# Reinforced Stock Trading

*Submitted in partial fulfilment of the requirements for the degree of*

# Bachelor of Technology

in

# Computer Science and Engineering

*by*


**SHAURYA CHOUDHARY**
**18BCE2113**


**Under the guidance of**


**Prof. Goapalakrishnan T**
**SCOPE**
**VIT, Vellore.**


VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

June, 2020

# DECLARATION

I hereby declare that the thesis entitled "Reinforced Stock Trading" submitted by me, for the award of the degree of *Bachelor of Technology in CSE* to VIT is a record of bonafide work carried out by me under the supervision of Prof. Gopalakrishnan T.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore
Date: 03 / 06 / 2020

shaurya choudhary

**Signature of the Candidate**

## CERTIFICATE

This is to certify that the thesis entitled "Thesis title" submitted by **Shaurya Choudhary (18BCE2113)**, **SCOPE**, VIT, for the award of the degree of *Bachelor of Technology in CSE*, is a record of bonafide work carried out by him under my supervision during the period, 01. 12. 2019 to 30.05.2020, as per the VIT code of academic and research ethics. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfils the requirements and regulations of the University and in my opinion meet the necessary standards for submission.

Place: Vellore
Date: 03 / 06 / 2020

**Signature of the Guide**

**Internal Examiner** **External Examiner**

Head of the Department
SCOPE

# ACKNOWLEDGEMENTS

I would like to express my special thanks of gratitude to my teacher Prof. Gopalakrishnan T who gave me the golden opportunity to do this wonderful project on Reinforcement Learning, which also helped me in doing a lot of Research and I came up with the implementation of Stock market prediction for short-term using deep learning algorithms. The project "Reinforced Stock Trading" has helped me realize the potential and real-life applications of Artificial Intelligence and I came to know about so many new things, I am really thankful to them.

Secondly, I would also like to thank my friends who helped me in making this project possible and a lot in finalizing this project within the limited time frame.

**Shaurya Choudhary**

# Executive Summary

Stock Markets have always been of much importance as a part of economy. Since the coming of AI, organizations have started to embrace algorithmic exchanging the acquisition of stocks and other budgetary resources.

The project focuses on the problem of predicting Stock Market prices and thus trying to maximize the profit through Trading. A Reinforcement Learning algo. Q-learning would be used for achieving our goal. It is based on reward and penalty approach for optimising Q-values. The agent is allowed 3 possible actions: Sit, Buy, Sell and is free to perform them according to its choice. After each decision, the outcome is evaluated and Neural Network is changed accordingly with the Q-values. This makes the agent repeat more of the decisions leading to Rearwards and avoid penalties.

TensorFlow and Keras modules on a python environment is used to design the neural network and train the agent. The model is trained on S&P500 dataset from the duration of last decade. Further details about implementation is discussed later in this report. The model is designed to work on short-term trading, and it keeps on getting better over time. As a result, on evaluating the model against test dataset, a profit of $705.41 is observed.

As promising as the results look, this implementation of trading style may have adverse effects in real-life as it accompanies more expenses and vulnerability.

Nonetheless, the potential of AI can be clearly demonstrated from this Reinforced Stock Trading implementation. This gives us some great insights and eases our workflow beyond human capabilities. The era of AI is just getting started and there’s much more to come.

# CONTENTS

**Page No.**

# List of Abbreviations

| | |
|---|---|
| S&P500 | Standard and Poor’s 500 |
| ML | Machine Learning |
| AI | Artificial Intelligence |
| Adj. | Adjusted |
| RL | Reinforcement Learning |
| Q-learning | Quality Learning |
| Algo. | Algorithm |
| TD | Temporal Difference |

# 1. INTRODUCTION

## 1.1. OBJECTIVE

Stock Markets have always been of much importance as a part of economy. For an economy to flourish, its money related market must be strong. Since the coming of AI, organizations have started to embrace algorithmic exchanging the acquisition of stocks and other budgetary resources. There has been demonstrated fruitful with this technique, and it has ascended in noticeable quality over time. Given its ascent, a few machine models have been created and embraced for algorithmic exchanging. One famous AI model for trading is the Reinforcement Learning. The same will be implemented in this project using TensorFlow and Keras, and in this part, they will be utilized to build up a model that can foresee stock costs.

## 1.2. BACKGROUND PROBLEM

Automation is taking over in pretty much every area, and the monetary market is no special case. Making mechanized algorithmic exchanging models will accommodate a quicker and progressively exact investigation of stocks before buy. Different markers can be analysed at a speed that people are unequipped for. Additionally, in stock market trading, it is perilous to work with feelings. AI models can take care of that issue. There is likewise a decrease in exchange costs, as there is no requirement for persistent management.

# 2. PROJECT DESCRIPTION

## 2.1. OVERVIEW

It's execution of Q-learning applied to (short-term) stock exchanging. The model uses n-day windows of shutting costs to decide whether the best move to make at a given time is to purchase, sell or sit.

Because of the momentary state portrayal, the model isn't truly adept at settling on choices over long-haul patterns, however is very acceptable at anticipating pinnacles and troughs.

### 2.2. DATA USED

The information that we will utilize will be the S&P500 (Standard and Poor's 500). As indicated by Wikipedia, it is An American financial exchange record dependent available capitalization of 500 huge organizations having regular stock recorded on the NYSE or NASDAQ. The same can be extracted from Yahoo Finance.

### 2.3. DATASET STRUCTURE

The data has the following columns:

1. **Date**: This indicates the date under consideration
2. **Open**: This indicates the price at which the market opens on the date
3. **High**: This indicates the highest market price on the date
4. **Low**: This indicates the lowest market price on the date
5. **Close**: This indicates the price at which the market closes on the date, adjusted for the split
6. **Adj. Close**: This indicates the adjusted closing price for both the split and dividends
7. **Volume**: This indicates the total volume of shares available

The date considered for training the model is as per the following:

Start: 03/05/2010

End: 29/05/2020

## 3. TECHNICAL SPECIFICATION

### 3.1. POSSIBLE ACTIONS

The implemented AI Agent will be allowed only certain actions that can be performed with the stocks. The possible actions are:

1. **Sit**: This means that based on the price and projected profit, the trader should hold a stock
2. **Sell**: This means that based on the price and projected profit, the trader should sell a stock
3. **Buy**: This means that based on the price and projected profit, the trader should buy a stock

### 3.2. ENVIRONMENT SETUP

Python is used for the development of this ML model. For AI implementation various Python specific modules have been used which are stated below. The details about the device used is also specified.

Machine Specifications:

- CPU: i5-9300H
- RAM: 16GB
- GPU: GTX 1050
- HDD: 1.5TB

Modules Used:

1. tensorflow
2. keras
3. numpy
4. collections
5. math
6. sys

## 4. DESIGN APPROACH AND DETAILS

### 4.1. DESIGN APPROACH

The Q-Learning algorithm is implemented for short-term stock trading. The goal of the model is to maximize the profit by performing any of the 3 possible actions. Q-learning is a RL algo. which uses a reward and penalty approach.

*Short-term trading* includes taking a position that can last from seconds to a few days. It is utilized as an option in contrast to the more conventional purchase and-hold methodology, in which you'd hold a situation for a considerable length of time, months or even years. Momentary exchanging focuses mostly around value activity, as opposed to the drawn-out essentials of an advantage. This exchanging style endeavours to benefit from speedy moves in showcase costs.

*Reinforcement Learning* briefly is a paradigm of Learning Process in which a learning agent learns, overtime, to behave optimally in a certain environment by interacting continuously in the environment. The agent during its course of learning experience various different situations in the environment it is in. These are called states. The agent while being in that state may choose from a set of allowable actions which may fetch different rewards (or penalties). The learning agent overtime learns to maximize these rewards so as to behave optimally at any given state it is in.

*Q-learning* is a basic form of Reinforcement Learning which uses Q-values (also called action values) to iteratively improve the behaviour of the learning agent.

- Q-**Values**: It gives an estimation of how good is it to take the action A at the state S.
- **Rewards and Episodes**: An agent over the course of its lifetime starts from a start state, makes a number of transitions from its current state to a next state based on its choice of action and also the environment the agent is interacting in. At every step of transition, the agent from a state takes an action, observes a reward from the environment, and then transits to another state. If at any point of time the agent ends up in one of the terminating states that means there are no further transition possible. This is said to be the completion of an episode
- **TD-Update**: This update rule to estimate the value of Q is applied at every time step of the agent's interaction with the environment.
- **ε-greedy Policy**: is a very simple policy of choosing actions using the current Q-value estimations.
  - With probability (1 – ε) choose the action which has the highest Q-value
  - With probability (ε) choose any action at random.

## 4.2. CODES AND STANDARDS

There are 4 scripts used in the implementation of this project:

- **agent**: The agent is defined to perform the Q-learning algorithm on a neural network designed from Keras module.
- **functions**: Several functions are defined which prove to be useful in training and evaluation of the model.
- **train**: This script is used to train the agent on out dataset with the help of functions script.
- **evaluate**: It is used to evaluate and assess the performance of the model.

SCRIPTS:

- *agent.py*

```
from keras.models import Sequential
from keras.models import load_model
from keras.layers import Dense
from keras.optimizers import Adam

import numpy as np
import random
from collections import deque


class Agent:
    def __init__(self, state_size, is_eval=False, model_name=""):
        self.state_size = state_size  # normalized previous days
        self.action_size = 3  # sit, buy, sell
        self.memory = deque(maxlen=1000)
        self.inventory = []
        self.model_name = model_name
        self.is_eval = is_eval

        self.gamma = 0.95
        self.epsilon = 1.0
        self.epsilon_min = 0.01
        self.epsilon_decay = 0.995

        self.model = load_model("models/" + model_name) if is_eval else self._model()

    def _model(self):
        model = Sequential()
        model.add(Dense(units=64, input_dim=self.state_size, activation="relu"))
        model.add(Dense(units=32, activation="relu"))
        model.add(Dense(units=8, activation="relu"))
        model.add(Dense(self.action_size, activation="linear"))
        model.compile(loss="mse", optimizer=Adam(lr=0.001))

        return model

    def act(self, state):
        if not self.is_eval and random.random() <= self.epsilon:
            return random.randrange(self.action_size)

        options = self.model.predict(state)
        return np.argmax(options[0])

    def expReplay(self, batch_size):
        mini_batch = []
        l = len(self.memory)
        for i in range(l - batch_size + 1, l):
            mini_batch.append(self.memory[i])

        for state, action, reward, next_state, done in mini_batch:
            target = reward
            if not done:
                target = reward + self.gamma * np.amax(self.model.predict(next_state)[0])

            target_f = self.model.predict(state)
            target_f[0][action] = target
            self.model.fit(state, target_f, epochs=1, verbose=0)

        if self.epsilon > self.epsilon_min:
            self.epsilon *= self.epsilon_decay
```

- *functions.py*

```
import numpy as np
import math


# prints formatted price
def formatPrice(n):
    return ("-$" if n < 0 else "$") + "{0:.2f}".format(abs(n))


# returns the vector containing stock data from a fixed file
def getStockDataVec(key):
    vec = []
    lines = open("data/" + key + ".csv", "r").read().splitlines()

    for line in lines[1:]:
        vec.append(float(line.split(",")[4]))

    return vec


# returns the sigmoid
def sigmoid(x):
    return 1 / (1 + math.exp(-x))


# returns an an n-day state representation ending at time t
def getState(data, t, n):
    d = t - n + 1
    block = data[d:t + 1] if d >= 0 else -d * [data[0]] + data[0:t + 1]
# pad with t0
    res = []
    for i in range(n - 1):
        res.append(sigmoid(block[i + 1] - block[i]))

    return np.array([res])
```

- *train.py*

```
from agent.agent import Agent
from functions import *
import sys

if len(sys.argv) != 4:
    print("Usage: python train.py [stock] [window] [episodes]")
    exit()

stock_name, window_size, episode_count = sys.argv[1], int(sys.argv[2]),
int(sys.argv[3])

agent = Agent(window_size)
data = getStockDataVec(stock_name)
l = len(data) - 1
batch_size = 32

for e in range(episode_count + 1):
    print("Episode " + str(e) + "/" + str(episode_count))
    state = getState(data, 0, window_size + 1)

    total_profit = 0
    agent.inventory = []

    for t in range(l):
        action = agent.act(state)

        # sit
        next_state = getState(data, t + 1, window_size + 1)
        reward = 0

        if action == 1:  # buy
            agent.inventory.append(data[t])
            print("Buy: " + formatPrice(data[t]))

        elif action == 2 and len(agent.inventory) > 0:  # sell
            bought_price = agent.inventory.pop(0)
            reward = max(data[t] - bought_price, 0)
            total_profit += data[t] - bought_price
            print("Sell: " + formatPrice(data[t]) + " | Profit: " +
formatPrice(data[t] - bought_price))

        done = True if t == l - 1 else False
        agent.memory.append((state, action, reward, next_state, done))
        state = next_state

        if done:
            print("--------------------------------")
            print("Total Profit: " + formatPrice(total_profit))
            print("--------------------------------")

        if len(agent.memory) > batch_size:
            agent.expReplay(batch_size)

    if e % 10 == 0:
        agent.model.save("models/model_ep" + str(e))
```

- *evaluate.py*

```
import sys

from keras.models import load_model

from agent.agent import Agent
from functions import *

if len(sys.argv) != 3:
    print("Usage: python evaluate.py [stock] [model]")
    exit()

stock_name, model_name = sys.argv[1], sys.argv[2]
model = load_model("models/" + model_name)
window_size = model.layers[0].input.shape.as_list()[1]

agent = Agent(window_size, True, model_name)
data = getStockDataVec(stock_name)
l = len(data) - 1
batch_size = 32

state = getState(data, 0, window_size + 1)
total_profit = 0
agent.inventory = []

for t in range(l):
    action = agent.act(state)

    # sit
    next_state = getState(data, t + 1, window_size + 1)
    reward = 0

    if action == 1:  # buy
        agent.inventory.append(data[t])
        print("Buy: " + formatPrice(data[t]))

    elif action == 2 and len(agent.inventory) > 0:  # sell
        bought_price = agent.inventory.pop(0)
        reward = max(data[t] - bought_price, 0)
        total_profit += data[t] - bought_price
        print("Sell: " + formatPrice(data[t]) + " | Profit: " +
formatPrice(data[t] - bought_price))

    done = True if t == l - 1 else False
    agent.memory.append((state, action, reward, next_state, done))
    state = next_state

    if done:
        print("--------------------------------")
        print(stock_name + " Total Profit: " +
formatPrice(total_profit))
        print("--------------------------------")
```

RUNNIG THE PROJECT:

For the sake of functionality, the script is designed to be efficiently run from the terminal. Different models can be trained from different Datasets, windows, and episodes.
Then we can choose any of the trained models to evaluate its performance.

4.3. TRADEOFFS

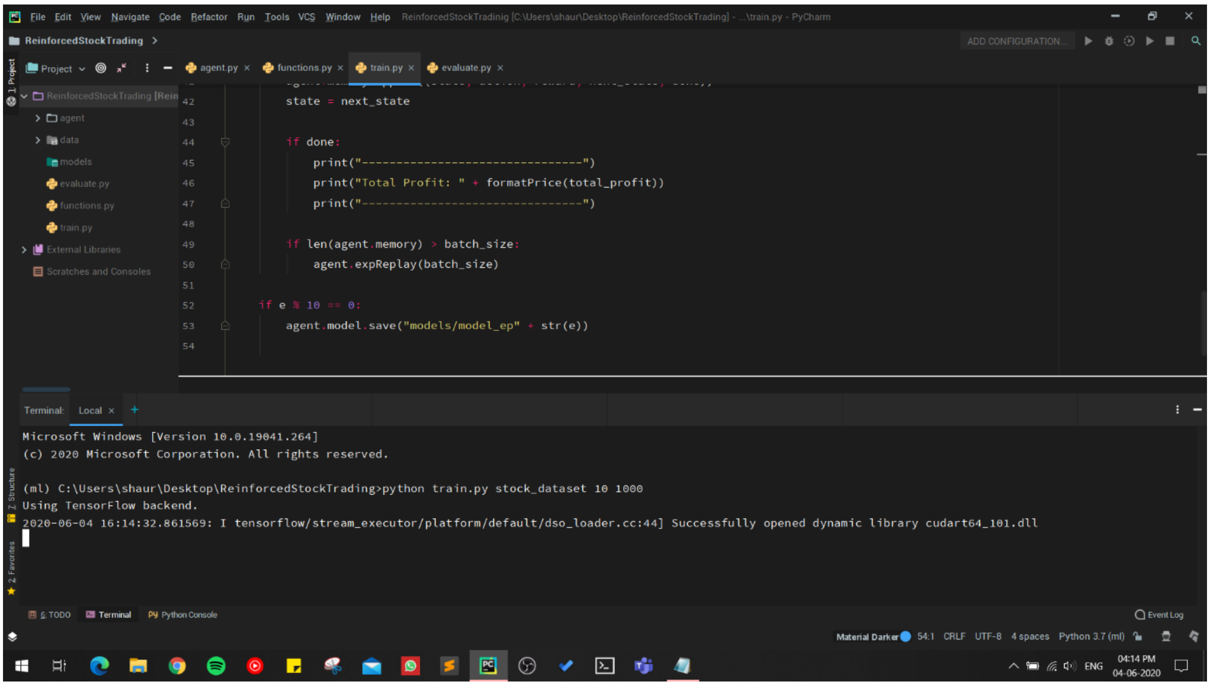
This style of short-term trading isn't reasonable for this present reality, as exchanging includes more expenses and vulnerability; consequently, this exchanging style could have adverse impacts.

As a result of the short-term state representation, the model is not very good at making decisions over long-term trends, but is quite good at predicting peaks and troughs.

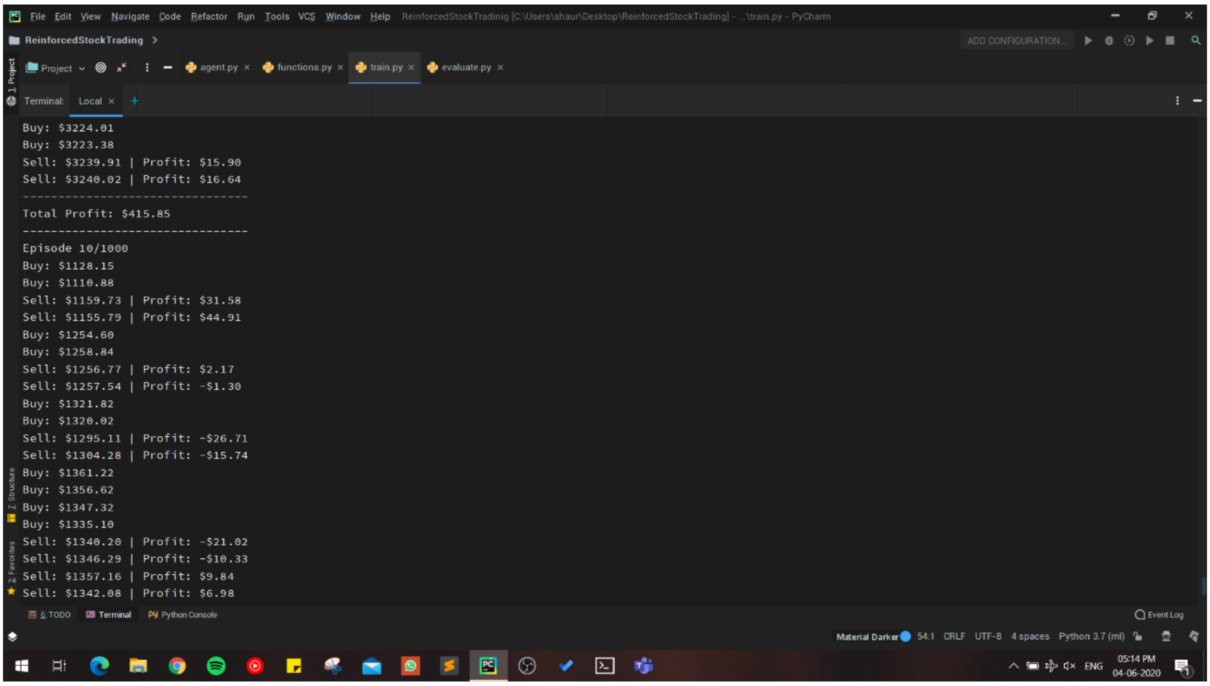
## 5. PROJECT DEMOSTRATION

- <u>Starting to train the Neural Network</u>

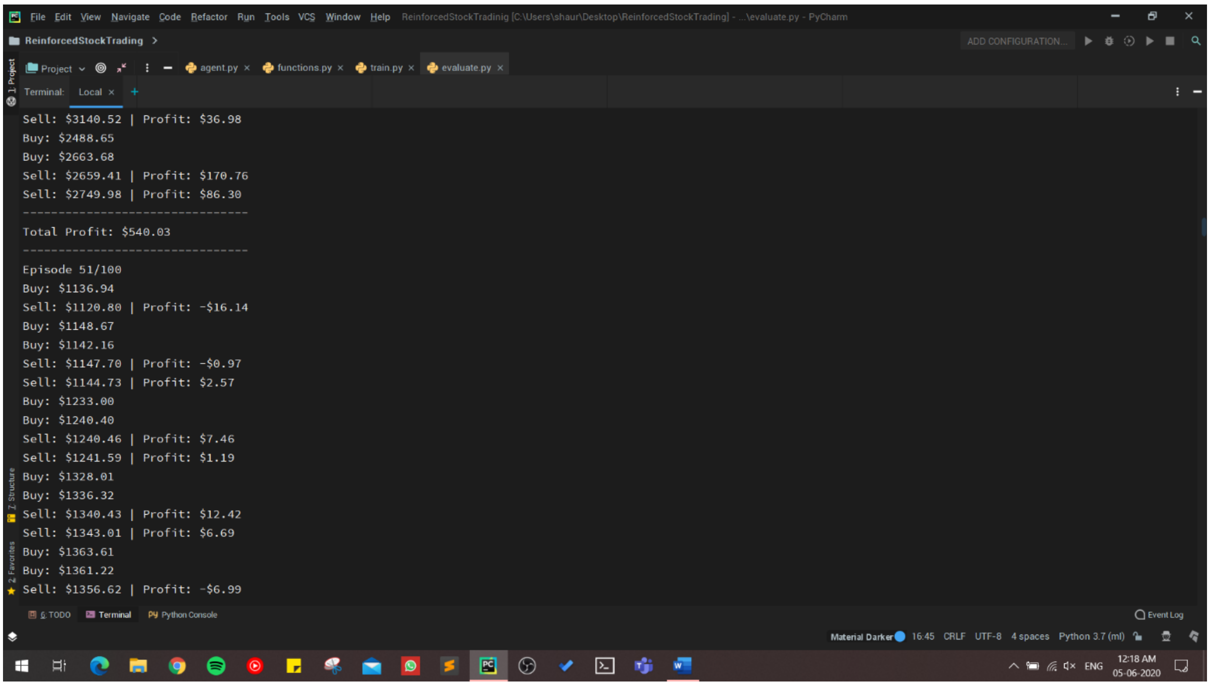
  (window = 10; episodes = 100)



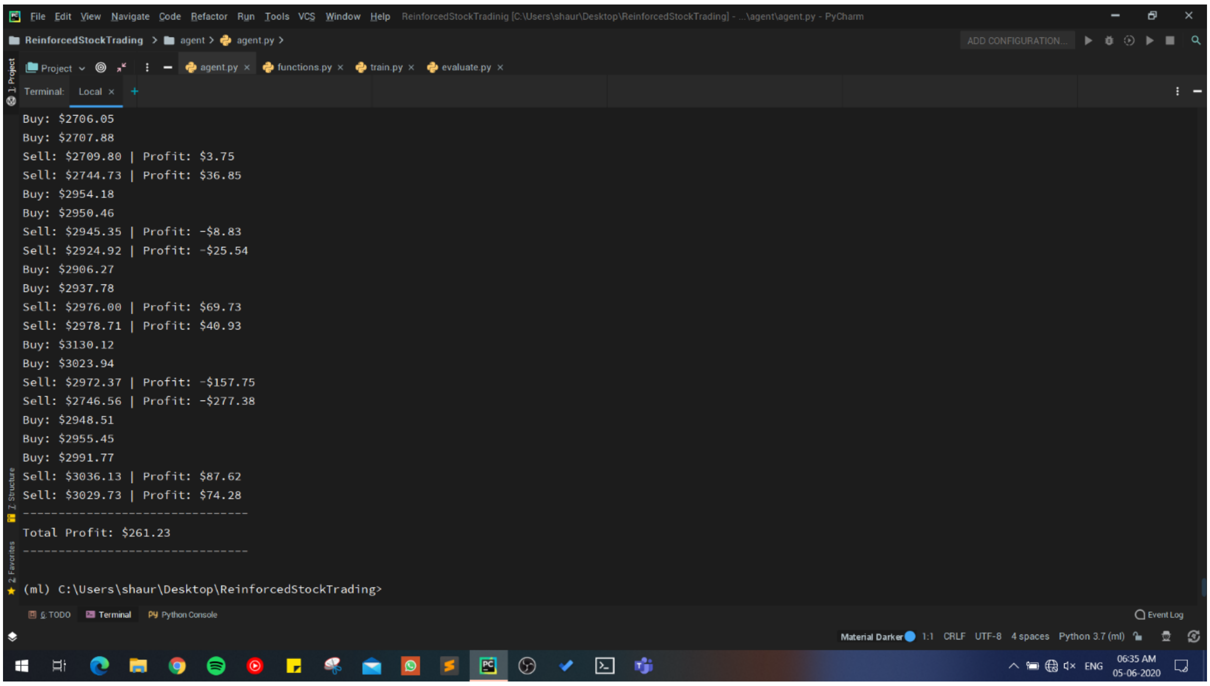
- Training the model



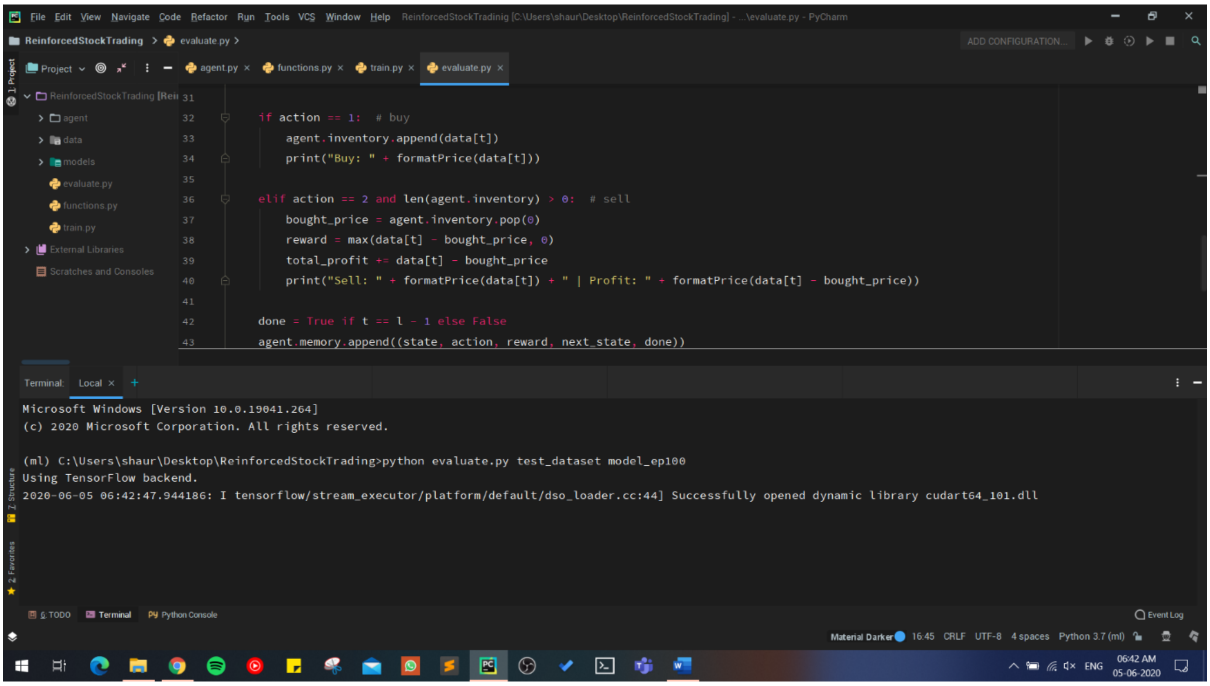

- 50% training completion



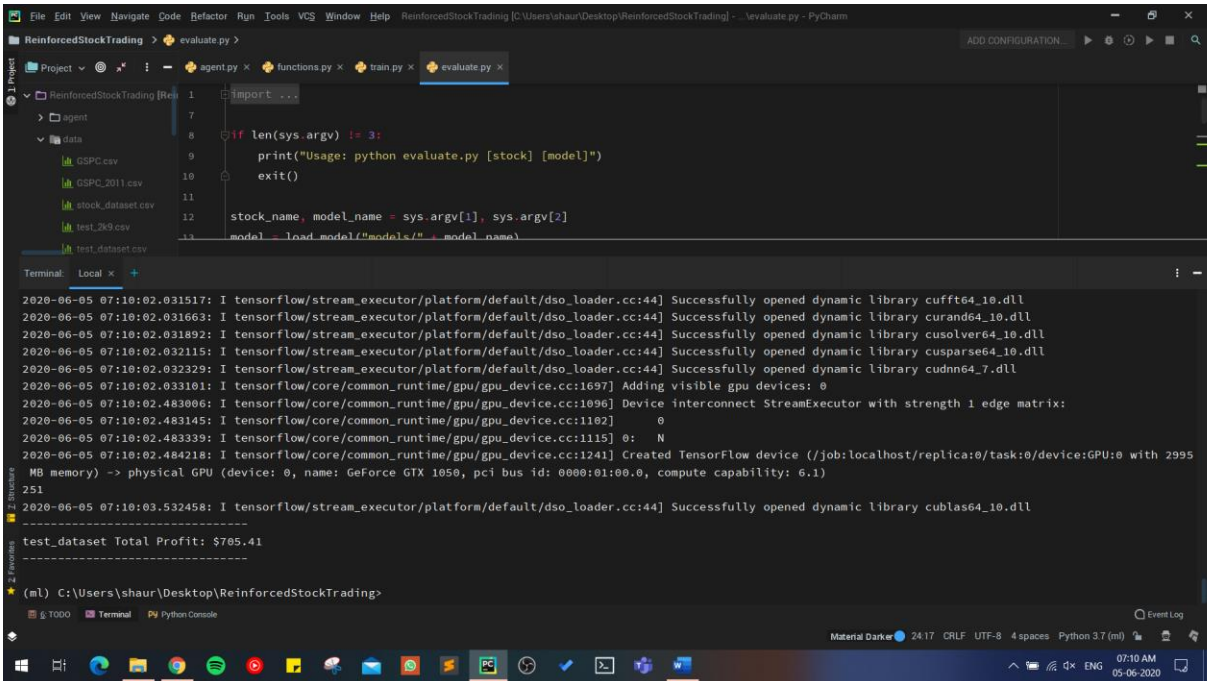
- <u>After Training Completion</u>




- <u>Starting Evaluation</u>



- <u>Final Result</u>




## 6. RESULT

After training the data, it was assessed using evaluate script. The model resulted in a **total profit of $705.41**. The best thing about the model was that the profits kept improving over time, indicating that it was learning well and taking better actions. The problem of overfitting was also not found in the trained neural network and it performed well against test data.

## 7. SUMMARY

Taking everything into account, AI can be applied to a few enterprises and can be applied effectively in money related markets, as you found in this project. We can consolidate various models, as we did with reinforcement learning, to deliver more grounded models that suit our utilization cases. The model used Q-learning algorithm that decided the best activity, in light of the condition of the stock costs, with the point of expanding benefits. At long last, we acquired an outcome that boasted an overall benefit and included expanding profits after some time, showing that the operator learned more with each state.

# References


[1] Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. Journal of artificial intelligence research, 4, 237-285.

[2] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016). Tensorflow: A system for large-scale machine learning. In 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16) (pp. 265-283).

[3] Gulli, A., & Pal, S. (2017). Deep learning with Keras. Packt Publishing Ltd.

[4] Kimoto, T., Asakawa, K., Yoda, M., & Takeoka, M. (1990, June). Stock market prediction system with modular neural networks. In 1990 IJCNN international joint conference on neural networks (pp. 1-6). IEEE.

[5] https://machinelearningmastery.com/introduction-python-deep-learning-library-tensorflow/#:~:text=TensorFlow%20is%20a%20Python%20library,built%20on%20top%20of%20TensorFlow.

[6] https://adventuresinmachinelearning.com/reinforcement-learning-tutorial-python-keras/

[7] https://www.geeksforgeeks.org/q-learning-in-python/

[8] Kimoto, T., Asakawa, K., Yoda, M., & Takeoka, M. (1990, June). Stock market prediction system with modular neural networks. In 1990 IJCNN international joint conference on neural networks (pp. 1-6). IEEE.