REINFORCEMENT LEARNING FOR AUTOMATED TRADING IN STOCK MARKET

LEE HONG JIAN

UNIVERSITI TEKNOLOGI MALAYSIA

PSZ 19:16 (Pind. 1/23)

UNIVERSITI TEKNOLOGI MALAYSIA
DECLARATION OF Choose an item.

Author's full name	:	LEE HONG JIAN			
Student's Matric No.	:	MCS241054	Academic Session	:	2024/2025 –2
Date of Birth	:	29/8/1998	UTM Email	:	leehongjian@graduate.utm.my
Choose an item. Title	:	REINFORCEMENT LEARNING FOR AUTOMATED TRADING IN STOCK MARKET			

I declare that this Choose an item. is classified as:

☒	**OPEN ACCESS**	I agree that my report to be published as a hard copy or made available through online open access.
☐	**RESTRICTED**	Contains restricted information as specified by the organization/institution where research was done. *(The library will block access for up to three (3) years)*
☐	**CONFIDENTIAL**	Contains confidential information as specified in the Official Secret Act 1972)

(If none of the options are selected, the first option will be chosen by default)

I acknowledged the intellectual property in the Choose an item. belongs to Universiti Teknologi Malaysia, and I agree to allow this to be placed in the library under the following terms :

1. This is the property of Universiti Teknologi Malaysia
2. The Library of Universiti Teknologi Malaysia has the right to make copies for the purpose of research only.
3. The Library of Universiti Teknologi Malaysia is allowed to make copies of this Choose an item. for academic exchange.

Signature of Student:

Signature : [illegible]

Full Name: LEE HONG JIAN
Date : 20/06/2025

Approved by Supervisor(s)

Signature of Supervisor I:	Signature of Supervisor II
Full Name of Supervisor I	Full Name of Supervisor II
Date :	Date :

NOTES : If the thesis is CONFIDENTIAL or RESTRICTED, please attach with the letter from the organization with period and reasons for confidentiality or restriction

“I hereby declare that I have read this thesis and in my
opinion this thesis is sufficient in term of scope and quality for the
award of the degree of Master in (Data Sciences)”

Signature	:	______________________________
Name of Supervisor I	:	
Date	:	

Signature	:	______________________________
Name of Supervisor II	:	
Date	:	

Signature	:	______________________________
Name of Supervisor III	:	
Date	:	

REINFORCEMENT LEARNING FOR AUTOMATED TRADING IN STOCK MARKET

LEE HONG JIAN

A thesis submitted in partial fulfilment of the
requirements for the award of the degree of
Master in (Data Science)

Faculty of Computing
Universiti Teknologi Malaysia

JUN 2025

DECLARATION

I declare that this thesis entitled *"REINFORCEMENT LEARNING FOR AUTOMATED TRADING IN STOCK MARKET"* is the result of my own research except as cited in the references. The thesis has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Signature	:	[illegible]
		....
Name	:	LEE HONG JIAN
Date	:	20 JUNE 2025

ACKNOWLEDGEMENT

In preparing this thesis, I was in contact with many people, researchers, academicians, and practitioners. They have contributed towards my understanding and thoughts. In particular, I wish to express my sincere appreciation to my main thesis supervisor, , for encouragement, guidance, critics and friendship. I am also very thankful to my co-supervisor and Associate Professor for their guidance, advices and motivation. Without their continued support and interest, this thesis would not have been the same as presented here.

I am also indebted to Universiti Teknologi Malaysia (UTM) for funding my master study. Librarians at UTM, Cardiff University of Wales and the National University of Singapore also deserve special thanks for their assistance in supplying the relevant literatures.

My fellow postgraduate student should also be recognised for their support. My sincere appreciation also extends to all my colleagues and others who have provided assistance at various occasions. Their views and tips are useful indeed. Unfortunately, it is not possible to list all of them in this limited space. I am grateful to all my family member.

ABSTRACT

The purpose of the research is to train agent that will not be influenced by the sentimental with using deep reinforcement learning (DRL) model in trading. In the reality of the financial market, the traditional trading strategy is based on the analysis (technical and fundamental) and statistical models to make decision, however, the limitation of traditional trading strategy also significant which when dueling with the highly dynamics of complex market. Nowadays, the main problem of the trading is the human emotional biases which is affected in the decision making and decrease the trading strategy of effective. In trading field, DRL was playing the importance roles in decision making because its will not distributed by emotional. An experienced trader may be affected by emotional by decision making however DRL can totally prevent the incident happen. The high potential of DRL in trading especially due to the dynamics market floating. DRL is one of the solutions for the emotional biases in financial markets. Compare to original/traditional systems, DRL algorithms able to learn directly from the actual market interactions and able to adjust the strategy to maximize the return rate based on the dynamics market. The learning of market interaction allows agents to improve the decision making and maximize the return rate. The aim is to create the policy that can maximize the cumulative reward over time. The suitable policy and reward mechanism is needed to train the agent that can fully eliminated the sentimental and the decision-making will be more discipline and the profit will be optimized. Every model has their own strengths and weaknesses; by identify the advantages of model will let the stakeholder apply those models in more appropriately toward the real-time market environment. The model with the higher value in F1-scores and accuracy, and optimized cumulative return rates based on the back testing will be selected for the future trading strategy in decision making. , DRL are able to working consistent and well in dynamics market environment such as the Two Sigma in DRL where the system will keep adapts the new various high amount of data and identify the profitable strategy that traditional trading strategy that unable to make it. The high profitability and adaptability without influenced by sentimental factor in the highly dynamics market environment.

ABSTRAK

Tujuan penyelidikan adalah untuk melatih ejen yang tidak akan dipengaruhi oleh sentimental dengan menggunakan model pembelajaran pengukuhan mendalam (DRL) dalam perdagangan. Dalam realiti pasaran kewangan, strategi perdagangan tradisional adalah berdasarkan analisis (teknikal dan asas) dan model statistik untuk membuat keputusan, walau bagaimanapun, batasan strategi perdagangan tradisional juga penting yang apabila bersaing dengan pasaran yang sangat dinamik. Pada masa kini, masalah utama perdagangan adalah kecenderungan emosi manusia yang terjejas dalam membuat keputusan dan mengurangkan strategi perdagangan yang berkesan. Dalam bidang perdagangan, DRL memainkan peranan penting dalam membuat keputusan kerana ia tidak akan diedarkan secara emosi. Pedagang yang berpengalaman mungkin dipengaruhi oleh emosi dengan membuat keputusan namun DRL boleh menghalang kejadian itu berlaku sepenuhnya. Potensi tinggi DRL dalam perdagangan terutamanya disebabkan oleh pasaran dinamik yang terapung. DRL adalah salah satu penyelesaian untuk kecenderungan emosi dalam pasaran kewangan. Bandingkan dengan sistem asal/tradisional, algoritma DRL dapat belajar terus daripada interaksi pasaran sebenar dan dapat melaraskan strategi untuk memaksimumkan kadar pulangan berdasarkan pasaran dinamik. Pembelajaran interaksi pasaran membolehkan ejen menambah baik membuat keputusan dan memaksimumkan kadar pulangan. Matlamatnya adalah untuk mencipta dasar yang boleh memaksimumkan ganjaran terkumpul dari semasa ke semasa. Polisi dan mekanisme ganjaran yang sesuai diperlukan untuk melatih ejen yang boleh menghapuskan sepenuhnya sentimental dan membuat keputusan akan lebih berdisiplin dan keuntungan akan dioptimumkan. Setiap model mempunyai kekuatan dan kelemahan mereka sendiri; dengan mengenal pasti kelebihan model akan membolehkan pihak berkepentingan menggunakan model tersebut dengan lebih sesuai ke arah persekitaran pasaran masa nyata. Model dengan nilai yang lebih tinggi dalam skor F1 dan ketepatan, dan kadar pulangan kumulatif yang dioptimumkan berdasarkan ujian belakang akan dipilih untuk strategi dagangan masa hadapan dalam membuat keputusan. , DRL mampu bekerja secara konsisten dan baik dalam persekitaran pasaran dinamik seperti Two Sigma dalam DRL di mana sistem akan terus menyesuaikan pelbagai jumlah data baru yang tinggi dan mengenal pasti strategi menguntungkan yang strategi perdagangan tradisional yang tidak dapat membuatnya. Keuntungan dan kebolehsuaian yang tinggi tanpa dipengaruhi oleh faktor sentimental dalam persekitaran pasaran yang sangat dinamik.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DRL	-	Deep Reinforcement Learning
SAC	-	Soft Actor Critic
PPO	-	Proximal Policy Optimization
DQN	-	Deep Q-Network

LIST OF SYMBOLS

S	States (all states space availability)
A	Action Space
P	Transition Function
R	Reward Function
γ	Discount factor (factor of discounts future reward)
$\rho 0$	Initial state distribution
$r_t(\theta)$	Probability ratio of the new policy to the existing policy
A_t	Advantage estimates at time step *t*
ϵ	Range for the clipping hyperparameter
$Q\ (s_t, a_t)$	Function of action-value
r_{t+1}	Reward from the environment
α	Rate of learning
$\max_{a'} Q\ (s_{t+1}, a')$	Maximum of the expected future reward at the state (s_{t+1})
$r\ (s_t, a_t)$	Immediate reward at the time step *t* for taking action a_t in the state s_t
$H(\pi(\cdot\mid s_t))$	Entropy of the policy, linked with exploration
Z	Z-Score
X	Value of the Data point
μ	Mean of the dataset
σ	Standard deviation of the dataset
Q3	75th percentile
Q1	25th percentile
min(X)	Minimum value of the data
max(X)	Maximum value of the data
R_t	Reward function at the time, t
P_t	Price of the stock at the time, t
P_{t+1}	Price of the stock at the next time step
S(x)	Output of the sigmoid function (range of values from 0 to 1)
e	Euler's number (around 2.718)
s_t	State at time, t

a_t	Action taken at time, t
r_{t+k}	Reward at time, t + k
θ	Weights of the neural network
$\pi_{\theta-\mathrm{old}}$	Old policy before update
π_{θ}	Policy parameterized by θ
$r_t(\theta)$	Probability ratio
$V_{\theta}(s_t)$	Estimated value function
$S[\pi_{\theta}]$	Entropy bonus for exploration
c_1 , c_2	Coefficients to balance loss terms

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Problem Background

Machine learning (ML) is the part of artificial intelligence (AI) which is ML mainly on creation of algorithms and models to let the agents learn from the data input and improve the performance over time without explicit programming (Vec et al. 2024). Generally, machine learning can divide into 3 parts which are supervised learning, unsupervised learning and reinforcement learning. Deep reinforcement learning (DRL) is where the agents are able to learn for making the decision by interphase with the real-time situation (environment) in the form of reward mechanism which is reward or penalty action will be based on the action taken by agent (Barto et al. 2025).

The aim is to create the policy that can maximize the cumulative reward over time. DRL can be used in many different fields such as agriculture, trading, food, and, et al (Georg et al. 2024). The benefits of DRL in trading are adaptability, improved decision making, automation and optimization, and, et al. DRL was enhanced adaptability in trading by the dynamic's algorithms that can be adjusted based on the real-time environment (market condition). Based on the algorithms, agents able to develop the strategic that suitable used for the particular situation due to volatility of market and "new" trends (Huang et al. 2024). The learning of market interaction allows agents to improve the decision making and maximize the return rate (Sangve et al. 2025). Based on the adaptive developed and trading of data-driven strategy, the effective of DRL was performing well than the original and the potential of DRL in decision making of forecasting. DRL agents maximized reduce the resources used and the trading process will be more effective and efficient in presence of automation (Kabbani and Duman, 2022).

In trading field, DRL was playing the importance roles in decision making because its will not distributed by emotional. An experienced trader may be affected by emotional by decision making however DRL can totally prevent the incident happen. The high potential of RL in trading especially due to the dynamics market floating (Kabbani and Duman, 2022). The emotional biases occur often in the decision-making phase at the financial market which can affect the overall outcome significantly (Aziz et al. 2024). The human behaviors such as loss aversion, overconfidence, regret aversion, and herding behavior, leads to lose step investment decisions (Rafandito et al. 2024).

In the reality of the financial market, the traditional trading strategy is based on the analysis (technical and fundamental) and statistical models to make decision, however, the limitation of traditional trading strategy also significant which when dueling with the highly dynamics of complex market. The traditional strategy relies on the assumption, rule-based, and predefined heuristics (sentimental) which will cause inflexibility and susceptibility toward market. In short, the traditional strategy often low capability in adaption of new data toward the dynamics market environment in the results the suboptimal performance while the high volatility of market (Chen et al., 2022).

The traditional algorithmic trading strategy was not demanded in the growing complexity of financial markets in global. DRL have introduced into the trading market due to the innovation method to increase the outcome of trading with minimize the emotional biases. In automation decision making process, DRL able to learn from the interaction of market and make the improved decision based on the long-term rewards instead of the short-term reaction to market fluctuation. The ability of DRL is consistent improve the strategy without the influenced by emotional biases in trading. Overestimate, overconfidence, overreaction towards the news and fear of loss are the main human behavior that will affect the emotional of trader and leads to wrong decision making (misjudgment). The investors will loss the optimal strategy due to emotional biases in the ends the stability of market and the return rate will be affected (Huang et al., 2024).

DRL is one of the solutions for the emotional biases in financial markets. Compare to original/traditional systems, DRL algorithms able to learn directly from the actual market interactions and able to adjust the strategy to maximize the return rate based on the dynamics market. Without intervene of human, DRL agents able to learn and adapt to delimited the factor of emotional biases in trading. By minimize the human intervene, enhance the trading strategy thereby the emotional biases will be reduced. The performance of the DRL was better than human traders as the consistency and prevent the impetuous decision in the turbulence financial market (Sangve et al., 2025). The critical issues of trading decision are emotional biases in the trading markets. The performance of traders and stability of market will be affected. Automation decision making process will minimizes the emotional biases. By using of DRL, traders able to often improved financial strategy that increase the performance over time.

The famous traditional trading strategy called Golden Cross strategy which is the traders will buy in stock when the 50-day moving average cross over the 200-day moving average and sell out when reverse occurs. The traditional strategy is simple and direct in the stable market environment only because its low adaptive toward the dynamics market environment. However, DRL are able to working consistent and well in dynamics market environment such as the Two Sigma in DRL where the system will keep adapts the new various high amount of data and identify the profitable strategy that traditional trading strategy that unable to make it. The high profitability and adaptability without influenced by sentimental factor in the highly dynamics market environment (Huang et al. 2024).

1.2 Problem Background

Nowadays, the main problem of the trading is the human emotional biases which is affected in the decision making and decrease the trading strategy of effective. Those human behaviors lead to the inconsistent of the human traders, impetuous action and irrational. Developing the system with minimize the human intervene meanwhile automated learning and adaption with the complexity of global financial markets is the main challenges. The potential solution that can overcome the emotional biases is by the automation decision making and learning directly from the interaction global financial market (Huang et al., 2024). Exploration of the used of DRL to minimize human emotional biases in trading. DRL develop and implement based on the trading strategy that able to learn and adapt directly to the real-time global financial market to eliminate the emotional biases in decision making and replaced with the DRL the traditional system. The estimated outcome is to replace traditional trading strategy and increase the return rate over time instead of making decision based on the emotional biases and dynamics market (Sangve et al., 2025). The Self-rewarding deep reinforcement learning (SRDRL) is the Model that created to increase the profit and efficiency by grants challenges with the reward mechanism for agent to learning. The key features of SRDRL such as self-rewarding mechanism, improved efficiency of learning, algorithmic trading of application, exploration and exploitation and reward shaping. There are some model for the SRDRL such as deep Q-network (DQN), proximal policy optimization (PPO) and soft actor-critic (SAC). In the learning paradigm, DQN is based on the value which is mainly focus on the action value estimation however PPO and SAC are policy-based which is will optimize the policy consistently to achieve the high profit. For the action space capability, DQN created for discrete spaces only; SAC for continuous spaces only; PPO are able to handle both discrete and continuous spaces. In terms of exploration and exploitation, SAC mainly on exploration but DQN and PPO are balance in the exploration and exploitation. In summary, DQN benefits in trading which is capability of learning from the large pool of market data, adaption of the dynamics market, keep improve the strategy and eliminate the sentimental in decision making that leads to highest profit and consistent. SAC benefits in trading are achieving the better exploration, increase the efficiency, improve the decision making continuously and adaption of dynamics market environment. However, PPO benefits in trading are ensure the stability of learning by controlling the policy, balance the ratio of the exploration and exploitation and adaption toward discrete and continuous actions (Haarnoja et al., 2018)

1.3 Problem Statement

The study is proposed to achieve the following objective:

a. To obtain the policy that give optimized return.
b. To train agent that will not be influenced by the sentimental with using SRDRL model.
c. Develop a dashboard that visualize the return of the agent that trained on different mechanism.
d. Compare the performance within model and discuss the strength and weakness between different model (DQN, PPO and SAC).

1.4 Scope Study

Computing tools (Python) will be used for the data collection and process in the research project. The range of time series data of the standard & poor 500 (S & P 500) from 01/01/2016 to 01/01/2024 will be collected from Yahoo Finance. After completion of data collection, the process of cleaning data will be conducted which is cleared the missing data and prepare work for the descriptive analysis. After that, the DQN, PPO and SAC will be used to do the decision making based on the policy and reward mechanism that can achieve the optimized return profit. The comparison of performance between model will using F1-score and optimized cumulative return rates. The strengths and weaknesses of the model also will be discussed based on the performance.

1.5 Significance of Study

In the project, the expected contribution is to train agent with the high efficiency and stable in sentimental. What kinds of the difference will be brought by using the different model in trading agents which is PPO, SAC and DQN. Indicate the strengths and weaknesses of the PPO, SAC and DQN in trading strategy. Helping stakeholder to harvest more profit without worry and sentimental impact. The trader will be automation decision making based on the policy and reward mechanism. The establish policy and reward mechanism is needed to train the agent that can fully eliminated the sentimental and the decision-making will be more discipline and the profit will be optimized. Every model has their own strengths and weaknesses; by identify the advantages of model will let the stakeholder apply those models in more appropriately toward the real-time market environment. The model with the higher value in F1-scores and accuracy, and optimized cumulative return rates based on the back testing will be selected for the future trading strategy in decision making.

CHAPTER 2

LITERATURE REVIEW

2.1 S&P 500 Introduction

Standard & Poor's Corporation introduced stock price indices that covering all kinds of industries in 1923. Within 3 years, they are launching the Composite Index which is comprising 90 stocks as initial. The composite index evolved into the Standard and Poor 500 (S&P 500) in 1957 to track the performance of the 500 listed companies in New York Stock Exchange (NYSE). Standard & Poor's Global Ratings is the agency for U.S.-based credit rating that provides rating of investment credit, data of financial (historical data) and analysis research on various equities which including stocks, bonds, commodities and others.

The companies that selected as the S&P 500 is the companies that can be represent as the economy of the United States because of their compositions in various fields. In reality cases, might be some of the companies will be bankrupt or dismissed for some of reason. The composition and weight of the index are updated/adjusted continuously time-to-time to ensure that its representability of the economy remains. Since the S&P 500 inception, over 900 new companies have been incorporated into the index, while an equal number have been removed from it (Siegel et al., 2006).

The S&P 500 index has consistently outperformed for most active money managers and mutual funds over time as extensively recorded. This performance can be attributed to the newly added that have higher performance than the old, dying companies that were delisted from the S&P 500 index.

2.2 Traditional Analysis in Trading

Traditional analysis in trading is the methods or methodologies that have be fully used/utilized until now (passed over 100 years) to predict the movement of market and make the correct decision based on the market information. There are the 2 main process for the traditional analysis in trading are fundamental analysis and technical analysis where fundamental analysis is to evaluates the healthiness of the financial and overall economic environment of assets by analysing economic, financial and qualitative factors that may affect the future price and technical analysis is to predict the trends of stock market price from the historical data (including volume and price) (Murphy, 1999). Traditional analysis methods still exist as foundation in the financial market.

Fundamental analysis is the analysis methods that to determine the background of the asset including companies or stock market. Fundamental analysis involving macroeconomic indicator study, earning of corporate, rate of interest and others data points that can access the intrinsic value of the asset (depends on the trader requirement or the pattern of study/view). Fundamental analysis is the cornerstone/basic of long-term investment strategy that widely used by investors in equity markets. The fundamental factors such as corporate earnings and sentimental of market or others factors that may affect stock prices, by fundamental analysis with modern tools can improve accuracy of prediction in the stock market (Inani et al., 2024). Macroeconomic indicators (GSP growth and inflation) are the main impact factors toward equity market; it was the main information of the fundamental analysis. By analysing and understanding the trends of macroeconomic to address the long-term viability of assets (Inani et al., 2024).

Technical analysis is the method of analysis the pattern of the price movement by using the price charts, volume analysis. The principle of technical analysis is the price reflects all the relevant information. The relevant information including sentimental of market, economic data and news. Traders able to predict the future price movement by analysis the price of market pattern repeat over time. The technical indicators such as moving averages and relative strength index (RSI). By technical indicators can determine the buy/sell signals effectively (Vanguelov, 2016). Technical analysis is the ability to provide real-time insights to allow the trader for react instantly. In the high-frequency of trading, the technical indicators are the main roles in algorithmic models where is combination of traditional analysis and machine learning (Liu and Florin et al., 2023).

Integration of fundamental and technical analysis approaches a balanced view of market. Fundamental analysis is to analysis the long-term viability of the asset and technical analysis is allowing the traders to determine the time of entries and exits by analysis the short-term price movement. Combination of fundamental and technical analysis leads to more informed investment decision. Fundamental analysis able to provide the insights into a value of company while technical analysis is to determine the correct timing for buy and sell points (Levi et al., 2021). The limitation of fundamental analysis unable to capture effectively toward the dynamics market and technical analysis is not familiar toward the long-term economic fundamentals of analysis. By combination of fundamental and technical analysis able to increase the accuracy of prediction but required calibration carefully to prevent overfitting of market signals.

2.3 Deep Reinforcement Learning Models in Trading

The aims of the DRL are to let the agents to learn optimal action through interactions with an environment with the policy in the Markov Decision Process (MDP) that will maximum the expected cumulative reward over time (Ezgi, 2024). MDP represented by a tuple:

$$M = (S, A, P, r, \rho 0, \gamma)$$

Where:

S is states (all states space availability);

A is actions (all possible action that will be taken by agent);

P is transition function (probability of transitioning from one state to another after action making);

R is reward function (the immediate reward will receive after the action taken in particular state);

γ is Discount factor (factor of discounts future reward);

$\rho 0$ is initial state distribution.

The goal of MDP is to find optimal policy that can bring the cumulative reward over time.

Deep Reinforcement Learning (DRL) has revolution of the trading algorithms by allowing/train an agent to learn optimum trading policy from the dynamic and complex financial market through trial and error. The reward mechanism was the marking system that based on the policy set to give the reward or penalty toward an agent's actions. There are the summary table of DRL model including their function, strength, limitation and finding for selecting the best 3 model to further research.

.

Table 2-1 Summary of DRL model in Trading

No	Model Name	Function of model in trading	Strength of model	Limitation of model	Results of paper	Citation
1	Deep-Q-Network (DQN)	By performing the deep neutral networks to calculate the Q-values, discrete trading actions (Sell, Hold, Buy) based on the mapping status	- Able to handles the high dimensional state spaces. - Without the handcrafted features, able to learn directly from the raw data (price of stock data)	- Trends to overestimate Q-values that may affect the grade of policy quality. - Only suitable for the discrete action spaces.	DQN-based trading outperformed rule-based strategies with higher cumulative returns and Sharpe ratios on historical stock data (Otabek et al., 2024).	Otabek, S., & Choi, J. (2024). Multi-level deep Q-networks for bitcoin trading strategies. *Scientific Reports (Nature Publisher Group), 14*(1), 771. doi: https://doi.org/10.1038/s41598-024-51408-w

		(observations of market)				
2	Proximal Policy Optimization (PPO)	By the surrogate clipped objectives to balance the exploration and exploitation. It will help in stable policy update for discrete and continuous spaces.	- Efficient of sample and robust to hyperparameter choices. - Well handle of stochastics policy and helps in improvement of exploration.	- intensive computation - May achieve the local optima without the adequate exploration.	PPO agents adapt well toward the dynamics market and outperformed than DQN and A2C models in portfolio management (Sun., 2023)	Sun, Q. (2023). *Reinforcement learning algorithms for stock trading* (Order No. 31765482). Available from ProQuest Dissertations & Theses Global. (3186188497). Retrieved from https://vpn.utm.my/dissertations-theses/reinforcement-learning-algorithms-stock-trading/docview/3186188497/se-2
3	Soft Actor-Critic (SAC)	SAC is the off-policy actor-critic	- Robust toward hyperparameter variations - Have a strong exploration in	- Complexity of computational	SAC agents overperforming than PPO and DDPG in	Kong, M., & So, J. (2023). Empirical analysis of automated stock trading using deep reinforcement

		algorithm that can help in optimize the maximum entropy objective and will helps in exploration and robustness	continuous action spaces	- Tunning issues for financial data.	the trading multiple assets with lower risk and high stability (Kong et al., 2023)	learning. *Applied Sciences, 13*(1), 633. doi:https://doi.org/10.3390/app13010633

Based on the Table 2.1, there are 3 DRL model that can used in trading but still not practical yet. Those 3 DRL models illustrate the rich diversity of approaches in automated trading. DQN model is values-based models which is excel in discrete decision but limitation in scalability. Actor-critic methods which are PPO, A2C and SAC are balance policy and value learning, handling continuous actions with the better flexibility. Twin delayed methods address the stability and market complexity issues. Despite advances, there are some of the common limitations including instability of training, sensitivity toward 'Noisy' data and computational overhead.

The Advantage Actor-Critic (A2C) and Twin Delayed DDPG (TD3) are also included in the similar research paper, which main focus on Jointed policy (optimal actor) and value (critic) networks to reduce the variances of gradient and increase the speed of training convergence, and TD3 improved DDPG by using the double Q-learning clipped to reduce the overestimation bias and delayed of policy update for stability. But they will not be further elaborated in this research.

The strengths, limitations and functions for each of the models are main consideration of the model selection for further research. The models in automated stock market trading are selected as the models that will used in the research. The 3 DRL models are PPO, DQN and SAC.

Table 2-2 Summary of the Three Models

No	Models	Reason
1	PPO	- Balance training stability and sample efficiency. - Able to handle discrete and continuous action spaces
2	DQN	- Able to handles the high dimensional state spaces - Basic DRL model
3	SAC	- Efficient Exploration - Sample efficient

Based on Table 2.2, there are the 3 models that selected as the research model. PPO able to balance the training stability and sample efficiency which is adaptive toward ‘Noisy’ data and non-stationary nature of stock market data. PPO able to handle discrete and continuous action spaces which is versatile toward different trading decision such as portfolio adjustments, buy or sell signals. Clopped objective able to prevent the large destructive policy updates which is able to reduce the risk of performance collapse. DQN model created as the basic of the DRL model that able to handles the high dimensional state spaces for the discrete action spaces with high effectively. With the Actor-critic structure allows agents to learn the complex policy while evaluating for the next better updates. SAC able efficient exploration which is important to avoid the premature convergence in the dynamics/volatile financial market. With the off-policy training, sample efficient where allowing agents learn faster from the historical data.

2.3.1 Proximal Policy Optimization (PPO) Model

Proximal Policy Optimization (PPO) is a reinforcement learning algorithm that created/designed to improve the policy gradient methods. The ability of PPO is to balance exploitation and exploration by using a novel objective function. Exploration is the strategy of testing by using the new dataset and exploitation is capitalizing on the existing strategy. By using clipped probability ratio as the part of the function of loss to avoid the large update extremely and ensure the stability of learning. The primary objective function for PPO is

$$L_{CLIP}(\theta)=E_t[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta),1-\epsilon,1+\epsilon)\ A_t)] \tag{2-1}$$

Where:

$r_t(\theta)$ is the probability ratio of the new policy to the existing policy;

A_t is the advantage estimate at time step *t*.

ϵ is the range for the clipping hyperparameter.

By using the function, able to prevent the policy from making unproductive update and help an agent to learn efficiently and stable (Schulman et al., 2017). PPO applied in stock trading and the aims is optimize the strategy of trading by allowing the agent to learn from the historical data (financial data). In the trading, PPO able to optimizes the decision of entry, exit and hold by interacting with the real-time stock market environment. PPO also using the MDP where State (S), Action(A) and Reward (R) as the basic of fundamental. State represents the condition of stock market. Action is the action taken such as buy, sell and hold. Reward is the profit or loss for every action taken. PPO showing the better performance in cumulative return over time and high stability (Sun, 2023). In the dynamics stock market, PPO able to optimize the policy by alternating between sampling data and the objective function (Schulman et al., 2017).

Attached some of the finding from the journal paper in Alireza et al., 2024. The results performance of PPO in stock market over time as following:

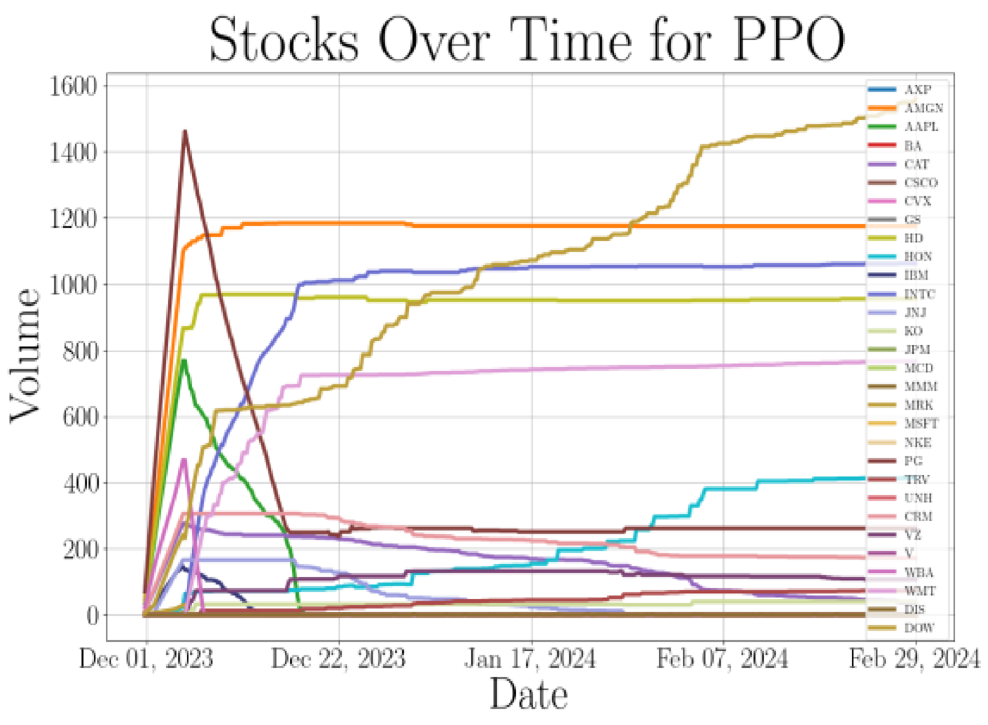


Figure 2-1 Stock Over Time for PPO (Alireza et al., 2024)

Based on the Figure 2.1, the graph showing the volume of trading of different stocks over time for PPO model in financial market. The stock behaviour in the graph showing that each of the companies represent each line for the volume of stock traded from 01 Dec 2023 until 29 February 2024. Volatility of the vary stock market such is the sharp rise in the volume during the early of Dec 2023. The significant changes will trigger the volume of trading activity. PPO model will be affected by the timing and the magnitude of the volume of trading. Larger volume of stock market will lead to PPO model more active and capitalizing responding toward signals. The PPO model performance in the varying level of volume was working fine and optimizing cumulative returns rates. In the summary of the Figure 2.1, PPO model able to work in dynamic stock market and adjust the trading strategy based on the real-time financial data. The increase of volume in trading as the model able to determine the opportunity for profit which means PPO able to adapt the dynamic dataset and finalize the aggressive investment strategy based on the data given.

2.3.2 Deep-Q-Network (DQN) Model

Deep Q-Network (DQN) model is the combination of deep neutral network and Q-learning to solve the complex decision-making which is highly suitable for stock market dataset. The DQN model able to handle the large state and action spaces where the dataset of stock market is large. By using deep neural network to approximate the Q-value function, DQN able to allow agent to learn the optimal strategy of trading that can obtain the optimum of cumulative reward over time. Traditional analysis focus on the prediction only, DQN integrate the prediction of stock price and the optimization of trading strategy (Kabbani et al., 2022). The DQN formula as following:

$$Q\ (s_t, a_t) \leftarrow Q\ (s_t, a_t) + \alpha\ [r_{t+1} + \gamma\ \max_{a'} Q\ (s_{t+1}, a') - Q\ (s_t, a_t)] \tag{2-2}$$

Where:

$Q(s_t, a_t)$ is the function of action-value;

r_{t+1} is the reward from the environment;

γ is the discount factor (linked with future reward); α is rate of learning;

$\max_{a'} Q(s_{t+1}, a')$ is the maximum of the expected future reward at the state (s_{t+1}).

In the DQN model, allow to train the agents by minimize the temporal different error between the predicted and actual Q-values, to ensure the agent by learning the optimal policy to explore the stock market environment (Sun, 2023). Key challenges in DQN for trading including the balance of exploration and exploitation and reward function design. DQN model will overfitting toward the specific market if the exploration is not balanced properly (Ezgi, 2024). The exploration is exploring the new strategy and exploiting is the profitable part. If the exploration is not balanced with exploitation which means ratio of exploration is lower than exploitation, the decision making of the agents that train by DQN model will hard to get the proper or correct decision. The traditional reward functions are not sufficient enough to capture the complexness of the financial market. In the traditional reward function only using the profit as the reward and it will lead to the biases especially when the agent main focus on selling actions. By using the self-rewarding mechanisms able to address the traditional reward function issues by allowing the agent to adapt to changing market condition effectively (Huang et al., 2024).

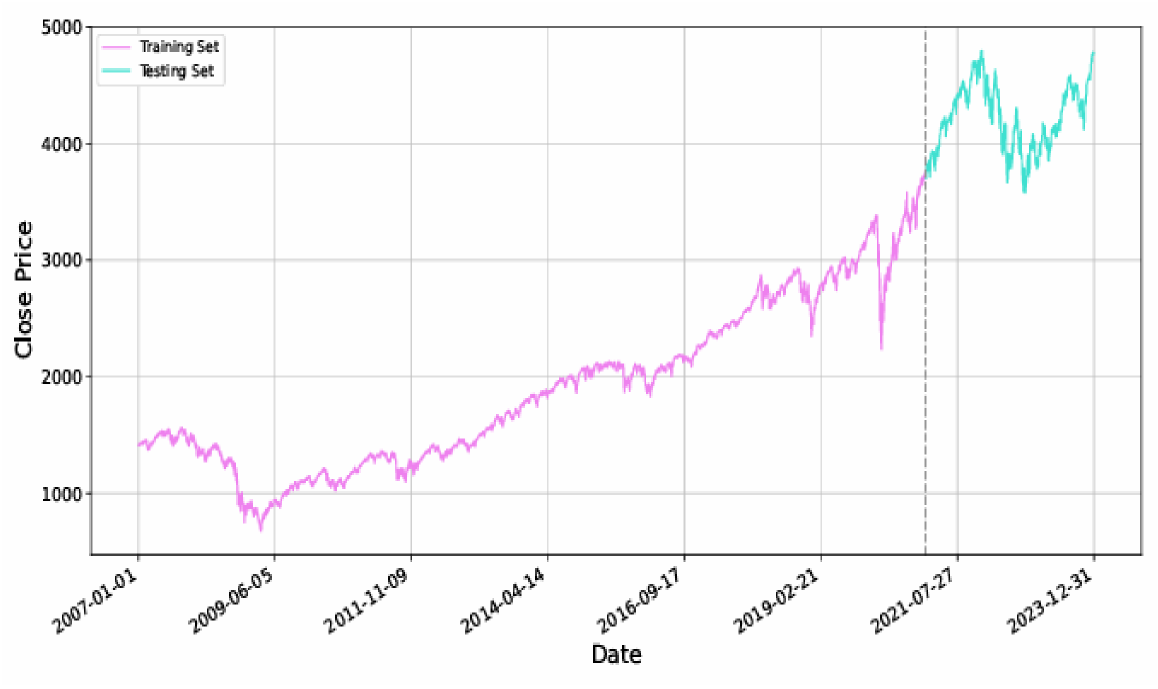


Figure 2-2 Close price over Data for DQN model (Huang et al., 2024)

The finding for the Figure 2.2 is the performance of training and testing, generalization, adaptability toward dynamics stock market and impact of model overfitting. In the training set of DQN where the agent will learn to make the decision based on the historical data given. The agents will learn the optimize actions (buy, hold and sell) based on the pass price trends. For the testing set is the agents to predict and adapt toward the future price movement. After 27 July 2021, the close price increased which is overperformed than previous pattern in training set. The ability of model to react toward the trend of stock data in testing periods will determine the success of the model. In the training set, DQN model able to well-generalize to unseen data in the testing set. If the model is overfitted toward the historical data will lose the ability of the model to adapt toward the unseen data.

In short, by enable the agent to learn optimal trading strategy autonomously DQN model able to handle high-dimensional state and action spaces which means make it more suitable for the dynamics financial stock market. However, the challenges such as trade-off of exploitation and exploration and reward function design are the items that can improve the model of DQN. Future improvement for the challenges of DQN are self-rewarding mechanisms and balancing the exploration and exploitation (Huang et al., 2024).

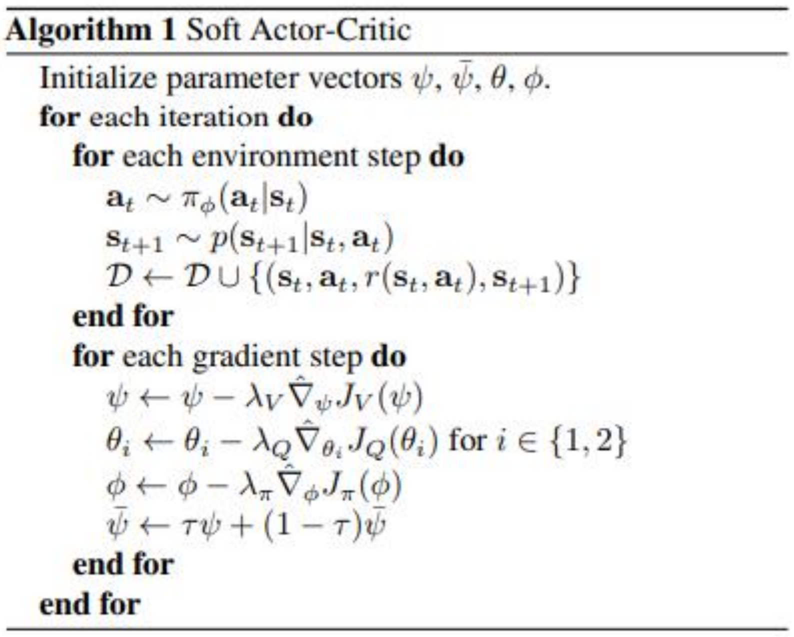
2.3.3 Soft Actor-Critic (SAC) Model

SAC is an off-policy algorithm (Sarthak Singh et al., 2022) that applied stochastic policy optimization with Deep Deterministic Policy Gradient (DDPG). It uses feature of entropy regularization but optimizes the trade-off between return and entropy. Off-policy differs from the normal DRL for it aims to reuse the historical experience, instead of new sample needed for each gradient step, and so the policy learned required increasing number of them.

SAC is also a maximum entropy framework (Tuomas Haarnoja et al., 2018) that augments the maximum reward RL objective with a maximizing entropy. Thus, it gives a significant improvement for its robustness in the face of model and estimation error, and also strengthen its exploration by obtaining different behaviours. So, combine these two features, the off-policy maximum entropy actor critic algorithm is called soft-actor critic algorithm, which is famous on its sample-efficient learning and stability.

Actor-critic algorithm is formed by three components, which are an actor-critic structure, an off-policy formulation that allows the efficiently reuse of old data, and an entropy maximization. This include two alternating steps which are policy evaluation and policy improvement. The policy evaluation means calculating the value function for a policy, while policy improvement is about to get a better policy based on the value function obtained. So, the actor is the policy, and the critic is the value function.

There are many other on-policy algorithm also use entropy but instead of maximizing it, they only use it to regularize the model. The algorithm of SAC can be showed in a snippet of the pseudo code like below:



```
Algorithm 1 Soft Actor-Critic
Initialize parameter vectors $\psi$, $\bar{\psi}$, $\theta$, $\phi$.
for each iteration do
  for each environment step do
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
  end for
  for each gradient step do
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$
  end for
end for
```

Figure 2-3 Pseudo code of SAC algorithm

In this pseudo code, the **ψ** is the parameters of the value network Vψ(s), while $\bar{\psi}$ is the parameters of the target value network V$\bar{\psi}$(s), θ are parameters of the Q-networks Qθ1(s,a), Qθ2(s,a), and φ is parameters of the policy (actor) πϕ(a|s). The first line is to sample an action from the current policy, the second line is the step that get into the next state, while the third line means the storing of transition in the replay buffer.

The Q-networks is used to estimate the soft Q-value (s, a), the value network is used to estimate soft-state value V(s) for stability, while policy network is applied to learn stochastic policy that maximizes Q-values and entropy. The replay buffer is set to store experience for off-policy updates, and target values are smoothened by the target network to stabilize the Q-learning.

According to the paper written by Tuomas Haarnoja et al. in 2018, several benchmark environments like Hopper-v1, Walker2d-v1, HalfCheetah-v1 and others are used for both the training and testing of the SAC. The outcome of the results are shown as the Figure 2.4 below.

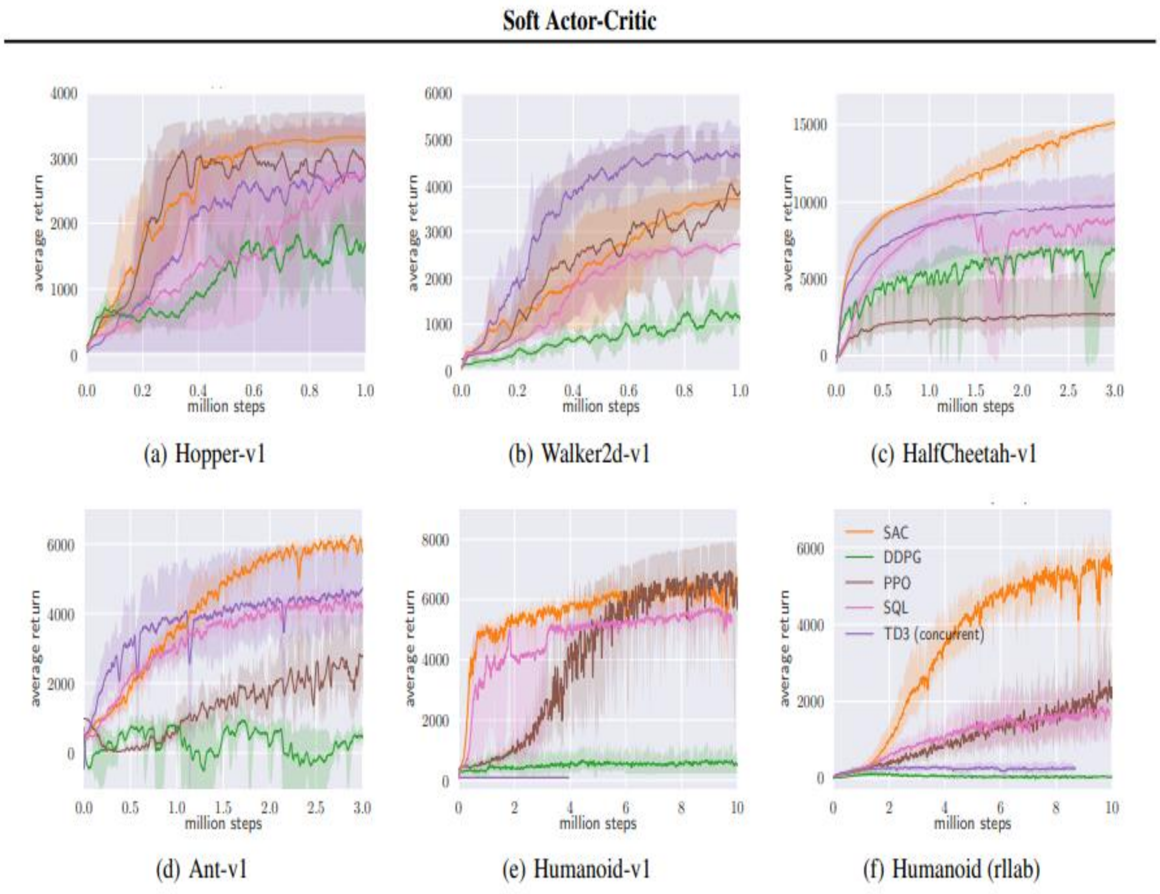


Figure 2-4 Training curves on continuous control benchmarks (Tuomas Haarnoja et al., 2018)

We can see that in the comparison of different models, the SAC indicated as the yellow line did outperform other model in most of the cases for both of the learning speed and final performance.

Besides that, the paper also discussed how the stochastic and deterministic policy of the SAC will affect the training and the testing outcomes. Stochastic policy will output a Gaussian probability distribution over action, and the action is chosen based on the formula of hyperbolic tangent function, which the output is within the range of -1 to 1, and if the output is close to -1, a reverse action will be taken, when it close to 0, no action is being taken, and if it is close to 1, there will be a positive action just based on how we define it. While the deterministic policy will only output a single action for each state. The results of using stochastic policy and deterministic policy in SAC is shown as Figure 2.5 below.

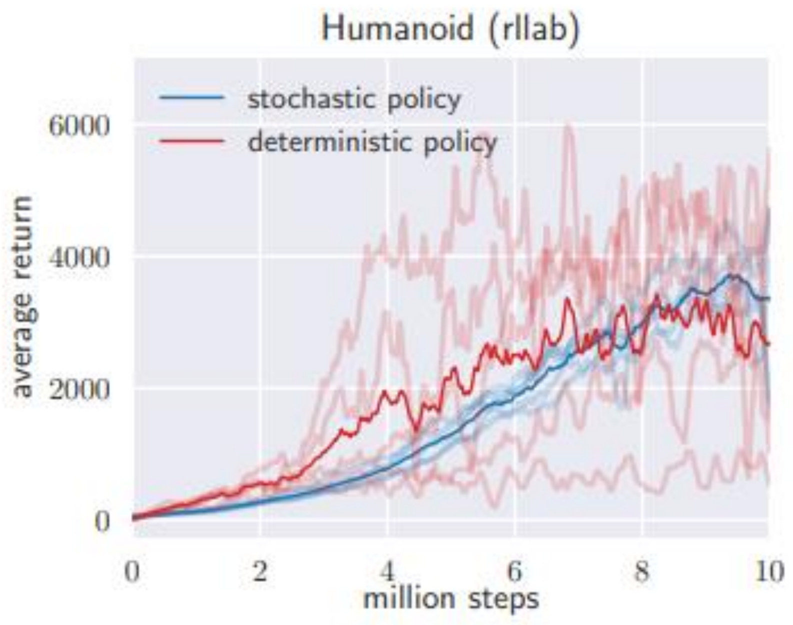


Figure 2-5 Comparison of stochastic policy and a deterministic variant (red) in the Humanoid (rllab) benchmark

As what Figure 2.5 shown above, stochastic SAC samples action and keeps them by maximizing the entropy, while the deterministic SAC only choose the mean action and drop the entropy term, and thus the stochastic SAC yield more stable and efficient learning.

2.4 Reward Function in Deep Reinforcement Learning

The reward mechanism in SAC model is the main process of agent's learning process. The function of reward defines the agent's action taken by evaluate and provide the feedback based on the decision making. For the trading, the reward is based on the profit and returns but SAC model introduces a more sophisticated mechanism by incorporating entropy maximization. The agent is not only maximizing the reward (profit) but encourage exploration by maximizing the entropy within the acceptable bounds. It helps agent from over-deterministic or exploiting a narrow set of strategy (Alireza et al., 2024). The maximum entropy in SAC reward function as following:

(2-3)

$$\text{Objective}=E_{\pi}\left[r\left(s_t, a_t\right)+\alpha \cdot H\left(\pi\left(\cdot \mid s_t\right)\right)\right]$$

Where:

$r(s_t, a_t)$ is the immediate reward at the time step *t* for taking action a_t in the state s_t.

$H(\pi(\cdot|s_t))$ is the entropy of the policy, linked with exploration.

α is the parameter of temperature that control the trade-off for exploitation and exploration.

The reward function is to create to capture the trade-off between maximizing profits and controlling risks in the financial stock market. The traditional reward function in trading model relies on the profit and loss calculation. However, SAC model approach to overcome the issues of overfitting toward stock market by encouraging exploration of different trading strategies through the entropy term. The stock trading is dynamics where exploration may lead to discover new or more profitable strategy that might be overlooked (Alireza et al., 2024). In the SAC model, the reward function (Li, 2024) including:

1. Profit and Return – standard of the financial metric where the agent will receive the reward based on the return of the trading action taken.
2. Cost of transaction – The penalty for the trades for incur cost to ensure agent optimize for the net profits instead of gross profits.
3. Adjustment of risk – The additional penalty or reward based on the volatility that can encourage agent to consider the long-term stability alongside short-term return.

The benefits of the reward function for SAC model are improved exploration, stability and adaptability. By encourage the randomness, SAC prevent the situation of suboptimal local optima which is common phenomena in traditional reinforcement learning methods that only focus on the maximizing profits (Haarnoja et al., 2018). Balanced the trade-off exploration and exploitation by the reward structure helps SAC achieve more stable condition for learning with high noise data (financial stock market) (Alireza et al., 2024). SAC able to adapt dynamics stock market data and adjust the trading strategy accordingly which means make it more robust in dynamics environments (Kong et al., 2022). The reward function in SAC is importance for balancing the profitability with exploration. By using traditional reward maximization and entropy term, SAC model able to provide more adaptable and robust trading strategy especially in complex dynamic financial stock market.

The reward function of DQN model can be divided into profit-based reward, dynamic reward shaping, multi-objective reward function and risk and profit trade-off. Profit-based reward is the straightforward reward design in DQN where the reward is change in asset value after a trading action which is buying or selling. The difference between the price of asset at the time of agent take action and the price at a later time. The approach has notable drawbacks. For example, sell-biased reward structures obtain dominate due to the reward is only received when the trade in closed condition, will lead to agent selling excessively over other actions (Sun, 2023). Dynamic Reward Shaping is to address the limitation of the static reward function. The method is to adjust the reward function in real-time with some of the factors affected such as transaction cost, market liquidity and volatility. DQN models can better navigate in the complex trading environment by reducing the risk of the strategy that made by agent trapped in suboptimal condition (Huang et al., 2024). Risk and profit trade-off is the combination of risk management with reward function by integrating risk-adjusted returns. By using Sharpe ratio, DQN agent is to encourage not only maximize profit but also required consider minimize the risk associated with decision of trading. This will lead to risk-averse and stable of seek for the long-term returns (Huang et al., 2024). Due to dynamic of the financial stock market, the single objective like profit maximization insufficient. The integration of multi-objective reward function where balance the short-term profit and long-term stability and risk minimize. The Self-Rewarding Deep Reinforcement Learning (SRDRL) introduce a reward system that can adjust dynamically based on the predicted rewards and expert feedback, combination of human's knowledge with learning process of the agent.

The challenges and the solution in reward function design for DQN is overestimation bias and partial fulfilment problem. Overestimation bias due to the Q-values, it will lead to unstable of leaning especially when the model interacts with the dynamics financial stock market. Self-rewarding DQN (SRDQN) able to overcome the overestimation bias issues by separating the action selection process from the value estimation process, it will lead to more stable and reliable trading strategy (Huang et al., 2024). Due to market constraints, agent may take action (buy or sell) but only partial fulfilment. By using action retention mechanism to improve the ability of agent to handle real-time dynamics financial stock market (Sun, 2023). In short, the reward function of DQN is the importance role to guiding the agent to make the decision in the trading environment. The traditional reward function in DQN have limitation due to only focus on profit, SRDQN is the advanced techniques that can overcome the issues such as overestimation bias and partial order fulfilment in the complex, non-stationary stock market.

The main challenge in PPO is to ensure the reward function encourage desirable behaviours without lead to large and unstable update of policy, PPO operates with a surrogate objective that is optimized iteratively. PPO works to minimize the discrepancies in action taken while optimize the long-term profit which is the cumulative reward over time. The involvement of advantage function of reward function in PPO where the difference between the expected return and the average return. It allowing the PPO model to prioritize the action that led to the higher expected reward. The reward function of PPO helps reduce the variance of advantage estimates by value function which is estimates the expected reward over time (Schulman et al., 2017).

The optimization of PPO reward function by the bonus of entropy. This design is to encourage exploration by penalty deterministic policy where it will promote more diverse action sequences in training of agent process. By using the agent to explore in the different state-action pairs more thoroughly, the entropy bonus can prevent premature convergence to suboptimal solution (Schulman et al., 2017). The reward function of PPO is created/designed to adapt toward the different problem domains such as the high-dimensional continuous control tasks. The reward function designed to guide the agent to learning effectively without overwhelming with the complexity. In the algorithmic trading, the reward function is based on the profit and loss. This reward function is sensitive toward the timing of entry and exit actions where only for completing a transaction (selling stock) trigger the reward. The reward is scalable to reflect the profitability of the trade (Sun, 2023).

In the summary, the design of the reward function in PPO model is importance to remain the stable and performance of the process of learning for the agent. The balance of exploration and exploitation where exploration is to explore the new actions and exploitation is policy learning. By using the advantage function and entropy bonuses, PPO able to optimize the policy effectively over the dynamics financial stock market.

2.5 Motivation

Even the advancements of the deep reinforcement learning in algorithmic trading, there are the challenges that prevent trading strategy optimization. Due to the high-dimensional and non-stationarity of financial stock data, the traditional deep reinforcement learning techniques is hard to capture the complex market dynamics. Those models might be overfitting by the historical data and it will be limited to generalize to unseen market conditions. The balance of exploration and exploitation in trading strategy is importance because the high trade-off toward exploration may lead to loss of large financial and low trade-off toward exploration can lead to poor strategic performance (Huang et al., 2024).

To overcome the challenges, the research optimizes the trading strategy by introduce the self-reward mechanism into the deep reinforcement learning (DRL). By modify the reward mechanism based on the trading performance and market condition, may lead to increase the adaptability of the agent to shifting condition and reduce the chances of overfitting. The aims is to train agent with the high efficiency and stable in sentimental. The establish policy and reward mechanism is needed to train the agent that can fully eliminated the sentimental and the decision-making will be more discipline and the profit will be optimized.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

The research mainly focusses on application of deep reinforcement learning (DRL) techniques to develop an automated trading system for decision making in the stock market. Automated trading in decision making including the buy, sell or hold decisions to achieve the optimal cumulative return over time. The traditional trading strategy relies often on the fixed rules or can be called as supervised learning. However, DRL able to provide the powerful framework toward model specially handling the dynamics, uncertain of the financial stock market with learning the optimal trading policy through interaction with the real-time environment. The trading environment is modeled as the Markov Decision Process (MDP) which is the agent learning based on the observe the current market state and make the decision (takes an action which is buy, sell and hold) and receive the reward accordingly toward the portfolio performance.

The objective of the DRL agent is to learn a policy π(a|s) that optimize the cumulative discounted reward G_t defined as:

$$G_t = k = \sum_{\infty}^{0} \gamma^k R_{t+k+1} \tag{3.1}$$

Where:

R_t is the immediate reward at time t (the time when profit or loss based on trades)

$\gamma \in [0,1]$ is the discount factor which is determine the vital of future rewards relative to immediate gains. The S&P 500 data enable to provide the high volume of historical data including historical OHLC data and volume data. OHLC data is data for Open, High, Low, Close prices. Volume data is daily trading volume. The technical indicators where the transformed into state representations to guide the agent to make the decision.

As mentioned at Chapter 2 Literature Review, the input data will be Standard and Poor 500 (S&P 500). The range of data is from 01/01/2016 to 01/01/2024 in the Yahoo Finance. The adjusted closing price of the S&P 500 will be taken for analysis. The reason for using an adjusted closing price is that some of the stock may have dividends share splitting or consolidation that will impact the price of stocks which belong to the S&P 500. After the price is adjusted, the price before the dividend or conditions above will be amended to ensure the consistency of the data.

After the data is collected, some descriptive analysis and data pre-processing will be conducted to have a holistic overview of the data. Then, the dataset will be split into two which are training data and test data with the assistance of some library in Python. The model selection based on the ability, strength and weakness of the deep reinforcement learning (DRL) models. The model selected will be used to train on the training data. After the models are trained, they will try to run the test data and then the error between the decision making and actual data will be measured in the unit of F1-score and cumulative return over time. In the research that found in Chapter 2 (Literature Review), DRL models in trading field are evaluated into Deep Q-Network (DQN), Soft Actor-Critic (SAC), and Proximal Policy Optimization (PPO). Each algorithm represents the different approaches by using the basic as the MDP. DQN suited for discrete action spaces, SAC optimized for continuous control and robust exploration, and PPO balancing training stability and sample efficiency through policy gradient methods.

Some parts from Box-Jenkins method will be applied in order to find the correlation between the moving average and the contemporary index's price, and also those moving average data will be generated in order to feed into the algorithms. Then, a dashboard that shows the effect of the models will be developed in order to provide clearer insights for the investors and traders. The performance of the DRL models will be compared based on the ability to generate the profitable trading strategy based on the historical S&P 500 data with the optimal cumulative returns. The methodology outlines the steps taken to prepare the data, define the problem, select and train models, tune hyperparameters, evaluate performance, and compare results, ultimately contributing to the development of an efficient RL-based trading system. Figure 3.1 shown the Research Framework for the research.

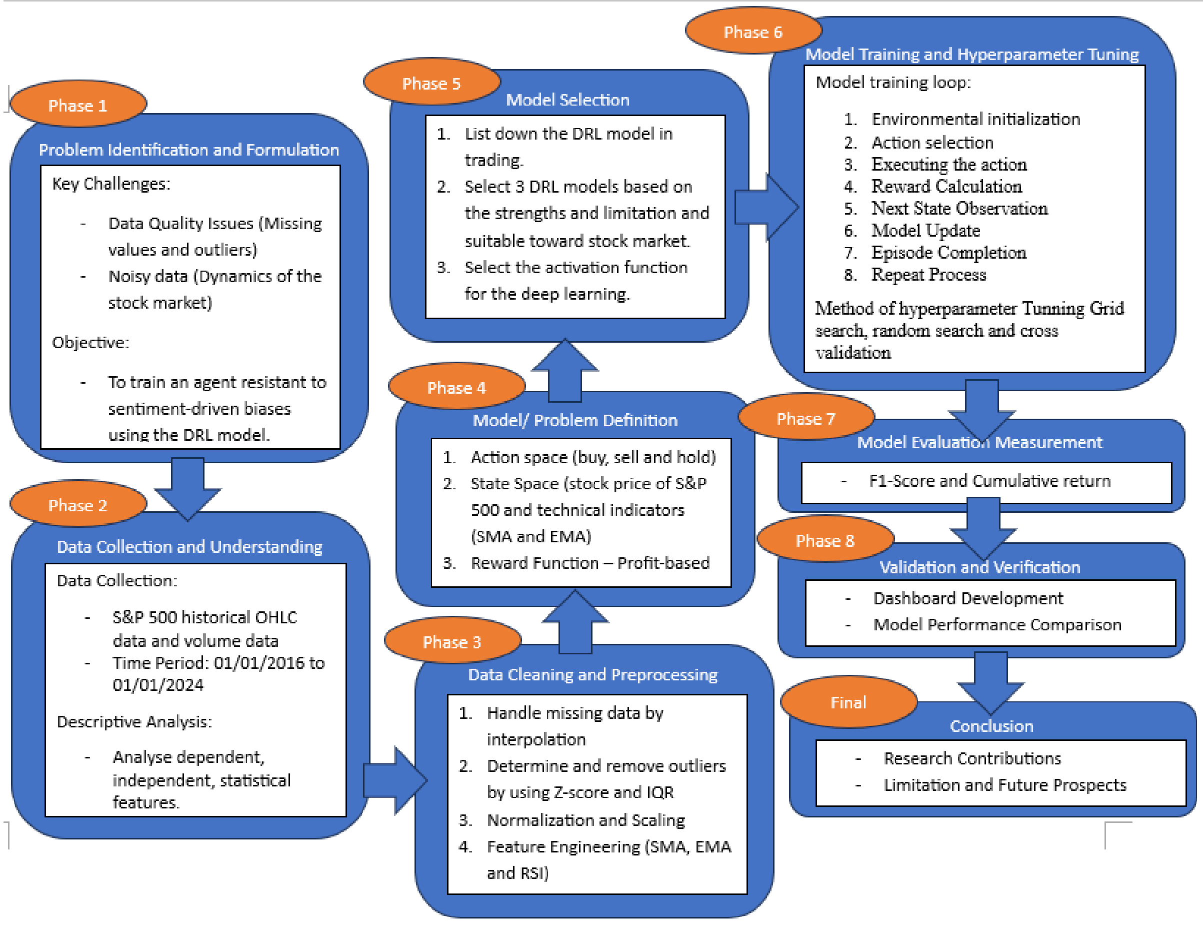

Phase 1
Problem Identification and Formulation
Key Challenges:
- Data Quality Issues (Missing values and outliers)
- Noisy data (Dynamics of the stock market)
Objective:
- To train an agent resistant to sentiment-driven biases using the DRL model.
Phase 2
Data Collection and Understanding
Data Collection:
- S&P 500 historical OHLC data and volume data
- Time Period: 01/01/2016 to 01/01/2024
Descriptive Analysis:
- Analyse dependent, independent, statistical features.
Phase 3
Data Cleaning and Preprocessing
1. Handle missing data by interpolation
2. Determine and remove outliers by using Z-score and IQR
3. Normalization and Scaling
4. Feature Engineering (SMA, EMA and RSI)
Phase 4
Model/ Problem Definition
1. Action space (buy, sell and hold)
2. State Space (stock price of S&P 500 and technical indicators (SMA and EMA)
3. Reward Function – Profit-based
Phase 5
Model Selection
1. List down the DRL model in trading.
2. Select 3 DRL models based on the strengths and limitation and suitable toward stock market.
3. Select the activation function for the deep learning.
Phase 6
Model Training and Hyperparameter Tuning
Model training loop:
1. Environmental initialization
2. Action selection
3. Executing the action
4. Reward Calculation
5. Next State Observation
6. Model Update
7. Episode Completion
8. Repeat Process
Method of hyperparameter Tunning Grid search, random search and cross validation
Phase 7
Model Evaluation Measurement
- F1-Score and Cumulative return
Phase 8
Validation and Verification
- Dashboard Development
- Model Performance Comparison
Final
Conclusion
- Research Contributions
- Limitation and Future Prospects

Figure 3-1 Research Framework

3.2 Data Cleaning and Pre-Processing

Data cleaning and pre-processing is to handle the missing data, remove outliers, normalization and scaling of data to make the data more readable and enable to analyzed to gain more accurate insight.

First step for the data cleaning process is to handle the missing data which means the missing values can be interpolation to make the complexity of the dataset. Missing data is the common issues for the time series data. Those gaps might be will affect the model's training. Interpolation will be applied to handle the missing values in the dataset of the S&P 500. After that, the outliers need to eliminated by using the statistical methods to identify the outliers which is using the Z-score or IQR. Remove outliers is to ensure the model will not train based on the irregular/unrepresentative data. There are the Z-score and IQR formula.

$$Z = \frac{X - \mu}{\sigma} \tag{3.2}$$

Where:

Z = Z-score

X = Value of the data point

μ = mean of the dataset

σ = Standard deviation of the dataset

$$IQR = Q_3 - Q_1 \tag{3.3}$$

Where:

Q3 = 75th percentile

Q1 = 25th percentile

Formula of IQR for the outlier detection,

(3.4)

$$Lower\ Bound = Q_1 - 1.5IQR$$

$$Upper\ Bound = Q_3 - 1.5IQR$$

If the data point is below lower bound or above the upper bound are the outliers. The outliers may affect the model as the misguided to learn of absurd patterns and lead to low accuracy.

Since the S&P 500 stock price is too large which makes the comparison meaningless. The normalization and scaling are importance to make DRL able to learn or calculate the results that approaches to intrinsic pattern of the data. Normalization and scaling are to ensure the features are on the same scale and able to improve the training of the model. Formula of the scaling is:

(3.5)

$$Scaled\ value = \frac{X - \min(X)}{\max(X) - \min(X)}$$

Where:

X = original data point

min(X) = minimum value of the data

max(X) = maximum value of the data

Scaled value is the scaled data point in the range [0,1].

3.3 Model/Problem Definition

The aims of the research is the application of deep reinforcement learning (DRL) techniques to develop an automated trading agent to make the decision toward the stock market. S&P 500 will be the input data for the research. With the Markov Decision Process (MDP) theory, the system designed to let the agent that can learns to make the optimal trading decisions which is buy, hold and sell based on the historical stock data to optimize the cumulative return rate over time.

The problem including the state space, action space and reward function. For the state space S_t, where the S_t is defined from the S&P 500 data such as OHLC (Open, high, low and close), closing price, moving average, relative strength index (RSI) and volume. The state space is the various features from the S&P 500 as the input of the DRL models to let the agent can learn the policy that dictates the action to take at each of the time step to optimize the return rate over time.

The action space for the research is discrete state which is the buy, hold and sell. Those discrete actions are representing the action taken for the agent can make at each time step as the decision making of trading. The reward function for the model will be evaluated with the cumulative rewards over time which is the financial gain or loss from the specific actions. The reward function R_t at each of the time step as following:

$$R_t = P_{t+1} - P_t \tag{3.6}$$

Where:

R_t is the reward function at the time, t

P_t is the price of the stock at the time, t

P_{t+1} is the price of the stock at the next time step

The aims of the agent are to optimize the cumulative reward over time with using the discounted return G_t to sum up the future rewards, discounting with the factor γ (where $0 \leq \gamma \leq 1$) to prioritize the immediate rewards.

The performance of the DRL models will be evaluated with their profitability and risk adjusted return as the reward function. By comparing each of the models against the baseline strategy such as buy and sell strategy. The research is to determine which of the model is most effective at the learning a trading policy that performs well toward the S&P 500 data for the long-term stock market strategy. The application of DRL for the automated trading ensure contribute toward the development of more efficient and profitable trading in decision making.

3.4 Model Selection

In the chapter 2, found some of the DRL models that are able to use in the trading system. Based on the chapter 2, the summary table of DRL model in the trading. Deep Reinforcement Learning (DRL) has revolution of the trading algorithms by allowing/train an agent to learn optimum trading policy from the dynamic and complex financial market through trial and error. The reward mechanism was the marking system that based on the policy set to give the reward or penalty toward an agent's actions. There are the summary table of DRL model including their function, strength, limitation and finding for selecting the best 3 model to further research.

Table 3-1 Summary of DRL model in trading

No	Model Name	Function of model in trading	Strength of model	Limitation of model	Results of paper	Citation
1	Deep-Q-Network (DQN)	By performing the deep neutral networks to calculate the Q-values, discrete trading actions (Sell, Hold, Buy) based on the mapping status (observations of market)	- Able to handles the high dimensional state spaces. - Without the handcrafted features, able to learn directly from the raw data (price of stock data)	- Trends to overestimate Q-values that may affect the grade of policy quality. - Only suitable for the discrete action spaces.	DQN-based trading outperformed rule-based strategies with higher cumulative returns and Sharpe ratios on historical stock data (Otabek et al., 2024).	Otabek et al., (2024). Multi-level deep Q-networks for bitcoin trading strategies. *Scientific Reports (Nature Publisher Group), 14*(1), 771. doi: https://doi.org/10.1038/s41598-024-51408-w

2	Proximal Policy Optimization (PPO)	By the surrogate clipped objectives to balance the exploration and exploitation. It will help in stable policy update for discrete and continuous spaces.	- Efficient of sample and robust to hyperparameter choices. - Well handle of stochastics policy and helps in improvement of exploration.	- intensive computation - May achieve the local optima without the adequate exploration.	PPO agents adapt well toward the dynamics market and outperformed than DQN and A2C models in portfolio management (Sun., 2023)	Sun, Q. (2023). *Reinforcement learning algorithms for stock trading* (Order No. 31765482). Available from ProQuest Dissertations & Theses Global. (3186188497). Retrieved from https://vpn.utm.my/dissertations-theses/reinforcement-learning-algorithms-stock-trading/docview/3186188497/se-2
3	Advantage Actor-Critic (A2C)	Jointed policy (optimal actor) and value (critic) networks to	- Better trade-off for bias-variance. - Convergence faster than pure policy gradient methods.	- Less sample-efficient than PPO. - Sensitive to 'Noisy' data of financial market.	A2C able to improve the returns rate over the baseline DRL	Goluža et al., (2024). *Deep reinforcement learning with positional context for intraday trading*. Ithaca: doi: https://doi.org/10.1007/s12530-024-09593-6

		reduce the variances of gradient and increase the speed of training convergence.			models (Goluža et al., 2024)	
4	Twin Delayed DDPG (TD3)	TD3 improved DDPG by using the double Q-learning clipped to reduce the overestimation bias and delayed of policy update for stability.	- More stable of training in continuous spaces - Better sample efficiency	- Sensitive to 'Noisy' data of financial market. - Additional computational overhead required.	TD3 achieved the higher cumulative returns and lower down the drawdowns than DDPG and PPO (Majidi et al., 2022)	Majidi et al., (2022). *Algorithmic trading using continuous action space deep reinforcement learning*. Ithaca: Retrieved from https://vpn.utm.my/working-papers/algorithmic-trading-using-continuous-action-space/docview/2723274890/se-2

5	Soft Actor-Critic (SAC)	SAC is the off-policy actor-critic algorithm that can help in optimize the maximum entropy objective and will helps in exploration and robustness	- Robust toward hyperparameter variations - Have a strong exploration in continuous action spaces	- Complexity of computational - Tunning issues for financial data.	SAC agents overperforming than PPO and DDPG in the trading multiple assets with lower risk and high stability (Kong et al., 2023)	Kong et al., (2023). Empirical analysis of automated stock trading using deep reinforcement learning. *Applied Sciences, 13*(1), 633. doi: https://doi.org/10.3390/app13010633

Based on the Table 3.1, there are 5 DRL model that can used in trading but still not practical yet. Those 5 DRL models illustrate the rich diversity of approaches in automated trading. DQN model is values-based models which is excel in discrete decision but limitation in scalability. Actor-critic methods which are PPO, A2C and SAC are balance policy and value learning, handling continuous actions with the better flexibility. Twin delayed methods address the stability and market complexity issues. Despite advances, there are some of the common limitations including instability of training, sensitivity toward 'Noisy' data and computational overhead.

The strengths, limitations and functions for each of the models are main consideration of the model selection for further research. After reviewing Table 2.1, 3 best-suited models in automated stock market trading are selected as the models that will used in the research. The 3 DRL models are PPO, DQN and SAC.

Table 3-2 Summary of the models selected

No	Models	Reason
1	PPO	- Balance training stability and sample efficiency. - Able to handle discrete and continuous action spaces
2	DQN	- Able to handles the high dimensional state spaces - Basic DRL model
3	SAC	- Efficient Exploration - Sample efficient

Based on Table 3.2, there are the 3 models that selected as the research model. PPO able to balance the training stability and sample efficiency which is adaptive toward ‘Noisy’ data and non-stationary nature of stock market data. PPO able to handle discrete and continuous action spaces which is versatile toward different trading decision such as portfolio adjustments, buy or sell signals. Clopped objective able to prevent the large destructive policy updates which is able to reduce the risk of performance collapse. DQN model created as the basic of the DRL model that able to handles the high dimensional state spaces for the discrete action spaces with high effectively. With the Actor-critic structure allows agents to learn the complex policy while evaluating for the next better updates. SAC able efficient exploration which is important to avoid the premature convergence in the dynamics/volatile financial market. With the off-policy training, sample efficient where allowing agents learn faster from the historical data.

The activation functions used in the hidden and output layers lies for the core of any neutral network-based DRL model. Those functions introduce non-linearity which is enable the network to learn the complex/dynamic patterns in the dataset such as the market behaviour and price movements. For the research, the Tanh (hyperbolic tangent) and sigmoid activation function selected for the underlying for neural network architectures that used for DRL models selected which is Deep Q-Network (DQN), Soft Actor-Critic (SAC), and Proximal Policy Optimization (PPO).

The Tanh function is a sigmoidal function which is the outputs values will be the range [-1,1]. Tanh function commonly used for the DRL because of ability to produce zero-centered outputs. Tanh function helps to maintain the balance during the weight updates and prevent the skewed gradients issues.

Formula Tanh function:

$$Tanh\ (x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.7}$$

Where the x is the input to the function which is the weighted sum of the inputs in the neural network layer.

The advantages of Tanh are zero-centered output, smooth gradient and efficient weight updates. Zero-centered output is the range from -1 to 1 and it helps to centering the data and enable to improve the convergence speed during training. Smooth gradient in the Tanh function is to make the suitable for backpropagation. Efficient weight updates to let the training becomes more balance due to the values are centered toward zero and helps to prevent the large oscillations in the process of learning. The challenges of Tanh also quite significant because of the vanishing gradient problem which means if the values is too large or too small, the gradient of Tanh will be very small and it will affect the network hard to updates efficiently during backpropagation. It will slow down the training process during the deep network.

The sigmoid activation function is one of the sigmoidal activation function which is the output values in the range of 0 to 1. It will help to map the values to probability space and used for the output layer of classification models. In the DRL models, sigmoid function able to applied where the output needs to represent probabilities or called as when the models required to make the binary decision (such as buy or sell).

The formula of sigmoid function:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{3.7}$$

Where:

S(x) is the output of the sigmoid function (range of values from 0 to 1)

e is the Euler’s number (around 2.718)

x is the input of the function

The advantages of the sigmoid are probabilistic output and smooth gradient. Probabilistic output as the output values only will between 0 and 1 which is can be interpreted as probabilities. It can help to predict the binary outcomes to make the decision either sell or buy. Smooth gradient of the sigmoid function like Tanh function, which is works well with backpropagation. The challenges of sigmoid function are not zero-centered and vanishing gradient problem. Sigmoid function unlike Tanh function because of the not zero centered which is the output value only 0 to 1 instead of Tanh -1 to 1. It will lead to skewed gradient during backpropagation. The convergence slowdown may happen as the updates to the weights can be biased toward the positive values. Another challenge of sigmoid function is similar with Tanh function which is the vanishing gradient problem where the input data is too large or too small may affect the deep network to learn effectively during training.

3.4.1 Deep-Q-Network (DQN) Model

Deep Q-Network (DQN) is the model-free, off-policy algorithm that can be used in deep reinforcement learning (DRL) which is combination of the Q-learning with the deep neutral networks to approximate the Q-function. DQN suitable for the large state spaces such as the dynamics market data set. In DQN, the neural network that used to approximate the Q-values for state-action pair.

The aim of DQN is to learn the optimal policy π∗(s) which is maximize the cumulative reward over time with keep updating Q-values by using deep neural network. The Q-value for the state-action pair as following:

(3.8)

$$Q(s_t, a_t) = E[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t, a_t]$$

Where:

s_t is the state at time, t

a_t is the action taken at time, t

r_{t+k} is the reward at time, t + k

γ is the discount factor which is the range ($0 < \gamma \leq 1$)

The purpose of the formula is to find a policy that can optimize the Q-values over time.

DQN able to utilize the Q-learning with the deep neural network to approximate the Q-values. The methodology for DQN algorithm will be as following:

1. Initialize Parameters and network
    a. Initialize Q-network. The neural network will initialize with the random weights to estimate the Q-value function. As the state s_t as the input and the output of the Q-values for all possible actions.

$$Q(s,a;\theta) \tag{3.9}$$

    Where θ is the weights of the neural network

    b. Initialize Target Q-network is the creation of the copy Q-network as the target Q-network which is initialized with the same weights.
2. Interact with the environment
    a. The agent observes the state, s_t of the environment at the each of time step, t
    b. The agent will select the action a_t based on the policy. The epsilon-greedy policy will select as initial policy where the agent will explore

more on the random actions with the probability, ϵ and the exploits the optimal action based on the Q-values with the probability 1-ϵ.

$$a_t = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \arg\max_a Q(s_t, a; \theta) & \text{with probability } 1 - \epsilon \end{cases}$$

Figure 3-2 Interact with the environment

3. Experience Replay
    a. Store the tuple which is (s_t, a_t, r_t, s_{t+1}) in the replay memory experience
    b. The replay memory experience helps to break the correlation between consecutive samples and it allowing the network to learn more effectively.
4. Sample Mini-Batch and Update Q-Network
    a. The random mini-batch of experiences (s_t, a_t, r_t, s_{t+1}) at each time step is sampled from the replay memory experience
    b. Targeted Q-network calculated as following:

(3.10)

$$y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-)$$

Where θ^- is the targeted weights of the neural network

    c. Loss function. The Mean squared error (MSE) between the predicted Q-value and the target was calculated as following:

(3.11)

$$L(\theta) = E\,[(y_t - Q(s_t, a_t; \theta))^2]$$

d. Update Q-network. By using the gradient descent to minimize the loss and update the Q-network weights as following:

$$\theta \leftarrow \theta - \alpha \nabla \theta L(\theta) \tag{3.12}$$

Where α is the learning rate.

5. Update Target Network. The target Q-network updated to match with the Q-network.
6. Repeat the Process for the interaction with environmental, replay memory experience and Q-network updates for each of the time step until the agent converges or a stopping criterion.

The key hyperparameters in DQN as following:

1. Discount Factor, γ. To determine/identify the importance of future rewards.
2. Learning Rate, α. To determine/identify the weights of the network are adjusted during training.
3. Epsilon, ϵ. The probability of select exploration (random action) rather than exploitation (current policy)
4. Batch Size. The numbers of samples from replay memory fir each update step.
5. Replay Memory Size. The maximum number for experiences stored in memory.
6. Target Network Update Frequency. The frequency for the targeted network is updated to match with Q-network.

The formula of the Q-values updates at each time step:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-) - Q(s_t, a_t)] \quad (3.13)$$

Where:

$Q(s_t, a_t)$ is the current Q-value

r_t is the reward received after action taken, a_t

γ is discount factor,

$\max_{a'} Q(s_{t+1}, a'; \theta^-)$ is the target Q-value form the target network

θ^- is the targeted weights of the neural network

After the training, the evaluation of DQN model required to check the performance of the DQN model. The evaluation will be including test the policy and performance metrics. Test the policy is to check the performance of agent on the learned policy at the test environmental. With the testing, will identify the valines of the policy. The cumulative reward over time as the performance metrics to evaluate the DQN model after training. The Figure 3.3 showed architecture of DQN.

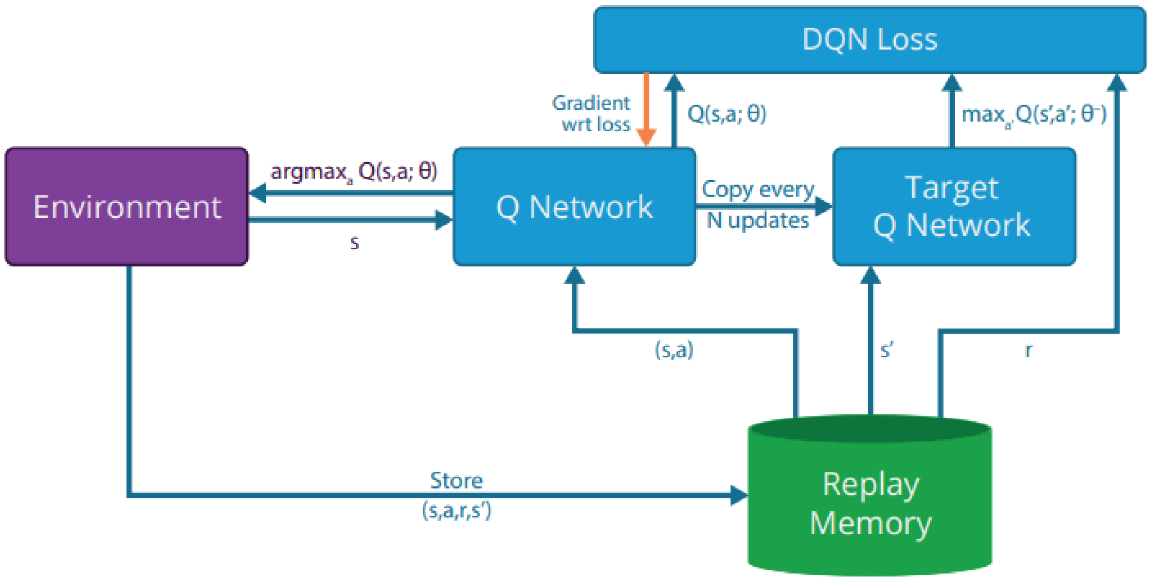


Figure 3-3 DQN architecture

3.4.2 Soft Actor-Critic (SAC) Model

SAC is a DRL algorithm which primarily used in robotics field. The algorithm aims to find the optimal policy that can obtain that maximum expected long-term reward and long-term entropy. The Equation 3.1 stated below indicate the logic behind this algorithm.

$$J(\pi_\theta) = E_{\pi_\theta}[\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t) + \alpha H\big(\pi(.|s_t)\big)] \tag{3.14}$$

The learning of the state-value function stated as Equation 3.15 is done through minimizing the squared residual error. Then it takes the first order of derivative of that state-value function to get the gradient shown as Equation 3.16.

(3.15)

$$J_V(\psi) = \mathbb{E}_{s_t \sim D}[\frac{1}{2}(V_\psi(S_t) - \mathbb{E}_{a_t \sim \pi_\phi}[Q_\theta(s_t, a_t) - \log \pi_\phi(a_t|s_t)])^2]$$

(3.16)

$$\hat{\nabla}_\psi J_V(\psi) = \hat{\nabla}_\psi V_\psi(s_t)(V_\psi(S_t) - Q_\theta(s_t, a_t) + \log \pi_\phi(a_t|s_t)$$

Then the learning of Q-function, which stands for quality function here is stated as Equation 3.17 below, and the next step Q-function is stated as Equation 3.18. The gradient for the learning can be obtained through Equation 3.19 below.

(3.17)

$$J_Q(\theta) = \mathbb{E}_{(s_t,a_t) \sim D}[\frac{1}{2}(Q_\theta(s_t, a_t) - \hat{Q}(s_t, a_t))^2]$$

(3.18)

$$\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p}[V_{\bar{\psi}}(s_{t+1})]$$

(3.19)

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(a_t, s_t)(Q_\theta(s_t, a_t) - r(s_t, a_t) + \gamma V_{\bar{\psi}}(s_{t+1}))$$

Then, the learning of the policy can be done through the Equation 3.20, and the gradient can be calculated through Equation 3.21.

(3.20)

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim D, \epsilon_t \sim N}[\log \pi_\phi(f_\phi(\epsilon_t; s_t)|s_t) - Q_\theta(s_t, f_\phi(\epsilon_t; s_t))]$$

$$\hat{\nabla}_{\phi} J_{\pi}(\phi) = \nabla_{\phi} \log \pi_{\phi}(a_t|s_t) + (\nabla_{a_t} \log \pi_{\phi}(a_t|s_t) - \nabla_{a_t} Q(s_t, a_t)) \nabla_{\phi} f_{\phi}(\epsilon_t; s_t) \tag{3.21}$$

The Figure 3.4 and Figure 3.5 showed the basic reinforcement learning for one time and architecture of actor critic

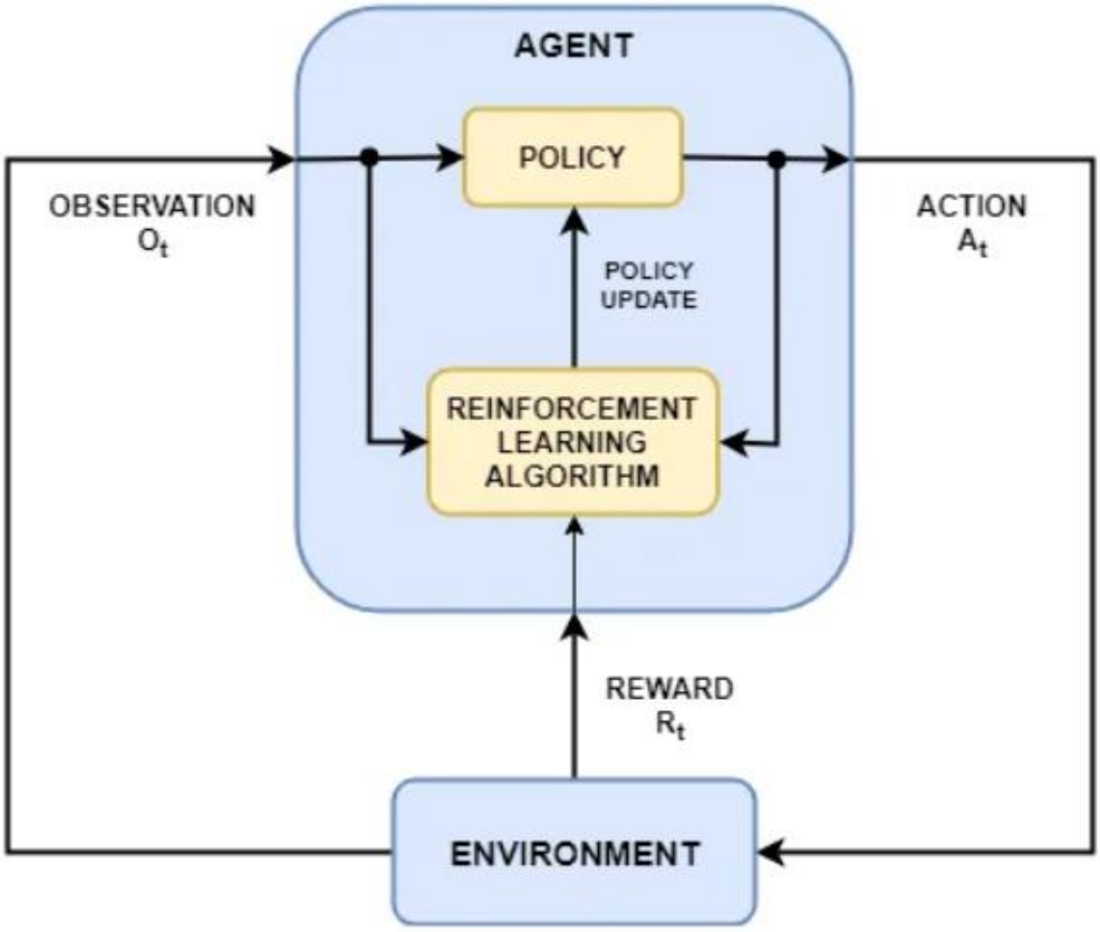


Figure 3-4 Basic reinforcement learning structure

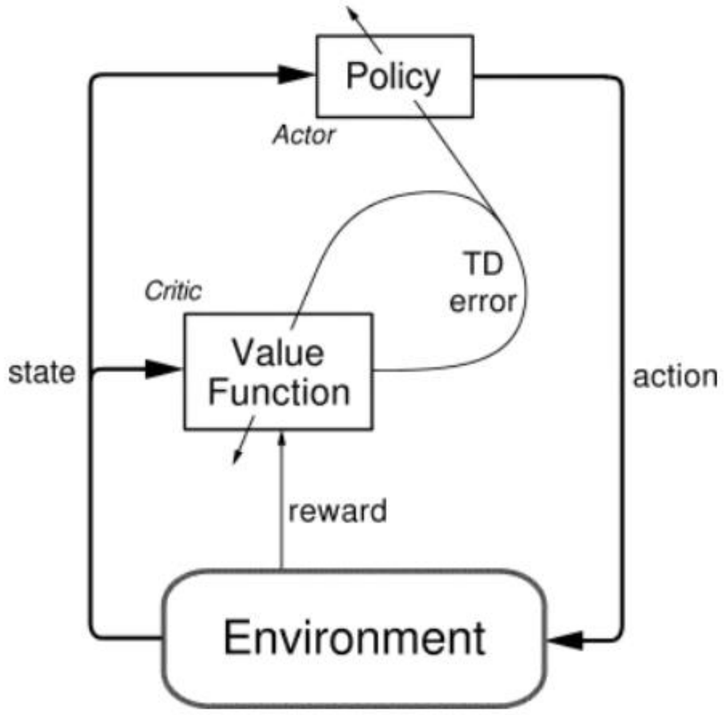


Figure 3-5 Soft-Critic architecture.

3.4.3 Proximal Policy Optimization (PPO) Model

Proximal Policy Optimization (PPO) is the DRL model with the policy gradient method that have developed by OpenAI. PPO able to prevent the large policy updates which is balance the exploitation and exploration. For the algorithm of PPO is actor-critic types which is maintaining actor(policy) and critic(value) in the same times. Actor is the policy network which is the action taken by given a state. Critic is value network which is estimate the value of the state. PPO uses a surrogate objective function that will avoid the large changes in policy to make the stability of learning process.

The Clipped surrogate objective as following:

$$L^{CLIP}(\theta) = E_{\mathrm{t}}[\min(r_t(\theta)A_t, \mathrm{clip}(r_t(\theta), 1-\in, 1+\in)A_t] \tag{3.22}$$

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta-\mathrm{old}}(a_t|s_t)} \tag{3.23}$$

Where:

$\pi_{\theta-\mathrm{old}}$ is the old policy before update

π_θ is the policy parameterized by θ

ϵ is the small hyperparameter to limit updates

A_t is the advantage function at time step, t

$r_t(\theta)$ is the probability ratio

Total loss function as following:

$$L(\theta) = E_{\mathrm{t}}[L^{CLIP}(\theta) - c_1 \,.\, (V_\theta(s_t) - R_t)^2 + c_2 . S[\pi_\theta](s_t)] \tag{3.24}$$

Where:

$V_\theta(s_t)$ is the estimated value function

R_t is the cumulative return

$S[\pi_\theta]$ is the entropy bonus for exploration

c_1 , c_2 is the coefficients to balance loss terms

The methodology for PPO algorithm will be as following:

1. Environmental setup
    a. Identify the state space and action space
2. Initialize Neural network
    a. Initialize the actor and critic networks with random weights
    b. Separate the network parameters depending on design
3. Data collection
    a. Collect N at the time steps for the data by run the policy in environment
    b. The transition store same as DQN (s_t, a_t, r_t, s_{t+1})
4. Advantage estimation
    a. By using generalized advantage estimation (GAE) to calculate the A_t

(3.25)

$$A_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \ldots\ldots$$

(3.26)

$$\delta_t = r_t + \gamma V\,(s_{t+1}) - V(s_t)$$

5. Policy and value update
    a. Calculate the surrogate loss by using clipped objective
    b. Update the policy and value network by using the stochastic gradient descent
    c. Repeat the several epochs over mini-batches of collected data
6. Repeat process
    a. Continue repeat the step 1-5 to achieve the performance threshold.

The hyperparameters for the PPO including:

- Discount Factor, γ. To determine/identify the importance of future rewards.
- Learning Rate, α. To determine/identify the weights of the network are adjusted during training.
- Clipping parameter, ϵ. The probability of select exploration (random action) rather than exploitation (current policy)
- Batch Size. The numbers of samples from replay memory fir each update step.
- GAE parameter, λ. To control weighting of different multi-step estimates

Figure 3.6 shown the architecture of PPO.

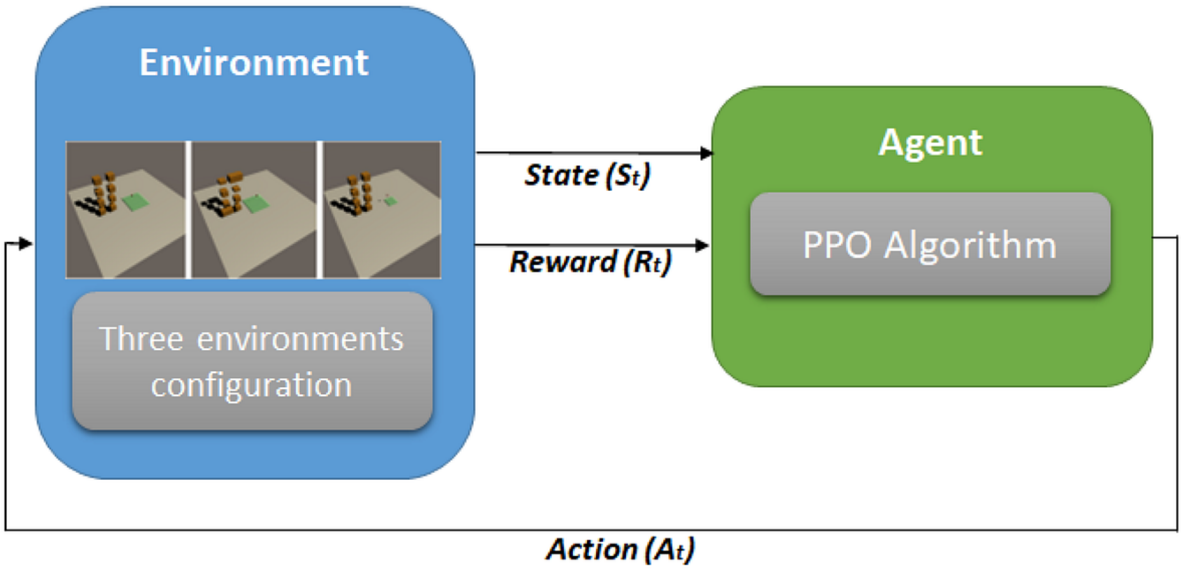


Figure 3-6 PPO architecture.

3.5 Model Training

3.5.1 Feature Engineering

The feature engineering will apply after data preprocessing completed. The feature engineering including as following:

- Raw features – historical price data (OHLC-Open, High, Low, Close) and volume (daily trading record)
- Time-based Features – Time of the day
- Normalization – the features will normalize to 0 to 1 to avoid the hinder model of learning.
- Technical indicators are moving averages (simple Moving Average (SMA) and Exponential Moving Average (EMA) and RSI

3.5.2 Data Splitting

The dataset will split to 3 session which is training data, validation data and test data. The percentage distribution of the dataset will be 80% for training data set, 20% for test data set. Training data is the 80% of available data for train model. Validation data is for hyperparameter tuning. Test data is to evaluate the model generalization after training and validation.

3.5.3 Deep Reinforcement Learning (DRL) Model Training Loop

The DRL model training loop can divide as following:

I. Environmental Initialization is the state space initialize as the environmental from the input data.

II. Action selection
   a. DQN – By using Q-network to approximate the Q-values, model will take the action based on the Q-values. The Q-values is the expected future reward for each time step in a given state. The model follows the epsilon-greedy strategy which is exploration in random action and exploitation is chose the best action.
   b. PPO – The model the select the action based on the policy network. The action taken is sampled from the probability distribution which is the

updated policy based on the advantage function. PPO will using the clipped objective function to enable the stability of training process and avoid the large changes in policy.

   c. SAC – The model will select the action based on the sampling from the policy distribution and the impact of entropy to encourage exploration. The policy updated to optimize the trade-off between expected reward and entropy.

III. Executing the action is the model will perform the action based on the selection at II which is buy, sell or hold.

IV. Reward Calculation. The reward will be calculated based on the profit or loss of the action taken.

V. Next State Observation: The new market will be observed after the executing action (step III)

VI. Experience Replay (for DQN only): The experience replay buffer to store states, actions, reward and next state. The random sample of the experiences from the buffer is used to train the Q-network.

VII. Model Update

   a. DQN. By apply the Bellman equation to adjust the Q-values to minimize the difference between predicted and actual rewards.
   b. PPO. By optimizing the clipped objective function where to balance the current and new policy to avoid large updates.
   c. SAC. By using soft bellman backup equation with the entropy regularization to encourage exploration for the policy network and values network.

VIII. Episode Completion. After passed the predefined number of the time steps, will be terminated to allow the model proceeds to the next time steps.

IX. Repeat process. The process will be repeated in multiple time steps to improve the policy of model and reward received based on the action.

3.5.4 Bias-Variance Trade Off

The bias-variance trade off will be consider during the model training. The high-bias may fail to capture the market trend (dynamic) which is the underfitting of the data and lead to poor decision making. The model with high bias because of the architecture too simple or the training data unable to provide sufficient features. A high variance model is overfitting of the data and leading to overperformance in training but poor in generalization of unseen data. The overfitting because of model is too complex. Figure 3.7 shown the bias-variance trade off.

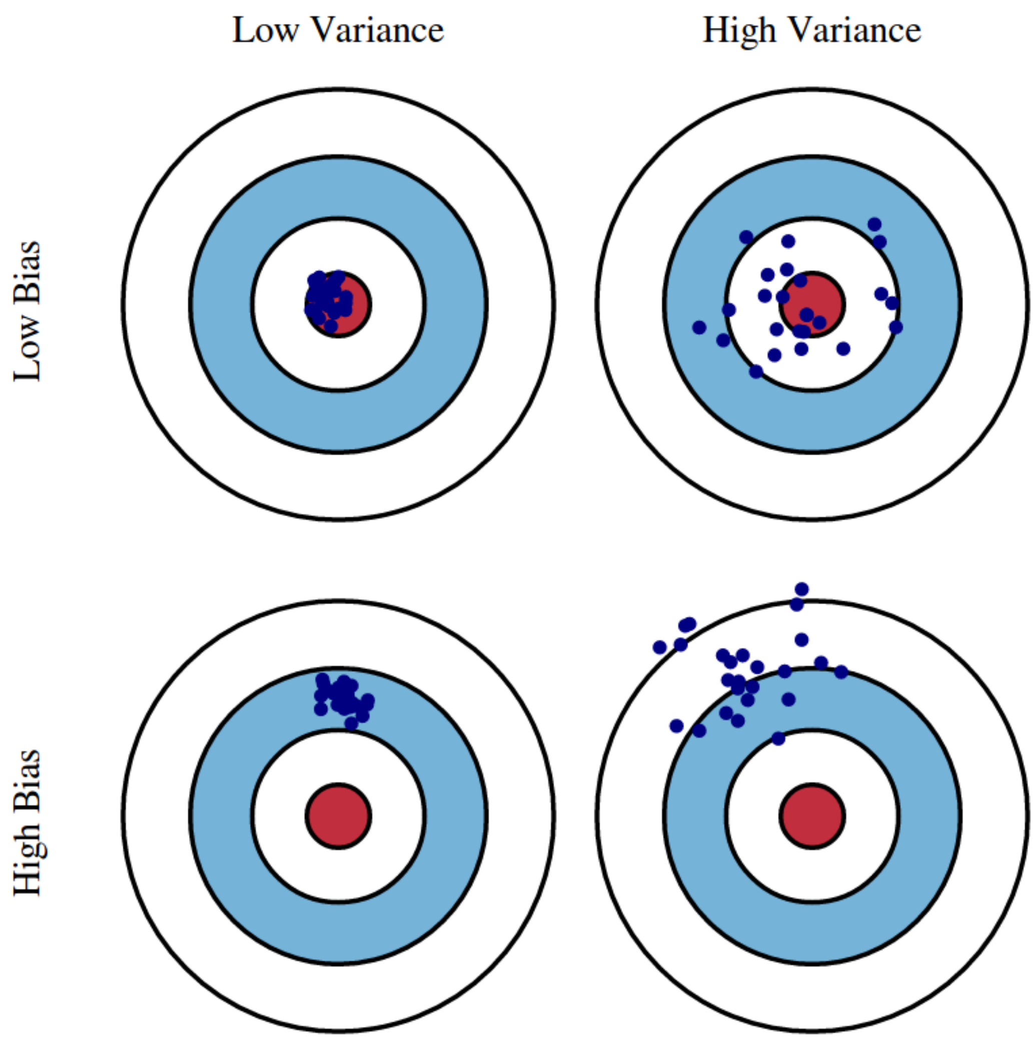


Figure 3-7 Summary of the bias-variance trade off

The methods of handling bias-variance trade-off is regularization techniques which is the weight decay and using to mitigate high variance issues. Early stopping based on the performance in the validation set will help to avoid overfitting issues. Exploration strategy will ensure the model not too greedy and exploitative too early, it will help to maintain to balance between bias and variance.

3.6 Hyperparameter Tuning

The general hyperparameter for DQN, PPO and SAC as following:

- Learning rate, α
- Discount factor, γ
- Batch Size
- Exploration Rate
- Replay buffer size
- Number of timesteps per episode

The model-specific hyperparameters of each of the DRL model as following:

1. DQN.
    a. Target Network Update Frequency – Control the target Q-network is updated with the parameter of main Q-network
    b. Double Q-Learning – help to mitigate the overestimation bias in Q-values
2. PPO.
    a. Clip Range – Prevent large changes in policy updates. It will help to increase the stability of the training process
    b. GAE lambda – generalized advantage estimation (GAE) to balance the bias and variance in estimation.
3. SAC

a. Entropy Coefficient – Control the trade-off of exploration and exploitation to adding entropy regularization to objective function. It will help to increase the exploration.
b. Target Entropy – To determine the exploration of the model during training.

The hyperparameter tuning methods can be divided into Grid search, random search, Bayesian optimization and cross-validation. The grid search is the systematically tries all the possible combination of the hyperparameter values within the predefined range. The grid search method is simple to implementation but computationally expensive due to exhaustive nature of the search. The step of the grid search as following:

1. Define the range for each hyperparameter (learning rate, discount factor and batch size)
2. Create the grid of all possible combination of hyperparameter values.
3. Train the model for all the possible combination of hyperparameter and evaluate the performance on the validation data by using the metrics such as total reward, Sharpe ratio and maximum drawdown.
4. Select the best performance of the combination for the hyperparameter on the validation set.
5. Select the combination with fine-tune.

Random search is select the combination of hyperparameter randomly from the search state. The cost of the random search is lower than grid search but not guarantee can get the optimal hyperparameter data set. The step for random search as following:

1. Define the range for each hyperparameter (learning rate, discount factor and batch size)
2. Search the sample hyperparameter values randomly.
3. Train the model based on the random hyperparameter combinations and evaluate the performance of the model on the validation data set.

4. Select the best performance of the combination hyperparameter based on the validation set.

Cross-validation is the generalization of model ability and avoid the overfitting during hyperparameter tuning. The data set will split into k-folds and the model can be train and evaluate on each of the fold. The methods able to provide the better estimate of the model's generalization ability but the higher cost of computational due to multiple training runs. The step of the cross-validation as following:

1. Split the training data into k folds
2. Train the model by using k-1 folds and validate for each of the fold.
3. Average the performance across all folds
4. Evaluation criterion for selecting hyperparameter by using the average performance

3.7 Evaluation Metrics/ Performance Measurement

Evaluation metrics is to assess the performance of DRL models (DQN, SAC and PPO). For this research, focus on the two critical performance measurement which is F1 score and cumulative return. F1 score is to measure the balance between precision and recall which can adapted for evaluate the decision of trading. Cumulative return is the measure the total return rate of the trading strategy making.

3.7.1 Accuracy Benchmark (F1-Score & Cumulative Return)

F1 score is used to evaluate the classification model for precision and recall phase. It will evaluate the performance of model in predicting buy and sell decision where the target with the minimum of false positive and false negatives.

$$\text{F1 Score} = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{3.27}$$

Where:

Precision measures how many of the predicted buy or sell action

Recall measures how many of the actual profitable buy or sell opportunity of the model.

Higher F1 score means that the precision and recall with the higher numbers of true negative or positive. Low F1 score means either the model has many false positive or negative, lead to poor trading decision.

Cumulative return is the financial metric which measures the total return of the trading strategy over a given period. The formula of cumulative return as following:

$$\text{Cumulative Return} = \frac{Final\ Portfolio\ value - Initial\ Portfolio\ value}{Initial\ Portfolio\ value} \tag{3.28}$$

Where:

Initial portfolio value is the starting value of the trading portfolio

Final portfolio value is the ending value of the portfolio

3.8 Dashboard Development

After the completion of the evaluation metrics/performance measures, dashboard development will be required to have the better visualization by using power BI. Dashboard is to monitor the performance of an automated trading system to enhance the output of the DRL models is more visualise. The purpose of the dashboard is to provide a user-friendly interface that display metrics such as cumulative return, F1 score, trade history and portfolio performance for the each of the DRL model which is DQN, SAC and PPO. It will help to get the real-time agent performance updates.

3.9 Model Comparison and Performance Analysis

After the dashboard developed, the performance comparison between DQN, SAC and PPO will be based on the cumulative return, F1-score and model stability and robustness over time. With the comparison between DQN, SCA and PPO, will help to identify the strengths and weaknesses of each model.

CHAPTER 4

INITIAL FINDINGS

4.1 Introduction

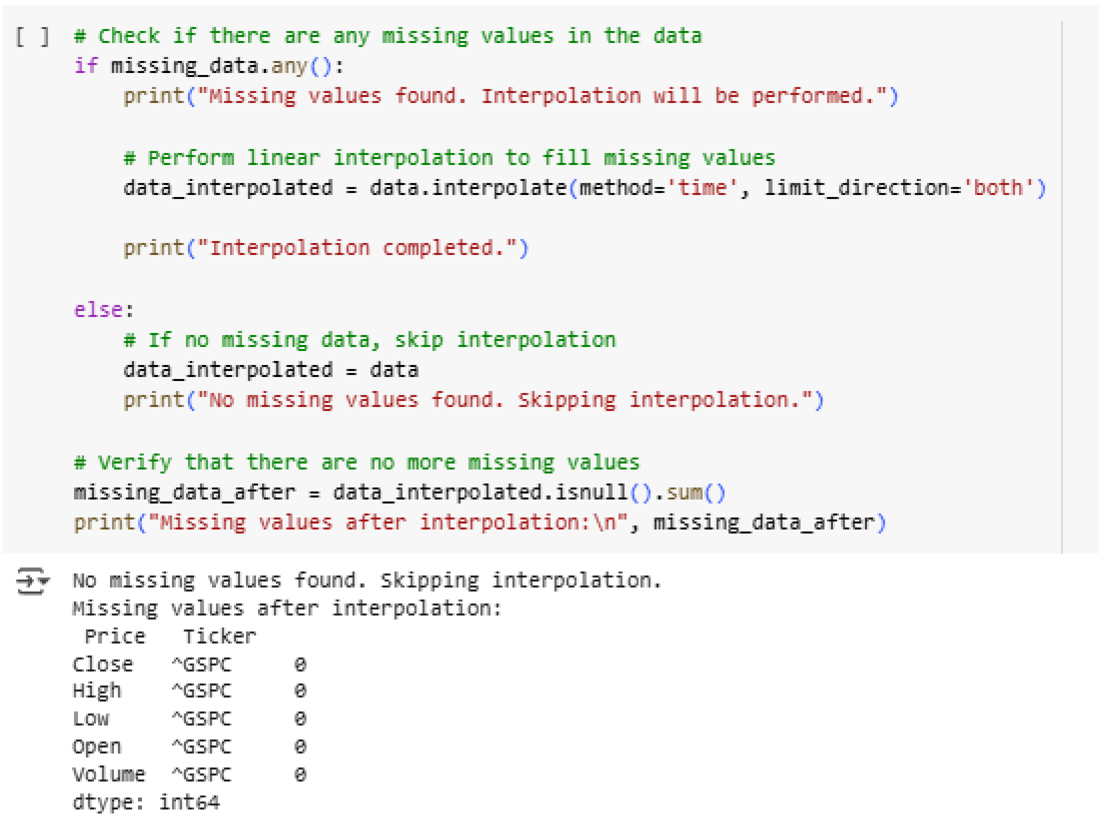
In the Chapter 4, the outlines covered the anticipated results and outcome from the implementation of deep reinforcement learning (DRL) for the automated trading in the stock market. Implementation of DRL in financial industry approach to its ability to optimize the trading strategy through the trial and error, learning based on the reward mechanism. In this research is to develop and evaluate the DRL-based trading system that able to autonomously make trading decision based on the real-time market data.

In the research includes performing of exploratory data analysis (EDA) to obtain the initial insights from the dataset (S&P 500) and understand the underlying trends and patterns. EDA is the essential process as can helps to identify the patterns of dataset, anomalies detection, formation of hypotheses and validate assumptions through the descriptive statistics and visual representations. The chapter will start with data preprocessing/data cleaning followed by EDA where including descriptive statistics employed to explore the data and visualization of the dataset. Those are able to provide the information for the feature engineering and the subsequent of training for the DRL models.

The expected outcome of the research is the creation of the robust DRL trading agents that can make the correct decision making based on the real-time market trends. EDA and feature engineering processes able to generate the guideline for the development of the model. Out of that, the potential challenges of the research such as volatility of the market, high adaptability in the different market conditions and overfitting will be addressed. In the expected outcome will including model development structure and the model evaluation. The model development structure and the model evaluation are the expected setting of the model will be used in the research.

4.2 Data Pre-Processing/Data Cleaning

The data will be cleaning by checking the missing data and outliers as the initial step. If the data consists of missing data will be handled with the interpolation method to make sure the data is not missing values as following as Figure 4.1.

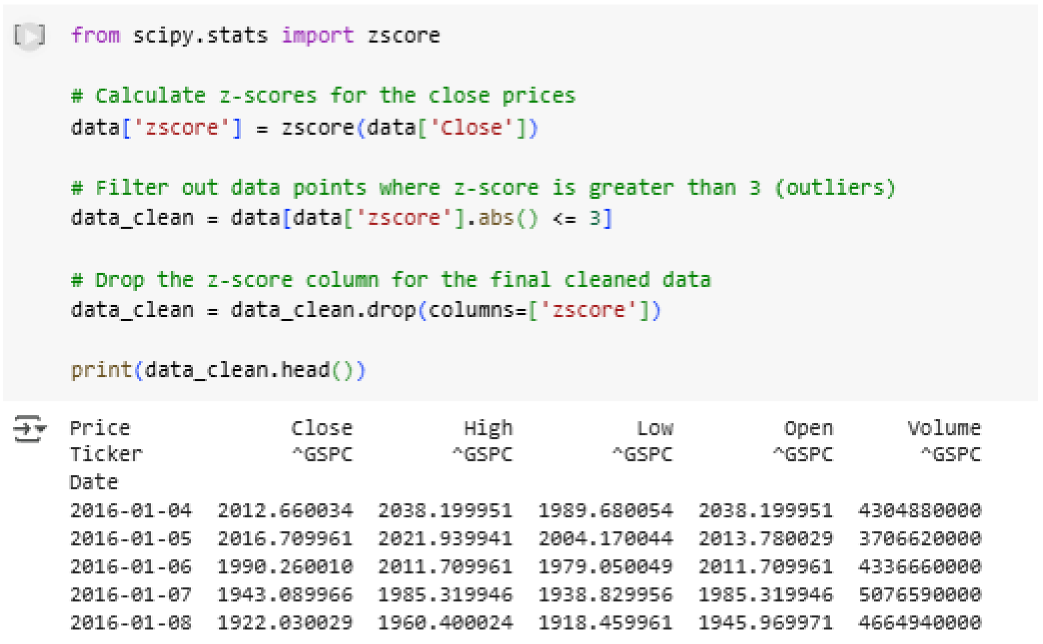


```
[ ] # Check if there are any missing values in the data
    if missing_data.any():
        print("Missing values found. Interpolation will be performed.")

        # Perform linear interpolation to fill missing values
        data_interpolated = data.interpolate(method='time', limit_direction='both')

        print("Interpolation completed.")

    else:
        # If no missing data, skip interpolation
        data_interpolated = data
        print("No missing values found. Skipping interpolation.")

    # Verify that there are no more missing values
    missing_data_after = data_interpolated.isnull().sum()
    print("Missing values after interpolation:\n", missing_data_after)
```

```
No missing values found. Skipping interpolation.
Missing values after interpolation:
 Price   Ticker
Close   ^GSPC     0
High    ^GSPC     0
Low     ^GSPC     0
Open    ^GSPC     0
Volume  ^GSPC     0
dtype: int64
```

Figure 4-1 Track of the missing values after interpolation

After that, the data will continue clean with remove outlier which is remove the abnormal data point to prevent the agents are learning the abnormal condition. The results after remove outliers as the Figure 4.2.

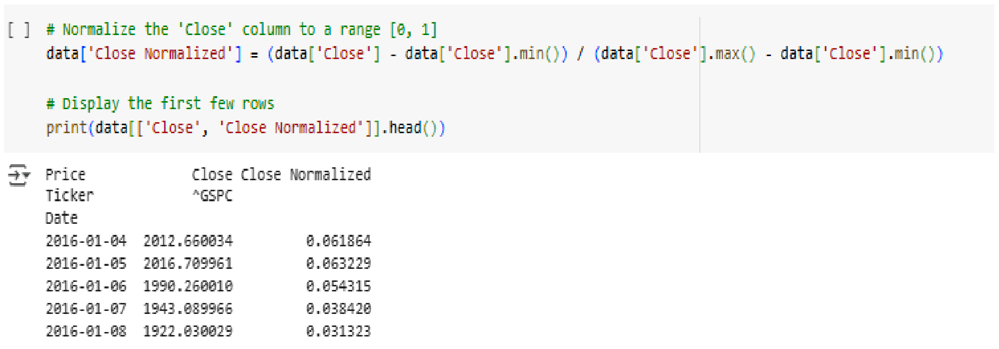


```
from scipy.stats import zscore

# Calculate z-scores for the close prices
data['zscore'] = zscore(data['Close'])

# Filter out data points where z-score is greater than 3 (outliers)
data_clean = data[data['zscore'].abs() <= 3]

# Drop the z-score column for the final cleaned data
data_clean = data_clean.drop(columns=['zscore'])

print(data_clean.head())
```

```
Price              Close         High          Low         Open       Volume
Ticker             ^GSPC        ^GSPC        ^GSPC        ^GSPC        ^GSPC
Date
2016-01-04   2012.660034  2038.199951  1989.680054  2038.199951   4304880000
2016-01-05   2016.709961  2021.939941  2004.170044  2013.780029   3706620000
2016-01-06   1990.260010  2011.709961  1979.050049  2011.709961   4336660000
2016-01-07   1943.089966  1985.319946  1938.829956  1985.319946   5076590000
2016-01-08   1922.030029  1960.400024  1918.459961  1945.969971   4664940000
```

Figure 4-2 Results after Remove Outliers

Normalize the feature of the ‘Close’ price to a range [0,1]. The Normalized ‘Close’ data will show at Figure 4.3.

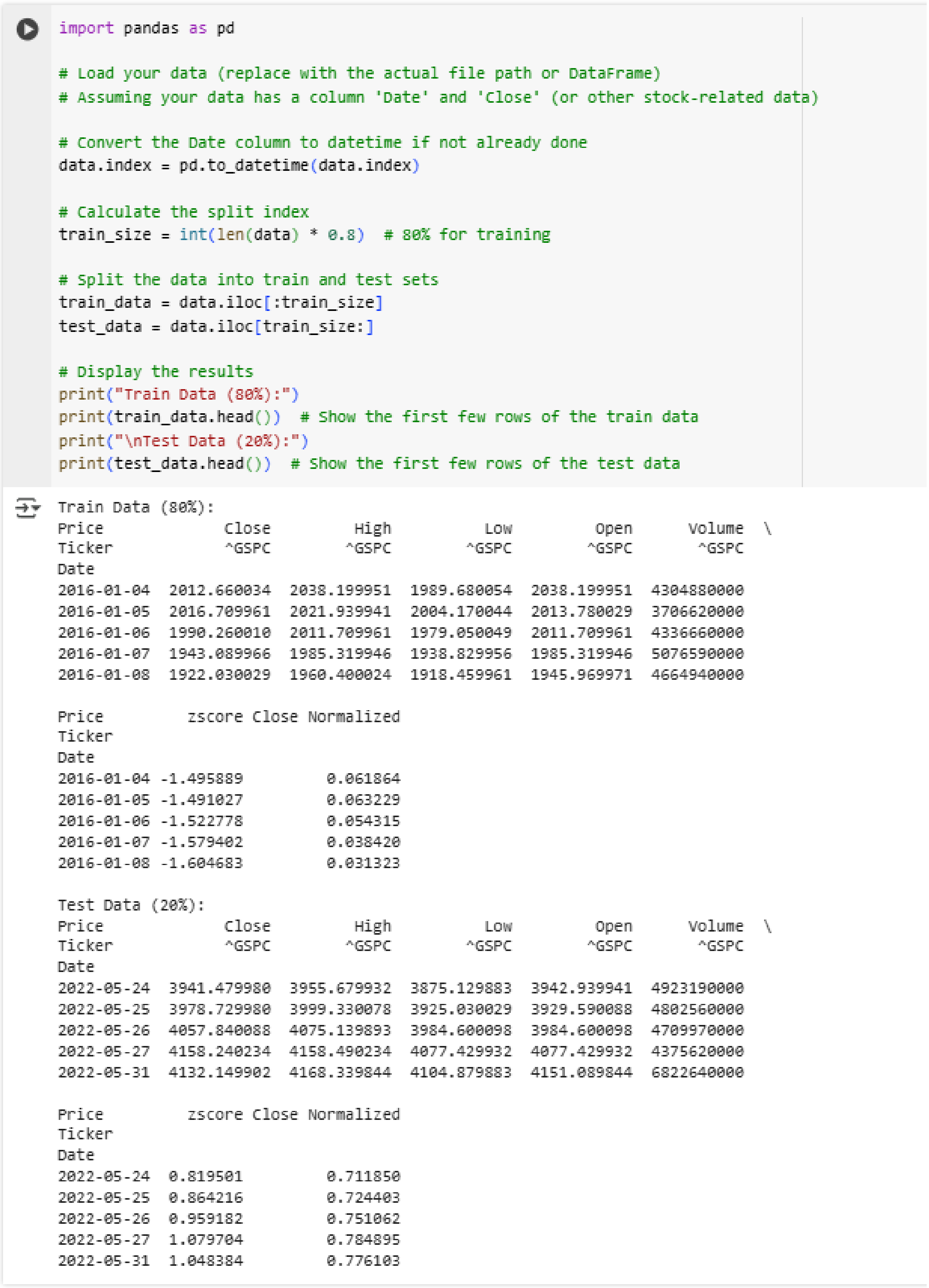


```
# Normalize the 'Close' column to a range [0, 1]
data['Close Normalized'] = (data['Close'] - data['Close'].min()) / (data['Close'].max() - data['Close'].min())

# Display the first few rows
print(data[['Close', 'Close Normalized']].head())
```

```
Price              Close Close Normalized
Ticker             ^GSPC
Date
2016-01-04   2012.660034         0.061864
2016-01-05   2016.709961         0.063229
2016-01-06   1990.260010         0.054315
2016-01-07   1943.089966         0.038420
2016-01-08   1922.030029         0.031323
```

Figure 4-3 Normalize the ‘Close’ column to a range [0,1]

The data will separate into training and test set which is 2016-2022 will be training set and 2023-2024 will be test set. The data will split as the Figure 4.4.

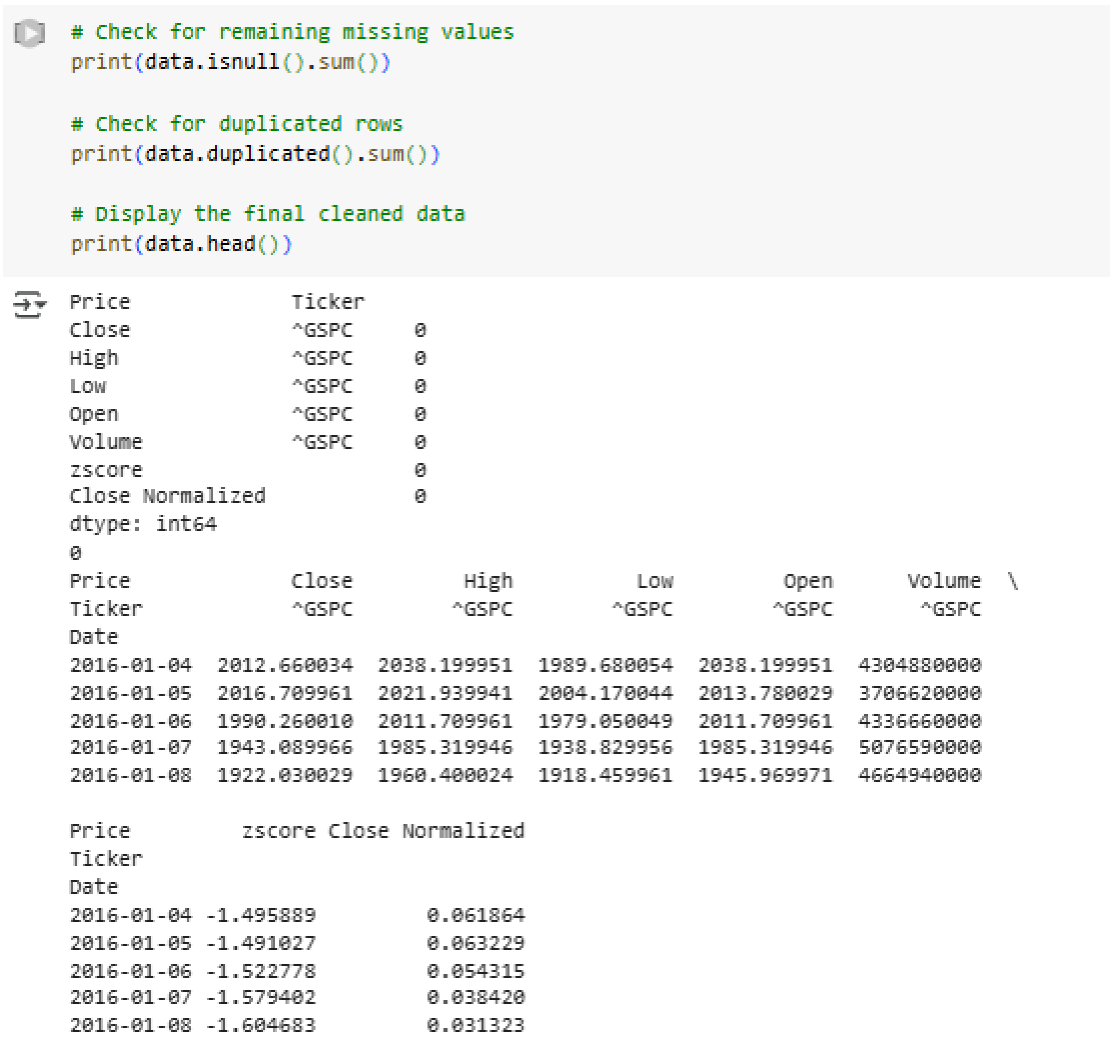


```python
import pandas as pd

# Load your data (replace with the actual file path or DataFrame)
# Assuming your data has a column 'Date' and 'Close' (or other stock-related data)

# Convert the Date column to datetime if not already done
data.index = pd.to_datetime(data.index)

# Calculate the split index
train_size = int(len(data) * 0.8)  # 80% for training

# Split the data into train and test sets
train_data = data.iloc[:train_size]
test_data = data.iloc[train_size:]

# Display the results
print("Train Data (80%):")
print(train_data.head())  # Show the first few rows of the train data
print("\nTest Data (20%):")
print(test_data.head())  # Show the first few rows of the test data
```

```
Train Data (80%):
Price             Close         High          Low         Open      Volume  \
Ticker            ^GSPC        ^GSPC        ^GSPC        ^GSPC       ^GSPC
Date
2016-01-04  2012.660034  2038.199951  1989.680054  2038.199951  4304880000
2016-01-05  2016.709961  2021.939941  2004.170044  2013.780029  3706620000
2016-01-06  1990.260010  2011.709961  1979.050049  2011.709961  4336660000
2016-01-07  1943.089966  1985.319946  1938.829956  1985.319946  5076590000
2016-01-08  1922.030029  1960.400024  1918.459961  1945.969971  4664940000

Price         zscore Close Normalized
Ticker
Date
2016-01-04 -1.495889         0.061864
2016-01-05 -1.491027         0.063229
2016-01-06 -1.522778         0.054315
2016-01-07 -1.579402         0.038420
2016-01-08 -1.604683         0.031323

Test Data (20%):
Price             Close         High          Low         Open      Volume  \
Ticker            ^GSPC        ^GSPC        ^GSPC        ^GSPC       ^GSPC
Date
2022-05-24  3941.479980  3955.679932  3875.129883  3942.939941  4923190000
2022-05-25  3978.729980  3999.330078  3925.030029  3929.590088  4802560000
2022-05-26  4057.840088  4075.139893  3984.600098  3984.600098  4709970000
2022-05-27  4158.240234  4158.490234  4077.429932  4077.429932  4375620000
2022-05-31  4132.149902  4168.339844  4104.879883  4151.089844  6822640000

Price         zscore Close Normalized
Ticker
Date
2022-05-24  0.819501         0.711850
2022-05-25  0.864216         0.724403
2022-05-26  0.959182         0.751062
2022-05-27  1.079704         0.784895
2022-05-31  1.048384         0.776103
```

Figure 4-4 Training and test dataset

After the preprocessing and data cleaning, will have the final check as the confirmation that data is fully cleaned. The cleaned data checking will be shows as Figure 4.5.



```
# Check for remaining missing values
print(data.isnull().sum())

# Check for duplicated rows
print(data.duplicated().sum())

# Display the final cleaned data
print(data.head())
```

```
Price             Ticker
Close             ^GSPC     0
High              ^GSPC     0
Low               ^GSPC     0
Open              ^GSPC     0
Volume            ^GSPC     0
zscore                      0
Close Normalized            0
dtype: int64
0
Price            Close         High          Low         Open      Volume  \
Ticker           ^GSPC        ^GSPC        ^GSPC        ^GSPC       ^GSPC
Date
2016-01-04  2012.660034  2038.199951  1989.680054  2038.199951  4304880000
2016-01-05  2016.709961  2021.939941  2004.170044  2013.780029  3706620000
2016-01-06  1990.260010  2011.709961  1979.050049  2011.709961  4336660000
2016-01-07  1943.089966  1985.319946  1938.829956  1985.319946  5076590000
2016-01-08  1922.030029  1960.400024  1918.459961  1945.969971  4664940000

Price         zscore Close Normalized
Ticker
Date
2016-01-04 -1.495889         0.061864
2016-01-05 -1.491027         0.063229
2016-01-06 -1.522778         0.054315
2016-01-07 -1.579402         0.038420
2016-01-08 -1.604683         0.031323
```

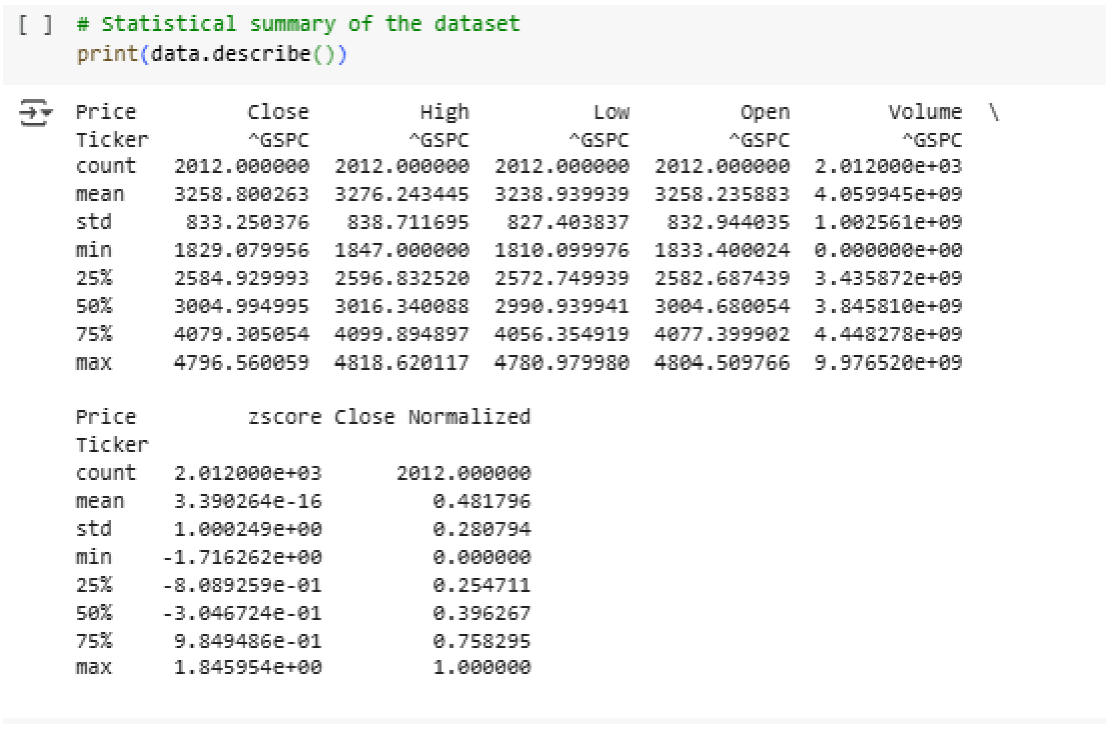
Figure 4-5 Final check for the dataset

4.3 Exploratory Data Analysis (EDA)

EDA is an essential stage in comprehending the fundamental framework of the data, identifying patterns, detecting anomalies, and forming hypotheses. The following subsections describe the visualizations, descriptive statistics, initial insights, and feature engineering performed during the EDA phase. EDA was carried out to understand the data patterns.

4.3.1 Descriptive Analysis

The daily price of the S&P 500 was taken from the Yahoo Finance by using the API of yfinance in Python. The first 75% of the dataset (2016-2022) is used for the training, while the rest is used for the validation and testing (2023-2024). The statistics analysis of the data of S&P 500 from 01/01/2016 to 01/01/2024 data points are shown in Figure 4.6.

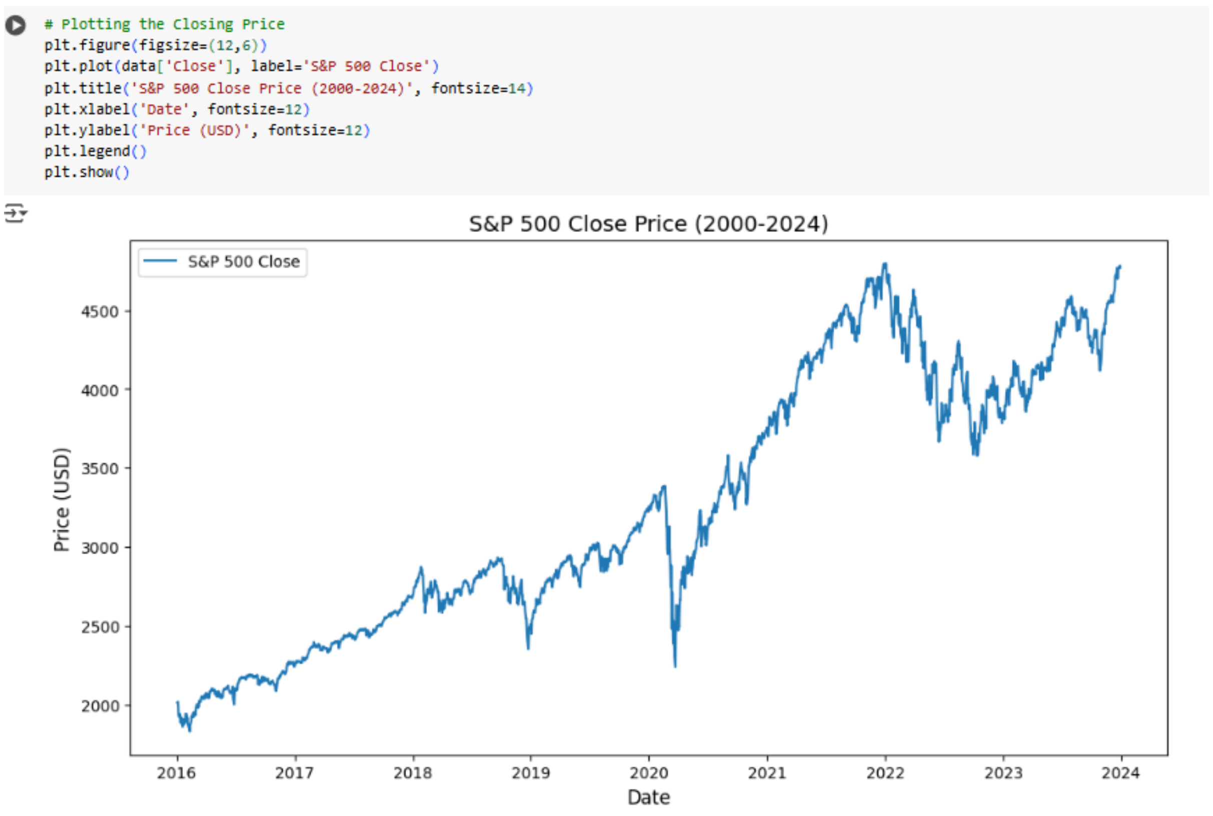


```
[ ] # Statistical summary of the dataset
    print(data.describe())
```

```
Price         Close         High          Low         Open        Volume  \
Ticker        ^GSPC        ^GSPC        ^GSPC        ^GSPC         ^GSPC
count   2012.000000  2012.000000  2012.000000  2012.000000  2.012000e+03
mean    3258.800263  3276.243445  3238.939939  3258.235883  4.059945e+09
std      833.250376   838.711695   827.403837   832.944035  1.002561e+09
min     1829.079956  1847.000000  1810.099976  1833.400024  0.000000e+00
25%     2584.929993  2596.832520  2572.749939  2582.687439  3.435872e+09
50%     3004.994995  3016.340088  2990.939941  3004.680054  3.845810e+09
75%     4079.305054  4099.894897  4056.354919  4077.399902  4.448278e+09
max     4796.560059  4818.620117  4780.979980  4804.509766  9.976520e+09

Price         zscore Close Normalized
Ticker
count   2.012000e+03      2012.000000
mean    3.390264e-16         0.481796
std     1.000249e+00         0.280794
min    -1.716262e+00         0.000000
25%    -8.089259e-01         0.254711
50%    -3.046724e-01         0.396267
75%     9.849486e-01         0.758295
max     1.845954e+00         1.000000
```

Figure 4-6 Statistics Analysis for the S&P 500

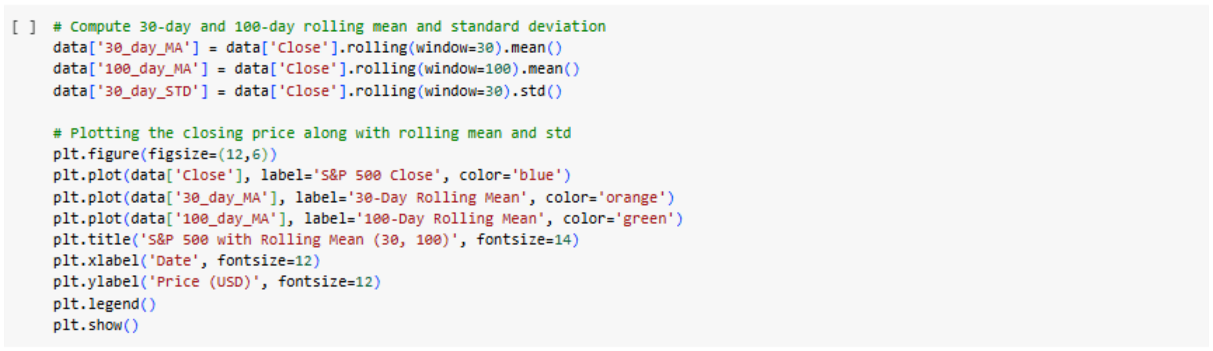
The data points are also depicted in Figure 4.7 It can be roughly described that generally, the trend is going up, although there is some sudden plummeting in this period, and from the start of 2022, the price is following a downtrend. As for the distribution of the data points, most of the data points fell below 3700 and the maximum is 4796.56 showing that the data is right-skewed.



```
# Plotting the Closing Price
plt.figure(figsize=(12,6))
plt.plot(data['Close'], label='S&P 500 Close')
plt.title('S&P 500 Close Price (2000-2024)', fontsize=14)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Price (USD)', fontsize=12)
plt.legend()
plt.show()
```

Figure 4-7 S&P 500 closing price from 01/01/2016 to 01/01/2024

Things get complicated when it comes to time-series stock market prediction. That is because the data tends to be non-stationary, that is the data points depend on the time at which they are observed. To get useful information, some of the traditional trading techniques from are being applied in this project such as the 30-day moving average and 100-day moving average as the features engineering. The Figure 4.8 shown the S&P 500 closing price with the 30-day moving average and 100-day moving average. As the Figure 4.8 shown that when the 30-day moving average and 100-day moving average lower than the closing price, which means the market is considered in the uptrend condition. When the 30-day moving average and 100-day moving average higher than the closing price, which means the market is considered in the downtrend condition. When the 30-day moving average crosses above the 100-day moving average, the condition is called Golden Cross which means the trends is potential for upward movement in the price. When the 30-day moving average crosses below the 100-day moving average, the condition is called Death Cross which means the trends is potential for downward movement in the price.



```
[ ] # Compute 30-day and 100-day rolling mean and standard deviation
    data['30_day_MA'] = data['Close'].rolling(window=30).mean()
    data['100_day_MA'] = data['Close'].rolling(window=100).mean()
    data['30_day_STD'] = data['Close'].rolling(window=30).std()

    # Plotting the closing price along with rolling mean and std
    plt.figure(figsize=(12,6))
    plt.plot(data['Close'], label='S&P 500 Close', color='blue')
    plt.plot(data['30_day_MA'], label='30-Day Rolling Mean', color='orange')
    plt.plot(data['100_day_MA'], label='100-Day Rolling Mean', color='green')
    plt.title('S&P 500 with Rolling Mean (30, 100)', fontsize=14)
    plt.xlabel('Date', fontsize=12)
    plt.ylabel('Price (USD)', fontsize=12)
    plt.legend()
    plt.show()
```

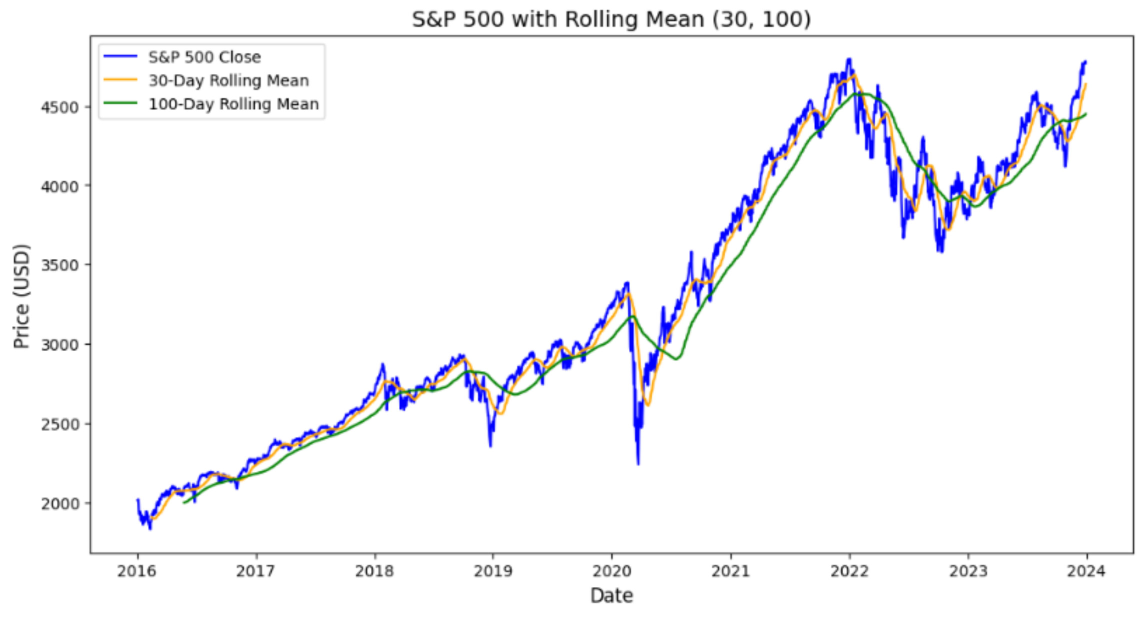
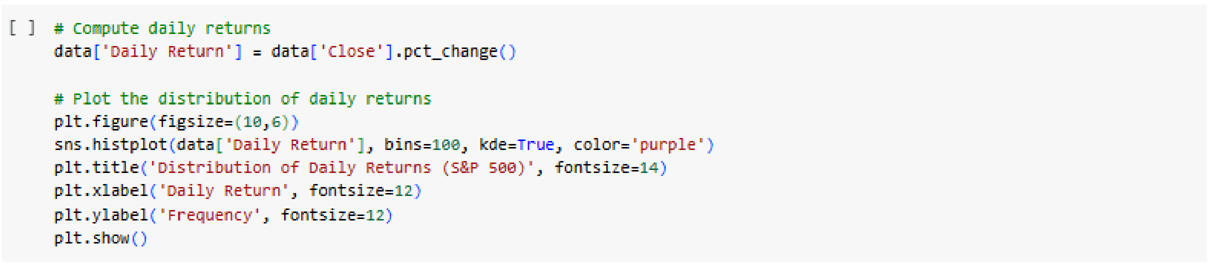


Figure 4-8 S&P 500 closing price with the 30-day moving average and 100-day moving average.

The distribution of the daily returns in the histogram will be plot to identify the frequency of the daily returns rate. In the Figure 4.9 shows the small price changes (in percentages) occurs most frequently but the significant market of fluctuations is still present and occasionally cause extreme returns. The distribution of the daily returns also helps to identify the behaviour of the returns either positive/ negative skew or kurtosis.



```
[ ]  # Compute daily returns
     data['Daily Return'] = data['Close'].pct_change()

     # Plot the distribution of daily returns
     plt.figure(figsize=(10,6))
     sns.histplot(data['Daily Return'], bins=100, kde=True, color='purple')
     plt.title('Distribution of Daily Returns (S&P 500)', fontsize=14)
     plt.xlabel('Daily Return', fontsize=12)
     plt.ylabel('Frequency', fontsize=12)
     plt.show()
```

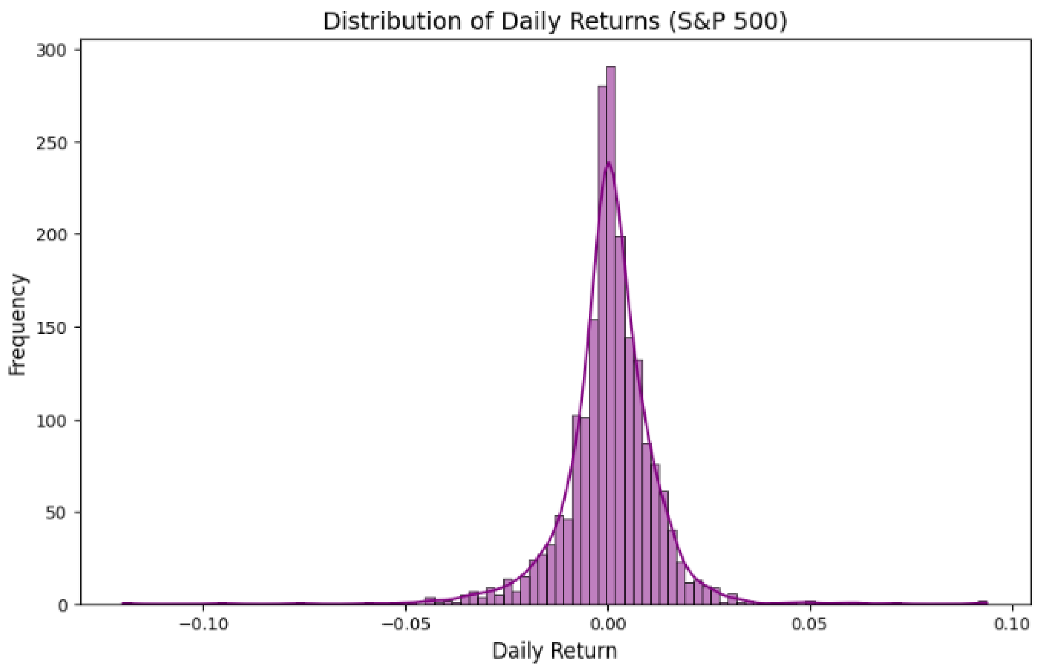
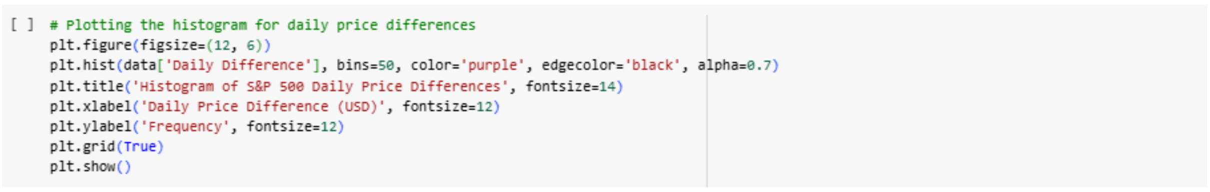


Figure 4-9 Distribution of Daily Returns in percentages (S&P 500)



```
[ ] # Plotting the histogram for daily price differences
    plt.figure(figsize=(12, 6))
    plt.hist(data['Daily Difference'], bins=50, color='purple', edgecolor='black', alpha=0.7)
    plt.title('Histogram of S&P 500 Daily Price Differences', fontsize=14)
    plt.xlabel('Daily Price Difference (USD)', fontsize=12)
    plt.ylabel('Frequency', fontsize=12)
    plt.grid(True)
    plt.show()
```

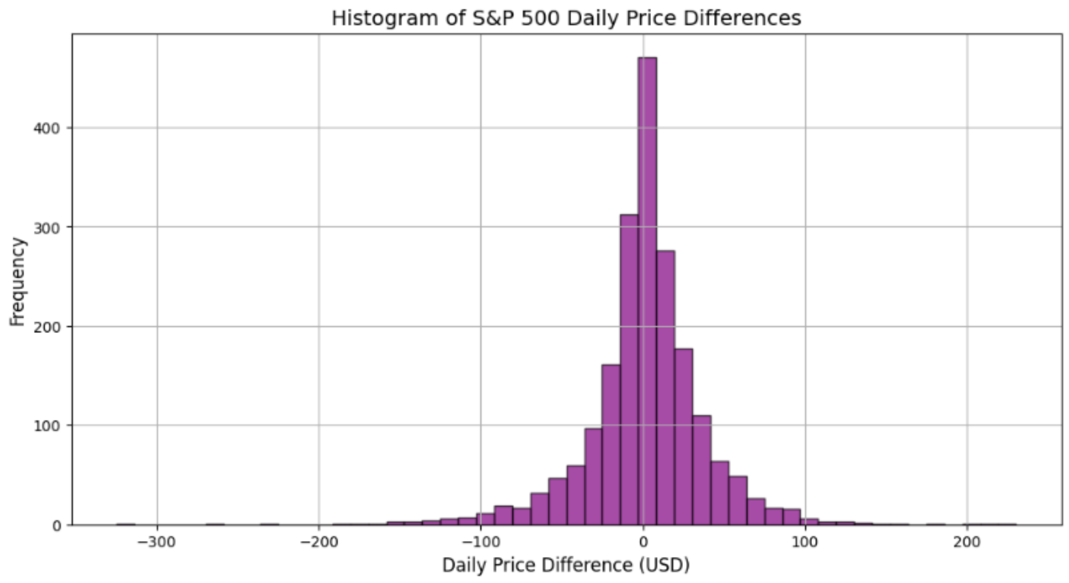
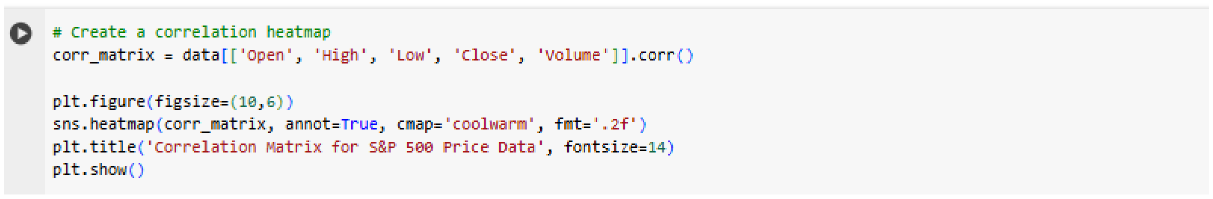


Figure 4-10 Distribution of Daily Returns in USD (S&P 500)

Correlation heatmap helps to identify the relation between the features. In the Figure 4.11 shows that the correlations between different price metrics/features (open, close, high, low and volume). The correlation between the features is strong because all of the metrics are inter-link with others features.



```
# Create a correlation heatmap
corr_matrix = data[['Open', 'High', 'Low', 'Close', 'Volume']].corr()

plt.figure(figsize=(10,6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix for S&P 500 Price Data', fontsize=14)
plt.show()
```

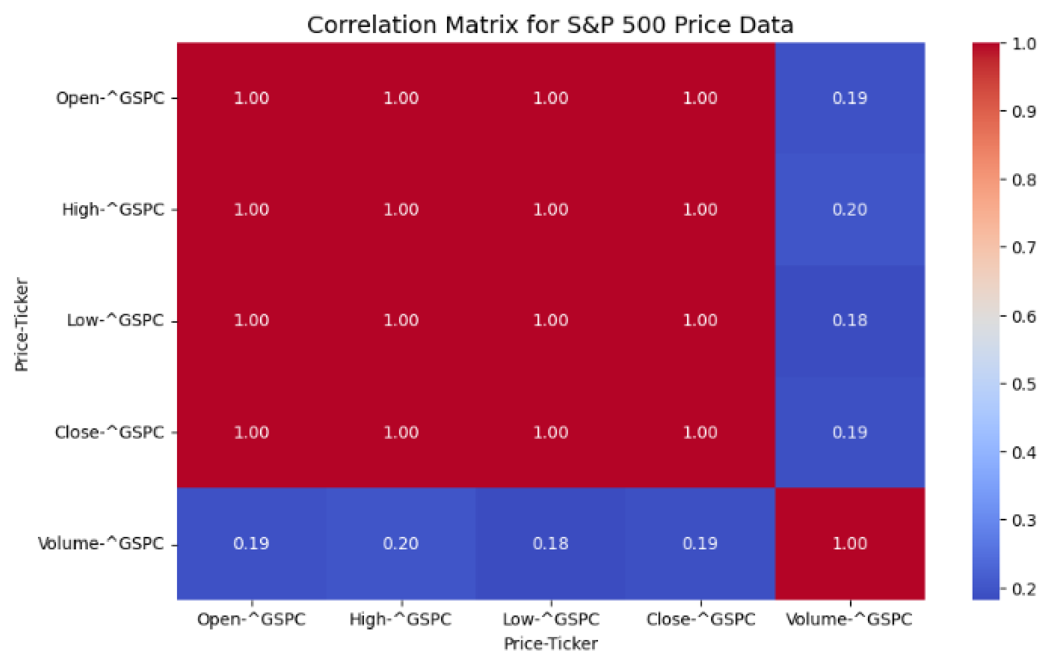
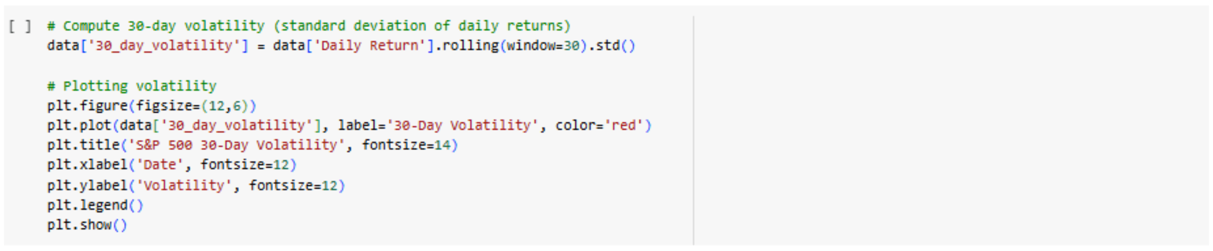


Figure 4-11 Correlation Matrix for S&P 500 Price Data

The volatility of the market over time can identify by using the volatility plot. In the figure 4.12 shows that the S&P 500 30-Day volatility. In 2020, the volatility of the market is sharpest because of the COVID-19 crash. The significant event leads to unexpected economic shutdowns caused by the pandemic. From 2016 until 2019 shows the stable periods in the economic and the confidence among investors. Crisis period during 2020 and recovery and post-COVID volatility after 2020. The figure 4.13 shows the difference in daily price either is positive or negative to identify the distribution of the price.



```
[ ] # Compute 30-day volatility (standard deviation of daily returns)
    data['30_day_volatility'] = data['Daily Return'].rolling(window=30).std()

    # Plotting volatility
    plt.figure(figsize=(12,6))
    plt.plot(data['30_day_volatility'], label='30-Day Volatility', color='red')
    plt.title('S&P 500 30-Day Volatility', fontsize=14)
    plt.xlabel('Date', fontsize=12)
    plt.ylabel('Volatility', fontsize=12)
    plt.legend()
    plt.show()
```

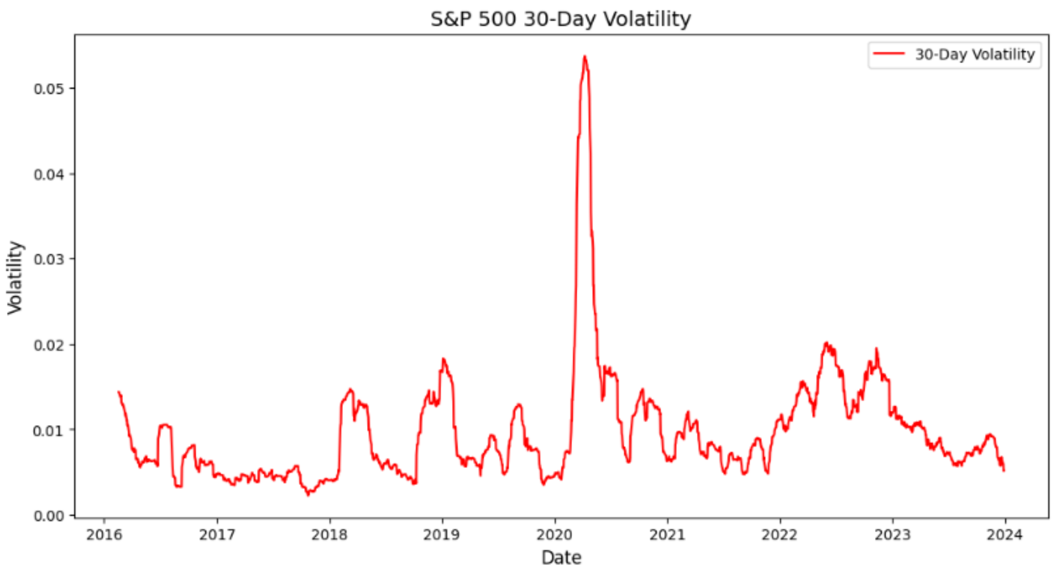
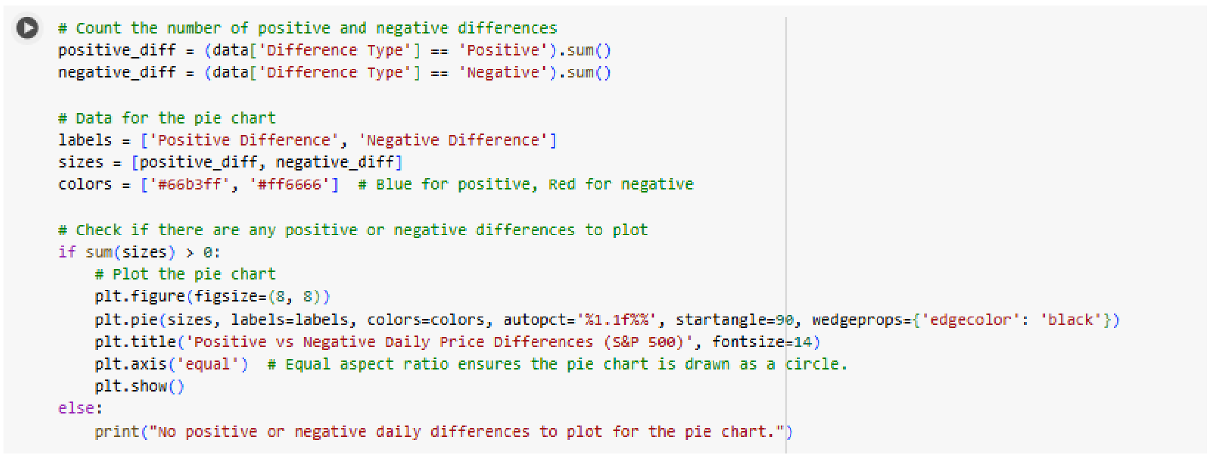


Figure 4-12 S&P 500 30-Day Volatility



```python
# Count the number of positive and negative differences
positive_diff = (data['Difference Type'] == 'Positive').sum()
negative_diff = (data['Difference Type'] == 'Negative').sum()

# Data for the pie chart
labels = ['Positive Difference', 'Negative Difference']
sizes = [positive_diff, negative_diff]
colors = ['#66b3ff', '#ff6666']  # Blue for positive, Red for negative

# Check if there are any positive or negative differences to plot
if sum(sizes) > 0:
    # Plot the pie chart
    plt.figure(figsize=(8, 8))
    plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=90, wedgeprops={'edgecolor': 'black'})
    plt.title('Positive vs Negative Daily Price Differences (S&P 500)', fontsize=14)
    plt.axis('equal')  # Equal aspect ratio ensures the pie chart is drawn as a circle.
    plt.show()
else:
    print("No positive or negative daily differences to plot for the pie chart.")
```

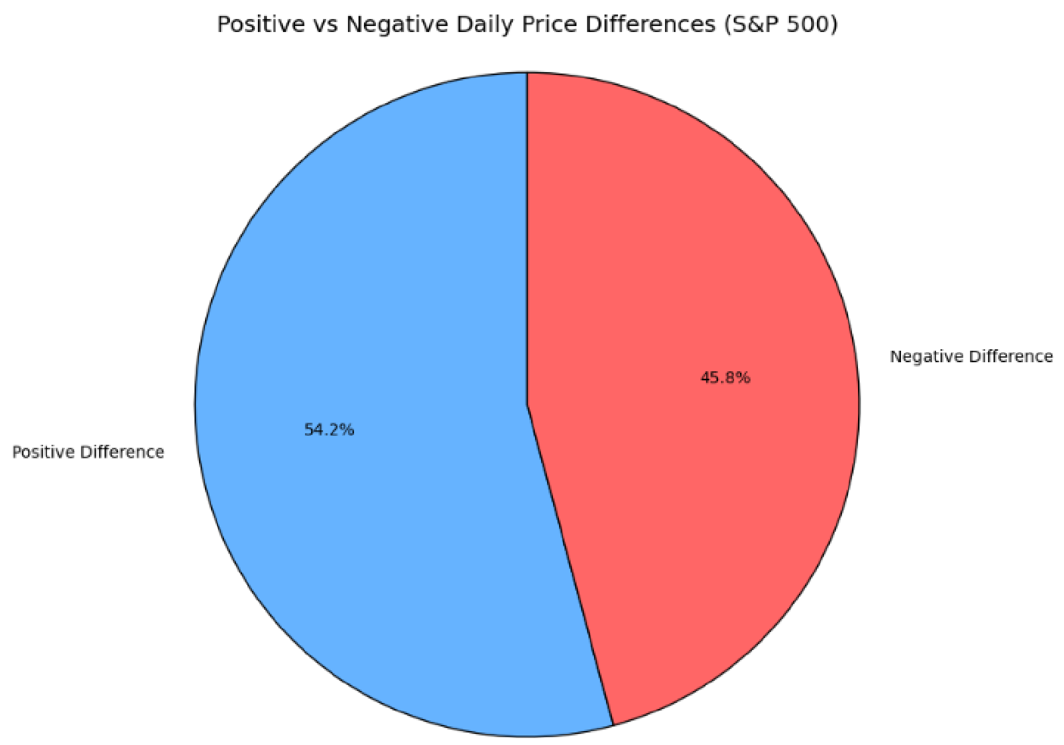


Figure 4-13 Positive vs Negative Daily Price Differences (S&P 500)

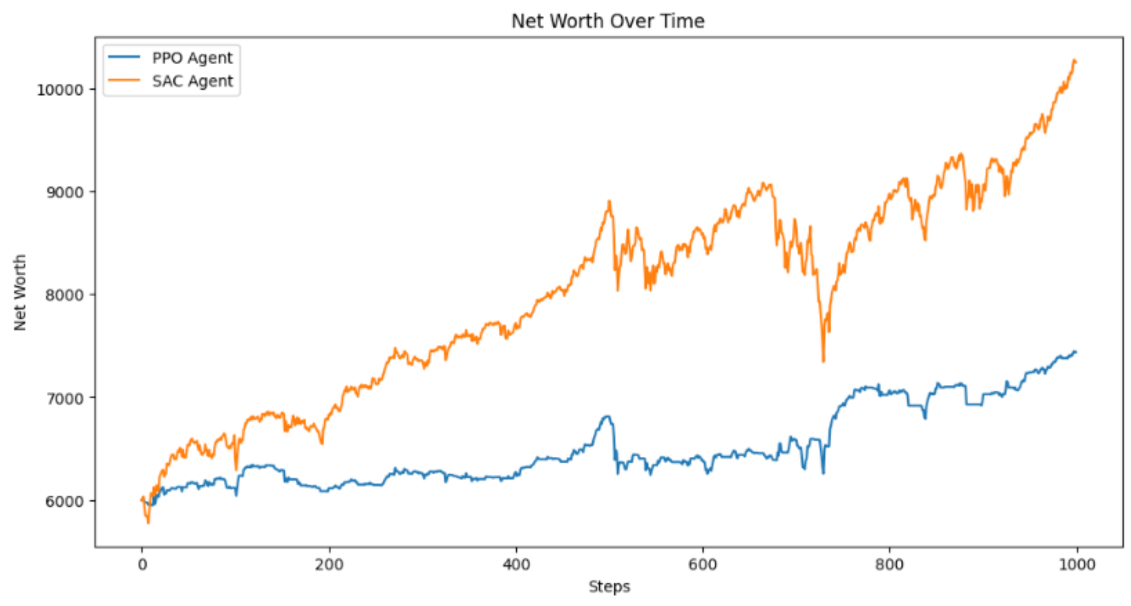
4.4 Expected Outcome

In the research, the expected contribution is to train agent with the high efficiency and stable in sentimental. What kinds of the difference will be brought by using the different model in trading agents which is PPO, SAC and DQN. Indicate the strengths and weaknesses of the PPO, SAC and DQN in trading strategy. Helping stakeholder to harvest more profit without worry and sentimental impact. The trader will be automation decision making based on the policy and reward mechanism. The establish policy and reward mechanism is needed to train the agent that can fully eliminated the sentimental and the decision-making will be more discipline and the profit will be optimized. Every model has their own strengths and weaknesses; by identify the advantages of model will let the stakeholder apply those models in more appropriately toward the real-time market environment. The model with the higher value in F1-scores and accuracy, and optimized cumulative return rates based on the back testing will be selected for the future trading strategy in decision making.

4.4.1 Model Development Structure

The DRL model of DQN, SAC and PPO have been developed and training including hyperparameter tuning in Python. Figure 4.14 and 4.15 are the performance results for DRL models Based on the Figure 4.14 and 4.15, the performance of the DQN, SAC and PPO can be compared with the same input which is S&P 500. For the net worth over time, PPO agent shows a steady increase. There are some of the stagnations or decline periods before the value begins to rise sharply. PPO able to make the profitable decision but encountering some of the limitation when facing with the certain market conditions. For SAC agent, it much smoother and more consistent rise in the net worth over time compared with the PPO agent. SAC agent could be the better at handling fluctuations in the market and more consistent. The DQN shows a more volatile net worth trajectory with the dramatically ups and downs. The indicates will cause the DQN agent has the high returns, it struggles to maintain the consistent growth and lose in stability of the decision-making process. The performance of the DRL models also can compare in the Sharpe ratio. SAC agent has the highest Sharpe ratio around 0.0747, which means it has a favorable balance between return and risk. SAC able to generate return with less volatility and better risk-adjusted performance. PPO agent with the lower Sharper ratio around 0.0342 which means the some of the return bring the risk where the risk-adjusted returns are much lower compared with SAC agent due to the high volatility of market in decision making process. DQN agent with the slightly higher Sharpe ratio to PPO around 0.0736. The return rate of DQN agent is decent, the volatility in the net worth but it loss the stability in decision making.

Based on the performance comparison, SAC outperform than PPO and DQN in terms of net worth growth and Sharpe ratio because SAC has the better ability to handle fluctuation of market and able to generate higher returns with lower risk. PPO has steady progress but faces more volatility and less favorable risk-adjusted returns compared with SAC. DQN with the highest volatility in performance where the significant fluctuations in net worth and a relatively low Sharpe ratio which means DQN hard to maintain the decision making in trading consistently. SAC more suitable for the dynamics stock market.



	Agent	Return	Standard Deviation	Sharpe Ratio
1	SAC Agent	0.000566	0.00757	0.074730
0	PPO Agent	0.000133	0.00388	0.034171

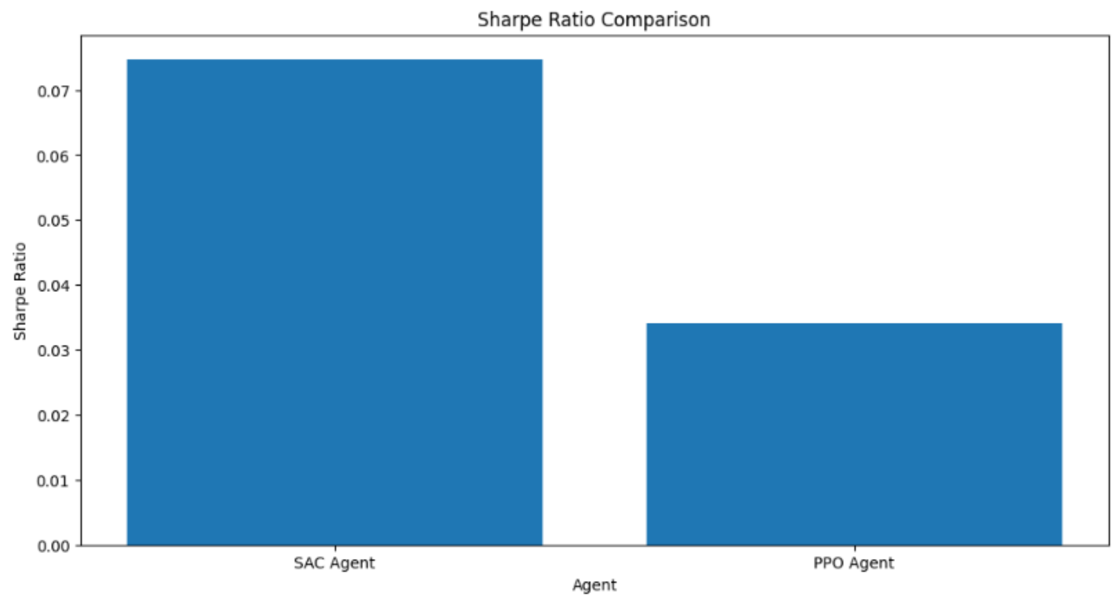
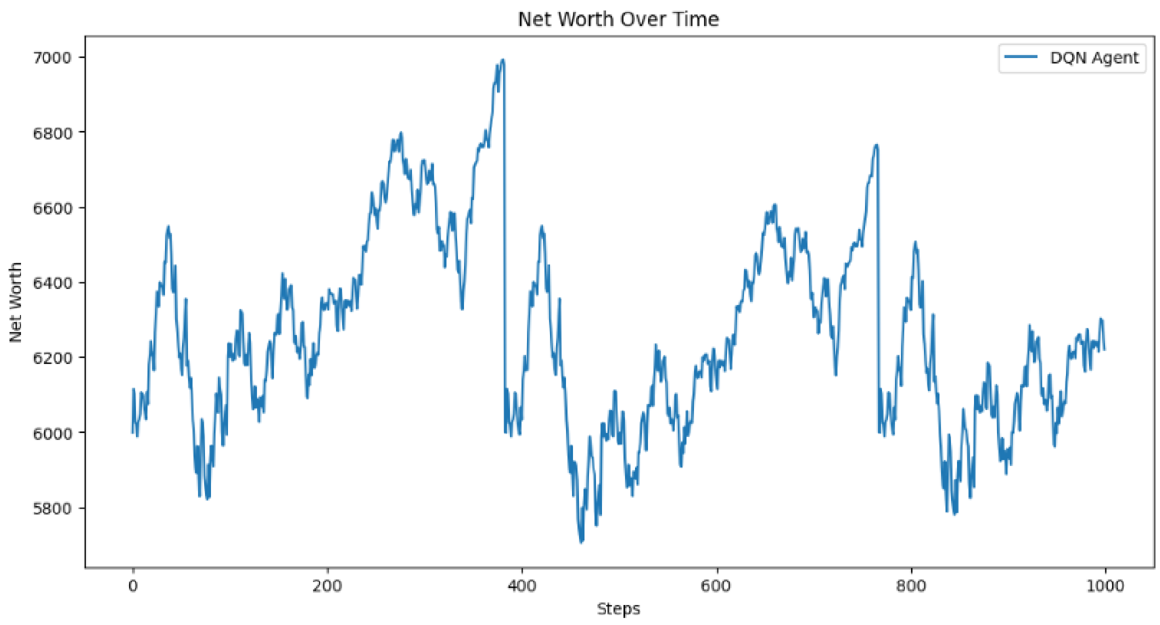


Figure 4-14 Output for PPO and SAC



	Agent	Return	Standard Deviation	Sharpe Ratio
0	DQN Agent	0.000561	0.007621	0.073644

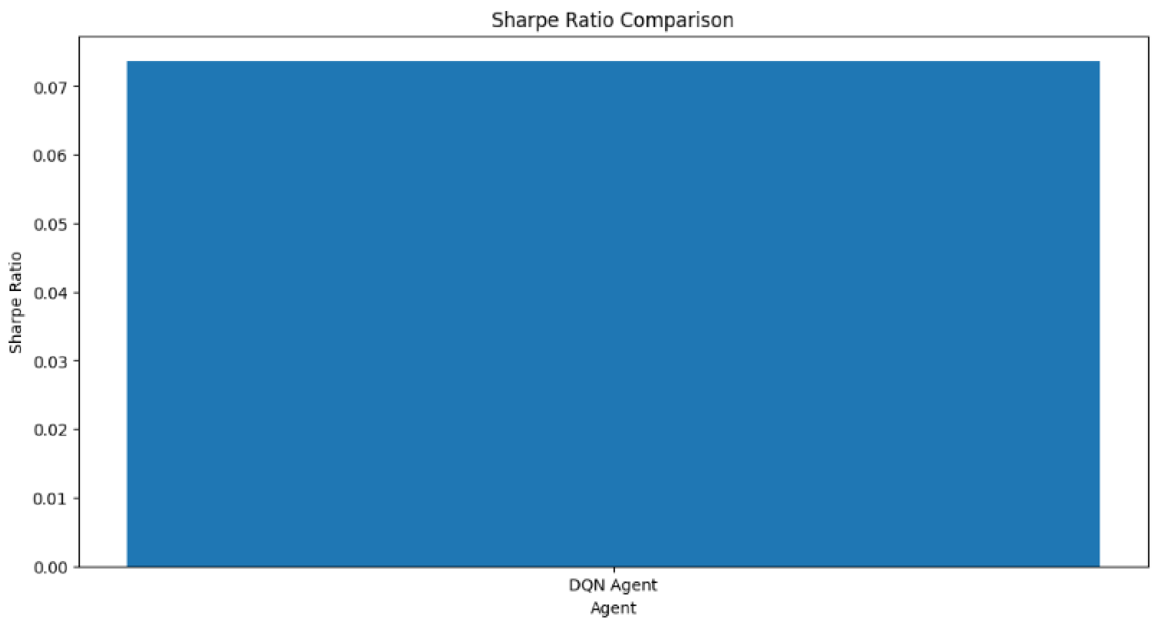


Figure 4-15 Output for DQN model

4.5 Chapter Summary

In summary, the chapter 4 will present the primary data sources which is the S&P 500 from the Yahoo Finance. The EDA involved descriptive statistics to identifying patterns, detecting anomalies, and forming hypotheses. Features engineering also added (30-day moving average and 100-day moving average) to let the agent familiar in the real-time stock market conditions. The expected outcome and the model development structure also included in the chapter as the future results. The preliminary results of the research have conducted and future works required to polish and explore the DRL model to achieve the objective that set for the research. Based on the performance of the DRL models for SAC, DQN and PPO in model development, SAC shows the better stability and highest cumulative return where the F1-score also higher than others 2 DRL models (DQN & SAC).

CHAPTER 5

CONCLUSION AND FUTURE WORKS

5.1 Introduction

The decision-making process in trading will be affected by the sentimental of human. In order to minimize the issues of human error with the emotional biases, deep reinforcement learning (DRL) models are vital to employ in the decision-making process of the trading strategy. The research is to using the deep reinforcement learning (DRL) for automated trading in stock market. The aims are to compare the different model performance among DQN, PPO and SAC, minimize the emotional biases and improved decision making which is automation in trading system. The study is proposed to achieve the following objective:

a. To obtain the policy that give optimized return.
b. To train agent that will not be influenced by the sentimental with using SRDRL model.
c. Develop a dashboard that visualize the return of the agent that trained on different mechanism.
d. Compare the performance within model and discuss the strength and weakness between different model (DQN, PPO and SAC).

The input data of the research is S&P 500. To enhance the results that obtained from model is accurate and reliable, data preprocessing and data cleaning steps are required to improve the quality of the data. After that EDA will takes place to identify the patterns of the dataset. Features engineering also added (30-day moving average and 100-day moving average) to let the agent familiar in the real-time stock market conditions.

5.2 Achievements

Data preparation was carried out as the initiate step to ensure the input data used for the model development was consistent and formatted. The data of S&P 500 was imported from the Yahoo Finance. The cleaned dataset can be obtained by handles the missing data points and removing the duplicates and outliers that occurred in the dataset. The implementation of the SAC, DQN and PPO for the trading strategy was evaluated the performance based on the key metrics such as F1-score and cumulative return over time. Based on the DRL models which are DQN, SAC and PPO, estimate SAC will outperform than others models (DQN and PPO) due to ability to handle continuous action spaces and high stability in dynamics environments. SAC is expected to be the best performance model for the automated stock trading, followed by PPO and DQN as the least effective. DQN estimate with the poorest performance due to the discrete action space might be unable to fully capture the complexity of the stock trading where continuous action is more preferred. PPO not fully exploit the advantages of continuous action spaces as SAC does.

5.3 Future Works/Recommendations

The dashboard development for the performance analysis to visualize and analyze the performance of different DRL models (DQN, SAC and PPO). The dashboard can including Sharpe ratio, net worth over time and agent comparison. Based on this, the simultaneous visulization in the same graph to show the performance of all DRL models. In the graph will required net worth comparison, Sharpe ratio analysis, volatility and rish analysis.

The policy analysis of 3 models will allow to analyze in details about the performance metrics where return, Sharpe ratio and standard deviation. The behavioral insight also can provide a clear breakdown of each of the DRL models including the action making at certion key time and correlate the action with the changes in net worth.

For the better performance of the models, the diversity of the data should be increased. The data of other indices like the Hang Seng Index of Hong Kong, the Straits Time Index of Singapore, the Financial Times Stock Exchange 100 Index of the United Kingdom, the Shanghai Composite Index of China, and the volume should be taken into account. This should give more information to the models so that the models can get more clues to make better predictions.

Next, the real-time deployment and backtesting should be required. The model testing in the live trading enviroment with the stock market enhance the insights into their true performance and adaptability. The backtesting frameworks such as QuantConnect or Backtrader to simulate the real-world trading environment, helps assess the refine strategy and risk-adjusted returns. High frequency trading (HFT) to make the model more applicable in the real-world trading.

REFERENCES

Aziz, et al. (2024), Role of behavioral biases in the investment decisions of Pakistan Stock Exchange investors: Moderating role of investment experience. Investment Management & Financial Innovations; Sumy Vol. 21, Iss. 1, (2024): 146-156.

Florin, C. D., Turcaș, F., Ștefania, A. N., Bențe, C., & Boiță, M. (2023). The impact of sentiment indices on the stock Exchange—The connections between quantitative sentiment indicators, technical analysis, and stock market. *Mathematics, 11*(14), 3128. doi:https://doi.org/10.3390/math11143128

Georg, et al. (2024), Current applications and potential future directions of reinforcement learning-based Digital Twins in agriculture, SBA Research gGmbH, Floragasse 7/5.OG, Vienna, 1040, Vienna, Austria.

Goluža, S., Kovačević, T., Bauman, T., & Kostanjčar, Z. (2024). Deep reinforcement learning with positional context for intraday trading. Ithaca: doi: https://doi.org/10.1007/s12530-024-09593-6

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*. Ithaca: Retrieved from https://vpn.utm.my/working-papers/soft-actor-critic-off-policy-maximum-entropy-deep/docview/2071194268/se-2

Huang, Y., Zhou, C., Zhang, L., & Lu, X. (2024). A self-rewarding mechanism in deep reinforcement learning for trading strategy optimization. Mathematics, 12(24), 4020. doi:https://doi.org/10.3390/math12244020

Inani, S. K., Pradhan, H., Kumar, S., & Biswas, B. (2024). Navigating the technical analysis in stock markets: Insights from bibliometric and topic modeling approaches. *Investment Management & Financial Innovations, 21*(1), 275-288. doi:https://doi.org/10.21511/imfi.21(1).2024.21

Jeremy J. Siegel & Jeremy D. Schwartz (2006). The Long-term Returns on the Original S&P 500 Firms, Financial Analysts Journal, vol.62(1), pp. 18-31, Taylor & Francis Ltd.

Kabbani, T., & Duman, E. (2022). Deep reinforcement learning approach for trading automation in the stock market. Ithaca: doi:https://doi.org/10.1109/ACCESS.2022.3203697

Kong, M., & So, J. (2023). Empirical analysis of automated stock trading using deep reinforcement learning. Applied Sciences, 13(1), 633. doi:https://doi.org/10.3390/app13010633

Korkmaz, E. (2024). *A survey analyzing generalization in deep reinforcement learning*. Ithaca: Retrieved from https://vpn.utm.my/working-papers/survey-analyzing-generalization-deep/docview/2910701950/se-2

Levi, Sriyank & P, Prathima & Merlyn, Sarah. (2021). "FUNDAMENTAL AND TECHNICAL ANALYSIS LEADS TO A SYSTEMATIC INVESTMENT DECISION IN STOCK MARKET EQUITIES". 3. 39-42.

Majidi, N., Shamsi, M., & Marvasti, F. (2022). Algorithmic trading using continuous action space deep reinforcement learning. Ithaca: Retrieved from https://vpn.utm.my/working-papers/algorithmic-trading-using-continuous-action-space/docview/2723274890/se-2

Mohammadshafie, A., Mirzaeinia, A., Jumakhan, H., & Mirzaeinia, A. (2024). *Deep reinforcement learning strategies in finance: Insights into asset holding, trading behavior, and purchase diversity*. Ithaca: Retrieved from https://vpn.utm.my/working-papers/deep-reinforcement-learning-strategies-finance/docview/3081452987/se-2

Murphy, J. J. (1999). Technical analysis of the financial markets: A comprehensive guide to trading methods and applications. Prentice Hall Press.

Otabek, S., & Choi, J. (2024). Multi-level deep Q-networks for bitcoin trading strategies. *Scientific Reports (Nature Publisher Group), 14*(1), 771. doi: https://doi.org/10.1038/s41598-024-51408-w

Prayudi, Rafandito & Purwanto, Eko. (2023). The Impact of Financial Literacy, Overconfidence Bias, Herding Bias and Loss Aversion Bias on Investment Decision. Indonesian Journal of Business Analytics. 3. 1873-1886. 10.55927/ijba.v3i5.5715.

Sangve, et al. (2025), ProfitPulse: Reinforcement Learning-Driven Trading Strategy. Artificial Intelligence & Data Science, Vishwakarma Institute of Technology, Pune, IND.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *ArXiv, abs/1707.06347*.

Sun, Q. (2023). Reinforcement learning algorithms for stock trading (Order No. 31765482). Available from ProQuest Dissertations & Theses Global. (3186188497). Retrieved from https://vpn.utm.my/dissertations-theses/reinforcement-learning-algorithms-stock-trading/docview/3186188497/se-2

Vanguelov, K. (2016). *Integration of technical trading behaviour in asset pricing* (Order No. 27821930). Available from ProQuest Dissertations & Theses Global. (2351278827). Retrieved from https://vpn.utm.my/dissertations-theses/integration-technical-trading-behaviour-asset/docview/2351278827/se-2

Vec, V., Tomažič, S., Kos, A., & Umek, A. (2024). Trends in real-time artificial intelligence methods in sports: A systematic review. Journal of Big Data, 11(1), 148. doi:https://doi.org/10.1186/s40537-024-01026-0

Appendix A Coding for Data Cleaning/Pre-processing

```
!pip install yfinance
!pip install plotly
!pip install yfinance matplotlib seaborn


import yfinance as yf
import pandas as pd


# Define the ticker symbol
ticker = "^GSPC"

# Fetch data from Yahoo Finance
data = yf.download(ticker, start="2016-01-01", end="2024-01-01")

# Check for missing values
missing_data = data.isnull().sum()
print("Missing values in each column:\n", missing_data)

# Check if there are any missing values in the data
if missing_data.any():
    print("Missing values found. Interpolation will be performed.")

    # Perform linear interpolation to fill missing values
    data_interpolated = data.interpolate(method='time', limit_direction='both')

    print("Interpolation completed.")

else:
    # If no missing data, skip interpolation
    data_interpolated = data
    print("No missing values found. Skipping interpolation.")

# Verify that there are no more missing values
missing_data_after = data_interpolated.isnull().sum()
print("Missing values after interpolation:\n", missing_data_after)

# Drop the 'Volume' column (for example)
data_interpolated = data_interpolated.drop(columns=['Volume'])

# Display the updated DataFrame
print(data_interpolated.head())

# Ensure the index is a datetime object
data.index = pd.to_datetime(data.index)

# Check the data types of all columns
print(data.dtypes)

from scipy.stats import zscore

# Calculate z-scores for the close prices
data['zscore'] = zscore(data['Close'])

# Filter out data points where z-score is greater than 3 (outliers)
data_clean = data[data['zscore'].abs() <= 3]

# Drop the z-score column for the final cleaned data
data_clean = data_clean.drop(columns=['zscore'])

print(data_clean.head())

# Load your data
# Assuming 'data_clean' is a DataFrame with a 'Close' column resulting from
previous steps
# Ensure 'Close' is numeric

# Add a 30-day rolling mean
data_clean['30_day_MA'] = data_clean['Close'].rolling(window=30).mean()

# Add a 100-day rolling mean
data_clean['100_day_MA'] = data_clean['Close'].rolling(window=100).mean()

# Display more rows to inspect the rolling means
print(data_clean.head(35))  # Displaying more rows so you can see the rolling means

# Normalize the 'Close' column to a range [0, 1]
data_clean['Close Normalized'] = (data_clean['Close'] - data_clean['Close'].min()) /
(data_clean['Close'].max() - data_clean['Close'].min())

# Display the first few rows
print(data_clean[['Close', 'Close Normalized']].head())

# Normalize the 'Close' column to a range [0, 1]
data['Close Normalized'] = (data['Close'] - data['Close'].min()) / (data['Close'].max() -
data['Close'].min())

# Display the first few rows
print(data[['Close', 'Close Normalized']].head())

import pandas as pd

# Load your data (replace with the actual file path or DataFrame)
# Assuming your data has a column 'Date' and 'Close' (or other stock-related data)

# Convert the Date column to datetime if not already done
data.index = pd.to_datetime(data.index)
```

```
# Calculate the split index
train_size = int(len(data) * 0.8)  # 80% for training

# Split the data into train and test sets
train_data = data.iloc[:train_size]
test_data = data.iloc[train_size:]

# Display the results
print("Train Data (80%):")
print(train_data.head())  # Show the first few rows of the train data
print("\nTest Data (20%):")
print(test_data.head())  # Show the first few rows of the test data

# Check for remaining missing values
print(data.isnull().sum())

# Check for duplicated rows
print(data.duplicated().sum())

# Display the final cleaned data
print(data.head())
```

Appendix B Coding for Descriptive Anlaysis

```
# Display the first few rows of the data
data.head()

# Check the structure of the data
data.info()

import matplotlib.pyplot as plt
import seaborn as sns

# Statistical summary of the dataset
print(data.describe())

# Plotting the Closing Price
plt.figure(figsize=(12,6))
plt.plot(data['Close'], label='S&P 500 Close')
plt.title('S&P 500 Close Price (2000-2024)', fontsize=14)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Price (USD)', fontsize=12)
plt.legend()
plt.show()

# Compute 30-day and 100-day rolling mean and standard deviation
data['30_day_MA'] = data['Close'].rolling(window=30).mean()
data['100_day_MA'] = data['Close'].rolling(window=100).mean()
data['30_day_STD'] = data['Close'].rolling(window=30).std()

# Plotting the closing price along with rolling mean and std
plt.figure(figsize=(12,6))
plt.plot(data['Close'], label='S&P 500 Close', color='blue')
plt.plot(data['30_day_MA'], label='30-Day Rolling Mean', color='orange')
plt.plot(data['100_day_MA'], label='100-Day Rolling Mean', color='green')
plt.title('S&P 500 with Rolling Mean (30, 100)', fontsize=14)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Price (USD)', fontsize=12)
plt.legend()
plt.show()

# Compute daily returns
data['Daily Return'] = data['Close'].pct_change()

# Plot the distribution of daily returns
plt.figure(figsize=(10,6))
sns.histplot(data['Daily Return'], bins=100, kde=True, color='purple')
plt.title('Distribution of Daily Returns (S&P 500)', fontsize=14)
plt.xlabel('Daily Return', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.show()

# Create a correlation heatmap
corr_matrix = data[['Open', 'High', 'Low', 'Close', 'Volume']].corr()

plt.figure(figsize=(10,6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix for S&P 500 Price Data', fontsize=14)
plt.show()

# Compute 30-day volatility (standard deviation of daily returns)
data['30_day_volatility'] = data['Daily Return'].rolling(window=30).std()

# Plotting volatility
plt.figure(figsize=(12,6))
plt.plot(data['30_day_volatility'], label='30-Day Volatility', color='red')
plt.title('S&P 500 30-Day Volatility', fontsize=14)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Volatility', fontsize=12)
plt.legend()
plt.show()

# Descriptive statistics for the entire dataset
descriptive_stats = data.describe()

# Display the descriptive statistics
print("Descriptive Statistics for S&P 500 Data:")
print(descriptive_stats)

# Skewness and Kurtosis for each column
skewness = data.skew()
kurtosis = data.kurtosis()

print("Skewness:\n", skewness)
print("\nKurtosis:\n", kurtosis)

# Compute variance and range
variance = data.var()
range_values = data.max() - data.min()

print("\nVariance:\n", variance)
print("\nRange:\n", range_values)

# Compute the correlation matrix
correlation_matrix = data.corr()

print("\nCorrelation Matrix:\n", correlation_matrix)

# Calculate the daily difference in the adjusted close price
data['Daily Difference'] = data['Close'].diff()

# Display the first few rows of the data with the daily difference
```

```
print(data[['Close', 'Daily Difference']].head())

# Plotting the histogram for daily price differences
plt.figure(figsize=(12, 6))
plt.hist(data['Daily Difference'], bins=50, color='purple', edgecolor='black',
        alpha=0.7)
plt.title('Histogram of S&P 500 Daily Price Differences', fontsize=14)
plt.xlabel('Daily Price Difference (USD)', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.grid(True)
plt.show()

# Create a new column to categorize the daily difference as positive or negative
data['Difference Type'] = ['Positive' if diff > 0 else 'Negative' if diff < 0 else 'No
        Change'
                    for diff in data['Daily Difference']]

# Check the first few rows of the data
data[['Close', 'Daily Difference', 'Difference Type']].head()

# Count the number of positive and negative differences
positive_diff = (data['Difference Type'] == 'Positive').sum()
negative_diff = (data['Difference Type'] == 'Negative').sum()

# Data for the pie chart
labels = ['Positive Difference', 'Negative Difference']
sizes = [positive_diff, negative_diff]
colors = ['#66b3ff', '#ff6666']  # Blue for positive, Red for negative

# Check if there are any positive or negative differences to plot
if sum(sizes) > 0:
    # Plot the pie chart
    plt.figure(figsize=(8, 8))
    plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=90,
        wedgeprops={'edgecolor': 'black'})
    plt.title('Positive vs Negative Daily Price Differences (S&P 500)', fontsize=14)
    plt.axis('equal')  # Equal aspect ratio ensures the pie chart is drawn as a circle.
    plt.show()
else:
    print("No positive or negative daily differences to plot for the pie chart.")
```

Appendix C Coding for Model Development

```
# 1. install the required packages
!pip install yfinance stable-baselines3 gym numpy pandas matplotlib


# Step 1: Install `mamba` (conda alternative)
!pip install -q condacolab
import condacolab
condacolab.install()

# Step 2: Install `ta-lib` using mamba (conda) via a precompiled binary
!mamba install -c conda-forge ta-lib

!pip install ta-lib

import yfinance as yf
import pandas as pd

tickers = [
    '^GSPC'
]

def get_data(tickers):
    stock_data = {}
    for ticker in tickers:
        df = yf.download(ticker, start="2016-01-01", end="2024-01-01")
        stock_data[ticker] = df
    return stock_data

stock_data = get_data(tickers)
```

```
for ticker, df in stock_data.items():
    df.to_csv(f'{ticker}.csv')

# Check the columns to see the multi-level structure
print(f"Original columns for {ticker}: {df.columns}")

# Flatten the multi-level columns (if applicable)
df.columns = [col[0] for col in df.columns]

# Now, print the flattened columns
print(f"Flattened columns for {ticker}: {df.columns}")

df.head()

stock_data = {}
stock_data[ticker] = df



# Calculate the split index
train_size = int(len(df) * 0.8)  # 80% for training

# split the data into training, validation and test sets
training_data = {}
test_data = {}

for ticker, df in stock_data.items():
    training_data[ticker] = df.iloc[:train_size]
    test_data[ticker] = df.iloc[train_size:]

import numpy as np

def add_technical_indicators(df):
    # calculate RSI 14
    delta = df['Close'].diff()
    up = delta.where(delta > 0, 0)
    down = -delta.where(delta < 0, 0)
    rs = up.rolling(window=14).mean() / down.rolling(window=14).mean()
    df['RSI'] = 100 - (100 / (1 + rs))

    # Tính toán EMA 12 và 26 kỳ cho MACD
    df['EMA12'] = df['Close'].ewm(span=12, adjust=False).mean()
    df['EMA26'] = df['Close'].ewm(span=26, adjust=False).mean()
    df['MACD'] = df['EMA12'] - df['EMA26']
    df['Signal'] = df['MACD'].ewm(span=9, adjust=False).mean()

    # Tính toán RSI 14 kỳ
    delta = df['Close'].diff()
    gain = (delta.where(delta > 0, 0)).rolling(window=14).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=14).mean()
    rs = gain / loss
    df['RSI'] = 100 - (100 / (1 + rs))

    # Tính toán CCI 20 kỳ
    tp = (df['High'] + df['Low'] + df['Close']) / 3
    sma_tp = tp.rolling(window=20).mean()
    mean_dev = tp.rolling(window=20).apply(lambda x: np.mean(np.abs(x -
x.mean())))
    df['CCI'] = (tp - sma_tp) / (0.015 * mean_dev)

    # Tính toán ADX 14 kỳ
    high_diff = df['High'].diff()
    low_diff = df['Low'].diff()
    df['+DM'] = np.where((high_diff > low_diff) & (high_diff > 0), high_diff, 0)
    df['-DM'] = np.where((low_diff > high_diff) & (low_diff > 0), low_diff, 0)
    tr = pd.concat([df['High'] - df['Low'], np.abs(df['High'] - df['Close'].shift(1)),
np.abs(df['Low'] - df['Close'].shift(1))], axis=1).max(axis=1)
    atr = tr.ewm(span=14, adjust=False).mean()
    df['+DI'] = 100 * (df['+DM'].ewm(span=14, adjust=False).mean() / atr)
    df['-DI'] = 100 * (df['-DM'].ewm(span=14, adjust=False).mean() / atr)
    dx = 100 * np.abs(df['+DI'] - df['-DI']) / (df['+DI'] + df['-DI'])
    df['ADX'] = dx.ewm(span=14, adjust=False).mean()

    # drop NaN values
    df.dropna(inplace=True)

    # keep Open, High, Low, Close, Volume, MACD, Signal, RSI, CCI, ADX
    df = df[['Open', 'High', 'Low', 'Close', 'Volume', 'MACD', 'Signal', 'RSI', 'CCI',
'ADX']]

    return df

# add technical indicators to the training data for each stock
for ticker, df in training_data.items():
    training_data[ticker] = add_technical_indicators(df)

# add technical indicators to the test data for each stock
for ticker, df in test_data.items():
    test_data[ticker] = add_technical_indicators(df)

import gymnasium as gym
from gymnasium import spaces
import numpy as np
import pandas as pd

class StockTradingEnv(gym.Env):
    metadata = {'render_modes': ['human']}

    def __init__(self, stock_data, transaction_cost_percent=0.0005):
        super(StockTradingEnv, self).__init__()

        # Remove any empty DataFrames
        self.stock_data = {ticker: df for ticker, df in stock_data.items() if not df.empty}
        self.tickers = list(self.stock_data.keys())

        if not self.tickers:
            raise ValueError("All provided stock data is empty")

        # Calculate the size of one stock's data
        sample_df = next(iter(self.stock_data.values()))
        self.n_features = len(sample_df.columns)

        # Define action and observation space
        self.action_space = spaces.Box(low=-1, high=1, shape=(len(self.tickers),),
dtype=np.float32)

        # Observation space: price data for each stock + balance + shares held + net
worth + max net worth + current step
        self.obs_shape = self.n_features * len(self.tickers) + 2 + len(self.tickers) + 2
        self.observation_space = spaces.Box(low=-np.inf, high=np.inf,
shape=(self.obs_shape,), dtype=np.float32)

        # Initialize account balance
        self.initial_balance = 6000
        self.balance = self.initial_balance
        self.net_worth = self.initial_balance
        self.max_net_worth = self.initial_balance
        self.shares_held = {ticker: 0 for ticker in self.tickers}
        self.total_shares_sold = {ticker: 0 for ticker in self.tickers}
        self.total_sales_value = {ticker: 0 for ticker in self.tickers}

        # Set the current step
        self.current_step = 0

        # Calculate the minimum length of data across all stocks
        self.max_steps = max(0, min(len(df) for df in self.stock_data.values()) - 1)
        # Transaction cost
        self.transaction_cost_percent = transaction_cost_percent

    def reset(self, seed=None, options=None):
        super().reset(seed=seed)
        self.balance = self.initial_balance
        self.net_worth = self.initial_balance
        self.max_net_worth = self.initial_balance
        self.shares_held = {ticker: 0 for ticker in self.tickers}
        self.total_shares_sold = {ticker: 0 for ticker in self.tickers}
        self.total_sales_value = {ticker: 0 for ticker in self.tickers}
        self.current_step = 0
        return self._next_observation(), {}

    def _next_observation(self):
        # initialize the frame
        frame = np.zeros(self.obs_shape)

        # Add stock data for each ticker
        idx = 0
        # Loop through each ticker
        for ticker in self.tickers:
            # Get the DataFrame for the current ticker
            df = self.stock_data[ticker]
            # If the current step is less than the length of the DataFrame, add the price
data for the current step
            if self.current_step < len(df):
                frame[idx:idx+self.n_features] = df.iloc[self.current_step].values
            # Otherwise, add the last price data available
            elif len(df) > 0:
                frame[idx:idx+self.n_features] = df.iloc[-1].values
            # Move the index to the next ticker
            idx += self.n_features
        # Add balance, shares held, net worth, max net worth, and current step
        frame[-4-len(self.tickers)] = self.balance # Balance
        frame[-3-len(self.tickers):-3] = [self.shares_held[ticker] for ticker in self.tickers]
# Shares held
        frame[-3] = self.net_worth # Net worth
        frame[-2] = self.max_net_worth # Max net worth
        frame[-1] = self.current_step # Current step

        return frame

    def step(self, actions):
        # update the current step
        self.current_step += 1

        # check if we have reached the maximum number of steps
        if self.current_step > self.max_steps:
            return self._next_observation(), 0, True, False, {}

        current_prices = {}
        # Loop through each ticker and perform the action
        for i, ticker in enumerate(self.tickers):
            # Get the current price of the stock
            current_prices[ticker] = self.stock_data[ticker].iloc[self.current_step]['Close']
            # get the action for the current ticker
            action = actions[i]

            if action > 0:  # Buy
                # Calculate the number of shares to buy
                shares_to_buy = int(self.balance * action / current_prices[ticker])
                # Calculate the cost of the shares
                cost = shares_to_buy * current_prices[ticker]
                # Transaction cost
                transaction_cost = cost * self.transaction_cost_percent
                # Update the balance and shares held
                self.balance -= (cost + transaction_cost)
                # Update the total shares sold
                self.shares_held[ticker] += shares_to_buy

            elif action < 0:  # Sell
                # Calculate the number of shares to sell
                shares_to_sell = int(self.shares_held[ticker] * abs(action))
                # Calculate the sale value
                sale = shares_to_sell * current_prices[ticker]
                # Transaction cost
                transaction_cost = sale * self.transaction_cost_percent
                # Update the balance and shares held
                self.balance += (sale - transaction_cost)
                # Update the total shares sold
                self.shares_held[ticker] -= shares_to_sell
                # Update the shares sold
                self.total_shares_sold[ticker] += shares_to_sell
                # Update the total sales value
                self.total_sales_value[ticker] += sale

        # Calculate the net worth
        self.net_worth = self.balance + sum(self.shares_held[ticker] *
current_prices[ticker] for ticker in self.tickers)
        # Update the max net worth
        self.max_net_worth = max(self.net_worth, self.max_net_worth)
        # Calculate the reward
        reward = self.net_worth - self.initial_balance
        # Check if the episode is done
        done = self.net_worth <= 0 or self.current_step >= self.max_steps

        obs = self._next_observation()
        return obs, reward, done, False, {}

    def render(self, mode='human'):
        # Print the current step, balance, shares held, net worth, and profit
        profit = self.net_worth - self.initial_balance
        print(f'Step: {self.current_step}')
        print(f'Balance: {self.balance:.2f}')
        for ticker in self.tickers:
            print(f'{ticker} Shares held: {self.shares_held[ticker]}')
        print(f'Net worth: {self.net_worth:.2f}')
        print(f'Profit: {profit:.2f}')

    def close(self):
        pass

def update_stock_data(self, new_stock_data, transaction_cost_percent=None):
    """
    Update the environment with new stock data.

    Parameters:
    new_stock_data (dict): Dictionary containing new stock data,
                           with keys as stock tickers and values as DataFrames.
    """
    # Remove empty DataFrames
    self.stock_data = {ticker: df for ticker, df in new_stock_data.items() if not
df.empty}
    self.tickers = list(self.stock_data.keys())

    if not self.tickers:
        raise ValueError("All new stock data are empty")

    # Update the number of features if needed
    sample_df = next(iter(self.stock_data.values()))
    self.n_features = len(sample_df.columns)

    # Update observation space
    self.obs_shape = self.n_features * len(self.tickers) + 2 + len(self.tickers) + 2
    self.observation_space = spaces.Box(low=-np.inf, high=np.inf,
shape=(self.obs_shape,), dtype=np.float32)

    # Update maximum steps
    self.max_steps = max(0, min(len(df) for df in self.stock_data.values()) - 1)

    # Update transaction cost if provided
    if transaction_cost_percent is not None:
        self.transaction_cost_percent = transaction_cost_percent

    # Reset the environment
    self.reset()

    print(f"The environment has been updated with {len(self.tickers)} new stocks.")

!pip install optuna

from stable_baselines3 import PPO, SAC, DQN
from stable_baselines3.common.vec_env import DummyVecEnv

class PPOAgent:
    def __init__(self, env, total_timesteps, hyperparams=None):
        if hyperparams is None:
            hyperparams = {}
        self.model = PPO("MlpPolicy", env, verbose=0, **hyperparams)
        self.model.learn(total_timesteps=total_timesteps)

    def predict(self, obs):
        action, _ = self.model.predict(obs)
        return action

class SACAgent:
    def __init__(self, env, total_timesteps, hyperparams=None):
        if hyperparams is None:
            hyperparams = {}
        self.model = SAC("MlpPolicy", env, verbose=0, **hyperparams)
        self.model.learn(total_timesteps=total_timesteps)

    def predict(self, obs):
        action, _ = self.model.predict(obs)
        return action

class DQNAgent:
    def __init__(self, env, total_timesteps, hyperparams=None):
        if hyperparams is None:
            hyperparams = {}
        self.model = DQN("MlpPolicy", env, verbose=0, **hyperparams)
        self.model.learn(total_timesteps=total_timesteps)

    def predict(self, obs):
        action, _ = self.model.predict(obs)
        return action

#Function to Create Environment + Train Model
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Function to create the environment and train the agents
def create_env_and_train_agents(training_data, total_timesteps,
                                ppo_params=None, sac_params=None):
    env = DummyVecEnv([lambda: StockTradingEnv(training_data)])
    ppo_agent = PPOAgent(env, total_timesteps, hyperparams=ppo_params)
    sac_agent = SACAgent(env, total_timesteps, hyperparams=sac_params)
    return env, ppo_agent, sac_agent

import matplotlib.pyplot as plt
```

```
# Function to visualize portfolio changes
def visualize_portfolio(steps, balances, net_worths, shares_held, tickers,
show_balance=True, show_net_worth=True, show_shares_held=True):
    fig, axs = plt.subplots(3, figsize=(12, 18))

    # Plot the balance
    if show_balance:
        axs[0].plot(steps, balances, label='Balance')
        axs[0].set_title('Balance Over Time')
        axs[0].set_xlabel('Steps')
        axs[0].set_ylabel('Balance')
        axs[0].legend()

    # Plot the net worth
    if show_net_worth:
        axs[1].plot(steps, net_worths, label='Net Worth', color='orange')
        axs[1].set_title('Net Worth Over Time')
        axs[1].set_xlabel('Steps')
        axs[1].set_ylabel('Net Worth')
        axs[1].legend()

    # Plot the shares held
    if show_shares_held:
        for ticker in tickers:
            axs[2].plot(steps, shares_held[ticker], label=f'Shares Held: {ticker}')
        axs[2].set_title('Shares Held Over Time')
        axs[2].set_xlabel('Steps')
        axs[2].set_ylabel('Shares Held')
        axs[2].legend()

    plt.tight_layout()
    plt.show()

# function to visualize the portfolio net worth
```

```
def visualize_portfolio_net_worth(steps, net_worths):
    plt.figure(figsize=(12, 6))
    plt.plot(steps, net_worths, label='Net Worth', color='orange')
    plt.title('Net Worth Over Time')
    plt.xlabel('Steps')
    plt.ylabel('Net Worth')
    plt.legend()
    plt.show()

# function to visualize the multiple portfolio net worths ( same chart )
def visualize_multiple_portfolio_net_worth(steps, net_worths_list, labels):
    plt.figure(figsize=(12, 6))
    for i, net_worths in enumerate(net_worths_list):
        plt.plot(steps, net_worths, label=labels[i])
    plt.title('Net Worth Over Time')
    plt.xlabel('Steps')
    plt.ylabel('Net Worth')
    plt.legend()
    plt.show()

#Function Visualize the Agent's performance

def test_agent(env, agent, stock_data, n_tests=1000, visualize=False):
    """
    Test a single agent and track performance metrics, with an option to visualize the
results.

    Parameters:
    - env: The trading environment.
    - agent: The agent to be tested.
    - stock_data: Data for the stocks in the environment.
    - n_tests: Number of tests to run (default: 1000).
    - visualize: Boolean flag to enable or disable visualization (default: False).
```

```
Returns:
- A dictionary containing steps, balances, net worths, and shares held.
"""
# Initialize metrics tracking
metrics = {
    'steps': [],
    'balances': [],
    'net_worths': [],
    'shares_held': {ticker: [] for ticker in stock_data.keys()}
}

# Reset the environment before starting the tests
obs = env.reset()

for i in range(n_tests):
    metrics['steps'].append(i)
    action = agent.predict(obs)
    obs, rewards, dones, infos = env.step(action)
    if visualize:
        env.render()

    # Track metrics
    metrics['balances'].append(env.get_attr('balance')[0])
    metrics['net_worths'].append(env.get_attr('net_worth')[0])
    env_shares_held = env.get_attr('shares_held')[0]

    # Update shares held for each ticker
    for ticker in stock_data.keys():
        if ticker in env_shares_held:
            metrics['shares_held'][ticker].append(env_shares_held[ticker])
        else:
            metrics['shares_held'][ticker].append(0)  # Append 0 if ticker is not found

    if dones:
        obs = env.reset()

    return metrics

#Function to Test and Visualize the Agent's performance

def test_and_visualize_agents(env, agents, train_data, n_tests=1000):
    metrics = {}
    for agent_name, agent in agents.items():
        print(f"Testing {agent_name}...")
        metrics[agent_name] = test_agent(env, agent, train_data, n_tests=n_tests,
visualize=True)
        print(f"Done testing {agent_name}!")

    print('-'*50)
    print('All agents tested!')
    print('-'*50)

    # Extract net worths for visualization
    net_worths = [metrics[agent_name]['net_worths'] for agent_name in agents.keys()]
    steps = next(iter(metrics.values()))['steps']  # Assuming all agents have the same
step count for simplicity

    # Visualize the performance metrics of multiple agents
    visualize_multiple_portfolio_net_worth(steps, net_worths, list(agents.keys()))

import optuna
from stable_baselines3.common.evaluation import evaluate_policy

def ppo_objective(trial):
    # Define the hyperparameter search space
    hyperparams = {
        'learning_rate': trial.suggest_loguniform('learning_rate', 1e-4, 1e-2),
        'gamma': trial.suggest_float('gamma', 0.90, 0.9999),
        'n_steps': trial.suggest_int('n_steps', 64, 2048, step=64),
        'ent_coef': trial.suggest_loguniform('ent_coef', 1e-8, 1e-1),
        'clip_range': trial.suggest_float('clip_range', 0.1, 0.4),
        'gae_lambda': trial.suggest_float('gae_lambda', 0.8, 1.0),
        'max_grad_norm': trial.suggest_float('max_grad_norm', 0.3, 5.0),
    }

    # Create environment
    env = DummyVecEnv([lambda: StockTradingEnv(training_data)])

    # Train agent with current hyperparams
    agent = PPOAgent(env, total_timesteps=5000, hyperparams=hyperparams)

    # Evaluate the model
    mean_reward, _ = evaluate_policy(agent.model, env, n_eval_episodes=5,
warn=False)
    return mean_reward

def sac_objective(trial):
    hyperparams = {
        'learning_rate': trial.suggest_loguniform('learning_rate', 1e-4, 1e-2),
        'gamma': trial.suggest_float('gamma', 0.90, 0.9999),
        'ent_coef': trial.suggest_loguniform('ent_coef', 1e-8, 1e-1),
        'tau': trial.suggest_float('tau', 0.005, 0.02),
        'batch_size': trial.suggest_categorical('batch_size', [64, 128, 256]),
        'train_freq': trial.suggest_categorical('train_freq', [1, 4, 8]),
    }

    env = DummyVecEnv([lambda: StockTradingEnv(training_data)])
    agent = SACAgent(env, total_timesteps=5000, hyperparams=hyperparams)

    mean_reward, _ = evaluate_policy(agent.model, env, n_eval_episodes=5,
warn=False)
    return mean_reward
```

```
# PPO tuning
ppo_study = optuna.create_study(direction='maximize')
ppo_study.optimize(ppo_objective, n_trials=30)
print("Best PPO trial:", ppo_study.best_trial.params)

# SAC tuning
sac_study = optuna.create_study(direction='maximize')
sac_study.optimize(sac_objective, n_trials=30)
print("Best SAC trial:", sac_study.best_trial.params)

# Final training with best params
best_ppo_params = ppo_study.best_trial.params
best_sac_params = sac_study.best_trial.params

# 1. Create the environment and train the agents
total_timesteps = 10000
env, ppo_agent, sac_agent = create_env_and_train_agents(training_data,
total_timesteps)

# 2. Test & visualize the agents
n_tests = 1000
agents = {
    'PPO Agent': ppo_agent,
    'SAC Agent': sac_agent,
}
test_and_visualize_agents(env, agents, training_data, n_tests=n_tests)

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def compare_and_plot_agents(agents_metrics, labels, risk_free_rate=0.0):
    # Function to compare returns, standard deviation, and sharpe ratio of agents
```

```
def compare_agents(agents_metrics, labels):
    returns = []
    stds = []
    sharpe_ratios = []

    for metrics in agents_metrics:
        net_worths = metrics['net_worths']
        # Calculate daily returns
        daily_returns = np.diff(net_worths) / net_worths[:-1]
        avg_return = np.mean(daily_returns)
        std_return = np.std(daily_returns)
        sharpe_ratio = (avg_return - risk_free_rate) / std_return

        returns.append(avg_return)
        stds.append(std_return)
        sharpe_ratios.append(sharpe_ratio)

    df = pd.DataFrame({
        'Agent': labels,
        'Return': returns,
        'Standard Deviation': stds,
        'Sharpe Ratio': sharpe_ratios
    })

    return df

# Compare agents
df = compare_agents(agents_metrics, labels)
# Sort the dataframe by sharpe ratio
df_sorted = df.sort_values(by='Sharpe Ratio', ascending=False)

# Display the dataframe
display(df_sorted)
```

```
    # Plot bar chart for sharpe ratio
    plt.figure(figsize=(12, 6))
    plt.bar(df_sorted['Agent'], df_sorted['Sharpe Ratio'])
    plt.title('Sharpe Ratio Comparison')
    plt.xlabel('Agent')
    plt.ylabel('Sharpe Ratio')
    plt.show()

# Create the environment using DummyVecEnv with test data
test_env = DummyVecEnv([lambda: StockTradingEnv(test_data)])

# 2. Test & visualize the agents on the test data
n_tests = 1000
test_agents = {
    'PPO Agent': ppo_agent,
    'SAC Agent': sac_agent,
}
test_and_visualize_agents(env, test_agents, test_data, n_tests=n_tests)

# 3. Compare the agents' performance on the test data ( returns, standard deviation,
and sharpe ratio )
test_agents_metrics = [test_agent(env, agent, test_data, n_tests=n_tests,
visualize=False) for agent in test_agents.values()]
compare_and_plot_agents(test_agents_metrics, list(test_agents.keys()))

ppo_agent.model.save("ppo_trading_model.zip")
sac_agent.model.save("sac_trading_model.zip")

from stable_baselines3 import PPO, SAC

# Recreate the environment
env = DummyVecEnv([lambda: StockTradingEnv(training_data)])

# Load the trained model
```

```
ppo_agent.model = PPO.load("ppo_trading_model.zip", env=env)
sac_agent.model = SAC.load("sac_trading_model.zip", env=env)

class StockTradingEnvDiscrete(gym.Env):
    metadata = {'render_modes': ['human']}

    def __init__(self, stock_data, transaction_cost_percent=0.0005):
        super(StockTradingEnvDiscrete, self).__init__()

        self.stock_data = {ticker: df for ticker, df in stock_data.items() if not df.empty}
        self.tickers = list(self.stock_data.keys())

        if not self.tickers:
            raise ValueError("All provided stock data is empty")

        sample_df = next(iter(self.stock_data.values()))
        self.n_features = len(sample_df.columns)

        # Discrete action space: 0 = Hold, 1 = Buy, 2 = Sell per stock
        self.action_space = spaces.Discrete(3)

        self.obs_shape = self.n_features * len(self.tickers) + 2 + len(self.tickers) + 2
        self.observation_space = spaces.Box(low=-np.inf, high=np.inf,
shape=(self.obs_shape,), dtype=np.float32)

        self.initial_balance = 6000
        self.transaction_cost_percent = transaction_cost_percent
        self.reset()

    def reset(self, seed=None, options=None):
        super().reset(seed=seed)
        self.balance = self.initial_balance
        self.net_worth = self.initial_balance
        self.max_net_worth = self.initial_balance
        self.shares_held = {ticker: 0 for ticker in self.tickers}
        self.total_shares_sold = {ticker: 0 for ticker in self.tickers}
        self.total_sales_value = {ticker: 0 for ticker in self.tickers}
        self.current_step = 0
        self.max_steps = max(0, min(len(df) for df in self.stock_data.values()) - 1)
        return self._next_observation(), {}

    def _next_observation(self):
        frame = np.zeros(self.obs_shape)
        idx = 0
        for ticker in self.tickers:
            df = self.stock_data[ticker]
            if self.current_step < len(df):
                frame[idx:idx+self.n_features] = df.iloc[self.current_step].values
            else:
                frame[idx:idx+self.n_features] = df.iloc[-1].values
            idx += self.n_features

        frame[-4-len(self.tickers)] = self.balance
        frame[-3-len(self.tickers):-3] = [self.shares_held[ticker] for ticker in self.tickers]
        frame[-3] = self.net_worth
        frame[-2] = self.max_net_worth
        frame[-1] = self.current_step

        return frame

    def step(self, action):
      self.current_step += 1
      done = self.current_step > self.max_steps

      ticker = self.tickers[0]
      df = self.stock_data[ticker]
      safe_step = min(self.current_step, len(df) - 1)
      current_price = df.iloc[safe_step]['Close']
```

```
        if action == 1:  # Buy
            shares_to_buy = int(self.balance / current_price)
            cost = shares_to_buy * current_price
            transaction_cost = cost * self.transaction_cost_percent
            self.balance -= (cost + transaction_cost)
            self.shares_held[ticker] += shares_to_buy

        elif action == 2:  # Sell
            shares_to_sell = self.shares_held[ticker]
            sale = shares_to_sell * current_price
            transaction_cost = sale * self.transaction_cost_percent
            self.balance += (sale - transaction_cost)
            self.shares_held[ticker] = 0
            self.total_shares_sold[ticker] += shares_to_sell
            self.total_sales_value[ticker] += sale

        self.net_worth = self.balance + self.shares_held[ticker] * current_price
        self.max_net_worth = max(self.max_net_worth, self.net_worth)
        reward = self.net_worth - self.initial_balance
        done = done or self.net_worth <= 0
        return self._next_observation(), reward, done, False, {}

    def render(self, mode='human'):
        profit = self.net_worth - self.initial_balance
        print(f'Step: {self.current_step}')
        print(f'Balance: {self.balance:.2f}')
        for ticker in self.tickers:
            print(f'{ticker} Shares held: {self.shares_held[ticker]}')
        print(f'Net worth: {self.net_worth:.2f}')
        print(f'Profit: {profit:.2f}')

def create_env_and_train_agents(training_data, total_timesteps,
                                dqn_params=None):
```

```
    env = DummyVecEnv([lambda: StockTradingEnvDiscrete(training_data)])
    dqn_agent = DQNAgent(env, total_timesteps, hyperparams=dqn_params)
    return env, dqn_agent

import optuna
from stable_baselines3 import DQN
from stable_baselines3.common.evaluation import evaluate_policy
from stable_baselines3.common.vec_env import DummyVecEnv

def dqn_objective(trial):
    # Define the hyperparameter search space for DQN
    hyperparams = {
        'learning_rate': trial.suggest_float('learning_rate', 1e-5, 1e-3, log=True),
        'gamma': trial.suggest_float('gamma', 0.90, 0.9999),
        'buffer_size': trial.suggest_int('buffer_size', 10000, 100000),
        'exploration_fraction': trial.suggest_float('exploration_fraction', 0.1, 0.5),
        'exploration_final_eps': trial.suggest_float('exploration_final_eps', 0.01, 0.1),
        'train_freq': trial.suggest_int('train_freq', 1, 32),
        'batch_size': trial.suggest_categorical('batch_size', [32, 64, 128, 256]),
        'target_update_interval': trial.suggest_int('target_update_interval', 100, 1000),
        'learning_starts': trial.suggest_int('learning_starts', 1000, 5000),
    }

    # Create environment (must use discrete actions!)
    env = DummyVecEnv([lambda: StockTradingEnvDiscrete(training_data)])

    # Train DQN agent
    model = DQN("MlpPolicy", env, verbose=0, **hyperparams)
    model.learn(total_timesteps=5000)

    # Evaluate the model
    mean_reward, _ = evaluate_policy(model, env, n_eval_episodes=5, warn=False)
    return mean_reward
```

```
# Run Optuna study for DQN
dqn_study = optuna.create_study(direction='maximize')
dqn_study.optimize(dqn_objective, n_trials=30)

print("Best DQN trial:", dqn_study.best_trial.params)

# 1. Create the environment and train the agents
total_timesteps = 10000
env, dqn_agent = create_env_and_train_agents(training_data, total_timesteps)

# 2. Test & visualize the agents
n_tests = 1000
agents = {
   'DQN Agent': dqn_agent,
}
test_and_visualize_agents(env, agents, training_data, n_tests=n_tests)


# Create the environment using DummyVecEnv with test data
test_env = DummyVecEnv([lambda: StockTradingEnvDiscrete(test_data)])

# 2. Test & visualize the agents on the test data
n_tests = 1000
test_agents = {
   'DQN Agent': dqn_agent,
}
test_and_visualize_agents(test_env, test_agents, test_data, n_tests=n_tests)

# 3. Compare the agents' performance on the test data ( returns, standard deviation,
and sharpe ratio )
test_agents_metrics = [test_agent(env, agent, test_data, n_tests=n_tests,
visualize=False) for agent in test_agents.values()]
compare_and_plot_agents(test_agents_metrics, list(test_agents.keys()))
```