**TECHNOCOLABS DATA SCIENCE INTERNSHIP**

**Project Report**

# Title: Optimizing Stock Trading Strategy With Reinforcement Learning


**Submitted By**

**[Team B]**

Anweasha Saha

Roshni

Jagadeesh T S

Prakash

Karina Farheen Shareef

Amey Mahendra Thakur

E. Sailesh

**Under the Guidance of**

**Deepthika Shiwani Muralikrishnan**

## AIM

The main emphasis and objective of our project is to analyse given raw data and do exploratory data analysis in order to fully comprehend and identify patterns. Then, using a Neural Network approach, construct a model and train it to get the desired outcomes. Finally, it will be deployed as a web application.

## ABSTRACT

We propose to determine a single exchange-traded fund (SPY) investing strategy that will maximise our total wealth. We compare our findings to the tried-and-true "buy-and-hold" and Moving average convergence divergence (MACD) strategies. It's a Machine Learning model which integrates Data Science and Web Development. We have deployed the app on the Heroku Cloud Application Platform. Here, we intend to base an evaluation on every basic criterion that is taken into account when establishing the pricing. As mentioned, we intend to forecast future price fluctuations for a specific stock. Prices from previous days, as well as financial media stories connected to the firm of interest, are used to create these forecasts. Reinforcement learning is all about taking the right steps to maximise your reward in a given situation. It is used by a variety of software and computers to determine the best feasible action or path in a given situation. The Reinforcement Machine Learning model is employed in this work to forecast the closure price using past data. We develop a predictor for multiple firms using these trained models that forecasts the every day close stock prices. By providing inputs such as open, high, and low prices, stock volume, and the latest events about each firm, this predictor may be used to determine the price at which the stock value will close for a certain day. The goal of this project is to learn and get hands-on experience in Data Analytics and Machine Learning.

## INTRODUCTION

We've always been captivated by the stock market's seeming unpredictability. There are hundreds of stocks to select from, and day traders can trade almost any of them. So, for a day trader, deciding what to trade is the first and most important step. The following stage is to come up with some ways to benefit from the trading opportunity (one stock, several stocks, exchange-traded funds, ETFs, etc.). Intraday

traders use a number of ways to profit from price movements in a particular asset. Day traders should look for equities with plenty of liquidity, moderate to high volatility, and a large number of followers. Isolating the present market trend from any surrounding noise and then capitalising on that trend is the key to finding the appropriate stocks for intraday trading.

This has traditionally been done in conjunction with the trade plan and current events. Various research techniques have been explored to automate this laborious procedure since the emergence of Data Science and Machine Learning. This automated trading method will assist in providing recommendations at the appropriate moment and with more accurate estimates. Mutual funds and hedge funds would benefit greatly from an automated trading approach that maximises profits. The type of profitable returns that may be expected will be accompanied by some risk. It's difficult to come up with a lucrative automated trading technique.

Every human being aspires to make as much money as possible in the stock market. It's critical to devise a well-balanced, low-risk plan that will benefit the majority of individuals. One such technique proposes the use of reinforcement learning agents to generate automated trading strategies based on previous data. This project was made because we were intrigued and we wanted to gain hands-on experience with the Machine Learning Project.

**REINFORCEMENT LEARNING**

Machine learning models are taught to make a series of judgments via reinforcement learning. In an unpredictable and potentially complex environment, the agent must learn to attain a goal. Artificial intelligence is put in a game-like environment in reinforcement learning. To find a solution to the problem, the computer uses trial and error. Artificial intelligence is given either rewards or penalties for the acts it takes in order to get it to accomplish what the programmer desires. Its purpose is to increase the total prize as much as possible.

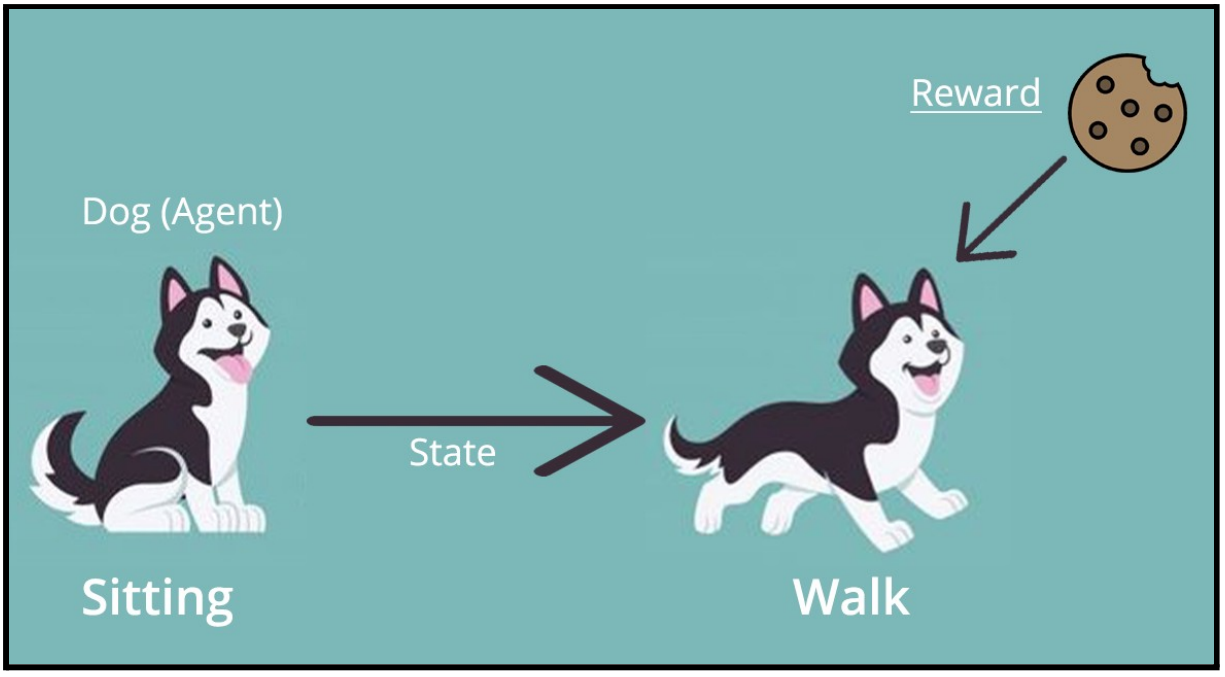
Despite the fact that the designer establishes the reward policy–that is, the game's rules–he provides the model with no clues or ideas for how to solve the game.

Starting with completely random trials and progressing to sophisticated tactics and superhuman skills, it's up to the model to find out how to do the task in order to maximise the reward.

In reinforcement learning, developers use a framework that rewards desired actions while penalising negative ones. To motivate the agent, this strategy assigns positive values to desired acts and negative values to undesirable behaviours. To obtain an ideal solution, the agent is programmed to seek long-term and greatest overall return.

These long-term objectives keep the agent from stagnating on smaller objectives. The agent eventually learns to ignore the unpleasant and focus on the good. This technique of learning has been used in artificial intelligence (AI) to drive unsupervised machine learning using incentives and punishments.

In simple terms, You let the dog do whatever it wants and then whenever it behaves well you go, “Good dog” and when it misbehaves you go “Bad Dog!” and then over time the dog learns to do more of the good dog things and fewer of the bad dog things.



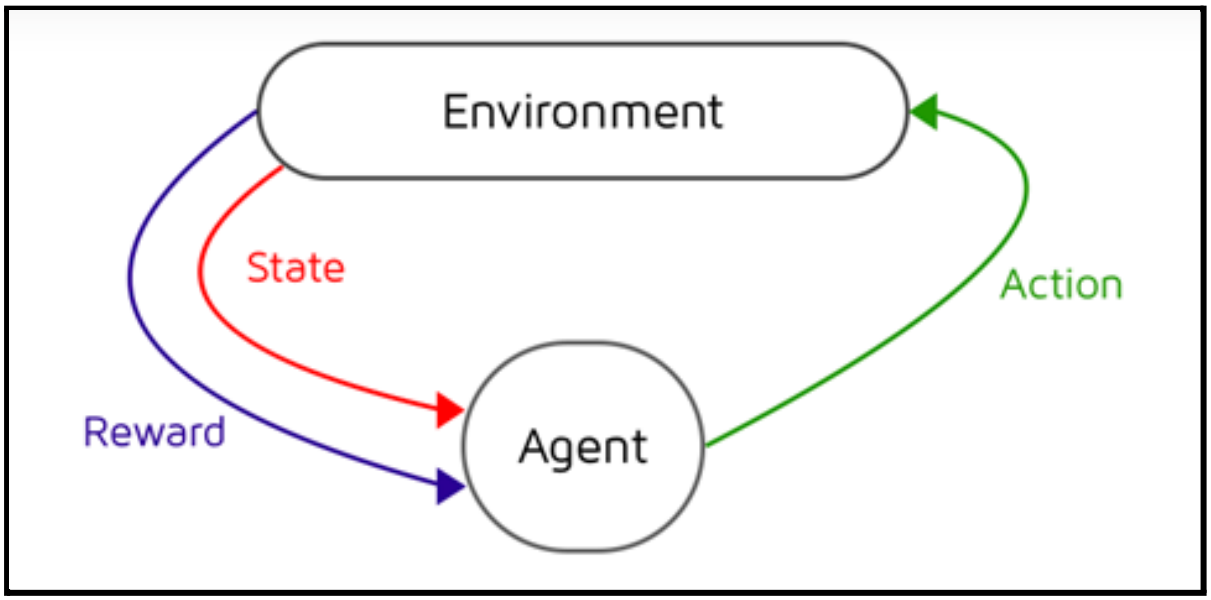
Trading is a never-ending process with no conclusion in sight. Because we don't have comprehensive information about the traders in the market, trading is likewise a stochastic Markov Decision Process. We employ model-free reinforcement learning, also known as Q-Learning because we don't know the reward function or transition probability.

**Markov Decision Process (MDP):**

It is a mathematical framework used for modelling decision-making problems where the outcomes are partly random and partly controllable.

A model of anticipating outcomes is the Markov decision process. The model, like a Markov chain, tries to predict a result based solely on the present state's information. The Markov decision process, on the other hand, takes into account the features of actions and motivations. The decision-maker may choose an action accessible in the current state at each phase of the process, causing the model to advance to the next step and rewarding the decision-maker.

At each step during the process, the decision-maker may choose to take any action available in the current state, resulting in the model moving to the next step and offering the decision-maker a reward.



An optimization problem may be given to a machine learning algorithm. The programme will use reinforcement learning to try to optimise the behaviours made inside a given environment in order to maximise the potential reward. Reinforcement learning employs Markov decision processes to establish an ideal balance of exploration and exploitation, whereas supervised learning approaches need accurate input/output pairings to construct a model. When the probability and rewards of a result are undefined or unknown, machine learning may employ reinforcement learning through the Markov decision process.

Terminologies:

1. Agent - An RL agent is a machine that we're teaching to make good judgments. For instance, a robot is being taught how to navigate a house without colliding.
2. Environment - The agent's environment is the setting in which the agent interacts. For example, the house where the Robot moves. The agent has no influence over the surroundings; all it has is control over its own actions. For instance, Although the Robot cannot control where a table is kept in the house, it can walk around it to avoid colliding with it.
3. State - The state defines the current situation of the agent, for example, It may be the Robot's specific position in the house, the alignment of its two legs, or its current posture, depending on how you approach the issue.
4. Action - The decision made by the agent in the current time step. For instance: it can move its right or left leg, or raise its arm, or lift an object, turn right or left, etc. We know ahead of time what actions or decisions the agent can take.
5. Policy - A policy is the thinking process that goes into deciding on a course of action. It is a probability distribution applied to the collection of activities in practice. Actions that are very rewarding will have a high likelihood, and vice versa.

Note: Even if an action has a low probability, that does not mean it will not be chosen. It's only that it has a lower chance of being chosen.

## Q-LEARNING

The 'Q in Q-learning stands for quality. Quality in this case represents how useful a given action is in gaining some future reward. Q-learning is an off-policy reinforcement learning algorithm that seeks to find the best action to take given the current state. It's considered off-policy because the q-learning function learns from actions that are outside the current policy, like taking random actions, and therefore a policy isn't needed. More specifically, q-learning seeks to learn a policy that maximizes the total reward.

An agent interacts with the environment in one of two ways – exploit or explore – and a Q-table is produced with the dimensions. An exploit option implies that all options are evaluated and the one with the greatest environmental benefit is chosen. An explore option is one that considers a random action without regard for the maximum future payoff.

## POINTS TO CONSIDER

1. Gamification of trading - Historical & Current Prices, Technical Data, Buy/Sell/Do Nothing, Profit & Loss.
2. Train the system - Each entry and exit is an individual game, Run through the price series sequentially and randomly.
3. Reward function design - Pure Profit & Loss on exit, otherwise zero, PnL from the start of trade to every time step t.
4. Features to use - Open, High, Low, Close, Volume (OHLCV), Technical indicators, Time of day, Day of Week, Time of year, Different time granularity.
5. Test the system - Sine waves, Trend curves, Random walks, Adding Noise to clean test curves.
6. Types of Deep Learning Algorithms.

## STEPS TO CREATE AND DEPLOY MODEL

1. Import necessary Libraries.
2. Load Dataset.
3. Perform Exploratory Data Analysis.
4. Create An Environment.

5. Prepare Data.
6. Train Data.
7. Test Data.
8. Evaluate Model.
9. Deploy Model.

**IMPORT NECESSARY LIBRARIES**

```
import time
import copy
import numpy as np
import pandas as pd
import chainer
import chainer.functions as F
import chainer.links as L
from plotly import tools, subplots
from plotly.graph_objs import *
from plotly.offline import init_notebook_mode, iplot, iplot_mpl
init_notebook_mode()
```

**LOAD DATASET**

Data on stock market values from 2013 to 2018 is included in this dataset. This dataset contains the following information: date, open, high, low, and close values, as well as volume and stock names. Also, the dataset is in CSV format.

```
data = pd.read_csv('../input/stock-prices/all_stocks_5yr.csv')
```

The following is a quick description of each component in the dataset:

1. DATE - The day on which stock is exchanged.
2. OPEN - Any listed stock's daily opening price is the price at which it is initially traded.
3. HIGH - The maximum price at which a stock can be bought or sold during a trading day is known as the high.

4. LOW - The minimum price at which a stock can be bought or sold during a trading day is known as the low.
5. CLOSE - During a standard trading session, the last price at which a stock trades is referred to as the closing price.
6. VOLUME - The total number of shares or contracts exchanged in a securities or market within a certain time period.
7. NAME - The name of the stock.

In [4]:
```
data.head()
```

Out[4]:

| | date | open | high | low | close | volume | Name |
|---|---|---|---|---|---|---|---|
| 0 | 2013-02-08 | 15.07 | 15.12 | 14.63 | 14.75 | 8407500 | AAL |
| 1 | 2013-02-11 | 14.89 | 15.01 | 14.26 | 14.46 | 8882000 | AAL |
| 2 | 2013-02-12 | 14.45 | 14.51 | 14.10 | 14.27 | 8126000 | AAL |
| 3 | 2013-02-13 | 14.30 | 14.94 | 14.25 | 14.66 | 10259500 | AAL |
| 4 | 2013-02-14 | 14.94 | 14.96 | 13.16 | 13.99 | 31879900 | AAL |

In [5]:
```
data.tail()
```

Out[5]:

| | date | open | high | low | close | volume | Name |
|---|---|---|---|---|---|---|---|
| 619035 | 2018-02-01 | 76.84 | 78.27 | 76.69 | 77.82 | 2982259 | ZTS |
| 619036 | 2018-02-02 | 77.53 | 78.12 | 76.73 | 76.78 | 2595187 | ZTS |
| 619037 | 2018-02-05 | 76.64 | 76.92 | 73.18 | 73.83 | 2962031 | ZTS |
| 619038 | 2018-02-06 | 72.74 | 74.56 | 72.13 | 73.27 | 4924323 | ZTS |
| 619039 | 2018-02-07 | 72.70 | 75.00 | 72.69 | 73.86 | 4534912 | ZTS |

**EXPLORATORY DATA ANALYSIS**

In every Data Analysis or Data Science project, exploratory data analysis, or EDA, is a critical stage. EDA is the investigation of a dataset to find patterns and anomalies as well as to generate hypotheses based on our knowledge of the information.

During the course of a data science or machine learning project, EDA consumes around half of the time spent on data analysis, feature selection, feature engineering, and other processes. Because it is the most essential element or backbone of a data science project, where one has to perform several tasks like data cleaning, dealing with missing values, handling outliers, treating unbalanced datasets, handling categorical features, and many others. To automate our tasks in exploratory data analysis, we may utilise python packages like dtale, pandas profiling, sweetviz, and autoviz.

```
import sweetviz as sv

advert_report = sv.analyze(df)

advert_report.show_html('EDA.html')
```

```
Done! Use 'show' commands to display/save. [100%] 00:00 -> (00:00 left)

Report EDA.html was generated! NOTEBOOK/COLAB USERS: the web browser MAY not pop up, re
gardless, the report IS saved in your notebook/colab files.
```

Find and resolve missing values in the dataset.

```
df.isnull().sum()
```

```
date      0
open     11
high      8
low       8
close     0
volume    0
Name      0
dtype: int64
```

```
df.dropna(inplace=True)
```

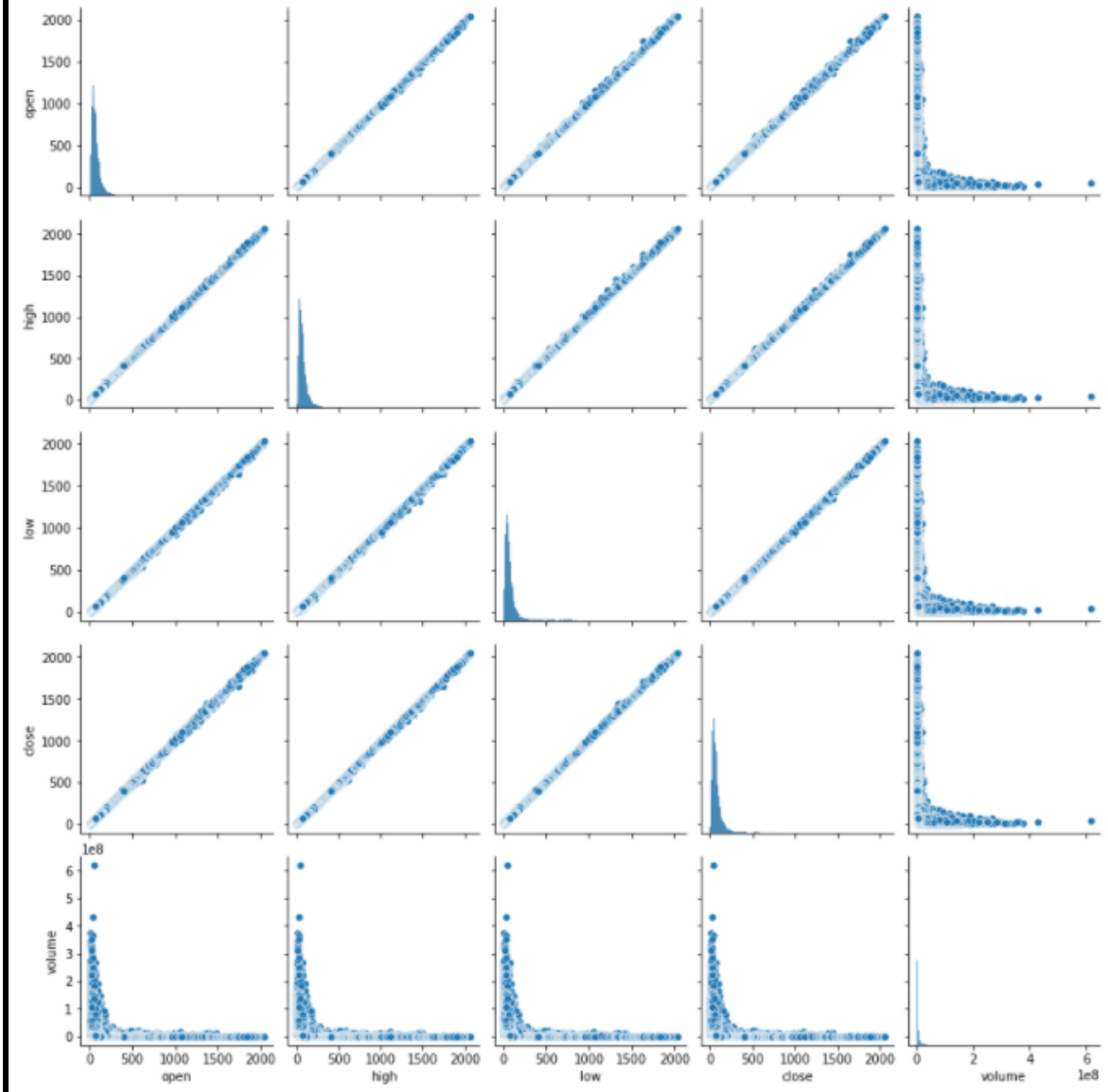
```
df.isnull().sum()
```

```
date      0
open      0
high      0
low       0
close     0
volume    0
Name      0
dtype: int64
```

Use a pairplot to visualise data and find patterns and relationships.

```
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x7ff4939743d0>
```



To conduct calculations for better analysis, use the Groupby() method to break the data into distinct groups.

```
df.groupby('Name').mean()
```

| Name | open | high | low | close | volume |
|---|---|---|---|---|---|
| A | 49.187863 | 49.600059 | 48.782026 | 49.202025 | 2.338039e+06 |
| AAL | 38.390495 | 38.955554 | 37.825605 | 38.393252 | 9.390321e+06 |
| AAP | 132.439631 | 133.818297 | 131.036025 | 132.433463 | 1.078043e+06 |
| AAPL | 109.055429 | 109.951118 | 108.141589 | 109.066698 | 5.404790e+07 |
| ABBV | 60.802801 | 61.474133 | 60.177275 | 60.864440 | 7.870683e+06 |
| ... | ... | ... | ... | ... | ... |
| XYL | 41.415473 | 41.763885 | 41.076026 | 41.434095 | 1.183141e+06 |
| YUM | 75.422099 | 76.027123 | 74.844914 | 75.451009 | 3.209032e+06 |
| ZBH | 105.542014 | 106.419854 | 104.698038 | 105.606291 | 1.297144e+06 |
| ZION | 32.161477 | 32.509290 | 31.802241 | 32.171790 | 2.621178e+06 |
| ZTS | 45.091389 | 45.488826 | 44.665588 | 45.098648 | 3.681878e+06 |

505 rows × 5 columns

```
def stocks(name):
    a = df[df['Name']==name]
    a = a.set_index('date')
    plt.plot(a['close'])

stocks('AMZN')
```

Detecting and removing outliers from the dataset.

Z-Score Method to detect outliers.

```
import numpy as np
outliers = []
def detect_outliers_zscore(df):
    thres = 3
    mean = np.mean(df)
    std = np.std(df)
    # print(mean, std)
    for i in df:
        z_score = (i-mean)/std
        if (np.abs(z_score) > thres):
            outliers.append(i)
    return outliers
sample_outliers = detect_outliers_zscore(df['close'])
print("Outliers from Z-scores method: ", sample_outliers)
```

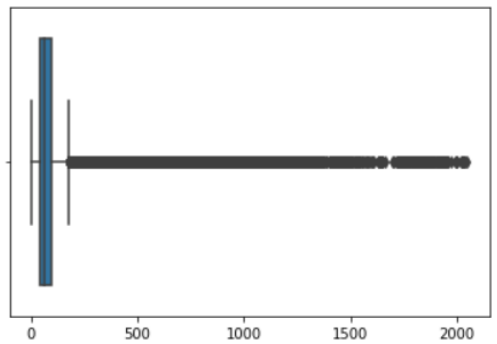
```
Outliers from Z-scores method:  [376.64, 381.37, 386.71, 393.62, 392.3, 384.66, 385.96, 384.
49, 386.95, 384.89, 387.78, 382.19, 381.25, 384.24, 388.97, 387.65, 395.96, 395.19, 402.2, 4
02.92, 399.2, 404.39, 398.08, 393.37, 398.79, 397.97, 396.44, 393.63, 398.03, 401.92, 401.0
1, 397.66, 390.98, 397.54, 395.87, 395.8, 399.61, 407.05, 404.54, 399.87, 387.6, 386.28, 39
4.43, 384.2, 403.01, 378.77, 377.17, 381.83, 375.43, 378.995, 383.66, 380.14, 378.59, 385.3
7, 384.8, 380.16, 385.655, 384.61, 382.72, 387.83, 380.09, 378.56, 378.49, 377.04, 381.2, 38
3.54, 382.65, 382.36, 385.11, 383.45, 386.04, 375.56, 389.51, 391.18, 389.8, 389.99, 445.1,
438.56, 429.31, 429.37, 421.78, 422.87, 423.04, 421.19, 419.1, 426.88, 433.69, 432.85, 431.0
2, 426.87, 432.28, 426.0, 425.24, 421.71, 423.86, 431.63, 427.63, 425.47, 431.42, 426.57, 42
9.23, 430.92, 430.99, 436.59, 430.78, 426.95, 423.5, 425.48, 430.77, 432.97, 429.92, 423.67,
427.26, 427.81, 439.39, 434.92, 436.29, 445.99, 440.84, 440.1, 438.1, 429.86, 434.09, 437.3
9, 437.71, 436.04, 436.72, 429.7, 434.39, 443.51, 455.57, 465.57, 461.19, 475.48, 483.01, 48
8.1, 488.0, 488.27, 482.18, 529.42, 531.41, 526.03, 529.0, 536.76, 536.15, 535.03, 531.9, 53
7.01, 529.46, 522.62, 524.0, 527.46, 525.91, 529.66, 531.52, 535.22, 535.02, 532.92, 515.78,
494.47, 463.37, 466.37, 500.77, 518.37, 518.01, 512.89, 496.54, 510.55, 504.72, 499.0, 517.5
4, 516.89, 522.24, 529.44, 521.38, 522.37, 527.39, 538.87, 540.26, 548.39, 538.4, 536.07, 53
3.75, 524.25, 504.06, 496.07, 511.89, 520.72, 532.54, 543.68, 537.48, 541.94, 533.16, 539.8,
550.19, 548.9, 544.83, 562.44, 570.76, 573.15, 560.88, 555.77, 563.91, 599.03, 608.61, 611.0
1, 617.1, 626.55, 625.9, 628.35, 625.31, 640.95, 655.65, 659.37, 655.49, 659.68, 673.25, 66
5.6, 642.35, 647.81, 643.3, 663.54, 661.27, 668.45, 678.99, 671.15, 675.34, 673.26, 664.8, 6
79.06, 676.01, 666.25, 672.64, 669.83, 677.33, 664.79, 662.32, 640.15, 657.91, 658.64, 675.7
7, 670.65, 664.14, 664.51, 663.15, 663.7, 662.79, 675.2, 693.97, 689.07, 675.89, 636.99, 63
3.79, 632.65, 607.94, 607.05, 617.74, 617.89, 581.81, 593.0, 570.18, 574.48, 571.77, 575.02,
596.38, 596.53, 601.25, 583.35, 635.35, 587.0, 574.81, 552.1, 531.07, 536.26, 502.13, 488.1,
482.07, 490.48, 503.82, 507.08, 521.1, 534.1, 525.0, 534.9, 559.5, 552.94, 554.04, 555.15, 5
55.23, 552.52, 579.04, 580.21, 577.49, 575.14, 562.8, 560.26, 559.47, 558.93, 569.61, 573.3
7, 577.02, 574.27, 559.44, 552.08, 553.98, 560.48, 569.63, 582.95, 579.87, 593.86, 598.69, 5
93.64, 598.5, 593.19, 586.14, 602.08, 591.43, 594.6, 595.93, 603.17, 614.82, 620.75, 625.89,
```

Use Boxplot and IQR (Inter-Quartile Range) for anomaly detection.

- Close

```
sns.boxplot(df['close'])
```

```
<AxesSubplot:xlabel='close'>
```




```
Q1 = df['close'].quantile(0.45)
Q3 = df['close'].quantile(0.55)
print(Q1,Q3)
iqr=Q3-Q1
iqr
```

```
57.47999999999999 67.71

10.230000000000004
```

```
upper = Q3+1.5*iqr
lower = Q1+1.5*iqr
print(upper,lower)
```

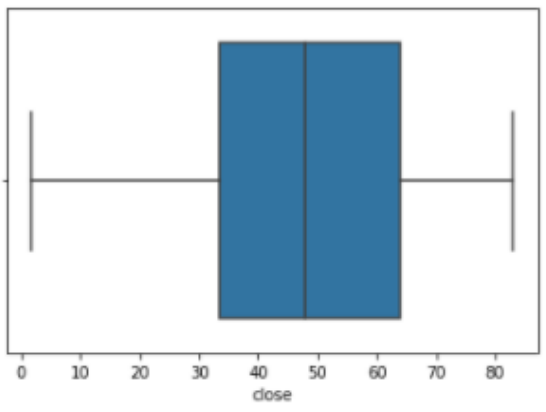
```
83.055 72.82499999999999
```

```
df1 = df[df['close']<upper]
df1.shape
```

```
(423200, 7)
```

```
sns.boxplot(df1['close'])
```

```
<AxesSubplot:xlabel='close'>
```



- Open

```
Q1 = df['open'].quantile(0.35)
Q3 = df['open'].quantile(0.65)
print(Q1,Q3)
iqr=Q3-Q1
iqr
```

```
48.44 78.79

30.35000000000001
```

```
upper = Q3+1.5*iqr
lower = Q1+1.5*iqr
print(upper,lower)
```

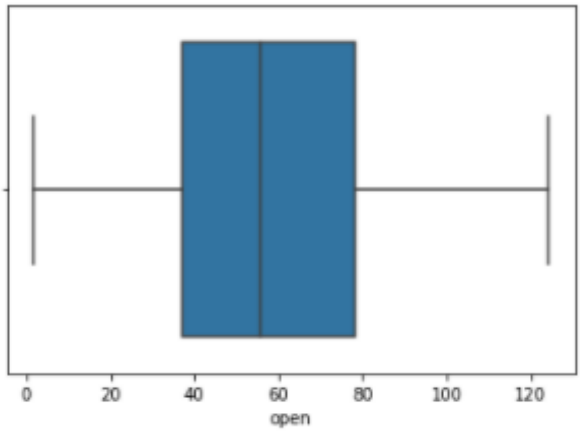
```
124.31500000000003 93.965
```

```
df1 = df[df['open']<upper]
df1.shape
```

```
(533292, 7)
```

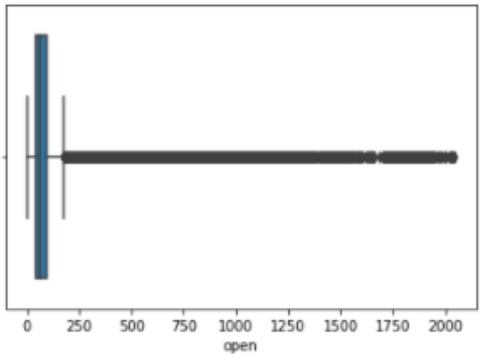
```
sns.boxplot(df1['open'])
```

```
<AxesSubplot:xlabel='open'>
```




```
sns.boxplot(df['open'])
```

```
<AxesSubplot:xlabel='open'>
```



- High

```
Q1 = df['high'].quantile(0.35)
Q3 = df['high'].quantile(0.65)
print(Q1,Q3)
iqr=Q3-Q1
iqr
```

```
48.89999999999999 79.47

30.570000000000007
```

```
upper = Q3+1.5*iqr
lower = Q1+1.5*iqr
print(upper,lower)
```

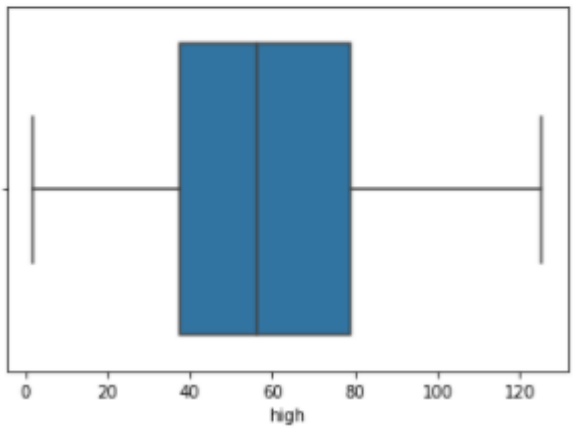
```
125.32500000000002 94.755
```

```
df1 = df[df['high']<upper]
df1.shape
```

```
(533191, 7)
```

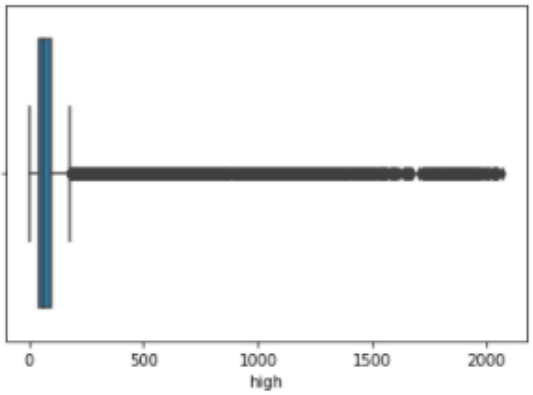
```
sns.boxplot(df1['high'])
```

```
<AxesSubplot:xlabel='high'>
```




```
sns.boxplot(df['high'])
```

```
<AxesSubplot:xlabel='high'>
```



- Low

```
Q1 = df['low'].quantile(0.35)
Q3 = df['low'].quantile(0.65)
print(Q1,Q3)
iqr=Q3-Q1
iqr
```

```
47.98 78.1

30.119999999999997
```

```
upper = Q3+1.5*iqr
lower = Q1+1.5*iqr
print(upper,lower)
```

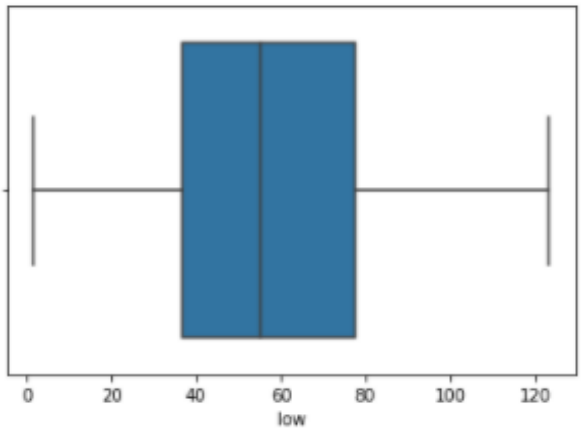
```
123.27999999999999 93.16
```

```
df1 = df[df['low']<upper]
df1.shape
```

```
(533387, 7)
```

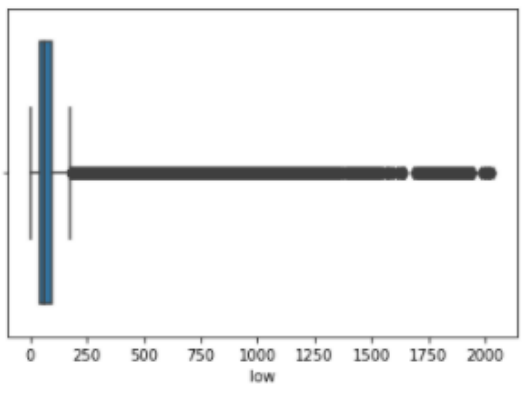
```
sns.boxplot(df1['low'])
```

```
<AxesSubplot:xlabel='low'>
```



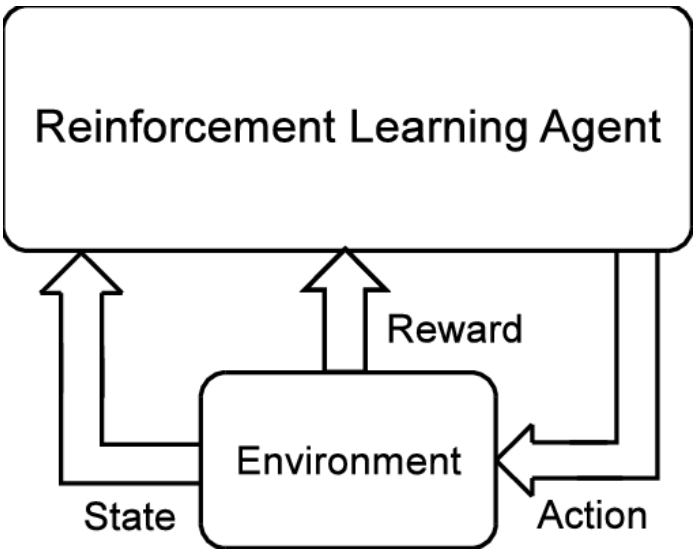

```
sns.boxplot(df['low'])
```

```
<AxesSubplot:xlabel='low'>
```



**CREATE AN ENVIRONMENT**

In the reinforcement learning problem, the environment is a critical component. It's critical to have a solid grasp of the underlying world with which the RL agent will interact. This aids us in developing the best design and learning strategy for the agent.




The reinforcement learning task is intended to be a simple framing of the challenge of learning from interaction in order to accomplish a goal. The agent is the learner and decision-maker. The environment is the object it interacts with, and it consists of everything outside the agent. The agent chooses actions, and the environment reacts to those actions, presenting the agent with new scenarios. The environment also produces rewards, which are unique numerical values that the agent attempts to maximise over time.

```
class Environment1:

    def __init__(self, df, history_t=90):
        self.data = df
        self.history_t = history_t
        self.reset()

    def reset(self):
        self.t = 0
        self.done = False
        self.profits = 0
        self.positions = []
        self.position_value = 0
        self.history = [0 for _ in range(self.history_t)]
        return [self.position_value] + self.history # obs

    def step(self, act):
        reward = 0

        # act = 0: stay, 1: buy, 2: sell
        if act == 1:
            self.positions.append(self.data.iloc[self.t, :]['close'])
        elif act == 2: # sell
            if len(self.positions) == 0:
                reward = -1
            else:
                profits = 0
                for p in self.positions:
                    profits += (self.data.iloc[self.t, :]['close'] - p)
                reward += profits
                self.profits += profits
                self.positions = []

        # set next time
        self.t += 1
        self.position_value = 0
        for p in self.positions:
            self.position_value += (self.data.iloc[self.t, :]['close'] - p)
        self.history.pop(0)
        self.history.append(self.data.iloc[self.t, :]['close'] - self.data.iloc[(self.t-1), :]
['close'])

        # clipping reward
        if reward > 0:
            reward = 1
        elif reward < 0:
            reward = -1

        return [self.position_value] + self.history, reward, self.done # obs, reward, done
```

```
env = Environment1(train)
print(env.reset())
for _ in range(3):
    pact = np.random.randint(3)
    print(env.step(pact))
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 13.52
9999999999998], -1, False)
([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 13.52999
9999999998, 7.380000000000003], -1, False)
([3.8999999999999986, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 13.529999999999998, 7.380000000000003, 3.8999999999999986], 0, False)
```

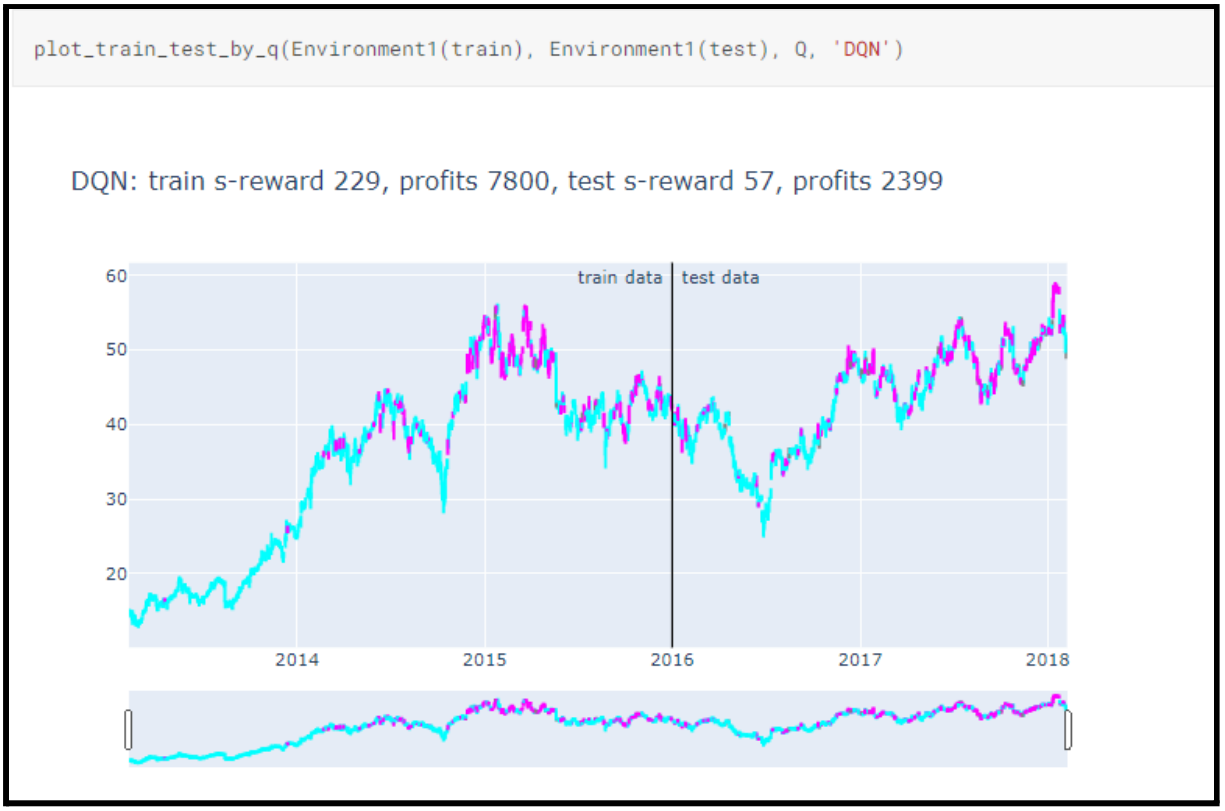
```
Q, total_losses, total_rewards = train_dqn(Environment1(train))
```

```
/opt/conda/lib/python3.7/site-packages/numpy/core/_asarray.py:83: VisibleDeprecationWarning:

Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuple
s-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you
must specify 'dtype=object' when creating the ndarray
```

| | | | | | |
|---|---|---|---|---|---|
| 5 | 0.0999999999999992 | 4715 | -36.4 | 1269.1880589734762 | 58.75414991378784 |
| 10 | 0.0999999999999992 | 9430 | 68.8 | 106.02257219804451 | 62.61797070503235 |
| 15 | 0.0999999999999992 | 14145 | 92.4 | 88.00889271469786 | 63.70128417015076 |
| 20 | 0.0999999999999992 | 18860 | 105.2 | 81.62211509905755 | 64.25300574302673 |
| 25 | 0.0999999999999992 | 23575 | 130.8 | 67.72250651605427 | 65.3246157169342 |
| 30 | 0.0999999999999992 | 28290 | 128.8 | 64.84192206077277 | 64.26934337615967 |
| 35 | 0.0999999999999992 | 33005 | 109.2 | 37.99149908022955 | 65.57610964775085 |
| 40 | 0.0999999999999992 | 37720 | 102.0 | 30.748561142757534 | 66.24323463439941 |
| 45 | 0.0999999999999992 | 42435 | 100.0 | 30.852891511656345 | 67.29631352424622 |
| 50 | 0.0999999999999992 | 47150 | 132.0 | 39.443355267494915 | 65.53461694717407 |
| 55 | 0.0999999999999992 | 51865 | 137.6 | 38.289829121623185 | 65.3227026462555 |
| 60 | 0.0999999999999992 | 56580 | 161.4 | 42.372054285090414 | 64.54252576828003 |

```
plot_train_test_by_q(Environment1(train), Environment1(test), Q, 'DQN')
```




**MODEL EVALUATION**

Model evaluation is a step in the model creation process that is often overlooked. This is the stage where the model's performance is determined. As a result, it's important to think about the model's results in terms of every conceivable assessment technique. Using various approaches can result in a variety of viewpoints.

**MODEL DEPLOYMENT**

The project is hosted on Heroku which is a cloud Platform as a container-based Service (PaaS). Heroku is used by developers to launch, manage, and grow contemporary programs. Heroku is an open-source software platform for machine learning and data science that makes it simple to develop and publish attractive, bespoke web apps. The benefit of web apps is that they are platform agnostic and may be operated by anybody with an Internet connection. Their code is run on a back-end server, which processes incoming requests and answers using a common protocol that all browsers can understand.

Necessary Files:

1. app.py
2. requirement.txt - Contains a list of all the dependencies that your code requires in order to function properly.
3. Procfile - In an app, a Procfile is a list of process types.
4. setup.sh

requirement.txt

This file contains all of the libraries that must be installed in order for the project to run. This file may be manually produced by walking through the project and identifying all of the libraries used. However, we used the pipreqs module instead, which generated a requirement.txt file for us.

```
plotly==5.3.1
numpy==1.21.2
streamlit==0.88.0
pandas==1.3.2
```

Procfile and setup.sh

Using these two files, we tell Heroku the needed commands to start our application. In the setup.sh file we created a streamlit folder with credentials.toml and a config.toml file.

Following is the command to be pasted in setup.sh file.

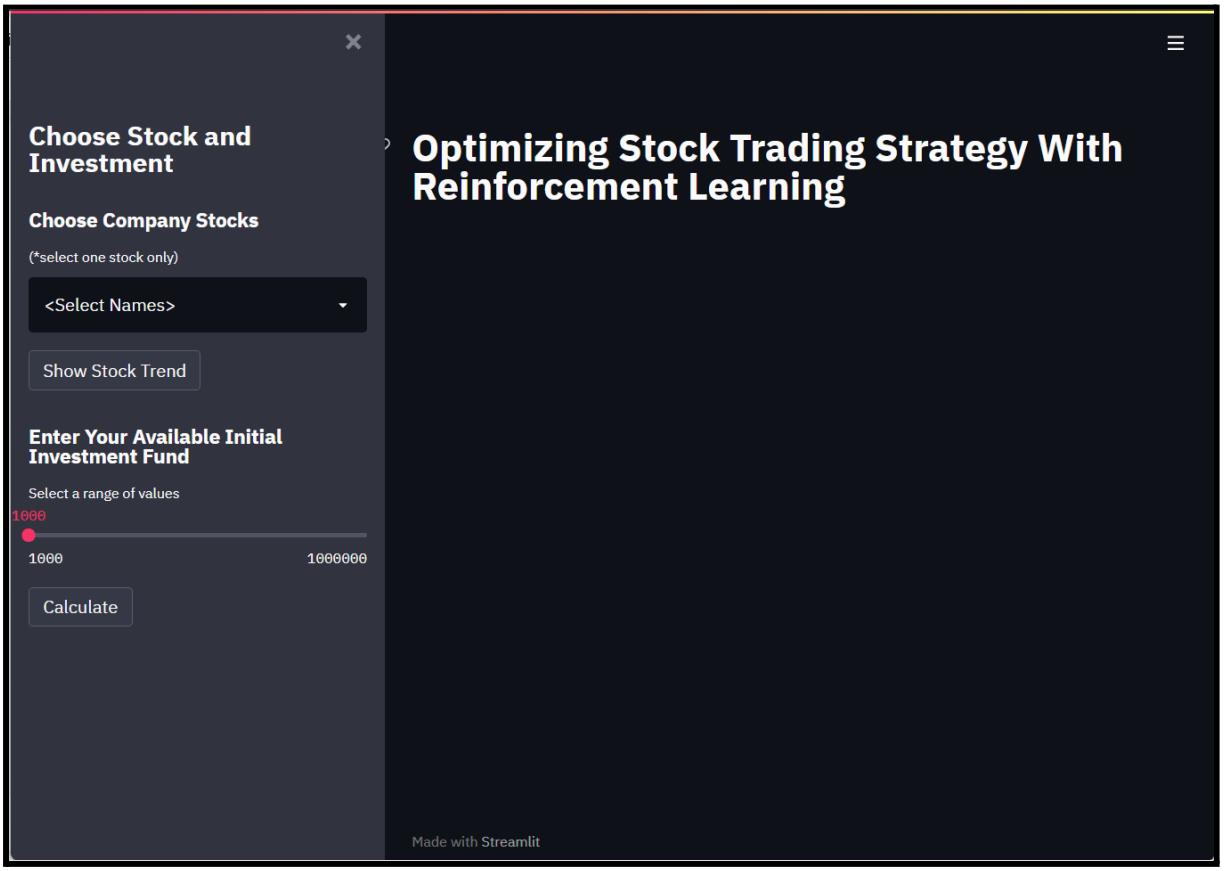
```
mkdir -p ~/.streamlit/
echo "\
[server]\n\
headless = true\n\
port = $PORT\n\
enableCORS = false\n\
\n\
" > ~/.streamlit/config.toml
```

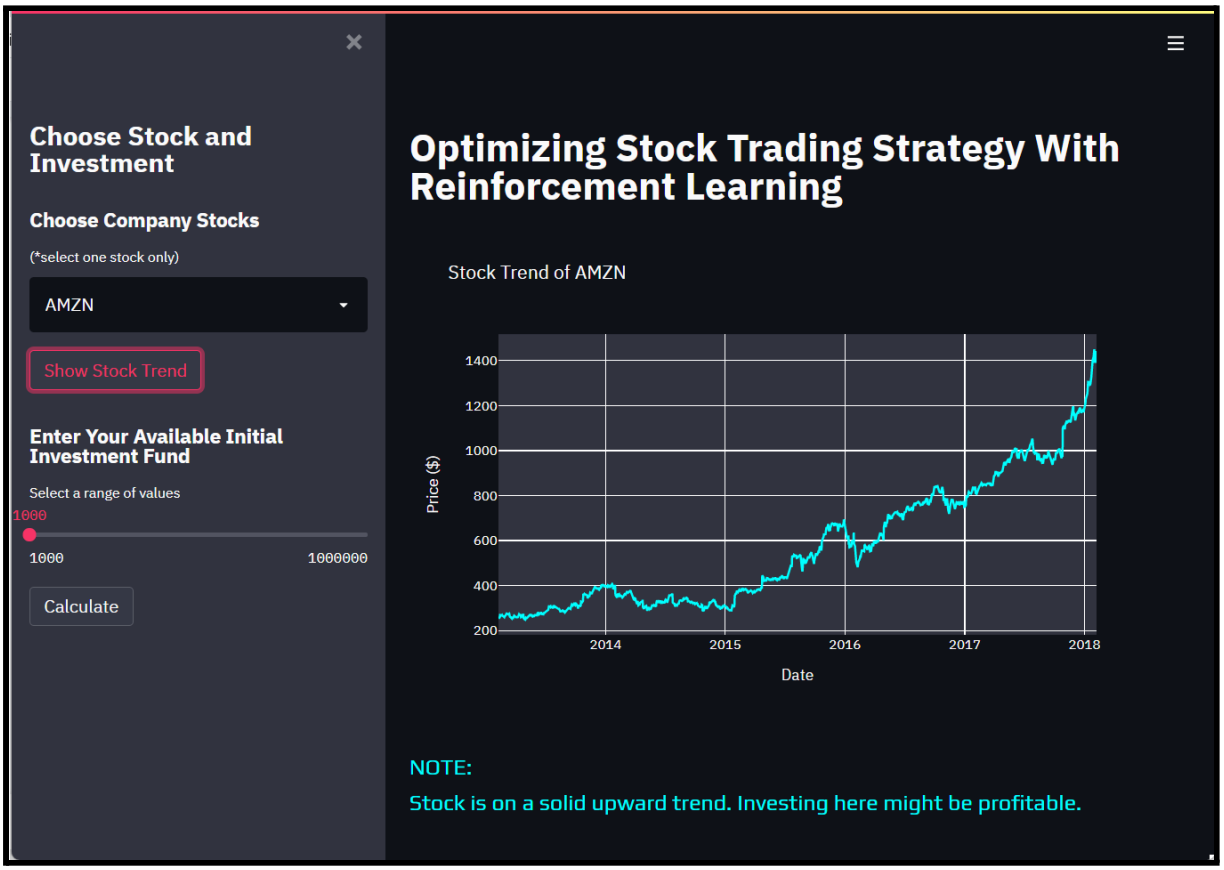
The setup.sh script is first performed using the Procfile, and then the application is executed using streamlit run.

Procfile looks like this.

```
web: sh setup.sh && streamlit run Stock-RL.py
```

## RESULTS


Choose Stock and Investment
Choose Company Stocks
(*select one stock only)
<Select Names>
Show Stock Trend
Enter Your Available Initial Investment Fund
Select a range of values
1000
1000
1000000
Calculate
Optimizing Stock Trading Strategy With Reinforcement Learning
Made with Streamlit


Choose Stock and Investment
Choose Company Stocks
(*select one stock only)
AMZN
Show Stock Trend
Enter Your Available Initial Investment Fund
Select a range of values
1000
1000
1000000
Calculate
Optimizing Stock Trading Strategy With Reinforcement Learning
Stock Trend of AMZN
Price ($)
1400
1200
1000
800
600
400
200
2014
2015
2016
2017
2018
Date
NOTE:
Stock is on a solid upward trend. Investing here might be profitable.

**WEB APPLICATION -** https://stock-trading-with-rl.herokuapp.com