PRESCRIPTIVE ANALYTICS BASED FRAMEWORK FOR INDIAN STOCK MARKET THROUGH A DETAILED STUDY OF DATA SCIENCE TECHNIQUES

by

Sulekha Aloorravi, B.Tech, PGP in Big Data and Machine Learning

DISSERTATION

Presented to the Swiss School of Business and Management Geneva

In Partial Fulfillment

Of the Requirements

For the Degree

DOCTOR OF BUSINESS ADMINISTRATION

SWISS SCHOOL OF BUSINESS AND MANAGEMENT GENEVA

DECEMBER, 2024

PRESCRIPTIVE ANALYTICS BASED FRAMEWORK FOR INDIAN STOCK MARKET THROUGH A DETAILED STUDY OF DATA SCIENCE TECHNIQUES

by

Sulekha Aloorravi

APPROVED BY

Dissertation chair

RECEIVED/APPROVED BY:

Admissions Director

Dedication

This thesis is dedicated to my loving husband, Dileep V, who has been my unwavering source of support and encouragement throughout my academic journey. Your belief in me has been my motivation, and your love has been my strength.

I also dedicate this work to my esteemed professor, Dr. Vijayakumar Varadarajan, whose guidance and wisdom have shaped my intellectual growth and inspired me to pursue excellence.

Lastly but most importantly, I dedicate this thesis to my nephew, Sai Sathvik, whose exceptional brilliance and love always inspires me to do better.

Acknowledgements

I would like to thank my thesis advisor, Prof. Dr. Vijayakumar Varadarajan, for his invaluable guidance, patience, and unwavering support throughout the entire research process. His expertise and mentorship have been instrumental in shaping the direction of this thesis, and I am truly grateful for his dedication to my academic growth.

I am also indebted to the members of my thesis committee, for their valuable insights, constructive feedback, and expertise in their respective fields. Their collective contributions have significantly enriched the quality of this work.

I would like to express my gratitude to my family for their unwavering belief in me and their constant encouragement. Your love and support have been my anchor throughout this process.

Finally, I want to acknowledge the numerous authors, researchers, and scholars whose work I have referenced and drawn upon during my research. Their contributions have formed the foundation upon which this thesis is built.

ABSTRACT

PRESCRIPTIVE ANALYTICS BASED FRAMEWORK FOR INDIAN STOCK MARKET THROUGH A DETAILED STUDY OF DATA SCIENCE TECHNIQUES

Sulekha Aloorravi
2024

Dissertation Chair:
Co-Chair:

This study will focus on developing a prescriptive framework that applies various techniques of data science, machine learning and deep learning to help individual investors understand market opportunities and make educated decisions in the Indian stock market. The study will review existing methodologies and research available in the field of Indian stock markets, identifying market opportunities and applying data science techniques.

The study will rely on publicly available data sets related to stock prices and financial reports of individual firms. The study will find answers for research questions and will prescribe a framework for educated decision making in identifying market opportunities. Finally, the study will conclude the results. The study will also recommend further studies and identify limitations of the study.

TABLE OF CONTENTS

List of Tables viii

List of Figures ix

CHAPTER I: INTRODUCTION 1

1.1 Introduction 1
1.2 Research Problem 2
1.3 Purpose of Research 2
1.4 Significance of the Study 2
1.5 Research Purpose and Questions 3

CHAPTER II: REVIEW OF LITERATURE 5

2.1 Theoretical Framework 5
2.2 Preliminary Literature Review Objectives 5
2.3 Methodology Adopted to Perform this Literature Review 6
2.4 Publisher Analysis 8
2.5 Relevance to Research Topic 9
2.6 Citation Analysis 10
2.7 Chronology of Publications 12
2.8. Research Question Synthesis 15
2.9 Comparative Analysis 33
2.10 Summary 37

CHAPTER III: METHODOLOGY 46

3.1 Overview of the Research Problem 46
3.2 Research Question Formulation 47
3.3 Research Design 47
3.4 Data Collection 47
3.5 Data Analysis 48
3.6 Framework Development 48
3.7 Framework Validation 48

CHAPTER IV: SLRPO INVESTMENT FRAMEWORK − DESIGN AND DEVELOPMENT 50

4.1 Introduction 50
4.2 Data Acquisition Module 52
4.3 Data Load Module 55
4.4 Indicators Module 56
4.5 Preprocessing Module 109

4.6 Weight Generator Module 113
4.7 Strength Score Module 123
4.8 Utility Functions Module 126
4.9 Strategy Domain Classification 135
4.10 SLRPO Base Strategies 137
4.11 Base Strategy − UCB1 Module 137
4.12 Base Strategy − Policy Network Module 149
4.13 Base Strategy − Policy Gradient Module 155
4.14 Main 8 Strategies of SLRPO and Portfolio Creation 160
4.14 Allocation − UI Module 173

CHAPTER V: RESULTS 181

5.1 Research Question 181
5.2 Research Sub Question One 182
5.3 Research Sub Question Two 183
5.4 Research Sub Question Three 183

CHAPTER VI: DISCUSSION 185

6.1 Discussion of Results of SLRPO Investment Framework 185
6.2 Summary of Findings 186

CHAPTER VII: SUMMARY, IMPLICATIONS, AND RECOMMENDATIONS 189

7.1 Summary 189
7.2 Implications 189
7.3 Recommendations for Future Research 189
7.4 Conclusion 190

REFERENCES 192

LIST OF TABLES

Table 4.1 NIFTY 50 Stocks 161
Table 4.2 Strategy 1 Portfolio Allocation of all NIFTY 50 Stocks 162
Table 4.3 Strategy 2 Portfolio Allocation of all NIFTY 50 Stocks 164
Table 4.4 Strategy 3 Portfolio Allocation of all NIFTY 50 Stocks 166
Table 4.5 Strategy 4 Portfolio Allocation of all NIFTY 50 Stocks 167
Table 4.6 Strategy 5 Portfolio Allocation of all NIFTY 50 Stocks 169
Table 4.7 Strategy 6 Portfolio Allocation of all NIFTY 50 Stocks 170
Table 4.8 Strategy 7 Portfolio Allocation of all NIFTY 50 Stocks 172
Table 4.9 Strategy 8 Portfolio Allocation of all NIFTY 50 Stocks 173
Table 6.1 Returns and Strategy Metrics 185
Table 6.2 Opportunity Cost of Foregoing SLRPO 187

LIST OF FIGURES

Figure 1.1 Reasons for Households not Investing in Mutual Funds 1
Figure 1.2 Reasons for Households not Investing in Equities 2
Figure 2.1 Total Research Papers by Search Strings 6
Figure 2.2 Total Research Papers by Publishers 8
Figure 2.3 Total Research Papers by Relevance 9
Figure 2.4 Total Relevant Research Papers by Publishers 10
Figure 2.5 Top 20 Most Cited Relevant Papers 11
Figure 2.6 Total Numbers of Citations of Relevant Papers by Publisher 11
Figure 2.7 Top Publisher Citations of Relevant Papers by Year 12
Figure 2.8 Publisher Citations of Relevant Papers by Recent Time 14
Figure 2.9 Comparative Analysis Topics 34
Figure 2.10 Category 1: Comparisons of Author and Year 35
Figure 2.11 Category 2: Comparing Areas of Focus 36
Figure 2.12 Category 3: Comparing Methodologies Applied 36
Figure 2.13 Category 4: Comparing Opportunities for Future Work 37
Figure 4.1 SLRPO Architecture 51
Figure 4.2 Stocks Features Data after Preprocessing 113
Figure 4.3 Normalized Features of Stocks Data 118
Figure 4.4 Strength Scores for NIFTY 50 Stocks 126
Figure 4.5 Strategy 1 - Results 177
Figure 4.6 Strategy 2 − Results 177
Figure 4.7 Strategy 3 − Results 178
Figure 4.8 Strategy 4 − Results 178
Figure 4.9 Strategy 5 − Results 179
Figure 4.10 Strategy 6 − Results 179
Figure 4.10 Strategy 7 − Results 180
Figure 4.11 Strategy 8 − Results 180

CHAPTER I:

INTRODUCTION

1.1 Introduction

Indian stock market is an exchange where investors can buy and sell stocks, bonds and other securities in India. Indian stock market is always buzzing with activities during the business day starting from the time of its opening bell at 9 am to its closing time at 3:30 pm IST. Stock market is an arena where an Indian house hold can invest money as an individual investor and gain profits or incur losses. Even though the stock market is busy during the trading day, SEBI investor survey 2015 states the following:

A number of studies and surveys indicate that a very small proportion of Indian population invests in securities market while countries with matured stock markets have participation rates of around 50 percent (SEBI INVESTOR SURVEY 2015, 2015, p. 1). There are a number of reasons why an Indian household does not invest in the stock markets either in Mutual Funds or in Equities.

The reasons for why do households not invest in mutual funds are listed in the following table (SEBI INVESTOR SURVEY 2015, 2015, p. 51):

Reason	Proportion
Not sure about safety of investments	33.3%
Inadequate returns	25.6%
Inadequate information	14.1%
Lack of expertise	12.6%
Investment not very liquid	8.3%
Others	6.1%

Figure 1.1 Reasons for Households not Investing in Mutual Funds

The reasons for why households do not invest in equities are listed in the following table (SEBI INVESTOR SURVEY 2015, 2015, p. 52):

Reason	Proportion
Not sure about safety of investments	25.6%
Inadequate returns	15.4%
Inadequate information	21.3%
Lack of expertise	15.6%
Investment not very liquid	13.4%
Others	8.6%

Figure 1.2 Reasons for Households not Investing in Equities

Any investment model relies on capital for the investment to grow. The investment model such as Indian stock market relies on public to invest their savings and the investments thus acts as a source for the growth of both the individual investor and the country's economy. In India, a very small proportion of the population invests in stock markets while there is a large proportion with capital and do not invest due to the lack of adequate knowledge or due to the lack of adequate understanding on how to utilize the public information available on the markets.

1.2 Research Problem

Lack of expertise or inadequate information can lead to Indian households with capital, losing opportunities to invest in the Indian stock markets. The scope of this research is to study in detail and apply various techniques of data science and machine learning to identify the lost opportunities and thus provide a framework that could help in educated decision-making by making use of publicly available data.

1.3 Purpose of Research

The purpose of this research is to perform a detailed analysis that would ultimately focus on the research of how to utilize the public information and convert it into the knowledge required for investing in stock markets and thus studies the research problem in scope and its corresponding research questions.

1.4 Significance of the Study

This study is aimed to address the following critical issues faced by Indian households holding capital but do not invest at the right time and in the right financial assets due to lack of adequate information:

Mitigating Financial Losses:

The main problem in man Indian households is that they have that could be invested in the stock markets, but a lack of expertise or access to adequate information often leads to missed investment opportunities. This study focuses on identifying these missed opportunities. By doing so, it has the potential to help households make informed decisions, thereby reducing the risk of financial losses.

Empowering Individual Investors:

Providing a framework for educated decision-making will be of great help to individual investors in situationa where they might not have access to expensive financial advisory services. This study has the potential to democratize investment by enabling households to independently evaluate and seize investment opportunities in the Indian stock markets by making use of data and thinking in the right direction.

Using advanced technologies such as data science and machine learning to study the stock markets and taking informed decisions, not only enables the Indian investors to make educated decision in the stock markets investment process but also helps in the economic growth of the country.

1.5 Research Purpose and Questions

How can data science techniques and methodologies help in developing a decision-making framework that helps in identifying the opportunities to invest in the Indian stock markets?

1.5.1 Research Sub Questions

1.5.1.1. What are the key aspects/variables that need to be considered in understanding opportunities?

1.5.1.2. What are the various techniques or algorithms that can be explored to identify opportunities?

1.5.1.3. How can prescriptive analytics help in developing a decision-making framework in the domain of Indian stock markets?

CHAPTER II:

REVIEW OF LITERATURE

2.1 Theoretical Framework

The main objective of this literature review is to provide a comprehensive review of literatures and practices in relation to the research questions and come up with a framework that helps to identify investment opportunities by understanding various market data parameters and economic factors.

Particularly, the study has the following sub-objectives:

1. To provide a comprehensive review of sources and key aspects/variables that needs to be considered in understanding investment opportunities in Indian stock markets;

2. To analyze various techniques or algorithms that can be explored to identify opportunities in Indian stock markets;

3. To review current methodologies and research in the domain of Indian stock markets and analytics.

4. To outline a decision-making framework in the domain of Indian stock markets.

The result of this study will be valuable to the industry practitioners as well as individual investors in developing better practice and tools for investment decision making.

2.2 Preliminary Literature Review Objectives

The preliminary literature review focused on researching on the following topics:

1. Understanding the importance of opportunity cost in investments,
2. Exploring the applications of data science and comprehending the key features in the field of investment analysis

2.3 Methodology Adopted to Perform this Literature Review

2.3.1 Search Tokens

Literature review for this research started by searching and exploring for existing research articles on the following search tokens in google scholar:

1. Machine learning in stock market
2. Deep learning in stock market
3. Data Science in stock market
4. Opportunity analysis in stock markets
5. Stock market
6. Indian stock market analysis

The overall landscape of research materials on the search tokens are distributed as follows:

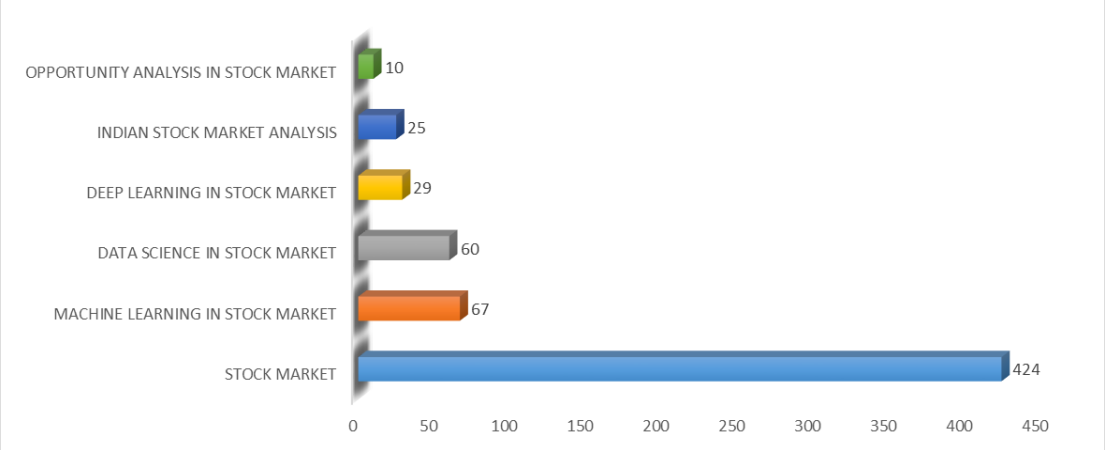

Figure 2.1 Total Research Papers by Search Strings

2.3.1.1 Search criteria 1

There were around **10** research materials and existing literature available on the concept covering opportunity analysis in stock markets that also covers string combinations such as *opportunity + stock + market* and *opportunity analysis + stock + market*.

2.3.1.2 Search criteria 2

There were around **25** research materials and existing literature available on the concept covering Indian stock market analysis that also covers string combinations such *as Indian + stock + market, Indian + stock + analysis, Indian + market + analysis* and *Indian + stock + market + analysis*.

2.3.1.3 Search criteria 3

There were around **29** research materials and existing literature available on the concept covering deep learning in stock market that also covers string combinations such as *deep + learning + stock, deep + learning + stock + market* and *deep + learning + market*.

2.3.1.4 Search criteria 4

There were around **60** research materials and existing literature available on the concept covering data science in stock market that also covers string combinations such as *data + science + stock, date + science + stock + market* and *data + science + market*.

2.3.1.5 Search criteria 5

There were around **67** research materials and existing literature available on the concept covering machine learning in stock market that also covers string combinations such as *machine + learning + stock, machine + learning + stock + market* and *machine + learning + market*.

2.3.1.6 Search criteria 6

The last, wider and the most generic search criteria was to find out all possible existing literature on stock markets or capital markets. There were around **424** research materials and existing literature available on the concepts covering stock market that also covers string combinations such as *stock + market,* capital *+ market and market + investment*.

The preceding statistics is based on the search tokens and the relevance of these gathered materials will further be discussed in the upcoming sections.

The publishers of the gathered research materials will be analyzed further.

2.4 Publisher Analysis

A total of **615** existing literature in the form of research papers, journals and books were collected and reviewed to understand the relevance of the existing literature by analyzing them in relation to my research in scope.

The total numbers of research papers by publishers are listed as follows:

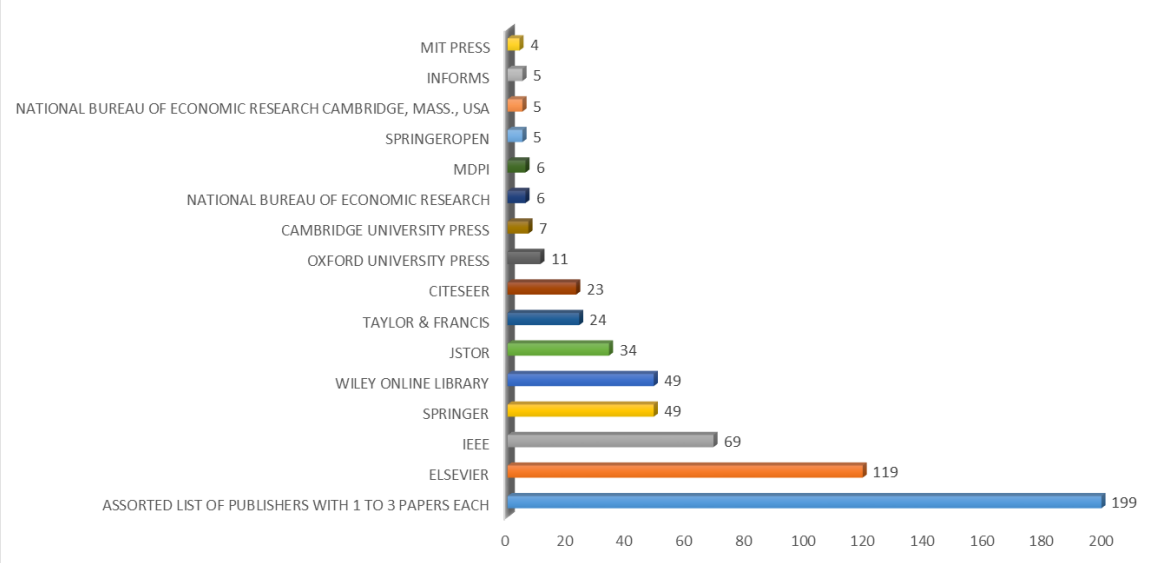

Figure 2.2 Total Research Papers by Publishers

The research papers covering the search tokens for this research are majorly published by Elsevier, IEEE, Springer, Wiley online library which covers journals on finance and JSTOR which covers journals of business. In the *Figure 2.2*, there is also an assorted list of multiple publishers that covers around a range of 1 to 3 research articles each. Our focus of study is mainly on the articles published by Elsevier, IEEE, Springer, Wiley online library and JSTOR.

The relevance of these 615 research papers and journals in relationship to the research in scope will be analyzed further.

2.5 Relevance to Research Topic

The abstract, methodology and conclusion of the 615 research papers identified for this literature review are studied to understand its relevance to the research in scope. Post analysis, it is identified that 407 research papers gathered were not relevant to the research topic under study and 208 have relevance from at least one perspective. The relevance of topics is represented as follows:

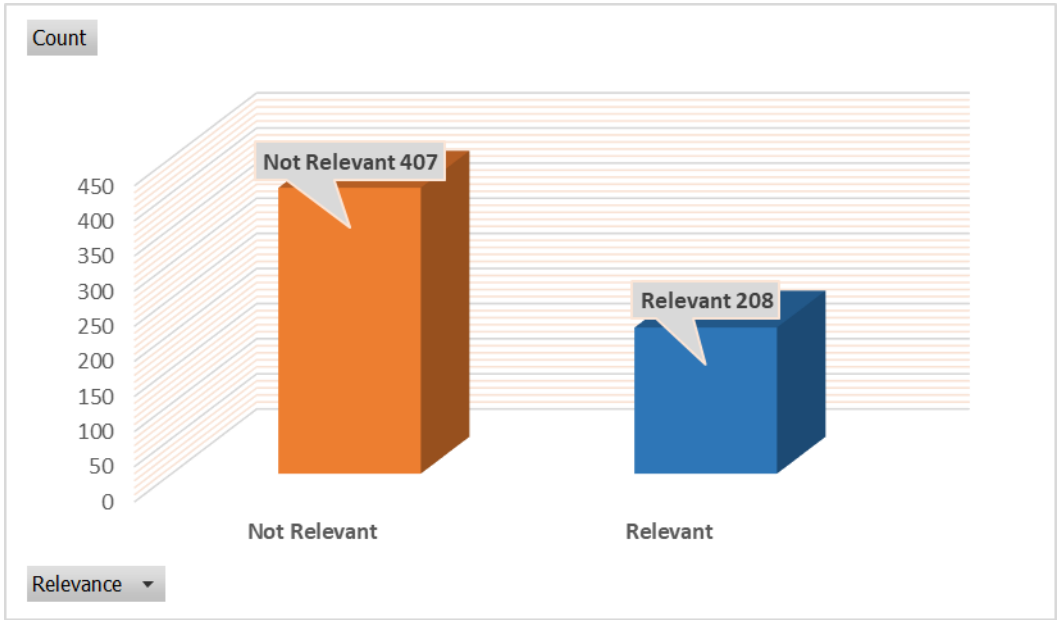

Figure 2.3 Total Research Papers by Relevance

The relevant papers were further reviewed, and the statistics is listed down in the order of their respective publishers.

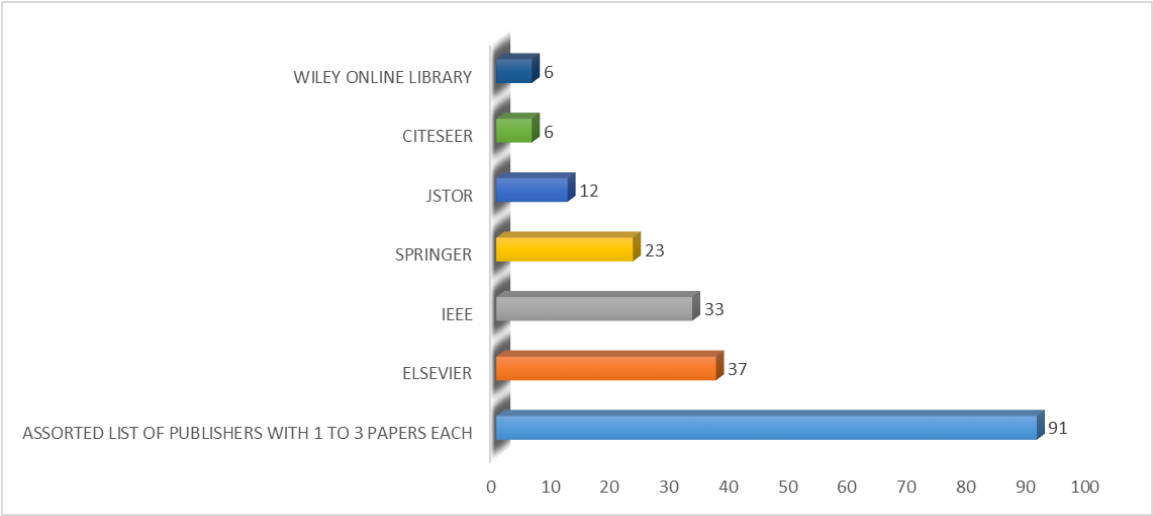

Figure 2.4 Total Relevant Research Papers by Publishers

The research papers covering the relevance for this research are majorly published by Elsevier, IEEE, Springer, JSTOR which covers journals of business, and Citeseer and Wiley online library which covers journals on finance. In the *Figure 2.4*, there is also an assorted list of multiple publishers that covers around a range of 1 to 3 research articles each. Our focus of study is mainly on the articles published by Elsevier, IEEE, Springer, JSTOR, and Citeseer and Wiley online library.

The number of citations of the relevant literature will be analyzed further.

2.6 Citation Analysis

Number of citations on a research paper denotes the recognition of a research in that field and a possible implementation of the concepts and a higher scope for enhancements. Analyzing the citations of various literatures gathered for this research is the next important step of this literature review.

Top 20 research titles with their corresponding number of citations are represented as follows.

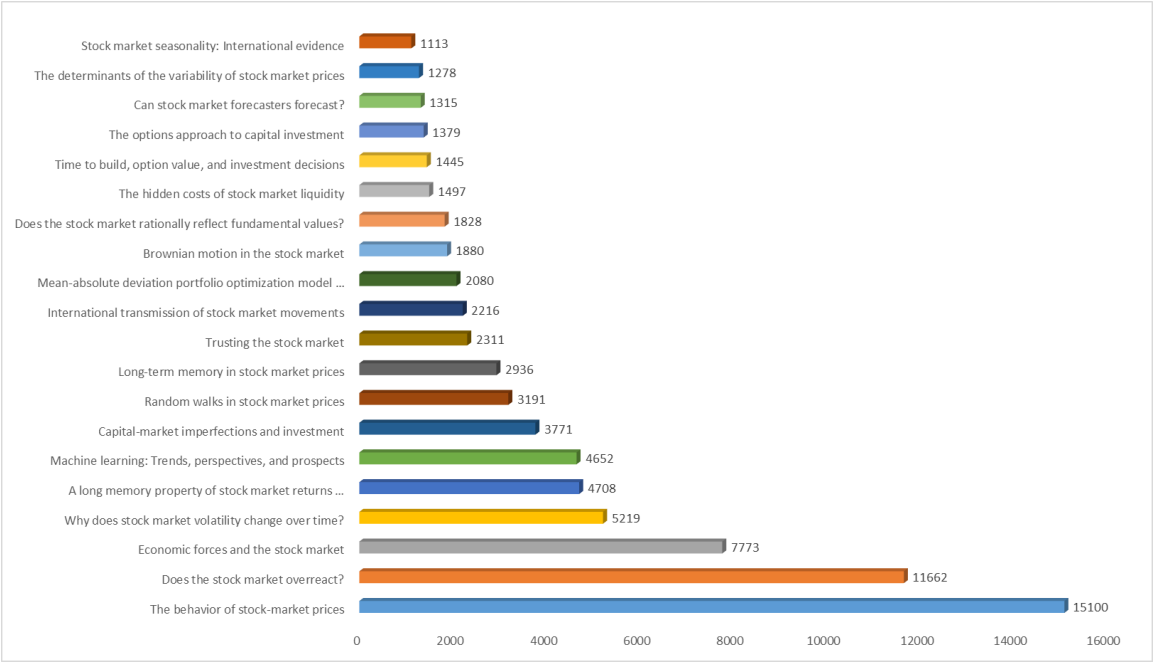

Figure 2.5 Top 20 Most Cited Relevant Papers

The literature citations are further analyzed by reviewing the statistics by publisher in the *Figure 2.6*. The maximum numbers of aggregated citations were on the papers published by JSTOR which publishes journals for business followed by Wiley online library which publishes journals for finance. This list is further followed by Elsevier, IEEE, Springer and Citeseer in a descending order.

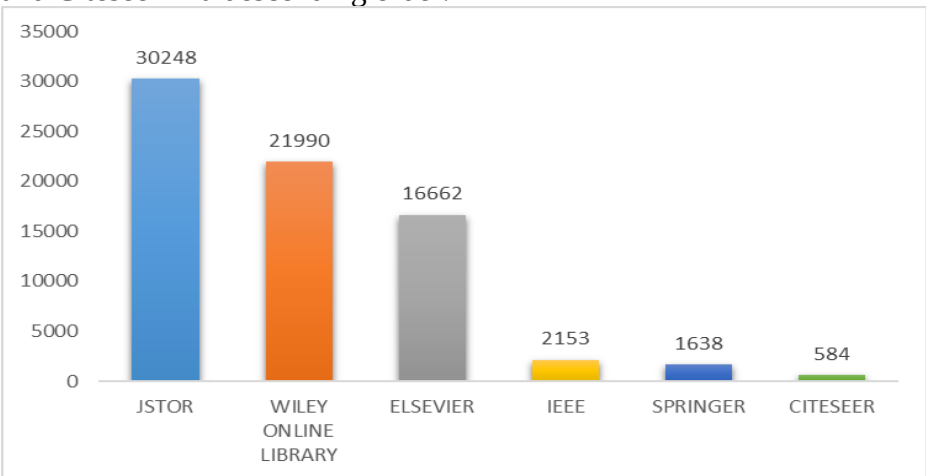

Figure 2.6 Total Numbers of Citations of Relevant Papers by Publisher

The chronology of publications with relevance to the year of publishing will be analyzed further.

2.7 Chronology of Publications

The greatest number of citations were from JSTOR and Wiley online library as mentioned in the Figure 2.6. The relevant literatures from these publishers were further analyzed to identify how recent the research was.

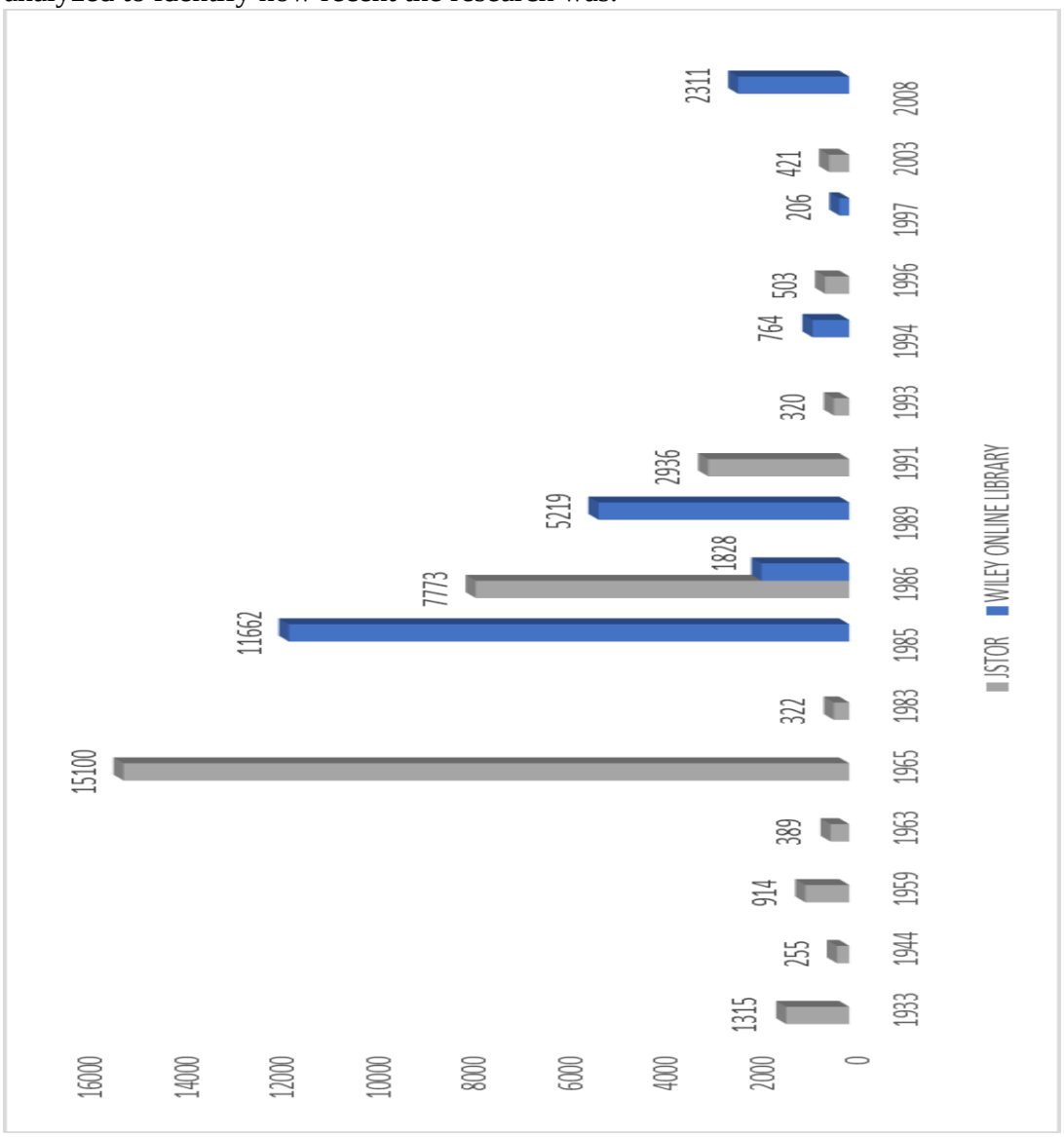

Figure 2.7 Top Publisher Citations of Relevant Papers by Year

All the research papers of JSTOR and Wiley that were identified as relevant papers by studying their abstract, methodology and conclusion happens to be published between the years 1933 to 2008. This indicates that there were no recent papers or journals from these publishers that can be shortlisted for this research. However, the already shortlisted papers will still be studied while synthesizing research questions based on the knowledge that can be derived from these papers and journals.

The most recent journal publishers were further analyzed for the number of citations.

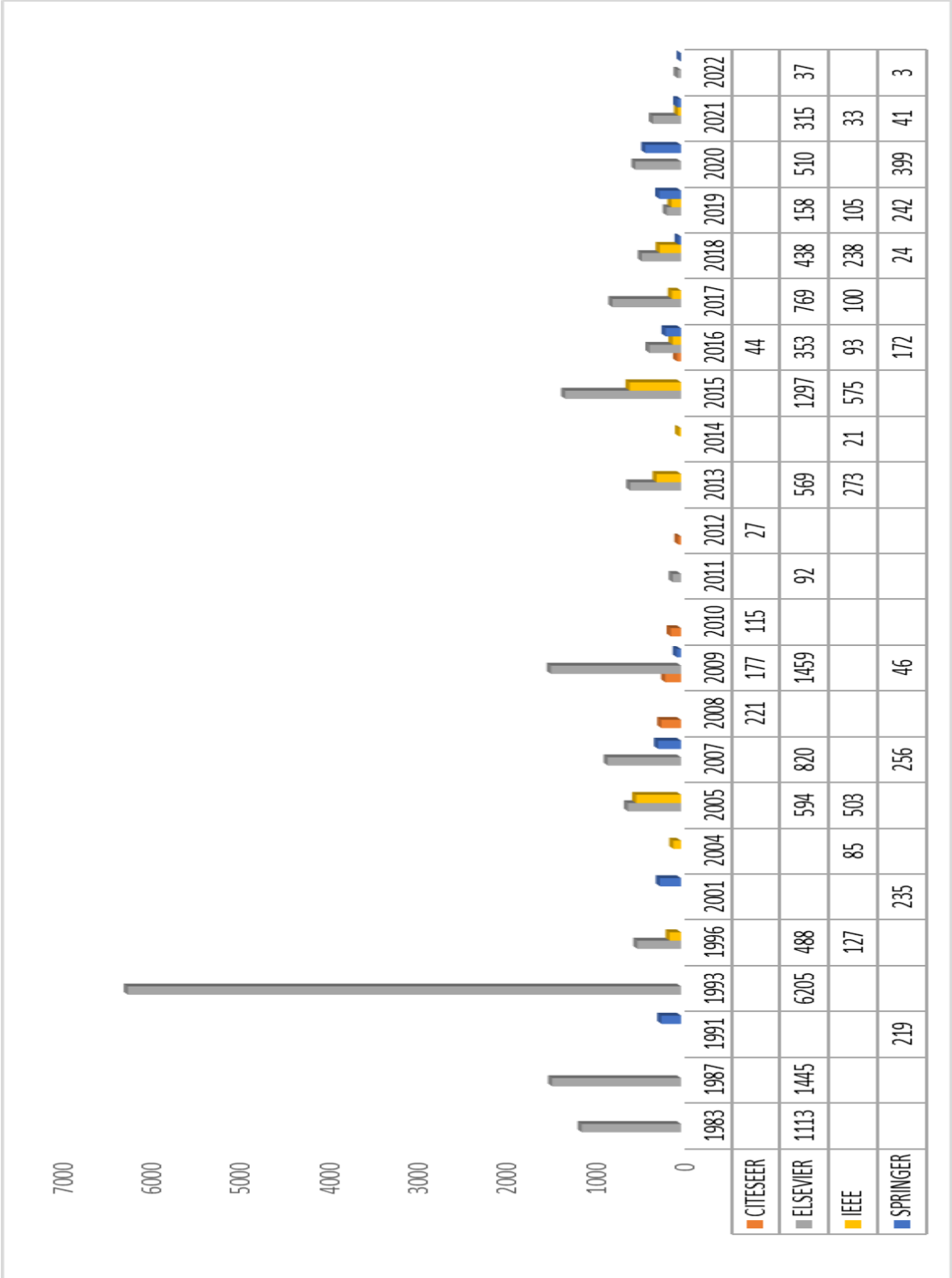

Figure 2.8 Publisher Citations of Relevant Papers by Recent Time

Elsevier, IEEE and Springer have the greatest number of citations on literature that were published between 2001 and 2022.

With these initial efforts to identify, analyze, review and shortlist the most relevant, most cited, highly reputed and recent literature, further analysis will be performed to synthesize the research questions.

2.8. Research Question Synthesis

The research questions for this research in scope are already defined in sections *1.5 Level 1* and *1.5.1 Level 2*.

The research sub questions under each of the Level 2 Research questions will be defined as follows and studied in detail:

1. What were the problems studied or researched?
2. What type of data is used to understand opportunities?
3. What methodologies were applied to solve the problem?
4. What were the opportunities for future work?

The Level 2 research questions (*Level 2*) are divided into three main concepts as defined below and the related research is synthesized under each concept.

1. Understanding the importance of opportunity cost in investments (covered in section 2.8.1 – Study 1, 2 & 3).
2. Exploring the applications of data science in the field of investment analysis (covered in section 2.8.2 – Study 4 to 9).
3. Comprehending the key features considered in the study of investment decision making (covered in section 2.8.3 – Study 4 to 9).

2.8.1 Understanding the importance of opportunity cost in investments

Study 1

Literature

a. What were the problems studied or researched?

The first study focuses on investigating decision making under situations where there is opportunity-cost time pressure. The authors were interested in how people adapt their decision-making strategies when faced with opportunity-cost environments and in the possible limits or even failures in adaptivity that may arise in such environments. They focused on how people decide when there are conflicts between the processing implications of opportunity costs and the implications of other variables such as goals and the degree of conflict (intercorrelation) between outcomes (Payne et al., 1996).

b. What methodologies were applied to solve the problem?

The adaptive decision maker: an effort/accuracy framework is discussed in this literature. Stronger tests of the effort/accuracy framework have been conducted by using computer simulations of the effort and accuracy characterizing typical decision strategies to generate hypotheses about processing patterns and then carrying out experimental studies of actual decision behavior to test these hypotheses. The focuses of these experiments were how opportunity-cost time pressure, along with variations in goals and context factors, influences the details of decision processing. Thus, the authors utilized a computerized system for collecting process-tracing data as well as choice data.

c. What type of data is used to understand opportunities?

The subjects' information acquisitions, response times and choices were monitored using the Mouselab software system. Probabilities and pay offs were captured for various scenarios and outcomes using this software.

d. What were the opportunities for future work?

The results of the research imply that strategy changes are needed under opportunity-cost time pressure. When deciding in high-velocity environments, the decision maker should focus on breadth of evaluation rather than on depth of evaluation.

More research which directly examines how alternative prescriptions for decision making fare in high- and low-velocity environments is needed (Payne et al., 1996).

e. Critical Analysis

The time criticality of decision making is more relevant and directly correlated in cases of market investment decisions since there is always a time factor involved for investors either when a decision needs to be made for buying, selling or holding a security or when a decision needs to be made on investing in the markets itself.
All these decisions come with a cost, and it is the cost of losing an opportunity or delaying an opportunity which in turn can be derived as the opportunity cost. It is also to be noted that the data used for this study comes from a simulated environment which gives an opportunity to study the behavior of real-life data with additional parameters.

Study 2

Literature

a. What were the problems studied or researched?

The second study focuses on opportunity cost and choice. The most important consequence of the relationship between choice and opportunity cost is the ex-ante or forward-looking property that cost must carry in this setting. Opportunity cost, the value placed on the rejected option by the chooser, is the obstacle to choice; it is that which must be considered, evaluated, and ultimately rejected before the preferred option is chosen. Opportunity cost in any particular choice is, of course, influenced by prior choices that have been made, but with respect to this choice itself, opportunity cost is choice-influencing rather than choice-influenced (Buchanan, 1991).

b. What methodologies were applied to solve the problem?

The research in this study is not focused on data collection and analysis. It is rather theoretical and focused on explaining the implications of opportunity cost in

decision making through economic principals and thought experiments (Buchanan, 1991).

c. What type of data is used to understand opportunities?

The data collection process is not employed in since the author focused on explaining the theoretical aspects of opportunity cost by presenting a conceptual framework that combines economic theory, rational choice theory, and the study of individual preferences and trade-offs (Buchanan, 1991).

d. What were the opportunities for future work?

With the familiar statement that 'sunk costs are irrelevant', economists acknowledge that the consequences of choices cannot influence choice itself. On the other hand, by their formalized constructions of cost schedules and cost functions, which necessarily imply measurability and objectifiability of costs, economists divorce cost from the choice process.

e. Critical Analysis

One of the strengths of this study is its focus on the theoretical concepts that influences opportunity cost. Similar to time, choice also is an important variable, rather a categorical variable, in investment decision making. Choice also influences the opportunity cost. Opportunity cost varies depending on the choice that was made, and it can either be directly or inversely correlated to the cost and in turn influences the decision-making process.

However, the lack of data collection process and empirical studies becomes a limitation to this study and provides an opportunity to explore further by gathering data that supports the theory behind the framework derived by the author in this study.

Study 3

Literature

a. What were the problems studied or researched?

The study focuses on opportunity cost and behavior. The study was concerned with whether decision makers would use opportunity cost information, or learn to use it, when such information was provided explicitly. As part of the issue, the author also studied whether decision makers' use of opportunity costs was affected by their risk attitude (Hoskin, 1983).

b. What methodologies were applied to solve the problem?

The experiment to understand opportunity cost and behavior was designed using newsboy problem. The experiment is explained as a distinction between ex post and ex ante opportunity cost information is important in selecting the task. Both kinds of information can be relevant to the same problem. Consider, for example, the standard inventory ordering problem, sometimes called the newsboy problem. The newsboy has information concerning the distribution of demand, the selling price, and the costs of a newspaper. Accordingly, the expected opportunity costs for a given decision can be calculated. In an ex-post sense, the newsboy can also determine the actual opportunity cost at the end of the day by keeping track of the number of customers who ask for the newspaper when it is out of stock. This assumes, of course, that the stock is not visible to the customer (Hoskin, 1983).

c. What type of data is used to understand opportunities?

The subjects used were 61 M.B.A. and Ph.D. students in a major business school. The study was conducted over a period three days in groups of five to ten. Subjects were randomly assigned to experimental and control groups. There were no significant demographic differences between the experimental groups (Hoskin, 1983).

d. What were the opportunities for future work?

While designing the experiment there were many limitations imposed on the research and study. Further research might need to look at varying some of the factors and constraints imposed in the study. Some of the factors suggested by the literature includes the risk of bankruptcy, reward systems and instability in the environment (Hoskin, 1983).

e. Critical Analysis

Both opportunity cost and behavior of investors towards opportunity cost is important in decision making. In most of the cases the uncertainty of risk taking, and its impact makes the investors not to proceed with an investment. The thought on uncertainty of risk can be minimized when the information or knowledge about opportunity cost is available to the investor. The main challenge in understanding opportunity cost comes up due to lack of enough information on this topic.

2.8.2 Exploring the applications of data science and comprehending the key features in the field of investment analysis

Study 4

Literature

a. What were the problems studied or researched?

The problem of predicting direction of movement of stock and stock price index for Indian stock markets is addressed in the paper. The study compares four prediction models, Artificial Neural Network (ANN), support vector machine (SVM), random forest and Naive-Bayes with two approaches for input to these models. The first approach for input data involves computation of ten technical parameters using stock trading data (open, high, low & close prices) while the second approach focuses on representing these technical parameters as trend deterministic data (Patel et al., 2015).

b. What type of data is used to understand applications?

Ten years of data of total two stock price indices (CNX Nifty, S&P BSE Sensex) and two stocks (Reliance Industries, Infosys Ltd.) from Jan 2003 to Dec 2012 is used in this study. All the data is obtained from <http://www.nseindia.com/> and <http://www.bseindi-a.com/> websites. Ten technical indicators were used as input variables.

c. What methodologies were applied to solve the problem?

Machine learning algorithms such as Artificial Neural Networks (ANN), Support Vector Machine (SVM), Random Forest and Naïve Bayes classifier were applied on the data to learn and predict from the data. Trend Deterministic Data Preparation Layer was proposed in the literature.

d. What were the opportunities for future work?

Apart from the ten technical indicators used in this literature, other macro-economic variables like currency exchange rates, inflation, government policies, interest rates etc. can also be explored. Apart from the categories predicted as ‘up’ or ‘down’ in this literature, authors are also suggesting to explore multiple categories like ‘highly possible to go up’, ‘highly possible to go down’, ‘less possible to go up’, ‘less possible to go down’ and ‘neutral signal’. While this paper deals with short term prediction, the scope of long-term prediction is also an open item for future work.

e. Critical Analysis

In exploring the applications of data science in the field of investment analysis, have studied the research paper to understand the areas where data science is applied in the field of stock markets. The major area discussed in this paper deals with predictive algorithms of machine learning that are applied to predict upward or downward movement of specific stocks and stock market indices. This is also an approach that helps in decision making and this approach also has many opportunities that can lead to future

work and future enhancements in the field of machine learning on investment decision making.

Study 5

Literature

a. What were the problems studied or researched?

The study revolves around utilizing deep learning networks for stock market analysis and prediction. The authors aim to explore the methodology and data representations used in deep learning models and their application to forecast stock market behavior. The study attempts to provide a comprehensive and objective assessment of both the advantages and drawbacks of deep learning algorithms for stock market analysis and prediction. The literature explores the effects of three unsupervised feature extraction methods - principal component analysis, auto encoder, and the restricted Boltzmann machine—on the network's overall ability to predict future market behavior. The literature also offers practical insights and potentially useful directions for further investigation into how deep learning networks can be effectively used for stock market analysis and prediction (Chong et al., 2017).

b. What type of data is used to understand applications?

The study uses financial data, including historical stock prices, trading volumes, and other relevant market indicators, to understand the applications of deep learning networks in stock market analysis. These data sources provide insights into the patterns and dynamics of stock market behavior. High-frequency intraday stock returns data from Korean stock market is used by the authors. The high-frequency data is used to get a large data set and in turn to overcome data-snooping and over-fitting problems inevitable in neural network or any other non-linear models.

c. What methodologies were applied to solve the problem?

The authors of the study propose a deep feature learning-based stock market prediction model. From the lagged stock returns data, the authors generate seven sets of features employing three data representations: principal component analysis, autoencoder, and restricted Boltzmann machine. Three layer deep neural networks were constructed to predict the future stock returns.

d. What were the opportunities for future work?

Factors such as trading volume and the price of a derivative linked to a stock can be considered as variables for future studies. Identifying the relationship between these factors and future price movements can also be considered further.

e. Critical Analysis

In the process of a more detailed research on the existing literature, the research paper is analyzed to learn the existing applications of machine learning algorithms at various stages of investment analysis. This paper deals with predictive analytics on the stock market returns by applying advanced and deep neural networks for the purpose of feature engineering and predictions. The advantage of applying such non-linear algorithms goes a long way in the investment decision making especially when domain knowledge is lesser or unavailable in the process of investment decision making and there is also an existence time pressure that impacts the decision-making process.

Study 6

Literature

a. What were the problems studied or researched?

The problems studied or researched in revolve around stock market forecasting using Bayesian regularized artificial neural networks (BRANNs). The author investigates the application of BRANNs to address the challenges of over fitting and improve the accuracy of stock market predictions. The prediction of stock price movement is

generally considered to be a challenging and important task for financial time series analysis. The complexity in predicting these trend lies in the inherent noise and volatility in daily stock price movement. One day future closing price of individual stocks is predicted using daily market prices and financial technical indicators (Ticknor, 2013).

b. What type of data is used to understand applications?

The study utilizes historical stock market data, including price movements, trading volumes, and other relevant financial indicators, to understand applications of BRANNs in stock market forecasting. The research data used for stock market predictions in the paper was collected for Goldman Sachs Group, Inc. (GS) and Microsoft Corp. (MSFT). The total number of samples for this study was 734 trading days, from 4 January 2010 to 31 December 2012. Each sample consisted of daily information including low price, high price, opening price, close price, and trading volume.

c. What methodologies were applied to solve the problem?

The methodology applied in the research involves the use of artificial neural networks (ANNs) with Bayesian regularization. The author incorporates Bayesian inference and regularization techniques into the neural network architecture to mitigate over fitting issues. The BRANN model is trained using historical data, and Bayesian techniques are employed to update and refine the model parameters based on the observed data. A Bayesian regularized artificial neural network is proposed as a novel method to forecast financial market behavior. The Bayesian regularized network assigns a probabilistic nature to the network weights, allowing the network to automatically and optimally penalize excessively complex models. This technique reduces the potential for over fitting and over training, improving the prediction quality and generalization of the network.

d. What were the opportunities for future work?

The opportunities for future work identified in the research include further exploration of advanced Bayesian regularization techniques to improve the robustness and generalization capabilities of the BRANN model. The author suggests investigating the impact of different hyper parameter choices and regularization priors on the model's performance. Additionally, future research could focus on comparing the performance of BRANNs with other forecasting models in different market conditions and exploring the potential integration of BRANNs with other machine learning techniques, such as ensemble methods or deep learning architectures. Furthermore, the application of BRANNs to different financial markets, data points and technical indicators and time periods present opportunities for future research.

e. Critical Analysis

The study presents a comprehensive analysis of the BRANN model's performance by comparing it to other forecasting models, such as autoregressive integrated moving average (ARIMA) and traditional ANNs. The author provides empirical evidence and evaluation metrics to demonstrate the effectiveness of the BRANN approach, highlighting its superiority in terms of prediction accuracy. The strength of this study is the integration of Bayesian regularization with artificial neural networks (ANNs) for stock market forecasting. The author recognizes the need to address over fitting issues in neural networks, and Bayesian regularization provides a valuable solution. By incorporating Bayesian inference and regularization techniques, the study enhances the generalization capabilities of the model and reduces the risk of over fitting.

However, one limitation of the study is its heavy technical focus and lack of practical application and real-world case studies. While the study presents the theoretical framework and demonstrates the model's performance using historical data, further

exploration of the model's application in real-time trading scenarios or investment decision-making would have enhanced the study's practical relevance.

Additionally, the study does not extensively discuss the interpretability of the BRANN model. Neural networks are known for their black-box nature, and understanding the model's decision-making process and the significance of input features is crucial for its practical implementation. Further research could address this limitation and explore methods to enhance the interpretability of BRANN models.

Study 7

Literature

a. What were the problems studied or researched?

The study revolves around stock market forecasting using ensemble deep Q-learning agents. The authors in the research aims at proposing an ensemble of reinforcement learning approaches which do not use annotations (i.e. market goes up or down) to learn, but rather learn how to maximize a return function over the training stage. In order to achieve this goal, the authors exploit a Q-learning agent trained several times with the same training data and investigate its ensemble behavior in important real-world stock markets (Carta et al., 2021).

b. What type of data is used to understand applications?

The data points collected for this study are mainly the historical stock market data, including price movements, trading volumes, and other relevant financial indicators. The chosen markets are related to futures: the Standard & Poor’s 500 (S&P500), and the German stock index (DAX). These datasets were acquired from a brokerage company, and the authors used the hourly resolution datasets of prices initially.

c. What methodologies were applied to solve the problem?

The methodology of focus for this study is deep Q-learning agents within an ensemble framework. An ensemble methodology for RL agents, which involves the use of different models trained at different iterations (epochs), with further analysis of the behavior of these ensembles through different agreement thresholds. It explores mixing Deep Q-learning RL strategies in ensemble methods considering different training iterations to predict different future markets. The final decision consists of an ensemble of decisions from different agents and is parametric to different levels of agreement thresholds and agents' configurations.

d. What were the opportunities for future work?

The opportunities for future work in this study and research include further exploration of advanced ensemble techniques to enhance the performance of the ensemble deep Q-learning approach. The authors suggest investigating the effectiveness of different aggregation methods and fusion techniques to improve the consensus forecast of the ensemble. Additionally, future research could focus on optimizing the hyper parameters and architecture of the deep Q-learning agents to further enhance their individual performance. Furthermore, the application of the ensemble deep Q-learning approach to different stock markets and the evaluation of its performance in various market conditions and various other products such as bonds and commodities markets can present opportunities for future research.

e. Critical Analysis

The study employs real-world stock market data and evaluates the performance of the proposed Multi-DQN ensemble model. The authors provide empirical evidence and evaluation metrics to demonstrate the effectiveness of the ensemble approach. This empirical validation enhances the practical relevance of the study and provides insights into the performance of the Multi-DQN model in real stock market scenarios.

The strength of this study is its focus on the integration of deep Q-learning agents and ensemble methods for stock market forecasting. The authors recognize the limitations of individual models and propose a collaborative approach to enhance prediction accuracy. By combining multiple deep Q-learning agents, the study harnesses the diversity of models and their collective knowledge to achieve improved forecasting performance.

However, one limitation of the study is the complexity of the proposed Multi-DQN ensemble model. The authors provide limited details on the implementation and training process of the individual deep Q-learning agents and the ensemble mechanism. A more in-depth explanation of these aspects would have provided a better understanding of the model and its practical implementation. Additionally, the study could have further discussed the interpretability of the Multi-DQN ensemble model. Deep Q-learning models are known for their black-box nature, and understanding the decision-making process of the ensemble and the significance of input features is essential for practical application and adoption.

Study 8

Literature

a. What were the problems studied or researched?

The problems studied or researched revolve around stock price direction forecasting using deep neural networks (DNNs) and technical analysis indicators. The authors aim to develop a methodology that combines feature selection techniques with DNNs to improve the accuracy of predicting the direction of stock prices. The authors discussed feature selection in the context of deep neural network models to predict the stock price direction. They investigated a set of 124 technical analysis indicators used as explanatory variables in the recent literature and specialized trading websites. They

applied three feature selection methods to shrink the feature set aiming to eliminate redundant information from similar indicators (Peng et al., 2021).

b. What type of data is used to understand applications?

The study uses historical stock market data, including price movements, trading volumes, and various technical analysis indicators, to understand applications of feature selection and DNNs in stock price direction forecasting. These data sources provide the necessary input for training and testing the DNN models.

The authors collected daily data between January 1st, 2008 and March 1st, 2019 from firms that composed financial market indexes from seven markets, namely: United States (S&P 100 Index), United Kingdom (FTSE 100 Index), France (CAC 40 Index), Germany (DAX-30 Index), Japan (Top 50 assets from NIKKEI 225 Index), China (Top 50 assets from SSE 180 Index) and Brazil (Bovespa Index).

c. What methodologies were applied to solve the problem?

The methodologies applied in the research involve the integration of feature selection techniques with DNNs. The authors employ various feature selection methods, such as correlation-based measures and mutual information, to identify and select the most relevant technical analysis indicators. They then utilize DNNs, such as feed forward neural networks, to capture complex patterns and relationships in the historical stock market data. The models are trained on the selected features to predict the direction of stock prices. The authors have tested the empirical performance of deep neural networks for seven markets by applying different settings of architecture and regularization techniques.

d. What were the opportunities for future work?

The opportunities for future work identified in the research include further exploration of advanced feature selection techniques to improve the model's performance

in selecting relevant indicators. The authors suggest investigating the impact of different feature selection algorithms and strategies on prediction accuracy. Additionally, future research could focus on enhancing the interpretability of the DNN models in stock price direction forecasting and exploring the combination of feature selection with other machine learning techniques, such as ensemble methods or deep learning architectures. Furthermore, the application of the proposed methodology to different financial markets and the evaluation of its performance across various time periods present opportunities for future research.

e. Critical Analysis

The study explores various feature selection algorithms, such as correlation-based measures and mutual information, and evaluates their impact on model performance. This comprehensive analysis allows for a better understanding of the strengths and limitations of different feature selection techniques in the context of stock price direction forecasting. The strength of this study is its focus on incorporating feature selection techniques into the forecasting process. The authors recognize the importance of identifying and utilizing the most informative indicators for stock price direction prediction. By applying feature selection methods, the study enhances the model's ability to extract meaningful patterns and reduce noise from the data. Moreover, the study utilizes real-world stock market data and evaluates the performance of the proposed methodology. The authors provide empirical evidence and evaluation metrics to demonstrate the effectiveness of their approach. This empirical validation enhances the credibility and practical applicability of the study's findings.

However, one limitation of the study is its focus solely on technical analysis indicators without considering other types of data or information. Stock market prices are influenced by various factors, including fundamental analysis, market news, and

macroeconomic indicators. A more comprehensive analysis that incorporates additional data sources could provide a more holistic view of stock price direction prediction. Additionally, the study does not extensively discuss the interpretability of the DNN models. Deep neural networks are known for their black-box nature, and understanding the underlying reasons for their predictions is essential for practical application and adoption. Further research could address this limitation and explore methods to enhance the interpretability of DNN models in stock price direction forecasting.

Study 9

Literature

a. What were the problems studied or researched?

The problems studied or researched revolve around forecasting stock prices using an ensemble approach that combines deep learning models with technical analysis. The authors aim to develop a methodology that leverages the strengths of both approaches to improve the accuracy of stock price predictions (Kamara et al., 2022).

b. What type of data is used to understand applications?

The study utilizes historical stock market data, including price movements, trading volumes, and various technical analysis indicators, to understand applications of the ensemble approach in stock price forecasting. These data points provide the necessary input for training and testing the deep learning models and technical analysis-based models.

c. What methodologies were applied to solve the problem?

The methodology applied in the research involves the integration of deep learning models with technical analysis indicators within a boosted hybrid ensemble framework. The authors employ various deep learning architectures, such as recurrent neural networks (RNNs) or convolutional neural networks (CNNs), to capture complex patterns

in the historical stock market data. They also incorporate technical analysis indicators, such as moving averages or relative strength index (RSI), to consider the insights from traditional technical analysis. Boosting techniques are applied to combine the predictions of individual models into an ensemble, enhancing the overall forecasting performance.

d. What were the opportunities for future work?

The opportunities for future work identified in the research include further exploration of advanced ensemble techniques to improve the performance of the boosted hybrid ensemble approach. The authors suggest investigating the impact of different ensemble algorithms, feature combinations, and model weighting strategies on prediction accuracy. Additionally, future research could focus on the integration of alternative technical analysis indicators or the inclusion of external factors, such as news sentiment or macroeconomic indicators, to enhance the robustness of the ensemble model. Furthermore, the application of the proposed methodology to different financial markets and the evaluation of its performance across various time periods present opportunities for future research.

e. Critical Analysis

The study explores various deep learning architectures, including recurrent neural networks (RNNs) and convolutional neural networks (CNNs), as well as technical analysis indicators. By combining these different approaches, the study provides a comprehensive analysis of the strengths and weaknesses of each model and their collective performance within the ensemble. The strength of this study is its focus on combining deep learning models and technical analysis to enhance stock price forecasting. The authors recognize the limitations and potential biases of individual models and propose an ensemble approach to mitigate these issues. By leveraging the

collective intelligence of multiple models, the study aims to achieve improved prediction accuracy and robustness.

Also, the study employs real-world stock market data and evaluates the performance of the proposed ensemble approach. The authors provide empirical evidence and evaluation metrics to demonstrate the effectiveness of their methodology. This empirical validation enhances the practical relevance of the study and provides insights into the performance of the boosted hybrid ensemble in real stock market scenarios. However, one limitation of the study is the lack of a detailed explanation of the ensemble mechanism and the weightings assigned to individual models. A more in-depth discussion of the ensemble approach and its optimization would have provided a better understanding of how the collective intelligence is harnessed and how individual model contributions are integrated. Additionally, the study could have further discussed the interpretability of the ensemble approach. Ensemble models can be considered as black-box systems, and understanding the decision-making process and the relative importance of input features in the ensemble would have enhanced the practical implementation and adoption.

2.9 Comparative Analysis

All the 9 literatures analyzed in this review are compared in a graphical mind map in the *Figure 2.9* and is divided into 4 categories:

- Papers
- Areas of focus
- Methodology applied and
- Opportunities for future work

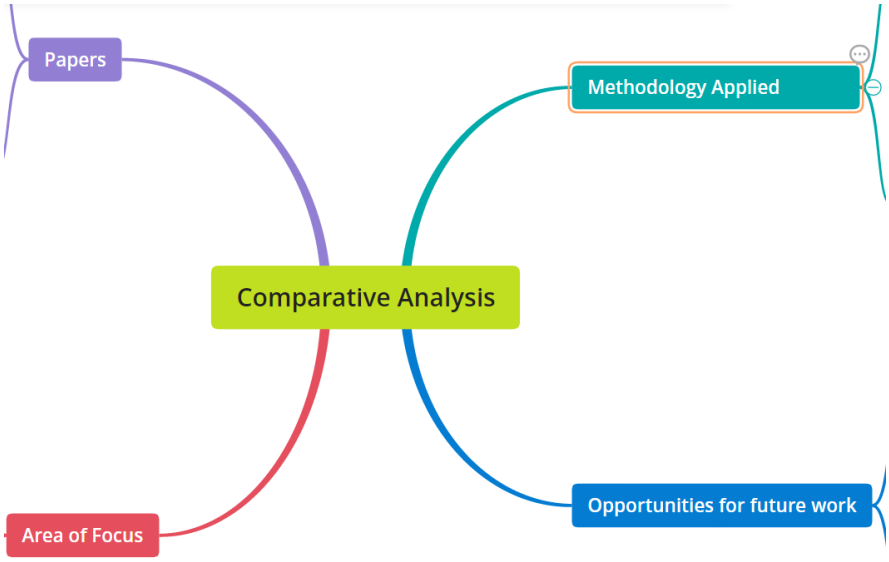

Figure 2.9 Comparative Analysis Topics

The Papers are further divided into three sections and depicted in *Figure 2.10.*

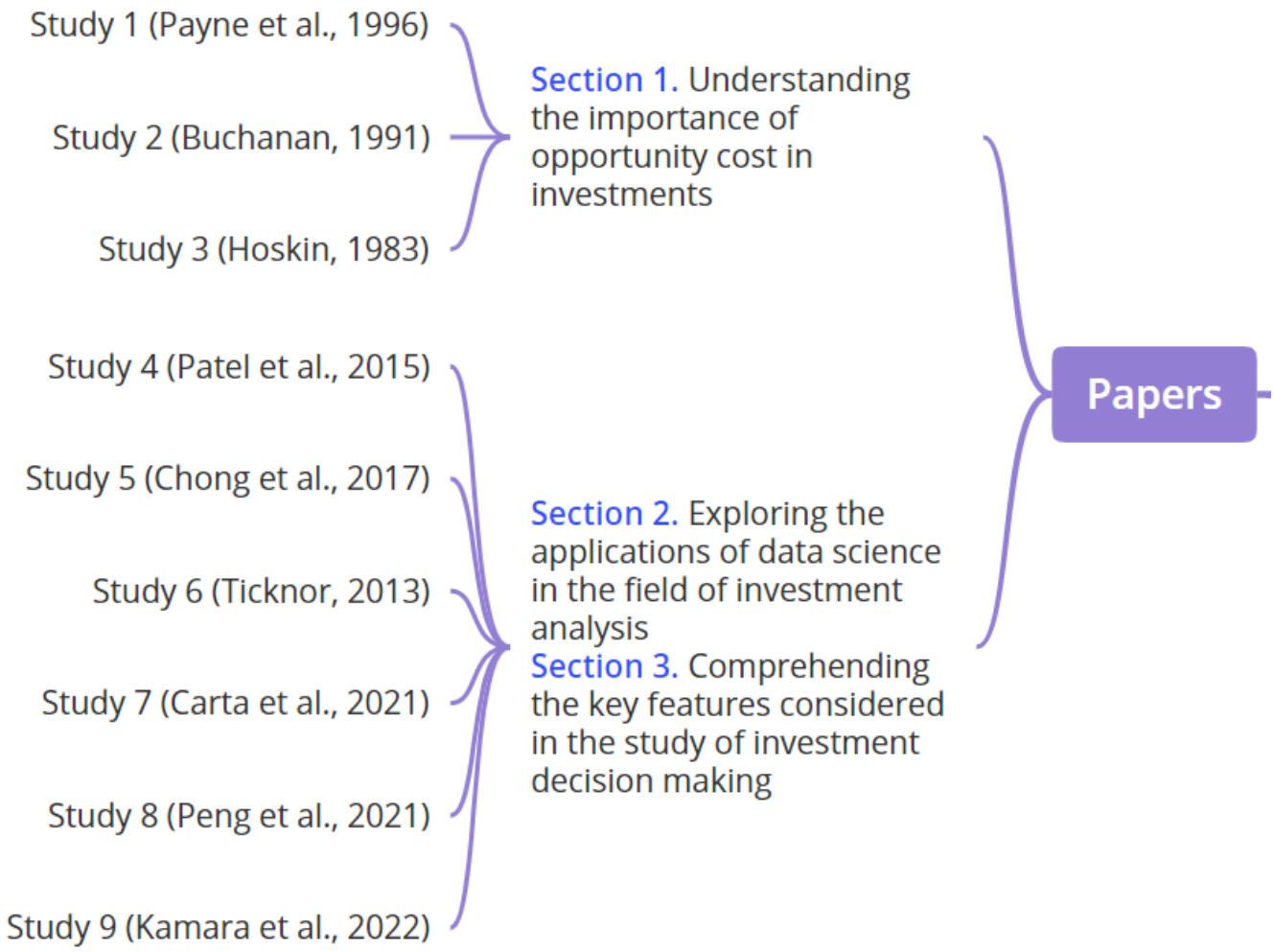

Figure 2.10 Category 1: Comparisons of Author and Year

The Areas of focus are further divided into three sections and depicted in *Figure 2.11*.

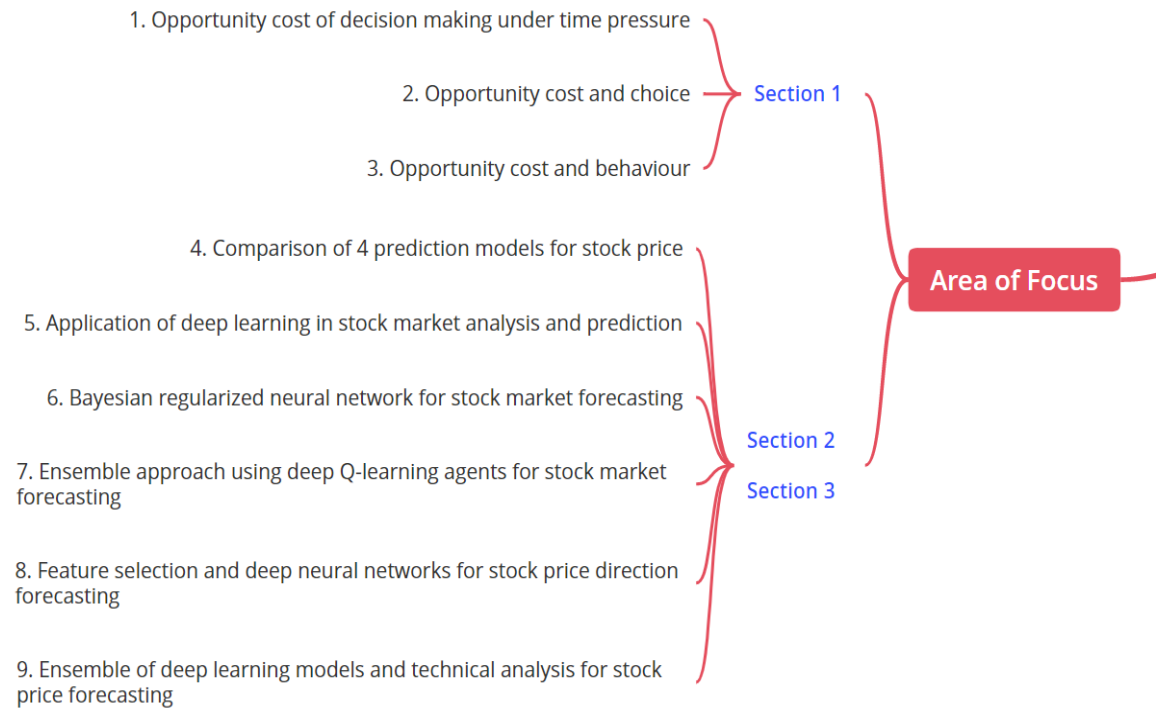

Figure 2.11 Category 2: Comparing Areas of Focus

The Areas of focus are further divided into three sections and depicted in *Figure 2.12*.

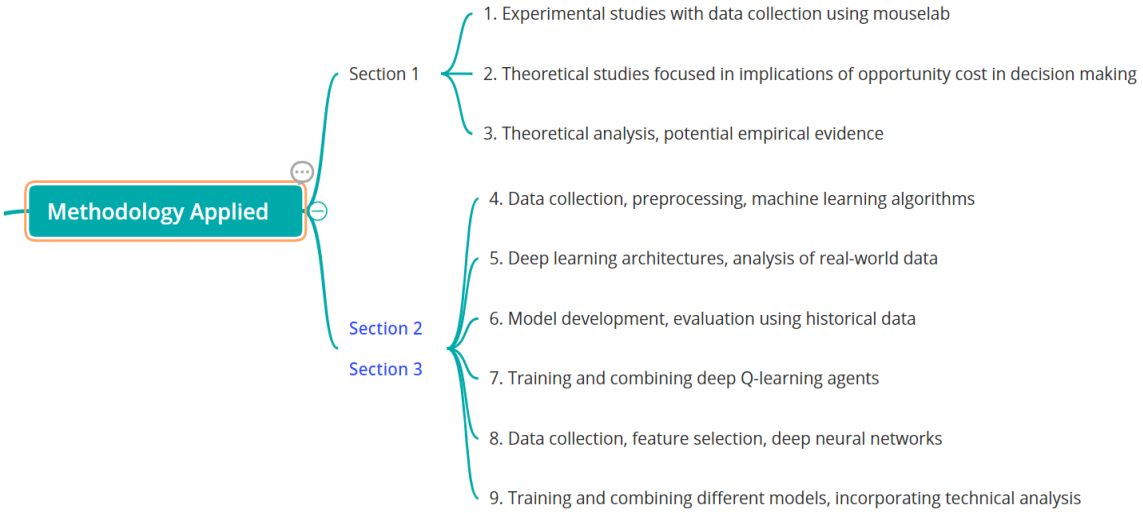

Figure 2.12 Category 3: Comparing Methodologies Applied

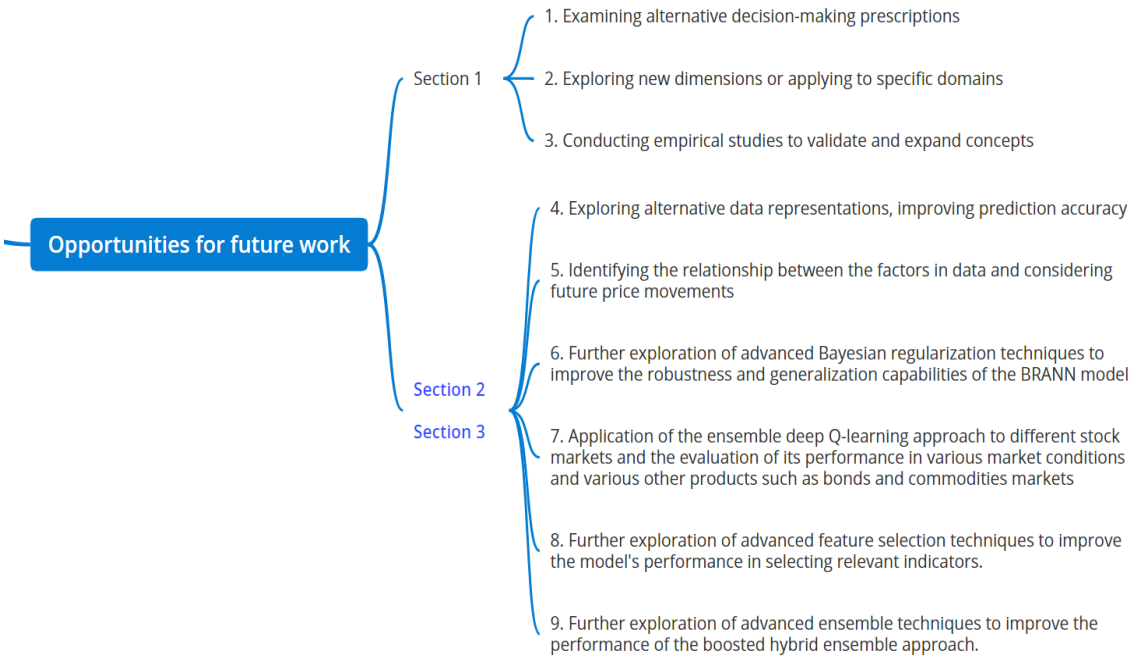

Figure 2.13 Category 4: Comparing Opportunities for Future Work

The comparisons performed from *Figures 2.9* to *2.13* gives an overall map of the studies undertaken in this literature review.

2.10 Summary

A total of 615 existing literature in the form of research papers, journals and books were collected and reviewed to understand the relevance of the existing literature by analyzing them in relation to the current study. Post analysis, it was identified that 407 research papers gathered were not relevant to the research topic under study and 208 have relevance from at least one perspective. The research papers covering the relevance for this research are majorly published by Elsevier, IEEE, Springer, JSTOR which covers journals of business, and Citeseer and Wiley online library which covers journals on finance.

Understanding the importance of opportunity cost in investments

The first part of this research is to analyze the importance of opportunity cost in investment choices by examining existing literature and to assess on how the existing literature can be studied, applied or extended further to carry forward by analyzing them against the problem statements of this research in scope.

The opportunity cost has been studied and discussed in some of the prominent journals or research papers. For this research sub section, 7 of the relevant research papers have been selected for this study and after analyzing in detail, 3 out of 7 had some relevance with respect to this specific research and were studied further.

Timing is very critical when it comes to decision making and it is even more significant when the decision is related to an investment. When both money and timing are dependent variables of an investment decisions, the decision need to be made faster and also accurate. This leads to a situation of dilemma since a decision with a time limit can also go wrong.

Decision-making dilemmas can arise because errors may result either from deciding too soon or from delaying decisions too long. Delay can result in lost opportunities or reductions in payoffs from the most accurate decision (Payne et al., 1996).

The opportunity cost by its definition is well explained in (Buchanan, 1991) as the anticipated value of 'that which might be' if choice were made differently. Note that it is not the value of 'that which might have been' without the qualifying reference to choice. In the absence of choice, it may be sometimes meaningful to discuss values of events that might have occurred but did not. It is also explained further that… First, if choice is made among separately valued options, someone must do the choosing. That is to say, a chooser is required, a person who decides. From this the second implication emerges. The

value placed on the option that is not chosen, the opportunity cost, must be that value that exists in the mind of the individual who chooses.

It is also explained in (Hoskin, 1983), that the decision impact of opportunity costs is of interest to accountants, economists, and behavioral scientists. From an economic point of view these costs are generally considered relevant, and explicitly reporting them for decision making could improve the quality of future decisions.

Exploring the applications of data science and comprehending the key features in the field of investment analysis

The second part of this research is to explore the applications of data science and comprehend the key features in the field of investment analysis and to assess on how the existing literature can be studied, applied or extended further to carry forward by analyzing them against the problem statements of this research in scope.

The problem of predicting direction of movement of stock and stock price index for Indian stock markets was addressed in the literature. The study compares four prediction models, Artificial Neural Network (ANN), support vector machine (SVM), random forest and Naive-Bayes with two approaches for input to these models. The first approach for input data involves computation of ten technical parameters using stock trading data (open, high, low & close prices) while the second approach focuses on representing these technical parameters as trend deterministic data. The major area discussed in the paper deals with predictive algorithms of machine learning that are applied to predict upward or downward movement of specific stocks and stock market indices. This is also an approach that helps in decision making and this approach also has many opportunities that can lead to future work and future enhancements in the field of machine learning on investment decision making (Patel et al., 2015).

Another study revolves around utilizing deep learning networks for stock market analysis and prediction. The authors aim to explore the methodology and data representations used in deep learning models and their application to forecast stock market behavior. The study attempts to provide a comprehensive and objective assessment of both the advantages and drawbacks of deep learning algorithms for stock market analysis and prediction. The literature explores the effects of three unsupervised feature extraction methods - principal component analysis, auto encoder, and the restricted Boltzmann machine—on the network's overall ability to predict future market behavior. The literature also offers practical insights and potentially useful directions for further investigation into how deep learning networks can be effectively used for stock market analysis and prediction. In the process of a more detailed research on the existing literature, the research paper is analyzed to learn the existing applications of machine learning algorithms at various stages of investment analysis. The paper deals with predictive analytics on the stock market returns by applying advanced and deep neural networks for the purpose of feature engineering and predictions. The advantage of applying such non-linear algorithms goes a long way in the investment decision making especially when domain knowledge is lesser or unavailable in the process of investment decision making and there is also an existence time pressure that impacts the decision-making process (Chong et al., 2017).

The problems then studied or researched revolve around stock market forecasting using Bayesian regularized artificial neural networks (BRANNs). The author investigates the application of BRANNs to address the challenges of over fitting and improve the accuracy of stock market predictions. The prediction of stock price movement is generally considered to be a challenging and important task for financial time series analysis. The complexity in predicting these trends lies in the inherent noise and volatility

in daily stock price movement. One day future closing price of individual stocks is predicted using daily market prices and financial technical indicators. The study presents a comprehensive analysis of the BRANN model's performance by comparing it to other forecasting models, such as autoregressive integrated moving average (ARIMA) and traditional ANNs. The author provides empirical evidence and evaluation metrics to demonstrate the effectiveness of the BRANN approach, highlighting its superiority in terms of prediction accuracy. The strength of this study is the integration of Bayesian regularization with artificial neural networks (ANNs) for stock market forecasting. The author recognizes the need to address over fitting issues in neural networks, and Bayesian regularization provides a valuable solution. By incorporating Bayesian inference and regularization techniques, the study enhances the generalization capabilities of the model and reduces the risk of over fitting. However, one limitation of the study is its heavy technical focus and lack of practical application and real-world case studies. While the study presents the theoretical framework and demonstrates the model's performance using historical data, further exploration of the model's application in real-time trading scenarios or investment decision-making would have enhanced the study's practical relevance. Additionally, the study does not extensively discuss the interpretability of the BRANN model. Neural networks are known for their black-box nature, and understanding the model's decision-making process and the significance of input features is crucial for its practical implementation. Further research could address this limitation and explore methods to enhance the interpretability of BRANN models (Ticknor, 2013).

Another study revolves around stock market forecasting using ensemble deep Q-learning agents. The authors in the research aims at proposing an ensemble of reinforcement learning approaches which do not use annotations (i.e. market goes up or down) to learn, but rather learn how to maximize a return function over the training stage.

In order to achieve this goal, the authors exploit a Q-learning agent trained several times with the same training data and investigate its ensemble behavior in important real-world stock markets. The study employs real-world stock market data and evaluates the performance of the proposed Multi-DQN ensemble model. The authors provide empirical evidence and evaluation metrics to demonstrate the effectiveness of the ensemble approach. This empirical validation enhances the practical relevance of the study and provides insights into the performance of the Multi-DQN model in real stock market scenarios. The strength of this study is its focus on the integration of deep Q-learning agents and ensemble methods for stock market forecasting. The authors recognize the limitations of individual models and propose a collaborative approach to enhance prediction accuracy. By combining multiple deep Q-learning agents, the study harnesses the diversity of models and their collective knowledge to achieve improved forecasting performance. However, one limitation of the study is the complexity of the proposed Multi-DQN ensemble model. The authors provide limited details on the implementation and training process of the individual deep Q-learning agents and the ensemble mechanism. A more in-depth explanation of these aspects would have provided a better understanding of the model and its practical implementation. Additionally, the study could have further discussed the interpretability of the Multi-DQN ensemble model. Deep Q-learning models are known for their black-box nature, and understanding the decision-making process of the ensemble and the significance of input features is essential for practical application and adoption (Carta et al., 2021).

The problems studied or researched in another literature revolve around stock price direction forecasting using deep neural networks (DNNs) and technical analysis indicators. The authors aim to develop a methodology that combines feature selection techniques with DNNs to improve the accuracy of predicting the direction of stock prices.

The authors discussed feature selection in the context of deep neural network models to predict the stock price direction. They investigated a set of 124 technical analysis indicators used as explanatory variables in the recent literature and specialized trading websites. They applied three feature selection methods to shrink the feature set aiming to eliminate redundant information from similar indicators. The study explores various feature selection algorithms, such as correlation-based measures and mutual information, and evaluates their impact on model performance. This comprehensive analysis allows for a better understanding of the strengths and limitations of different feature selection techniques in the context of stock price direction forecasting. The strength of this study is its focus on incorporating feature selection techniques into the forecasting process. The authors recognize the importance of identifying and utilizing the most informative indicators for stock price direction prediction. By applying feature selection methods, the study enhances the model's ability to extract meaningful patterns and reduce noise from the data. Moreover, the study utilizes real-world stock market data and evaluates the performance of the proposed methodology. The authors provide empirical evidence and evaluation metrics to demonstrate the effectiveness of their approach. This empirical validation enhances the credibility and practical applicability of the study's findings. However, one limitation of the study is its focus solely on technical analysis indicators without considering other types of data or information. Stock market prices are influenced by various factors, including fundamental analysis, market news, and macroeconomic indicators. A more comprehensive analysis that incorporates additional data sources could provide a more holistic view of stock price direction prediction. Additionally, the study does not extensively discuss the interpretability of the DNN models. Deep neural networks are known for their black-box nature, and understanding the underlying reasons for their predictions is essential for practical application and

adoption. Further research could address this limitation and explore methods to enhance the interpretability of DNN models in stock price direction forecasting (Peng et al., 2021).

The next problems studied or researched revolve around forecasting stock prices using an ensemble approach that combines deep learning models with technical analysis. The authors aim to develop a methodology that leverages the strengths of both approaches to improve the accuracy of stock price predictions. The study explores various deep learning architectures, including recurrent neural networks (RNNs) and convolutional neural networks (CNNs), as well as technical analysis indicators. By combining these different approaches, the study provides a comprehensive analysis of the strengths and weaknesses of each model and their collective performance within the ensemble. The strength of this study is its focus on combining deep learning models and technical analysis to enhance stock price forecasting. The authors recognize the limitations and potential biases of individual models and propose an ensemble approach to mitigate these issues. By leveraging the collective intelligence of multiple models, the study aims to achieve improved prediction accuracy and robustness. Also, the study employs real-world stock market data and evaluates the performance of the proposed ensemble approach. The authors provide empirical evidence and evaluation metrics to demonstrate the effectiveness of their methodology. This empirical validation enhances the practical relevance of the study and provides insights into the performance of the boosted hybrid ensemble in real stock market scenarios. However, one limitation of the study is the lack of a detailed explanation of the ensemble mechanism and the weightings assigned to individual models. A more in-depth discussion of the ensemble approach and its optimization would have provided a better understanding of how the collective intelligence is harnessed and how individual model contributions are integrated.

Additionally, the study could have further discussed the interpretability of the ensemble approach. Ensemble models can be considered as black-box systems, and understanding the decision-making process and the relative importance of input features in the ensemble would have enhanced the practical implementation and adoption (Kamara et al., 2022).

From this literature review, the analysis reveals that opportunity cost is studied with time, choice and behavior as influencing factors while the applications of machine learning as well as deep learning are mainly around stock market forecasting.

The purpose of my research work is to make use of technologies such as machine learning and deep learning, and publicly available data to develop a framework that could help an individual investor to take an informed decision without losing the investment opportunities available in the market and to understand the opportunity cost of losing such opportunities.

CHAPTER III:

METHODOLOGY

3.1 Overview of the Research Problem

The primary research method for this study is literature review and a new investment strategy modeling.

Following steps will be adopted for the research methodology:

1. Perform a detailed literature review to understand various investment strategies and data science techniques used in stock markets.
2. Identify the gaps in investment strategy modeling using methods of artificial intelligence and machine leraning.
3. Identify the AIML modeling techniques that has high potential to generate investment strategies and the areas where there are gaps.
4. Shortlist the modeling techniques that will be studied in detail to derive at a new investment strategy that can benefit investors.
5. Decide the data space for the investment strategy that will be more relevant and beneficial within the Indian stock markets.
6. Design data procurement process and decide on the data points that will be procured.
7. Perform further analysis and identify potential financial indicator based features that can be created from the procured data.
8. Design the algorithm incorporating data science techniques with financial indicators and newly derived features to device a new strategy that can generate portfolio allocations using NIFTY 50 Index as the base.
9. Calculate the historical returns of the developed portfolio strategy and compare against market benchmark to decide on the suitability of the strategy.

10. Design a front end interface that can show the comparison of all the new strategies developed in this research and analyse their fitment for Indian investors.

3.2 Research Question Formulation

This research is primarily performed with the objective to answer the following questions:

How can data science techniques and methodologies help in developing a decision-making framework that helps in identifying the opportunities to invest in the Indian stock markets?

3.2.1 Research Sub Questions

1. What are the key aspects/variables that need to be considered in understanding opportunities?
2. What are the various techniques or algorithms that can be explored to identify opportunities?
3. How can prescriptive analytics help in developing a decision-making framework in the domain of Indian stock markets?

3.3 Research Design

An appropriate research design is defined based on the research questions and objectives. The research design will be a mixed approach consisting of the study of various quantitative methodologies along with relevant quantitative aspects through a detailed literature review and then the development of a new investment strategy framework for informed decision making through opportunity analysis of Indian stock market.

3.4 Data Collection

The data collection methodology for the research in scope will be secondary data collection. The data will be procured from publicly available and authentic data source –

Yahoo finance, since the data in question for this analysis is related to stocks and their behavior. The data that would be collected through Yahoo Finance API will consist of stock fundamentals such as sales, expenses, operating profit, profit before tax, net profit, assets and liabilities etc. The data will also consist of daily stock prices for various stocks in the market index. The data in scope for this analysis are the NIFTY 50 stocks from the Indian Stock markets.

3.5 Data Analysis

Data Analysis will be performed on the publicly available data using statistical techniques and data science methods. Various machine learning and data science techniques will be applied to identify patterns and factors associated with lost investment opportunities. Quantitative data where applicable will be analyzed to gain insights into the decision-making process and challenges faced by Indian households.

3.6 Framework Development

A framework will be developed for educated decision-making by integrating the findings from the data analysis, literature review, and qualitative analysis. The framework will be addressing the identified challenges of expertise and information gaps, leveraging data science and machine learning techniques. The framework will be designed as a practical, user-friendly model, and aligned with the specific needs of Indian households.

The detailed design and development of the framework will be discussed in Chapter 4.

3.7 Framework Validation

The effectiveness of the developed framework will be evaluated through quantitative and qualitative assessments. Appropriate evaluation metrics will be used to measure the impact of the framework on educated decision-making and investment

behavior. The feedback from the end users will also be used to validate the framework's practicality and usefulness.

CHAPTER IV:

SLRPO INVESTMENT FRAMEWORK – DESIGN AND DEVELOPMENT

4.1 Introduction

In this research, I am introducing a new investment strategy named Strategic Reinforcement Learning Portfolio Optimizer and abbreviated it to SRLPO interchangeably used as SLRPO for the ease of usage. The purpose of this strategy is to provide a suitable portfolio allocation of NIFTY 50 stocks for long-term investments by developing multiple portfolio allocation strategies and identifying the best by analyzing the opportunity cost of investing in one strategy compared to the other.

The following theoretical constructs will be studied in detail and operationalized to develop SRLPO investment strategy.

1. Technical Indicators
2. Fundamental Indicators
3. Reinforcement Learning
4. Upper Confidence Bound1
5. Policy Network
6. Policy Search
7. Policy Gradient Methods

A new feature named Strength score will be introduced based on the technical and fundamental indicators and it will be used as input for SLRPO strategy development. The architecture of SLRPO Strategy is designed as depicted in Figure 4.1.

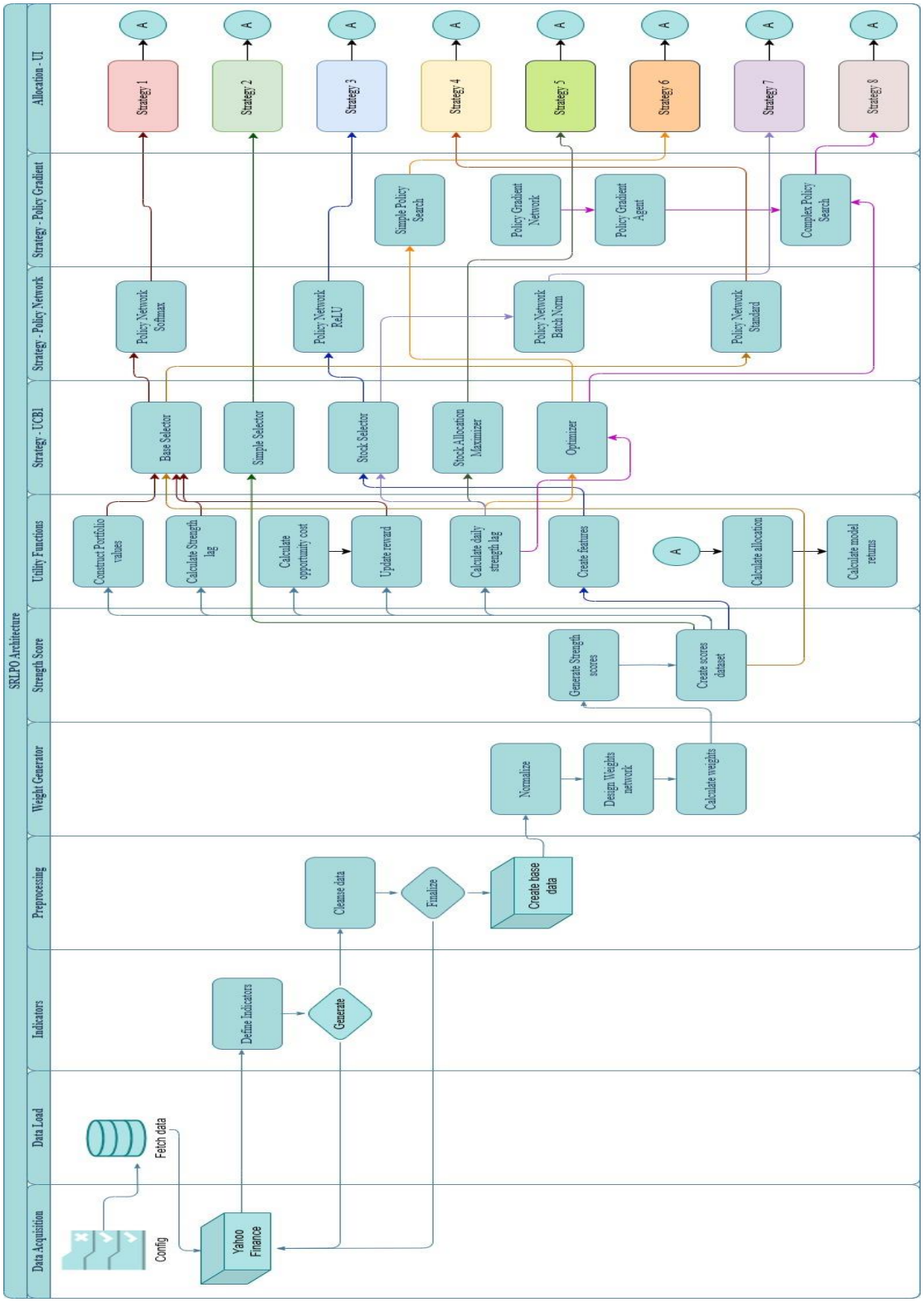

Figure 4.1 SLRPO Architecture

The SLRPO architecture in Figure 4.1 describes various modules designed to develop the architecture for SLRPO and the functionalities performed in each module. The modules are listed as follows:

1. Data Acquisition
2. Data Load
3. Indicators
4. Preprocessing
5. Weight Generator
6. Strength Score
7. Utility Functions
8. Strategy – UCB1
9. Strategy – Policy Network
10. Strategy – Policy Gradient
11. Allocation – UI

4.2 Data Acquisition Module

Data acquisition module for SLRPO focuses on systematically collecting relevant financial data from a reliable data source. This module is crucial for conducting accurate and meaningful financial analysis to create better strategies. In this module, the data requirements will be specified as part of the SLRPO code and then retrieved from the data source and structured in a way it can be used for financial analysis.

This module focuses on the following key implementations:

Data Source Identification:

Yahoo finance is leveraged as the most reliable data source for building SLRPO. Yahoo finance is a well-known and credible provider of financial data. It offers

comprehensive information on stocks. Choosing yahoo finance as the data source ensures reliability and accuracy of the finance data.

Target data specification:

Data acquisition implementation requires ticker symbol and date range as inputs to fetch the data required for this analysis.

Ticker:

Ticker symbol uniquely identifies the stock of interest. Specifying the ticker symbol helps in precise targeting of data acquisition by focusing on specific set of stocks relevant to the research question.

Date Range:

Data range is specified in the data acquisition module to specifically decide on the temporal scope of the data to be collected. This ensures that the data is extracted for the specific period required for this analysis and will be used for studying the long-term trend of the stocks.

Historical data retrieval:

The purpose of the data acquisition module is to fetch the historical data such as daily prices and trading volumes within the specified data range. Historical data is crucial for conducting time-series analysis, understanding past performance and identifying patterns over time.

Collection of stock information:

In addition to the historical prices, this module also retrieves detailed information about the stock metadata such as sector, market cap and other required fundamental attributes. This data helps in understanding broader characteristics and environment of the stock and to facilitate more comprehensive analysis.

Automation and Efficiency:

The use of yfinance library in Python to automate the data retrieval process makes it very efficient to retrieve the data for this analysis. Automation of the data retrieval process improves the efficiency of the data acquisition module, minimizes the risk of manual errors that can occur while downloading large datasets and reduces the manual efforts required in the process.

Structuring data for analysis:

The data acquisition module is designed to retrieve both historical stock prices and fundamental stock information in a structured format. This structured format makes the data immediately usable and also makes it easy to proceed with further transformations on the data so that it becomes suitable for developing SLRPO strategy.

Logic Explanation:

The code for this module starts by defining libraries required for data acquisition process.

```
import configparser
import yfinance as yf
```

The *configparser* is a standard library in Python used for importing configuration files. The configurations needed for the SLRPO strategy will be defined in the config file and imported using this library.

Yfinance is a Python library used for accessing financial data from Yahoo finance. It has methods defined for downloading historical prices and stock information.

The next step in this module is to define a function named read_config to read the configuration settings from a config file where the list of tickers needed to be used for this analysis are defined.

```
def read_config(config_file):
    config = configparser.ConfigParser()
```

```
        config.read(config_file)
        if 'Settings' in config:
            settings = config['Settings']
            tickers = [item.strip() for item in settings.get('tickers').split('\n') if
item.strip()]
            return tickers
        else:
            raise ValueError("No 'Settings' section found in the config file")
```

The final responsibility of this module is to define the function fetch_data to fetch the data from yahoo finance using the tickers defined from config file, as well as the start date and end date needed for this research.

```
    def fetch_data(ticker, start_date, end_date):
        stock = yf.Ticker(ticker)
        hist = stock.history(start=start_date, end=end_date)
        return stock.info, hist
```

The purpose of this module is to systematically define and collect the data needed for the SLRPO strategy development in a well-defined structure and from a reliable source suitable for conducting robust financial research.

4.3 Data Load Module

Data load module is the continuation of data acquisition module, and it focuses on loading the data from yahoo finance. The data load is part of the data processing pipeline, and it focuses on invoking the data acquisition module to gather the data needed for SLRPO implementation. The key concepts designed in the data load module include the following:

Initiating Data Acquisition:

Data acquisition is initiated in the data load module by calling the fetch_data function. The tickers for the stocks, start date and end date are provided as parameters for this function.

Centralized Data Management:

The data retrieval process is managed centrally in the data load module. A streamlined approach to loading data from yahoo finance is initiated in the data load module and further processing of data begins from this module.

Interface with Analytical Tools:

The module also ensures that the data is readily available for further processing needed for SLRPO strategy development. The purpose of this module is to ensure that the data is loaded into SLRPO eco system so that it can enable detailed development for the rest of the strategy.

4.4 Indicators Module

The third module of SLRPO strategy is the *Indicators* module. For an investment strategy to be designed successfully, understanding the statistical nature of the stock prices and identifying hidden insights from it is very important. To do that, this module defines a class named *technical_indicators* and covers the definitions of the following indicators.

4.4.1 Technical Indicators

The list of technical indicators considered to design the SLRPO strategy are as follows:

1. Average Directional Index (ADX)
2. Accumulation/Distribution Line (AD)
3. Average True Range (ATR)
4. Bollinger Bands

5. Chaikin Money Flow (CMF)
6. Donchian Channel
7. Ease of Movement (EMV)
8. Keltner Channel
9. Money Flow Index (MFI)
10. Moving Average Convergence Divergence (MACD)
11. Moving Averages (SMA and EMA)
12. On-Balance Volume (OBV)
13. Parabolic SAR
14. Periodic High/Low
15. Rate of Change (ROC)
16. Relative Strength (RS)
17. Relative Strength Index (RSI)
18. Stochastic Oscillator
19. Supertrend
20. Weighted Volume
21. Volume at Price
22. Volume Price Trend (VPT)
23. Volume Weighted Average Price (VWAP)

1. Average Directional Index (ADX)

The Average Directional Index (ADX) is the first feature chosen for SLRPO strategy and it is used to quantify the technical strength of a trend in the capital markets or the stock markets. Developed by J. Welles Wilder (Wilder, 1978), the ADX is part of the Directional Movement System, which also includes the Plus Directional Indicator (+DI) and the Minus Directional Indicator (-DI). The ADX ranges from 0 to 100, where

higher values indicate a stronger trend, and lower values indicate a weaker trend or a range-bound market. It does not indicate the direction of the trend, only its strength. ADX can be calculated by following these steps:

1. Compute the True Range (TR), which measures market volatility.
2. Determine the Directional Movement (DM), which differentiates between upward and downward movements.
3. Smooth the TR and DM values using a specified window period (commonly 14 periods).
4. Calculate the Directional Indicators (+DI and -DI).
5. Compute the Directional Index (DX) and then average it to obtain the ADX.

For SLRPO strategy, the code for ADX is defined as follows:

```
def calculate_adx(self, data, window=14):
    high, low, close = data['High'], data['Low'], data['Close']
    tr1 = high - low
    tr2 = abs(high - close.shift())
    tr3 = abs(low - close.shift())
    tr = pd.DataFrame({'tr1': tr1, 'tr2': tr2, 'tr3': tr3}).max(axis=1)
    plus_dm = high.diff()
    minus_dm = low.diff()
    plus_dm[plus_dm < 0] = 0
    minus_dm[minus_dm > 0] = 0
    minus_dm = minus_dm.abs()
    smooth_tr = tr.rolling(window).sum()
    smooth_plus_dm = plus_dm.rolling(window).sum()
    smooth_minus_dm = minus_dm.rolling(window).sum()
```

```
    plus_di = 100 * (smooth_plus_dm / smooth_tr)
    minus_di = 100 * (smooth_minus_dm / smooth_tr)
    dx = (abs(plus_di - minus_di) / (plus_di + minus_di)) * 100
    adx = dx.rolling(window).mean()
    return adx
```

In the preceding code,

1. High, Low, and Close prices are extracted from the data.
2. True Range (TR) is calculated using three different methods to capture the most significant price movement.
3. Directional Movements (DM) are computed to distinguish between upward and downward price changes.
4. The TR and DM values are smoothed using a rolling window to reduce noise.
5. Directional Indicators (+DI and -DI) are calculated as percentages of the smoothed values.
6. The Directional Index (DX) is determined and then averaged to produce the ADX, which reflects the trend strength.

By understanding the Average Directional Index (ADX) and its application in assessing trend strength, SLRPO strategy will have a valuable feature for evaluating market conditions. However, trend strength is just one aspect of market analysis. To gain a more comprehensive understanding, it's important to also consider indicators that analyze the flow of money into and out of a security. One such indicator is the Accumulation/Distribution Line (AD), which provides insights into buying and selling pressure by considering both price and volume. The next feature considered for SLRPO strategy is AD Line.

2. Accumulation/Distribution Line (AD)

The next feature chosen for SLRPO strategy is Accumulation/Distribution Line (AD Line). This is a technical analysis indicator that helps investors understand the flow of money into and out of a security. This indicator is developed by Marc Chaikin (Chaikin, n.d.) and it combines both price and volume data to assess whether a stock is being bought or sold over a specific period. Using this indicator helps in understanding the trends between AD Line and the stock price.

AD Line can be calculated by calculating the following parameters:

Money Flow Multiplier

Money Flow Volume

Accumulation / Distribution Line

Money Flow Multiplier

MFM is calculated using the daily stock price as follows:

$$\boldsymbol{Numerator = (Close - Low) - (High - Close)}$$

$$\boldsymbol{Denominator = High - Low}$$

$$\boldsymbol{MFM = \frac{Numerator}{Denominator}}$$

When the close price is near the high, the MFM will be closer to +1.

When the close price is near the low, the MFM will be closer to -1.

The Money Flow Multiplier (MFM) is calculated to determine the proportion of the day's trading volume that is attributable to buying or selling pressure.

Money Flow Volume

MFV is calculated as follows:

$$\boldsymbol{MFV = MFM \times Volume}$$

The Money Flow Volume (MFV) adjusts the trading volume based on the MFM to reflect the actual flow of money.

Accumulation / Distribution Line

Accumulation / Distribution Line is calculated as follows:

$$\boldsymbol{AD\ Line} = \sum \boldsymbol{MFV}$$

Running sum is calculated for the AD Line by adding each day's MFV with the previous day's AD Line.

For SLRPO strategy, the code for AD Line is defined as follows:

```
def calculate_accumulation_distribution(self, data):
    high, low, close, volume = data['High'], data['Low'], data['Close'],
data['Volume']
    mfm = ((close - low) - (high - close)) / (high - low)
    mfm.replace([float('inf'), -float('inf')], 0, inplace=True)
    mfv = mfm * volume
    ad = mfv.cumsum()
    return ad
```

In the preceding code,

1. MFM is calculated by calculating the numerator and denominator as explained above and dividing them.
2. MFV is calculated by multiplying MFM with Volume.
3. AD Line is calculated by cumulatively summing the MFV.

Considering AD Line as one of the features helps SLRPO strategy to gather insights on the flow of money and to analyze buying and selling pressure for stocks in the market. Let us further consider the next feature Average True Range (ATR) which is useful in understanding market volatility.

3. Average True Range (ATR)

Average True Range (ATR) is another feature that will be included in SLRPO strategy to understand market volatility using stock data. This indicator was developed by J. Welles Wilder (Wilder, 1978).

ATR can be used to understand the degree of price movement over a specific period.

The ATR is calculated using the following parameters:

True Range

Average True Range

True Range

Three values for True Range are calculated as follows:

True Range 1 is the difference between current High and current Low of the stock price.

$$\boldsymbol{TR1 = High - Low}$$

True Range 2 is the difference between the absolute value of current High and the Close price at T-1.

$$\boldsymbol{TR2 = ABS(High - Close_{t-1})}$$

True Range 3 is the difference between the absolute value of current Low and the Close price at T-1.

$$\boldsymbol{TR3 = ABS(Low - Close_{t-1})}$$

The final True Range of each period is calculated as the maximum of TR1, TR2 and TR3.

$$\boldsymbol{TR = MAX(TR1, TR2, TR3)}$$

Average True Range

The next step of ATR indicator is the calculation of Average True Range.

ATR is an average of True Ranges over a period N.

$$\boldsymbol{ATR = \frac{\sum_{i=1}^{N} TR_i}{N}}$$

For SLRPO strategy, the code for ATR is defined as follows:

```
def calculate_atr(self, data, window=14):
    high, low, close = data['High'], data['Low'], data['Close']
    tr1 = high - low
    tr2 = abs(high - close.shift())
    tr3 = abs(low - close.shift())
    true_range = pd.DataFrame({'tr1': tr1, 'tr2': tr2, 'tr3': tr3}).max(axis=1)
    atr = true_range.rolling(window=window).mean()
    return atr
```

In the preceding code,

1. True Range 1 is calculated as the difference between the current high and current low.
2. True Range 2 is calculated as the absolute difference between the current high and the previous close.
3. True Range 3 is calculated as the absolute difference between the current low and the previous close.
4. The final True Range is calculated as the max of the above TR1, 2 and 3.
5. The Average True Range (ATR) is calculated by taking the rolling mean of the True Range over a specified number of periods which in this strategy is defined as 14.

Considering Average True Range as one of the features helps SLRPO strategy to gather insights on the market volatility and to manage the risks and make informed decisions on stocks in the market. Let us now consider the next feature Bollinger Bands which is useful in understanding market dynamics further.

4. Bollinger Bands

The next feature for SLRPO strategy will be Bollinger Bands. This indicator was developed by John A. Bollinger. Bollinger Bands is another market indicator that can

help in understanding market volatility by identifying overbought or oversold conditions in the market (Bollinger, 2002).

Bollinger Bands are calculated using the following parameters:

1. Middle Band
2. Upper Band
3. Lower Band

Middle Band

The middle band is a simple moving average of the price calculated over a specific period. The default window of SLRPO strategy will be set to 20, which is also the number chosen usually for Bollinger Bands.

$$\boldsymbol{MB} = \mathbf{SMA}(\mathbf{n})$$

MB – Middle Band

SMA – Simple Moving Average

n – Selected Window

Upper Band

The upper band is calculated as the sum of the middle band and a multiple of standard deviation. The default value for the number of standard deviations is for SLRPO strategy will be set to 2, which is also the number chosen usually for Bollinger Bands.

$$\boldsymbol{UB} = \boldsymbol{MB} + (\boldsymbol{\sigma} * \boldsymbol{ns})$$

UB – Upper Band

MB – Middle Band

σ – Standard Deviation

ns – Number of Standard Deviations

Lower Band

The lower band is calculated as the difference between the middle band and a multiple of standard deviation. The default value for the number of standard deviations is for SLRPO strategy will be set to 2, which is also the number chosen usually for Bollinger Bands.

$$\boldsymbol{LB = MB - (\sigma * ns)}$$

LB – Lower Band

MB – Middle Band

σ – Standard Deviation

ns – Number of Standard Deviations

For SLRPO strategy, the code for Bollinger Bands is defined as follows:

```
def calculate_bollinger_bands(self, data, window=20, num_of_std=2):
    sma = data.rolling(window=window).mean()
    std = data.rolling(window=window).std()
    upper_band = sma + (std * num_of_std)
    lower_band = sma - (std * num_of_std)
    bollinger_bands = pd.DataFrame({'Upper Band': upper_band, 'Middle Band': sma, 'Lower Band': lower_band})
    return bollinger_bands
```

In the preceding code,

1. The Simple Moving Average (sma) of price is calculated for a period of 20. This is the middle band.
2. The Standard Deviation (std) is calculated for a period of 20.
3. The upper band is calculated as sum of sma + 2*std.
4. The lower band is calculated as the difference of sma – 2*std.
5. Bollinger Bands are represented as upper band, middle band and lower band.

Considering Bollinger Bands as one of the features helps SLRPO strategy to gather more insights on the market volatility and to identify overbought and oversold conditions on stocks in the market. Let us now consider the next feature Chaikin Money Flow (CMF) which is useful in understanding market dynamics further.

5. Chaikin Money Flow (CMF)

Chaikin Money Flow (CMF) is the next feature that will be included in SLRPO strategy to understand buying and selling pressure in the markets for a security using stock data. This indicator was developed by Marc Chaikin. CMF can be used to understand the market sentiment and price movements of a security (Chaikin, n.d.). The CMF is calculated using the following parameters:

1. Money Flow Multiplier (MFM)
2. Money Flow Volume (MFV)
3. Chaikin Money Flow (CMF)

Money Flow Multiplier (MFM):

The MFM is calculated as follows using the daily stock prices of a security.

$$\boldsymbol{MFM = \frac{(Close - Low) - (High - Close)}{High - Low}}$$

Money Flow Volume (MFV)

The MFV is calculated by multiplying MFM with Volume.

$$\boldsymbol{MFV = \text{MFM} \times \text{Volume}}$$

Chaikin Money Flow (CMF)

The CMF is calculated as the ratio between the sum of MFV over a window period *n* and the Volume over the same window.

$$CMF = \frac{\sum_{i=1}^{n} MFV_i}{\sum_{i=1}^{n} Volume_i}$$

For SLRPO strategy, the code for CMF is defined as follows:

```
def calculate_chaikin_money_flow(self, data, window=20):
    high, low, close, volume = data['High'], data['Low'], data['Close'],
data['Volume']
    mfm = ((close - low) - (high - close)) / (high - low)
    mfm.replace([float('inf'), -float('inf')], 0, inplace=True)
    mfv = mfm * volume
    cmf = mfv.rolling(window=window).sum() /
volume.rolling(window=window).sum()
    return cmf
```

In the preceding code,

1. Money Flow Multiplier is calculated as defined in the MFM formula in this section.
2. The infinity values in the results of MFM are then replaced by 0 to avoid calculation errors in the next steps.
3. Money Flow Volume is calculated as the product of MFM with Volume.
4. CMF is then calculated as the ratio between the rolling sum of MFV with the rolling sum of Volume over the same window.

Considering Chaikin Money Flow as one of the features helps SLRPO strategy to gather insights into buying and selling pressure through volume and price analysis on stocks in the market. Let us now consider the next feature Donchian Channel which is useful in understanding market dynamics further.

6. Donchian Channel

Donchian channel is the next feature that will be included in SLRPO strategy to understand the trends in price changes by identifying the highest highs and the lowest

lows in the price of a stock. This indicator was developed by Richard Donchian (Investopedia, 2023).

The Donchian channel is calculated using the following parameters:

1. Upper Band
2. Lower Band
3. Middle Band

Upper Band

The upper band is calculated as the highest high over a selected window.

Lower Band

The lower band is calculated as the lowest low over a selected window.

Middle Band

The middle band is calculated as the average of the upper and lower bands.

For SLRPO strategy, the code for Donchian channel is defined as follows:

```
def calculate_donchian_channel(self, data, window=20):
    upper_band = data['High'].rolling(window=window).max()
    lower_band = data['Low'].rolling(window=window).min()
    middle_band = (upper_band + lower_band) / 2
    donchian_channel = pd.DataFrame({'Upper Band': upper_band, 'Middle Band': middle_band, 'Lower Band': lower_band})
    return donchian_channel
```

In the preceding code,

1. Upper band is calculated as the maximum of the high stock price in the rolling window.
2. Lower band is calculated as the minimum of the low stock price in the rolling window.

3. Middle band is calculated as the mean of upper band and the lower band.
4. Donchian channel is then returned as a combined data frame of all the above values.

Considering Donchian Channel as one of the features helps SLRPO strategy in identifying breakouts and trend reversals and for understanding price range and volatility. Let us now consider the next feature Ease of Movement (EMV) which is useful in understanding market dynamics further.

7. Ease of Movement (EMV)

Ease of Movement (EMV) is another feature that will be included in SLRPO strategy to understand the relationship between the rate of price change and volume using stock data. This indicator was developed by Richard W. Arms, Jr. EMV can be used to understand the momentum behind price movements and identify potential buying or selling opportunities (Arms, 1996).

The EMV is calculated using the following parameters:

- Midpoint Moved or Distance Moved
- Box Ratio
- Ease of Movement value
- Smoothing

Midpoint Moved or Distance Moved:

The midpoint moved or distance moved is the parameter to calculate the midpoint of the high and low prices for the current and previous periods:

$$\boldsymbol{Distance\ Moved} = \frac{(High_t + Low_t)}{2} - \frac{(High_{t-1} + Low_{t-1})}{2}$$

This measures the change in the midpoint of the price range from one period to the next.

Box Ratio:

The next parameter is the box ratio which normalizes the midpoint move by the trading volume and the price range:

$$\boldsymbol{Box\ Ratio} = \frac{Volume / Volume\ Divisor}{High_t - Low_t}$$

Ease of Movement value:

The EMV value for the period is calculated as:

$$\boldsymbol{EMV} = \frac{Distance\ Moved}{Box\ Ratio}$$

This value indicates how easily the price is moving. A positive EMV indicates that prices are moving up easily, while a negative EMV indicates that prices are moving down easily.

Smoothing:

The final step in EMV calculation is to smooth the EMV value using moving average so that the value becomes more useful for analysis. This is an optional parameter that can be calculated.

$$\boldsymbol{Smooth\ EMV} = Moving\ Average(EMV, n)$$

n - the number of periods for the moving average.

For SLRPO strategy, the code for EMV is defined as follows:

```
def calculate_ease_of_movement(self, data, window=14,
volume_divisor=100000000):
    high, low, volume = data['High'], data['Low'], data['Volume']
    distance_moved = ((high + low) / 2) - ((high + low) / 2).shift(1)
    box_ratio = (volume / volume_divisor) / (high - low)
    emv = distance_moved / box_ratio
```

```
        emv.replace([float('inf'), -float('inf')], 0, inplace=True)  # Handle division by
zero
        emv_smoothed = emv.rolling(window=window).mean()
        return emv_smoothed
```

In the preceding code,

1. The Distance Moved is calculated as the difference between the average of current high and low and the average of previous high and low.
2. The Box Ratio is calculated as the normalized ratio of volume and the difference between current high and low.
3. The EMV is calculated as the ratio of Distance Moved and the Box Ratio.
4. Smoothing is then performed on the EMV.

Considering Ease of Movement (EMV) as one of the features helps SLRPO strategy in understanding the momentum of price movements by integrating volume. Let us now consider the next feature Keltner Channel which is useful in understanding market trends and volatility further.

8. Keltner Channel

The Keltner Channel is another feature that will be included in SLRPO strategy to identify potential trends and trading opportunities by combining aspects of volatility and moving averages. This indicator was developed by Chester W. Keltner in the 1960s (Keltner, 1960).

The Keltner Channel has been refined over time and is calculated using the Exponential Moving Average (EMA) and the Average True Range (ATR).

The Keltner Channel is calculated using the following parameters:

- Middle Line
- High Low

- High Close
- Low Close
- Ranges
- True Range
- Average True Range
- Upper Band
- Lower Band

Middle Line

The middle line is calculated as follows:

$$\boldsymbol{MiddleLine_t} = Close_t \times (1 - \alpha) + MiddleLine_{t-1} \times \alpha$$

High Low

The High Low is calculated as follows:

$$\boldsymbol{HighLow_t} = High_t - Low_t$$

High Close

The High Close is calculated as follows:

$$\boldsymbol{HighClose_t} = |High_t - Close_{t-1}|$$

Low Close

The Low Close is calculated as follows:

$$\boldsymbol{LowClose_t} = |Low_t - Close_{t-1}|$$

Ranges

The ranges are calculated as follows:

$$\boldsymbol{Ranges_t} = \begin{matrix} HighLow_t \\ HighClose_t \\ LowClose_t \end{matrix}$$

True Range

The true range is calculated as follows:

$$\boldsymbol{TrueRange_t} = \max\left(Ranges_t\right)$$

Average True Range

The ATR is calculated as follows:

$$\boldsymbol{ATR_t} = \frac{1}{n}\sum_{i=t-n+1}^{t} TrueRange_i$$

n – the ATR window

Upper Band

The upper band is calculated as follows:

$$\boldsymbol{UpperBand_t} = MiddleLine_t + (k \times ATR_t)$$

k – multiplier

Lower Band

The lower band is calculated as follows:

$$\boldsymbol{LowerBand_t} = MiddleLine_t - (k \times ATR_t)$$

For SLRPO strategy, the code for Keltner Channel is defined as follows:

```
def calculate_keltner_channel(self, data, ema_period=20, atr_period=10,
multiplier=2):
    middle_line = data['Close'].ewm(span=ema_period, adjust=False).mean()
    high_low = data['High'] - data['Low']
    high_close = (data['High'] - data['Close'].shift()).abs()
    low_close = (data['Low'] - data['Close'].shift()).abs()
    ranges = pd.concat([high_low, high_close, low_close], axis=1)
    true_range = ranges.max(axis=1)
    atr = true_range.rolling(window=atr_period).mean()
    upper_band = middle_line + (multiplier * atr)
    lower_band = middle_line - (multiplier * atr)
```

```
        keltner_channel = pd.DataFrame({'Upper Band': upper_band, 'Middle Line':
middle_line, 'Lower Band': lower_band})
        return keltner_channel
```

In the preceding code,

1. The parameters required for Keltner channel are calculated as explained in the equations of this section.
2. The Exponential Moving Average period is set to 20 and the Average True Range period is set to 10.
3. The multiplier for upper band and lower band is set to 2.

Considering Keltner Channel as one of the features helps SLRPO strategy in understanding the trend and volatility of prices. Let us now consider the next feature Money Flow Index (MFI) which is useful in understanding market dynamics further.

9. Money Flow Index (MFI)

Money Flow Index (MFI) is another feature that will be included in SLRPO strategy, and it uses both price and volume data to identify overbought and oversold conditions in a security. MFI was developed by Gene Quong and Avrum Soudack (Murphy, 1999).

The MFI is calculated using the following parameters:

1. Typical Price
2. Raw Money Flow
3. Positive Flow
4. Negative Flow
5. Sum Positive Money Flow
6. Sum Negative Money Flow
7. Money Flow Ratio

8. Money Flow Index

Typical Price

The Typical Price is calculated as follows:

$$\boldsymbol{TypicalPrice_t} = \frac{High_t + Low_t + Close_t}{3}$$

Raw Money Flow

Raw Money Flow is calculated as follows:

$$\boldsymbol{RawMoneyFlow_t} = TypicalPrice_t \times Volume_t$$

Positive Flow

Positive Flow is calculated as follows:

$$\boldsymbol{PositiveFlow_t} = \begin{cases} RawMoneyFlow_t, & TypicalPrice_t \geq TypicalPrice_{t-1} \\ 0, & Otherwise \end{cases}$$

Negative Flow

Negative Flow is calculated as follows:

$$\boldsymbol{NegativeFlow_t} = \begin{cases} RawMoneyFlow_t, & TypicalPrice_t \leq TypicalPrice_{t-1} \\ 0, & Otherwise \end{cases}$$

Sum Positive Money Flow

The sum positive money flow is calculated as follows:

$$\boldsymbol{SumPositiveMF_t} = \sum_{i=t-period+1}^{t} PositiveFlow_i$$

period – Selected period

Sum Negative Money Flow

The sum negative money flow is calculated as follows:

$$\boldsymbol{SumNegativeMF_t} = \sum_{i=t-period+1}^{t} NegativeFlow_i$$

period – Selected period

Money Flow Ratio

The money flow ratio is calculated as follows:

$$\boldsymbol{MFR_t} = \frac{SumPositiveMF_t}{SumNegativeMF_t}$$

Money Flow Index

The money flow index is calculated as follows:

$$\boldsymbol{MFI_t} = 100 - \left(\frac{100}{1 + MFR_t}\right)$$

For SLRPO strategy, the code for Money Flow Index is defined as follows:

```
def calculate_money_flow_index(self, data, period=14):
    typical_price = (data['High'] + data['Low'] + data['Close']) / 3
    raw_money_flow = typical_price * data['Volume']
    positive_flow = raw_money_flow.copy()
    negative_flow = raw_money_flow.copy()

    positive_flow[typical_price <= typical_price.shift()] = 0
```

```
    negative_flow[typical_price >= typical_price.shift()] = 0
    positive_mf_sum = positive_flow.rolling(window=period).sum()
    negative_mf_sum = negative_flow.rolling(window=period).sum()
    mfr = positive_mf_sum / negative_mf_sum
    mfi = 100 - (100 / (1 + mfr))
    return mfi
```

In the preceding code,

1. The parameters required for MFI are calculated as explained in the equations of this section.
2. The rolling window period for the calculations are set to 14.

Considering Money Flow Index as one of the features helps SLRPO strategy to gather insights on the market momentum and liquidity. Let us now consider the next feature Moving Average Convergence Divergence (MACD) which is useful in understanding market momentum furthermore.

10. Moving Average Convergence Divergence (MACD)

Moving Average Convergence Divergence (MACD) is another feature that will be included in SLRPO strategy, and it is used to identify changes in the strength, direction, momentum, and duration of a trend in the price of a stock. MACD was developed by Gerald Appel in the late 1970s (Appel, 2005).

The MACD is calculated using the following parameters:

1. EMA Fast
2. EMA Slow
3. MACD Line
4. Signal Line

EMA Fast

Exponential Moving Average (EMA) Fast is calculated as follows:

$$\boldsymbol{EMAFast_t} = \left(\frac{Close_t \times \left(1 - \propto_{fast}\right) + EMAFast_{t-1} \times \propto_{fast}}{1}\right)$$

where $\propto_{fast} = \frac{2}{fast+1}$

EMA Slow

Exponential Moving Average (EMA) Slow is calculated as follows:

$$\boldsymbol{EMASlow_t} = \left(\frac{Close_t \times (1 - \propto_{slow}) + EMASlow_{t-1} \times \propto_{slow}}{1}\right)$$

where $\propto_{slow} = \frac{2}{slow+1}$

MACD Line

Moving Average Convergence Divergence (MACD) is calculated as follows:

$$\boldsymbol{MACD_t} = EMAFast_t - EMASlow_t$$

Signal Line

Signal Line is calculated as follows:

$$\boldsymbol{SignalLine_t} = \left(\frac{MACD_t \times \left(1 - \alpha_{signal}\right) + SignalLine_{t-} \times \alpha_{signal}}{1}\right)$$

where $\alpha_{signal} = \frac{2}{signal+1}$

For SLRPO strategy, the code for MACD is defined as follows:

```
def calculate_macd(self, data, slow=26, fast=12, signal=9):
    ema_fast = data['Close'].ewm(span=fast, adjust=False).mean()
    ema_slow = data['Close'].ewm(span=slow, adjust=False).mean()
    macd = ema_fast - ema_slow
    signal_line = macd.ewm(span=signal, adjust=False).mean()
    macd_df = pd.DataFrame({'MACD': macd, 'Signal Line': signal_line})
    return macd_df
```

In the preceding code,

1. The parameters required for MACD are calculated as explained in the equations of this section.
2. The value for fast is set to 12, slow is set to 26 and signal is set to 9.
3. MACD_DF is returned as a data frame with combined data of MACD Line and Signal Line.

Considering Moving Average Convergence Divergence (MACD) as one of the features helps SLRPO strategy in identifying trends and momentum shifts. Let us now consider the next feature Moving Averages themselves which is useful in smoothing the price data and understanding underlying trends.

11. Moving Averages (SMA and EMA)

Moving Averages (SMA and EMA) is another feature that will be included in SLRPO strategy, it helps to smooth out price data over a specified period, making it easier to identify trends and potential reversal points. Two of the most used types of moving averages are the Simple Moving Average (SMA) and the Exponential Moving Average (EMA). Moving averages have been discussed and refined throughout various technical analysis performed on stock markets data. One of the sources of reference for moving averages is 'Technical Analysis of the Financial Markets' (Murphy, 1999).
The Moving Averages are calculated using the following parameters:

1. Simple Moving Average
2. Exponential Moving Average

Simple Moving Average

The Simple Moving Average is calculated as follows:

$$\boldsymbol{SMA_t} = \frac{1}{n}\sum_{i=t-n+1}^{t} data_i$$

n - the window period

Exponential Moving Average

The Exponential Moving Average is calculated as follows:

$$\boldsymbol{EMA_t} = \alpha + data_t + (1 - \alpha) \times EMA_{t-1}$$

$$\alpha = \frac{2}{n+1}$$

α – smoothing factor

n – period for the window

For SLRPO strategy, the code for Moving averages is defined as follows:

```
def calculate_moving_averages(self, data, period):
    sma = data.rolling(window=period).mean()
    ema = data.ewm(span=period, adjust=False).mean()
    moving_averages = pd.DataFrame({'SMA': sma, 'EMA': ema})
    return moving_averages
```

In the preceding code,

1. The parameters required for SMA and EMA are calculated as explained in the equations of this section.
2. The rolling window period for the calculations are defined as custom parameters.
3. Moving Averages are returned as a data frame with combined data of SMA and EMA.

Considering Moving Averages as one of the features helps SLRPO strategy to smoothen the price data and gather insights on the price trends. Let us now consider the next feature On-Balance Volume (OBV) which is useful in understanding market trends furthermore.

12. On-Balance Volume (OBV)

On-Balance Volume (OBV) is another feature that will be included in SLRPO strategy, and it uses volume to understand the price movement. According to OBV, significant changes in volume can signal potential price movements. OBV was developed by Joe Granville in the 1960s (Granville, 1963).

The OBV is calculated using the following parameters:

1. Initial OBV
2. OBV

Initial OBV

The OBV is initialized as follows:

$$\boldsymbol{OBV_0} = 0$$

OBV

OBV is calculated as follows:

$$\boldsymbol{OBV_t} = \begin{cases} OBV_{t-1} + Volume_t & if\ Close_t > Close_{t-1} \\ OBV_{t-1} - Volume_t & if\ Close_t < Close_{t-1} \\ OBV_{t-1} & if\ Close_t = Close_{t-1} \end{cases}$$

For SLRPO strategy, the code for On-Balance Volume (OBV) is defined as follows:

```
def calculate_on_balance_volume(self, data):
    close = data['Close']
    volume = data['Volume']
    obv = pd.Series(index=close.index)
    obv.iloc[0] = volume.iloc[0] if close.iloc[0] > close.iloc[0] else 0
    for i in range(1, len(close)):
        if close[i] > close[i - 1]:
            obv[i] = obv[i - 1] + volume[i]
```

```
        elif close[i] < close[i - 1]:
            obv[i] = obv[i - 1] - volume[i]
        else:
            obv[i] = obv[i - 1]
    return obv
```

In the preceding code,

1. The OBV is initialized to 0 and then calculated further.
2. The parameters required for OBV are calculated as explained in the equations of this section.

Considering On-Balance Volume (OBV) as one of the features helps SLRPO strategy to gather insights on the price movements and trends. Let us now consider the next feature Parabolic SAR which is useful in understanding market trends furthermore.

13. Parabolic SAR

Parabolic SAR (Stop and Reverse) is another feature that will be included in SLRPO strategy, and it is used to identify potential reversal points in the price movement of an asset and to optimize trading strategies. Parabolic SAR was developed by J. Welles Wilder Jr. (Wilder, 1978).

The Parabolic SAR is represented as a series of dots placed above or below the price bars on a chart:

- When the dots are below the price, it indicates an uptrend.
- When the dots are above the price, it indicates a downtrend.

The Parabolic SAR is calculated using the following parameters:

1. Extreme Point
2. Acceleration Factor
3. Initialize SAR

4. Initialize Trend
5. SAR

Extreme Point

The extreme point is calculated as the highest high in an uptrend or the lowest low in a downtrend.

$$\boldsymbol{EP_0} = \begin{cases} High_0 \; If \; uptrend \; is \; True \\ Low_0 \; If \; uptrend \; is \; False \end{cases}$$

Acceleration Factor

The acceleration factor is a value that is initialized at 0.02 and increases by 0.02 each time a new EP is reached, up to a maximum of 0.20.

$$\boldsymbol{AF_0} = 0.02$$

Initialize SAR

The initial SAR is calculated as follows:

$$\boldsymbol{SAR}[\mathbf{0}] = \begin{cases} High[0] & if \; High[0] < Low[0] \\ Low[0] & Otherwise \end{cases}$$

Initialize Trend

The initial Trend is calculated as follows:

$$\boldsymbol{Uptrend} = \begin{cases} True \; if \; High[0] < Low[0] \\ False \; Otherwise \end{cases}$$

SAR

The SAR is calculated as follows:

$$\boldsymbol{SAR_i} = SAR_{i-1} + AF \times (EP - SAR_{i-1})$$

For SLRPO strategy, the code for Parabolic SAR is defined as follows:

```
        def calculate_parabolic_sar(self, high, low, af_start=0.02, af_increment=0.02,
af_max=0.2):
            sar = high if high[0] < low[0] else low
            sar.iloc[0] = high[0] if high[0] < low[0] else low[0]
            uptrend = high[0] < low[0]
            ep = high[0] if uptrend else low[0]
            af = af_start
            for i in range(1, len(high)):
                if uptrend:
                    sar[i] = sar[i - 1] + af * (ep - sar[i - 1])
                    if high[i] > ep:
                        ep = high[i]
                        af = min(af + af_increment, af_max)
                    if low[i] < sar[i]:
                        uptrend = False
                        sar[i] = ep
                        ep = low[i]
                        af = af_start
                else:
                    sar[i] = sar[i - 1] + af * (ep - sar[i - 1])
                    if low[i] < ep:
                        ep = low[i]
                        af = min(af + af_increment, af_max)
                    if high[i] > sar[i]:
                        uptrend = True
```

```
                sar[i] = ep
                ep = high[i]
                af = af_start
        return sar
```

In the preceding code,

1. The parameters required for Parabolic SAR are calculated as explained in the equations of this section.
2. The AF Start is set to 0.02, incremented by 0.02 and AF Max is set to 0.2.
3. The SAR is updated in the loop that iterates through each data point to update the SAR value based on the current trend direction.

Considering Parabolic SAR as one of the features helps SLRPO strategy in identifying potential trend reversals and managing trailing stop losses. Let us now consider the next feature Periodic High/Low which is useful in understanding how it can be used to spot key support and resistance levels and enhance market analysis further.

14. Periodic High/Low

Periodic High/Low is another feature that will be included in SLRPO strategy, and it is used to identify the highest high and the lowest low within a specified period. One of the sources of reference for Periodic High/Low is ‘Technical Analysis of the Financial Markets’ (Murphy, 1999).

The Periodic High/Low is calculated using the following parameters:

1. Periodic High
2. Periodic Low

Periodic High

The Periodic High is the highest price reached within a specified period. It is calculated as follows:

$$PeriodicHigh = \max\left(High_{1,}\ High_2, \ldots High_n\right)$$

Periodic Low

The Periodic Low is the lowest price reached within a specified period. It is calculated as follows:

$$PeriodicLow = \max\left(Low_{1,}\ Low, \ldots Low_n\right)$$

For SLRPO strategy, the code for Periodic High/Low is defined as follows:

```
def calculate_periodic_high_low(self, data, period):
    periodic_high = data.rolling(window=period).max()
    periodic_low = data.rolling(window=period).min()
    high_low_df = pd.DataFrame({'Periodic High': periodic_high, 'Periodic
Low': periodic_low})
    return high_low_df
```

In the preceding code,

1. The parameters required for Periodic High/Low are calculated as explained in the equations of this section.
2. Periodic High/Low are returned as a data frame with combined data of Periodic High and Periodic Low.

Considering Periodic High/Low as one of the features helps SLRPO strategy to gather insights on the price movements. Let us now consider the next feature Rate of Change (RoC) which quantifies the speed at which prices are changing.

15. Rate of Change (ROC)

Rate of Change (ROC) is another feature that will be included in SLRPO strategy, and it is a momentum-based technical analysis indicator that measures the percentage change in price between the current stock price and a stock price in the past. It helps in

understanding the speed at which a security's price is changing, providing insights into the momentum and strength of a trend. One of the sources of reference for ROC is 'Technical Analysis of the Financial Markets' (Murphy, 1999).

The ROC is calculated using the following parameters:

1. Difference in price
2. Rate of Change (ROC)

Difference in price:

The difference in price is calculated as follows:

$$\Delta \boldsymbol{Price} = Price_t - Price_{t-n}$$

Price - is the price of the security at the most recent period.

n - is the number of periods chosen for the calculation.

Rate of Change (ROC):

The rate of change is calculated as follows:

$$\boldsymbol{ROC_t} = \left(\frac{\Delta Price}{Price_{t-n}}\right) \times 100$$

n - is the number of periods chosen for the calculation.

For SLRPO strategy, the code for Rate of Change (ROC) is defined as follows:

```
def calculate_rate_of_change(self, data, period):
    roc = ((data - data.shift(period)) / data.shift(period)) * 100
    return roc
```

In the preceding code,

1. The parameters required for ROC are calculated as explained in the equations of this section.
2. The period for the calculations is provided as input while calling the function.

Considering Rate of Change (ROC) as one of the features helps SLRPO strategy to gather insights on the market momentum and strength of a trend. Let us now consider the next feature Relative Strength (RS) which is useful to identify strong performing assets.

16. Relative Strength

Relative Strength is another feature that will be included in SLRPO strategy, and it is used to compare the performance of one asset against another or against a benchmark index. RS is used to identify how well an asset is performing relative to its peers or the market as a whole. RS is a technical indicator extensively used in technical analysis and was widely discussed by Robert W. Colby in his comprehensive guide on technical analysis (Colby, 2003).

The RS is calculated using the following parameters:

1. Delta
2. Gain
3. Loss
4. Relative Strength

Delta

The Delta value is calculated as follows:

$$\Delta \boldsymbol{Price_t} = \ Price_t - \ Price_{t-1}$$

Gain

The Gain is calculated as follows:

$$\boldsymbol{Gain_t} = \frac{1}{n} \sum_{i=t-n+1}^{t} \max\left(\Delta Price_i, 0\right)$$

Loss

The Loss is calculated as follows:

$$\boldsymbol{Loss_t} = \frac{1}{n} \sum_{i=t-n+1}^{t} \max\left(-\Delta Price_i, 0\right)$$

Relative Strength

The relative strength is calculated as follows:

$$\boldsymbol{RS_t} = \frac{Gain_t}{Loss_t}$$

For SLRPO strategy, the code for Relative Strength is defined as follows:

```
def calculate_relative_strength(self, data, period=14):
    delta = data.diff(1)
    gain = (delta.where(delta > 0, 0)).rolling(window=period).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=period).mean()
    rs = gain / loss
    return rs
```

In the preceding code,

1. The parameters required for RSI are calculated as explained in the equations of this section.
2. The rolling window period for the calculations are set to 14.

Considering Relative Strength (RS) as one of the features helps SLRPO strategy in comparing the performance of assets against benchmarks or peers. Let us now consider the next feature Relative Strength Index (RSI) which is an extension of Relative Strength and provides additional insights on the market momentum.

17. Relative Strength Index (RSI)

Relative Strength Index (RSI) is another feature that will be included in SLRPO strategy and is used in technical analysis to measure the speed and change of price movements. The value of RSI falls between 0 to 100 and it provides information on whether a stock is overbought or oversold. RSI was developed by J. Welles Wilder in 1978 (Wilder, 1978).

The RSI is calculated using the following parameters. Some of the parameters in RSI overlaps with the parameters in Relative Strength:

1. Delta
2. Gain
3. Loss
4. Relative Strength
5. Relative Strength Index

Delta

The Delta value is calculated as follows:

$$\Delta \boldsymbol{Price_t} = \ Price_t - \ Price_{t-1}$$

Gain

The Gain is calculated as follows:

$$\boldsymbol{Gain_t} = \frac{1}{n} \sum_{i=t-n+1}^{t} \max\left(\Delta Price_i, 0\right)$$

Loss

The Loss is calculated as follows:

$$\boldsymbol{Loss_t} = \frac{1}{n} \sum_{i=t-n+1}^{t} \max\left(-\Delta Price_i, 0\right)$$

Relative Strength

The relative strength is calculated as follows:

$$\boldsymbol{RS_t} = \frac{Gain_t}{Loss_t}$$

Relative Strength Index

The relative strength index is calculated as follows:

$$\boldsymbol{RSI_t} = 100 - \left(\frac{100}{1 + RS_t}\right)$$

For SLRPO strategy, the code for Relative Strength Index is defined as follows:

```
def calculate_rsi(self, data, period=14):
    delta = data.diff(1)
    gain = (delta.where(delta > 0, 0)).rolling(window=period).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=period).mean()
    rs = gain / loss
    rsi = 100 - (100 / (1 + rs))
    return rsi
```

In the preceding code,

1. The parameters required for RSI are calculated as explained in the equations of this section.
2. The rolling window period for the calculations are set to 14.

Considering Relative Strength Index as one of the features helps SLRPO strategy to identify overbought and oversold conditions, as well as potential reversals and trend strength. Let us now consider the next feature Stochastic Oscillator which will be used to understand how it can complement our analysis and enhance SLRPO Strategy.

18. Stochastic Oscillator

Stochastic Oscillator is another feature that will be included in SLRPO strategy, and it compares a security's closing price to its price range over a specified period, providing insights into the momentum and potential reversal points of the asset. Stochastic Oscillator was developed by George Lane in the late 1950s (Lane, 1984).

The Stochastic Oscillator is calculated using the following parameters:

1. Lowest Low
2. Highest High
3. %K Line
4. % D Line

Lowest Low

The lowest low is calculated as follows:

$$\boldsymbol{LowMin_t} = \min\left(Low_{t-k+1}, Low_{t-k+2}, \dots, Low_t\right)$$

k – Rolling window period

Highest High

The highest high is calculated as follows:

$$\boldsymbol{HighMax_t} = \max\left(High_{t-k+1}, High_{t-k+2}, \dots, High_t\right)$$

k – Rolling window period

%K Line

The %K Line is calculated as follows:

$$\%\boldsymbol{K_t} = \left(\frac{Close_t - LowMin_t}{HighMax_t - LowMin_t}\right) \times 100$$

%D Line

The %D Line (Moving Average of %K) is calculated as follows:

$$\%\boldsymbol{D_t} = \frac{1}{d}\sum_{i=t-d+1}^{t} \%K_i$$

d – window period for D Line

For SLRPO strategy, the code for Stochastic Oscillator is defined as follows:

```
def calculate_stochastic_oscillator(self, data, k_period=14, d_period=3):
```

```
    low_min = data['Low'].rolling(window=k_period).min()
    high_max = data['High'].rolling(window=k_period).max()
    k_line = ((data['Close'] - low_min) / (high_max - low_min)) * 100
    d_line = k_line.rolling(window=d_period).mean()
    stochastic_df = pd.DataFrame({'%K': k_line, '%D': d_line})
    return stochastic_df
```

In the preceding code,

1. The parameters required for Stochastic Oscillator are calculated as explained in the equations of this section.
2. The rolling window period k is set to 14 and d is set to 3 for the calculations.

Considering Stochastic Oscillator as one of the features helps SLRPO strategy in identifying overbought and oversold conditions and potential reversal points. Let us now consider the next feature Supertrend which uses price and volatility data to generate buy and sell signals.

19. Supertrend

Supertrend is another feature that will be included in SLRPO strategy, and it helps to identify the prevailing trend of a security and provides clear buy and sell signals. It combines price action and volatility to determine the direction of the trend. Supertrend was widely discussed by Olivier Seban, a French trader and author (Seban, 2008).

The Supertrend is calculated using the following parameters:

1. High Low Average
2. Average True Range
3. Basic Upper Band
4. Basic Lower Band
5. Final Upper Band

6. Final Lower Band
7. Supertrend

High Low Average

The high low average is calculated as follows:

$$\boldsymbol{HL2_t} = \frac{High_t + Low_t}{2}$$

Average True Range

The average true range is calculated as follows:

$$\boldsymbol{ATR_t} = \frac{1}{n} \sum_{i=t-n+1}^{t} \max\left(High_i - Low_i, |High_i - Close_{i-1}|, |Low_i - Close_{i-1}|\right)$$

n – Rolling window period

Basic Upper Band

The basic upper band is calculated as follows:

$$\boldsymbol{BasicUpperBand_t} = HL2_t + (k \times ATR_t)$$

k - multiplier

Basic Lower Band

The basic lower band is calculated as follows:

$$\boldsymbol{BasicLowerBand_t} = HL2_t - (k \times ATR_t)$$

k - multiplier

Final Upper Band

The final upper band is calculated as follows:

$$\boldsymbol{FinalUpperBand_t} = \begin{cases} min(BasicUpperBand_t, FinalUpperBand_{t-1}), & if\ Close_{t-1} \leq FinalUpperBand_{t-1} \\ BasicUpperBand_t, & Otherwise \end{cases}$$

Final Lower Band

The final lower band is calculated as follows:

$$\boldsymbol{FinalLowerBand_t} = \begin{cases} max(BasicLowerBand_t, FinalLowerBand_{t-1}), & if\ Close_{t-1} \geq FinalLowerBand_{t-1} \\ BasicLowerBand_t, & Otherwise \end{cases}$$

Supertrend

The Supertrend is calculated as follows:

$$\boldsymbol{Supertrend_t} = \begin{cases} FinalUpperBand_t, & if\ Close_{t-1} \leq FinalUpperBand_{t-1} \\ FinalLowerBand_t, & Otherwise \end{cases}$$

For SLRPO strategy, the code for Supertrend is defined as follows:

```
    def calculate_supertrend(self, data, period=14, multiplier=3):
        hl2 = (data['High'] + data['Low']) / 2
        atr = hl2.rolling(window=period).apply(lambda x: max(x.max() - x.min(),
abs(x.max() - x[-1]), abs(x.min() - x[-1])).mean(), raw=True)
        basic_upperband = hl2 + (multiplier * atr)
        basic_lowerband = hl2 - (multiplier * atr)
        final_upperband = basic_upperband.copy()
        final_lowerband = basic_lowerband.copy()
        for i in range(1, len(final_upperband)):
            if data['Close'][i-1] <= final_upperband[i-1]:
                final_upperband[i] = min(basic_upperband[i], final_upperband[i-1])
            if data['Close'][i-1] >= final_lowerband[i-1]:
                final_lowerband[i] = max(basic_lowerband[i], final_lowerband[i-1])
        supertrend = pd.Series(data.index)
        for i in range(1, len(supertrend)):
            if data['Close'][i-1] <= final_upperband[i-1]:
                supertrend[i] = final_upperband[i]
            else:
                supertrend[i] = final_lowerband[i]
```

return supertrend

In the preceding code,

1. The parameters required for Supertrend are calculated as explained in the equations of this section.
2. The rolling window period for the calculations are set to 14 and the multiplier is set to 3.

Considering Supertrend as one of the features helps SLRPO strategy in identifying trend directions and generating buy and sell signals. Let us now consider the next feature Weighted Volume which is a Volume based indicator and helps in understanding market activity.

20. Weighted Volume

Weighted Volume is another feature that will be included in SLRPO strategy, and it combines price and volume data to provide insights into the strength and significance of price movements. Weighted Volume adjusts the volume based on the price change as opposed to any other simple volume indicators that considers only the trading volume. One of the sources of reference for Weighted Volume is 'Technical Analysis of the Financial Markets' (Murphy, 1999).

The Weighted Volume is calculated using the following parameters:

1. Price Range
2. Weighted Volume

Price Range

The price range is calculated as follows:

$$\boldsymbol{PriceRange_t} = High_t - Low_t$$

Weighted Volume

The weighted volume is calculated as follows:

$$\boldsymbol{WeightedVolume_t} = Volume_t - PriceRange_t$$

For SLRPO strategy, the code for Weighted Volume is defined as follows:

```
def calculate_weighted_volume(self, data):
    price_range = data['High'] - data['Low']
    weighted_volume = data['Volume'] * price_range
    return weighted_volume
```

In the preceding code,

1. The parameters required for Weighted Volume are calculated as explained in the equations of this section.

Considering Weighted Volume as one of the features helps SLRPO strategy in combining price and volume data to assess the strength and significance of price movements. Let us now consider the next feature Volume at Price which provides detailed insights into the distribution of trading volume across different price levels.

21. Volume at Price

Volume at Price is another feature that will be included in SLRPO strategy, and it helps identify key price levels where significant trading activity has occurred, which can act as support or resistance levels. One of the sources of reference for Volume at Price is 'Technical Analysis of the Financial Markets' (Murphy, 1999).

The Volume at Price is calculated using the following parameters:

1. Volume at Each Close Price

Volume at Each Close Price:

The volume at each close price is calculated as:

$$\boldsymbol{VolumeAtPrice(C)} = \sum_{t \in T(C)} Volume_t$$

This groups the data by each unique closing price and sums the volumes for each group, providing the total trading volume for each closing price.

For SLRPO strategy, the code for Volume at Price is defined as follows:

```
def calculate_volume_at_price(self, data):
    volume_at_price = data.groupby('Close')['Volume'].sum()
    return volume_at_price.reset_index(drop = True)
```

In the preceding code,

1. The parameters required for Volume at Price are calculated as explained in the equations of this section.

Considering Volume at Price as one of the features helps SLRPO strategy to gather insights on the trading volume across different price levels. Let us now consider the next feature Volume Price Trend (VPT) which is useful in understanding market momentum furthermore.

22. Volume Price Trend (VPT)

Volume Price Trend (VPT) is another feature that will be included in SLRPO strategy, and it combines price and volume to provide insights into the strength and direction of a trend. The VPT indicator is like the On-Balance Volume (OBV) but includes price changes in its calculation, making it more sensitive to both volume and price fluctuations. One of the sources of reference for Volume Price Trend (VPT) is 'Technical Analysis of the Financial Markets' (Murphy, 1999).

The VPT is calculated using the following parameters:

1. Change in Close Price
2. Percentage Change in Close Price
3. VPT Contribution
4. Cumulative sum of VPT Contributions

Change in Close Price

The change in close price is calculated as follows:

$$\Delta \boldsymbol{Close_t} = Close_t - Close_{t-1}$$

Percentage Change in Close Price

The percentage change in close price is calculated as follows:

$$\boldsymbol{PercentageChange_t} = \frac{\Delta Close_t}{Close_{t-1}}$$

VPT Contribution

The VPT contribution is calculated as follows:

$$\boldsymbol{VPTContribution_t} = Volume_t \times PercentageChange_t$$

Cumulative sum of VPT Contributions

The cumulative sum of VPT contributions is calculated as follows:

$$\boldsymbol{VPT_t} = \sum_{i=1}^{t} VPTContribution_i$$

For SLRPO strategy, the code for VPT is defined as follows:

```
def calculate_volume_price_trend(self, data):
    close = data['Close']
    volume = data['Volume']
    vpt = (volume * ((close - close.shift(1)) / close.shift(1))).cumsum()
    return vpt
```

In the preceding code,

1. The parameters required for VPT are calculated as explained in the equations of this section.

Considering Volume Price Trend (VPT) as one of the features helps SLRPO strategy in understanding the strength of price movements by considering the cumulative volume. Let us now consider the next feature Volume Weighted Average Price (VWAP)

which helps to identify the average price at which a security has traded throughout the day, adjusted for volume.

23. Volume Weighted Average Price (VWAP)

Volume Weighted Average Price (VWAP) is another feature that will be included in SLRPO strategy, and it uses both price and volume data to calculate the average price a security has traded at throughout the day. One of the sources of reference for Volume Weighted Average Price (VWAP) is "Algorithmic Trading and DMA: An Introduction to Direct Access Trading Strategies" by Barry Johnson (Johnson, 2010).

The VWAP is calculated using the following parameters:

1. Cumulative Sum of the Product of Close Price and Volume
2. Cumulative Sum of Volume
3. VWAP

Cumulative Sum of the Product of Close Price and Volume

The cumulative sum of the product of close price and volume is calculated as follows:

$$\boldsymbol{CS(C \times V_t)} = \sum_{i=1}^{t} (Close_i \times Volume_i)$$

Cumulative Sum of Volume

The cumulative sum of volume is calculated as follows:

$$\boldsymbol{CS(V_t)} = \sum_{i=1}^{t} Volume_i$$

VWAP

The VWAP is calculated as follows:

$$\boldsymbol{VWAP_t} = \frac{CS(C \times V_t)}{CS(V_t)}$$

For SLRPO strategy, the code for VWAP is defined as follows:

```
def calculate_vwap(self, data):
    vwap = (data['Close'] * data['Volume']).cumsum() /
data['Volume'].cumsum()
    return vwap
```

In the preceding code,

1. The parameters required for VWAP are calculated as explained in the equations of this section.

These technical indicators are chosen because they collectively provide a well-rounded analysis of market conditions. These indicators are relevant for SLRPO strategy to identify trends, measure momentum, assess volatility, and understand volume dynamics. By covering different aspects of technical analysis, these indicators enable SLRPO in making informed investment decisions and prescribe portfolios.

4.4.2 Fundamental Indicators

The list of fundamental indicators considered to design the SLRPO strategy are as follows (Investopedia, n.d.):

1. Earnings Per Share (EPS)
2. Price to Earnings Ratio (P/E)
3. Projected Earnings Growth (PEG)
4. Free Cash Flow (FCF)
5. Price to Book Ratio (P/B)
6. Return on Equity (ROE)
7. Dividend Payout Ratio (DPR)
8. Price to Sales Ratio (P/S)
9. Dividend Yield Ratio

10. Debt-to-Equity Ratio (D/E)

11. Book Value

12. Revenue

13. Revenue Growth

The definitions and equations for these fundamental indicators are explained as follows:

Earnings Per Share (EPS)

Earnings Per Share (EPS) is one of the fundamental features that will be included in SLRPO strategy. EPS is the measure of the amount of net income earned per share of a company's common stock.

$$\boldsymbol{EPS} = \frac{Net\ Income - Preferred\ Dividends}{Average\ Otstanding\ Shares}$$

Price to Earnings Ratio (P/E)

Price to Earnings Ratio (P/E) is another fundamental feature that will be included in SLRPO strategy. P/E ratio is the measure of the price that the investors are willing to pay per earnings.

$$\boldsymbol{P/E\ Ratio} = \frac{Market\ Price\ per\ Share}{Earnings\ Per\ Share\ (EPS)}$$

Projected Earnings Growth (PEG)

Projected Earnings Growth (PEG) is another fundamental feature that will be included in SLRPO strategy. PEG is the measure of the ratio between P/E Ratio and Annual EPS Growth Rate.

$$\boldsymbol{PEG\ Ratio} = \frac{P/E\ Ratio}{Annual\ EPS\ Growth\ Rate}$$

Free Cash Flow (FCF)

Free Cash Flow (FCF) is another fundamental feature that will be included in SLRPO strategy. FCF is the measure of the difference between Operating Cash Flow and Capital Expenditures for a company.

$$\boldsymbol{FCF} = \text{Operating Cash Flow} - \text{Capital Expenditures}$$

Price to Book Ratio (P/B)

Price to Book Ratio (P/B) is another fundamental feature that will be included in SLRPO strategy. P/B is the measure of the ratio of Market Price per share to the Book Value per share.

$$\boldsymbol{P/B\ Ratio} = \frac{Market\ Price\ per\ Share}{Book\ Value\ per\ Share}$$

Return on Equity (ROE)

Return on Equity (ROE) is another fundamental feature that will be included in SLRPO strategy. ROE is the measure of the ratio of Net Income to the Shareholders' equity.

$$\boldsymbol{ROE} = \frac{Net\ Income}{Shareholders'Equity}$$

Dividend Payout Ratio (DPR)

Dividend Payout Ratio (DPR) is another fundamental feature that will be included in SLRPO strategy. DPR is the measure of the ratio of Dividends per Share to the Earnings Per Share (EPS).

$$\boldsymbol{DPR} = \frac{Dividends\ per\ Share}{Earnings\ Per\ Share\ (EPS)}$$

Price to Sales Ratio (P/S)

Price to Sales Ratio (P/S) is another fundamental feature that will be included in SLRPO strategy. P/S is the measure of the ratio of Market Price per Share to the Sales per Share.

$$\boldsymbol{P/SRatio} = \frac{Market\ Price\ per\ Share}{Sales\ per\ Share}$$

Dividend Yield Ratio

Dividend Yield Ratio is another fundamental feature that will be included in SLRPO strategy. Dividend Yield Ratio is the measure of the ratio of Annual Dividends per Share to the Market Price per Share.

$$\boldsymbol{Dividend\ Yield} = \frac{Annual\ Dividends\ per\ Share}{Market\ Price\ per\ Share}$$

Debt-to-Equity Ratio (D/E)

Debt-to-Equity Ratio (D/E) is another fundamental feature that will be included in SLRPO strategy. Debt-to-Equity Ratio (D/E) is the measure of the ratio of Total Liabilities to the Shareholders' equity.

$$\boldsymbol{D/ERatio} = \frac{Total\ Liabilities}{Shareholders'Equity}$$

Book Value

Book Value is another fundamental feature that will be included in SLRPO strategy. Book Value is the measure of the difference between Total Assets and Total Liabilities.

$$\boldsymbol{Book\ Value} = Total\ Assets - Total\ Liabilities$$

Revenue

Revenue is another fundamental feature that will be included in SLRPO strategy. Revenue is the measure of the Total Sales of a company.

$$\boldsymbol{Revenue} = Total\ Sales$$

Revenue Growth

Revenue growth is another fundamental feature that will be included in SLRPO strategy. Revenue growth measures the increase in a company's sales from one period to the next.

$$\textbf{\textit{Revenue Growth}} = \frac{Current\ Period\ Revenue - Previous\ Period\ Revenue}{Previous\ Period\ Revenue} \times 100$$

For SLRPO strategy, the code for Fundamental Indicators is defined as follows:

```
        def get_stock_financials(self, symbol):
            stock = yf.Ticker(symbol)
            info = stock.info
            financials = {
                'Earnings Per Share (EPS)': info.get('trailingEps', 'Not available'),
                'Price to Earnings Ratio (P/E)': info.get('trailingPE', 'Not available'),
                'Projected Earnings Growth (PEG)': info.get('pegRatio', 'Not available'),
                'Free Cash Flow (FCF)': info.get('freeCashflow', 'Not available'),
                'Price to Book Ratio (P/B)': info.get('priceToBook', 'Not available'),
                'Return on Equity (ROE)': info.get('returnOnEquity', 'Not available'),
                'Dividend Payout Ratio (DPR)': info.get('payoutRatio', 'Not available'),
                'Price to Sales Ratio (P/S)': info.get('priceToSalesTrailing12Months', 'Not
available'),
                'Dividend Yield Ratio': info.get('dividendYield', 'Not available') * 100 if
info.get('dividendYield') is not None else 'Not available',
                'Debt-to-Equity Ratio (D/E)': info.get('debtToEquity', 'Not available'),
                'Book Value': info.get('bookValue', 'Not available'),
                'Revenue': info.get('totalRevenue', 'Not available'),
                'Revenue Growth': info.get('revenueGrowth', 'Not available') * 100 if
info.get('revenueGrowth') is not None else 'Not available'
            }
```

```
        financial_metrics = pd.DataFrame([financials])
        return financial_metrics
```

In the preceding code,

1. The fundamental indicators are loaded from Yahoo Finance.
2. All the fundamental indicators discussed in this section are used for this analysis.

These fundamental indicators are chosen since they cover various aspects of economic factors, financial factors, company performance, qualitative and quantitative factors. These fundamental indicators offer a balanced view of both short-term and long-term prospects, making them essential tools for SLRPO investment strategy design.

4.4.2 Generator

The Generator is the next functionality of the Indicators module that is responsible for generating the results for both Technical and Fundamental indicators that are defined in the Indicators module. This function acts as an essential part of the SLRPO Strategy providing a comprehensive set of indicators that can be used for further financial analysis and modeling.

The code for Generator is defined as follows:

```
    def generate_tech_indicators(ticker, start_date, end_date):
        tc = ti()
        info, hist = fetch_data(ticker, start_date, end_date)
        data = hist.reset_index()
        data2 =  data.copy(deep = True)
        data2['ADX'] = tc.calculate_adx(data)
        data2['AD'] = tc.calculate_accumulation_distribution(data)
        data2['ATR'] = tc.calculate_atr(data, window=14)
```

```
        data2[['BB_Upper Band', 'BB_Middle Band', 'BB_Lower Band']] =
tc.calculate_bollinger_bands(data['Close'])
        data2['CMF'] = tc.calculate_chaikin_money_flow(data)
        data2[['DCC_Upper Band', 'DCC_Middle Band', 'DCC_Lower Band']]
=tc.calculate_donchian_channel(data)
        data2['EOM'] = tc.calculate_ease_of_movement(data)
        data2[['KC_Upper Band', 'KC_Middle Band', 'KC_Lower Band']]
=tc.calculate_keltner_channel(data)
        data2['MFI'] = tc.calculate_money_flow_index(data)
        data2[['MACD', 'MACD_Signal Line']] = tc.calculate_macd(data)
        data2[['SMA', 'EMA']] = tc.calculate_moving_averages(data["Close"],
period=20)
        data2['BV'] = tc.calculate_on_balance_volume(data)
        data2['PS'] = tc.calculate_parabolic_sar(data['High'], data['Low'])
        data2[['Periodic High', 'Periodic Low']] =
tc.calculate_periodic_high_low(data['Close'], period=30)
        data2['RoC'] = tc.calculate_rate_of_change(data['Close'], period=10)
        data2['RS'] = tc.calculate_relative_strength(data['Close'])
        data2['RS'] = data2['RS'].replace(np.inf, 0)
        data2['RSI'] = tc.calculate_rsi(data['Close'])
        data2['RSI'] = data2['RSI'].replace(np.inf, 0)
        data2[['SOK', 'SOD']] = tc.calculate_stochastic_oscillator(data)
        data2['ST'] = tc.calculate_supertrend(data)
        data2['WV'] = tc.calculate_weighted_volume(data)
        data2['VaP'] = tc.calculate_volume_at_price(data)
```

```
data2['VPT'] = tc.calculate_volume_price_trend(data)
data2['VWAP'] = tc.calculate_vwap(data)
fin = tc.get_stock_financials(ticker)
datfinal = pd.concat([data2, fin])
datfinal = datfinal.fillna(method = 'bfill')
datfinal = datfinal.fillna(method = 'ffill')
datfinal = datfinal.fillna(0)
return datfinal
```

Key Functions of the Generator:

1. Integration of Technical and Fundamental Analysis
2. Data Acquisition and Preparation
3. Calling Technical Indicators
4. Calling Fundamental Indicators
5. Data Integration and Cleansing
6. Output Generation

Integration of Technical and Fundamental Analysis

The Generator function '*generate_tech_indicators*' handles both technical and fundamental analysis by generating the function calls for both types of indicators. This dual approach provides a holistic view of the stock's performance and potential future movements. While technical analysis focuses on historical price and volume data to identify patterns and trends, fundamental analysis looks at the intrinsic value of a stock based on financial statements and economic indicators. Both are essential for SLRPO strategy to work effectively.

Data Acquisition and Preparation

The Generator function also handles data acquisition by invoking the function needed to fetch the data '*fetch_data*' from Yahoo finance. This step ensures that both historical trading data and relevant stock information are available for SLRPO strategy. The historical data is also cleansed and structured in this function so that it can be provided as input to calculate the indicators.

Calling Technical Indicators

All the technical indicators discussed in this research under section *3.2.3.1 Technical Indicators* are called in the Generator function so that the indicators are calculated on the data fetched from Yahoo finance and cleansed in this module.

Calling Fundamental Indicators

All the fundamental indicators discussed in this research under section *3.2.3.2 Fundamental Indicators* are called in the Generator function so that the indicators are calculated on the data fetched from Yahoo finance and cleansed in this module. The fundamental indicators are calculated using the *get_stock_financials* function.

Data Integration and Cleansing

The technical and fundamental indicators are extracted and combined into one data frame. The data is then cleansed and prepared by handling missing values through backward filling and forward filling methods. The cleansed data will be used for further analysis of the SLRPO Strategy.

Output Generation

The Generator function returns the final data set that will be provided as input to the next modules of SLRPO strategy.

With the data being ready in the *Indicators* module, the next module handles the preprocessing required for SLRPO strategy development.

4.5 Preprocessing Module

The Preprocessing module is responsible to cleanse, finalize and prepare data for subsequent analysis. The main goal of this module is to ensure that the data is ready for SLRPO Investment strategy development and to create a final data set for all the stocks along with their corresponding technical and fundamental indicators.

The code for the preprocessor consists of the following functions and steps:

```
def read_config(config_file):
    config = configparser.ConfigParser()
    config.read(config_file)
    if 'Settings' in config:
        settings = config['Settings']
        tickers = [item.strip() for item in settings.get('tickers').split('\n') if item.strip()]
        return tickers
    else:
        raise ValueError("No 'Settings' section found in the config file")
start_date = '2020-01-01'
end_date = '2024-01-31'
config_file = 'config.ini'
tickers = read_config(config_file)
output_df = []
for ticker in tickers:
    ticker_df = generate_tech_indicators(ticker, start_date, end_date)
    ticker_df['Ticker'] = ticker
    output_df.append(ticker_df)
stocks_data = pd.concat(output_df)
```

```
        stocks_data = stocks_data.replace('Not available', 0)
        stocks_data = stocks_data.replace(np.inf, 0)
        problematic_indices = []
        for i, col in enumerate(stocks_data .columns):
          try:
            problematic_rows = np.where(np.isinf(stocks_data [col]) |
(np.abs(stocks_data [col]) > np.finfo(np.float64).max))[0]
            problematic_indices.extend([(row, i) for row in problematic_rows])
            if problematic_indices:
              print("Problematic values found at the following indices:")
              for row, col in problematic_indices:
                print(f"Row: {row}, Column: {stocks_data .columns[col]}")
                stocks_data .iloc[row, col] = 0
          except:
            pass
        stocks_data.to_csv('stocks_data.csv', index_label = False, index = False)
```

Key Functions of the Preprocessing Module are:

1. Configuration and Data Initialization
2. Data Collection and Integration
3. Data Cleansing
4. Error Handling
5. Finalizing the Data

Configuration and Data Initialization

The functionality of this step is to configure the setting required to read the config file and to read the list of stocks from the config file. Nifty 50 stocks are of interest for

this analysis and hence the list of Nifty 50 stocks will be provided in the configuration file as input.

The configuration also involves setting the start date and the end date required for the SLRPO strategy development. This step ensures that the data is configured to be collected in the right period so that it spans around the appropriate time frame for analysis.

Data Collection and Integration

The ‘generate_tech_indicators’ method is called to collect data for all stocks in the Nifty 50 portfolio configured in the configuration file. The stock data collected will be used to generate technical and fundamental indicators for the specified data range. The data is then appended in a list and converted into a single data set named ‘stocks_data’.

Data Cleansing

Cleansing of missing values and infinite values are also handled in this module to avoid any calculation errors that might come up due to ‘Not Available’ or ‘Infinite’ data that were generated during the feature creation process for Technical and Fundamental indicators. This Preprocessing module also checks for problematic values that might be too large or invalid. This includes detecting infinite values or values that exceed the maximum allowable float value. Any problematic values are identified, logged, and replaced with zeros to maintain data integrity.

Error Handling

The preprocessing module includes mechanisms to handle errors gracefully, such as missing configuration sections or issues during data processing. This robustness ensures that the preprocessing step can handle unexpected issues without failing entirely.

Finalizing the Data

The Preprocessing module also performs a final check to ensure that there are no remaining problematic data points present in the stocks data. This step is essential to ensure that there are no invalid data, and all values are within the expected ranges and the final input data set is clean. The cleaned and finalized dataset is then exported to a CSV file named ‘stocks_data.csv’. This ensures that the data is saved in a standardized format that can be easily accessed and used in subsequent stages of the SLRPO strategy.

The final stocks_data.csv is represented as follows:

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits	ADX	AD	...	Price to Book Ratio (P/B)	Return on Equity (ROE)	Dividend Payout Ratio (DPR)	Price to Sales Ratio (P/S)	Dividend Yield Ratio	Debt-to-Equity Ratio (D/E)
0	2020-01-01 00:00:00+05:30	206.97	208.40	204.64	205.83	1,553,127.00	0.00	0.00	23.46	-572,195.49	...	9.65	0.08	0.04	3.67	0.04	147.81
1	2020-01-02 00:00:00+05:30	205.98	211.13	205.48	209.14	2,991,937.00	0.00	0.00	23.46	320,135.47	...	9.65	0.08	0.04	3.67	0.04	147.81
2	2020-01-03 00:00:00+05:30	208.20	210.28	203.80	206.27	2,512,421.00	0.00	0.00	23.46	-274,407.80	...	9.65	0.08	0.04	3.67	0.04	147.81
3	2020-01-06 00:00:00+05:30	205.73	205.73	195.83	197.61	4,353,179.00	0.00	0.00	23.46	-3,060,439.70	...	9.65	0.08	0.04	3.67	0.04	147.81
4	2020-01-07 00:00:00+05:30	198.60	203.70	198.60	202.06	2,966,120.00	0.00	0.00	23.46	-1,994,935.51	...	9.65	0.08	0.04	3.67	0.04	147.81

Figure 4.2 Stocks Features Data after Preprocessing

The Preprocessing Module plays an important role in this strategy creation pipeline by ensuring that the data is clean, consistent, and properly structured for further analysis of SLRPO. This module reads configuration settings, collects and integrates data, handles missing and invalid values, and performs final integrity checks. This meticulous approach to data preprocessing is essential for accurate and meaningful financial analysis of the data for SLRPO strategy. The next module developed for this study is the Weight Generator module.

4.6 Weight Generator Module

The weight generator module is a very important module for SLRPO Strategy because it introduces a new concept in this research and study. The new concept is the ‘Strength Score’. Strength Score is a new metric introduced in this Thesis to quantify the

overall performance and health of a stock based on a comprehensive set of Technical and Fundamental indicators studies and included as features for the SLRPO strategy.

The weight generator module is designed to perform various tasks that are prerequisites for generating Strength Score.

Before discussing the key functions, it is relevant to look at the code for the functions performed by Weight Generator together and then discuss each of the functions handled.

The code for the Weight Generator module consists of the following definitions and steps.

Code snippet for Base Input data configuration is as follows:

```
config_file = 'config.ini'
tickers = read_config(config_file)
stocks = pd.read_csv('stocks_data.csv', low_memory=False)
stocks.fillna(0, inplace = True)
date_column = stocks.iloc[:, 0]
ticker_columns = stocks.iloc[:, -1]
```

Code snippet for Normalization is as follows:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
standardized_stocks = pd.DataFrame(scaler.fit_transform(stocks.iloc[:, 1:-1]),
columns=stocks.columns[1:-1])
normalized_stocks = pd.concat([date_column, standardized_stocks,
ticker_columns], axis=1)
standardized_stocks.index = date_column
features = standardized_stocks.copy(deep = True)
```

Code snippets for Weight Network Architecture, Strength Score definition and Weight calculation are as follows:

```
def calculate_weights(features, ticker):
    df = features.copy(deep = True)
    df = df.fillna(method = 'bfill')
    df = df.fillna(method = 'ffill')
    df = df.fillna(0)
    df = df[df['Ticker']==ticker]
    df.reset_index(inplace = True, drop = True)
    df.index =  df['Date']
    df.drop(columns = ['Date', 'Ticker'], inplace = True)
    X = df.drop('Return on Equity (ROE)', axis=1)
    y = df['Return on Equity (ROE)']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    model = keras.Sequential([
        keras.layers.Dense(256, activation='softmax',
input_shape=(X_train_scaled.shape[1],)),
        keras.layers.Dense(128, activation='softmax'),
        keras.layers.Dense(64, activation='softmax'),
        keras.layers.Dense(55, activation='softmax'),
        keras.layers.Dense(50, activation='softmax'),
```

```
        keras.layers.Dense(1)
    ])
    model.compile(optimizer='adam', loss='mean_squared_error')
    model.fit(X_train_scaled, y_train, epochs=10, batch_size=32)
    loss = model.evaluate(X_test_scaled, y_test)
    print('Test loss:', loss)
    predictions = model.predict(X_test_scaled)
    weights = model.layers[4].get_weights()[1]
    print('Weights:', weights)
    daily_scores = pd.DataFrame(index=X.index)
    for date, technical_scores in X.iterrows():
        all_scores = technical_scores.tolist()
        strength_score = sum([score * weight for score, weight in zip(all_scores,
weights)]) / sum(weights)
        daily_scores.at[date, 'Strength Score'] = strength_score
    return daily_scores
```

Key Functions of the Weight Generator Module are:

1. Normalization
2. Designing Weight Networks
3. Calculating Weights

1. Normalization

Normalization is the first step performed in the Weight Generator module to ensure all the features in the data that includes technical and fundamental indicators are on the same unit scale and can be processed further equally. Normalization is required to

ensure that the weights are calculated accurately, and the subsequent Strength Score generated is relevant for the stocks in this research.

Reading Data:

The stock data is loaded into the Weight Generator module in the form of a CSV file and the data includes all the features generated in the Preprocessing module.

Data cleansing and missing data handling is performed on the data after it is loaded to ensure that at every step the data integrity is preserved.

Organizing Data:

The Date column in the dataset is converted into index so that it can be used effectively in Time series data processing. The Ticker column from the data is separated from the dataset before normalizing the data since Ticker is an identifier and not a column to be normalized. This leaves only the feature columns in the dataset to be normalized.

Min Max Scaling:

The normalization of all the numerical features in the data are performed using Mix Max Scaling.

Min Max Scaling is calculated as follows:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

X – actual data point of the feature

X_{max} – maximum value of the feature

X_{min} – minimum value of the feature

X_{norm} – normalized value of the feature

Normalized Data Frame creation:

The normalized data is then converted back into the original data frame format. The Ticker column is added back to the data and the Date column is added as index for the dataset.

The normalized data for each feature is represented as follows:

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits	ADX	AD	...	Price to Book Ratio (P/B)	Return on Equity (ROE)	Dividend Payout Ratio (DPR)	Price to Sales Ratio (P/S)	Dividend Yield Ratio	Debt-to-Equity Ratio (D/E)	Book Value	Revenue
0	2020-01-01 00:00:00+05:30	0.02	0.02	0.02	0.02	0.00	0.00	0.00	0.21	0.40	...	0.01	0.22	0.01	0.01	0.00	0.48	0.12	0.11
1	2020-01-02 00:00:00+05:30	0.02	0.02	0.02	0.02	0.00	0.00	0.00	0.21	0.40	...	0.01	0.22	0.01	0.01	0.00	0.48	0.12	0.11
2	2020-01-03 00:00:00+05:30	0.02	0.02	0.02	0.02	0.00	0.00	0.00	0.21	0.40	...	0.01	0.22	0.01	0.01	0.00	0.48	0.12	0.11
3	2020-01-06 00:00:00+05:30	0.02	0.02	0.02	0.02	0.01	0.00	0.00	0.21	0.40	...	0.01	0.22	0.01	0.01	0.00	0.48	0.12	0.11
4	2020-01-07 00:00:00+05:30	0.02	0.02	0.02	0.02	0.00	0.00	0.00	0.21	0.40	...	0.01	0.22	0.01	0.01	0.00	0.48	0.12	0.11

Figure 4.3 Normalized Features of Stocks Data

Consistency and Integrity:

It is important to maintain the consistency and integrity of the data throughout the SLRPO Strategy. The Date column is set as the index of the data to ensure the data is consistent for further processing in the form of time series and all the features are maintained with data integrity. A deep copy of the data is made on the normalized features to ensure that the original dataset is preserved for verification purposes.

Data Cleansing:

Data cleansing and handling of missing values are performed in this module again to ensure that the normalized data does not through any calculation errors in the future steps. The missing values are handled using forward filling and backfilling methods. This ensures that the dataset is complete and ready for the weight calculation and strength score generation steps.

Defining Features and Targets:

The clean dataset is then divided into Features (X) and Target (y).

Features (X): All columns except the target variable (ROE) are used as features. These features represent the technical and fundamental indicators included in this research.

Target (y): The Return on Equity (ROE) column is used as the target variable that the model aims to predict. ROE is chosen as target variable since it is a key measure of a company's profitability and financial performance.

Splitting the Data:

The data is split into training and testing sets using a 70-30 split. This ensures that the model is trained on a subset of the data and evaluated on a separate subset to test its performance and generalization ability.

By carefully separating, normalizing, and reintegrating the date and ticker columns, the normalization process prepares a consistent and comprehensive dataset for subsequent analysis. This process is fundamental for accurate weight calculation and reliable strength score generation, enabling robust and insightful analysis in the SLRPO strategy.

2. Designing Weight Networks

The design for the newly introduced concept of ***Strength Score*** in this Thesis begins from designing the weight networks which is a neural network required to generate the weight parameters for Strength Score calculation. The goal of this networks is to consider the normalized technical and fundamental indicators as input features and predict the Return on Equity (RoE) for the Nifty 50 stocks under this study. The weights generated from this network will be considered as the input weights used on the features to calculate Strength scores. The Strength Score is a combination of normalized technical indicators and the prediction power of the network on RoE. The Strength Score and its explanation will be discussed further in the Strength Score module. This section in the Weight Generator module focuses on designing the neural networks.

The neural networks architecture designed in this module is used to learn the importance (weights) of various technical and fundamental indicators in predicting the Return on Equity (ROE) of stocks. It is to be noted that the goal of the SLRPO Investment Strategy developed in this Thesis is not to predict the Return on Equity (ROE) but to design the investment strategies with better opportunity costs. The weight calculation and the network design are an intermediate step in the feature engineering process for SLRPO.

The detailed breakdown of the architecture of the weight calculation neural networks model is explained as follows:

Input Layer

The number of features in the normalized stock data is configured as input in the input layer of the neural networks. Each neuron in the input layer represents one of these normalized features. The input shape is defined by the number of features, ensuring that the model can process the entire feature set.

Hidden Layers

The multiple Hidden layers of the neural networks are designed such that each layer has decreasing number of neurons. These layers help the network learn complex relationships between the features and the target variable (ROE). The architecture includes the following hidden layers.

- First Hidden Layer: 256 neurons with the Softmax activation function.
- Second Hidden Layer: 128 neurons with the Softmax activation function.
- Third Hidden Layer: 64 neurons with the Softmax activation function.
- Fourth Hidden Layer: 55 neurons with the Softmax activation function.
- Fifth Hidden Layer: 50 neurons with the Softmax activation function.

- Activation Function (Softmax): The Softmax activation function is used in the hidden layers to ensure that the output values are probabilities and falls between 0 and 1 as well as they total to 1. This is important since the output need to be interpreted as weights.

Output Layer

The output layer is designed with a single neuron and linear activation function since the purpose of this neuron is to predict the Return on Equity (ROE) of the stocks.

Model Compilation

The network is designed as a model with multiple layers, and it need to be compiled in this step. The network will be compiled using Adam optimizer. It is an adaptive learning rate optimization algorithm that has proven effective for a wide range of tasks. The mean squared error (MSE) loss function will be used to measure the difference between the predicted and the actual ROE values. The model aims to minimize this loss during training.

Training

The model is trained using the following parameters:

- Epochs is set to 10. The training dataset iterates through the network 10 times.
- Batch size is set to 32. The model updates the weights after processing 32 samples at a time.
- Model then trains on the training data and learns to adjust the weights of the neurons to minimize the MSE loss, effectively learning the importance of each indicator in predicting ROE.

Evaluation

The trained model is then evaluated on the test data ensure it generalizes well to unseen data.

The loss on the test set is printed to give an indication of the model's predictive accuracy.

The Network architecture designed in this module leverages the power of neural networks to capture complex relationships between indicators, enabling more accurate and insightful financial analysis that is needed to further develop the SLRPO strategy.

3. Calculating Weights

The process of calculating weights in the Weight Generator module involves using the designed neural networks to learn the importance of various technical and fundamental indicators in predicting the Return on Equity (ROE).

The steps discussed in this section coexist and overlaps with the steps discussed in section *2. Designing Weight Networks*. The steps are discussed as follows.

Weight Extraction

The weights for Strength score calculation is extracted from the second-to-last layer (fifth hidden layer) of the network.

These weights represent the relative importance of each indicator in predicting the ROE.

This layer is chosen because it provides a balance between high-level feature abstraction and direct output influence.

Strength Score Generation

- The extracted weights are then applied to the daily features of the stock to calculate the strength score.
- The strength score is then derived as a weighted average of all indicator values, reflecting the overall performance of the stock.

By training the network on historical stocks data, extracting learned weights, and applying these weights to calculate strength scores, the Weight Generator module

provides a robust tool for assessing stock performance and generating inputs that are needed to develop the SLRPO Investment strategy. The next module of SLRPO is the Strength score module which is a continuation of the Weight Generator module.

4.7 Strength Score Module

The Strength Score module is a continuation of Weight Generator module and generates the Strength Scores for each stock and their features. This module is designed to handle various tasks, including generation of strength scores, consolidation and export of stocks data.

The code for the Strength Score module consists of the following definitions and steps.

Code snipper for daily score generation is as follows:

```
scores = []
for ticker in tickers:
   try:
      score = calculate_weights(normalized_stocks, ticker)
      score.columns = [ticker]
      score = score.fillna(method = 'bfill')
      scores.append(score)
      print(score)
   except:
      pass
```

The code snipper for data consolidation and export is as follows:

```
df_scores = pd.concat(scores, axis = 1)
df_scores.to_csv('df_scores.csv')
```

Key Functions of the Strength Score Module are:

1. Generating Strength Scores

2. Consolidation and Export

1. Generating Strength Score

The ***Strength Score*** is a composite metric introduced in this research to quantify the overall performance and health of a stock by combining various technical and fundamental indicators. It is a single value that encapsulates the relative importance of multiple indicators, providing a comprehensive assessment of a stock's condition.

Key Characteristics of the Strength Score are defined as follows:

- Composite Metric
- Indicator Weightage
- Normalized metric
- Daily Calculation
- Holistic Assessment

Composite Metric

The Strength score integrates 23 Technical indicators that are discussed throughout SLRPO Strategy and considered as features and 13 Fundamental indicators that are also discussed throughout SLRPO Strategy and considered as features, into a single unified metric.

Indicator Weightage

Each indicator contributes to the Strength Score based on its relative importance, determined through the trained neural network designed in the Weight Generator module. This network assigns weights to each indicator, reflecting its significance in predicting the stock's performance.

Normalized metric

The Strength Score is normalized to ensure comparability across different stocks and time periods.

This normalization process adjusts the score to a common scale, typically between 0 and 1, facilitating direct comparisons.

Daily Calculation

The Strength Score is calculated daily, providing a dynamic and up-to-date measure of a stock's performance.

Holistic Assessment

By incorporating both technical and fundamental data, the Strength Score offers a holistic view of the stock, capturing short-term market dynamics as well as long-term financial health.

Calculation of Strength Score:

The Strength Score is calculated for each day as the weighted sum of all indicator values in the stocks dataset. The Strength Score is calculated as follows:

$$Strength\ Score = \frac{\sum(feature\ \times weight)}{\sum weights}$$

feature – each individual normalized indicator in the data set

weight – weight for each indicator extracted from the neural network

weights – All weights extracted from the neural networks

This weighted sum provides a single value that reflects the overall performance of the stock, considering the relative importance of each indicator.

2. Consolidation and Export

The daily Strength Scores are calculated in this module using the Strength Score and consolidated into one single data frame ‘df_scores’.

The consolidated data frame with Strength scores is represented as follows:

Date	ADANIENT.NS	ADANIPORTS.NS	APOLLOHOSP.NS	ASIANPAINT.NS	AXISBANK.NS	BPCL.NS	BAJAJ-AUTO.NS	BAJFINANCE.NS
2020-01-01 00:00:00+05:30	0.26	0.45	0.08	0.11	-0.04	0.18	0.66	0.52
2020-01-02 00:00:00+05:30	0.26	0.46	0.07	0.13	-0.05	0.18	0.69	0.60
2020-01-03 00:00:00+05:30	0.26	0.46	0.07	0.13	-0.05	0.19	0.69	0.59
2020-01-06 00:00:00+05:30	0.26	0.46	0.07	0.12	-0.05	0.18	0.69	0.60
2020-01-07 00:00:00+05:30	0.26	0.46	0.07	0.12	-0.05	0.18	0.69	0.60

Figure 4.4 Strength Scores for NIFTY 50 Stocks

The consolidated data is then exported into a CSV file ‘df_scores.csv’ for further processing.

Generating strength scores involves a systematic process of applying learned weights to normalized technical and fundamental indicators to produce a single metric that quantifies the overall performance of a stock. By calculating the weighted sum of indicators and normalizing the scores, this module provides a robust and comprehensive metric that can be used for various analytical and decision-making purposes.

4.8 Utility Functions Module

The Utility Functions Module is a supporting module for SLRPO Strategy, and it is designed to collate all the support functions necessary for the overall strategy and the upcoming modules. This module ensures that all utility functions are centralized in one location, so that they are easily accessible and efficient to use. The functions discussed in this module supports the Strategy modules and a detailed explanation of some of the concepts behind the domain related functions will be discussed in the Strategy module. Below are the list utility functions covered within this module:

1. Construct Portfolio Values
2. Calculate Strength Lag
3. Calculate Opportunity Cost
4. Update Reward

5. Calculate Daily Strength Lag
6. Create Features
7. Calculate allocation
8. Calculate model returns
9. Create Download Data Frame

Construct Portfolio Values

The construct portfolio values function is designed to transform raw stock data into a format that is suitable for portfolio analysis of SLRPO Strategies and their subsequent calculations.

The code for Construct Portfolio Values function is defined as follows:

```
def construct_portfolio_values(stocks):
    stocks_data = pd.read_csv(stocks)
    stocks_data = stocks_data.drop(columns = ['Date'])
    return np.array(stocks_data).T
```

The objective of the construct_portfolio_values function is to convert stock price data from its CSV file into a structured n-dimensional array that can be easily manipulated and analyzed.

Calculate Strength Lag

The calculate strength lag function is designed to measure the performance change, or strength lag, of the SLRPO input portfolio over a specific period. This function compares the portfolio's value at the end of the period to its value at the beginning, providing a simple yet powerful metric for assessing the overall performance.

The code for Calculate Strength Lag is defined as follows:

```
def calculate_strength_lag(portfolio_values):
    if len(portfolio_values) < 2:
```

```
        return 0
    return (portfolio_values[-1] - portfolio_values[0]) / portfolio_values[0]
```

The objective of the calculate_strength_lag function is to quantify the percentage change in SLRPO portfolio values over time.

The Strength Lag is calculated as follows:

$$StrengthLag = \frac{FinalValue - InitialValue}{InitialValue}$$

All values correspond to the input Portfolio of stocks data.

Calculate Opportunity Cost

The calculate opportunity cost function is designed to quantify the opportunity cost associated with selecting a particular portfolio in the SLRPO Strategy over the best available alternative within the various portfolios available for the NIFTY 50 stocks considered for this thesis.

The code for Calculate Opportunity Cost is defined as follows:

```
def calculate_opportunity_cost(chosen_portfolio_return, all_portfolios_returns):
    if not all_portfolios_returns:
        return 0
    best_alternative_return = max(all_portfolios_returns)
    return best_alternative_return - chosen_portfolio_return
```

The objective of the calculate_opportunity_cost function is to measure the difference in returns between the chosen portfolio and the best-performing alternative portfolio. Opportunity cost is a very important decision-making aspect for the SLRPO Strategy and has been discussed and analyzed in detail throughout this thesis.

Update Reward

The update reward function is designed to adjust the reward associated with an investment decision based on the opportunity costs incurred over time. This function leverages the concept of opportunity cost to provide a dynamic measure of how well a chosen portfolio performs relative to potential alternatives on a day-by-day basis.

The code for Update Reward is defined as follows:

```
def update_reward(chosen_return, all_returns):
    opportunity_costs = [calculate_opportunity_cost(chosen_return, [day_return]) for day_return in all_returns]
    return -opportunity_costs[-1] if opportunity_costs else 0
```

The objective of the update_reward function is to calculate a penalty (negative reward) that reflects the cost of missed opportunities by not selecting the best-performing portfolio each day. This penalty helps SLRPO Strategy understand the impact of its investment decisions over time and provides a feedback mechanism for improving future decisions.

Calculate Daily Strength Lag

The calculate daily strength lag function is designed to measure the daily percentage change in the values of stocks Strength Score. This function helps to quantify the day-to-day fluctuations in the data, providing insights into the volatility and performance dynamics over time.

The code for Calculate Daily Strength Lag is defined as follows:

```
def calculate_daily_strength_lag(data):
    return data.pct_change().fillna(0)
```

The Daily Strength Lag is calculated as follows:

$$PercentageChange = \frac{Score_t - Score_{t-1}}{Score_{t-1}}$$

Score – Strength Score

The objective of the calculate_daily_strength_lag function is to compute the daily strength lag, which is the percentage change in value of Strength Scores from one day to the next. This metric is crucial for analyzing trends, volatility, and the overall behavior of the stocks data over time.

Create Features

The create features function is designed to generate features for each stock based on its recent percentage change in Strength Score. These features are used as input for various portfolio development strategies designed within the SLRPO Investment Strategy in this thesis.

The code for Create Features is defined as follows:

```
def create_features(stock_data):
    features = []
    for stock, data in stock_data.items():
        if len(data) > 1:
            last_pct_change = data.iloc[-2:].pct_change().iloc[-1]
            features.append(last_pct_change)
        else:
            features.append(np.nan)
    return np.array(features)
```

The objective of the create_features function is to create a set of features that capture the recent performance of each stock in the Strength Scores dataset. By focusing on the recent percentage change, the function aims to provide a snapshot of the stock's most recent behavior, which can be critical for short-term predictions and decisions in the SLRPO Strategy.

Calculate allocation

The calc allocation function is designed to calculate allocation percentages based on Upper Confidence Bound (UCB) values which will be discussed in detail in the upcoming modules on Strategies. These allocations are used to distribute NIFTY 50 stocks into creating new portfolios according to their UCB values for SLRPO Strategy. The code for Calculate Allocation is defined as follows:

```
def calc_allocation(ucb_values):
    ucb_values = ucb_values
    min_ucb = np.min(ucb_values)
    max_ucb = np.max(ucb_values)
    scaled_ucb_values = 100 * (ucb_values - min_ucb) / (max_ucb - min_ucb)
    normalized_ucb_percentages = (scaled_ucb_values /
np.sum(scaled_ucb_values))
    return normalized_ucb_percentages
```

The objective of the calc_allocation function is to transform UCB values into normalized allocation percentages. This process ensures that investments are allocated proportionally to the stocks with higher UCB values, which typically represent better-performing or less risky investments.

Calculate Model Returns

The calc model returns function is designed to calculate the returns of the SLRPO model portfolio over a specified period. It considers the allocation of investments, the start and end dates of the analysis period, and the total investment amount to compute various performance metrics such as total gain/loss, cumulative returns, and annualized returns.

The code of Calculate Model Returns is defined as follows:

```
        def calc_model_returns(stocks_data, portfolio, allocation, start_date, end_date,
investment):
            analysis_data = stocks_data[['Date', 'Close', 'Ticker']]
            input_data = analysis_data.pivot_table(values='Close', index='Date',
columns='Ticker')
            input_data.reset_index(inplace=True)
            input_data['Date'] = pd.to_datetime(input_data['Date'])
            input_data['Date'] = input_data['Date'].dt.strftime('%d-%m-%Y')
            calculations = input_data[(input_data['Date'] == start_date) | (input_data['Date']
== end_date)].copy(deep=True)
            calculations.reset_index(inplace=True, drop=True)
            calculations.index = calculations['Date']
            calculations.drop(columns=['Date'], inplace=True)
            calculations = calculations[portfolio].copy(deep=True)
            diff_row = calculations.iloc[1] - calculations.iloc[0]
            calculations.loc['Difference'] = diff_row
            calculations.loc['Allocation'] = allocation
            calculations.loc['Invested'] = investment * calculations.loc['Allocation']
            calculations.loc['No_of_Shares'] = calculations.loc['Invested'] /
calculations.loc[start_date]
            calculations.loc['Return_on_Exit'] = calculations.loc['No_of_Shares'] *
calculations.loc[end_date]
            calculations.loc['Gain_Loss'] = calculations.loc['Difference'] *
calculations.loc['No_of_Shares']
            Total_Gain_Loss = calculations.loc['Gain_Loss'].sum()
```

```
        Total_Returns = calculations.loc['Return_on_Exit'].sum()
        Cumulative_Returns = (Total_Gain_Loss / investment) * 100
        start_date = datetime.strptime(start_date, '%d-%m-%Y')
        end_date = datetime.strptime(end_date, '%d-%m-%Y')
        num_years = (end_date - start_date).days / 365.25
        Annualized_Returns = ((1 + Cumulative_Returns/100) ** (1 / num_years) - 1)
* 100
        returns_df = pd.DataFrame(columns=['Total'])
        returns_df.loc['Investment'] = investment
        returns_df.loc['Total_Gain_Loss'] = Total_Gain_Loss
        returns_df.loc['Total_Returns'] = Total_Returns
        returns_df.loc['Cumulative_Returns'] = Cumulative_Returns
        returns_df.loc['Investment'] = returns_df.loc['Investment'].apply(lambda x:
'{:.2f}'.format(x))
        returns_df.loc['Total_Gain_Loss'] =
returns_df.loc['Total_Gain_Loss'].apply(lambda x: '{:.2f}'.format(x))
        returns_df.loc['Total_Returns'] = returns_df.loc['Total_Returns'].apply(lambda
x: '{:.2f}'.format(x))
        returns_df.loc['Annualized_Returns'] = Annualized_Returns
        returns_df.loc['Cumulative_Returns'] =
returns_df.loc['Cumulative_Returns'].apply(lambda x: '{:.2f}%'.format(x))
        returns_df.loc['Annualized_Returns'] =
returns_df.loc['Annualized_Returns'].apply(lambda x: '{:.2f}%'.format(x))
        return returns_df
```

The Annualized Returns for the portfolio over a period are calculated as follows:

$$\textbf{\textit{Annualized Returns}} = \left(\left(1 + \frac{Cumulative\ Returns}{100}\right)^{\frac{1}{Number\ of\ Years}} - 1 \right) \times 100$$

The objective of the calc_model_returns function is to provide a comprehensive evaluation of a portfolio's performance over a specific time frame. This involves calculating the initial and final values of the portfolio, determining the returns generated, and normalizing these returns to provide meaningful metrics for SLRPO Strategy investment decision-making.

Create Download Data Frame

The create download data frame function is designed to create a structured Data Frame that pairs a list of stocks with their corresponding allocation percentages. This Data Frame is useful for organizing and exporting investment allocation data in a clear and easily understandable format so that multiple strategies designed in SLRPO can be compared against each other for the results analysis.

The code for Create Download Data Frame is defined as follows:

```
def create_download_df(stocks, allocations):
    if len(stocks) != len(allocations):
        raise ValueError("Stocks and allocations lists must be of the same length.")
    df = pd.DataFrame({
        'Stocks': stocks,
        'Allocations %': allocations
    })
    return df
```

The objective of the create_download_df function is to generate a DataFrame that consolidates stocks and their respective allocations. This allows for the data to be easily reviewed, analyzed, and exported for results analysis of SLRPO strategy.

The Utility Functions Module consolidates all the essential supporting functions required for the complete SLRPO strategy and its relevant modules. By centralizing these functions, the module ensures that the overall strategy is efficient, maintainable, and easy to implement. Each function plays a specific role in data handling, calculation, and preparation, collectively contributing to a robust and comprehensive analytical framework.

4.9 Strategy Domain Classification

Now that all the prerequisite modules for SLRPO Investment strategy have been explained, the domain where these strategies are designed will be discussed in this section. The Strategy domains for SLRPO are classified into three major concepts of reinforcement learning.

- Upper Confidence Bound (UCB1)
- Policy Network
- Policy Gradient

Before delving further into the three main strategies built on these concepts, let us investigate the theoretical explanation of these concepts.

Reinforcement Leaning

Reinforcement Learning (RL) is a branch of machine learning where an agent learns to make decisions by performing certain actions in an environment to achieve maximum cumulative reward. In reinforcement learning, an agent interacts with its environment in discrete time steps. At each time step, the agent receives a state from the environment, chooses an action based on this state, and receives a reward along with the

next state. The goal of the agent is to learn a policy that maximizes the expected cumulative reward over time. Reinforcement Learning (RL) as a formalized concept in machine learning was extensively developed by Richard S. Sutton and Andrew G. Barto (Sutton, 1988).

Upper Confidence Bound (UCB1)

The Upper Confidence Bound (UCB1) algorithm is a widely used approach in the domain of multi-armed bandits, a fundamental problem in reinforcement learning. This problem involves an agent that must choose between multiple options (arms), each with an unknown reward distribution. The goal is to maximize the total reward over a series of trials by finding an optimal balance between exploration and exploitation (Auer, Cesa-Bianchi, & Fischer, 2002).

Exploration - is trying out different arms to gather information.

Exploitation - is selecting the arm known to yield the highest reward.

The UCB1 algorithm was introduced by Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer in their paper titled "Finite-time Analysis of the Multiarmed Bandit Problem."

Policy Network

A Policy Network is a type of neural network used in reinforcement learning (RL) to directly map states of an environment to actions to be taken by an agent. In reinforcement learning, an agent learns to make decisions by interacting with an environment. The goal is to maximize some notion of cumulative reward over time. The agent observes the state of the environment, takes an action, and receives a reward and a new state from the environment (Williams, 1992).

The Policy Network is a model that helps the agent decide which action to take given the current state.

Policy Gradient

Policy Gradient methods are a class of algorithms in reinforcement learning that optimize the policy directly. In reinforcement learning, an agent interacts with an environment in discrete time steps. At each step, the agent receives a state from the environment, takes an action based on its policy, and receives a reward. The goal of the agent is to maximize the cumulative reward over time (Williams, 1992).

Policy gradient methods achieve this by directly adjusting the parameters of the policy in the direction that increases the expected reward.

4.10 SLRPO Base Strategies

The three Base Strategies developed as part of SLRPO Investment strategy architecture are as follows:

1. Base Strategy – UCB1
2. Base Strategy – Policy Network
3. Base Strategy – Policy Gradient

These strategies are developed based on the concepts of reinforcement learning blended with the domain of Indian stock markets.

4.11 Base Strategy – UCB1 Module

The Base Strategy – UCB1 is developed on the concept of Upper Confidence Bound (UCB1) algorithm. The main components of Base Strategy – UCB1 are as follows:

1. Base Selector
2. Simple Selector
3. Stock Selector
4. Stock Allocation Maximizer
5. Optimizer

4.11.1 Base Selector

The Base Selector is the first sub strategy of Base Strategy - UCB1. The Base Selector is designed using the UCB1 algorithm and it provides a mechanism for selecting stocks based on their potential for high returns. It also ensures that less explored stocks are given a chance to be selected.

The code for Base Selector is defined as follows:

```
class UCB1_BaseSelector:
    def __init__(self, n_stocks):
        self.counts = np.zeros(n_stocks, dtype=np.float32)
        self.values = np.zeros(n_stocks, dtype=np.float32)
    def select_stock(self):
        n_stocks = len(self.counts)
        for stock in range(n_stocks):
            if self.counts[stock] == 0:
                return stock
        ucb_values = self.values + np.sqrt(2 * np.log(sum(self.counts)) / self.counts)
        return np.argmax(ucb_values)
    def update(self, chosen_stock, reward):
        self.counts[chosen_stock] += 1
        n = self.counts[chosen_stock]
        value = self.values[chosen_stock]
        new_value = ((n - 1) / n) * value + (1 / n) * reward
        self.values[chosen_stock] = new_value
```

In the preceding code,

- The Base Selector class implements the Upper Confidence Bound (UCB1) algorithm for selecting stocks.
- The Base Selector class initializes two arrays: counts and values.
- The select stock method in Base Selector class calculates the UCB values for each stock based on a confidence interval and returns the maximum value indices.
- UCB Values are calculated as follows:

$$UCB\ Values_{chosenstock} = Values_i + \sqrt{\frac{2 \cdot \ln(\sum Counts)}{Counts_{chosenstock}}}$$

- The update method in Base Selector class updates the UCB Values based on the reward for each stock selection. The updated UCB Values are calculated as follows.

$$New\ Value = \left(\frac{Counts_{chosenstock} - 1}{Counts_{chosenstock}}\right) \times Value + \left(\frac{1}{Counts_{chosenstock}}\right) \times Reward$$

- The Base Selector is a fundamental implementation of UCB1 algorithm for the SLRPO Strategy.

4.11.2 Simple Selector

The Simple Selector is the second sub strategy of Base Strategy - UCB1. The Simple Selector is also designed using the UCB1 algorithm and it is a slightly different implementation compared to Base Selector. The objective of this sub strategy is also to provide a mechanism for selecting stocks based on their potential for high returns.
The code for Simple Selector is defined as follows:

```
class UCB1_SimpleSelector:
    def __init__(self, num_stocks):
        self.num_stocks = num_stocks
```

```
        self.counts = np.zeros(num_stocks)
        self.values = np.zeros(num_stocks)
    def select_stock(self):
        max_upper_bound = 0
        selected_stock = None
        for stock in range(self.num_stocks):
            if self.counts[stock] > 0:
                average_reward = self.values[stock]
                delta_i = np.sqrt(2 * np.log(sum(self.counts) + 1) / self.counts[stock])
                upper_bound = average_reward + delta_i
            else:
                upper_bound = 1e400
            if upper_bound > max_upper_bound:
                max_upper_bound = upper_bound
                selected_stock = stock
        return selected_stock
    def update(self, stock, reward):
        self.counts[stock] += 1
        self.values[stock] = ((self.values[stock] * (self.counts[stock] - 1)) + reward) /
self.counts[stock]
```

In the preceding code,

- The Simple Selector class also implements the Upper Confidence Bound (UCB1) algorithm for selecting stocks.
- The Simple Selector class initializes number of stocks and the two arrays: counts and values.

- The select stock method in Base Selector class calculates the UCB values for each stock based on a confidence interval until it reaches an upper bound value that has been manually configured in the select stock method.
- UCB Values are calculated as follows:

$$UCB\ Values_{chosenstock} = Values_i + \sqrt{\frac{2 \cdot \ln\left(\sum Counts + 1\right)}{Counts_{chosenstock}}}$$

- The update method in Simple Selector class also updates the UCB Values based on the reward for each stock selection.
- The updated UCB Values are calculated as follows.

$$New\ Value = \left(\frac{Value\ \times (Counts_{chosenstock} - 1) + Reward}{Counts_{chosenstock}}\right)$$

The Simple Selector is a simple but slightly different implementation of UCB1 algorithm for the SLRPO Strategy where the UCB calculation is set to an upper bound manually.

4.11.3 Stock Selector

The Stock Selector is the third sub strategy of Base Strategy - UCB1. The Stock Selector is also designed using the UCB1 algorithm and it provides a mechanism for selecting stocks based on their potential for high returns. The stock selection in Stock Selector sub strategy is done in a way that balances the exploration of potentially profitable stocks and the exploitation of stocks that have already shown good performance by implementing a best stock selection logic.

The code for Stock Selector is defined as follows:

```
class UCB1_StockSelector:
    def __init__(self, stocks_strength, num_days, num_stocks_to_select):
```

```
        self.stocks_strength = stocks_strength
        self.num_days = num_days
        self.num_stocks_to_select = num_stocks_to_select
        self.n_stocks = len(stocks_strength.columns)
        self.counts = [0] * self.n_stocks
        self.total_reward = [0] * self.n_stocks
    def select_stocks(self):
        for day in range(self.num_days):
            ucb_values = self._calculate_ucb_values(day)
            chosen_stock = np.argmax(ucb_values)
            reward = self.stocks_strength.iloc[day, chosen_stock]
            self.counts[chosen_stock] += 1
            self.total_reward[chosen_stock] += reward
        return self._get_best_stocks()
    def _calculate_ucb_values(self, day):
        ucb_values = [0] * self.n_stocks
        for stock in range(self.n_stocks):
            if self.counts[stock] == 0:
                ucb_values[stock] = float('inf')
            else:
                average_reward = self.total_reward[stock] / self.counts[stock]
                delta_i = math.sqrt(2 * math.log(day + 1) / self.counts[stock])
                ucb_values[stock] = average_reward + delta_i
        return ucb_values
    def _get_best_stocks(self):
```

```
        top_stocks_indices = np.argsort(self.total_reward)[-
self.num_stocks_to_select:]
        best_stocks = [self.stocks_strength.columns[index] for index in
top_stocks_indices]
        return best_stocks
```

In the preceding code,

- The Stock Selector class also implements the Upper Confidence Bound (UCB1) algorithm for selecting stocks.
- The Stock Selector class initializes stock strength, number of days, number of stocks to select as input.
- It also initializes number of stocks and two arrays: counts and total rewards.
- The select stock method of Stock Selector class calculates the UCB values for each stock based on a confidence interval until it reaches maximum values automatically configured in the select stock method.
- The select stock method returns the best stocks based on the number of best stocks provided as limit in the get best stocks method.
- The UCB values in Stock Selector class are calculated daily and the rewards are updated for each trade day for the stocks data. This calculation happens in the calculate ucb values method.
- UCB Values are calculated as follows:

$$Average\ Reward = \frac{Total\ Reward_{chosenstock}}{Counts_{chosenstock}}$$

$$\Delta = \sqrt{\frac{2 \cdot \ln(day + 1)}{Counts_{chosenstock}}}$$

$$UCB\ Values_{chosenstock} = Average\ Reward + \Delta$$

The Stock Selector is a more complex and different implementation of UCB1 algorithm for the SLRPO Strategy where the UCB calculation is repeated for every trading day and limited best set of stocks is chosen based on historical performance.

4.11.4 Stock Allocation Maximizer

The Stock Allocation Maximizer is the fourth sub strategy of Base Strategy - UCB1. The Stock Allocation Maximizer is also designed using the UCB1 algorithm and it provides a mechanism for selecting stocks based on their potential for high returns. The stock selection in Stock Allocation Maximizer sub strategy is done in a way that balances the exploration of potentially profitable stocks and the exploitation of stocks that have already shown good performance but the number of stocks in this sub strategy are not restricted to a value. This sub strategy also considers the Stock Strength change instead of Stock Strength as its input. Considering the complete portfolio for allocation gives an opportunity of manual decision making.

The code for Stock Allocation Maximizer is defined as follows:

```
class UCB1_StockAllocationMaximizer:
    def __init__(self, stock_strength_change, num_days):
        self.stock_strength_change = stock_strength_change
        self.num_days = num_days
        self.n_stocks = len(stock_strength_change.columns)
        self.counts = [0] * self.n_stocks
        self.total_reward = [0] * self.n_stocks
        self.chosen_stocks_and_allocation_each_day = []
    def allocate_stocks(self):
```

```
            for day in range(self.num_days):
                ucb_values = self._calculate_ucb_values(day)
                total_ucb = sum(ucb_values)
                allocations = [(ucb / total_ucb) * 100 for ucb in ucb_values]
                stock_allocation_pairs = list(zip(self.stock_strength_change.columns,
allocations))
        self.chosen_stocks_and_allocation_each_day.append(stock_allocation_pairs)
                self._update_rewards(day, ucb_values)
            return self.chosen_stocks_and_allocation_each_day
        def _calculate_ucb_values(self, day):
            ucb_values = []
            for stock in range(self.n_stocks):
                if self.counts[stock] == 0:
                    ucb_values.append(float('inf'))
                else:
                    average_reward = self.total_reward[stock] / self.counts[stock]
                    confidence = math.sqrt(2 * math.log(day + 1) / self.counts[stock])
                    ucb_values.append(average_reward + confidence)
            return ucb_values
        def _update_rewards(self, day, ucb_values):
            for stock in range(self.n_stocks):
                reward = self.stock_strength_change.iloc[day, stock]
                self.counts[stock] += 1
                self.total_reward[stock] += reward
```

In the preceding code,

- The Stock Allocation Maximizer class also implements the Upper Confidence Bound (UCB1) algorithm for selecting stocks.
- The Stock Allocation Maximizer class initializes stock strength change, number of days to select as input.
- It also initializes number of stocks and three arrays: counts, total rewards and chosen stocks & allocations each day.
- The allocate stock method in the Stock Allocation Maximizer class calculates the ucb values daily and creates allocations based on chosen stock and its ucb values.
- The allocate stock method also updates the rewards daily.
- The allocate stock method returns the stock and its allocation for each day.

UCB Values are calculated as follows:

$$Average\ Reward = \frac{Total\ Reward_{chosenstock}}{Counts_{chosenstock}}$$

$$Confidence = \sqrt{\frac{2 \cdot \ln{(day + 1)}}{Counts_{chosenstock}}}$$

$$UCB\ Values_{chosenstock} = Average\ Reward + Confidence$$

The Stock Allocation Maximizer is a more complex and different implementation of UCB1 algorithm for the SLRPO Strategy where the UCB calculation is repeated for every trading day and considers all stocks from the portfolio of NIFTY 50. It also returns daily stock allocations by considering stock score change as the input.

4.11.5 Optimizer

The Optimizer is the fifth sub strategy of Base Strategy - UCB1. The Optimizer is also designed using the UCB1 algorithm and it provides a mechanism for selecting stocks based on their potential for high returns. The stock selection in Optimizer sub strategy is implemented such that it is a hybrid version of Stock Allocation Maximizer sub strategy and the Base Selector sub strategy. The Optimizer is designed with Stock Strength Change and UCB calculation logic similar to Stock Allocation Maximizer but the overall design is very simple similar to Base Selector strategy. The Optimizer performs daily updates of UCB values but does not perform daily allocations of stocks instead it provides the final best allocation for all stocks.

The code for Optimizer is defined as follows:

```
class UCB1_Optimizer:
    def __init__(self, stock_strength_change, num_days):
        self.stock_strength_change = stock_strength_change
        self.num_days = num_days
        self.n_stocks = len(stock_strength_change.columns)
        self.counts = [0] * self.n_stocks
        self.total_reward = [0] * self.n_stocks
        self.ucb_values = [0] * self.n_stocks
    def optimize(self):
        for day in range(self.num_days):
            self._calculate_ucb_values(day)
            chosen_stock = np.argmax(self.ucb_values)
            reward = self.stock_strength_change.iloc[day, chosen_stock]
            self.counts[chosen_stock] += 1
            self.total_reward[chosen_stock] += reward
```

```
        return self.total_reward
    def _calculate_ucb_values(self, day):
        for stock in range(self.n_stocks):
            if self.counts[stock] == 0:
                self.ucb_values[stock] = float('inf')
            else:
                average_reward = self.total_reward[stock] / self.counts[stock]
                confidence = math.sqrt(2 * math.log(day + 1) / self.counts[stock])
                self.ucb_values[stock] = average_reward + confidence
```

In the preceding code,

- The Optimizer class also implements the Upper Confidence Bound (UCB1) algorithm for selecting stocks.
- The Optimizer class initializes stock strength change, number of days to select as input.
- It also initializes number of stocks and three arrays: counts, total rewards and UCB values.
- The optimize method in the Optimizer class performs daily updates on the UCB values based on chosen stock and reward.
- UCB Values are calculated as follows:

$$Average\ Reward = \frac{Total\ Reward_{chosenstock}}{Counts_{chosenstock}}$$

$$Confidence = \sqrt{\frac{2 \cdot \ln{(day + 1)}}{Counts_{chosenstock}}}$$

$$UCB\ Values_{chosenstock} = Average\ Reward + \ Confidence$$

The Optimizer is a simpler and hybrid version of the Stock Allocation Maximizer and Base Selector. It is designed to consider the stock strength score change as input and provide one best portfolio with allocations that considers all the stocks in the NIFTY 50 list studied in this thesis.

4.12 Base Strategy – Policy Network Module

The Base Strategy – Policy Network is developed on the concept of Policy Network algorithm. The main components of Base Strategy – Policy Network are as follows:

1. Policy Network Softmax
2. Policy Network ReLU
3. Policy Network Batch Norm
4. Policy Network Standard

4.12.1 Policy Network Softmax

The Policy Network Softmax class is the first sub strategy of Base Strategy - Policy Network. This sub strategy will be used to create a policy for reinforcement learning based models in the SLRPO strategy. Policy Network helps the reinforcement learning agent with guidance on the next action to be taken. The Policy Network Softmax class is a neural network designed with Softmax as the activation function.

The code for Policy Network Softmax is defined as follows:

```
class PolicyNetworkSoftmax:
    def __init__(self, input_dim, output_dim):
        self.input_dim = input_dim
        self.output_dim = output_dim
        self.model = None
```

```
            self._create_network()
        def _create_network(self):
            tf.random.set_seed(1)
            self.model = Sequential([
                Dense(128, input_dim=self.input_dim, activation='softmax'),
                Dense(64, activation='softmax'),
                Dense(32, activation='softmax'),
                Dense(16, activation='softmax'),
                Dense(self.output_dim, activation='softmax')
            ])
            self.model.compile(optimizer='adam', loss='mse')
        def get_model(self):
            return self.model
```

In the preceding code,

- The Policy Network Softmax class initializes input dimension and output dimension for the Softmax network.
- It also initializes the network with a model variable to store the neural network model and a method to create the network.
- The create network method creates the Softmax network with 5 dense layers, the activation function as Softmax in all the layers.
- The create network method also compiles the model with Adam optimizer and with Mean Square Error as Loss function.

The Policy Network Softmax is a simple sub strategy developed in the Base Strategy – Policy Network to provide probability distributions in each layer and to generate a policy that can be used in the SLRPO Investment strategy implementation.

4.12.2 Policy Network ReLU

The Policy Network ReLU class is the second sub strategy of Base Strategy - Policy Network. This sub strategy will also be used to create a policy for reinforcement learning based models in the SLRPO strategy. This Policy Network also helps the reinforcement learning agent with guidance on the next action to be taken. The Policy Network ReLU class is a neural network designed with ReLU as the activation function. The code for Policy Network ReLU is defined as follows:

```
class PolicyNetworkReLU:
    def __init__(self, input_size):
        self.input_size = input_size
        self.model = None
        self._create_network()
    def _create_network(self):
        tf.random.set_seed(1)
        self.model = Sequential([
            Dense(64, activation='relu', input_shape=(self.input_size,)),
            Dense(32, activation='relu'),
            Dense(self.input_size, activation='softmax')
        ])
        self.model.compile(optimizer='adam', loss='mse')
    def get_model(self):
        return self.model
```

In the preceding code,

- The Policy Network ReLU class initializes input size for the ReLU network.

- It also initializes the network with a model variable to store the neural network model and a method to create the network.
- The create network method creates the ReLU network with 3 dense layers.
- The activation function for the first two dense layers is set as ReLU and for the last dense layer as Softmax since the output from the policy is expected to be a probability distribution.
- The create network method also compiles the model with Adam optimizer and with Mean Square Error as Loss function.

The Policy Network ReLU is another simple sub strategy developed in the Base Strategy – Policy Network to provide probability distributions in the final layer and to generate a policy that can be used in the SLRPO Investment strategy implementation.

4.12.3 Policy Network Batch Norm

The Policy Network Batch Norm class is the third sub strategy of Base Strategy - Policy Network. This sub strategy will also be used to create a policy for reinforcement learning based models in the SLRPO strategy. This Policy Network also helps the reinforcement learning agent with guidance on the next action to be taken. The Policy Network Batch Norm class is a neural network designed with multiple complex layers and Softmax as the final activation function.

The code for Policy Network Batch Norm is defined as follows:

```
class PolicyNetworkBatchNorm:
    def __init__(self, input_size):
        self.input_size = input_size
        self.model = None
        self._create_network()
    def _create_network(self):
```

```
        tf.random.set_seed(1234)
        self.model = Sequential()
        self.model.add(Dense(128, activation='relu', input_shape=(self.input_size,)))
        self.model.add(BatchNormalization())
        self.model.add(Dense(64, activation='relu'))
        self.model.add(Dropout(0.3))
        self.model.add(Dense(64, activation='relu'))
        self.model.add(BatchNormalization())
        self.model.add(Dropout(0.3))
        self.model.add(Dense(self.input_size, activation='softmax'))
        optimizer = Adam(learning_rate=0.001)
        self.model.compile(optimizer=optimizer, loss='mse')
    def get_model(self):
        return self.model
```

In the preceding code,

- The Policy Network Batch Norm class initializes input size for the Batch Norm network.
- It also initializes the network with a model variable to store the neural network model and a method to create the network.
- The create network method creates the Batch Norm network with multiple layers that are complex and are a combination of sequential, dense, batch normalization and drop out layers.
- The activation function for the all the hidden layers is set as ReLU and for the last dense layer as Softmax since the output from the policy is expected to be a probability distribution.

- The create network method also compiles the model with Adam optimizer and with Mean Square Error as Loss function.

The Policy Network Batch Norm is another sub strategy developed in the Base Strategy – Policy Network to provide probability distributions in the final layer and to generate a policy that can be used in the SLRPO Investment strategy implementation. This is a complex policy network compared to all other base strategy policy networks and is expected to provide more hidden insights on the stocks data.

4.12.4 Policy Network Standard

The Policy Network Standard class is the fourth and final sub strategy of Base Strategy - Policy Network. This sub strategy will also be used to create a policy for reinforcement learning based models in the SLRPO strategy. This Policy Network also helps the reinforcement learning agent with guidance on the next action to be taken. The Policy Network Standard class is a neural network designed as a hybrid version of Policy Network Softmax and Policy Network ReLU.

The code for Policy Network Standard is defined as follows:

```
class PolicyNetworkStandard:
    def __init__(self, input_dim, output_dim):
        self.input_dim = input_dim
        self.output_dim = output_dim
        self.model = self._create_network()
    def _create_network(self):
        tf.random.set_seed(1234)
        model = Sequential([
            Dense(64, input_dim=self.input_dim, activation='relu'),
            Dense(32, activation='relu'),
```

```
            Dense(self.output_dim, activation='softmax')
        ])
        model.compile(optimizer='adam', loss='mse')
        return model
    def get_model(self):
        return self.model
```

In the preceding code,

- The Policy Network Standard class initializes both input dimension and output dimension for the Hybrid network unlike the Policy Network ReLU where only input size is provided as input.
- It also initializes the network with a model variable to store the neural network model and a method to create the network.
- The create network method creates a Hybrid network with 3 dense layers.
- The activation function for the first two dense layers is set as ReLU and the last dense layer as Softmax in the last layer since the output from the policy is expected to be a probability distribution.
- The create network method also compiles the model with Adam optimizer and with Mean Square Error as Loss function.

The Policy Network Standard is another simple, but hybrid sub strategy developed in the Base Strategy – Policy Network to provide probability distributions in the final layer and to generate a policy that can be used in the SLRPO Investment strategy implementation.

4.13 Base Strategy – Policy Gradient Module

The final base strategy for SLRPO Investment strategy is the Policy Gradient. Unlike UCB1 and Policy Network base strategies, Policy Gradient is not a combination

of multiple sub strategies but is a combination of multiple components of a Policy Gradient that implements Reinforcement Learning.

Policy Gradient has the following four components which will be used in building new strategies in this thesis:

- Simple Policy Search
- Policy Gradient Network
- Policy Gradient Agent
- Complex Policy Search

Let us now discuss each of these components and their code.

4.13.1 Simple Policy Search

Simple Policy Search is designed to generate a policy needed for stock selection. This policy is developed to represent a probability distribution of stocks based on their historical performance. Using this policy, it will be possible to select stocks with higher performance since they will be given higher rewards and gets a higher probability for selection.

The code for Simple Policy Search is defined as follows:

```
def simple_policy_search(stock_strength_change, total_reward):
    n_stocks = len(stock_strength_change.columns)
    total = sum(total_reward)
    policy = [reward / total for reward in total_reward]
    return policy
```

In the preceding code,

- The Simple Policy Search function is defined with two input parameters:
- Stock Strength Change: This is the change in Strength Scores that are provided as input to the SLRPO strategy.

- Total Reward: This is the list of total rewards calculated by one of the UCB1 algorithms.
- Policy is then calculated as follows:

$$Policy[i] = \left[\frac{Total\ Reward\ [i]}{Total\ of\ all\ Rewards}\right]$$

This is a simple and straight forward policy that will be incorporated into one of the sub strategies of SLRPO Investment strategy.

4.13.2 Policy Gradient Network

Policy Gradient Network is a component of the Base Strategy – Policy Gradient. This network is designed as a use case specific network which can be used within the policy gradient component of reinforcement learning.

The code for Policy Gradient Network is defined as follows:

```
class PolicyGradientNetwork(tf.keras.Model):
    def __init__(self, n_stocks):
        super(PolicyGradientNetwork, self).__init__()
        self.dense1 = tf.keras.layers.Dense(128, activation='relu')
        self.dense2 = tf.keras.layers.Dense(64, activation='relu')
        self.output_layer = tf.keras.layers.Dense(n_stocks, activation='softmax')
    def call(self, inputs):
        x = self.dense1(inputs)
        x = self.dense2(x)
        return self.output_layer(x)
```

In the preceding code,

- The Policy Gradient Network takes in the number of stocks as input and generates an output which will be a probability distribution.

- In reinforcement learning of SLRPO Strategy, this probability distribution will be used for deciding the actions.
- The network is designed with two dense layers and one output layer.

Softmax is considered as the activation function since we expect the output to be probability distributions.

4.13.3 Policy Gradient Agent

Policy Gradient Agent is the third component of the Base Strategy – Policy Gradient. The objective of this agent is to utilize the Policy Gradient Network to learn a policy which provides probabilities as output for stock selection. The training process uses gradient descent to adjust the weights of the network.

The code for Policy Gradient Agent is defined as follows:

```
class PolicyGradientAgent:
    def __init__(self, n_stocks):
        self.n_stocks = n_stocks
        self.model = PolicyGradientNetwork(n_stocks)
        self.optimizer = tf.keras.optimizers.Adam(learning_rate=0.01)
    def train(self, stocks_strength, total_reward):
        with tf.GradientTape() as tape:
            policy = self.model(stocks_strength)
            loss = self.compute_loss(policy, total_reward)
        gradients = tape.gradient(loss, self.model.trainable_variables)
        self.optimizer.apply_gradients(zip(gradients, self.model.trainable_variables))
    def compute_loss(self, policy, total_reward):
        log_probs = tf.math.log(policy)
        return -tf.reduce_sum(log_probs * total_reward)
```

```
def get_policy(self, stocks_strength):
    policy = self.model.predict(np.array([stocks_strength]))
    return policy
```

In the preceding code,

- The Policy Gradient Agent takes the number of stocks as input.
- The agent then initializes the Policy Network and an optimizer for the network.
- The agent then trains the network using stock strength score and adjusts the weights using gradient descent.
- The agent adjusts the loss function using total reward and then returns the policy.

4.13.4 Complex Policy Search

The final component of Base Strategy – Policy Gradient is the Complex Policy Search. The objective of this component is to use the Policy Gradient Agent efficiently to get better stock allocations for SLRPO investment strategy.

The code for Complex Policy Search is defined as follows:

```
def complex_policy_search(stocks_strength, total_reward, n_stocks):
    if not isinstance(stocks_strength, np.ndarray):
        stocks_strength = np.array(stocks_strength)
    agent = PolicyGradientAgent(n_stocks)
    agent.train(stocks_strength, total_reward)
    return agent.get_policy(stocks_strength)
```

In the preceding code,

1. The complex policy search function takes in stock strength score, total reward and number of stocks as input.

2. The Policy Gradient Agent is trained using the above inputs and the policy is generated as output.

These are various components and their functions implemented in Base Strategy – Policy Gradient.

4.14 Main 8 Strategies of SLRPO and Portfolio Creation

In this thesis, I am introducing the following 8 main strategies of **SLRPO** which are developed to blend the Base Strategies and Utility functions to create new portfolios based on the NIFTY 50 stocks, their technical indicators, fundamental indicators and newly introduce **Strength Score**.

The input data required for execution of these 8 strategies are configured as follows:

```
config_file = 'config.ini'
tickers = read_config(config_file)
stocks_strength = pd.read_csv('df_scores.csv')
stocks_strength.index = stocks_strength['Date']
stocks_strength.drop(columns = 'Date', inplace = True)
num_rounds = len(stocks_strength)
num_stocks = len(tickers)
stocks_data = pd.read_csv('stocks_data.csv')
start_date = '01-01-2020'
end_date = '30-01-2024'
Investment = 100000
```

The stocks considered for this analysis are as follows:

ADANIENT.NS	BRITANNIA.NS	HEROMOTOCO.NS	LTIM.NS	SUNPHARMA.NS
ADANIPORTS.NS	CIPLA.NS	HINDUNILVR.NS	M&M.NS	TATACONSUM.NS
APOLLOHOSP.NS	COALINDIA.NS	HINDALCO.NS	MARUTI.NS	TATAMOTORS.NS
ASIANPAINT.NS	DIVISLAB.NS	ICICIBANK.NS	NESTLEIND.NS	TATASTEEL.NS
AXISBANK.NS	DRREDDY.NS	INDUSINDBK.NS	NTPC.NS	TCS.NS
BPCL.NS	EICHERMOT.NS	INFY.NS	ONGC.NS	TECHM.NS
BAJAJ-AUTO.NS	GRASIM.NS	ITC.NS	POWERGRID.NS	TITAN.NS
BAJFINANCE.NS	HCLTECH.NS	JSWSTEEL.NS	RELIANCE.NS	ULTRACEMCO.NS
BAJAJFINSV.NS	HDFCBANK.NS	KOTAKBANK.NS	SBILIFE.NS	UPL.NS
BHARTIARTL.NS	HDFCLIFE.NS	LT.NS	SBIN.NS	WIPRO.NS

Table 4.1 NIFTY 50 Stocks

4.14.1 Strategy1

Strategy 1 is developed with a blend of the following components:

- Base Strategy – Policy Network Softmax
- Base Strategy – UCB1 Base Selector
- Utility Function – Construct Portfolio Values
- Utility Function – Calculate Strength Lag
- Utility Function – Update Reward
- Utility Function – Calculate Allocation

The code for Strategy 1 is defined as follows:

```
class Strategy1:
    def __init__(self, tickers):
        self.tickers = tickers
        self.n_portfolios = len(tickers)
        self.portfolio_values = construct_portfolio_values('df_scores.csv')
        self.n_days = len(self.portfolio_values)
        self.policy_network = PolicyNetworkSoftmax(input_dim=1, output_dim=1)
        self.model = self.policy_network.get_model()
        self.ucb_selector = UCB1_BaseSelector(self.n_portfolios)
        np.random.seed(1)
        random.seed(1)
```

```
            tf.random.set_seed(1)
        def strategy1(self):
            for day in range(1, self.n_days):
                chosen_stock = self.ucb_selector.select_stock()
                chosen_portfolio_values = self.portfolio_values[:day, chosen_stock]
                chosen_return =
np.array(calculate_strength_lag(chosen_portfolio_values))
                all_portfolios_returns = [calculate_strength_lag(self.portfolio_values[:day,
i]) for i in range(self.n_portfolios)]
                allocation = np.array(self.model.predict(chosen_return.reshape(1, -1))[0])
                reward = update_reward(allocation, all_portfolios_returns)
                self.ucb_selector.update(chosen_stock, reward)
            ucb_values = self.ucb_selector.values
            allocation3 = calc_allocation(ucb_values)
            return allocation3
```

The output of Strategy1 is a new portfolio allocation of NIFTY 50 stocks as follows:

Stocks	Allocations %	Stocks	Allocations %	Stocks	Allocations %	Stocks	Allocations %	Stocks	Allocations %
ADANIENT.NS	2%	BRITANNIA.NS	2%	HEROMOTOCO.NS	0%	LTIM.NS	2%	SUNPHARMA.NS	2%
ADANIPORTS.NS	2%	CIPLA.NS	3%	HINDUNILVR.NS	3%	M&M.NS	2%	TATACONSUM.NS	3%
APOLLOHOSP.NS	2%	COALINDIA.NS	2%	HINDALCO.NS	2%	MARUTI.NS	2%	TATAMOTORS.NS	1%
ASIANPAINT.NS	2%	DIVISLAB.NS	2%	ICICIBANK.NS	1%	NESTLEIND.NS	5%	TATASTEEL.NS	2%
AXISBANK.NS	2%	DRREDDY.NS	2%	INDUSINDBK.NS	3%	NTPC.NS	1%	TCS.NS	2%
BPCL.NS	2%	EICHERMOT.NS	2%	INFY.NS	2%	ONGC.NS	2%	TECHM.NS	2%
BAJAJ-AUTO.NS	1%	GRASIM.NS	8%	ITC.NS	2%	POWERGRID.NS	2%	TITAN.NS	2%
BAJFINANCE.NS	1%	HCLTECH.NS	2%	JSWSTEEL.NS	2%	RELIANCE.NS	2%	ULTRACEMCO.NS	2%
BAJAJFINSV.NS	2%	HDFCBANK.NS	2%	KOTAKBANK.NS	2%	SBILIFE.NS	1%	UPL.NS	1%
BHARTIARTL.NS	2%	HDFCLIFE.NS	1%	LT.NS	1%	SBIN.NS	0%	WIPRO.NS	2%

Table 4.2 Strategy 1 Portfolio Allocation of all NIFTY 50 Stocks

The Table 4.2 demonstrates the ability of Strategy 1 to analyze the historical data of NIFTY 50 stocks and construct a portfolio that provides an allocation for long term

investments. Strategy 1 considers all 50 stocks since there are no limits set on the number of stocks to be chosen in this strategy. The returns of this strategy will be discussed in the Chapter 6.

4.14.2 Strategy2

Strategy 2 is developed with a blend of the following components:

- Base Strategy – UCB1 Simple Selector
- Limiting Portfolio size to 7

The code for Strategy 2 is defined as follows:

```
class Strategy2:
    def __init__(self, num_stocks, num_rounds, tickers, stocks_strength):
        self.num_stocks = num_stocks
        self.num_rounds = num_rounds
        self.tickers = tickers
        self.stocks_strength = stocks_strength
        self.portfolio_size = 7
        self.ucb = UCB1_SimpleSelector(num_stocks)
        np.random.seed(1)
        random.seed(1)
        tf.random.set_seed(1)
    def strategy2(self):
        for round in range(self.num_rounds):
            selected_stock = self.ucb.select_stock()
            reward = self.stocks_strength.iloc[round, selected_stock]
            self.ucb.update(selected_stock, reward)
        top_stocks_indices = np.argsort(-self.ucb.values)[:self.portfolio_size]
```

```
        top_stocks = [self.tickers[i] for i in top_stocks_indices]
        allocation = [1.0 / len(top_stocks) for _ in top_stocks]
        return top_stocks, allocation
```

The output of Strategy2 is a new portfolio allocation of NIFTY 50 stocks as follows:

Stocks	Allocations %
SBIN.NS	14%
LT.NS	14%
UPL.NS	14%
SBILIFE.NS	14%
NTPC.NS	14%
BAJAJ-AUTO.NS	14%
BAJFINANCE.NS	14%

Table 4.3 Strategy 2 Portfolio Allocation of all NIFTY 50 Stocks

The Table 4.3 demonstrates the ability of Strategy 2 to analyze the historical data of NIFTY 50 stocks and construct a portfolio that provides an allocation for long term investments. Strategy 2 considers only the top 7 stocks since there are limits set on the number of stocks to be chosen in this strategy. The returns of this strategy will be discussed in the Chapter 6.

4.14.3 Strategy3

Strategy 3 is developed with a blend of the following components:

- Base Strategy – UCB1 Stock Selector
- Utility Function – Create Features
- Base Strategy – Policy Network ReLU
- Limiting Portfolio size to 10

The code for Strategy 3 is defined as follows:

```
    class Strategy3:
        def __init__(self, stocks_strength, num_stocks, portfolio_size):
            self.stocks_strength = stocks_strength
            self.num_stocks = num_stocks
```

```
        self.portfolio_size = portfolio_size
        np.random.seed(1)
        random.seed(1)
        tf.random.set_seed(1)
        self.selector = UCB1_StockSelector(stocks_strength, len(stocks_strength),
portfolio_size)
    def select_stocks(self):
        selected_stocks = self.selector.select_stocks()
        return selected_stocks
    def get_stock_data(self, selected_stocks):
        selected_stock_data = {stock: self.stocks_strength[stock] for stock in
selected_stocks}
        return selected_stock_data
    def strategy3(self):
        selected_stocks = self.select_stocks()
        selected_stock_data = self.get_stock_data(selected_stocks)
        stock_features = create_features(selected_stock_data)
        policy_network = PolicyNetworkReLU(len(selected_stocks))
        model = policy_network.get_model()
        allocations = model.predict(np.array([stock_features.flatten()]))
        allocation_list = allocations.tolist()[0]
        return selected_stocks, allocation_list
```

The output of Strategy3 is a new portfolio allocation of NIFTY 50 stocks as follows:

Stocks	Allocations %
EICHERMOT.NS	10%
ICICIBANK.NS	10%
WIPRO.NS	10%
BAJFINANCE.NS	10%
BAJAJ-AUTO.NS	10%
NTPC.NS	10%
SBILIFE.NS	10%
UPL.NS	10%
LT.NS	10%
SBIN.NS	10%

Table 4.4 Strategy 3 Portfolio Allocation of all NIFTY 50 Stocks

The Table 4.4 demonstrates the ability of Strategy 3 to analyze the historical data of NIFTY 50 stocks and construct a portfolio that provides an allocation for long term investments. Strategy 3 considers only the top 10 stocks since there are limits set on the number of stocks to be chosen in this strategy. The returns of this strategy will be discussed in the Chapter 6.

4.14.4 Strategy4

Strategy 4 is developed with a blend of the following components:

- Base Strategy – UCB1 Base Selector
- Base Strategy – Policy Network Standard
- Utility Function – Calculate Allocation

The code for Strategy 4 is defined as follows:

```
class Strategy4:
    def __init__(self, num_stocks, stocks_strength, tickers):
        self.num_stocks = num_stocks
        self.stocks_strength = stocks_strength
        self.tickers = tickers
        np.random.seed(1)
```

```
            random.seed(1)
            tf.random.set_seed(1)
            self.ucb = UCB1_BaseSelector(num_stocks)
            self.policy_network = PolicyNetworkStandard(input_dim=1,
output_dim=num_stocks)
            self.model = self.policy_network.get_model()
        def strategy4(self):
            for day in range(self.stocks_strength.shape[0]):
                chosen_stock = self.ucb.select_stock()
                current_data =
np.array([self.stocks_strength[self.tickers[chosen_stock]].iloc[day]])
                allocation = self.model.predict(current_data.reshape(1, -1))[0]
                reward = self.stocks_strength[self.tickers[chosen_stock]].iloc[day]
                self.ucb.update(chosen_stock, reward)
                self.model.fit(current_data.reshape(1, -1), allocation.reshape(1, -1),
epochs=1, verbose=0)
            ucb_values = self.ucb.values
            allocation_final = calc_allocation(ucb_values)
            return allocation_final
```

The output of Strategy3 is a new portfolio allocation of NIFTY 50 stocks as follows:

Stocks	Allocations %	Stocks	Allocations %	Stocks	Allocations %	Stocks	Allocations %	Stocks	Allocations %
ADANIENT.NS	2%	BRITANNIA.NS	2%	HEROMOTOCO.NS	0%	LTIM.NS	2%	SUNPHARMA.NS	2%
ADANIPORTS.NS	2%	CIPLA.NS	1%	HINDUNILVR.NS	2%	M&M.NS	2%	TATACONSUM.NS	2%
APOLLOHOSP.NS	2%	COALINDIA.NS	2%	HINDALCO.NS	2%	MARUTI.NS	2%	TATAMOTORS.NS	2%
ASIANPAINT.NS	2%	DIVISLAB.NS	2%	ICICIBANK.NS	2%	NESTLEIND.NS	1%	TATASTEEL.NS	2%
AXISBANK.NS	2%	DRREDDY.NS	2%	INDUSINDBK.NS	2%	NTPC.NS	2%	TCS.NS	2%
BPCL.NS	2%	EICHERMOT.NS	2%	INFY.NS	2%	ONGC.NS	2%	TECHM.NS	2%
BAJAJ-AUTO.NS	2%	GRASIM.NS	1%	ITC.NS	2%	POWERGRID.NS	2%	TITAN.NS	2%
BAJFINANCE.NS	2%	HCLTECH.NS	2%	JSWSTEEL.NS	2%	RELIANCE.NS	2%	ULTRACEMCO.NS	2%
BAJAJFINSV.NS	2%	HDFCBANK.NS	2%	KOTAKBANK.NS	2%	SBILIFE.NS	2%	UPL.NS	2%
BHARTIARTL.NS	2%	HDFCLIFE.NS	2%	LT.NS	3%	SBIN.NS	3%	WIPRO.NS	2%

Table 4.5 Strategy 4 Portfolio Allocation of all NIFTY 50 Stocks

The Table 4.5 demonstrates the ability of Strategy 4 to analyze the historical data of NIFTY 50 stocks and construct a portfolio that provides an allocation for long term investments. Strategy 4 considers all NIFTY 50 stocks since there are no limits set on the number of stocks to be chosen in this strategy. The returns of this strategy will be discussed in the Chapter 6.

4.14.5 Strategy5

Strategy 5 is developed with a blend of the following components:

- Base Strategy – UCB1 Stock Allocation Maximizer
- Utility Function – Calculate Daily Strength Lag
- Utility Function – Calculate Allocation

The code for Strategy 5 is defined as follows:

```
class Strategy5:
    def __init__(self, stocks_strength, tickers):
        np.random.seed(1)
        random.seed(1)
        tf.random.set_seed(1)
        self.stocks_strength = stocks_strength
        self.stock_strength_change = calculate_daily_strength_lag(stocks_strength)
        self.tickers = tickers
        self.allocator =
UCB1_StockAllocationMaximizer(self.stock_strength_change,
len(self.stock_strength_change))
    def strategy5(self):
        chosen_stocks_and_allocation_each_day = self.allocator.allocate_stocks()
        last_day_allocations = chosen_stocks_and_allocation_each_day[-1]
```

```
        print(f"Day {len(self.stocks_strength)}: {last_day_allocations}")
        investment_policy = [stock_with_alloc[1] for stock_with_alloc in
last_day_allocations]
        allocation_final = calc_allocation(investment_policy)
        return allocation_final
```

The output of Strategy5 is a new portfolio allocation of NIFTY 50 stocks as follows:

Stocks	Allocations %	Stocks	Allocations %	Stocks	Allocations %	Stocks	Allocations %	Stocks	Allocations %
ADANIENT.NS	2%	BRITANNIA.NS	2%	HEROMOTOCO.NS	2%	LTIM.NS	2%	SUNPHARMA.NS	2%
ADANIPORTS.NS	2%	CIPLA.NS	2%	HINDUNILVR.NS	2%	M&M.NS	2%	TATACONSUM.NS	2%
APOLLOHOSP.NS	2%	COALINDIA.NS	0%	HINDALCO.NS	2%	MARUTI.NS	2%	TATAMOTORS.NS	2%
ASIANPAINT.NS	2%	DIVISLAB.NS	2%	ICICIBANK.NS	2%	NESTLEIND.NS	2%	TATASTEEL.NS	0%
AXISBANK.NS	2%	DRREDDY.NS	2%	INDUSINDBK.NS	3%	NTPC.NS	2%	TCS.NS	2%
BPCL.NS	2%	EICHERMOT.NS	2%	INFY.NS	2%	ONGC.NS	2%	TECHM.NS	2%
BAJAJ-AUTO.NS	2%	GRASIM.NS	2%	ITC.NS	2%	POWERGRID.NS	2%	TITAN.NS	2%
BAJFINANCE.NS	2%	HCLTECH.NS	2%	JSWSTEEL.NS	2%	RELIANCE.NS	2%	ULTRACEMCO.NS	2%
BAJAJFINSV.NS	2%	HDFCBANK.NS	2%	KOTAKBANK.NS	2%	SBILIFE.NS	2%	UPL.NS	2%
BHARTIARTL.NS	2%	HDFCLIFE.NS	2%	LT.NS	2%	SBIN.NS	2%	WIPRO.NS	3%

Table 4.6 Strategy 5 Portfolio Allocation of all NIFTY 50 Stocks

The Table 4.6 demonstrates the ability of Strategy 5 to analyze the historical data of NIFTY 50 stocks and construct a portfolio that provides an allocation for long term investments. Strategy 5 considers all NIFTY 50 stocks since there are no limits set on the number of stocks to be chosen in this strategy. The returns of this strategy will be discussed in the Chapter 6.

4.14.6 Strategy6

Strategy 6 is developed with a blend of the following components:

- Base Strategy – UCB1 Optimizer
- Base Strategy – Policy Gradient Simple Policy Search
- Utility Function – Calculate Daily Strength Lag
- Utility Function – Calculate Allocation

The code for Strategy 6 is defined as follows:

```
        class Strategy6:
```

```
        def __init__(self, stocks_strength):
            np.random.seed(1)
            random.seed(1)
            tf.random.set_seed(1)
            self.stock_strength_change = calculate_daily_strength_lag(stocks_strength)
            self.optimizer = UCB1_Optimizer(self.stock_strength_change,
len(self.stock_strength_change))
        def strategy6(self):
            total_reward = self.optimizer.optimize()
            investment_policy = simple_policy_search(self.stock_strength_change,
total_reward)
            allocation_final = calc_allocation(investment_policy)
            return allocation_final
```

The output of Strategy6 is a new portfolio allocation of NIFTY 50 stocks as follows:

Stocks	Allocations %	Stocks	Allocations %	Stocks	Allocations %	Stocks	Allocations %	Stocks	Allocations %
ADANIENT.NS	1%	BRITANNIA.NS	0%	HEROMOTOCO.NS	9%	LTIM.NS	1%	SUNPHARMA.NS	1%
ADANIPORTS.NS	1%	CIPLA.NS	1%	HINDUNILVR.NS	1%	M&M.NS	1%	TATACONSUM.NS	1%
APOLLOHOSP.NS	1%	COALINDIA.NS	2%	HINDALCO.NS	1%	MARUTI.NS	1%	TATAMOTORS.NS	1%
ASIANPAINT.NS	1%	DIVISLAB.NS	37%	ICICIBANK.NS	1%	NESTLEIND.NS	1%	TATASTEEL.NS	1%
AXISBANK.NS	1%	DRREDDY.NS	1%	INDUSINDBK.NS	4%	NTPC.NS	1%	TCS.NS	1%
BPCL.NS	0%	EICHERMOT.NS	1%	INFY.NS	1%	ONGC.NS	1%	TECHM.NS	1%
BAJAJ-AUTO.NS	1%	GRASIM.NS	1%	ITC.NS	0%	POWERGRID.NS	1%	TITAN.NS	1%
BAJFINANCE.NS	1%	HCLTECH.NS	1%	JSWSTEEL.NS	1%	RELIANCE.NS	1%	ULTRACEMCO.NS	1%
BAJAJFINSV.NS	1%	HDFCBANK.NS	1%	KOTAKBANK.NS	1%	SBILIFE.NS	1%	UPL.NS	1%
BHARTIARTL.NS	1%	HDFCLIFE.NS	1%	LT.NS	1%	SBIN.NS	1%	WIPRO.NS	2%

Table 4.7 Strategy 6 Portfolio Allocation of all NIFTY 50 Stocks

The Table 4.7 demonstrates the ability of Strategy 6 to analyze the historical data of NIFTY 50 stocks and construct a portfolio that provides an allocation for long term investments. Strategy 6 considers all NIFTY 50 stocks since there are no limits set on the number of stocks to be chosen in this strategy. The returns of this strategy will be discussed in the Chapter 6.

4.14.7 Strategy7

Strategy 7 is developed with a blend of the following components:

- Base Strategy – UCB1 Stock Selector
- Base Strategy – Policy Network Batch Norm
- Utility Function – Calculate Daily Strength Lag
- Utility Function – Create Features

The code for Strategy 7 is defined as follows:

```
class Strategy7:
    def __init__(self, stocks_strength, portfolio_size):
        np.random.seed(1)
        random.seed(1)
        tf.random.set_seed(1)
        self.stock_strength_change = calculate_daily_strength_lag(stocks_strength)
        self.selector = UCB1_StockSelector(self.stock_strength_change,
len(self.stock_strength_change), portfolio_size)
        self.portfolio_size = portfolio_size
    def strategy7(self):
        selected_stocks = self.selector.select_stocks()
        selected_stocks_strength = {stock: self.stock_strength_change[stock] for
stock in selected_stocks}
        stock_features = create_features(selected_stocks_strength)
        policy_network = PolicyNetworkBatchNorm(len(selected_stocks))
        model = policy_network.get_model()
        allocations = model.predict(np.array([stock_features.flatten()]))
        allocation_final = allocations.tolist()[0]
```

```
return selected_stocks, allocation_final
```

The output of Strategy7 is a new portfolio allocation of NIFTY 50 stocks as follows:

Stocks	Allocations %
UPL.NS	15%
LTIM.NS	10%
TATASTEEL.NS	13%
HINDUNILVR.NS	20%
ITC.NS	10%
BRITANNIA.NS	14%
BPCL.NS	18%

Table 4.8 Strategy 7 Portfolio Allocation of all NIFTY 50 Stocks

The Table 4.8 demonstrates the ability of Strategy 7 to analyze the historical data of NIFTY 50 stocks and construct a portfolio that provides an allocation for long term investments. Strategy 7 considers only the top 7 stocks since there are limits set on the number of stocks to be chosen in this strategy. The returns of this strategy will be discussed in the Chapter 6.

4.14.8 Strategy8

Strategy 8 is developed with a blend of the following components:

- Base Strategy – UCB1 Optimizer
- Base Strategy – Policy Network Gradient Complex Policy Search
- Utility Function – Calculate Daily Strength Lag
- Utility Function – Calculate Allocation

The code for Strategy 8 is defined as follows:

```
class Strategy8:
    def __init__(self, stocks_strength, num_stocks):
        np.random.seed(1)
        random.seed(1)
        tf.random.set_seed(1)
        self.stocks_strength = stocks_strength
```

```
            self.stock_strength_change =
calculate_daily_strength_lag(self.stocks_strength)
            self.num_stocks = num_stocks
            self.optimizer = UCB1_Optimizer(self.stock_strength_change,
len(self.stock_strength_change))
        def strategy8(self):
            total_reward = self.optimizer.optimize()
            investment_policy = complex_policy_search(self.stocks_strength,
total_reward, self.num_stocks)
            ucb_values = investment_policy[0][0]
            allocation_final = calc_allocation(ucb_values)
            return allocation_final
```

The output of Strategy8 is a new portfolio allocation of NIFTY 50 stocks as follows:

Stocks	Allocations %	Stocks	Allocations %	Stocks	Allocations %	Stocks	Allocations %	Stocks	Allocations %
ADANIENT.NS	2%	BRITANNIA.NS	2%	HEROMOTOCO.NS	2%	LTIM.NS	2%	SUNPHARMA.NS	1%
ADANIPORTS.NS	2%	CIPLA.NS	1%	HINDUNILVR.NS	3%	M&M.NS	2%	TATACONSUM.NS	3%
APOLLOHOSP.NS	2%	COALINDIA.NS	0%	HINDALCO.NS	2%	MARUTI.NS	1%	TATAMOTORS.NS	1%
ASIANPAINT.NS	5%	DIVISLAB.NS	0%	ICICIBANK.NS	3%	NESTLEIND.NS	1%	TATASTEEL.NS	3%
AXISBANK.NS	1%	DRREDDY.NS	3%	INDUSINDBK.NS	1%	NTPC.NS	1%	TCS.NS	2%
BPCL.NS	3%	EICHERMOT.NS	2%	INFY.NS	2%	ONGC.NS	3%	TECHM.NS	3%
BAJAJ-AUTO.NS	2%	GRASIM.NS	1%	ITC.NS	3%	POWERGRID.NS	2%	TITAN.NS	3%
BAJFINANCE.NS	2%	HCLTECH.NS	4%	JSWSTEEL.NS	2%	RELIANCE.NS	1%	ULTRACEMCO.NS	2%
BAJAJFINSV.NS	2%	HDFCBANK.NS	3%	KOTAKBANK.NS	2%	SBILIFE.NS	2%	UPL.NS	2%
BHARTIARTL.NS	2%	HDFCLIFE.NS	1%	LT.NS	4%	SBIN.NS	1%	WIPRO.NS	3%

Table 4.9 Strategy 8 Portfolio Allocation of all NIFTY 50 Stocks

The Table 4.9 demonstrates the ability of Strategy 8 to analyze the historical data of NIFTY 50 stocks and construct a portfolio that provides an allocation for long term investments. Strategy 8 considers only the top 10 stocks since there are limits set on the number of stocks to be chosen in this strategy. The returns of this strategy will be discussed in the Chapter 6.

4.14 Allocation – UI Module

The final module of the SLRPO strategy is the Allocation UI module which is the front-end interface that is used to run all 8 strategies and review the portfolio allocations of each strategy, download the results and take informed decisions based on the portfolio allocations and their historical returns.

The complete SLRPO Strategy is developed in Python and the Front-end application to run the Strategy is developed using Streamlit in Python.

The code for configuration of the input variables required to run the app is defined as follows:

```
config_file = 'config.ini'
tickers = read_config(config_file)
stocks_strength = pd.read_csv('df_scores.csv')
stocks_strength.index = stocks_strength['Date']
stocks_strength.drop(columns = 'Date', inplace = True)
num_rounds = len(stocks_strength)
num_stocks = len(tickers)
stocks_data = pd.read_csv('stocks_data.csv')
start_date = '01-01-2020'
end_date = '30-01-2024'
Investment = 100000
```

The code for executing all the 8 strategies are defined as follows:

```
strategy1 = Strategy1(tickers)
allocation1 = strategy1.strategy1()
strategy2 = Strategy2(num_stocks, num_rounds, tickers, stocks_strength)
top_stocks2, allocation2 = strategy2.strategy2()
strategy3 = Strategy3(stocks_strength, num_stocks, portfolio_size = 10)
```

```
selected_stocks3, allocation3 = strategy3.strategy3()
strategy4 = Strategy4(num_stocks, stocks_strength, tickers)
allocation4 = strategy4.strategy4()
strategy5 = Strategy5(stocks_strength, tickers)
allocation5 = strategy5.strategy5()
strategy6 = Strategy6(stocks_strength)
allocation6 = strategy6.strategy6()
strategy7 = Strategy7(stocks_strength, portfolio_size=7)
selected_stocks7, allocation7 = strategy7.strategy7()
strategy8 = Strategy8(stocks_strength, num_stocks)
allocation8 = strategy8.strategy8()
```

The code for downloading the results of the strategies is defined as follows:

```
def create_download_df(stocks, allocations):
    if len(stocks) != len(allocations):
        raise ValueError("Stocks and allocations lists must be of the same length.")
    df = pd.DataFrame({
        'Stocks': stocks,
        'Allocations %': allocations
    })
    return df
configurations = [
    {"title": "strategy 1", "tickers": tickers, "allocation": allocation1},
    {"title": "strategy 2", "tickers": top_stocks2, "allocation": allocation2},
    {"title": "strategy 3", "tickers": selected_stocks3, "allocation": allocation3},
    {"title": "strategy 4", "tickers": tickers, "allocation": allocation4},
```

```
        {"title": "strategy 5", "tickers": tickers, "allocation": allocation5},
        {"title": "strategy 6", "tickers": tickers, "allocation": allocation6},
        {"title": "strategy 7", "tickers": selected_stocks7, "allocation": allocation7},
        {"title": "strategy 8", "tickers": tickers, "allocation": allocation8}
    ]
```

The code for the front-end user interface is defined as follows:

```
    def main():
        st.title('Investment Strategy Recommendations for NIFTY 50')
        for config in configurations:
            st.subheader(f'Results for {config["title"]}')
            df = calc_model_returns(stocks_data, config['tickers'], config['allocation'],
start_date, end_date, Investment)
            df_strategy = create_download_df(config['tickers'], config['allocation'])
            st.dataframe(df)
            csv = df_strategy.to_csv(index=False)
            b64 = base64.b64encode(csv.encode()).decode()
            file_name = f"{config['title']}_data.csv"
            link_text = f"Download {config['title']}"
            link = f'<a href="data:file/csv;base64,{b64}"
download="{file_name}">{link_text}</a>'
            st.markdown(link, unsafe_allow_html=True)
    if __name__ == '__main__':
        main()
```

The results of the strategies and its corresponding returns can be analyzed and downloaded by running the UI. The output of the UI is generated as follows and the

result of each strategy is captured in the UI along with an option to download the Portfolio allocations for the strategy.

The results for Strategy 1 are represented in the UI as follows:

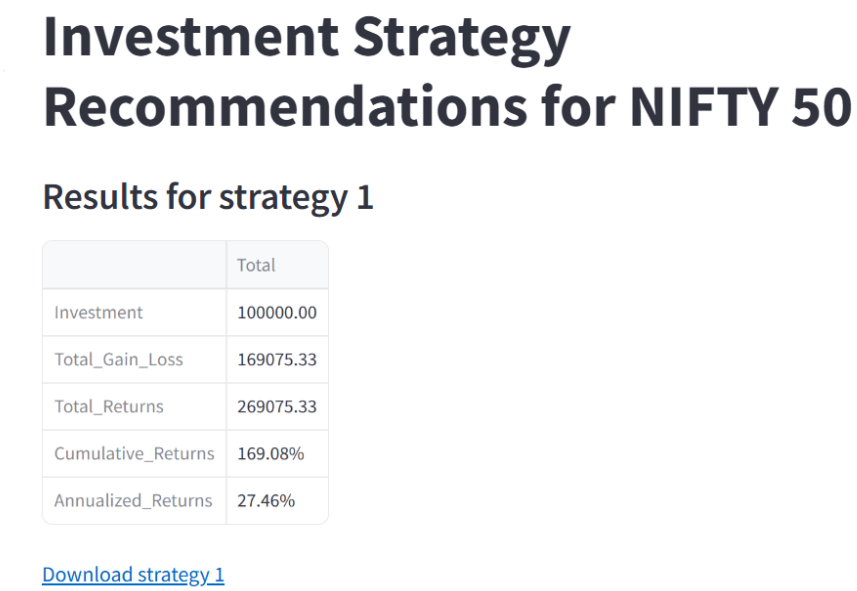

Figure 4.5 Strategy 1 - Results

The results for Strategy 2 are represented in the UI as follows:

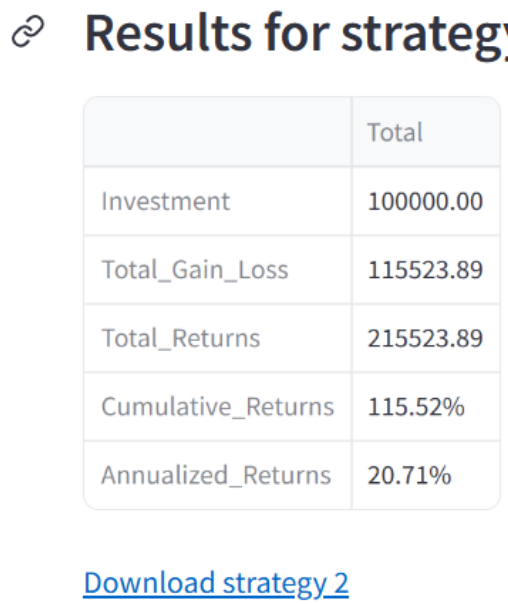

Figure 4.6 Strategy 2 – Results

The results for Strategy 3 are represented in the UI as follows:

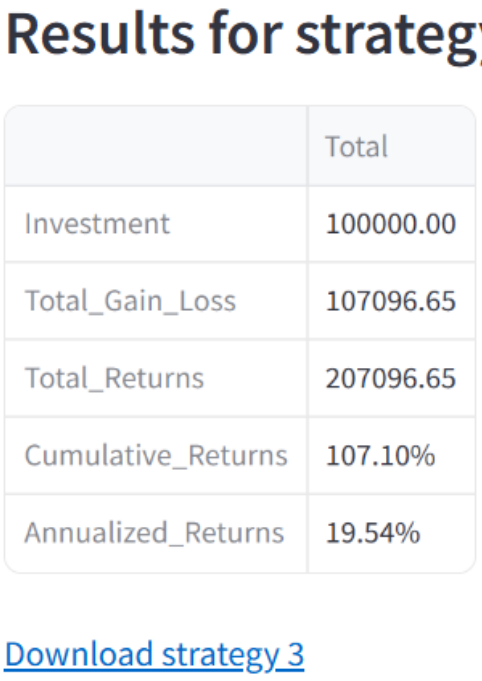

Figure 4.7 Strategy 3 – Results

The results for Strategy 4 are represented in the UI as follows:

Figure 4.8 Strategy 4 – Results

The results for Strategy 5 are represented in the UI as follows:

Results for strategy 5

	Total
Investment	100000.00
Total_Gain_Loss	156562.61
Total_Returns	256562.61
Cumulative_Returns	156.56%
Annualized_Returns	25.98%

Download strategy 5

Figure 4.9 Strategy 5 – Results

The results for Strategy 6 are represented in the UI as follows:

Results for strategy 6

	Total
Investment	100000.00
Total_Gain_Loss	137566.43
Total_Returns	237566.43
Cumulative_Returns	137.57%
Annualized_Returns	23.63%

Download strategy 6

Figure 4.10 Strategy 6 – Results

The results for Strategy 7 are represented in the UI as follows:

Results for strategy 7

	Total
Investment	100000.00
Total_Gain_Loss	134708.92
Total_Returns	234708.93
Cumulative_Returns	134.71%
Annualized_Returns	23.26%

Download strategy 7

Figure 4.10 Strategy 7 – Results

The results for Strategy 8 are represented in the UI as follows:

Results for strategy 8

	Total
Investment	100000.00
Total_Gain_Loss	170569.52
Total_Returns	270569.52
Cumulative_Returns	170.57%
Annualized_Returns	27.63%

Download strategy 8

Figure 4.11 Strategy 8 – Results

The results for all the strategies are captured in the Allocation UI and will be analyzed further in the Chapter 6.

CHAPTER V:

RESULTS

5.1 Research Question

How can data science techniques and methodologies help in developing a decision-making framework that helps in identifying the opportunities to invest in the Indian stock markets?

The complete research and study performed in this thesis led to the development of a new Investment strategy names SLRPO strategy. The development of this strategy involved various stages of Data Science life cycle that includes data collection, data preprocessing, exploratory data analysis, predictive modeling and reinforcement learning.

All these techniques of data science are used to develop SLRPO strategy with the following 11 modules:

1. Data Acquisition
2. Data Load
3. Indicators
4. Preprocessing
5. Weight Generator
6. Strength Score
7. Utility Functions
8. Strategy – UCB1
9. Strategy – Policy Network
10. Strategy – Policy Gradient
11. Allocation – UI

The complete architecture of SLRPO Strategy is available in Figure 4.1 SLRPO Architecture.

- Each module in the SLRPO Architecture is responsible for one or more of the data science techniques to create the SLRPO Investment Framework.
- Data Acquisition and Data Load modules are responsible for Data collection process in the Data Science Life Cycle.
- Preprocessing module is responsible for Data preprocessing and cleansing in the Data Science Life Cycle.
- Indicators, Weight Generator and Strength Score modules are responsible for Feature Engineering in the Data Science Life Cycle.
- Utility Functions, Strategy - UCB1, Strategy – Policy Network and Strategy – Policy Gradient modules are responsible for Model Development and Evaluation in the Data Science Life Cycle.
- Allocation UI is responsible for results review and analysis of the strategies and their model results.

5.2 Research Sub Question One

What are the key aspects/variables that need to be considered in understanding opportunities?

Understanding opportunities in the Indian Stock markets involves understanding the financial and economic data of the stock markets. For this analysis and framework development, 23 Technical indicators and 13 Fundamental indicators are considered to understand the market sentiment, economic impact, company performance and the share price. These concepts are covered in detailed under the Chapter 4, Section 4.4 along with explanation.

A new indicator named '**Strength Score**' is developed in this thesis to get one common feature derived from all of the Technical and Fundamental indicators considered for this study such that the feature can exhibit the conslidated impact on the hisorical data

of NIFTY 50 stocks. This feature can be applied to any other portfolio of stocks in the markets in general. This concept is covered in detailed under the Chapter 4, Section 4.6 and 4.7 along with explanation.

5.3 Research Sub Question Two

What are the various techniques or algorithms that can be explored to identify opportunities?

Reinforcement Learning and Neural Networks are explored extensively to develop SLRPO Strategy.

The concepts behind Upper Confidence Bound (UCB1) algorithm, Policy Network, Policy Gradient and Neural Networks are used to develop the 8 strategies that generates portfolios and allocations in the SLRPO strategy.

The techniques are covered in detailed under the Chapter 4, Section 4.9 along with explanation.

5.4 Research Sub Question Three

How can prescriptive analytics help in developing a decision-making framework in the domain of Indian stock markets?

The complete design of SLRPO strategy in Chapter 4 using various techniques of data science has led to developing 8 strategies that generates 1 portfolio each along with the allocation of investments in each portfolio and the insights on the portfolio such as its total gain, total loss, total returns, cumulative returns and annualized returns for an investment of Rs. 100000/- over a period of 4 years starting from 1st Jan 2020 to 30th Jan 2024. These metrics and the portfolio outcome provides insights on which portfolio to choose for long term investments. The portfolio allocations from each strategy is different and the number of stocks chosen also differs in the strategies. The choice of

strategy depends on the invester risk profile and the opportunity cost involved in investing in one profile over the other.

The comparison of results and the actionable insights from the prescriptive analysis of SLRPO strategy will be discussed further in Chapter 6.

CHAPTER VI:

DISCUSSION

6.1 Discussion of Results of SLRPO Investment Framework

The SLRPO Investment Framework is made up of 8 strategies. The following aspects of all 8 strategies will be reviewed and discussed in this section.

- No. of Year of investment
- Investment
- Total Gain/ Total Loss
- Total Returns
- Cumulative Returns
- Annualized Returns

All the metrics for the strategies are represented in Table 6.1

Strategy	No. of Years	Investment (INR)	Total Gain / Loss	Total Returns	Cumulative Returns	Annualized Returns	Portfolio Size
1	4	100000	1,69,075.33	2,69,075.33	169.08%	27.46%	30
2	4	100000	1,15,523.89	2,15,523.89	115.52%	20.71%	7
3	4	100000	1,07,096.65	2,07,096.65	107.10%	19.54%	10
4	4	100000	1,67,356.66	2,67,356.66	167.36%	27.26%	30
5	4	100000	1,56,562.61	2,56,562.61	156.56%	25.98%	30
6	4	100000	1,37,566.43	2,37,566.43	137.57%	23.63%	30
7	4	100000	1,34,708.92	2,34,708.93	134.71%	23.26%	7
8	4	100000	1,70,569.52	2,70,569.52	170.57%	27.63%	30

Table 6.1 Returns and Strategy Metrics

The Portfolio allocation recommended by Strategy 8 (Table 4.9) has the maximum annualized returns in all the 8 SLRPO Strategies. The other best strategies being Strategy 1 (Table 4.2) and Strategy 4 (Table 4.5) respectively. It is to be noted that all these three strategies have considered all 50 NIFTY 50 stocks. Hence the risk is distributed across the portfolio and is well balance.

On the other hand, if the investors want to choose a smaller portfolio with better annualized returns, the other best options will be Strategy 7 (Table 4.8) , Strategy 2 (Table 4.3) and Strategy 3 (Table 4.4) respectively.

Industry Benchmark Analysis

Fixed Deposit

Fixed Deposit of money in an bank is the simplest, risk free and a goto form of investment for most of the Indian investors. According to Groww (Groww, 2024), the interest on fixed deposit across various banks in India ranges between 3% to 7% for an investment period between 7 days to 10 years.

NIFTY 50 Index

An industry benchmark of annualized returns is considered from NSE for the NIFTY 50 stocks. According to NSE (National Stock Exchange of India, 2022), NIFTY 50 Index has given an annualized returns of 14.2% since June 1999. The annualized returns for 5 years was 17.6% and 3 years was 18.2%.

The goal of the SLRPO strategy is to give better portfolio distribution of the NIFTY 50 stocks and a better performance and annualized returns compared to industry benchmark.

6.2 Summary of Findings

Opportunity Cost Analysis

Opportunity cost is the cost incurred in chosing one form of investment over another.

Considering the annualized returns of SLRPO framework as the best form of investment. Opportunity cost will be the cost incurred in foregoing SLRPO investment strategies for any other known form of investments.

The following Table shows the opportunity cost of not choosing SLRPO strategy for investment.

Strategy	No. of Years	Investment (INR)	Annualized Returns	Opportunity Cost (Fixed Deposit at maximum 7% interest)	Opportunity Cost (NIFTY 50 Index Benchmark at maximum 18% Returns)	Opportunity Cost INR (Fixed Deposit)	Opportunity Cost INR (NIFTY 50 INDEX)
1	4	100000	27.46%	20.46%	9.46%	20460	9460
2	4	100000	20.71%	13.71%	2.71%	13710	2710
3	4	100000	19.54%	12.54%	1.54%	12540	1540
4	4	100000	27.26%	20.26%	9.26%	20260	9260
5	4	100000	25.98%	18.98%	7.98%	18980	7980
6	4	100000	23.63%	16.63%	5.63%	16630	5630
7	4	100000	23.26%	16.26%	5.26%	16260	5260
8	4	100000	27.63%	20.63%	9.63%	20630	9630
			Average Opportunity Cost	**17.4%**	**6.4%**	**17434**	**6434**

Table 6.2 Opportunity Cost of Foregoing SLRPO

The Table 6.2 demonstrates the impact of foregoing SLRPO for other investment strategies.

On an average, if none of the SLRPO strategies are chosen but a Fixed Deposit is chosen, the opportunity cost would be around 17.4% of Annualized returns and Rs.17434 in INR per year.

On an average, if none of the SLRPO strategies are chosen but NIFTY 50 Index benchmark is chosen, the opportunity cost would be around 6.4% of Annualized returns and Rs.6434 in INR per year.

Considering most of the Indian investors choose Fixed Deposit as the safest and risk free form of investment, the overall opportunity cost is significantly higher.

CHAPTER VII:

SUMMARY, IMPLICATIONS, AND RECOMMENDATIONS

7.1 Summary

Informed decision making in the Indian stock markets is possible with the right kind of data analysis, applications of data science techniques and domain knowledge to understand the markets and the data. SLRPO is an investment strategy that is developed in this thesis by considering all of these aspects of Indian Stock markets.

SLRPO strategy is designed by applying various techniques of data science, machine learning, deep learning and portfolio optimization on Indian Stock markets.

Opportunity cost of investing in Industry Benchmarks vs SLRPO Strategy was studied and analyzed in this research.

From the overall analysis of the results in Chapters 5 and 6, it is evident that a beneficial framework and strategy can be developed when data is analyzed with the help of right tools.

7.2 Implications

The SLRPO investment strategies developed in this thesis is global in nature and its applications can be extended to different stock markets, different set of stocks and different portfolios not only in the Indian market but also in the Global markets.

The prescriptive nature of the SLRPO strategy provides required information to make informed investment decisions. It is important to choose the portfolios according to the individual risk profiles, understand the nature of stocks that are provided as input to the strategies, understand fundamentals of the comanpanies chosen for investment and make informed decisions.

7.3 Recommendations for Future Research

The opportunity for future research lies in the choice of data science techniques, choice of feature engineering and the choice of input data in building similar investment strategies.

Reinforcement learning specifically focused on Upper Confidence Bound, Policy Network, Policy Search and Policy Gradient were the techniques chosen for this research. There are opportuities to expland further with other aspect of reinforcement learning and other machine learning algorithms.

The focus of this research was to provide actionable insights through prescriptive analyics. There are opportunities to do further research using predictive models and regression techiques. There are also opportunities to build recommendations using Generative AI and Large Language Models.

7.4 Conclusion

This research started with an extensive literature review of around 615 research papers in the domain of Indian stock markets and data science. Throughout the review, there were hundreds of techniques used to understand the markets and predict future stock prices for individual stocks while there was minimal research in developing portfolio recommendations and actionable insights. The SLRPO Investment strategy and the Stocks Stength Score developed in this research will act as important tools to provide actionable insights, portfolio recommendations and opportunity cost as well as opportunity analysis in Indian stocks markets for an informed decision making. The novel approach taken in developing SLRPO strategy using multiple combinations of Reinforcement learning techniques will pave a way for a new direction of analysis that can be performed on the stock markets. The SLRPO strategy provides actionable insights to an investor for informed decision making along with information that helps the

investor to choose between one investment vs another for better outcomes from an investment.

REFERENCES

Appel, G. (2005) “Technical Analysis: Power Tools for Active Investors,” *New York: Financial Times Press.*

Arms, R.W.Jr. (1996) “Trading Without Fear: Eliminating the Human Emotion,” *New York: John Wiley & Sons*.

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). “Finite-time analysis of the multiarmed bandit problem,” *Machine Learning*, 47(2-3), 235-256.

Bollinger, J. (2002) “Bollinger on Bollinger Bands, New York,” *McGraw-Hill*.

Buchanan, J.M. (1991) “Opportunity Cost,” *The World of Economics*, pp. 520–525.

Carta, S. et al. (2021) “Multi-DQN: An ensemble of deep Q-learning agents for stock market forecasting,” *Expert Systems with Applications*, 164, p. 113820.

Chaikin, M. (n.d.) “Accumulation/Distribution Line,” *FM Labs*. Available at: https://www.fmlabs.com/reference/ (Accessed: 2 April 2023)

Chaikin, M. (no date) “Chaikin Money Flow,” *Chaikin Analytics*. Available at: What is Chaikin Money Flow? - Chaikin Analytics (Accessed: 23 June 2023).

Chong, E., Han, C. and Park, F.C. (2017) “Deep Learning Networks for Stock Market Analysis and prediction: Methodology, Data Representations, and case studies,” *Expert Systems with Applications*, 83, pp. 187–205.

Colby, R.W. (2003) “The Encyclopedia of Technical Market Indicators. 2nd edn,” *New York: McGraw-Hill.*

Dang, H. and Mei, B. (2022) “Stock Movement Prediction Using Price Factor and Deep Learning,” *International Journal of Computer and Information Engineering*, 16(3), pp. 73–76.

Fernando, J. (2021) “Opportunity Cost,” *https://www.investopedia.com/*. Available at: https://www.investopedia.com/terms/o/opportunitycost.asp (Accessed: March 19,

2022).

Granville, J.E. (1963) “Granville's New Key to Stock Market Profits,” *Englewood Cliffs, NJ: Prentice-Hall.*

Groww. (2024) “Fixed Deposit Interest Rates 2024,” *Groww*. Available at: https://groww.in/fixed-deposits/fd-interest-rates (Accessed: 1 July 2024).

Hoskin, R.E. (1983) “Opportunity cost and behavior,” *Journal of Accounting Research,* 21(1), p. 78.

Investopedia (2023) “Donchian Channels,” *Investopedia*. Available at: https://www.investopedia.com/terms/d/donchianchannels.asp (Accessed: August 4, 2023).

Investopedia (no date) “Technical Indicators,” *Investopedia*. Available at: https://www.investopedia.com/terms/t/technicalindicator.asp (Accessed: August 2, 2023).

Johnson, B. (2010) “Algorithmic Trading and DMA: An Introduction to Direct Access Trading Strategies,” *4Myeloma Press.*

Kamara, A.F., Chen, E. and Pan, Z. (2022) “An ensemble of a boosted hybrid of deep learning models and technical analysis for forecasting stock prices,” *Information Sciences*, 594, pp. 1–19.

Keltner, C.W. (1960) “How to Make Money in Commodities,” *New York: Prentice Hall.*

Konstantinov, G., Chorus, A. and Rebmann, J. (2020) “A network and machine learning approach to factor, asset, and blended allocation,” *The Journal of Portfolio Management*, 46(6), pp. 54–71.

Lane, G.C. (1984) “Lane's Stochastics: The Essential Toolkit for Stock and Futures Traders,” *Technical Analysis of Stocks & Commodities,* September.

Mousavi Anzahaei, S.M. and Nikoomaram, H. (2022) “A comparative study of the performance of Stock trading strategies based on LGBM and CatBoost algorithms.,” *International Journal of Finance & Managerial Accounting*, 7(26), pp. 63–75.

Murphy, J.J. (1999) “Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications,” *New York: New York Institute of Finance.*

National Stock Exchange of India (2022) “25 Years Journey of Nifty50,” *NSE India*. Available at: https://archives.nseindia.com/content/indices/25_Years_Journey_of_Nifty50_2022-01.pdf (Accessed: 1 July 2024).

Patel, J. et al. (2015) “Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques,” *Expert Systems with Applications*, 42(1), pp. 259–268.

Payne, J.W., Bettman, J.R. and Luce, M.F. (1996) “When Time is money: Decision behavior under opportunity-cost time pressure,” *Organizational Behavior and Human Decision Processes*, 66(2), pp. 131–152.

Peng, Y. et al. (2021) “Feature selection and deep neural networks for stock price direction forecasting using technical analysis indicators,” *Machine Learning with Applications*, 5, p. 100060.

Saha, S., Gao, J. and Gerlach, R. (2022) “A survey of the application of graph-based approaches in stock market analysis and prediction,” *International Journal of Data Science and Analytics*, pp. 1–15.

Seban, O. (2008) “Tous Les Secrets Des Traders Gagnants,” *Paris: Olivier Seban Publications.*

SEBI (2015) “SEBI Investor Survey 2015,” *Securities and Exchange Board of India*, pp. 1–52.

Sutton, R. S. (1988) “Learning to predict by the methods of temporal differences,” *Machine Learning*, 3(1), 9-44.

Ticknor, J.L. (2013) “A Bayesian Regularized Artificial Neural Network for Stock Market Forecasting,” *Expert Systems with Applications*, 40(14), pp. 5501–5506.

Wilder, J.W. (1978) “New Concepts in Technical Trading Systems. Greensboro, N.C.” *Trend Research*.

Williams, R.J. (1992) “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine Learning*, 8(3-4), 229-256.