Assignment: Stock Trading

Objective and Description

In this assignment, you will design a simplified stock trading environment and evaluate the performance of reinforcement learning algorithms within it. The objective is to maximize the total portfolio value by making trading decisions at the end of each trading day. For each stock ticker, you may perform only one action per day—either hold, sell up to N shares at the closing price, or buy up to N shares at the closing price.

- You are required to submit your **code and report** via KLMS. Please read the following requirements for the detail:

Environment Description:

- Period : Each time step corresponds to a single trading day.
- State : Define your own state representation. For example, it may include features such as the opening price, closing price, and trading volume of the corresponding day.
- Action : For each stock ticker, you may perform only one action per day—either hold, sell up to N shares at the closing price, or buy up to N shares at the closing price.
- Reward : Define your own reward function.

Coding Implementaion:

1. **Environment Construction**
    - (Option) We recoommend yfinance module to query data stock for the implementation. But this is not mandatory.
    - Implement a **Gym-like trading environment** that allows trading of N-different code of stocks.
        - You should use gymnasium.Env to create your custom environment.
            - Refer to this link about gymnasium.Env.
        - You should define your state space and action space using **spaces object** (gymnasium.spaces).
            - Refer to this link about gymnasium.spaces.
        - Implement the following methods:
            - reset function
                - returns (state: YourStateType, info:Dict)
            - step function
                - input (action: YourActionType)
                - returns (next_state : YourStateType, reward: float, done: bool, truncated : bool, info:Dict)
                - The agent should be able to:

        - Trade up to N different code of stocks.
        - **Sell** any number of the stocks currently owned at the **closing price**.
    - The following is an example of the enviornment class:

```
  import gymnasium as gym
  from gymnasium import spaces
  import yfinance as yF


  class YourEnv(gym.Env):

      def __init__(self, balance:float,
                         start_date:str = "2020-10-19",
                         end_date:str = "2024-10-25",
                         code:str = "AAPL"):


        # Define your state space and action space.
        # The folloiwng is an example:

        self.observation_space = spaces.Dict(
        {
            "agent": spaces.Box(
                low=np.array([0, 0, 0], dtype=np.float32),
                high=np.array([np.inf, np.inf, np.inf],
dtype=np.float32),
                shape=(3,),
                dtype=float
            ),

            "market": spaces.Box(
                low=data.min().to_numpy(dtype=np.float32),
                high=data.max().to_numpy(dtype=np.float32),
                shape=(len(self.column_names),),
                dtype=float
            )
        })

         # We have 3 actions, corresponding to "sell", "buy",
"hold".
        self.action_space = spaces.Discrete(3)

        start_date = dt.datetime.strptime(start_date, "%Y-%m-%d")
        end_date = dt.datetime.strptime(end_date, "%Y-%m-%d")

        ticker = yF.Ticker(code)
        df = ticker.history(interval='1d',
```

```
                                start=start_date.strftime("%Y-%m-%d"),
                                end=end_date.strftime("%Y-%m-%d"),
auto_adjust=False)

      def step(self, action):

        # implemnt your logic

      def reset(self):

        # implement your logic
```

2. **Train and Evluation**
    - $N = 1$ case (Single Ticker):
        - Train
            - Use a single stock for training.
            - The agent should start with a random initial balance.
            - Optimize the reward function by applying appropriate scaling or shaping to achieve the best trading performance during the training period.
        - Test
            - Evaluate the trained agent under the following test scenarios: a. Evaluate on the same stock ticker used for training. b. Evaluate on three different stock tickers that were not used during training.
            - You should evaluate the trained agent on data from a period not used during training.
    - $N > 1$ case (Multiple Tickers):
        - Train
            - Use 4 different stock tickers for training.
            - The agent should start with a random initial balance.
        - Test
            - Evaluate the trained agent on data from a period not used during training, using the following test scenarios:
                1. Seen-Ticker Test: Evaluate on the same 4 training tickers.
                2. Mixed Test Set:
                    - Use 2 training tickers + 2 unseen tickers; start with a random initial balance
                3. Use 4 entirely new stock tickers not seen during training
    - You should report the final portfolio value on the last day.
      $\sum_i (\text{Shares of i}) * (\text{Closed Price of i at the End of the Period}) + \text{Remaining balance}$

.
    - Training data can be chosen arbitrarily, but test data must exclude any time period used during training.

3. **Algorithm Comparison**
    - Select and implement **two different algorithms** (e.g., DQN vs. PPO, or any pair of your choice).
    - Train and evaluate both algorithms within your trading environment.
    - Compare their performance based on quantitative metrics. Provide a mean variance plot.

Report

- Code implementation of your custom environment.
- Description of the environment design (state and action space, reward function).
- Experimental setup and performance comparison results.
- Brief discussion of findings.
    - After completing the experiments, analyze and interpret the results in your report.
    - Discuss how the reward function design influenced the agent's trading performance and whether the trained policy generalized effectively to other stock tickers.
    - Provide insights on what factors contributed to success or failure and suggest possible improvements for future experiments.