STOCK PRICES PREDICTION USING REINFORCEMENT LEARNING ALGORITHMS

STOCK PRICES PREDICTION USING REINFORCEMENT LEARNING ALGORITHMS

A project report submitted in partial
fulfillment of the requirements for the degree of
Master of Science

By

Subash Shankar

University of Colorado Denver, 2020

Master of Science in Computer Science.

ABSTRACT

This project mainly focuses on designing a Reinforcement Learning Agent to predict the right time to Buy, Sell and Hold stocks to make maximum profit. Considering the current unstable economic trends, an automated stock trading mechanism is absolutely essential for investment companies and for people to make accurate decisions to make a profitable return. Primarily stock market predictions are usually done using machine learning strategies but, in this project, we will design a Reinforcement Learning Agent to be more specific an Actor-Critic model to predict stock prices. The implementation starts with an exploratory data analysis of the dataset and in the end a significant increase in profit over time was achieved.

This Project Report is approved for recommendation to the Graduate Committee.

Project Advisor:

Ashis Biswas

_____

[This page should be included ONLY in theses that are copyrighted. If you include this page, change the page numbering for following pages to vi instead of v using the *Insert/Page Numbers* command.]

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 Problem

There is a rapid and uneven fluctuation in the stock market, people and other investment companies need a better solution to make good profits. Since the introduction of Machine Learning Algorithms, people/companies are starting to come up with better solutions (i.e.) an algorithmic fail proof solution for stock trading. Using these models will produce a faster and more accurate analysis of stocks before investing. These machine learning models/approaches have then evolved since its introduction and have shown promising increase in efficiency and results over time. In recent researches, a combination of Reinforcement learning with time-series modeling has produced more accurate results for stock market prediction.

1.2 Project Statement

Stock market analysis using Actor-Critic Reinforcement Learning and Time-Series modeling to predict the best moves whether to Buy, Sell or Hold based on the stock prices to maximize profits.

1.3 Approach

This project is divided into,

1. Choose a stock dataset.
2. Exploratory Data Analysis on the stock dataset to better understand the data.
3. Splitting the dataset into Train (80 %) and Test (20 %) data.
4. Designing a Reinforcement Learning agent (Actor-Critic model) to predict the best actions (Buy, Sell and Hold).

5. Pass the dataset through the Actor-Critic model to train and test the dataset, to predict the profit which is the evaluation metric for this project.

1.4 Organization of this Project Report

Chapter 2 covers the background knowledge necessary to understand the key concepts of Reinforcement Learning. Chapter 3 covers the architecture, flow and implementation of the project. Chapter 4 covers methodologies, evaluation metric and results of the project. Chapter 5 includes conclusion and future work for this project.

2. BACKGROUND

2.1 Key Concepts

Before we head into the project, it is essential that we understand some key concepts of Reinforcement Learning like Markov Decision Process, Bellman's equation, Q-Learning, Reward function, Policy Gradient, Deep Q-Learning, Actor-Critic Algorithm.

2.1.1 Markov Decision Process

Markov Decision process is generally used to describe an environment for Reinforcement learning. A Markov Decision Process is a tuple (S, A, P, R, γ) S is a finite set of states[1]. A is a finite set of actions, P is a state transition probability matrix P a ss0 = P[S_{t+1} = s 0 |S_t = s, A_t = a], R is a reward function, Ra s = E[R_{t+1}|S_t = s, A_t = a] and γ is a discount factor, γ ∈ [0, 1].

2.1.2 Bellman's Equation

Bellman's equation is a guiding principle to design Reinforcement Learning Algorithms. Bellman's equation can be represented as,

$$v(s) = E[R_{t+1} + \gamma v(S_{t+1})|S_t = s]$$

it can be decomposed into immediate reward (R[t+1]) plus the value of successor state (v[S (t+1)]) with a discount factor (γ)

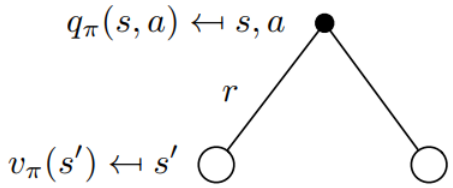


Figure 1: Backup Diagram for State-Value Function

2.1.3 Q-Learning

Q-Learning is a value-based Reinforcement Learning Algorithm to find the best action given the current state. It uses q-values to improve the behavior of the learning agent. By definition, Q-values are defined for states and actions Q(S, A), the best action A to take given the current state S. This estimation will be computed by TD-Update rule,

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q(S',A') - Q(S,A)\right)$$

2.1.4 Reward Function

A reward can be described as a value or feedback or signal to indicate how well the Reinforcement Learning agent is performing. Consider an example of a mouse trying to find cheese in an environment filled with cheese and traps. The mouse would get a positive reward every time it finds the cheese and a negative reward every time it gets trapped. The main goal is to maximize the value of the reward function.

2.1.5 Policy Gradient

A policy tells how to act in a particular state. The main objective of Reinforcement Learning is to find a policy that would maximize the rewards or minimize the cost. All finite Markov Processes have at least one optimal policy. So, deriving a policy to directly maximize the rewards is called policy gradient.

2.1.6 Deep Q-Learning

Deep Q-Learning is an off-policy method where we use neural networks to approximate the Q-value function[9]. Similar to supervised learning, here we train the neural network to try to minimize the loss function and randomly sampling the different transactions like state, action and rewards. In other words, we use supervised learning to fit the Q-function.

2.1.6.1 Proximal Policy Optimization (PPO)

Proximal Policy Optimization is an on-policy algorithm used to control policy gradient update to make sure that the new policy is not too different from the old policy. PPO is stable, fast and much simpler to implement. PPO improves stability by discouraging large policy changes.

2.1.6.2 Trust Region Policy Optimization (TRPO)

It is an On-policy Algorithm[2]. To improve performance and stability, TRPO updates policy by taking the largest step possible while enforcing KL divergence on the size of the policy update at each step. Unlike DDPG, TRPO can used for either discrete or continuous action spaces.

2.1.7 Actor-Critic Algorithm

Actor-Critic model combines both a policy gradient and value function. Here, the Actor represents the policy, and the Critic represents the value functions. The actor takes the state of the environment and returns the best action or policy. The critic evaluates the policy/actions and then guides the actor to update its policy gradient.

2.1.7.1 Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient is an off-policy algorithm that concurrently learns a Q-function and a policy. It combines both Q-Learning and policy gradient and uses neural networks as function approximators. It can be understood as deep Q-Learning for continuous action spaces. It uses off-policy data and Bellman's equation to learn the Q-function and uses the Q-function to learn the policy.

2.2 Related Work

2.2.1 Predicting Stock prices using Deep Learning models

In past researches, LSTM and recurrent neural networks have been used to predict stock prices[10]. The available dataset is initially pre-processed, for example min-max scaling and other data preprocessing techniques are used. A comparison of different neural network models like, mode with fewer to a greater number of nodes, model with single to multiple layers is used to train and test the dataset. In the predicted results it is found that LSTM or GRU layers slightly outperform a simple RNN layer since they are able to use more long-term dependencies.

2.2.2 Stock Trading using simple Q-Learning

Other more simpler versions for stock predictions using just Q-Learning have also been implemented. The goal would be to derive a policy that tells the agent what action should be performed under a particular situation. The implementation goes like any other Q-Learning solution where a Q-Table is initialized, an Action is chosen and performed, the reward is

awarded, and we update the Q-Table. The implementation begins with mapping the states to the actions, the once the agent interacts with the environment it obtains the state, action, reward and new state and which action to take is decided. Q(S, A) is updated using Bellman's Approximation.

3. ARCHITECTURE

3.1 High Level Design

Reinforcement Learning is an area of machine learning, it is about taking suitable action to maximize the rewards i.e., learning from positive or negative rewards. It consists of agent which interacts with its environment via actions at discrete time steps and receives a reward.

1. The agent takes an action in an environment.
2. The action is interpreted into a reward R and a representation of the state S.
3. These two are then fed back into the agent.

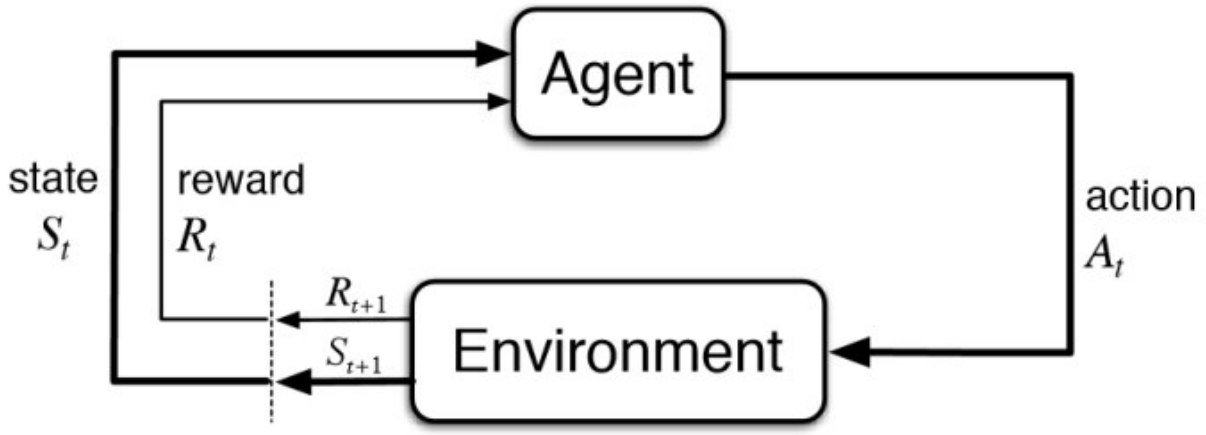


Figure 2: Architecture of Agent Environment feedback loop

Actor-Critic model combines both a policy gradient and value function.

1. The Actor is used to model the policy and Critic is used to model the value function.
2. The Actor takes in the state of the environment and returns the best action
3. The critic then evaluates the action and uses Temporal Difference Learning to model the value function.
4. Now the critic uses this approximation to advise the actor to update its policy.

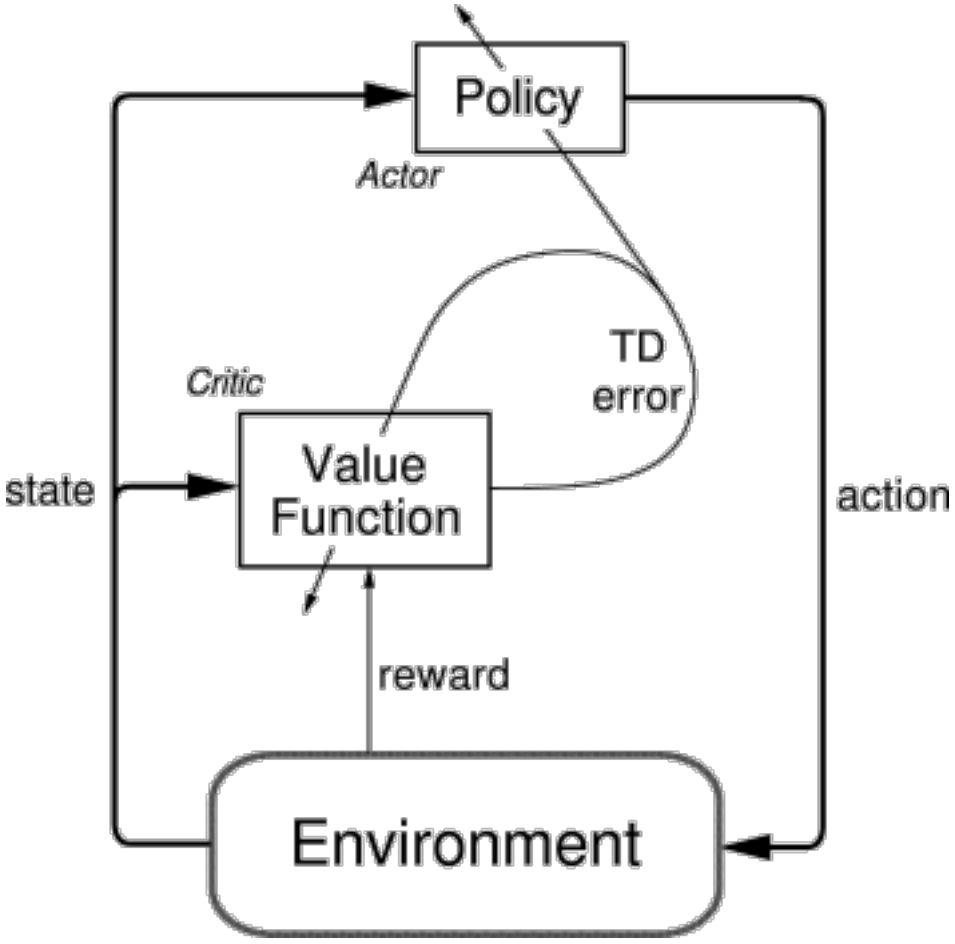


Figure 3: High level Architecture of Actor-Critic Learning Loop.

3.2 Project Flow

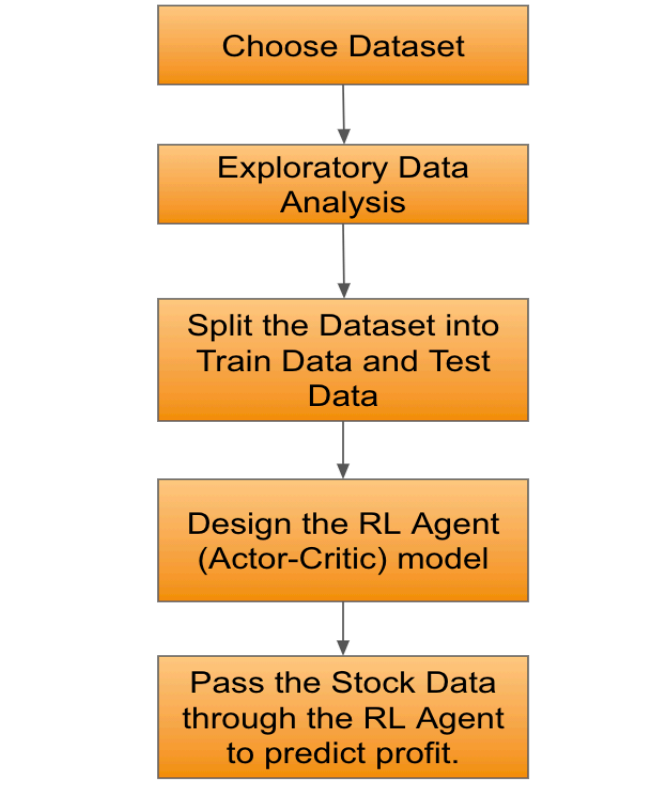


Figure 4: Project Flow

3.3 Implementation

In our problem, the agent is the Trader, state space is stock prices, t – the day you want to predict and window – how many days to go back in time. Our action space is 3 (Buy, Sell or Hold). We implement an Actor-Critic model to predict the stock prices. The Actor-Critic is two interacting neural network models. The Actor is used to model the policy and Critic is used to model the value function. The actor takes the state and returns the best policy or action. The Critic evaluates the policy and computes an approximation to update the Actor. The critic is trained using Temporal Difference Learning Algorithm. Based on this model we get three possible outcomes, Buy, Sell or Hold the stock.

The Actor scripts builds a policy that maps states to action. We will use Keras package to implement the neural network. The Actor layer will predict the Action Probabilities. The Critic script maps states to action (Q-Value) pairs. The final output of the critic layer will give action values (Q-Values). The agent is created, and outputs of Actor-Critic scripts is provided as an input. The policy model obtained from the actor network maps states to actions and instantiates the actor layer and the critic layer instantiates the critic model. Given a state, the actor outputs an action and a probability for each action. We get a policy based on the action probabilities and a set of actions is carried out by the agent through softmax activation since we are dealing with modeling probabilities. Both the networks learn from sampled experiences to predict the actions for the next state. The critic uses Temporal Difference Learning to target the Q-Values, Q_targets = rewards + self.gamma * Q_targets_next * (1 - dones)[4]. We Target the Q_Value and fit it into the critic model. Then, we calculate the action gradient which is like a gradient of Q values with respect to actions from the critic network and use this to train the actor. At each step, the agents decides whether to buy or sell or to hold. When it buys a stock, the buying price is stored and when selling the buying is popped out and subtracted from the selling price to calculate the profit whether positive or negative.

4. METHODOLOGY AND RESULTS

4.1 Methodology

In this project we used Actor-Critic model to predict stock prices and maximize profits. Actor Critic model combines both a policy gradient and value function. The Actor represents the policy, and the Critic represents the value functions.

4.2 Performance Metric

The performance metric is Profit and we can observe a significant and increasing profit over time.

4.3 Results

The main goal of this project is to achieve higher profit. The performance metric is profit. Below are some of the preliminary results of the project.

	Date	Open	High	Low	Close	Adj Close	Volume
0	2000-01-03	1469.250000	1478.000000	1438.359985	1455.219971	1455.219971	931800000
1	2000-01-04	1455.219971	1455.219971	1397.430054	1399.420044	1399.420044	1009000000
2	2000-01-05	1399.420044	1413.270020	1377.680054	1402.109985	1402.109985	1085500000
3	2000-01-06	1402.109985	1411.900024	1392.099976	1403.449951	1403.449951	1092300000
4	2000-01-07	1403.449951	1441.469971	1400.729980	1441.469971	1441.469971	1225200000

Figure 5: Dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5257 entries, 0 to 5256
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       5257 non-null   datetime64[ns]
 1   Open       5257 non-null   float64
 2   High       5257 non-null   float64
 3   Low        5257 non-null   float64
 4   Close      5257 non-null   float64
 5   Adj Close  5257 non-null   float64
 6   Volume     5257 non-null   int64
dtypes: datetime64[ns](1), float64(5), int64(1)
memory usage: 287.6 KB
```

Figure 6: Dataset Info

```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

Figure 7: Null Value count

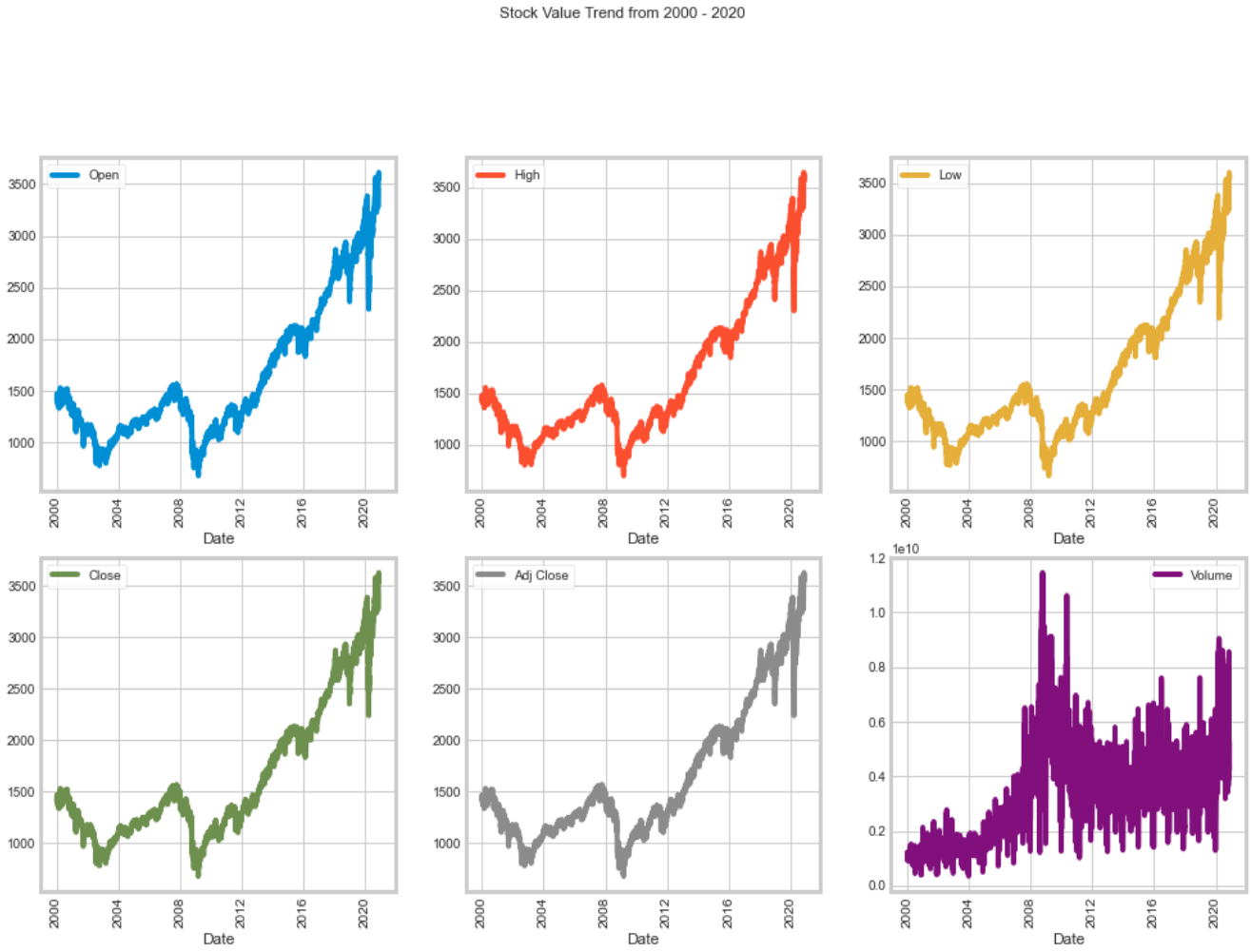
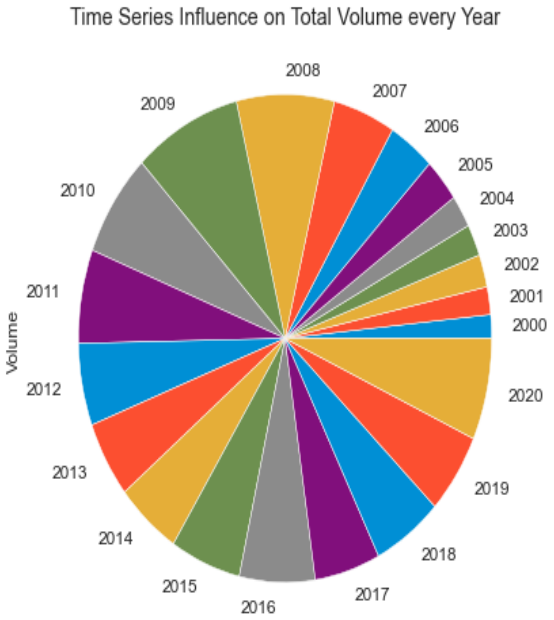


Figure 8: Glimpse of how the market shares varied over time



	Volume
2000	260691260000
2001	305552780000
2002	360114650000
2003	348881650000
2004	358647350000
2005	483815720000
2006	601022760000
2007	810086850000
2008	1273405400000
2009	1404448300000
2010	1151481590000
2011	1035482070000
2012	907919280000
2013	846459590000
2014	845405230000
2015	921218600000
2016	984118470000
2017	856488130000
2018	906717990000
2019	892267400000
2020	1123326990000

Figure 9: Time series influence on volume trade every year; plot to understand the trend better

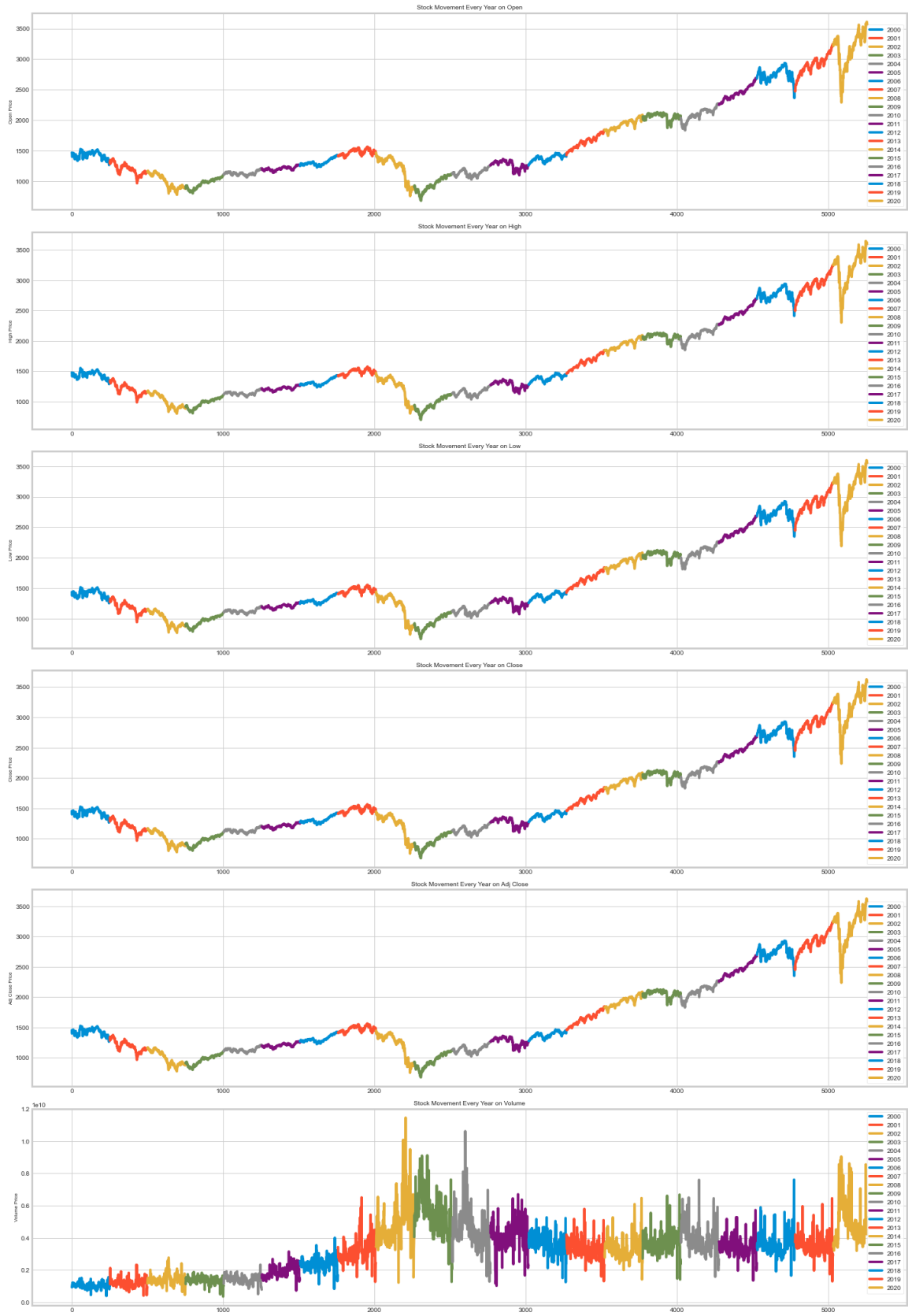


Figure 10: Extensive data analysis on Time-series data to find patterns

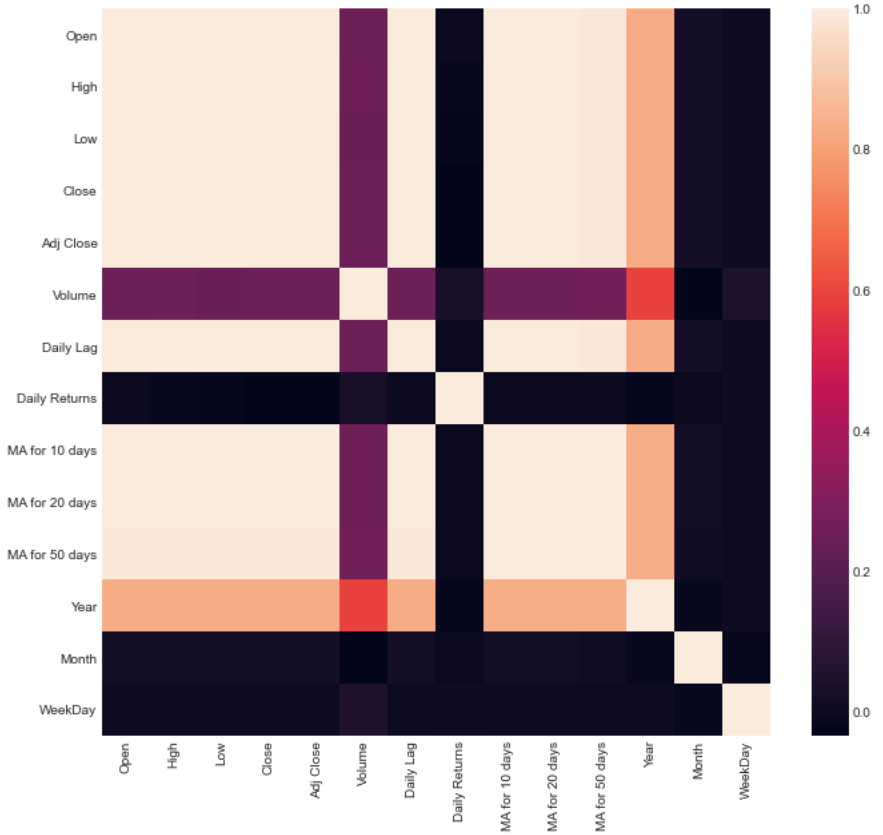
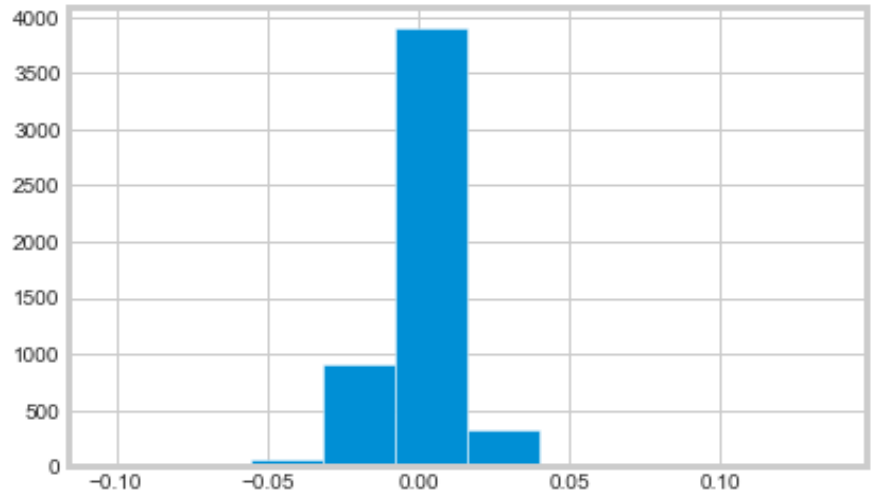


Figure 11: Correlation Analysis



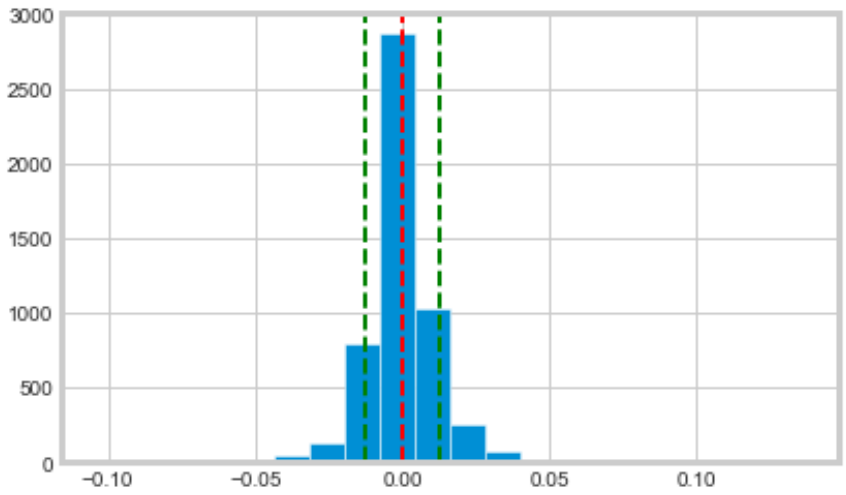


Figure 12,13: Daily Returns

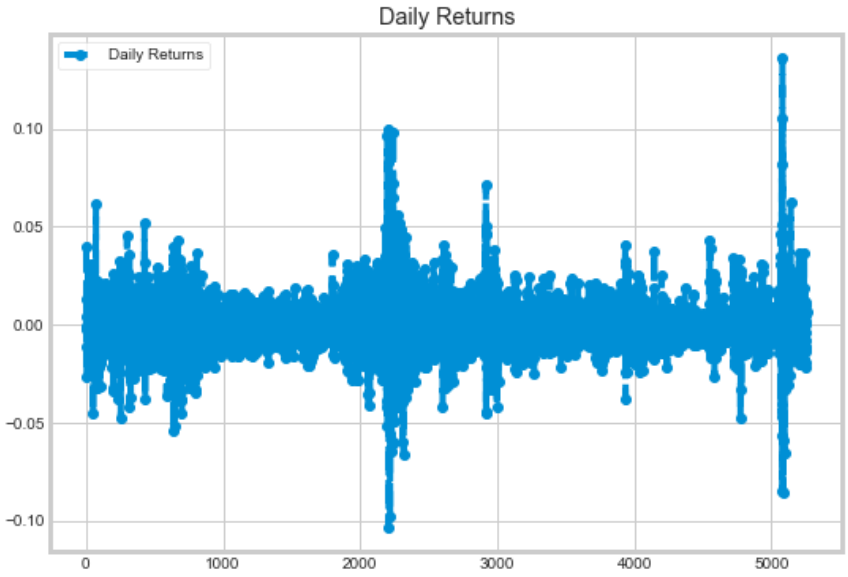


Figure 14: Daily Returns Percentage

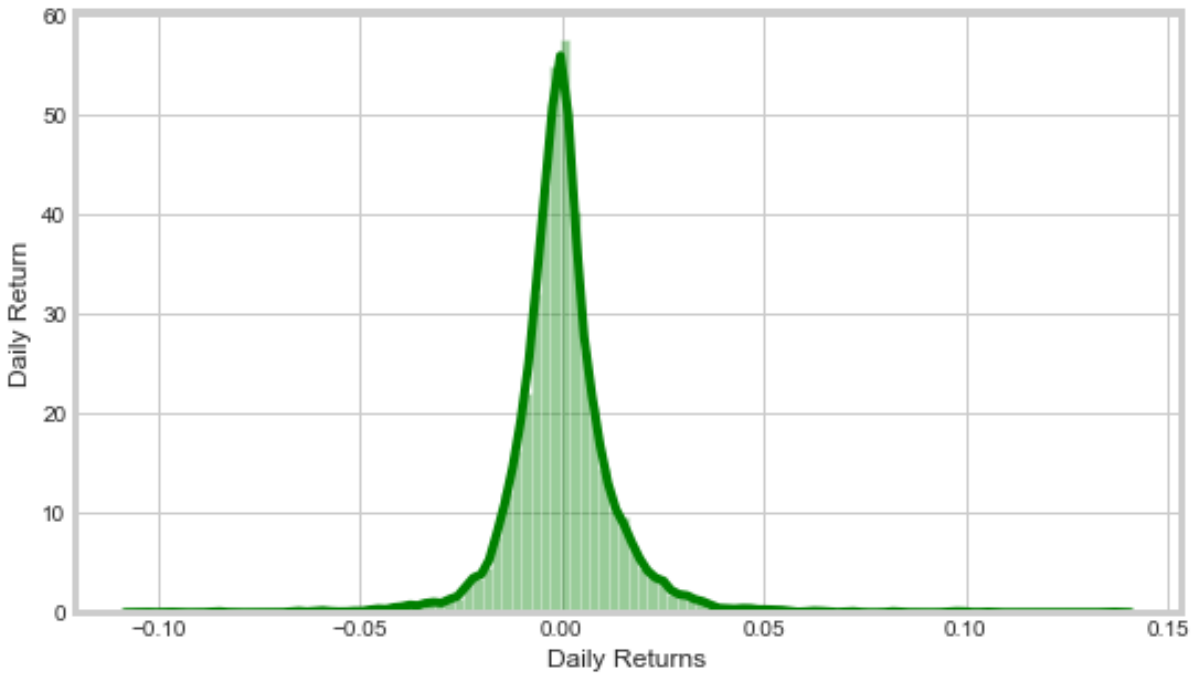
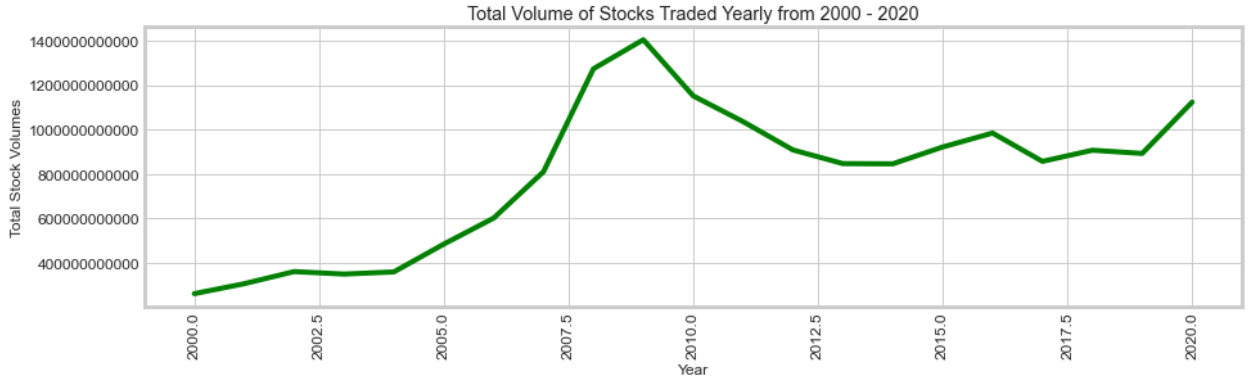


Figure 15: Daily Returns percentage Change; univariate distribution of Stock



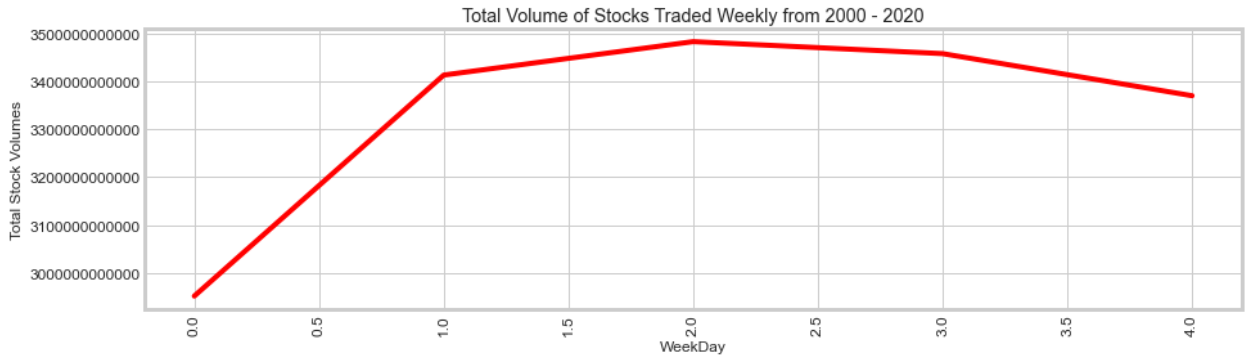


Figure 16: Visualizing total volume of Stock Traded yearly, monthly and weekly

Figure 17: Train and Test Data Split

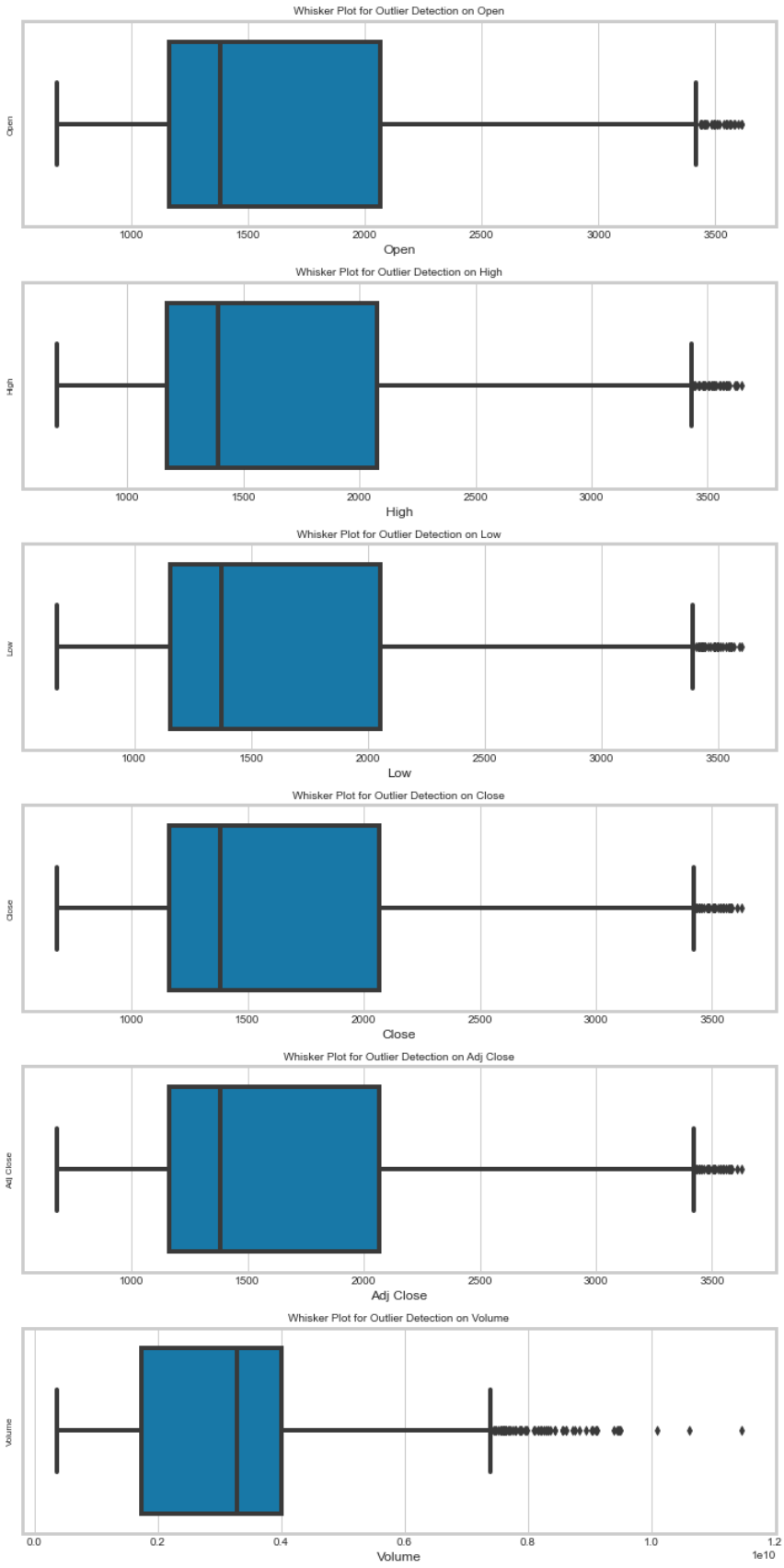


Figure 18: Whisker plots to detect the presence of outlier

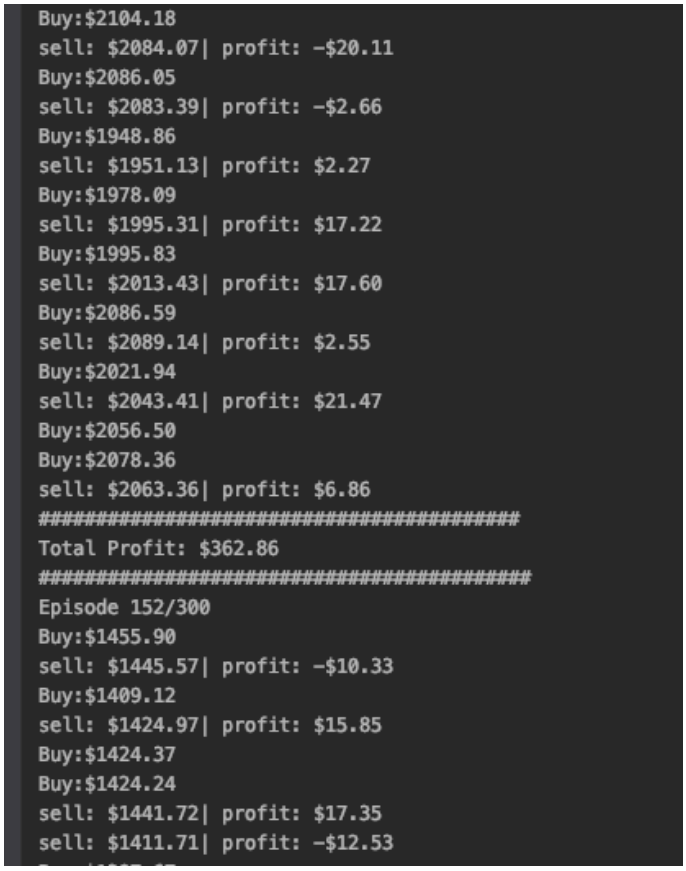


Figure 19: Buy, Sell stocks to see profit.

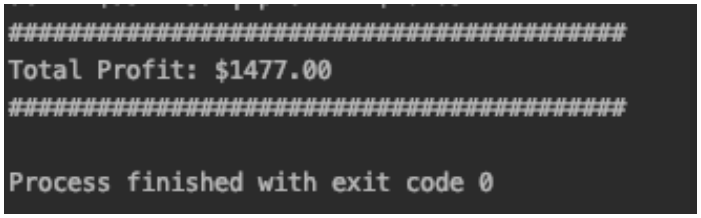


Figure 18: Final Total Profit

5. CONCLUSION

5.1 Summary

In this project we used Reinforcement Learning Algorithm (Actor-Critic Model) to predict stock prices. We saw the different literature readings that are necessary to understand these Reinforcement Learning Algorithms. This model has provided a better insight on when to buy stocks, when the rates are high and selling those stocks now would be more profitable than in the future and when to hold on to a stock i.e., the Actor-Critic model predicted the best action to be performed. The main aim was to maximize profit and we saw an increasing profit over time. In the end a significant (>$10,000) profit was observed. So, in conclusion the Actor-Critic model has proved to show a significant increase in profit, much more stable and efficient than other Reinforcement Learning Algorithms.

5.2 Future Work

Reinforcement Learning and other Machine Learning algorithms are evolving every day. For Stock Prediction using Reinforcement Learning, it would be interesting to impart LSTM networks and see how the agent reacts to it. Other ideas would be Reward Engineering and Stock Environment with multiple agents. Also, using an Actor-Critic model is one of solutions available for stock prediction, in the future a comparative analysis between Actor-Critic model, Deep Q-Network (DQN), Double Deep Q-Network (DDQN) and Dueling Double Deep Q-Network (DDDQN) to see which performs better.

REFERENCS

[1] Andrieu, Christophe, Arnaud Doucet, and Roman Holenstein. "Particle markov chain monte carlo methods." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 72, no. 3 (2010): 269-342.

[2] Schulman, John, S. Levine, P. Abbeel, Michael I. Jordan and P. Moritz. "Trust Region Policy Optimization." *ICML* (2015).

[3] Lillicrap, T., J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and Daan Wierstra. "Continuous control with deep reinforcement learning." *CoRR* abs/1509.02971 (2016): n. pag.

[4] Szepesvári, Csaba. "Algorithms for reinforcement learning." *Synthesis lectures on artificial intelligence and machine learning* 4, no. 1 (2010): 1-103.

[5] Khamassi, Mehdi, Loïc Lachèze, Benoît Girard, Alain Berthoz, and Agnès Guillot. "Actor–Critic models of reinforcement learning in the basal ganglia: from natural to artificial rats." *Adaptive Behavior* 13, no. 2 (2005): 131-148.

[6] Lee, Alex, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. "Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model." *Advances in Neural Information Processing Systems* 33 (2020).

[7] Mnih, Volodymyr, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous methods for deep reinforcement learning." In *International conference on machine learning*, pp. 1928-1937. 2016.

[8] Singh, Ritika, and Shashi Srivastava. "Stock prediction using deep learning." *Multimedia Tools and Applications* 76, no. 18 (2017): 18569-18584.

[9] Fan, Jianqing, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang. "A theoretical analysis of deep Q-learning." In *Learning for Dynamics and Control*, pp. 486-489. PMLR, 2020.

[10] Gu, Shixiang, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. "Continuous deep q-learning with model-based acceleration." In *International Conference on Machine Learning*, pp. 2829-2838. 2016.