Percentile)
{
    portfolio.tradeDuration[symbol] = 0;

    signal = "Enter Short";
}

else if ((allData.Item3 < ratio50Percentile) || ((allData.Item3  < ratio70Percentile) && (portfolio.tradeDuration[symbol] >= 300)))
{
    portfolio.tradeDuration[symbol] = 0;

    signal = "Enter Short";
}

else if ((allData.Item3 < ratio50Percentile) || ((allData.Item3 < ratio70Percentile) && (portfolio.tradeDuration[symbol] >= 300)))
{
    portfolio.tradeDuration[symbol] = 0;

    signal = "Exit Long";
}

else if ((allData.Item3 > ratio50Percentile) || ((allData.Item3 > ratio30Percentile) && (portfolio.tradeDuration[symbol] >= 300)))
{
    portfolio.tradeDuration[symbol] = 0;

    signal = "Exit Short";
}
```

FIGURE 5.5: Strategy to Generate Long Short Signal

values, it gets connected to the Advanced Market Dependencies API

SocketSender: Once an object is created, it connects with IP address and port provided A TCP IP Connection is established, and from here, we will send the data to the executioner. It processes the symbols by putting them in a queue and then separately processing them on threads different from the main program. Now our MarketData is connected. If it fails, we report that.

A variation json object is created. This variation has a start time, stop time, a symbol list and a threshold. Then, a dictionary of 100 variations is created in a json file To replace our strategy, we can simply replace the variation json file to implement



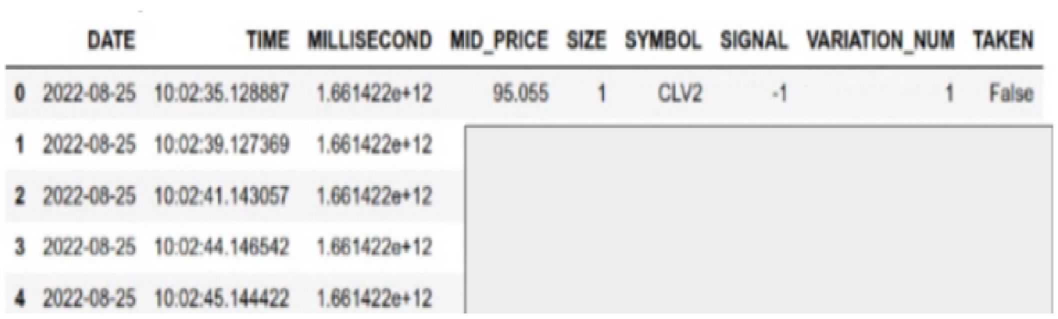
	DATETIME	MID_PRICE	SIZE	SYMBOL	SIGNAL	VARIATION_NUM
0	2022-08-25 10:02:35.128887	95.055	1	CLV2	-1	1
1	2022-08-25 10:02:39.127369	1776.450	1	GCZ2	1	5
2	2022-08-25 10:02:41.143057	95.055	1	CLV2	-1	7
3	2022-08-25 10:02:44.146542	1776.500	1	GCZ2	1	10

FIGURE 5.6: Log Table of Strategy

a new strategy. Our code can work simultaneously on 100 different strategies. We can treat any strategy as a black box, simply load the variation file and obtain the required results. This is the most important part of our code.
The strategy maker gives us the variation.json file. We simply load the variation file and our entire process from here on is automated. The variation file stores the thresholds and all the data points for the portfolio. The strategy object uses these values from variation file to run the strategy The number of portfolios is equal to the number of variations: With the variation.json file, we have finally created our diverse portfolios Description of a portfolio: Dictionary containing symbolwise latest order book is stored in a dictionary map¡symbol, LatestOrderBook¿
The next part of the code includes a HistoricalDataLogger that creates the Historical Time Tick Dataset for the futures and other symbols. It is sent a list of symbols for which it has to create the historical datasets Here we create timers for up to 60 variations. We create timers for all different variations possible. It will call Portfolio callback function every minute.In the portfolio timer callback, we loop for all symbols in each variation and then, we process for each symbol:
To process for a given symbol, a process book is called which is a function of the strategy object created We send the following values to the Executioner.

5.1.3 Executioner

Executioner is the portion of code that actually carries out the transactions on the Stock Exchanges using some external APIs Executioner is an aggregator: If it has



	DATE	TIME	MILLISECOND	MID_PRICE	SIZE	SYMBOL	SIGNAL	VARIATION_NUM	TAKEN
0	2022-08-25	10:02:35.128887	1.661422e+12	95.055	1	CLV2	-1	1	False
1	2022-08-25	10:02:39.127369	1.661422e+12						
2	2022-08-25	10:02:41.143057	1.661422e+12						
3	2022-08-25	10:02:44.146542	1.661422e+12						
4	2022-08-25	10:02:45.144422	1.661422e+12						

FIGURE 5.7: Executioner Input

multiple signals from different portfolios of the same symbol If it has 10 long signals and 6 short signals for a given symbol(future), it will only do 4 trades and executes 4 logs on the market. The lower the latency of the signal generation framework and signal sending framework, the lower slippage becomes.

Slippage is the actual difference between the price at strategy compile time and the price between the signal execution time. The slippage for a given future should be less than 5 BIPS, We can get profit from slippage too.

Load the variations to initialize the Portfolios Load the contract time from the contract config file Load the underlying symbols of all the portfolios to the market-DataManger(2 for loops) Wait for 1 min to let the API get all the required symbols(Get all symbols that have top 4 maturity date and then subscribe to the required symbols so that we can get their order books)

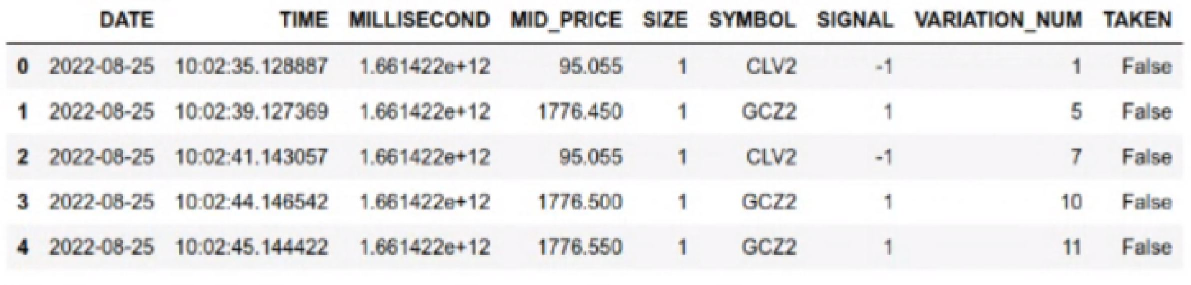
The lower the latency of the signal generation framework and signal sending framework, the lower slippage becomes.

Theoretical PnL report: Long signals - Short signals

Actual PnL report: Long executions - Short Executions

Mid Price = (Ask Price + Bid Price)/2

Mid Price is the actual price we send to the executioner at which it will try to buy the futures. The slippage will be calculated using this MidPrice The following values were sent to the Executioner:



	DATE	TIME	MILLISECOND	MID_PRICE	SIZE	SYMBOL	SIGNAL	VARIATION_NUM	TAKEN
0	2022-08-25	10:02:35.128887	1.661422e+12	95.055	1	CLV2	-1	1	False
1	2022-08-25	10:02:39.127369	1.661422e+12	1776.450	1	GCZ2	1	5	False
2	2022-08-25	10:02:41.143057	1.661422e+12	95.055	1	CLV2	-1	7	False
3	2022-08-25	10:02:44.146542	1.661422e+12	1776.500	1	GCZ2	1	10	False
4	2022-08-25	10:02:45.144422	1.661422e+12	1776.550	1	GCZ2	1	11	False

FIGURE 5.8: Executioner Log

5.2 Structural Vulnerabilities

Network Flow The computer connection through a TCP network is vulnerable during increased traffic. Resource sharing and Deadlock The portfolios and variational strategies run on parallel concurrent threads and hence are vulnerable to resource sharing bugs and deadlock situations.

5.3 Generating Trading Logs using GANs

We will now generate trading logs using AI by using Generative Adversarial Networks (GANs) to learn the statistical properties of existing trading logs and generate new logs that are statistically similar. Here is a high-level overview of how we use GANs to generate these trading logs:
Data Collection: Collect a large dataset of real trading logs from a financial market. We have already completed this process through the Algorithm Trading framework created above.
Data Preprocessing: Preprocess the dataset to ensure that it is clean and formatted in a way that can be used for training the AI model. This involves cleaning up missing or erroneous data, normalizing the data, and splitting it into training and testing sets.
Model Training: Train the GAN model on the preprocessed trading logs dataset. The goal is to train the model to generate new trading logs that are similar to the

real logs.
Model Evaluation: Evaluate the performance of the trained model by measuring the accuracy of the generated logs compared to the real logs.
Model Deployment: Once the model is trained and evaluated, it can be deployed to generate synthetic trading logs that can be used for various purposes, such as testing trading strategies, simulating market conditions, or training other machine learning models.
In the GAN model for generating trading logs, the specific values used for the epochs, activation functions, batch size, and dimensions are as follows:
Epochs: 10,000
Activation functions:
Generator network: ReLU (Rectified Linear Unit) activation function for all layers except the output layer, which uses a linear activation function
Discriminator network: LeakyReLU activation function for all layers except the output layer, which uses a sigmoid activation function
Batch size: 128
Input dimensions: 100-dimensional noise vector for the generator network
Output dimensions:
Generator network: 4-dimensional vector representing the Mid Price, Volume, Date, and Time of each synthetic trading log
Discriminator network: Single scalar value representing the probability that the input trading log is real
The number of epochs determines how many times the generator and discriminator networks will be trained on the dataset. The activation functions are used to introduce non-linearity into the network and improve its ability to capture complex patterns in the data. The batch size determines how many training samples are processed in each iteration of the optimization algorithm. The input and output dimensions define the size and shape of the data flowing through the networks.

In the GAN model for generating trading logs, the generator network is responsible for generating synthetic trading logs that have the same statistical properties as the real trading logs. The architecture of the generator network in the GAN model consists of several layers of neural networks, including convolutional layers, activation layers, and fully connected layers. The generator network takes in a random noise vector as input and outputs a synthetic trading log that resembles the real trading log. During the training process, the generator network receives feedback from the discriminator network and learns to generate synthetic data that is increasingly similar to the real data. The feedback is based on the loss function that measures the difference between the synthetic and real data. The generator network adjusts its weights and biases to minimize the loss function and improve the quality of the synthetic data.

In the GAN model for generating trading logs, the discriminator network is responsible for determining whether a given trading log is real or synthetic. The architecture of the discriminator network in the GAN model consists of several layers of neural networks, including convolutional layers, activation layers, and fully connected layers. The discriminator network takes in a trading log as input and outputs a probability score that indicates whether the trading log is real or synthetic. During the training process, the discriminator network receives both real and synthetic trading logs and learns to distinguish between them by adjusting its weights and biases to maximize the loss function. The loss function measures the difference between the discriminator's predicted probability and the true label (real or synthetic) of the input trading log. The feedback loop between the generator and discriminator networks ensures that both networks improve their performance over time.

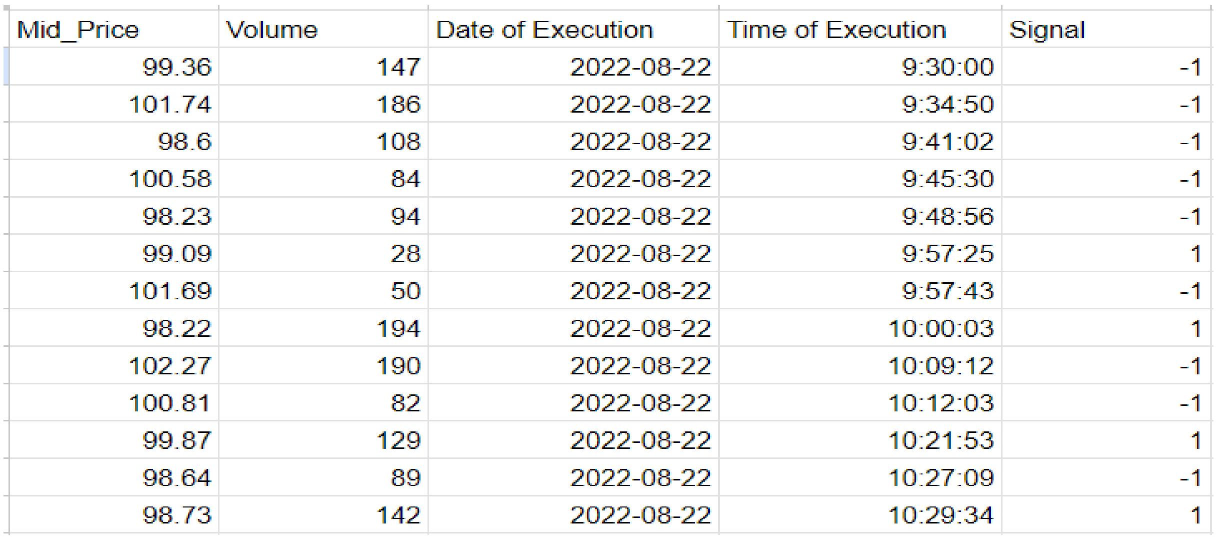
The generator network can be used to create new trading logs that can be used for research or educational purposes, without revealing sensitive or confidential information from real trading data. The discriminator network can be used to classify new trading logs as real or synthetic. This can be useful for detecting anomalies or fraud in real trading data or for evaluating the quality of the synthetic trading logs created by the generator network.

Chapter 6

Results and Conclusions

6.1 Results

We have constructed a comprehensive Low/Mid Frequency C live trading infrastructure that is secure, versatile and efficient. The basic summary of the trading framework is as follows:
Created a secure low latency pipeline to retrieve, archive and transmit execution information for subscribed futures contracts via an API
Generated long/short trading signals by employing systematic trading strategies and transforming unstructured data into proprietary data
Analyzed Slippage, Theoretical and Actual PL logs, system performance and trading results through extensive backtesting and validation
We obtained comprehensive trading logs from the executioner portion of the code. Once we obtain the dataset, we used a generative adversarial network to generate synthetic tabular trading data.The GAN model is trained on the dataset of real trading logs and generates new logs that have the same statistical properties as the real logs, including the mid-price, volume, date, and time of execution.The synthetic trading logs produced can be used for research and educational purposes, without revealing confidential information or putting real trading data at risk. However, it



Mid_Price	Volume	Date of Execution	Time of Execution	Signal
99.36	147	2022-08-22	9:30:00	-1
101.74	186	2022-08-22	9:34:50	-1
98.6	108	2022-08-22	9:41:02	-1
100.58	84	2022-08-22	9:45:30	-1
98.23	94	2022-08-22	9:48:56	-1
99.09	28	2022-08-22	9:57:25	1
101.69	50	2022-08-22	9:57:43	-1
98.22	194	2022-08-22	10:00:03	1
102.27	190	2022-08-22	10:09:12	-1
100.81	82	2022-08-22	10:12:03	-1
99.87	129	2022-08-22	10:21:53	1
98.64	89	2022-08-22	10:27:09	-1
98.73	142	2022-08-22	10:29:34	1

FIGURE 6.1: The Synthetic Trading logs generated by the GAN model

is important to note that the quality of the generated logs depends on the quality and size of the original training dataset, the complexity of the AI model, and the accuracy of the evaluation metrics used to assess the performance of the model.

The accuracy of a GAN model is not typically reported as a single metric, since the model is not a classification or regression model that outputs a predicted value for each input. Rather, the GAN model generates synthetic data that is intended to mimic the characteristics of the real data. In the case of the trading log GAN model above, the quality of the generated data could be evaluated by comparing it to the real trading logs in terms of statistical properties such as mean, variance, and distribution. Additionally, the generated data could be visually inspected to see if it resembles the real data in terms of patterns and trends. Therefore, while accuracy is not a meaningful metric for this type of model, the success of the GAN model can be evaluated by the quality of the synthetic data it generates and how closely it matches the characteristics of the real data.

The comprehensive trading logs of a company can be beneficial in various academic and non-academic ways. Some of the most beneficial academic uses of trading logs include:
Research and analysis: Trading logs can be used by researchers to analyze market

trends and patterns, identify trading strategies, and gain insights into market behavior.
Education and training: Trading logs can be used as a teaching tool for finance and economics students to help them understand the workings of financial markets and the principles of trading.
Algorithmic trading: Trading logs can be used to train machine learning models to develop algorithms for automated trading.
Risk management: Trading logs can be used to evaluate the risks associated with different trading strategies and develop risk management tools.

Some of the most beneficial non-academic uses of trading logs include:
Compliance and regulatory purposes: Trading logs can be used to comply with regulatory requirements and demonstrate compliance with trading regulations.
Performance evaluation: Trading logs can be used to evaluate the performance of traders and trading strategies, and make informed decisions about future investments.
Auditing and forensic investigations: Trading logs can be used for forensic investigations and audits to identify potential fraud or misconduct in trading activities.
Business development and strategy: Trading logs can be used to inform business development and strategy decisions, such as identifying new market opportunities or evaluating the impact of mergers and acquisitions.

6.2 Conclusions

We have trained a new improved GAN network against the comprehensive dataset generated through the executor logs.This generated a dataset that is very close to an actual log of a trading algorithm.The results thus obtained enable us to actually mirror a real time trading infrastructure. If we can mirror a real time trading algorithm, we can slowly and steadily introduce wrong values in our logs and try to trick/ set up the remaining algorithms. We have also tried to create a report on the types of effects the network connection and TCP/IP vulnerability have on the algorithm, as well as the resource sharing and deadlock constraints and fixes.

6.3 Future Scope

We can start work on creating a sweeper GAN that will find mimicking Trading algorithms and remove/flag them from the current environment. We can also learn about Black swan events and their effect on the markets in a protected code environment without any capital loss.

Chapter 7

References

7.1

Taking Over the Stock Market: Adversarial Perturbations Against Algorithmic Traders https://arxiv.org/abs/2010.09246

https://www.researchgate.net/publication/$365760678_{A}lgorithmic_{T}rading-Changing_{T}he_{P}aradig$

Aıt−Sahalia, Y., Saglam, M.(2014)*.Highfrequencymarketmaking : Takingadvantageofspeed(N*
//ssrn.com/abstract = 2342011

Anand, A., Venkataraman, K.(2016)*.Marketconditions, fragility, andtheeconomicsofmarketma*
10.1016/*j.jfineco*.2016.03.006

Angel, J.J., Harris, L.E., Spatt, C.S.(2011)*.Equitytradinginthe*21*stcentury.QuarterlyJournalof*
10.1142/*S*2010139211000067

Angel, J.J., Harris, L.E., Spatt, C.S.(2015)*.Equitytradinginthe*21*stcentury : Anupdate.Quarterl*
10.1142/*S*2010139215500020

Aquilina, M., Ysusi, C.(2016)*.Arehigh−frequencytradersanticipatingtheorderflow?cross−venueevidencefromtheukmarket.MarketMicrostructureandLiquidity,* 2(34), 1–35*.doi* :
10.1142/*S*2382626617500058

Arnuk, S., Saluzzi, J.(2009)*.Latencyarbitrage : Therealpowerbehindpredatoryhighfrequencytra*
*//pdfs.semanticscholar.org/*0*f*6*d/*8*a*58*f*1*ec*09*fc*107*e*7*df*18*f*786*b*55605*c*5*af*5*.pdf*

Avramovic, A.(2012)*.Manufacturingvolume : Thestocksplitsolution*(*MarketCommentary*)*.Cre*
//edge.credit − suisse.com

Baldauf, M., Mollner, J.(2015)*.High−frequencytradingandmarketperformance*(*WorkingPape*
//ssrn.com/abstract = 2674767

Battalio, R., Corwin, S.A., Jennings, R.(2016)*.Canbrokershaveitall?Ontherelationbetweenmake*
takefeesandlimitorderexecutionquality.TheJournalofFinance, 71(5), 2193–2238*.doi* :
10.1111/*jofi*.12422

Benos, E., Sagade, S.(2016)*.Pricediscoveryandthecross−sectionofhighfrequencytrading.Journ*
10.1016/*j.finmar*.2016.03.004

https : *//www.researchgate.net/publication/*320965956$_T$*he*$_f$*lash*$_c$*rash*$_{ar}$*eview*

https : *//assets.kpmg.com/content/dam/kpmg/xx/pdf/*2016/08/*swift − it.pdf*

https : *//www.researchgate.net/publication/*290963568$_1$*MDB*$_T$*he*$_B$*ackground*

https : *//pure.manchester.ac.uk/ws/portalfiles/portal/*61187680/*HFT*$_W$*P*$_2$017$_0$8$_3$1*.pdf*

https : *//www.ncbi.nlm.nih.gov/pmc/articles/PMC*8020781/

10.17485/*ijst*/2016/*v*9*i*8/87905

https : *//papers.nips.cc/paper/*2014/*hash/*5*ca*3*e*9*b*122*f*61*f*8*f*06494*c*97*b*1*afccf*3 −
Abstract.html

https : *//towardsdatascience.com/aifortrading* − 2*edd*6*fac*689*d*

https : *//arxiv.org/ftp/arxiv/papers/*2106/2106.06364*.pdf* : : *text* = *GANshttps* :
*//arxiv.org/abs/*1701.04862

Appendix A

Appendix: Machine Learning Models and Code Base

The following GitHub link contains the entire code base, all the datasets and the required dependencies.
https://github.com/atusharkm/Masters-Theis-Project-18MF3IM35

Code Base

```
# Importing the necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Dropout, Input, concatenate, BatchNorm
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import plot_model

# Defining the generator model
```

```
def generator_model():
    model = Sequential()

    # Adding the input layer
    model.add(Dense(32, input_dim=100))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))

    # Adding the hidden layers
    model.add(Dense(64))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Dropout(0.2))

    model.add(Dense(128))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Dropout(0.2))

    model.add(Dense(256))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Dropout(0.2))

    # Adding the output layer
    model.add(Dense(1, activation='tanh'))

    # Printing the summary of the model
    model.summary()
    # Returning the generator model
    noise = Input(shape=(100,))
    img = model(noise)
    return Model(noise, img)

# Defining the discriminator model
def discriminator_model():
    model = Sequential()

    # Adding the input layer
    model.add(Dense(256, input_dim=10))
    model.add(LeakyReLU(alpha=0.2))

    # Adding the hidden layers
    model.add(Dense(128))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(0.2))

    model.add(Dense(64))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(0.2))

    model.add(Dense(32))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(0.2))

    # Adding the output layer
    model.add(Dense(1, activation='sigmoid'))

    # Printing the summary of the model
    model.summary()

    # Returning the discriminator model
    img = Input(shape=(10,))
    validity = model(img)
    return Model(img, validity)

# Combining the generator and discriminator models to form the GAN model
def GAN_model(generator, discriminator):
    # Freezing the weights of the discriminator during training of the generator
    discriminator.trainable = False

    # Combining the two models
    gan_input = Input(shape=(100,))
    x = generator(gan_input)
    gan_output = discriminator(x)
    gan = Model(gan_input, gan_output)

    # Compiling the GAN model
    optimizer = Adam(lr=0.0002, beta_1=0.5)
    gan.compile(loss='binary_crossentropy', optimizer=optimizer)

    # Printing the summary of the model
    gan.summary()

    # Returning the GAN model
    return gan

# Plotting the GAN model
plot_model(generator_model(), to_file='generator_model.png', show_shapes=True)
plot_model(discriminator_model(), to_file='discriminator_model.png', show_shapes


-----------------------------------------------------------------------------
*****************************************************************************

Contract Time

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace rivalStrategy
{
    internal class ContractTime
    {
        public string contractTime { get; set; }
    }
}

Email Sender

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Mail;
using System.Text;
using System.Threading.Tasks;

namespace rivalStrategy
{
    internal class EmailSender
    {
        private string username = null;
        private string password = null;

        public EmailSender(string user, string pass)
        {
            this.username = user;
            this.password = pass;
        }

        //Function to send email whenever an error occurs
        public bool sendEmail(string subject, string mssg)
        {
            MailMessage mailMessage = new MailMessage();
            mailMessage.Subject = subject;
            mailMessage.Body = DateTime.Now.ToString() + "\n" + mssg;
            mailMessage.From = new MailAddress("tarun@26milescapital.com");
            mailMessage.To.Add(new MailAddress("tarun@26milescapital.com"));

            try
            {
                SmtpClient smtp = new SmtpClient();
                smtp.Port = 587;
                smtp.EnableSsl = true;
                smtp.Host = "smtp.gmail.com";
                smtp.UseDefaultCredentials = false;
                smtp.Credentials = new NetworkCredential(username, password);
                smtp.DeliveryMethod = SmtpDeliveryMethod.Network;
                smtp.Send(mailMessage);
                return true;
            }
            catch (Exception ex)
            {
                RivalLogger.WriteErrorLine("Email Not Sent!!");
                return false;
            }
        }
    }
}
```

Historical Data Logger

```
using RivalAdvancedMDAPI.Models.MarketDataMessages;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using System.Timers;

namespace rivalStrategy
{
    internal class historicalDataLogger
    {
        //Data structures required to store the data
        private static Dictionary<string, double> prevCummTradeQty = new Dictiona
        private static Dictionary<string, BookMessage> tempBookData = new Diction
        private static Dictionary<string, LastMessage> tempLastMessageData = new
        public static HashSet<string> symbols = new HashSet<string>();

        //File paths required to log the data

        public static void logHistoricalData(Object obj, ElapsedEventArgs arg)
        {
            tempBookData = marketDataStorage.bookMessageData;
            tempLastMessageData = marketDataStorage.lastMessageData;
            foreach (var symbol in symbols)
            {
                if (!prevCummTradeQty.ContainsKey(symbol))
                {
                    prevCummTradeQty.Add(symbol, tempLastMessageData[symbol].Cumu
                }
                else
                {
                    //RivalLogger.WriteLine("Logging in File!!");
                    BookMessage book = tempBookData[symbol];
                    double cumVolumeTraded = tempLastMessageData[symbol].Cumulati
                    int len = book.Asks.Length;
                    if (len < 10)
                    {
                        return;
                    }
                    double askVWAP = 0;
                    double bidVWAP = 0;
                    double askQty = 0;
                    double bidQty = 0;
                    for (int i = 0; i < 10; i++)
                    {
                        askVWAP += book.Asks[i].Price * book.Asks[i].Size;
                        bidVWAP += book.Bids[i].Price * book.Bids[i].Size;

                        askQty += book.Asks[i].Size;
                        bidQty += book.Bids[i].Size;
                    }
                    askVWAP /= askQty;
                    bidVWAP /= bidQty;
                    double ratio = (askVWAP - bidVWAP) / (askVWAP + bidVWAP);
                    double oneMinVolume = cumVolumeTraded - prevCummTradeQty[symb
                    prevCummTradeQty[symbol] = cumVolumeTraded;
                    string data = DateTime.Now.ToString() + "," + symbol + "," +
                    var t = Task.Run(() => RivalLogger.WriteOutputToFile(data, Pr
                }
            }
        }
    }
}
```

Load Data

```
using System;
using System.Collections.Generic;
using System.IO;

namespace rivalStrategy
{
    internal static class LoadData
    {
```

```
//Initialise the data structures required to load the data
public static Dictionary<string, Queue<double[]>> dataQueue = new Diction

//File paths required
public static string loadFilePath = "";

public static void getData()
{
    if (File.Exists(loadFilePath))
    {
        using (var reader = new StreamReader(@loadFilePath))
        {
            while (!reader.EndOfStream)
            {
                var line = reader.ReadLine();
                if (string.IsNullOrEmpty(line))
                    continue;
                var values = line.Split(',');
                double[] data = new double[2];
                string symbol = values[1];

                data[0] = Convert.ToDouble(values[4]);
                data[1] = Convert.ToDouble(values[5]);
                if (dataQueue.ContainsKey(symbol))
                {
                    if (dataQueue[symbol].Count < 1000)
                        dataQueue[symbol].Enqueue(data);
                    else
                    {
                        dataQueue[symbol].Dequeue();
```

```
                                dataQueue[symbol].Enqueue(data);
                            }
                        }
                        else
                        {
                            dataQueue.Add(symbol, new Queue<double[]>());
                            dataQueue[symbol].Enqueue(data);
                        }
                    }
                }
            }
            return;
        }
    }
}
```

Market Data Manager

```
using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
using RivalAdvancedMDAPI.Enums;
using RivalAdvancedMDAPI.Models.Requests;
using RivalAdvancedMDAPI.Models.Responses;
using RivalAdvancedMDAPI.Services;



namespace rivalStrategy
```

```
{
    internal class MarketDataManager
    {
        //Initialisations for the API Connection
        private MdfrTcpClient mdfrClient;
        private rivalClientConfig clientConfig;
        private Queue<InstrumentData> subReqQueue = new Queue<InstrumentData>();
        private bool bProcessQueue = false;
        private Thread subscriptionThread;

        //Data Structures to store Data
        public static HashSet<string> underlyingList = new HashSet<string>();
        public static Dictionary<string, List<InstrumentData>> instrumentList = n

        //File paths required to store the data



        //Constructor
        public MarketDataManager(rivalClientConfig cfg)
        {
            this.clientConfig = cfg;
            init();
        }

        //Subscribe the events that we want
        private void init()
        {
            TimeSpan ts = TimeSpan.FromSeconds(clientConfig.timeSpanIntervalInSec
            this.mdfrClient = new MdfrTcpClient(clientConfig.mdServerIP, clientCo
            this.mdfrClient.Connected += this.Server_Connected;
            this.mdfrClient.Disconnected += this.Server_Disconnected;
            this.mdfrClient.InstrumentDefinitionReceived += this.onInstrumentDefi

            //Subscribe to data that you want
            this.mdfrClient.BookReceived += marketDataStorage.onBookRecieve;
            this.mdfrClient.LastReceived += marketDataStorage.onLastMessageRecie



            this.bProcessQueue = true;
            this.subscriptionThread = new Thread(new ThreadStart(this.ProcessQue
            {
                Name = "MarketDataManager_SubscriptionQueueThread"
            };
            this.subscriptionThread.Start();

        }

        //Connect the Client to the marketData server
        public bool Connect()
        {
            RivalLogger.WriteLine("Connecting to MDFR - " + clientConfig.mdServer
            if (this.mdfrClient == null)
            {
                return false;
            }

            this.mdfrClient.Connect();

            var loginResult = this.mdfrClient.PublishLoginRequestAsync(clientConf
            if (!loginResult.Success)
            {
                RivalLogger.WriteErrorLine("Login Failed");
                Environment.Exit(0);
                return false;
            }
            return loginResult.Success;
        }

        /*
         * Specify the instruments whose definition you want to download
         * By specifying '*' you mean every symbol of every exchange
         */
        public void DownloadInstrumentDefinition()
        {
            var instrumentRequest = new InstrumentDefinitionRequest
            {
                User = clientConfig.mdUsername,
                InstrumentRequests = new List<InstrumentRequestForDefinition>
                {
                    new InstrumentRequestForDefinition
                    {
                        ClientInstrumentId = "1",
                        RequestType = InstrumentDefinitionType.SingleInstrument,
                        SecurityType = MdfrSecurityType.SECURITY_FUTURE,
                        Symbol = "*", // front month 10-yr futures
                        Exchange = "*"
                    }
                }
            };
            _ = this.mdfrClient.PublishInstrumentDefinitionRequestAsync(instrumen
            Console.WriteLine("PublishRequestDone!");
        }

        //On server Connection
        public void Server_Connected(MdfrTcpClient Client)
        {
            RivalLogger.WriteLine("MDFR connection successful.");
        }

        //On Server Disconnection
        public void Server_Disconnected(string reason, MdfrTcpClient Client)
        {
            RivalLogger.WriteErrorLine("Server Disconnected!!");
            Environment.Exit(0);
        }



        /*
         * This function chooses the top 4 symbols under a particular underlying
         * Then the symbols which is most liquid is chosen accordingly
         */
        public void onInstrumentDefinitionRecieve(InstrumentData def)
        {
            if (underlyingList.Contains(def.UnderlyingSymbol))
            {
                if (def.MaturityDate != Program.curDate)
                {
                    //RivalLogger.WriteOutputToFile(def.UnderlyingSymbol + "," +
                    if (instrumentList.ContainsKey(def.UnderlyingSymbol))
```

```
                {
                    instrumentList[def.UnderlyingSymbol].Add(def);
                    if (instrumentList[def.UnderlyingSymbol].Count > 4)
                    {
                        int ind = -1;
                        long? max = -1;
                        for (int i = 0; i < instrumentList[def.UnderlyingSymb
                        {
                            if (instrumentList[def.UnderlyingSymbol][i].Matur
                            {
                                max = instrumentList[def.UnderlyingSymbol][i]
                                ind = i;
                            }
                        }
                        InstrumentData temp = instrumentList[def.UnderlyingSy
                        instrumentList[def.UnderlyingSymbol][ind] = temp;
                        instrumentList[def.UnderlyingSymbol].RemoveAt(4);
                    }
                }
                else
                {
                    instrumentList[def.UnderlyingSymbol] = new List<Instrumen
                    instrumentList[def.UnderlyingSymbol].Add(def);
                }
            }
        }
    }

    //Subscribe to the queue for the processing of the symbols
    public void subscribeToMarketDataAsync()
```

```
        {
            foreach (var item in instrumentList)
            {
                for (int i = 0; i < instrumentList[item.Key].Count; i++)
                {
                    subReqQueue.Enqueue(instrumentList[item.Key][i]);
                }
            }
        }

        /*
         * This function process the processQueue on a different thread than the
         * process the symbols periodically
         */
        private void ProcessQueue()
        {
            while (bProcessQueue)
            {
                if (this.subReqQueue.Count > 0)
                {
                    lock (this.subReqQueue)
                    {
                        var request = this.subReqQueue.Dequeue();
                        var marketDataRequest = new MarketDataRequest
                        {
                            User = this.clientConfig.mdUsername,
                            InstrumentRequests = new List<InstrumentRequestForMar
                            {
                                new InstrumentRequestForMarketData
                                {
                                    ClientInstrumentId = $"mdsubreq-{this.clientC
                                    Feed = request.Exchange,
                                    Source = request.Exchange,
                                    GenericMarketDataSymbol = request.GenericMark
                                }
                            }
                        };

                        var marketDataResult = this.mdfrClient.PublishMarketDataR
                    }
                }
                else
                {
                    if (this.subReqQueue.Count == 0)
                    {
                        Task.Delay(1000).Wait();
                    }
                }
            }
        }
    }
}

--------------------------------------------------------------------------------
********************************************************************************
```