SIGMA WEDGE HACKATHON

NAME : KIRUTHIKA V G
ROLL NO : 21PT10

Introduction:

The objective of this program is to build a simple model to make decisions on certain days regarding whether to place a buy order trade for the following day, with the aim of maximising 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 organised 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 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.

Q - learning algorithm:

- The algorithm begins by initialising transition counts and Q-values to track state transitions and their associated values.
- It then defines parameters like the learning rate (alpha) and discount factor (gamma) to regulate the algorithm's learning rate and future reward importance.
- The core of the algorithm lies in the update_transitions_and_Q function, where transition counts and Q-values are incrementally updated based on observed state transitions using the Q-learning update rule.
- Additionally, the algorithm adjusts the portfolio value according to state transitions indicating buying or selling opportunities.
- Once the updates are complete, the transition distribution is calculated to determine the likelihood of transitioning between different states.
- Finally, the algorithm outputs the transition distribution and the final portfolio value.