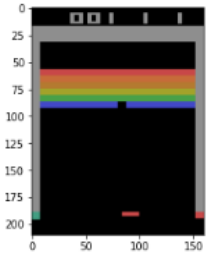
```
[13] #to clear tensor board log files
     !rm -rf  /root/ray_results
```

```
from ray.rllib.agents.ppo import PPOTrainer
import ray.rllib.agents.dqn as dqn
import gym
import matplotlib.pyplot as plt
from IPython import display
from ray import tune
import datetime
%load_ext tensorboard
import tensorflow as tf
import tensorboardX
```

## Render the environment for visual

```
[5]
    env = gym.make('Breakout-v0')
    env.reset()
    for _ in range(100):
        plt.imshow(env.render(mode='rgb_array'))
        display.display(plt.gcf())
        display.clear_output(wait=True)
        action = env.action_space.sample()
        observation, reward, done, info = env.step(env.action_space.sample())

        print("Reward", reward)
        env.step(action)
```

```
Reward 0.0
```




Code Helper

```
[6]  # Import the RL algorithm (Trainer) we would like to use.


    def evaluation_fn(result):
        return result['episode_reward_mean']


    def objective_fn(config):
        trainer = dqn.DQNTrainer(config=config)
        for i in range(100):
          # Perform one iteration of training the policy with DQN
          result = trainer.train()
          intermediate_score = evaluation_fn(result)

          # Feed the score back back to Tune.
          tune.report(iterations=i, mean_reward=intermediate_score)
```

Configure Set up

```
[11]
    config = dqn.DEFAULT_CONFIG.copy()
    config["env"] = 'Breakout-v0'
    config["dueling"] = True
    config["double_q"] = True
    config["gamma"] = tune.uniform(0, 1)
    config["train_batch_size"] = tune.grid_search([25,32,62])
    config['num_gpus'] =  1
    config['lr'] = 0.001
```

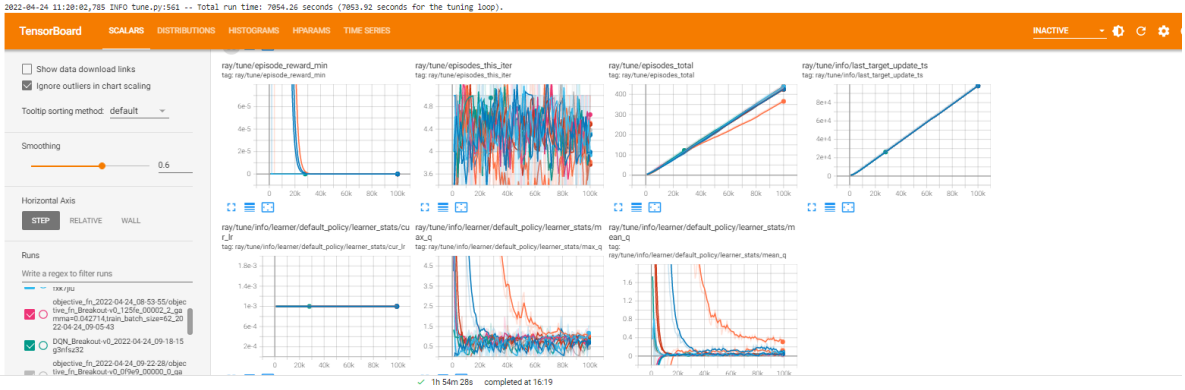
Run Analysis

```
[15]
    analysis = tune.run(
            objective_fn,
            metric="mean_reward",
            mode="max",
            #stop = {'episode_reward_mean': 20},
            num_samples=3,
            resources_per_trial={'gpu': 1},
            config=config
            )
```

```
[15] analysis = tune.run(
            objective_fn,
            metric="mean_reward",
            mode="max",
            #stop = {'episode_reward_mean': 20},
            num_samples=3,
            resources_per_trial={'gpu': 1},
            config=config
            )

    %reload_ext tensorboard
    %tensorboard --logdir  /root/ray_results
    #print("Best hyperparameters found were: ", analysis.best_config)
```

| | | | | | |
|---|---|---|---|---|---|
| objective_fn_Breakout-v0_0f9e9_00003 TERMINATED | 0.0653461 | 25 100 | 721.737 | 99 | 1.29 |
| objective_fn_Breakout-v0_0f9e9_00004 TERMINATED | 0.875074 | 32 100 | 746.724 | 99 | 1.37 |
| objective_fn_Breakout-v0_0f9e9_00005 TERMINATED | 0.388985 | 62 100 | 868.189 | 99 | 1.47 |
| objective_fn_Breakout-v0_0f9e9_00006 TERMINATED | 0.108556 | 25 100 | 730.065 | 99 | 1.61 |
| objective_fn_Breakout-v0_0f9e9_00007 TERMINATED | 0.82391 | 32 100 | 745.653 | 99 | 1.57 |
| objective_fn_Breakout-v0_0f9e9_00008 TERMINATED | 0.337854 | 62 100 | 866.576 | 99 | 1.46 |

```
2022-04-24 11:20:02,785 INFO tune.py:561 -- Total run time: 7054.26 seconds (7053.92 seconds for the tuning loop).
```



In [ ]:
```
!pip install --upgrade xlrd
```

```
Requirement already satisfied: xlrd in /usr/local/lib/python3.7/dist-packages (1.1.
0)
Collecting xlrd
  Downloading xlrd-2.0.1-py2.py3-none-any.whl (96 kB)
     |████████████████████████████████| 96 kB 3.4 MB/s
Installing collected packages: xlrd
  Attempting uninstall: xlrd
    Found existing installation: xlrd 1.1.0
    Uninstalling xlrd-1.1.0:
      Successfully uninstalled xlrd-1.1.0
Successfully installed xlrd-2.0.1
```

In [ ]:
```
import numpy as np
from numpy.random import default_rng
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import math
import numpy as np
import random
import pandas as pd
from datetime import datetime
import seaborn as sns
import matplotlib.pyplot as plt
from collections import namedtuple
#DQN package
#!pip install torch
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

In [ ]:
```
device
```

Out[ ]:
```
device(type='cuda')
```

In [ ]:
```
from google.colab import files
uploaded = files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Quantitative Trading2.xls to Quantitative Trading2.xls
```

# Helper Functions

In [ ]:
```
class DQN(nn.Module):

    def __init__(self, input_size, size_hidden, output_size, typedqn = 'Dqn'):

        super().__init__()
        self.typedqn = typedqn
        self.fc1 = nn.Linear(input_size, size_hidden)
        self.bn1 = nn.BatchNorm1d(size_hidden)
```

```
        self.fc2 = nn.Linear(size_hidden, size_hidden)
        self.bn2 = nn.BatchNorm1d(size_hidden)

        self.fc3 = nn.Linear(size_hidden, size_hidden)
        self.bn3 = nn.BatchNorm1d(size_hidden)

        if self.typedqn == 'Duelling':
            print('Duelling Activated')
            self.Value = nn.Linear(size_hidden, 1)
            self.Advantage = nn.Linear(size_hidden, output_size)
        if typedqn == 'Dqn':
            print('Dqn')
            self.fc4 = nn.Linear(size_hidden, output_size)


    def forward(self, x):
        h1 = F.relu(self.bn1(self.fc1(x.float())))
        h2 = F.relu(self.bn2(self.fc2(h1)))
        h3 = F.relu(self.bn3(self.fc3(h2)))

        if self.typedqn == 'Dqn':
            output = self.fc4(h3.view(h3.size(0), -1))
            return output

        if self.typedqn == 'Duelling':

            value = self.Value(h3.view(h3.size(0), -1))
            advantage = self.Advantage(h3.view(h3.size(0), -1))

            return value + advantage - advantage.mean()
```

In [ ]:

```
class E_Greedy_Policy():

    def __init__(self, epsilon, decay, min_epsilon):

        self.epsilon = epsilon
        self.epsilon_start = epsilon
        self.decay = decay
        self.epsilon_min = min_epsilon

    def __call__(self, state):

        is_greedy = random.random() > self.epsilon

        if is_greedy :
            # we select greedy action
            with torch.no_grad():
                Q_network.eval()
                # index of the maximum over dimension 1.
                index_action = Q_network(state).max(1)[1].view(1, 1).cpu()[0][0].ite
                Q_network.train()
        else:
            # we sample a random action
            index_action = random.randint(0,2) # will randomely choose a number with

        return index_action

    def update_epsilon(self):

        self.epsilon = self.epsilon*self.decay
        if self.epsilon < self.epsilon_min:
            self.epsilon = self.epsilon_min
```

```
    def reset(self):
        self.epsilon = self.epsilon_start
```

In [ ]:

```
class ReplayMemory:

    def __init__(self, capacity):
        self.capacity = capacity
        self.memory = []
        self.position = 0
    #just add to memory state,action, state+1,reward in a tensor
    def push(self, state, action, next_state, reward):
        """Saves a transition."""
        if len(self.memory) < self.capacity:
            self.memory.append(None)


        action_tensor = torch.tensor([action], device=device).unsqueeze(0)

        reward = torch.tensor([reward], device=device).unsqueeze(0)/10. # reward sca

        self.memory[self.position] = Transition(state, action_tensor, next_state, re
        self.position = (self.position + 1) % self.capacity

    #might need changing
    def sample(self, batch_size):
        return random.sample(self.memory, batch_size)

    def __len__(self):
        return len(self.memory)
```

In [ ]:

```
def optimize_model(doubledqn = False):

    transitions = memory.sample(BATCH_SIZE)
    batch = Transition(*zip(*transitions))

    # Compute a mask of non-final states and concatenate the batch elements
    non_final_mask = torch.tensor(tuple(map(lambda s: s is not None,
                                            batch.next_state)), device=device, dtype=t
    non_final_next_states = torch.cat([s for s in batch.next_state
                                                if s is not None])

    state_batch = torch.cat(batch.state)

    action_batch = torch.cat(batch.action)
    reward_batch = torch.cat(batch.reward)

    # Compute Q values using policy net
    Q_values = Q_network(state_batch).gather(1, action_batch)

    # Compute next Q values using Q_targets
    next_Q_values = torch.zeros( BATCH_SIZE, device=device)
    if doubledqn == False :
      # Compute next Q values using Q_targets
      next_Q_values[non_final_mask] = Q_target(non_final_next_states).max(1)[0].deta
      next_Q_values = next_Q_values.unsqueeze(1)
    else:

      Qaction = Q_network(non_final_next_states).max(1)[1].detach()
      next_Q_values = Q_target(non_final_next_states)[Qaction].unsqueeze(1)
      #ComputeTarget
```

```
    target_Q_values = (next_Q_values * GAMMA) + reward_batch
    loss = F.mse_loss(Q_values, target_Q_values)
    # Optimize the model
    optimizer.zero_grad()
    loss.backward()

    # Trick: gradient clipping
    for param in Q_network.parameters():
        param.grad.data.clamp_(-1, 1)

    optimizer.step()



    return loss
```

# Environment

## define class

In [ ]:

```
class StockTradingEnv:
    """
        Specifying the action and observation componenets that would help the agent

    """
    def __init__(self,
                 fee=0.1,
                 starting_date="2015-01-24",
                 max_training_date = '2020-12-31',
                 inital_cash_balance = 50000,
                 initial_number_shares = 10):

        # Defining the parameters specific to the environment
        self.df = pd.read_excel("Quantitative Trading2.xls", sheet_name='SPY (1)')
        self.action_list = ['Buy','Sell','Hold']
        self.feature_list = ['Daily Close Returns ', 'Open','High','Low', 'WR(6)', '
        self.feature_list_normalisation = ['Open','High','Low']
        self.normalizefeature = 'Close'
        #setup
        self.fee = fee #Stock trading commission fee per trade
        self.start_date = starting_date
        self.current_step = 0
        self.eventlog = [] #eventlog to keep track
        self.max_training_date = max_training_date

        #used to calculate portfoliovalue
        self.initial_number_shares = initial_number_shares
        self.inital_cash_balance = inital_cash_balance
        self.cash_balance = inital_cash_balance
        self.number_shares = initial_number_shares
        self.inventory =[]

        #Preprocessing
        self.df = self.df[self.df['Date']>=datetime.strptime(self.start_date, '%Y-%m

        self.training_data = self.df[self.df['Date']<datetime.strptime(self.max_trai
        for f in self.feature_list_normalisation:
          self.training_data[f] = self.training_data[f]/self.training_data[self.norm

        self.start_date = self.df['Date'].min()
```

```
        self.start_price = self.df[self.df['Date']==self.start_date]['Close'].values


        self.initial_value = self.cash_balance + self.number_shares* self.start_pric
        self.wallet = [self.initial_value]
        self.inventory = [self.start_price]*self.number_shares

        #self.training_data,self.observations, self.states, self.transition_matrix,
        #data preprocessing

    def get_state(self,step):
      analytic = np.array(self.training_data[self.feature_list].iloc[step])
      portfolio = np.array([self.number_shares,self.cash_balance])
      state_tensor = np.concatenate((analytic,portfolio))
      state_tensor = torch.tensor(state_tensor, device=device).unsqueeze(0)
      return state_tensor


    def step(self,action):
        """
        Move the step and recalculate portfolio Value
        """

        #Check the line below
        current_price = self.df.iloc[self.current_step]['Close']
        reward = 0
        ## Add condition if buy but enough cash -> hold (bump example) and similarly
        if action == 'Buy':
            if current_price < self.cash_balance:
              self.number_shares +=1
              self.cash_balance = self.cash_balance -self.fee - current_price
              self.wallet.append(self.cash_balance + self.number_shares* current_pri
              self.inventory.append(current_price)

        if action == 'Sell':
            if self.number_shares >0 :
                self.number_shares -=1
                self.cash_balance = self.cash_balance -self.fee + current_price
                self.wallet.append(self.cash_balance + self.number_shares* current_p
                bought_price = self.inventory.pop(0)
                reward = max(((current_price - bought_price )/bought_price), 0)

            #penalty to sell when no position held
            else:
               reward = -0.001


        if action == 'Hold':
            self.wallet.append(self.cash_balance + self.number_shares* current_price



        #create pandas or array of array which will include = step, action, state, w
        self.eventlog.append([self.current_step, self.number_shares,self.cash_balanc
        self.current_step+=1
        next_state = self.get_state(self.current_step)
        return next_state, reward

    def display(self):
        print ('Portfolio contains :' + str(self.number_shares ))
        print ('Initial Portfolio Value was : '+ str(self.initial_value))
        print (self.wallet)

        print ('inventory')
```

```
        print(*self.inventory, sep = ", ")

    def reset(self):
        self.current_step = 0
        self.cash_balance = self.inital_cash_balance
        self.number_shares = self.initial_number_shares
        self.inventory =[self.start_price]*self.number_shares
        self.eventlog = []
```

## set up enviroment

In [ ]:
```
trading = StockTradingEnv(starting_date="2015-01-24",max_training_date = '2020-12-31
```

In [ ]:
```
trading.display()
```

```
Portfolio contains :10
Initial Portfolio Value was : 52054.49997
[52054.49997]
inventory
205.449997, 205.449997, 205.449997, 205.449997, 205.449997, 205.449997, 205.449997,
205.449997, 205.449997, 205.449997
```

In [ ]:
```
Transition = namedtuple('Transition', ('state', 'action', 'next_state', 'reward'))
policy = E_Greedy_Policy(0.99, decay=0.997, min_epsilon=0.001)

OBS_SIZE = len(trading.feature_list)+2
HIDDEN_SIZE = 6  # mean of the output/input layer
ACTION_SIZE = 3
num_episodes = 1000
memory = ReplayMemory(250)#increase memory to try to avoid sampling from the same da
policy.reset()
BATCH_SIZE = 32
GAMMA = 0.7 # preference for long term reward
rewards_history = []
loss_history = []

doubledqn = True
duelingdqn = False

if duelingdqn == True:
  typeddqn = 'Duelling'
else:
  typeddqn = 'Dqn'

Q_network = DQN(OBS_SIZE, HIDDEN_SIZE, ACTION_SIZE,typedqn = typeddqn).to(device)
Q_target = DQN(OBS_SIZE, HIDDEN_SIZE, ACTION_SIZE,typedqn = typeddqn).to(device)
Q_target.load_state_dict(Q_network.state_dict())
Q_target.eval()

TARGET_UPDATE = 10
optimizer = optim.SGD(Q_network.parameters(), lr=0.01)
```

```
Dqn
Dqn
```

In [54]:
```
#number of episode
for i_episode in range(num_episodes):
    state = trading.reset()
    total_reward = 0
```

```
    #per episode - fill the memory with every state,i.e every day in our period
    for i in  range(0,len(trading.training_data.index)-2) :


        # Get action and act in the world. Dirty way improve coding
        state_tensor= trading.get_state(trading.current_step)
        action = policy(state_tensor)
        action_name = trading.action_list[action]

        next_state, reward = trading.step(action_name)


        total_reward += float(reward)

        # Store the transition in memory
        memory.push(state_tensor, action, next_state, float(reward))
        #BATCH_SIZE = min(BATCH_SIZE_initial,len(memory.memory))
        # Perform one step of the optimization
        if len(memory.memory)>BATCH_SIZE:
            started_training = True
            l = optimize_model(doubledqn)


    policy.update_epsilon()
    rewards_history.append( float(total_reward) )
    loss_history.append(l.detach().cpu())




    # Update the target network, copying all weights and biases in DQN
    if i_episode % TARGET_UPDATE == 0:

        Q_target.load_state_dict(Q_network.state_dict())

    if (i_episode) % 10 == 0:

        print('Episode ', i_episode, ': ', 'reward :',  total_reward, 'eps: ',
              policy.epsilon, ' loss:', l.detach().cpu())
        print( sum(rewards_history[-10:])/10)

print('Complete')
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:32: UserWarning: Using
a target size (torch.Size([32, 32, 3])) that is different to the input size (torch.S
ize([32, 1])). This will likely lead to incorrect results due to broadcasting. Pleas
e ensure they have the same size.
Episode  0 :  reward : 18.682230801983106 eps:  0.98703  loss: tensor(7499.2109)
1.8682230801983106
Episode  10 :  reward : 10.091921488931657 eps:  0.9578156659018894  loss: tensor(0.
0991)
13.397120696176213
Episode  20 :  reward : 7.071260993511062 eps:  0.9294660241807036  loss: tensor(0.0
084)
11.083428154234767
Episode  30 :  reward : 6.221215631630003 eps:  0.9019554814785994  loss: tensor(0.0
004)
8.59070968308818
Episode  40 :  reward : 10.582734708756336 eps:  0.8752592019556485  loss: tensor(0.
0358)
12.208835442757803
Episode  50 :  reward : 16.168952544366377 eps:  0.8493530848686524  loss: tensor(0.
```

```
0008)
12.708342001112147
Episode  60 :  reward : 13.176216999025836 eps:  0.8242137428135844  loss: tensor(0.
0296)
11.352757274181558
Episode  70 :  reward : 3.2878783001867764 eps:  0.79981848061202  loss: tensor(0.02
67)
11.931924656405283
Episode  80 :  reward : 9.209856978745416 eps:  0.776145274822487  loss: tensor(0.00
05)
9.92221068030933
Episode  90 :  reward : 2.7323179000877564 eps:  0.7531727538582468  loss: tensor(2.
2773e-05)
3.669099149753979
Episode  100 :  reward : 1.2229018515655412 eps:  0.7308801786935519  loss: tensor
(1.5621e-07)
1.7806455337797584
Episode  110 :  reward : 1.3869453954906894 eps:  0.7092474241409645  loss: tensor
(1.9535e-07)
1.4610017535590492
Episode  120 :  reward : 1.3703866289041715 eps:  0.688254960682834  loss: tensor(2.
7313e-07)
1.475596811082064
Episode  130 :  reward : 1.7444949871829225 eps:  0.667883836840529  loss: tensor(1.
7706e-07)
1.342003710276432
Episode  140 :  reward : 1.6069871453708098 eps:  0.6481156620655096  loss: tensor
(6.1932e-08)
1.1561816232444162
Episode  150 :  reward : 0.7673780983649789 eps:  0.6289325901367944  loss: tensor
(3.6228e-08)
1.1571360030278415
Episode  160 :  reward : 0.7963563872237165 eps:  0.61031730304983  loss: tensor(5.2
113e-08)
1.1047373197381738
Episode  170 :  reward : 1.1521021243177378 eps:  0.5922529953822255  loss: tensor
(1.4838e-07)
1.0555320574095173
Episode  180 :  reward : 0.8942014949959565 eps:  0.5747233591222303  loss: tensor
(7.3973e-08)
0.9485618193739368
Episode  190 :  reward : 0.7614902087097148 eps:  0.5577125689462629  loss: tensor
(7.7972e-08)
0.936916187777918
Episode  200 :  reward : 0.6108558554949427 eps:  0.5412052679321988  loss: tensor
(3.7478e-07)
0.8828534195887044
Episode  210 :  reward : 1.2013600044940138 eps:  0.5251865536955204  loss: tensor
(6.1625e-08)
0.7908529815496396
Episode  220 :  reward : 0.7635083271311173 eps:  0.5096419649358108  loss: tensor
(1.0345e-07)
0.7189682364296637
Episode  230 :  reward : 0.5761046949477531 eps:  0.4945574683814483  loss: tensor
(2.7371e-07)
0.6724985299061924
Episode  240 :  reward : 0.7598815231447277 eps:  0.4799194461207152  loss: tensor
(1.6102e-07)
0.6694930440092743
Episode  250 :  reward : 0.6539945796089337 eps:  0.46571468330788196  loss: tensor
(5.1986e-08)
0.7082136077979477
Episode  260 :  reward : 0.5196137256309733 eps:  0.4519303562331707  loss: tensor
(1.8561e-08)
```

```
0.5719432105423098
Episode  270 :  reward : 0.5212923688962968 eps:  0.43855402074582583  loss: tensor
(2.5718e-08)
0.5531776156099715
Episode  280 :  reward : 0.6299536930480565 eps:  0.4255736010198415  loss: tensor
(1.3398e-07)
0.5908264684214706
Episode  290 :  reward : 0.4568653353346969 eps:  0.4129773786522035  loss: tensor
(5.7610e-08)
0.5474835317153861
Episode  300 :  reward : 0.724478000228548 eps:  0.4007539820838039  loss: tensor(7.
4623e-08)
0.6231565134233007
Episode  310 :  reward : 0.48873952698132084 eps:  0.38889237633347773  loss: tensor
(1.6982e-07)
0.5563581941122079
Episode  320 :  reward : 0.4063709863835462 eps:  0.37738185303589367  loss: tensor
(1.7014e-08)
0.5309398552353597
Episode  330 :  reward : 0.41523426175680733 eps:  0.3662120207743062  loss: tensor
(4.6218e-08)
0.48022335848852366
Episode  340 :  reward : 0.5527575745150262 eps:  0.35537279569944036  loss: tensor
(7.7191e-08)
0.5330366272629485
Episode  350 :  reward : 0.5281199966420863 eps:  0.3448543924260414  loss: tensor
(1.4299e-07)
0.5642732764569429
Episode  360 :  reward : 0.3770649431052356 eps:  0.33464731519886964  loss: tensor
(6.3955e-08)
0.48393522548375517
Episode  370 :  reward : 0.4892842323439436 eps:  0.32474234932016743  loss: tensor
(8.1462e-08)
0.47963392102444746
Episode  380 :  reward : 0.38854592533844245 eps:  0.3151305528308563  loss: tensor
(6.2116e-08)
0.41772212657119157
Episode  390 :  reward : 0.3824570621369908 eps:  0.30580324843795753  loss: tensor
(1.0512e-07)
0.4416766344120816
Episode  400 :  reward : 0.4517849872325957 eps:  0.296752015680945  loss: tensor(2.
2083e-08)
0.4256455078634091
Episode  410 :  reward : 0.5651816225153377 eps:  0.2879686833299618  loss: tensor
(3.8079e-08)
0.4422399945566659
Episode  420 :  reward : 0.4839997128065774 eps:  0.2794453220090349  loss: tensor
(7.5997e-08)
0.4672289360961863
Episode  430 :  reward : 0.44128120498716633 eps:  0.2711742370376297  loss: tensor
(3.8878e-08)
0.4048761347149
Episode  440 :  reward : 0.3744601126413262 eps:  0.26314796148408265  loss: tensor
(3.3555e-08)
0.3548274034036977
Episode  450 :  reward : 0.25186820981739716 eps:  0.25535924942463895  loss: tensor
(2.3829e-08)
0.36550196510971555
Episode  460 :  reward : 0.3824528168878403 eps:  0.2478010694020113  loss: tensor
(2.0879e-08)
0.4030108529445339
Episode  470 :  reward : 0.4915533734817623 eps:  0.24046659807755363  loss: tensor
(2.7392e-08)
0.34323848390748984
```

```
Episode  480 :  reward : 0.22442269081185232 eps:  0.23334921407131903  loss: tensor
(3.0276e-08)
0.30660681036942744
Episode  490 :  reward : 0.511055445607062 eps:  0.22644249198444116  loss: tensor
(3.3631e-08)
0.35568873358368347
Episode  500 :  reward : 0.29495329006552307 eps:  0.21974019659844254  loss: tensor
(3.7613e-08)
0.3125046320273889
Episode  510 :  reward : 0.4633938588974193 eps:  0.21323627724623298  loss: tensor
(2.2762e-08)
0.3896121536569686
Episode  520 :  reward : 0.2077932331640847 eps:  0.20692486234971638  loss: tensor
(2.0699e-08)
0.2665973602829873
Episode  530 :  reward : 0.3321398623433456 eps:  0.20080025411907482  loss: tensor
(1.8014e-08)
0.29003701890479994
Episode  540 :  reward : 0.23536551181491575 eps:  0.19485692340894423  loss: tensor
(6.2326e-09)
0.2926897209513716
Episode  550 :  reward : 0.10002607755857915 eps:  0.18908950472683844  loss: tensor
(5.4518e-09)
0.2452004925062039
Episode  560 :  reward : 0.23951149001519573 eps:  0.1834927913893146  loss: tensor
(2.9316e-08)
0.31161183748474036
Episode  570 :  reward : 0.18303456922601474 eps:  0.1780617308215078  loss: tensor
(1.3404e-08)
0.2278229299669235
Episode  580 :  reward : 0.3434904009740406 eps:  0.17279141999579084  loss: tensor
(5.3668e-08)
0.3506411379833406
Episode  590 :  reward : 0.20191331463308942 eps:  0.16767710100544203  loss: tensor
(2.1753e-08)
0.2087363880162112
Episode  600 :  reward : 0.2254546979468896 eps:  0.16271415676932394  loss: tensor
(1.4336e-08)
0.24138671279701862
Episode  610 :  reward : 0.5252382241868144 eps:  0.15789810686369657  loss: tensor
(3.0759e-08)
0.24248631942562052
Episode  620 :  reward : 0.43120434058284646 eps:  0.1532246034774011  loss: tensor
(4.0785e-08)
0.31949130888770055
Episode  630 :  reward : 0.15443893531162456 eps:  0.1486894274867632  loss: tensor
(2.6731e-08)
0.25713932914769416
Episode  640 :  reward : 0.17096833085610486 eps:  0.1442884846466721  loss: tensor
(5.5306e-08)
0.22322747806977059
Episode  650 :  reward : 0.18057291853353116 eps:  0.14001780189439708  loss: tensor
(3.2261e-08)
0.20163576417664175
Episode  660 :  reward : 0.20555973416427675 eps:  0.13587352376280437  loss: tensor
(1.0556e-07)
0.1909641908145845
Episode  670 :  reward : 0.2693247103015724 eps:  0.13185190889973625  loss: tensor
(3.5847e-09)
0.17475828345232644
Episode  680 :  reward : 0.15981369765556427 eps:  0.12794932669041076  loss: tensor
(1.1594e-08)
0.19796344839207478
Episode  690 :  reward : 0.1576429414176489 eps:  0.12416225397979205  loss: tensor
```

```
(2.0902e-08)
0.1876291431794134
Episode  700 :  reward : 0.05888384063503748 eps:  0.12048727189197293  loss: tensor
(7.5737e-09)
0.12531682845140532
Episode  710 :  reward : 0.14767563949712031 eps:  0.11692106274369779  loss: tensor
(9.0491e-08)
0.19505978601652785
Episode  720 :  reward : 0.17859703036125446 eps:  0.11346040704924014  loss: tensor
(1.5289e-08)
0.19521323947633623
Episode  730 :  reward : 0.1644056229313069 eps:  0.11010218061393003  loss: tensor
(1.9440e-08)
0.178467419124174
Episode  740 :  reward : 0.149839778382346 eps:  0.10684335171370828  loss: tensor
(2.4177e-08)
0.18865481873371073
Episode  750 :  reward : 1.1227283208132888 eps:  0.10368097835816062  loss: tensor
(1.1262e-07)
0.2842934632746931
Episode  760 :  reward : 0.3576035294371286 eps:  0.10061220563456125  loss: tensor
(1.6080e-07)
0.43880250510896424
Episode  770 :  reward : 0.10097284813790872 eps:  0.09763426313052807  loss: tensor
(1.3167e-08)
0.12701011350551591
Episode  780 :  reward : 0.28038389733088714 eps:  0.09474446243296256  loss: tensor
(4.7735e-08)
0.4785442844661743
Episode  790 :  reward : 0.08651567746451791 eps:  0.09194019470101675  loss: tensor
(9.9685e-08)
0.32819014125925594
Episode  800 :  reward : 0.18126896632378411 eps:  0.08921892831089603  loss: tensor
(6.5886e-08)
0.1486744924434258
Episode  810 :  reward : 0.13519147943951768 eps:  0.08657820657037152  loss: tensor
(1.6600e-08)
0.1825741452667745
Episode  820 :  reward : 0.015799705158202698 eps:  0.08401564550093889  loss: tenso
r(8.0101e-09)
0.13376541573643838
Episode  830 :  reward : 0.34484759291600137 eps:  0.08152893168562139  loss: tensor
(1.2538e-08)
0.4433059116275035
Episode  840 :  reward : 0.0882775096981786 eps:  0.07911582018047386  loss: tensor
(9.2957e-09)
0.1240110152753687
Episode  850 :  reward : 0.10454728165165228 eps:  0.0767741324879028  loss: tensor
(1.3249e-09)
0.27294082196429076
Episode  860 :  reward : 0.19915235445877905 eps:  0.07450175458997244  loss: tensor
(7.0291e-08)
0.15609960150873875
Episode  870 :  reward : 0.5628509192398148 eps:  0.07229663503992138  loss: tensor
(7.6919e-08)
0.7079592756389241
Episode  880 :  reward : 0.07063198917519575 eps:  0.07015678311016703  loss: tensor
(1.3108e-08)
0.12258590830649711
Episode  890 :  reward : 0.020663837484324314 eps:  0.06808026699512584  loss: tenso
r(6.2295e-09)
0.27569654118675413
Episode  900 :  reward : 0.004191248976473074 eps:  0.0660652120672268  loss: tensor
(1.7221e-08)
```

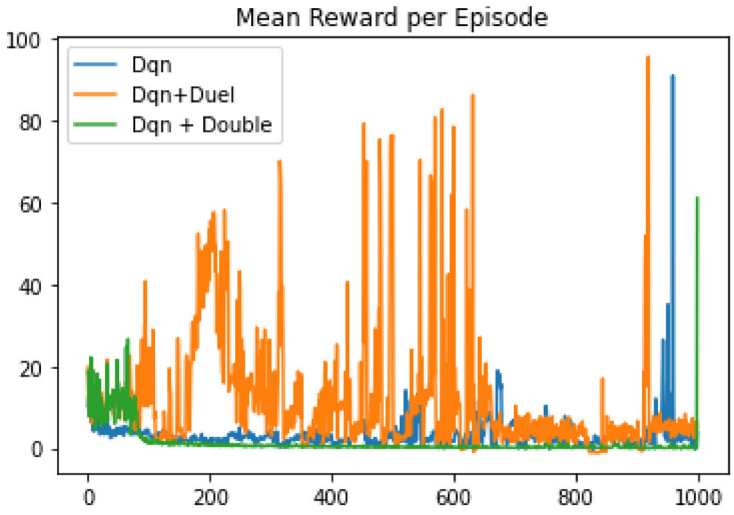
```
0.2808507732159313
Episode  910 :  reward : 0.03924731850173159 eps:  0.06410979918454389  loss: tensor
(6.3546e-08)
0.09472445457833856
Episode  920 :  reward : 0.12795421735355295 eps:  0.0622122630485196  loss: tensor
(1.4397e-08)
0.07987360207903123
Episode  930 :  reward : 0.04734644100337713 eps:  0.06037089061029686  loss: tensor
(7.9523e-09)
0.08140699328005897
Episode  940 :  reward : 0.2182722640393738 eps:  0.05858401952422075  loss: tensor
(7.3679e-08)
0.2785641937503244
Episode  950 :  reward : 0.20955210380695732 eps:  0.056850036647113855  loss: tenso
r(3.0423e-08)
0.08818456615985645
Episode  960 :  reward : -0.03946787532896424 eps:  0.05516737658197032  loss: tenso
r(1.5047e-08)
0.25823256562386054
Episode  970 :  reward : 0.16887206913883918 eps:  0.05353452026475406  loss: tensor
(3.4365e-07)
0.11654685657885719
Episode  980 :  reward : 0.2021268301333068 eps:  0.05194999359302514  loss: tensor
(6.4811e-08)
0.24687682660990423
Episode  990 :  reward : 0.04746490150434769 eps:  0.050412366095156445  loss: tenso
r(9.2139e-09)
0.16256457831017912
Complete
```

In [56]:

```
if doubledqn == False and duelingdqn == False:
  totalrewardqn =rewards_history
  totallossdqn = loss_history
if doubledqn == False and duelingdqn == True:
  totalrewardueldqn =rewards_history
  totallossduelqn = loss_history
if doubledqn == True and duelingdqn == False:
  totalrewarddqn =rewards_history
  totallossddqn = loss_history
if doubledqn == True and duelingdqn == False:
  totalrewardddqn =rewards_history
  totallossdddqn = loss_history
```

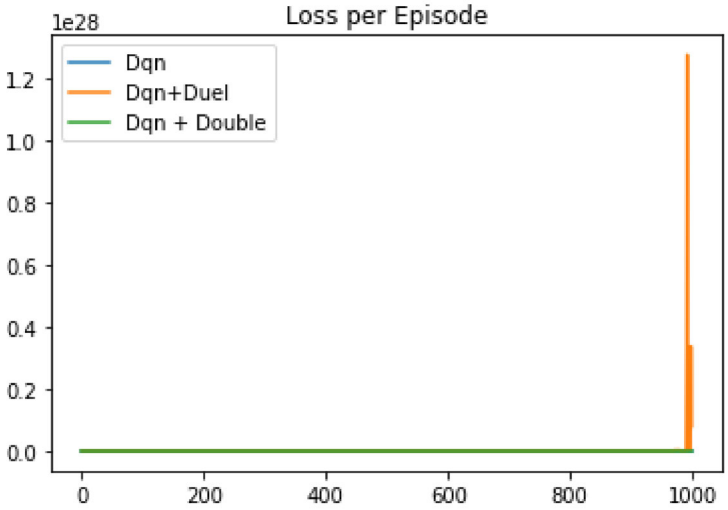
In [58]:

```
plt.plot( totalrewardqn, label = 'Dqn')
plt.plot( totalrewardueldqn, label = 'Dqn+Duel')
plt.plot( totalrewarddqn, label = 'Dqn + Double')
totalrewarddqn
plt.title('Mean Reward per Episode')
plt.legend()
plt.show()
```




In [59]:

```
plt.plot( totallossdqn, label = 'Dqn')
plt.plot( totallossduelqn, label = 'Dqn+Duel')
plt.plot( totalrewarddqn, label = 'Dqn + Double')
totalrewarddqn
plt.title('Loss per Episode')
plt.legend()
plt.show()
```




In [ ]:

```
policy.epsilon
```

Out[ ]:

```
0.98703
```

In [1]:
```
import numpy as np
from numpy.random import default_rng
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import math
import numpy as np
import random
import pandas as pd
from datetime import datetime
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
from sklearn import preprocessing
```

# Part 1 - Defining the environment.

In [2]:
```
class StockTradingEnv:
    """
        Specifying the action and observation componenets that would help the agent

    """
    def __init__(self,
                 fee=0.1,
                 starting_date="2015-01-24",
                 max_training_date = '2020-12-31',
                 inital_cash_balance = 50000,
                 initial_number_shares = 10):

        # Defining the parameters specific to the environment
        self.df = pd.read_excel("Quantitative Trading2.xls", sheet_name='SPY (1)')
        self.action_list = ['Buy','Sell','Hold']
        self.feature_list = ['Daily Close Returns ', 'Open','High','Low','Close', 'W
        #setup
        self.fee = fee #Stock trading commission fee per trade
        self.start_date = starting_date
        self.current_step = 0
        self.eventlog = [] #eventlog to keep track
        self.max_training_date = max_training_date

        #used to calculate portfoliovalue
        self.initial_number_shares = initial_number_shares
        self.inital_cash_balance = inital_cash_balance
        self.cash_balance = self.inital_cash_balance
        self.number_shares = self.initial_number_shares

        #To_review

        self.df = self.df[self.df['Date']>=datetime.strptime(self.start_date, '%Y-%m
        self.training_data = self.df[self.df['Date']<datetime.strptime(self.max_trai
        self.start_date = self.df['Date'].min()

        #maybe needed.
        self.initial_value = self.cash_balance + self.number_shares* self.df[self.df
        self.wallet = [self.initial_value]

        #self.training_data,self.observations, self.states, self.transition_matrix,
        #data preprocessing
```

```
    def reward_function(self,action, step):
        ''' Reward Policy, if :
        - Sell : Next day return *-100
        - Buy : Next day return *100
        - Hold : if held and movement superior to 4% negative reward : abs(next day
        while if movement undertrading fee, positive reward
        '''

        #Check the line below
        if action == 'Buy':
            reward = (self.df.iloc[step + 1]['Daily Close Returns '])*100

        if action == 'Sell':
            reward = (self.df.iloc[step + 1]['Daily Close Returns '])*-100

        if action == 'Hold':
            if abs(self.df.iloc[step + 1]['Daily Close Returns ']*100) >4:
                reward = abs(self.df.iloc[step + 1]['Daily Close Returns '])*-100/2

            elif abs(self.df.iloc[step + 1]['Daily Close Returns '])*100 <self.fee:
                reward = abs(self.df.iloc[step + 1]['Daily Close Returns '])*200

            else :
                reward = 0

        return reward

    def step(self,action):
        """
        Move the step and recalculate portfolio Value
        """

        #Check the line below
        self._state = self.df.iloc[self.current_step]
        current_price = self.df.iloc[self.current_step]['Close']

        ## Add condition if buy but enough cash -> hold (bump example) and similarly
        if action == 'Buy':
            #transition function
            if current_price > self.cash_balance:
                self.wallet.append(self.cash_balance + self.number_shares* current_p
                comment = 'Not Enough Cash to Buy'
            else :
                self.number_shares +=1
                self.cash_balance = self.cash_balance -self.fee - current_price
                self.wallet.append(self.cash_balance + self.number_shares* current_p
                comment = ''

        if action == 'Sell':
            if self.number_shares ==0 :
                self.wallet.append(self.cash_balance + self.number_shares* current_p
                comment = 'Not Enough Shares to Sell '

            else:
                self.number_shares -=1
                self.cash_balance = self.cash_balance -self.fee + current_price
                self.wallet.append(self.cash_balance + self.number_shares* current_p
                comment = ''

        if action == 'Hold':
            self.wallet.append(self.cash_balance + self.number_shares* current_price
            comment = ''
```

```
        #create pandas or array of array which will include = step, action, state, w
        self.eventlog.append([self.current_step, self.number_shares,self.cash_balanc
        self.current_step+=1


    def display(self):
        print ('Portfolio contains :' + str(self.number_shares ))
        print ('Initial Portfolio Value was : '+ str(self.initial_value))
        print (self.wallet)
        #to add
        print ('total reward')

    def reset(self):
        self.current_step = 0
        self.cash_balance = self.inital_cash_balance
        self.number_shares = self.initial_number_shares
        self.eventlog = []
        self.initial_value = self.cash_balance + self.number_shares* self.df[self.df

    def qlearning_preprocessing (self):

        #kmeans
        print('Clustering')
        kmeans = KMeans(n_clusters=7)

        standard = preprocessing.scale(self.training_data[self.feature_list])
        standard = pd.DataFrame(standard, columns =self.feature_list)

        y = kmeans.fit_predict(standard)
        self.training_data['Cluster'] = y
        centroid = kmeans.cluster_centers_


        fig, ax = plt.subplots()
        plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 6], s
        for i,txt in enumerate(['state_0', 'state_1', 'state_2','state_3','state_4',
            ax.annotate(txt, (kmeans.cluster_centers_[i, 0]+0.08, kmeans.cluster_cen
        plt.title('State Representation')

        ax.set_xlim(xmin=(kmeans.cluster_centers_[:, 0]).min()-0.2, xmax=(kmeans.clu
        plt.xlabel(self.feature_list[0])
        plt.ylabel(self.feature_list[6])
        plt.show()
        plt.clf()

        print('Calculating historical expected transition matrix')
        self.training_data['next_cluster']=self.training_data['Cluster'].shift(-1)
        self.training_data['Count'] = 1
        transition_matrix = pd.pivot_table(self.training_data, values='Count', index
        transition_matrix = transition_matrix/sum(self.training_data['Count'])
        transition_matrix =transition_matrix.fillna(0)*100

        sns.heatmap(transition_matrix, cmap = 'Blues',annot=True, fmt ='.2f')
        plt.title('State Transition Matrix')
        plt.show()
        plt.clf()

        print('Calculating historical expected return matrix for state-action')
        matrix_rewards =  np.zeros((7, 3))
        simulated_er = []
        n=0

        #In order to descretise the return, we looked at expected return by simulati
```

```
        for i in range(len(self.training_data.index)-1) :
            for action in range (3):
                reward = self.reward_function(self.action_list[action],i)
                simulated_er.append([i,self.action_list[action],reward, self.trainin

        simulated_er= pd.DataFrame(simulated_er, columns = ['i','Action', 'Reward','
        #to debug : simulated_er.to_csv('histreward.csv')
        reward_matrix = pd.pivot_table(simulated_er, values='Reward', index=['State'
        reward_matrix=reward_matrix[ self.action_list]
        sns.heatmap(reward_matrix, cmap = 'Blues',annot=True, fmt ='.2f')
        plt.title('reward_matrix')
        plt.show()
        plt.clf()
        print('End of Preprocessing')
        return self.training_data, transition_matrix, reward_matrix
```

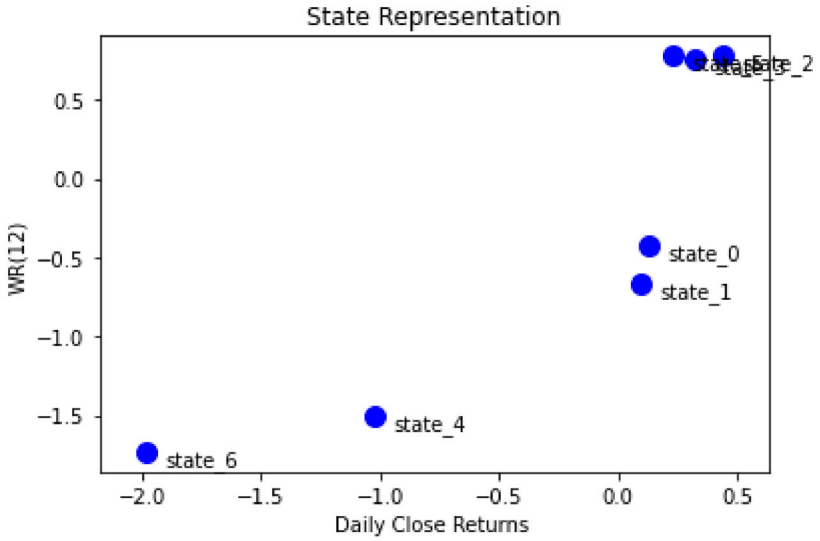
# Part 2 - Q-Learning Agent.

## Setting up the environment

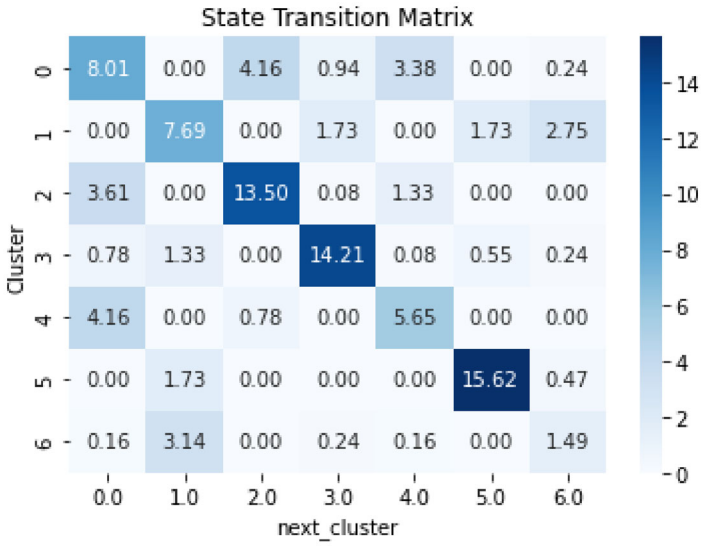
In [3]:

```
trading = StockTradingEnv()
print ('Extracting time period, Defining State, and returning transition Matrix annd
trading.training_data, transition_matrix, reward_matrix = trading.qlearning_preproce
```

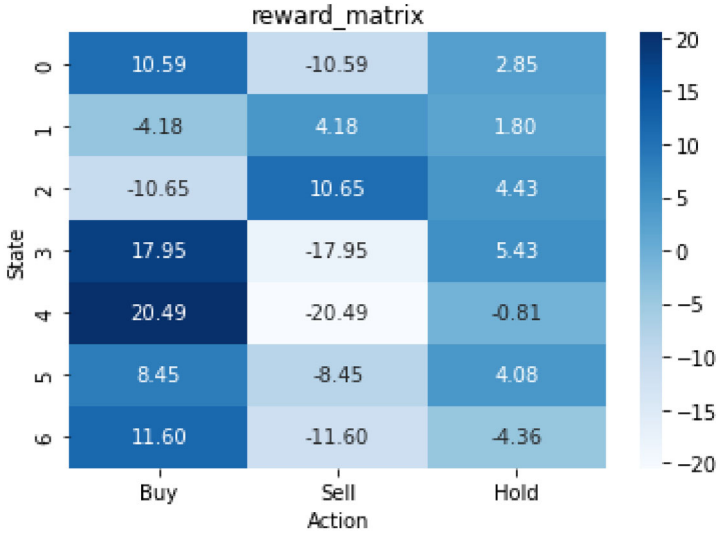
```
Extracting time period, Defining State, and returning transition Matrix annd state t
ransition function
Clustering
```




```
Calculating historical expected transition matrix
```




```
Calculating historical expected return matrix for state-action
```




```
End of Preprocessing
<Figure size 432x288 with 0 Axes>
```

# Setting up the Q-Learning Agent

In [38]:

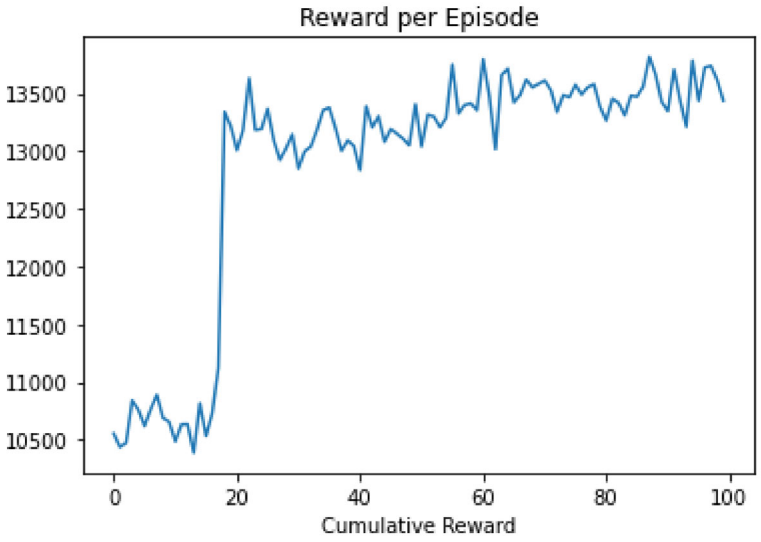
```
#each episode are training on 1 year of data as we are trying to maximize our return
def Qlearning_Agent(epsilon,alpha, gamma, reward_matrix, is_eps_decay = False,episod
    episode_log =[]
    emptyDict = {}
    decay = 0.99
    epsilon_min = 0.1


    Q = np.zeros(reward_matrix.shape)

    for episode in range (episoden):
        rewardlist =[]
        cumulative_reward = 0
        trading.reset()

        for i in range(len(trading.training_data.index)-1):
            #to add code
            possible_action = trading.action_list


            state = trading.training_data.iloc[trading.current_step]['Cluster']
            q_values = [Q[state,possible_action.index(a)] for a in possible_action]
```

```
            best_actions =  np.array(possible_action)[np.where(q_values == np.max(q_
            best_actions_q_values = [Q[state,possible_action.index(x)] for x in best

            #Policy Epsilon-greedy


            is_greedy = random.random() > epsilon

            if is_greedy :
            # we select greedy action
                a = np.random.choice(best_actions)
            else:
            # we sample a random action
                 a = np.random.choice(possible_action)


            # Environment updating
            r = reward_matrix.iloc[state][a]
            cumulative_reward += r
            rewardlist.append(r)
            s_old = state
            state = trading.training_data.iloc[trading.current_step+1]['Cluster']

            # here, the transition function is stochastic. Next state is dependend o
            q_values = [Q[state,possible_action.index(a)] for a in possible_action]
            q_updated = Q[s_old,possible_action.index(a)] + alpha * ( r + gamma * np
            Q[s_old,possible_action.index(a)] = q_updated
            trading.step(a)

        episode_log.append([episode,cumulative_reward])
        if is_eps_decay == True:
            epsilon = epsilon*decay
        if epsilon < epsilon_min:
            epsilon = epsilon_min

    episode_log_df = pd.DataFrame(episode_log,columns=['episode','Cumulative Rewards



    return episode_log_df,Q
```

## Selected Alpha, Beta and policy

In [43]:
```
ep_log, Qlearned = Qlearning_Agent(0.1,0.01,0.8, reward_matrix)
```

In [44]:
```
plt.plot(ep_log['Cumulative Rewards'])
plt.title('Reward per Episode')
plt.xlabel('Episodes')
plt.xlabel('Cumulative Reward')
plt.show()
```




In [61]:

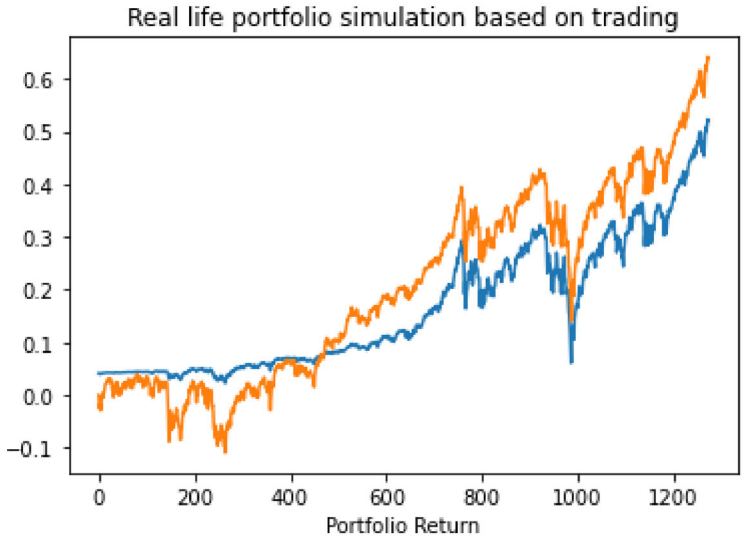
```
last_episode = pd.DataFrame(trading.eventlog,columns =['episode', 'shares', 'cash','
last_episode['NAV'] = last_episode['cash'] + last_episode['shares']*last_episode['pr
last_episode['Return'] = last_episode['NAV']/trading.inital_cash_balance - 1
last_episode['Price Return'] = last_episode['price']/trading.df[trading.df['Date']==
last_episode['Cash%'] = last_episode['cash'] /last_episode['NAV'] *100
last_episode.head(5)
```

Out[61]:

| | episode | shares | cash | price | action | comment | NAV | Return | Price Return |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 9 | 50205.349997 | 205.449997 | Sell | | 52054.399970 | 0.041088 | 0.000000 |
| **1** | 1 | 10 | 50002.509992 | 202.740005 | Buy | | 52029.910042 | 0.040598 | -0.013191 |
| **2** | 2 | 11 | 49802.269993 | 200.139999 | Buy | | 52003.809982 | 0.040076 | -0.025846 |
| **3** | 3 | 12 | 49600.179988 | 201.990005 | Buy | | 52024.060048 | 0.040481 | -0.016841 |
| **4** | 4 | 13 | 49400.629991 | 199.449997 | Buy | | 51993.479952 | 0.039870 | -0.029204 |

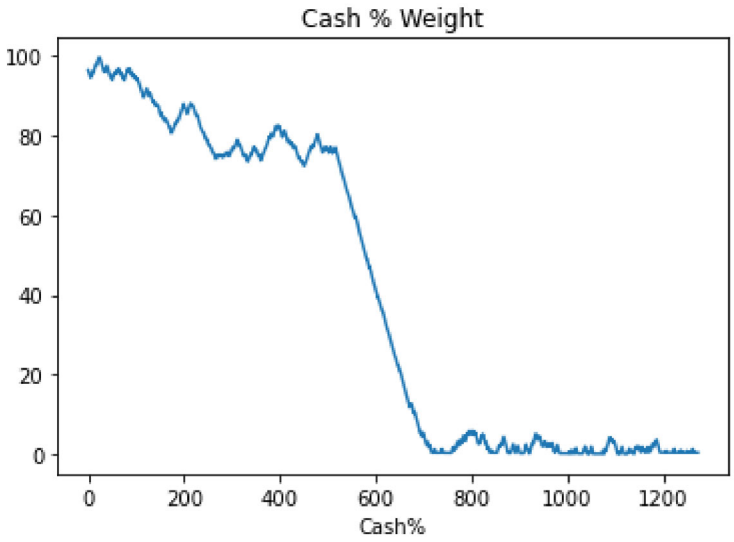
In [62]:

```
#Real life portfolio simulation based on signal
plt.plot(last_episode['Return'], label = 'Strategy Return')
plt.plot(last_episode['Price Return'], label = 'ETF Return')
plt.title('Real life portfolio simulation based on trading')
plt.xlabel('Days cumulated')
plt.xlabel('Portfolio Return')
plt.show()
```




In [60]:

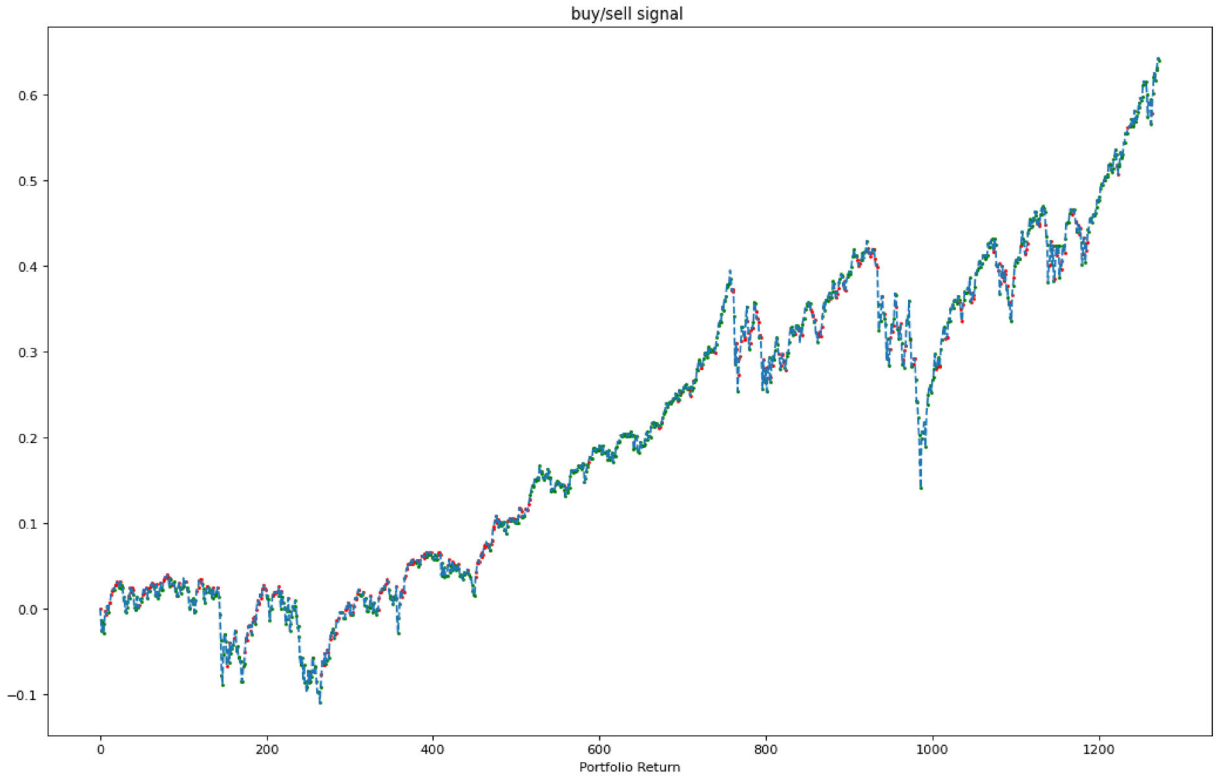
```
#Real life portfolio simulation based on signal
plt.plot(last_episode['Cash%'])
plt.title('Cash % Weight')
plt.xlabel('Days cumulated')
plt.xlabel('Cash%')
plt.show()
```




In [82]:

```
figure(figsize=(16,10), dpi=80)
plt.plot(last_episode['episode'],last_episode['Price Return'], linestyle='dashed', l

plt.scatter(last_episode[last_episode['action']=='Sell']['episode']
            ,last_episode[last_episode['action']=='Sell']['Price Return']
            ,marker="."
            ,color = 'red'
            , label = 'Sell',s = 10)

plt.scatter(last_episode[last_episode['action']=='Buy']['episode']
            ,last_episode[last_episode['action']=='Buy']['Price Return']
            ,marker="."
            ,color = 'green'
            , label = 'Buy'
           , s =10)

plt.title('buy/sell signal ')
plt.xlabel('Days cumulated')
```

```
plt.xlabel('Portfolio Return')
plt.show()
```




# Other combination

In [5]:

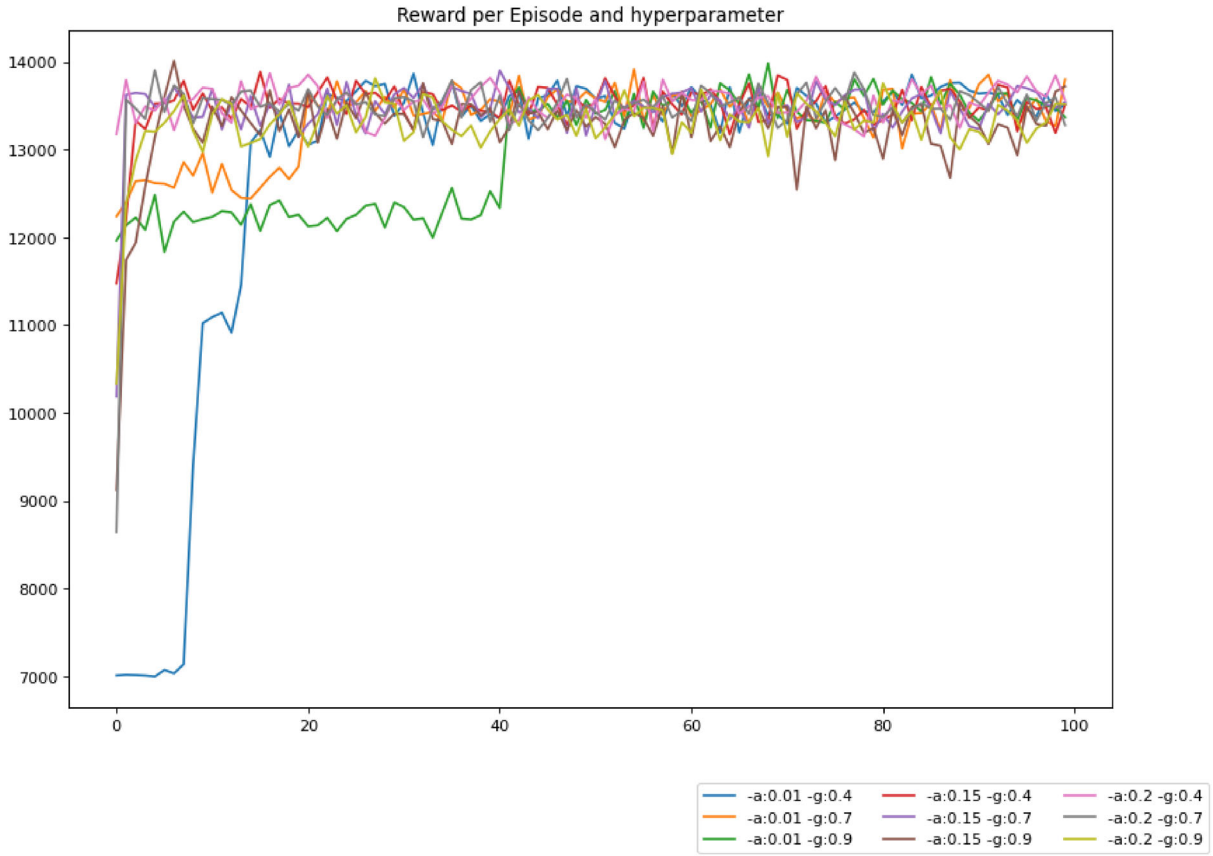
```
alpha_test = [0.01, 0.15, 0.2]
gamma_test = [0.4,0.7,0.9]
epsilon = 0.1

fulllog = None
QLdictionary = {}
for alpha in alpha_test :
    for g in gamma_test :
        hypcomb ='-a:'+str(round(alpha,2))+' -g:'+str(round(g,1))
        print ('-a:'+str(round(alpha,2))+' -g:'+str(round(g,1)))
        ep_log, Qlearned = Qlearning_Agent(epsilon,alpha,g, reward_matrix)
        ep_log['Hyperparameter'] = hypcomb
        QLdictionary[hypcomb] = Qlearned
        if fulllog is None :
            fulllog =ep_log
        else:
            fulllog =fulllog.append(ep_log)
```

```
-a:0.01 -g:0.4
-a:0.01 -g:0.7
-a:0.01 -g:0.9
-a:0.15 -g:0.4
-a:0.15 -g:0.7
-a:0.15 -g:0.9
-a:0.2 -g:0.4
-a:0.2 -g:0.7
-a:0.2 -g:0.9
```

In [6]:

```
figure(figsize=(12,8), dpi=80)
for h in fulllog['Hyperparameter'].drop_duplicates():
    temp =  fulllog[fulllog['Hyperparameter'] == h]
    plt.plot(temp['Cumulative Rewards'], label = h)
```

```
plt.title('Reward per Episode and hyperparameter')
plt.xlabel('Episodes')
plt.xlabel('Cumulative Reward')
plt.legend(fancybox = True, bbox_to_anchor=(1.1, -0.1), ncol = 3 )
plt.show()
```




In [7]:

```
hyperparametergrid = []
for h in fulllog['Hyperparameter'].drop_duplicates():
    temp =  fulllog[fulllog['Hyperparameter'] == h]
    maxreward = temp['Cumulative Rewards'].max()
    minreward = temp['Cumulative Rewards'].min()
    stdreward = temp['Cumulative Rewards'].std()
    meanreward = temp['Cumulative Rewards'].mean()
    print ('For ' +h + ' max,min,std reward : ' + str(round(maxreward,2)) +', '+str(
          +str(round(stdreward,2))+', '+str(round(meanreward,2)))
```

```
For -a:0.01 -g:0.4 max,min,std reward : 13867.75, 6994.93, 1845.22, 12821.49
For -a:0.01 -g:0.7 max,min,std reward : 13915.17, 12238.25, 398.42, 13343.74
For -a:0.01 -g:0.9 max,min,std reward : 13984.6, 11833.2, 659.44, 13001.18
For -a:0.15 -g:0.4 max,min,std reward : 13888.66, 11475.95, 292.65, 13489.37
For -a:0.15 -g:0.7 max,min,std reward : 13903.51, 10186.32, 373.52, 13475.98
For -a:0.15 -g:0.9 max,min,std reward : 14012.58, 9123.96, 526.78, 13257.69
For -a:0.2 -g:0.4 max,min,std reward : 13870.5, 13121.0, 179.55, 13536.49
For -a:0.2 -g:0.7 max,min,std reward : 13905.45, 8643.47, 509.36, 13466.48
For -a:0.2 -g:0.9 max,min,std reward : 13814.42, 10332.82, 373.68, 13305.18
```

In [33]:

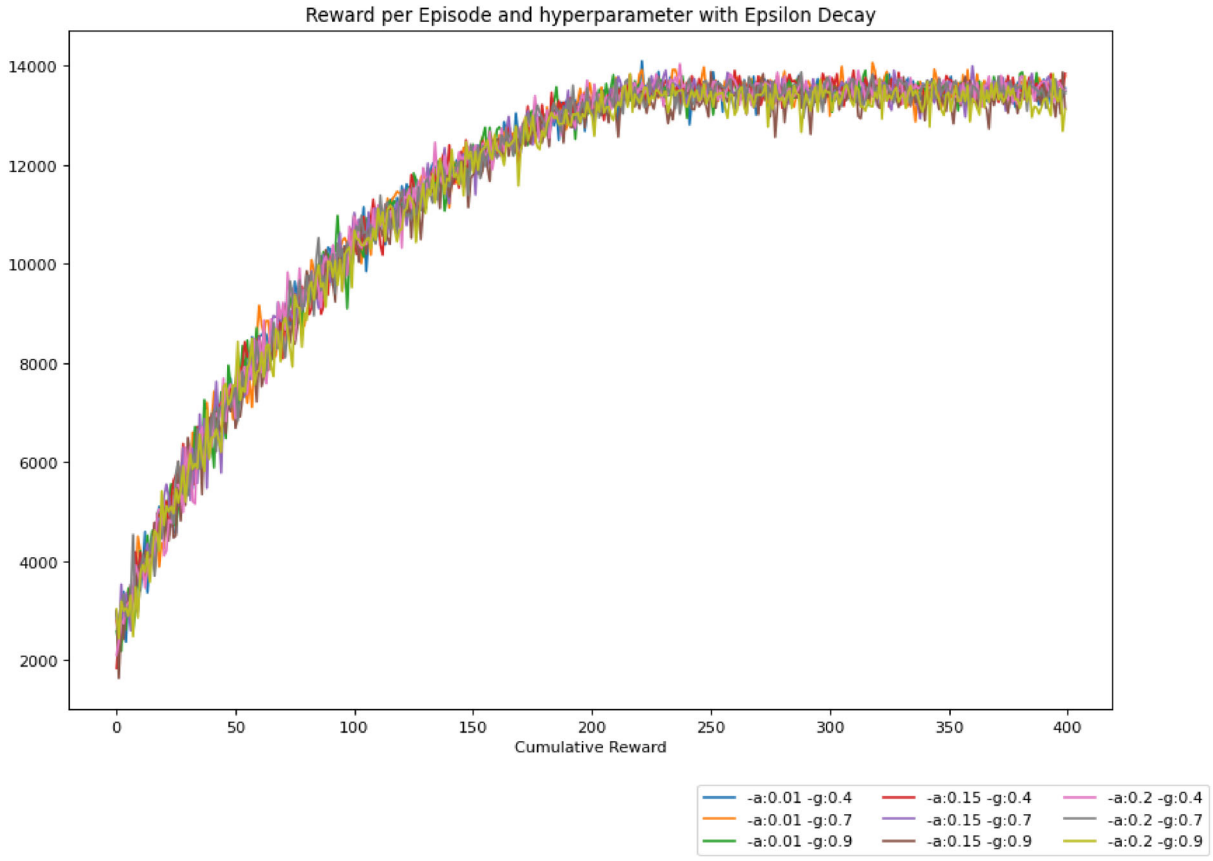
```
alpha_test = [0.01, 0.15, 0.2]
gamma_test = [0.4,0.7,0.9]
starting_epsilon = 0.9


fulllogd = None
QLdictionaryd = {}
for alpha in alpha_test :
```

```
    for g in gamma_test :
            hypcomb ='-a:'+str(round(alpha,2))+' -g:'+str(round(g,1))
            print ('-a:'+str(round(alpha,2))+' -g:'+str(round(g,1)))
            ep_log, Qlearned = Qlearning_Agent(starting_epsilon,alpha,g, reward_matr
            ep_log['Hyperparameter'] = hypcomb
            QLdictionaryd[hypcomb] = Qlearned
            if fulllogd is None :
                fulllogd =ep_log
            else:
                fulllogd =fulllogd.append(ep_log)
```

```
-a:0.01 -g:0.4
-a:0.01 -g:0.7
-a:0.01 -g:0.9
-a:0.15 -g:0.4
-a:0.15 -g:0.7
-a:0.15 -g:0.9
-a:0.2 -g:0.4
-a:0.2 -g:0.7
-a:0.2 -g:0.9
```

In [34]:

```
figure(figsize=(12,8), dpi=80)
for h in fulllogd['Hyperparameter'].drop_duplicates():
    temp =  fulllogd[fulllogd['Hyperparameter'] == h]
    plt.plot(temp['Cumulative Rewards'], label = h)

plt.title('Reward per Episode and hyperparameter with Epsilon Decay')
plt.xlabel('Episodes')
plt.xlabel('Cumulative Reward')
plt.legend(fancybox = True, bbox_to_anchor=(1.1, -0.1), ncol = 3 )
plt.show()
```




In [35]:

```
for h in fulllogd['Hyperparameter'].drop_duplicates():
    temp =  fulllogd[fulllogd['Hyperparameter'] == h]
    maxreward = temp['Cumulative Rewards'].max()
```

```
    minreward = temp['Cumulative Rewards'].min()
    stdreward = temp['Cumulative Rewards'].std()
    meanreward = temp['Cumulative Rewards'].mean()
    print ('For ' +h + ' max,min,std reward : ' + str(round(maxreward,2)) +', '+str(
           +str(round(stdreward,2))+', '+str(round(meanreward,2)))
```

```
For -a:0.01 -g:0.4 max,min,std reward : 14095.11, 2367.69, 2913.97, 11527.4
For -a:0.01 -g:0.7 max,min,std reward : 14059.42, 2362.89, 2910.2, 11566.36
For -a:0.01 -g:0.9 max,min,std reward : 13904.62, 2186.37, 2910.79, 11521.41
For -a:0.15 -g:0.4 max,min,std reward : 13901.55, 1837.71, 2891.83, 11543.85
For -a:0.15 -g:0.7 max,min,std reward : 13991.14, 2429.46, 2891.02, 11512.42
For -a:0.15 -g:0.9 max,min,std reward : 13881.99, 1636.86, 2889.41, 11368.69
For -a:0.2 -g:0.4 max,min,std reward : 14034.55, 2103.49, 2928.78, 11538.98
For -a:0.2 -g:0.7 max,min,std reward : 13870.39, 2536.28, 2911.06, 11490.7
For -a:0.2 -g:0.9 max,min,std reward : 13835.43, 2434.76, 2883.79, 11378.11
```

In [ ]: