GROUP WORK PROJECT # 3
GROUP NUMBER: 6524

MScFE 622: Stochastic Modeling

FULL LEGAL NAME	LOCATION (COUNTRY)	EMAIL ADDRESS	MARK X FOR ANY NON-CONTRIBUTING MEMBER
Shong Xian Lee	Singapore	shongxian5601@gmail.com	
Emmanuel Hansingo	Zambia	emmanuel.hansingo@gmail.com	
Ketty Muwowo	Zambia	kettymuwowo@gmail.com	

Statement of integrity: By typing the names of all group members in the text boxes below, you confirm that the assignment submitted is original work produced by the group (excluding any non-contributing members identified with an "X" above).	
Team member 1	Shong Xian Lee
Team member 2	Emmanuel Hansingo
Team member 3	Ketty Muwowo

Use the box below to explain any attempts to reach out to a non-contributing member. Type (N/A) if all members contributed. **Note:** You may be required to provide proof of your outreach to non-contributing members upon request.

Step 2

Description

There could be a tradeoff between exploration and exploitation from reinforcement learning. It is because the value of alpha and greedy policy will alter the capabilities of exploration and exploitation. It should find a point to leverage both states. But there is a problem of lack of knowledge of the transitions between different states. Thus, the only action we can do is to learn from the past and focus on stocks with higher scores. A higher alpha value means faster adaptability toward current data. Greater greedy policy value means lesser exploration and focus more on exploitation. Through the paper of Huo, there are two input parameters; delta and number of trials that are required.

Pseudocode

BEGIN

// Step 1: Initialize Parameters

Initialize number of trials, N

Initialize delta (δ) as the time interval for each trial

Set K, the number of assets (K-armed bandit problem)

Set exploration parameter (epsilon or greedy policy value)

Set learning rate, alpha (higher alpha means faster adaptability)

// Step 2: Historical Data Analysis

For each asset i in K:

Compute historical return over time changes, delta

Set initial values (mean, std) for a normal distribution based on historical returns

// Step 3: Sequential Decision-Making

For each trial t = 1 to N:

Select portfolio weightage T = (ω1,t,...,ωK,t)

// Step 4: Action Selection and Update

For each step in total number of trials:

If a significant change is detected in the reward signal:

Update the Q-function using the fixed alpha method

Record the action taken and update expected rewards

// Step 5: Portfolio Evaluation

Observe return R of selected assets

Calculate the reward based on portfolio weightage and returns

Update portfolio weights based on observed rewards

// Step 6: Evaluation and Comparison

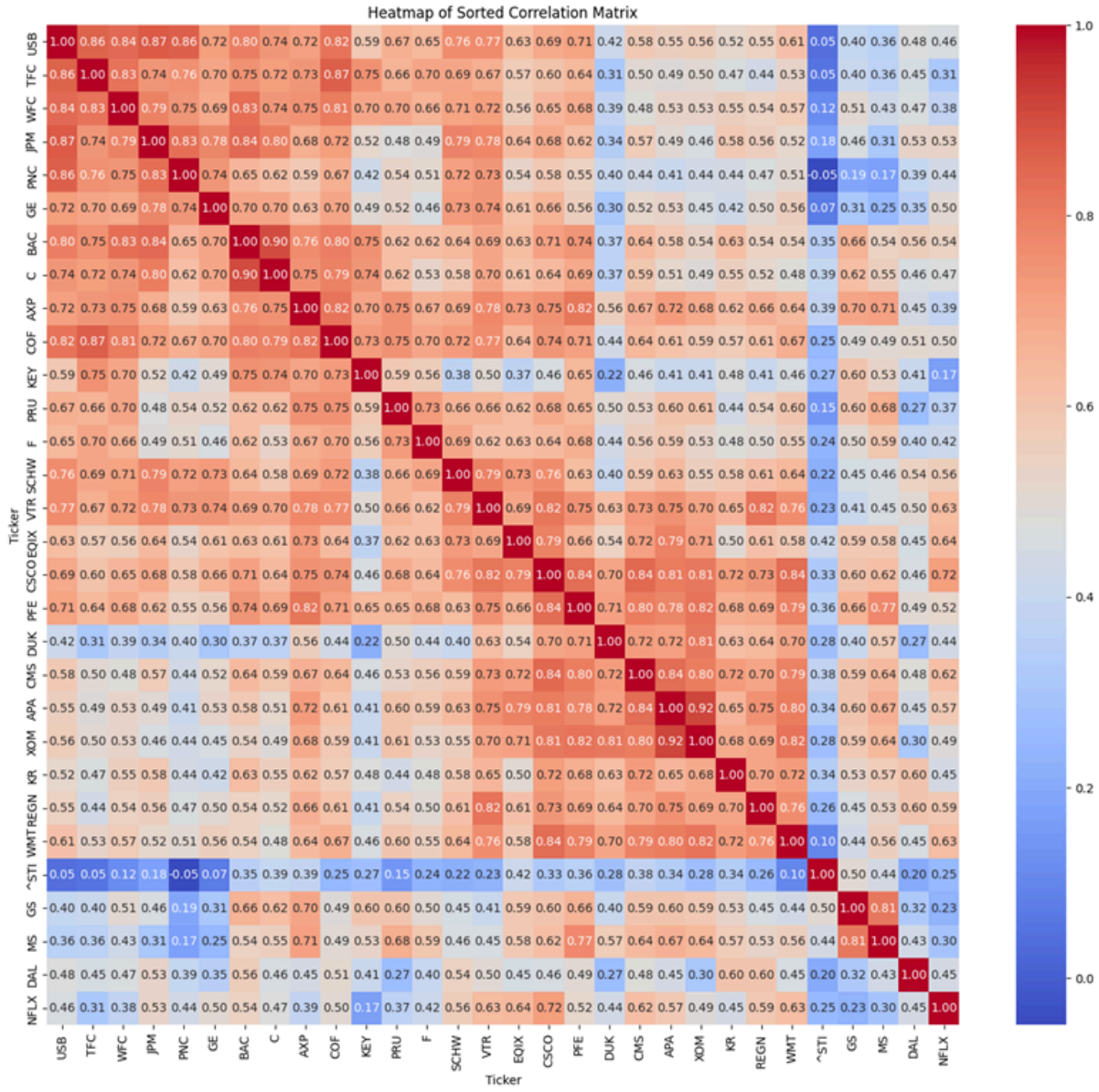
Display comparison plot of:

- Portfolio return vs. optimal action

- Average return vs. optimal action

END

Step 4: Correlation matrix (sorted)



We chose hierarchical clustering using the Ward method to sort the securities based on their pairwise correlations. Ward's method was used because it minimizes the total variance within the clusters. Pairwise correlations help identify the pairs of financial institutions that have similar or different behavior, which can be useful in clustering and other analyses. This method was used because it helps to group similar stocks, thereby making it easier to visualize clusters of highly correlated stocks. This approach is ideal because it makes use of the entire correlation structure and reduces the distances between similar stocks in the matrix. By sorting this way, the heatmap shows which companies behave similarly in terms of their stock price movements by grouping them together, allowing for more informed decisions on portfolio diversification.

Step 5: UBC action

Using the multi-armed bandit methodology studied in Module 6, students will get together and discuss the performance of the algorithm using the Upper-Confidence Bound (UBC) algorithm as mentioned in the module's lesson notes and not the Huo paper. This discussion will help the team understand the algorithm in order to replicate it in the next step. As a reminder, the assigned reading from Sutton/Barto Chapter 2 will help for this and the next few questions. Note: there are no deliverables required for this step.

Reinforcement learning is adopting training information to evaluate the action where maximizing the reward and reducing the level of regret rather than providing instruction. Thus, it required sustainable exploration action, instead of relying on one method (Exploration & exploitation). The number of K-armed bandit problems depends on how many repeated times have been set. It will tend to maximize the reward during the K-times (round). While there is a benchmark, q*(a) that actual reward Qt(a) demands to match with it. If we want to keep the number of actions, while increasing the rewards at the same time, it can work out through adjusting the epsilon greedy policy. If the epsilon greedy value increases, this means the model is exploring and finding better alternative options. In the opposite direction, reduction of epsilon greedy value described focuses on short term information and tends to exploit the information to maximize the return. The equation of greedy action selection method can be observed as follows:

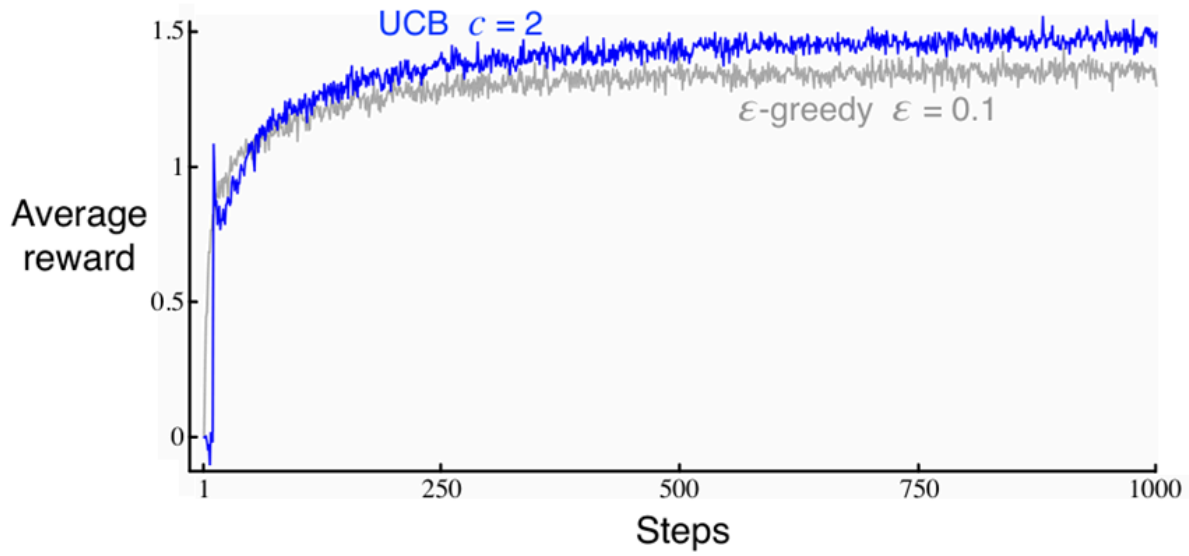
$$A_t \doteq \arg\max_a Q_t(a),$$

A balance between exploration and exploration is required as the accuracy of current does not represent as accurate in the future. Although the result of greedy action looks nice at present, it could miss out some alternative ways that performed well. After leveraging the pros and cons, it is recommended to select an optimal non-greedy action that closes enough toward the maximal reward and reduces the uncertainties. Upper confidence bound (UCB) action is capable of identifying the balanced point. The equation of the action can be interpreted as below:

$$A_t \doteq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

The natural logarithm, In denote with t refers to the duration, Nt(a) describes the number of times that action a has been selected corresponding to the time, t. C refers to the degree of exploration with an assumption of always greater than 0. The square root inside the equation is measuring uncertainty or variance that estimate within action a. The argmax is retrieved the max value from the possible true value of action a with the confidence level, c (Barto, 2018) (Huo & Fu, 2017).

Each round of action uncertainty is expected to reduce. It is because the use of natural logarithm as the number of t increases over the time, it will lead to increasing use of natural logarithm and result in a smaller value at the very last step which means lower down the regret level. The line graph shown in Barot (2018) is able to deliver the meaning of it and it became flattened at the end.


UCB $c = 2$
ε-greedy $\varepsilon = 0.1$
Average reward
1.5
1
0.5
0
1
250
500
750
1000
Steps

Step 6

Member A

Begin

#Step 1 Import Ticker

OBTAIN Stock information (eg. Adjustment close) via yahoo finance with tickers

#Step 2 Set parameters

SET number of arms, NK, epsilon, alpha, nepisodes (number of round), hold

OBTAIN remaining Total period by minus hold value

#Step 3 Initialize q function and action record

FOR the range in number of rounds, nepisodes

SET q value and action record

FOR the range in number of total periods

OBTAIN the value for optimal action

CALCULATE the rewards across the holding period

COMPUTE Q function based on chosen action, a's holding period

INITIALIZE upper boundary of Q value

FOR the range in the number of arms,

IF number of actions equal to zero

COMPUTE Q value's upper limit with an increment of 1

ELSE compute the q value according to number of action and the square root of natural logarithms /number of actions **(qvalue[aa] + sqrt(log(tt + 1) / nactions[aa]))**

COMPUTE the reward over action period

COMPUTE the average reward value across total period were divided by nepisodes

COMPUTE the optimal average across total period where optimal value of action divided by nepisodes

ENDFOR

#Step 4 visualization

DISPLAY the plot where comparison of average return and UCB method

Member B

Member B implements the steps in Python.

Member C

- The code starts by importing the numpy library for array creation and manipulation.
- **num_stocks**: Counts the number of different stocks, which are the "arms" in the multi-armed bandit problem.
- **num_days**: Counts the number of trading days over which the algorithm will run.
- **estimated_values**: Array to keep track of the estimated mean return for each stock.
- **selection_counts**: Array to keep track of the number of times each stock has been selected.

- **ucb_cumulative_rewards**: Array to track the cumulative rewards earned over the days.
- We then have a loop that iterates through every trading day from day 1
- **ucb_values**: Calculated for each stock using the formula:

$$UCB_i = \mu_i + \sqrt{\frac{2log(t)}{n_i}}$$

- μ is the estimated value of the stock
- t is the current day (time)
- n is the number of times the stock has been selected
- **1e-5**: This is a small constant added to avoid division by zero when a stock has not been selected yet.
- **selected_stock**: Index of the stock with the highest UCB value, indicating the stock to select for the current day.
- **reward**: The actual return of the selected stock on the current day.
- **selection_counts[selected_stock]**: Increments the count for the selected stock
- **estimated_values[selected_stock]**: Updates the estimated mean return for the selected stock using the incremental mean formula.
- **ucb_cumulative_rewards[day]**: Adds the reward from the current day to the cumulative total.
- Lastly the output prints out the final selected stock and the estimated values

Step 7

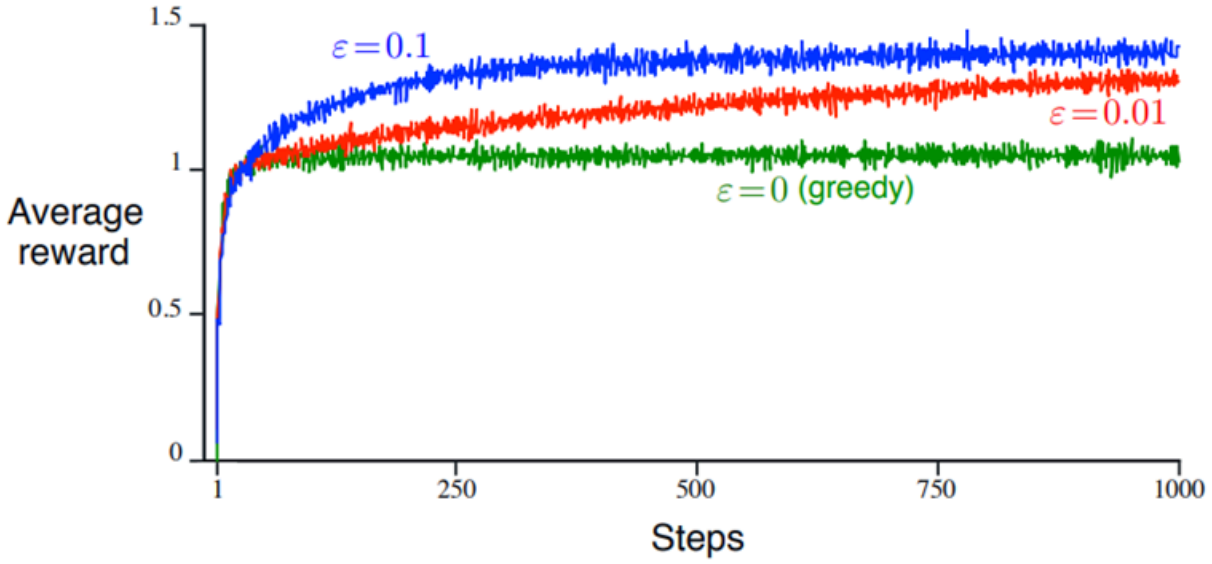
As mentioned in step 5, the prior objective of greedy policy tends to maximize action's return. With the epsilon greedy policy, it is able to adjust the extent of exploration and exploitation. The value of epsilon-greed will directly affect the greedy policy. Greater epsilon greedy value (less greedy) means the model is exploring and finding better alternative options. Lower epsilon greedy value (greater greedy) means the model is exploiting and focuses on short term results.

The reason behind the epsilon-greedy policy is that epsilon (ϵ\epsilonϵ) determines the probability of exploring alternative actions. A higher epsilon value introduces a non-zero

probability of selecting any action randomly, even if it might not be the best fit according to the current information. This encourages the agent to try out different actions, potentially discovering better strategies in the long run. In contrast, a lower epsilon value favors exploiting the current knowledge by consistently choosing the action that appears to yield the highest immediate reward.

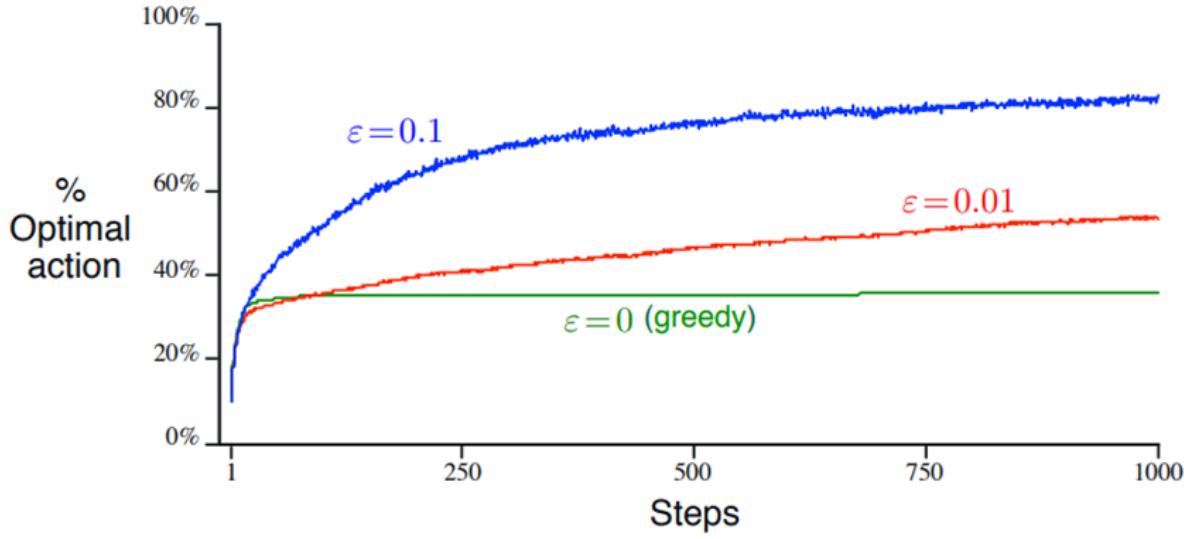
The benefit of the epsilon-greedy method is that it balances exploration and exploitation, allowing the agent to improve its understanding of the environment over time. Even though the number of steps taken by the agent is limited, the use of a non-zero epsilon ensures that all actions have a chance of being selected, leading to more comprehensive learning. This process helps the agent's estimated action-value function Q(s,a)Q(s, a)Q(s,a) to approach the true optimal action-value function q*(s,a)q^\ast(s, a)q*(s,a), ensuring more consistent rewards.

As highlighted in Sutton and Barto (2018), the epsilon-greedy method with a moderate epsilon value (e.g., 0.1, which allows for greater exploration) tends to achieve higher average rewards compared to lower epsilon values (e.g., 0.01 or 0), which focus more on exploitation.



With a higher epsilon value (greater exploration), the agent is more likely to find optimal actions by exploring alternative solutions, which can be beneficial in the long term. However, continuous exploration can lead to missed opportunities for

exploiting the best-known actions, while excessive exploitation might ignore other potentially good actions.



Therefore, it is crucial to identify a balance between exploration and exploitation. A well-balanced epsilon-greedy policy enables the agent to explore enough to discover better strategies while exploiting the best-known actions to maximize immediate rewards.

Step 8

Team Member B

Pseudocode for the Epsilon-Greedy Algorithm

BEGIN

// Step 1: Initialize the parameters

epsilon = 0.1 # The probability of exploration (between 0 and 1)

number_of_arms = N # The number of bandit arms or actions

counts = array of size number_of_arms, initialized to 0 #Counts of the number of times each arm was pulled

values = array of size number_of_arms, initialized to 0 #Estimated values of each arm

// Step 2: Define a Function to select the arm to pull

function select_arm():

random_value = random number between 0 and 1

if random_value > epsilon:
Exploitation: choose the arm with the highest estimated value

return index of the arm with the maximum value in values array

else:

Exploration:

choose a random arm

return a random index between 0 and number_of_arms - 1

// Step 3 : Define a Function to update the estimated values of the arms

function update_values(arm_index, reward):

Update the counts for the selected arm

counts[arm_index] = counts[arm_index] + 1

Calculate the new estimated value for the selected arm

using incremental update formula to avoid summing over all past rewards
n = counts[arm_index]
value= values[arm_index]
new_value = value + (1 / n) * (reward - value)
Update the estimated value in the values array
values[arm_index] = new_value

// Step 4: Create a main loop for each episode or trial for each episode:

// Step 5: Select an arm using the epsilon-greedy strategy

arm_to_pull = select_arm()

// Step 6: Pull the selected arm and observe the reward
reward = pull_arm(arm_to_pull) #This is a hypothetical function that simulates pulling an arm
// Step 7: Update the estimated values based on the observed reward

update_values(arm_to_pull, reward)

#The arm with the highest estimated value is considered the best arm to pull

END

Team Member C

Member C implements those steps in Python.

Team member A

Before starting the process, it has initialized the parameters that are required for an epsilon-greedy algorithm. First of all, the probability of exploration (epsilon) has been set as 0.1 and the number of bandit arms or action will initialize in the variable of number of arms. The Count variable stored the number of arms that were pulled. While values variables stored the estimated value of each arm.

In step 2, it begins to define the function to select the arm to pull which is called select_arm. Moving into this function, it sets a random number between 0 and 1 in the random_value variable. After creating, IF-THEN_ELSE used, if random_value is greater than epsilon, it will proceed with exploitation action where choose the arm with highest estimated value and return index of the arm with the maximum value in the values array.

ELSE, it will proceed with an exploration method where choose a random arm and return a random index between 0 and number_of_arm -1.

Step 3, it will structure a function called update_value with two input parameters called arm_index and reward. Follow by updating the counts for the selected arm. Then, count the arm_index that results from arm_index add up with 1. Afterward, it will calculate the new estimated value for the selected arm and use an incremental update formula to avoid over-summing with existing rewards. Moving forward, it counts the number of arm_index and is stored in the variable n. For the value variable, it stored the values of arm_index. The value in variable, new_value is computed using the value variable add with 1/n times with the difference between reward and value. After doing calculation, it will update the estimated value in the values array that is stored in the armed_index variable.

In step 4, it will create a main loop for each episode or trial for each episode and step 5 will select an arm using the epsilon-greedy strategy that is stored in the arm_to_pull variable. In step 6, it will pull the result from arm_to_pull and observe the reward. It will result in a reward variable. Final step, it will update the estimated value based on the observed reward. The decision-making process will be selected with the highest estimated value.

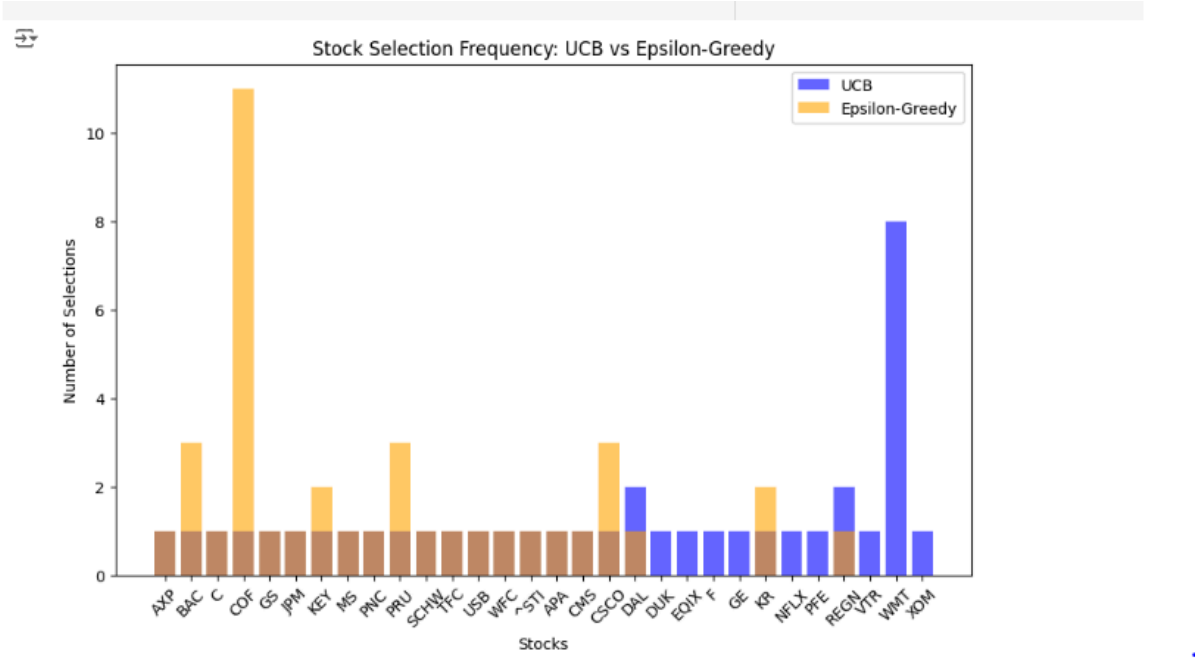
Step 9

Comparison of the UCB and Epsilon-Greedy Algorithms

1. Description of Our Results

In our analysis, we ran both the Upper Confidence Bound (UCB) and Epsilon-Greedy algorithms to select financial and non-financial stocks using daily returns from Sep 2008 and Oct 2008.

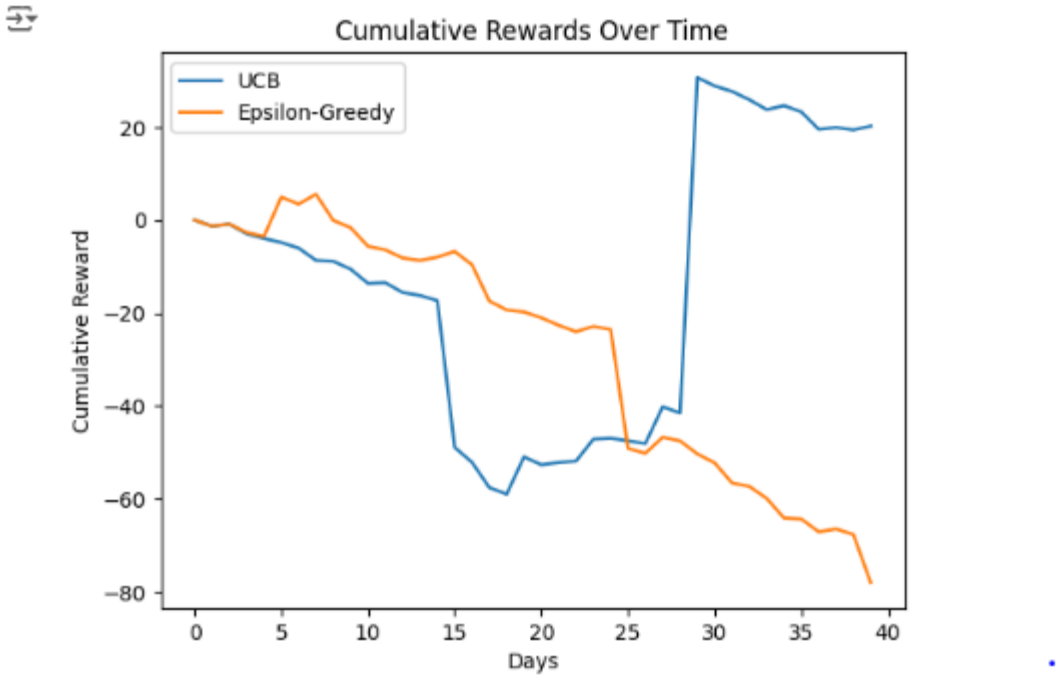
Based on the procedures performed we obtained the Stock Selection Frequencies shown in the plot below



The UCB algorithm showed a strong preference for certain stocks, particularly WMT, selecting it much more frequently compared to others.

The Epsilon-Greedy algorithm had a more distributed selection pattern but showed a noticeable preference for COF, choosing it significantly more often than other stocks.

Based on the Cumulative Rewards Over Time as show below



Initially, both algorithms performed similarly, but as time went on, they diverged. The UCB algorithm eventually led to a sharp increase in cumulative rewards around day 30, although it experienced a steep decline earlier.

On the other hand, the Epsilon-Greedy algorithm's cumulative rewards remained relatively stable but did not experience the same sharp increases or decreases as the UCB algorithm.

2. Comparison with Huo's Paper

When comparing our results with the findings in Huo's paper, several key differences emerge:

Selection Patterns: Huo's paper may have demonstrated different stock preferences based on the dataset and parameters used. The UCB algorithm in our results displayed a more aggressive approach, leading to significant fluctuations in cumulative rewards.

The Epsilon-Greedy approach in our study was more conservative, which aligns with Huo's observations.

Cumulative Reward Trends: The sharp increase in UCB's rewards around day 30 is a crucial point of divergence. If Huo's paper shows a more consistent trend or different timing for such changes, this could highlight the impact of dataset or algorithm tuning.

Epsilon-Greedy's more stable yet lower cumulative reward might suggest it is less risky but potentially less rewarding in certain market conditions, a point that could be further analyzed against Huo's conclusions.

3. Presentation of Key Differences

Stock Selection Frequency Graph: This graph visually emphasizes the UCB algorithm's focus on a few stocks versus the more evenly distributed approach of Epsilon-Greedy. The distinct peaks in UCB's selections compared to the spread in Epsilon-Greedy are significant and may indicate different risk-taking strategies.

Cumulative Rewards Graph: The cumulative rewards graph highlights the volatility of UCB compared to the steadier path of Epsilon-Greedy. The dramatic spike and subsequent dip in UCB's rewards are critical points for discussion, especially when compared to Huo's findings. If Huo observed different patterns, this could suggest the sensitivity of UCB to specific market conditions or its ability to capitalize on sudden opportunities, which is less evident in Epsilon-Greedy's performance.

Step 11

It was observed that in the case of the UCB, using data from September - October 2008, the algorithm selected the PFE as the preferred stock and this result changed to DUK when considering more recent data, June- July 2024. Using more recent data improved the adaptability of both UCB and epsilon-greedy algorithms in the context of changing market conditions. UCB was shown to be more robust and balanced, while epsilon-greedy was more aggressive in exploiting recent trends. This aggressive exploitation also meant that repeated iterations of the algorithm on the same data resulted in the algorithm picking different stocks. Since epsilon-greedy directly responds to the immediate performance of stocks, there is a risk of overfitting to recent trends. This could lead to suboptimal performance if the market environment changes again. However, since the algorithm is quick to adapt to recent changes in stock performance, especially when given an epsilon value that allows for sufficient exploration. Stocks that have recently started performing better were selected more frequently as the algorithm updated its estimated values. The performance of UCB was generally stable due to its emphasis on both exploration and exploitation. Due to its robustness, it tends to perform well even with fluctuating data, as the confidence interval widens for less frequently selected stocks, ensuring that all options are periodically re-evaluated.

References

Barto, R. S. (2018). *Reinforcement learning: an introduction* . CA 94042, USA.

Huo, X. g., & Fu, F. (2017). Risk-aware multi-armed bandit problem with application to portfolio selection . *Royal Society Open Science*.