SIGMA WEDGE HACKATHON (UPDATED)

NAME: KIRUTHIKA V G
ROLL NO: 21PT10

Introduction:

This program aims to build a simple model to make decisions on certain days regarding whether to place a buy order trade for the following day, aiming to maximize the portfolio value. This is achieved by classifying the daily returns into different states (Bull, Flat, Bear) and defining a value function to determine the portfolio value based on these states.

Pull the price data:

- It leverages QuantRocket functionalities - Moonshot, a module for backtesting trading strategies. It begins by cloning the "moonshot-intro" strategy, possibly containing trading logic or signal generation algorithms.
- Historical price data is collected using QuantRocket's data collection capabilities, universes, akin to Moonshot's data sourcing process.
- The retrieved data is filtered and organized into a universe of securities, mirroring Moonshot's universe creation step. Subsequently, the code fetches historical prices for **AAPL stock** within the designated universe and date range, similar to Moonshot's data access methods.
- Display options are configured to ensure a comprehensive output presentation. The retrieved prices are then processed and stored in a pandas DataFrame, facilitating further analysis.
- Additionally, manual creation of date strings aligns with corresponding closing prices, enabling efficient extraction, organisation, and presentation of historical price data for subsequent financial analysis or algorithmic trading strategies.

Algorithm:

Initially, I used a straightforward method for updating transition counts and calculating portfolio value based on observed state transitions. But the difficulties I faced are iterating the dataframe sequentially in a loop and only used for a limited amount of data. So I moved to the Q learning algorithm

for optimization, handle stochastic transitions easily and handle streaming data updates efficiently.

Q - learning algorithm(updated):

- The algorithm initializes variables including the number of possible states, transition counts, Q-values, and portfolio value. It also sets parameters such as the learning rate and discount factor.
- It defines a function to update transition counts, Q-values, and make buy decisions based on observed state transitions.
- The main loop iterates through the DataFrame to simulate the trading environment over time. It obtains the previous and current states from the DataFrame and updates the portfolio value based on buy decisions.
- Within the update function, transition counts are incremented, states are converted to integer indices, and Q-values are updated using the Q-learning update rule.
- Transition probabilities are calculated based on updated transition counts, and buy decisions are made by comparing these probabilities to determine whether to buy.
- Finally, after iterating through the data, the algorithm outputs the transition distribution and the final portfolio value.
- The algorithm incorporates key elements of Q-learning, such as updating values based on rewards and future rewards, calculating probabilities, and making decisions to maximize the portfolio value based on historical data.

Updations I have done:

- Included logic to make buy decisions based on transition probabilities calculated using the Q-learning update rule.
- The portfolio value, transition counts, Q-values, and make buy decisions are updated based on buy decisions made using transition probabilities calculated.
- It calculates the transition probabilities at each time step in a streaming fashion, maintains a record of transition counts and uses them to calculate the transition distribution, which represents the likelihood of transitioning from one state to another based on historical observations.