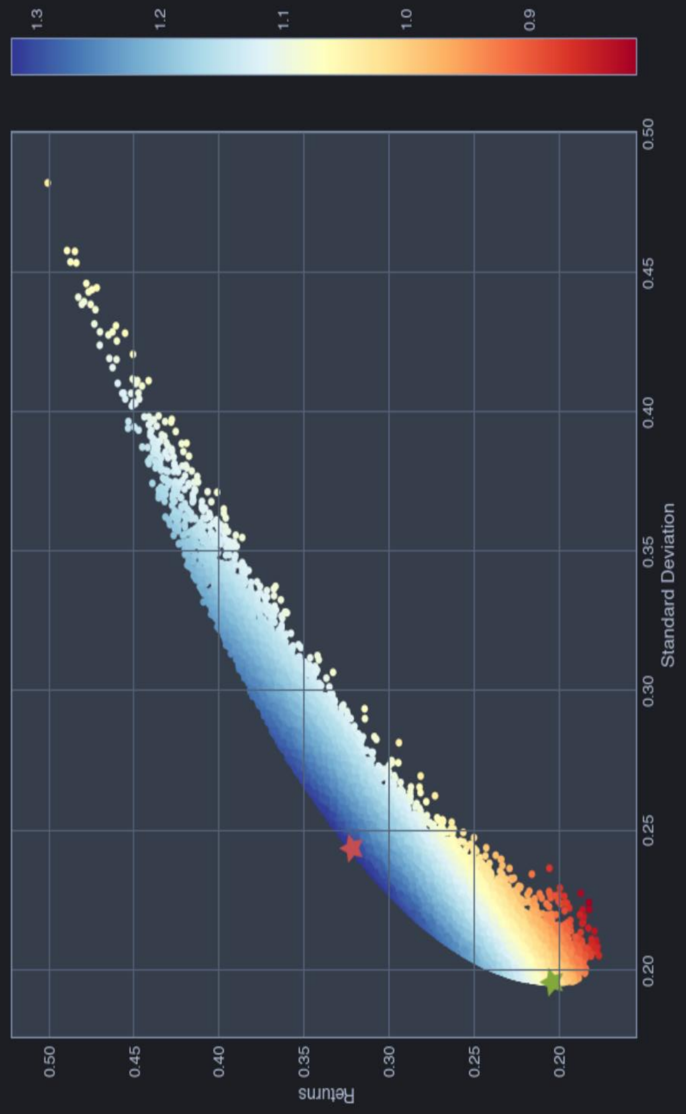
Engineering Optimization
J Component

Topic: **Optimization in Finance**

1.Portfolio Optimization using Markowitz model
2. Predicting Stock Prices using Reinforcement Learning

Members:

1.Lakshmi K Sathyan : 20BEE0045
2.Shruthi Suresh Nair : 20BEE0053



Problem Statements

Problem:

- Which assets should be included in an optimized portfolio, and how should one change its composition according to the market?
- How to detect opportunities in the different assets in the market, and take profit by trading them?

Solutions:

- Portfolio optimisation using Markowitz model
- Prediction of stock price to get maximum profit using Reinforcement learning

What is portfolio optimization?

- Portfolio optimization is the process of selecting the best portfolio (asset distribution), out of the set of all portfolios being considered, according to some objective.
- The objective typically maximizes factors such as expected return, and minimizes costs like financial risk. Factors being considered may range from tangible (such as assets, liabilities, earnings or other fundamentals) to intangible (such as selective divestment).

Approach

Portfolio optimization often takes place in two stages:

- optimizing weights of asset classes to hold, and
- optimizing weights of assets within the same asset class.

Harry Markowitz developed the "critical line method", a general procedure for quadratic programming that can handle additional linear constraints and upper and lower bounds on holdings. Moreover, in this context, the approach provides a method for determining the entire set of efficient portfolios.

Markowitz Model

- In finance, the Markowitz model is a portfolio optimization model that assists in the selection of the most efficient portfolio by analyzing various possible portfolios of the given securities.
- Here, by choosing securities that do not 'move' exactly together, the HM model shows investors how to reduce their risk.
- The HM model is also called mean-variance model due to the fact that it is based on expected returns (mean) and the standard deviation (variance) of the various portfolios.

Markowitz Model

Markowitz made the following assumptions while developing the HM model:

- Risk of a portfolio is based on the variability of returns from said portfolio.
- An investor is risk averse.
- An investor prefers to increase consumption.
- Analysis is based on single period model of investment.
- The investor's utility function is concave and increasing, due to their risk aversion and consumption preference.
- An investor is rational in nature

To choose the best portfolio from a number of possible portfolios, each with different return and risk, two separate decisions are to be made, detailed in the below sections:

- Determination of a set of efficient portfolios.
- Selection of the best portfolio out of the efficient set.

Methodology

1. Determination of efficient set

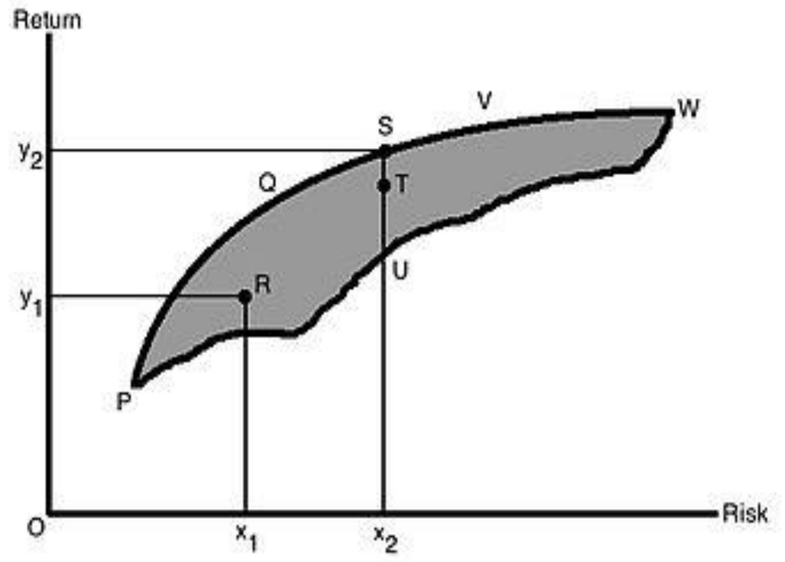
A portfolio that gives maximum return for a given risk, or minimum risk for given return is an efficient portfolio. Thus, portfolios are selected as follows:

(a) From the portfolios that have the same return, the investor will prefer the portfolio with lower risk

(b) From the portfolios that have the same risk level, an investor will prefer the portfolio with higher rate of return.

Any investor would like to have higher return. And as they are risk averse, they want to have lower risk. In the Figure , the shaded area PVWP includes all the possible securities an investor can invest in. The efficient portfolios are the ones that lie on the boundary of PQVW. For example, at risk level x2, there are three portfolios S, T, U. But portfolio S is called the **efficient portfolio** as it has the highest return, y2, compared to T and U[needs dot]. All the portfolios that lie on the boundary of PQVW are efficient portfolios for a given risk level.

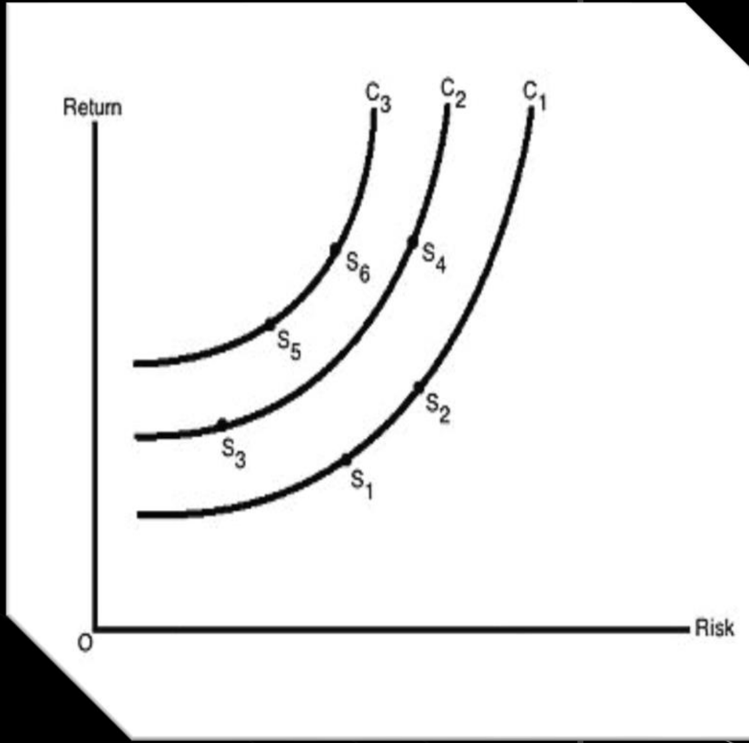
The boundary PQVW is called the **Efficient Frontier**. All portfolios that lie below the Efficient Frontier are not good enough because the return would be lower for the given risk. Portfolios that lie to the right of the Efficient Frontier would not be good enough, as there is higher risk for a given rate of return. All portfolios lying on the boundary of PQVW are called Efficient Portfolios. The Efficient Frontier is the same for all investors, as all investors want maximum return with the lowest possible risk and they are risk averse.



2.Choosing the best portfolio

For selection of the optimal portfolio or the best portfolio, the risk-return preferences are analyzed. An investor who is highly risk averse will hold a portfolio on the lower left hand of the frontier, and an investor who isn’t too risk averse will choose a portfolio on the upper portion of the frontier.

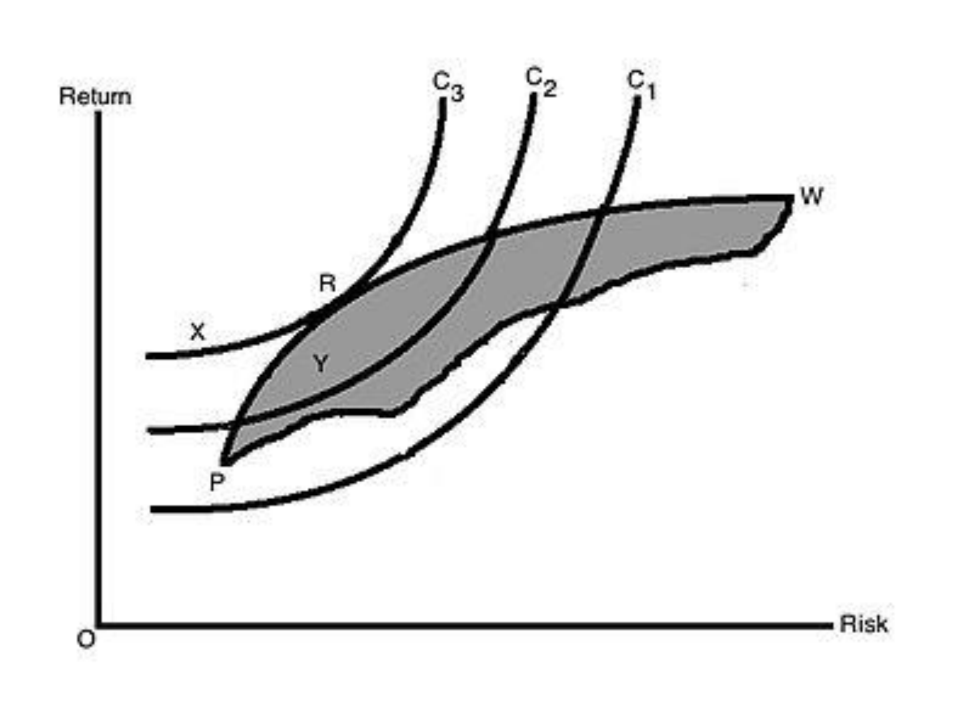
Figure on the right shows the risk-return indifference curve for the investors. Indifference curves C1, C2 and C3 are shown. Each of the different points on a particular indifference curve shows a different combination of risk and return, which provide the same satisfaction to the investors. Each curve to the left represents higher utility or satisfaction. The goal of the investor would be to maximize their satisfaction by moving to a curve that is higher. An investor might have satisfaction represented by C2, but if their satisfaction/utility increases, the investor then moves to curve C3. Thus, at any point of time, an investor will be indifferent between combinations S1 and S2, or S5 and S6.



The investor's optimal portfolio is found at the point of tangency of the efficient frontier with the indifference curve. This point marks the highest level of satisfaction the investor can obtain. This is shown in Figure on the right.

R is the point where the efficient frontier is tangent to indifference curve C3, and is an efficient portfolio. With this portfolio, the investor will get highest satisfaction as well as best risk-return combination (a portfolio that provides the highest possible return for a given amount of risk).

Any other portfolio, say X, isn't the optimal portfolio even though it lies on the same indifference curve as it is outside the feasible portfolio available in the market. Portfolio Y is also not optimal as it does not lie on the best feasible indifference curve, even though it is a feasible market portfolio. Another investor having other sets of indifference curves might have some different portfolio as their best/optimal portfolio.



In the market for portfolios that consists of risky and risk-free securities, the CML represents the equilibrium condition. The Capital Market Line says that the return from a portfolio is the risk-free rate plus risk premium. Risk premium is the product of the market price of risk and the quantity of risk, and the risk is the standard deviation of the portfolio.

The CML equation is :

RP = IRF + (RM – IRF)σP/σM

where,

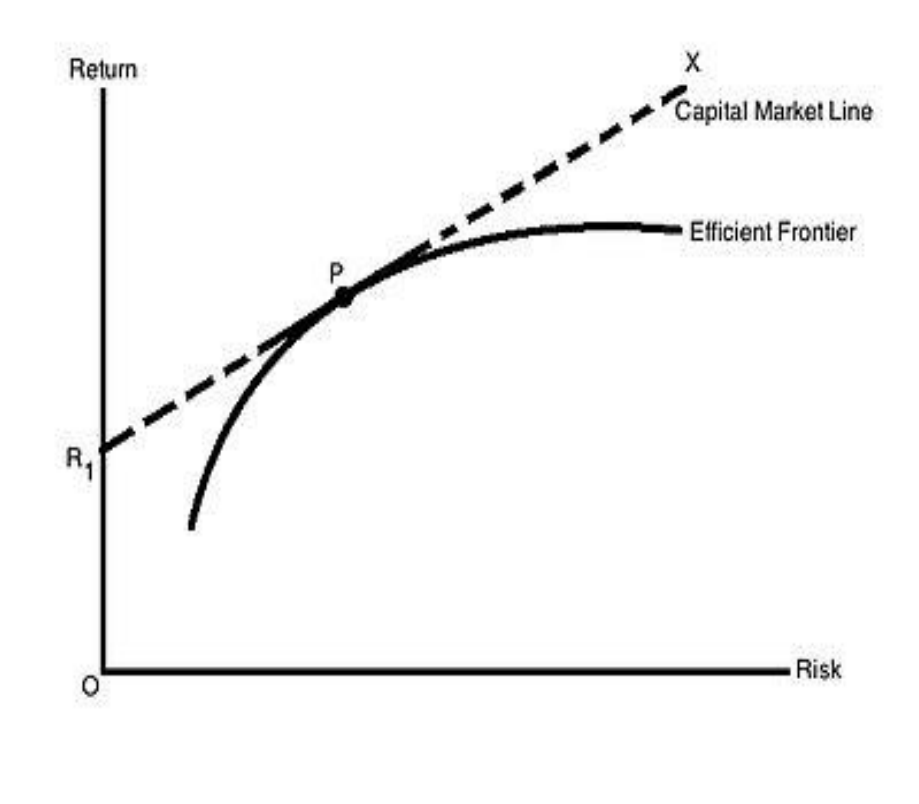
RP = expected return of portfolio

RM = return on the market portfolio

IRF = risk-free rate of interest

σM = standard deviation of the market portfolio

σP = standard deviation of portfolio



(RM – IRF)/σM is the slope of CML. (RM – IRF) is a measure of the risk premium, or the reward for holding risky portfolio instead of risk-free portfolio. σM is the risk of the market portfolio. Therefore, the slope measures the reward per unit of market risk.

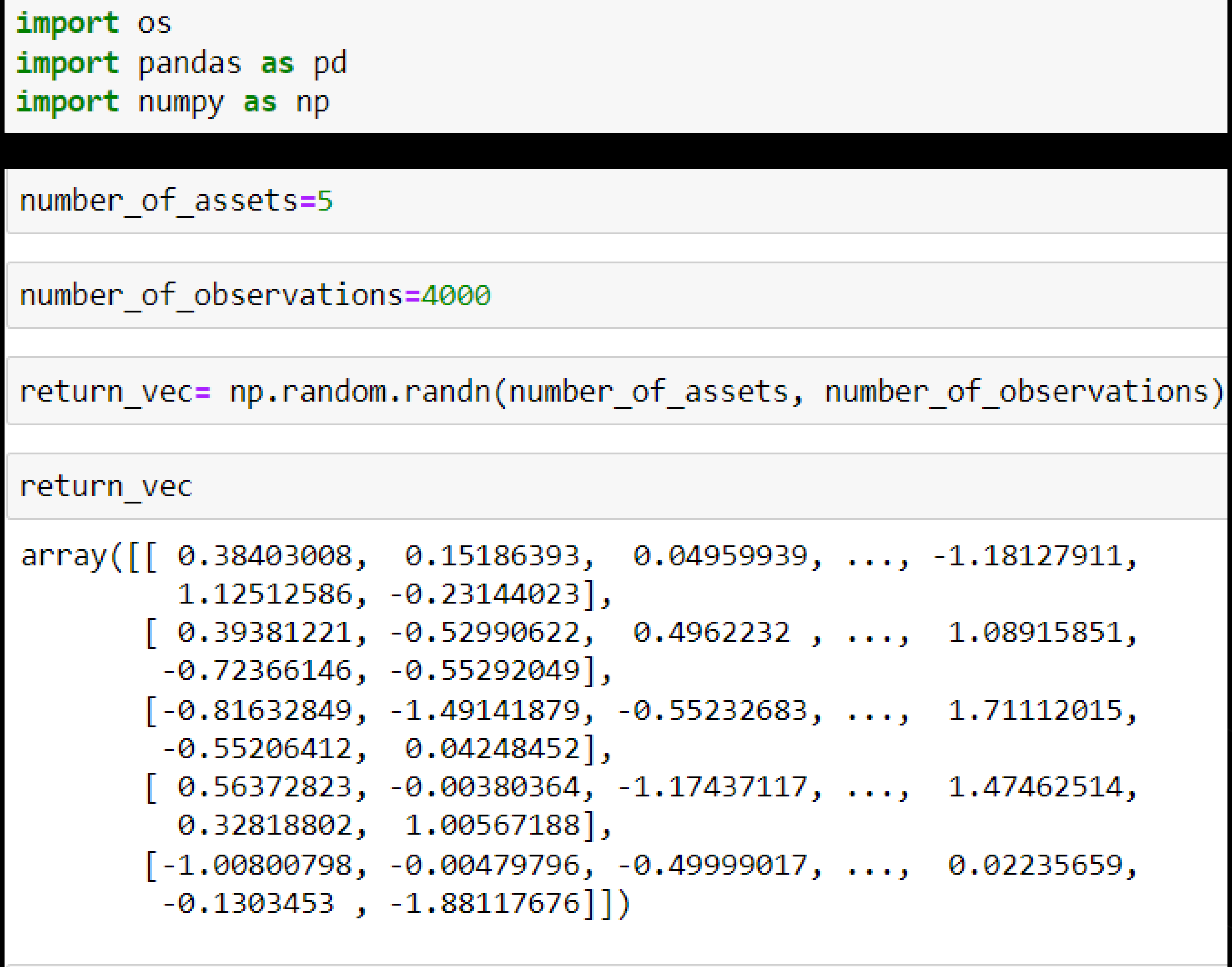
The characteristic features of CML are:

1. At the tangent point, i.e. Portfolio P, is the optimum combination of risky investments and the market portfolio.

2. Only efficient portfolios that consist of risk free investments and the market portfolio P lie on the CML.

3. CML is always upward sloping as the price of risk has to be positive. A rational investor will not invest unless they know they will be compensated for that risk.

CODE IN PYTHON



```
import os
import pandas as pd
import numpy as np
```

```
number_of_assets=5
```

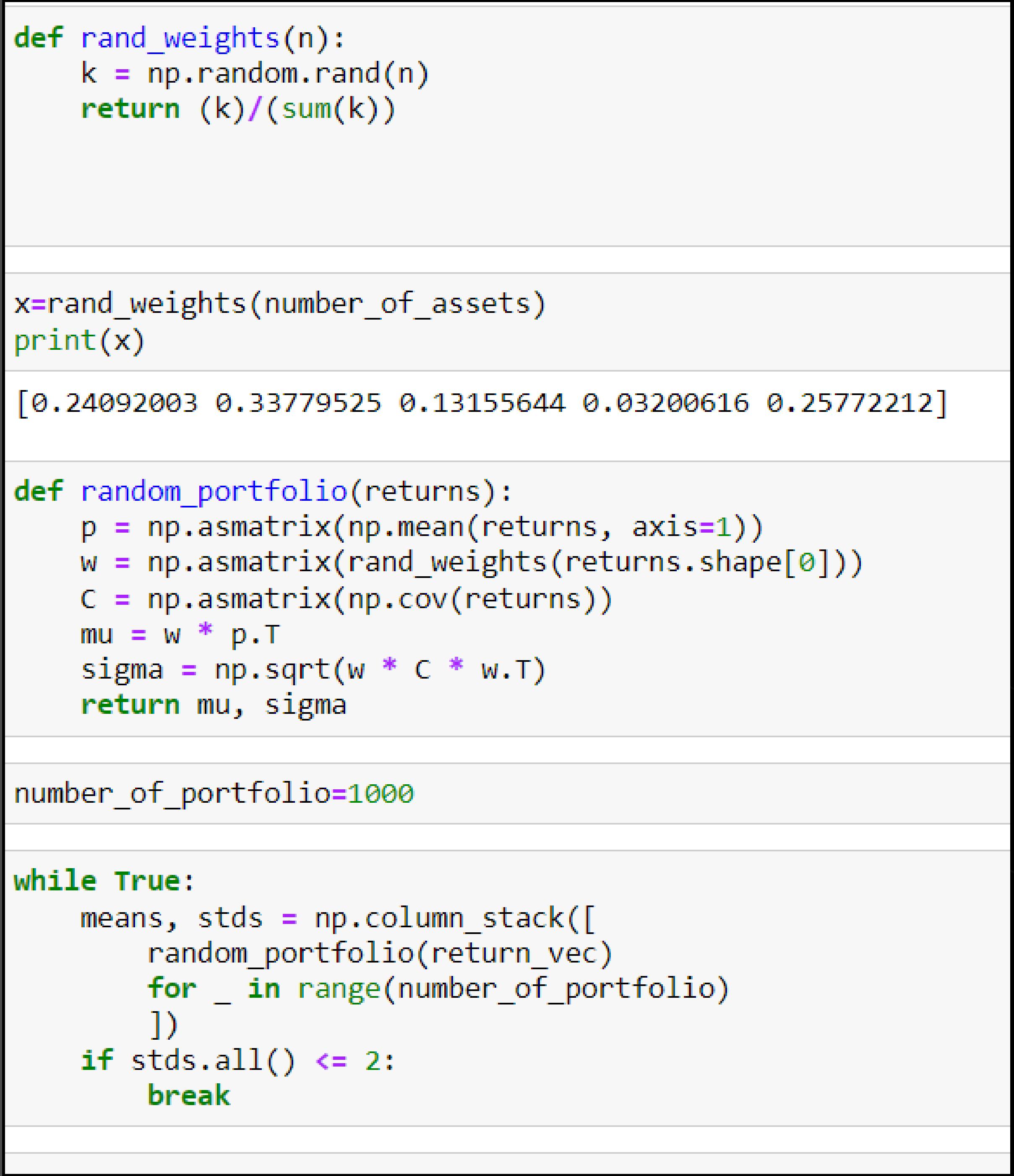
```
number_of_observations=4000
```

```
return_vec= np.random.randn(number_of_assets, number_of_observations)
```

```
return_vec
```

```
array([[ 0.38403008,  0.15186393,  0.04959939, ..., -1.18127911,
         1.12512586, -0.23144023],
       [ 0.39381221, -0.52990622,  0.4962232 , ...,  1.08915851,
        -0.72366146, -0.55292049],
       [-0.81632849, -1.49141879, -0.55232683, ...,  1.71112015,
        -0.55206412,  0.04248452],
       [ 0.56372823, -0.00380364, -1.17437117, ...,  1.47462514,
         0.32818802,  1.00567188],
       [-1.00800798, -0.00479796, -0.49999017, ...,  0.02235659,
        -0.1303453 , -1.88117676]])
```

- We start with importing a few modules, which we need later and produce a series of normally distributed returns.
- Assume that we have 5 assets, each with a return series of length 4000. We can use numpy.random.randn to sample returns from a normal distribution.
- These return series can be used to create a wide range of portfolios, which all have different returns and risks (standard deviation). We can produce a wide range of random weight vectors and plot those portfolios.



```python
def rand_weights(n):
    k = np.random.rand(n)
    return (k)/(sum(k))
```

```python
x=rand_weights(number_of_assets)
print(x)
```

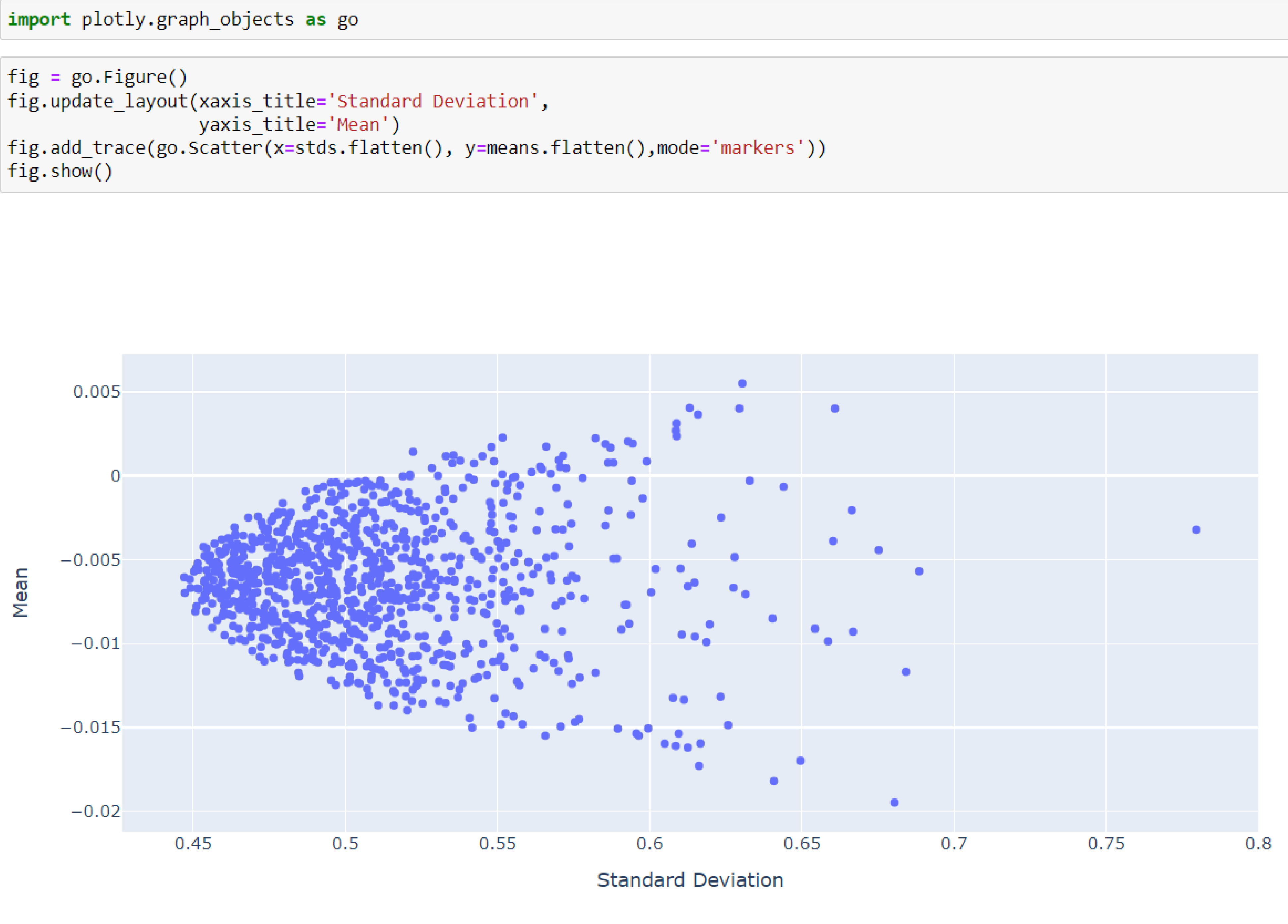
```
[0.24092003 0.33779525 0.13155644 0.03200616 0.25772212]
```

```python
def random_portfolio(returns):
    p = np.asmatrix(np.mean(returns, axis=1))
    w = np.asmatrix(rand_weights(returns.shape[0]))
    C = np.asmatrix(np.cov(returns))
    mu = w * p.T
    sigma = np.sqrt(w * C * w.T)
    return mu, sigma
```

```python
number_of_portfolio=1000
```

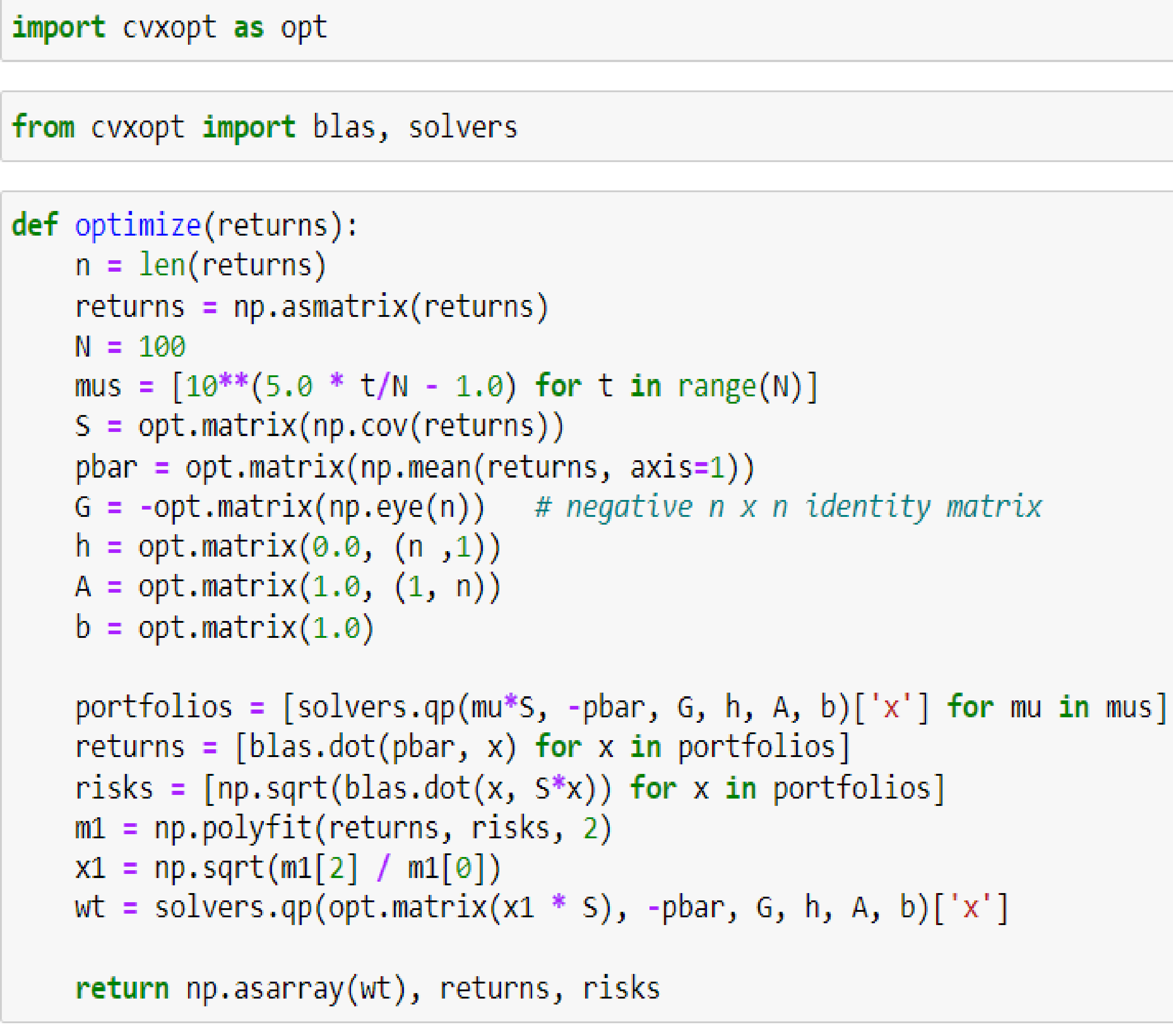
```python
while True:
    means, stds = np.column_stack([
        random_portfolio(return_vec)
        for _ in range(number_of_portfolio)
        ])
    if stds.all() <= 2:
        break
```

- Next, we evaluate how many of these random portfolios would perform. Towards this goal we are calculating the mean returns as well as the volatility (here we are using standard deviation). We have a filter that only allows to plot portfolios with a standard deviation of < 2 for better illustration.
- mu= P(T)*W => Expected return
- Sigma = sqrt(W(T)*C*W) => Standard deviation.
- P=> Vector of mean
- C=> Covariance Matrix
- W=> Weight vector of the portfolio
- Then we generate the mean returns and risk for 1000 portfolios



```
import plotly.graph_objects as go
```

```
fig = go.Figure()
fig.update_layout(xaxis_title='Standard Deviation',
                  yaxis_title='Mean')
fig.add_trace(go.Scatter(x=stds.flatten(), y=means.flatten(),mode='markers'))
fig.show()
```



```
import cvxopt as opt

from cvxopt import blas, solvers

def optimize(returns):
    n = len(returns)
    returns = np.asmatrix(returns)
    N = 100
    mus = [10**(5.0 * t/N - 1.0) for t in range(N)]
    S = opt.matrix(np.cov(returns))
    pbar = opt.matrix(np.mean(returns, axis=1))
    G = -opt.matrix(np.eye(n))   # negative n x n identity matrix
    h = opt.matrix(0.0, (n ,1))
    A = opt.matrix(1.0, (1, n))
    b = opt.matrix(1.0)

    portfolios = [solvers.qp(mu*S, -pbar, G, h, A, b)['x'] for mu in mus]
    returns = [blas.dot(pbar, x) for x in portfolios]
    risks = [np.sqrt(blas.dot(x, S*x)) for x in portfolios]
    m1 = np.polyfit(returns, risks, 2)
    x1 = np.sqrt(m1[2] / m1[0])
    wt = solvers.qp(opt.matrix(x1 * S), -pbar, G, h, A, b)['x']

    return np.asarray(wt), returns, risks
```

- Upon plotting the portfolios, we observe that they form a characteristic parabolic shape called the ‘Markowitz bullet‘ with the boundaries being called the ‘efficient frontier‘, where we have the lowest variance for a given expected.

- Once we have a good representation of our portfolios as the blue dots show, we can calculate the efficient frontier line in Markowitz-style. This is done by minimizing => W(T)*C*W (risk) whilst keeping the sum of the weights equal to 1.

- To find the minimum variance, we use cvxopt, a convex solver which can be easily downloaded.

- R(T)*W= mu

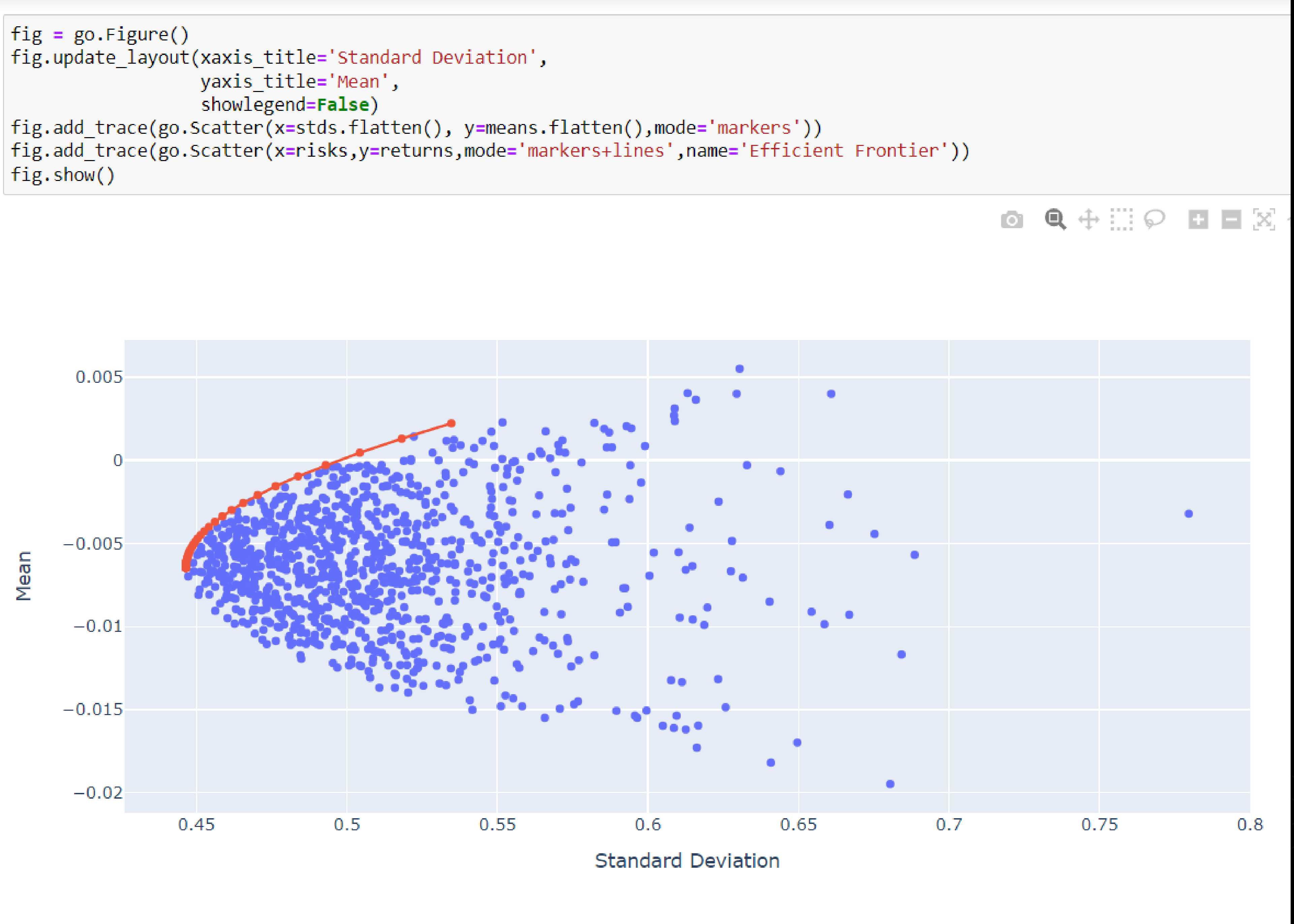
According to CVXOPT API, we can solve the optimization problem in this form:

$$min \frac{1}{2} x^T P x + q^T x$$

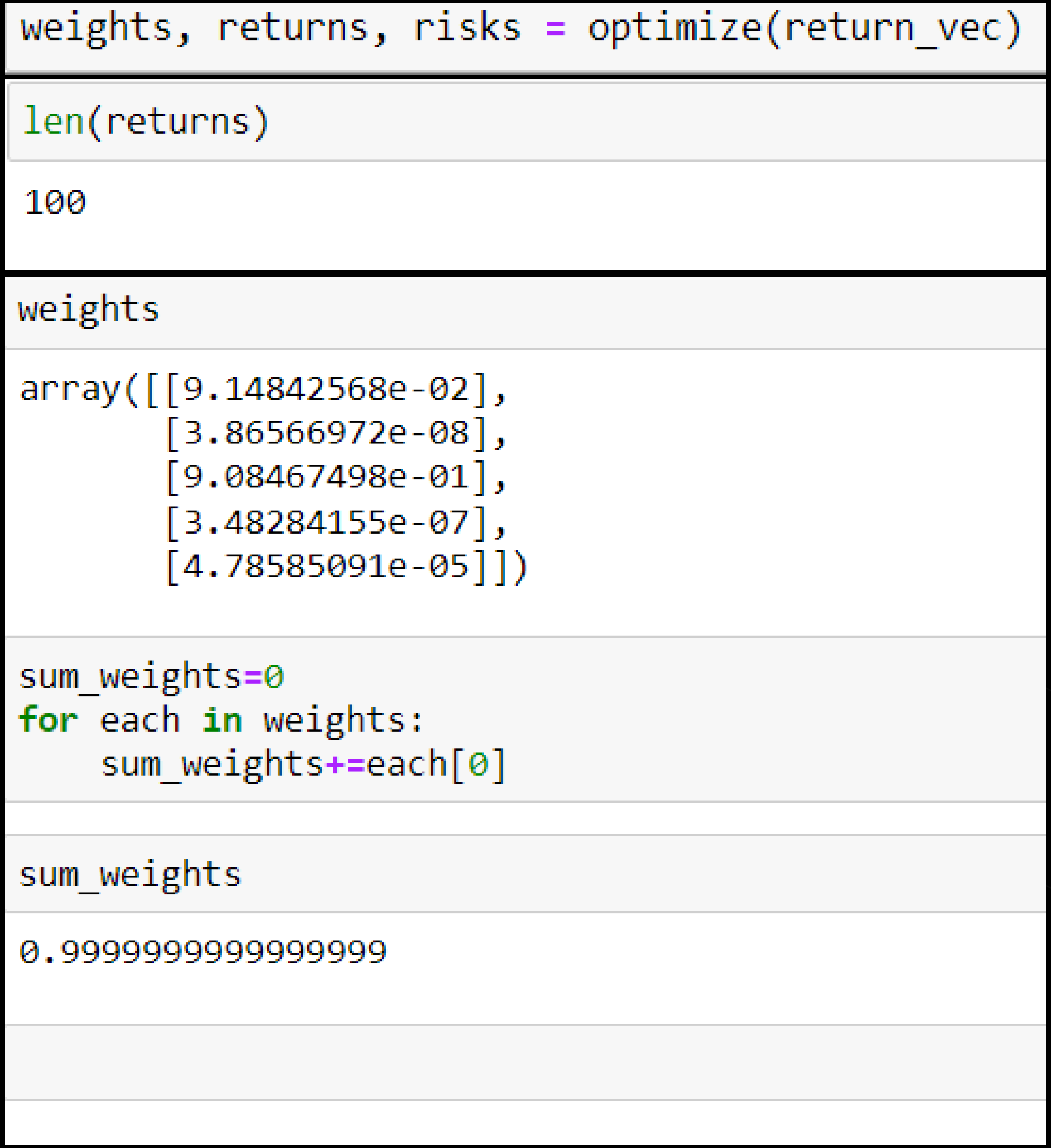
$$\text{s.t. } Gx \leq h$$

$$Ax = b$$

It is solving a minimization problem, with two types of linear constraints. One is an inequality constraint, and another is an equality constraint. To use the package to solve for the best x, that minimizing the object function, under the linear constraints, we just need to transform the specific question to identify matrics P, q, G, h, A, b.



```
fig = go.Figure()
fig.update_layout(xaxis_title='Standard Deviation',
                  yaxis_title='Mean',
                  showlegend=False)
fig.add_trace(go.Scatter(x=stds.flatten(), y=means.flatten(),mode='markers'))
fig.add_trace(go.Scatter(x=risks,y=returns,mode='markers+lines',name='Efficient Frontier'))
fig.show()
```



```
weights, returns, risks = optimize(return_vec)
```

```
len(returns)
```

```
100
```

```
weights
```

```
array([[9.14842568e-02],
       [3.86566972e-08],
       [9.08467498e-01],
       [3.48284155e-07],
       [4.78585091e-05]])
```

```
sum_weights=0
for each in weights:
    sum_weights+=each[0]
```

```
sum_weights
```

```
0.9999999999999999
```

- In red we see the optimal portfolios for each of the desired returns (i.e. the mus).
- In addition, we get the one optimal portfolio returned

Predicting Stock Prices using Reinforcement Learning

The stock market is an interesting medium to earn and invest money. It is also a lucrative option that increases your greed and leads to drastic decisions. This is majorly due to the volatile nature of the market. It is a gamble that can often lead to a profit or a loss. There is no proper prediction model for stock prices. The price movement is highly influenced by the demand and supply ratio.

There are multiple ways and method to predict the stock prices, in this project we use reinforcement learning for the same

What is Reinforcement Learning?

Reinforcement learning is another type of machine learning besides supervised and unsupervised learning. This is an agent-based learning system where the agent takes actions in an environment where the goal is to maximize the record. Reinforcement learning does not require the usage of labeled data like supervised learning.

Reinforcement learning works very well with less historical data. It makes use of the value function and calculates it on the basis of the policy that is decided for that action.

How can we predict stock market prices using reinforcement learning?

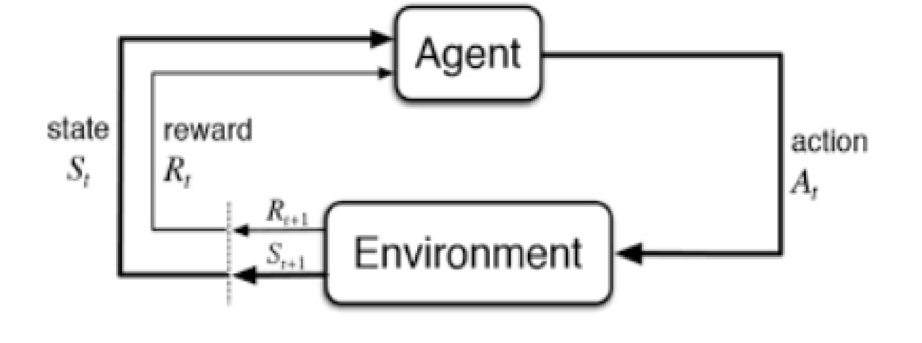
The concept of reinforcement learning can be applied to the stock price prediction for a specific stock as it uses the same fundamentals of requiring lesser historical data, working in an agent-based system to predict higher returns based on the current environment.

Steps for designing a reinforcement learning model is –

- Importing Libraries
- Create the agent who will make all decisions
- Define basic functions for formatting the values, sigmoid function, reading the data file, etc
- Train the agent
- Evaluate the agent performance

Defining the Reinforcement Learning Environment

- Agent – An Agent A that works in Environment E
- Action – Buy/Sell/Hold
- States – Data values
- Rewards – Profit / Loss



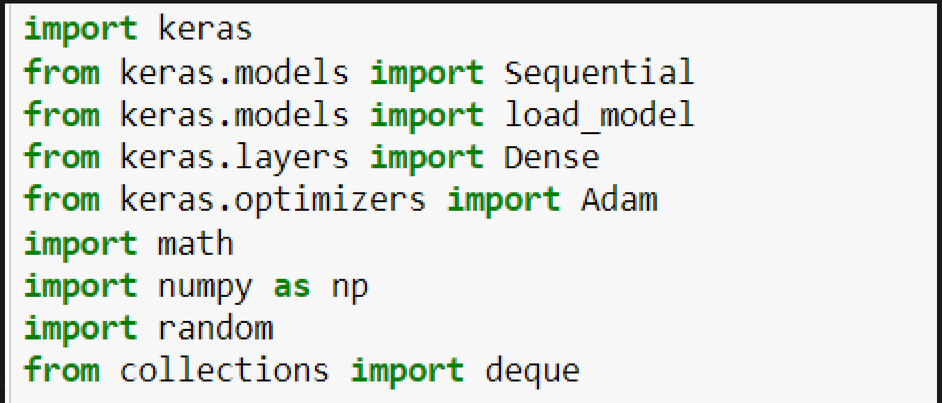
The Role of Q – Learning

Q-learning is a model-free reinforcement learning algorithm to learn the quality of actions telling an agent what action to take under what circumstances. Q-learning finds an optimal policy in the sense of **maximizing the expected value** of the total reward over any successive steps, starting from the current state.

CODING

1. Obtain the data from Yahoo finance website of any company of 5 years for training the model.

2. Import the required python libraries for modeling the neural network layers and the NumPy library for some basic operations.

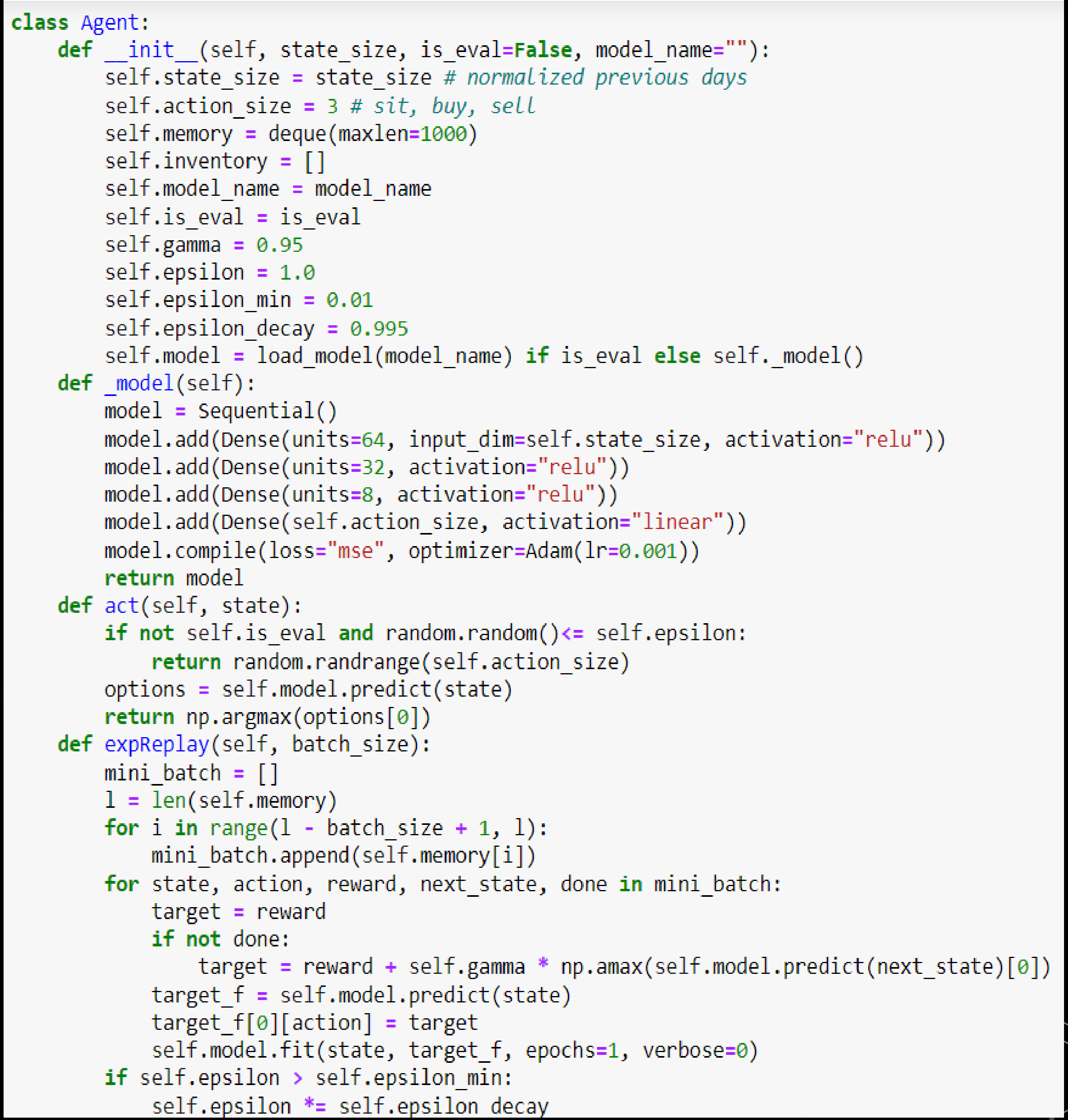


```python
import keras
from keras.models import Sequential
from keras.models import load_model
from keras.layers import Dense
from keras.optimizers import Adam
import math
import numpy as np
import random
from collections import deque
```

Creating the Agent

The Agent code begins with some basic initializations for the various parameters. Some static variables like gamma, epsilon, epsilon_min, and epsilon_decay are defined. These are threshold constant values that are used to drive the entire buying and selling process for stock and keep the parameters in stride. These min and decay values serve like threshold values in the normal distribution.

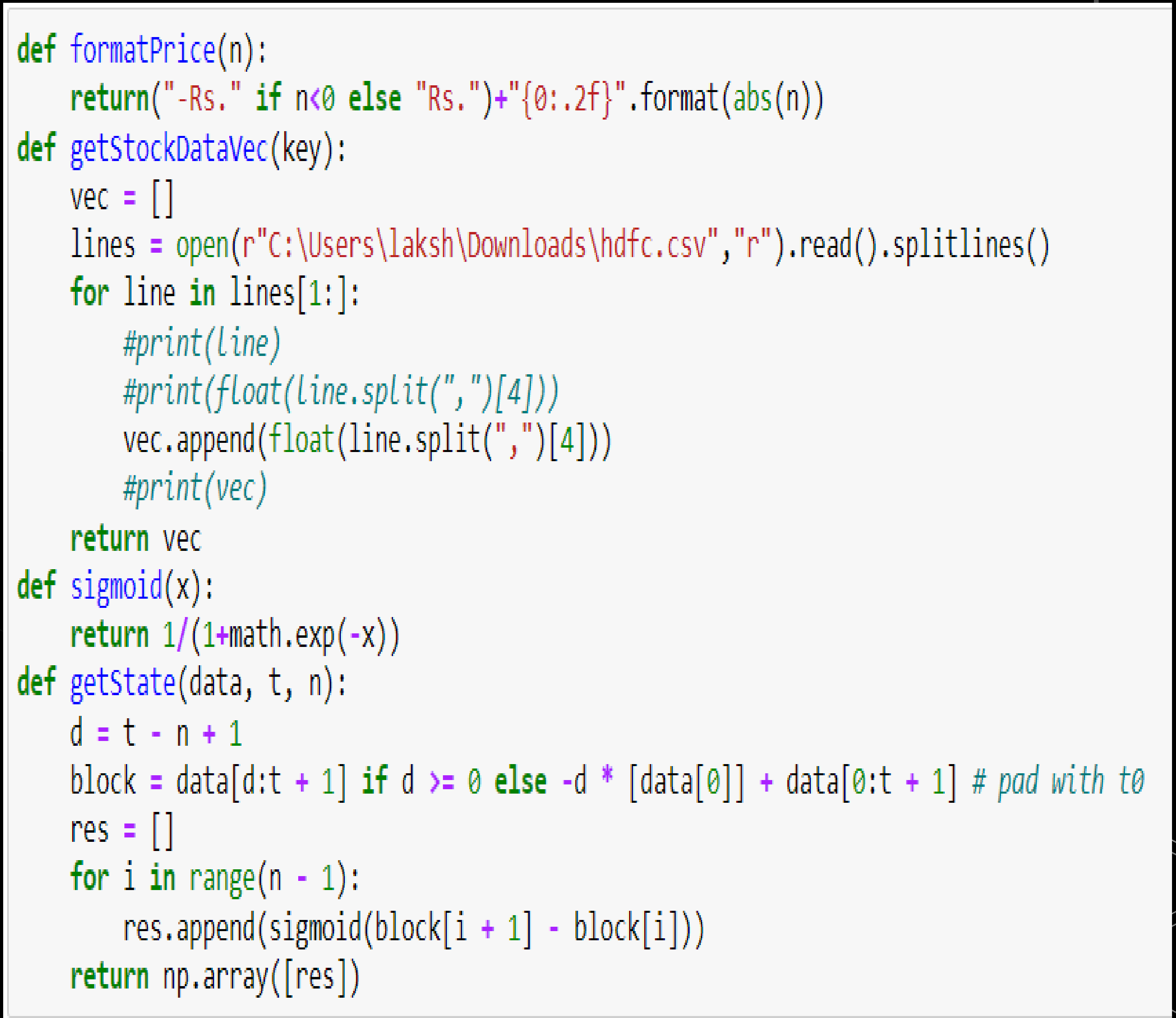
The agent designs the layered neural network model to take action of either buy, sell, or hold. This kind of action it takes by looking at its previous prediction and also the current environment state. The act method is used to predict the next action to be taken. If the memory gets full, there is another method called expReplay designed to reset the memory.



```
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(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
```

Define Basic Functions

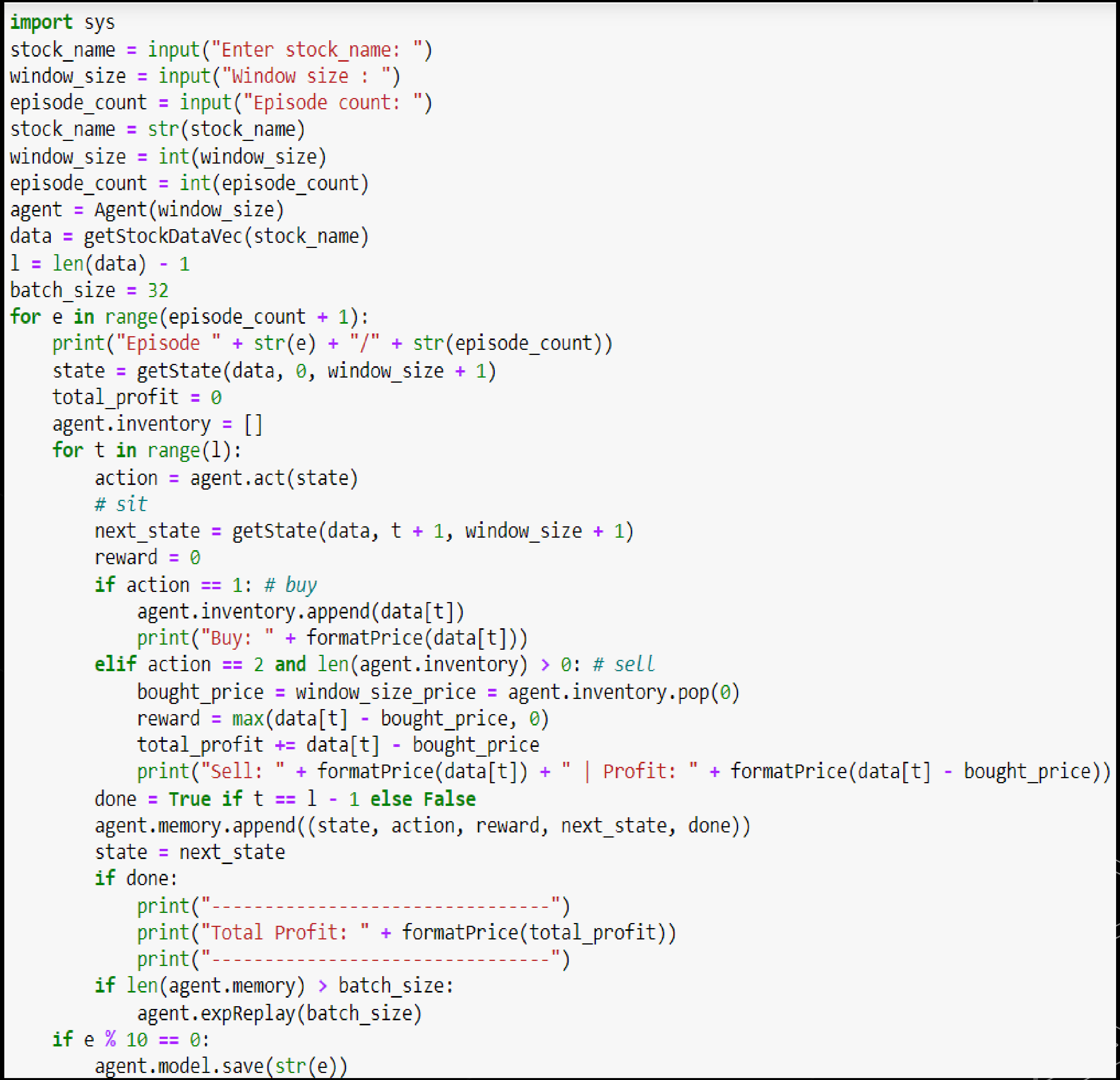
- The formatprice() is written to structure the format of the currency.
- The getStockDataVec() will bring the stock data into python.
- Defining the sigmoid function as a mathematical calculation or expression
- The getState() is coded in such a manner that it gives the current state of the data.



```
def formatPrice(n):
    return("-Rs." if n<0 else "Rs.")+"{0:.2f}".format(abs(n))
def getStockDataVec(key):
    vec = []
    lines = open(r"C:\Users\laksh\Downloads\hdfc.csv","r").read().splitlines()
    for line in lines[1:]:
        #print(line)
        #print(float(line.split(",")[4]))
        vec.append(float(line.split(",")[4]))
        #print(vec)
    return vec
def sigmoid(x):
    return 1/(1+math.exp(-x))
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])
```

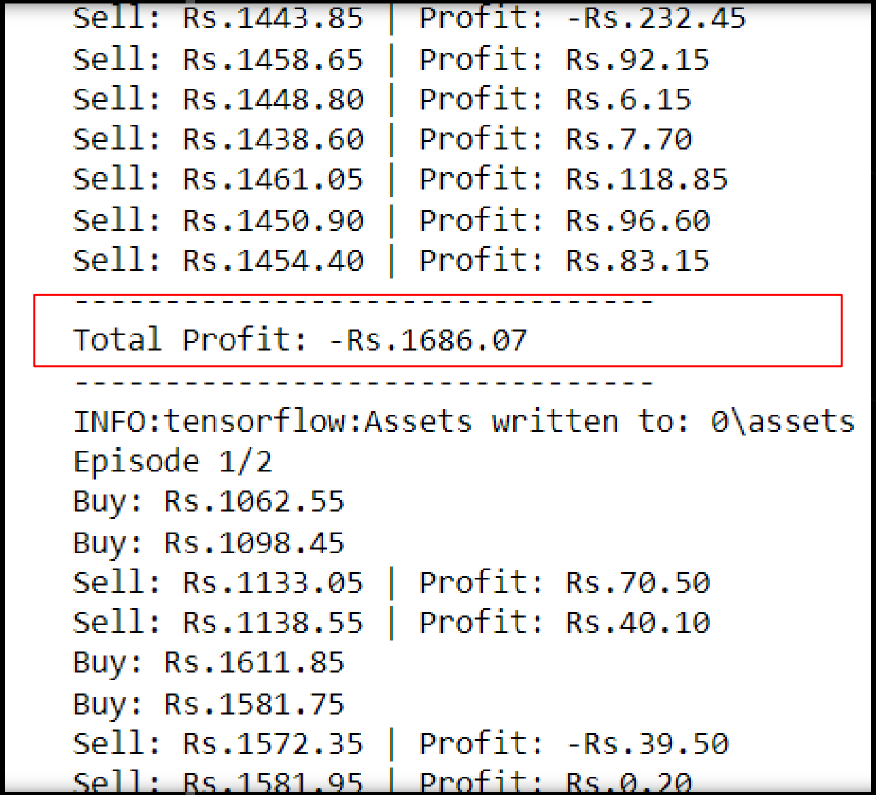
Training the Agent

- Depending on the action that is predicted by the model, the buy/sell call adds or subtracts money. It trains via multiple episodes which are the same as epochs in deep learning.
- The model is then saved subsequently.
- Once the model has been trained depending on new data, we can test the model for the profit/loss that the model is giving. We can accordingly evaluate the credibility of the model.
- The model function under Agent class is written for evaluating the credibility of the model which as of now has not been shown in the project. We limit our result to total(maximum) profit from the stock

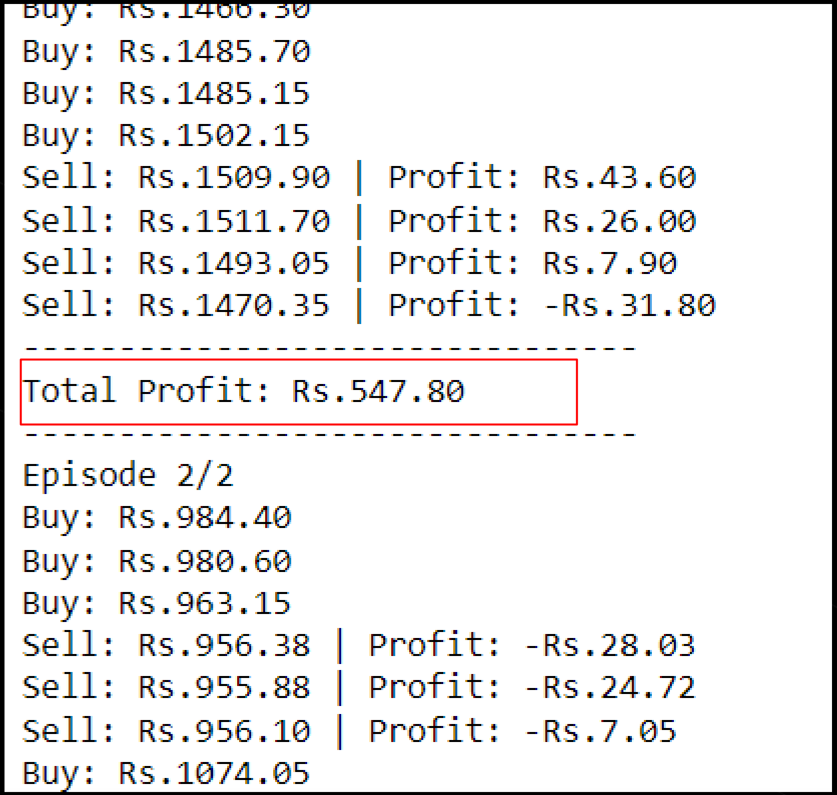


```python
import sys
stock_name = input("Enter stock_name: ")
window_size = input("Window size : ")
episode_count = input("Episode count: ")
stock_name = str(stock_name)
window_size = int(window_size)
episode_count = int(episode_count)
agent = Agent(window_size)
data = getStockDataVec(stock_name)
l = 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 = window_size_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(str(e))
```

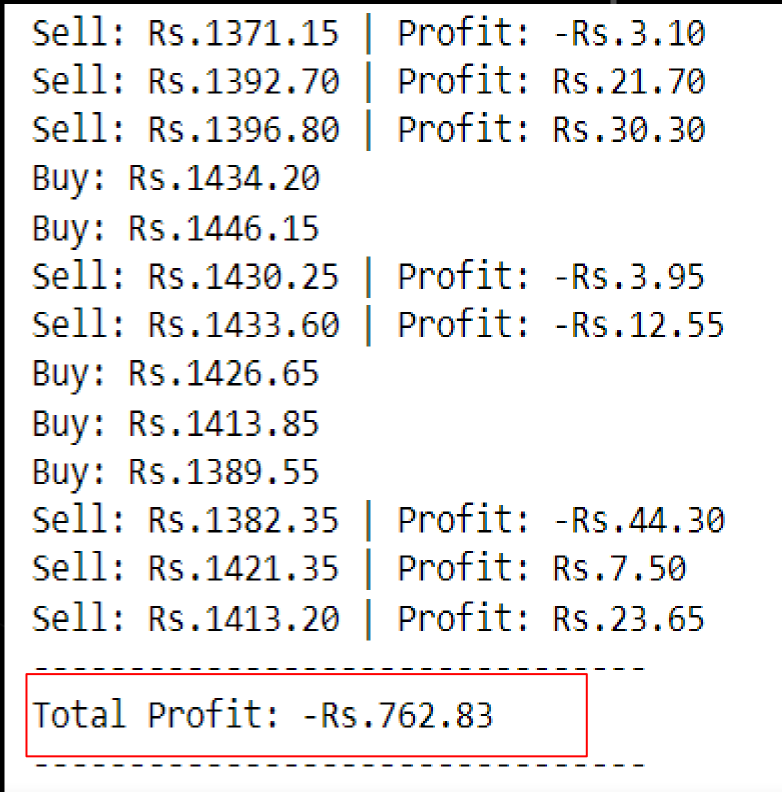
RESULTS:



```
Sell: Rs.1443.85 | Profit: -Rs.232.45
Sell: Rs.1458.65 | Profit: Rs.92.15
Sell: Rs.1448.80 | Profit: Rs.6.15
Sell: Rs.1438.60 | Profit: Rs.7.70
Sell: Rs.1461.05 | Profit: Rs.118.85
Sell: Rs.1450.90 | Profit: Rs.96.60
Sell: Rs.1454.40 | Profit: Rs.83.15
--------------------------------
Total Profit: -Rs.1686.07
--------------------------------
INFO:tensorflow:Assets written to: 0\assets
Episode 1/2
Buy: Rs.1062.55
Buy: Rs.1098.45
Sell: Rs.1133.05 | Profit: Rs.70.50
Sell: Rs.1138.55 | Profit: Rs.40.10
Buy: Rs.1611.85
Buy: Rs.1581.75
Sell: Rs.1572.35 | Profit: -Rs.39.50
Sell: Rs.1581.95 | Profit: Rs.0.20
```



```
Buy: Rs.1466.30
Buy: Rs.1485.70
Buy: Rs.1485.15
Buy: Rs.1502.15
Sell: Rs.1509.90 | Profit: Rs.43.60
Sell: Rs.1511.70 | Profit: Rs.26.00
Sell: Rs.1493.05 | Profit: Rs.7.90
Sell: Rs.1470.35 | Profit: -Rs.31.80
--------------------------------
Total Profit: Rs.547.80
--------------------------------
Episode 2/2
Buy: Rs.984.40
Buy: Rs.980.60
Buy: Rs.963.15
Sell: Rs.956.38 | Profit: -Rs.28.03
Sell: Rs.955.88 | Profit: -Rs.24.72
Sell: Rs.956.10 | Profit: -Rs.7.05
Buy: Rs.1074.05
```



```
Sell: Rs.1371.15 | Profit: -Rs.3.10
Sell: Rs.1392.70 | Profit: Rs.21.70
Sell: Rs.1396.80 | Profit: Rs.30.30
Buy: Rs.1434.20
Buy: Rs.1446.15
Sell: Rs.1430.25 | Profit: -Rs.3.95
Sell: Rs.1433.60 | Profit: -Rs.12.55
Buy: Rs.1426.65
Buy: Rs.1413.85
Buy: Rs.1389.55
Sell: Rs.1382.35 | Profit: -Rs.44.30
Sell: Rs.1421.35 | Profit: Rs.7.50
Sell: Rs.1413.20 | Profit: Rs.23.65
--------------------------------
Total Profit: -Rs.762.83
--------------------------------
```

Conclusion

Reinforcement learning gives the best possible results for stock predictions. By using Q learning, different experiments can be performed. In this project we did 2 iterations (2 epochs/episodes) due to which we failed to get the best result. As the number of epochs increase, the model is trained in a better and efficient way to give the best result with maximum profit.

More research in reinforcement learning will enable the application of reinforcement learning at a more confident stage.

Thank You
shift
{
[
}
]
4