## 

## 

## 

**Columbia University**

**STATGU5243**

**Final Project: Q-Learning Trading Strategy**

MuQing Wen (mw3821)

Final Report

May 18th, 2025

## 

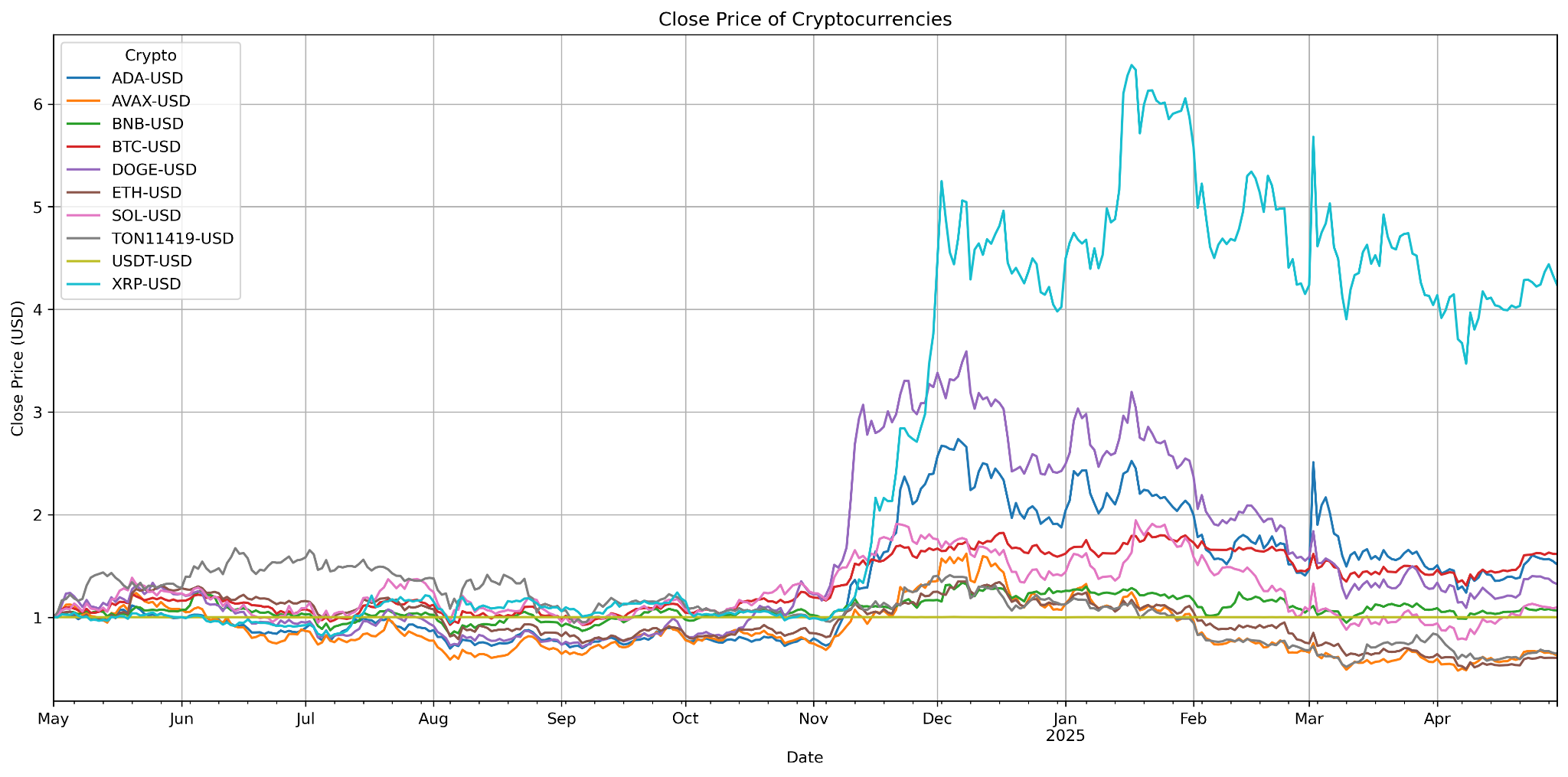
## **Part 1: Business Problem Discussion**

In volatile financial markets like cryptocurrency, making trading decisions is challenging due to the high uncertainty and non-stationary nature of asset prices. This project aims to address the business question: Can machine learning predictions of next-day prices be translated into profitable real-life trading strategies?

We apply a Q-learning framework to develop an automated trading agent that dynamically decides when to buy, hold, or sell a cryptocurrency, using only predictive signals and internal portfolio status as its state representation. The objective is to maximize portfolio value over time by learning optimal policies through repeated interaction with the trading environment.

## **Part 2: Data Overview**

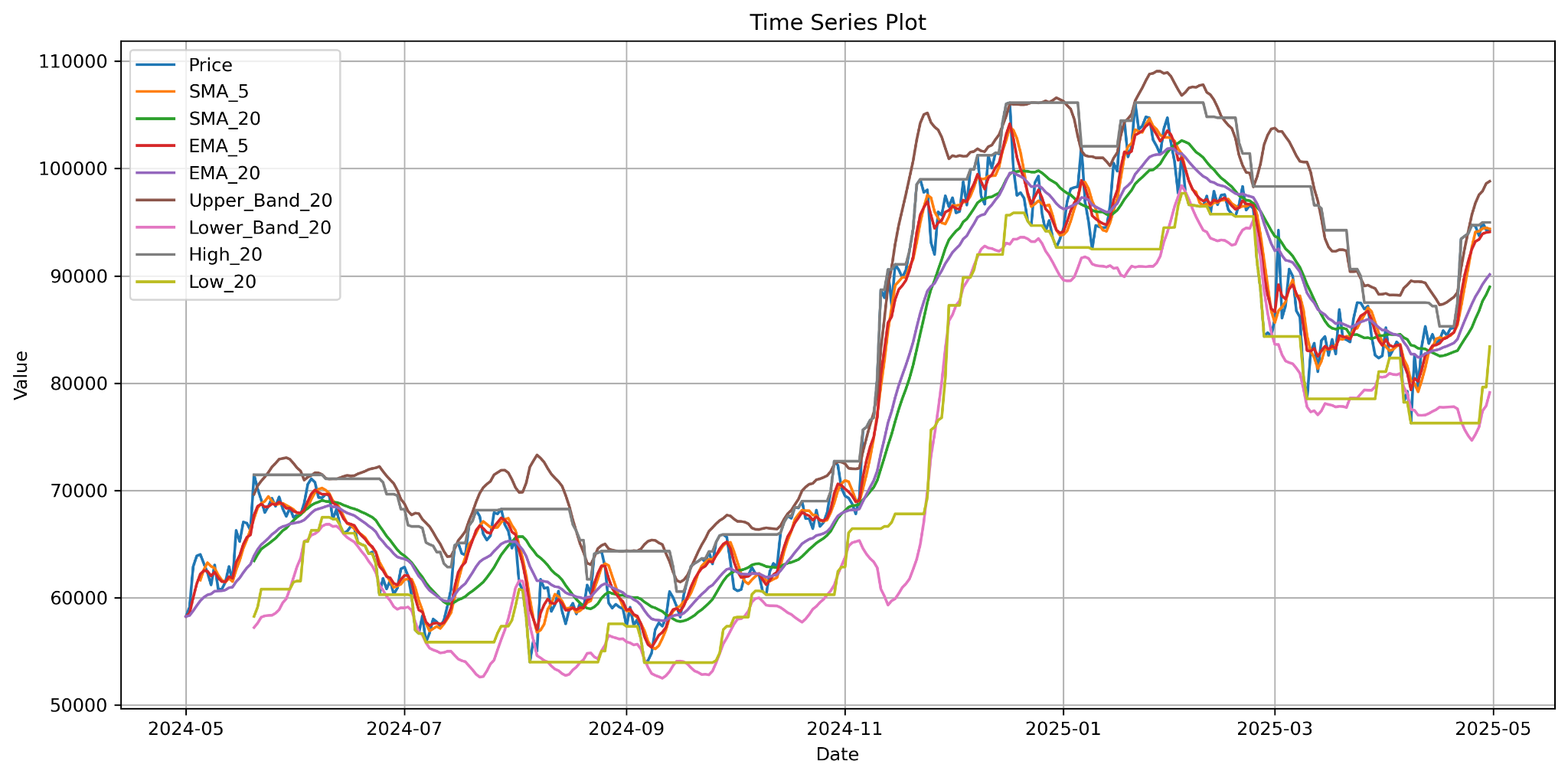
This dataset contains daily closing prices of 10 major cryptocurrencies from May 2024 to April 2025. Each asset has 365 data points, representing one year of trading days. The line chart below shows normalized closing prices, this allows for a direct comparison of relative growth and decline, regardless of the original price level.



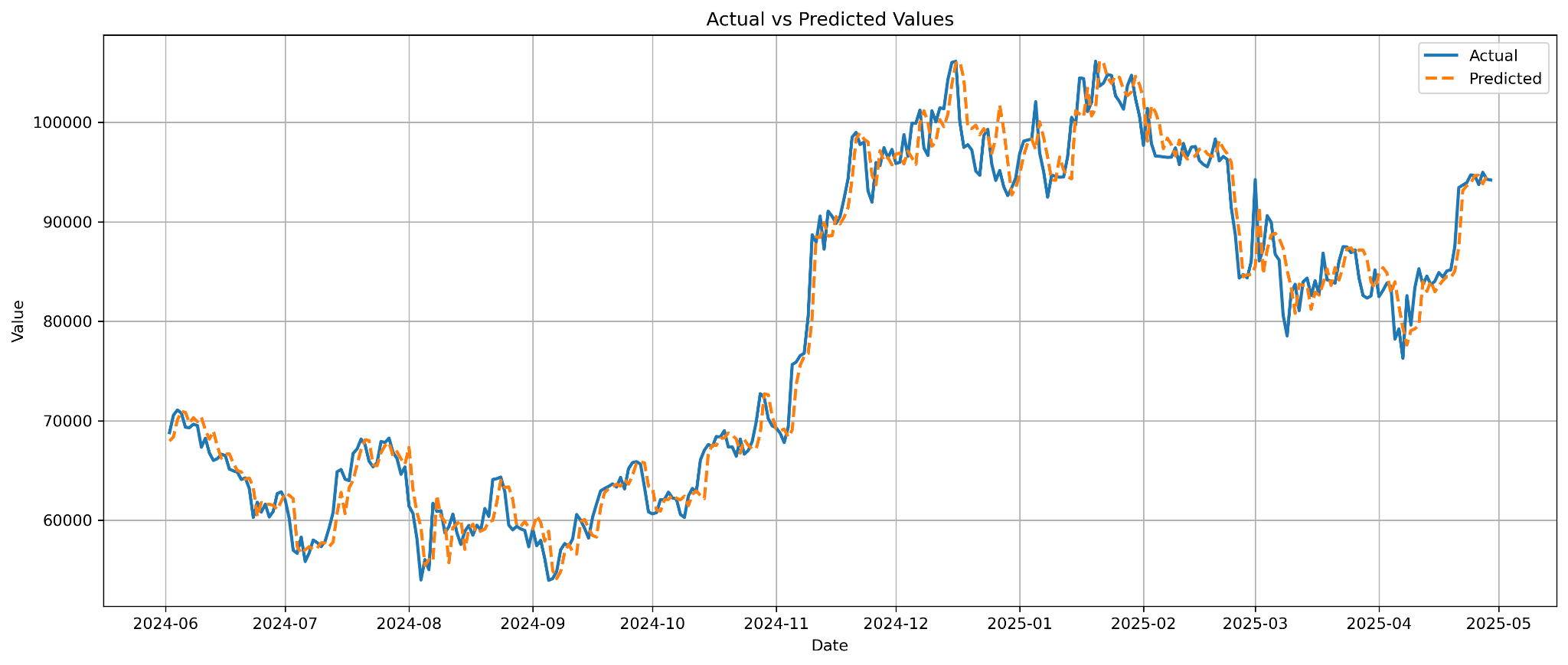
## **Part 3: Methodology**

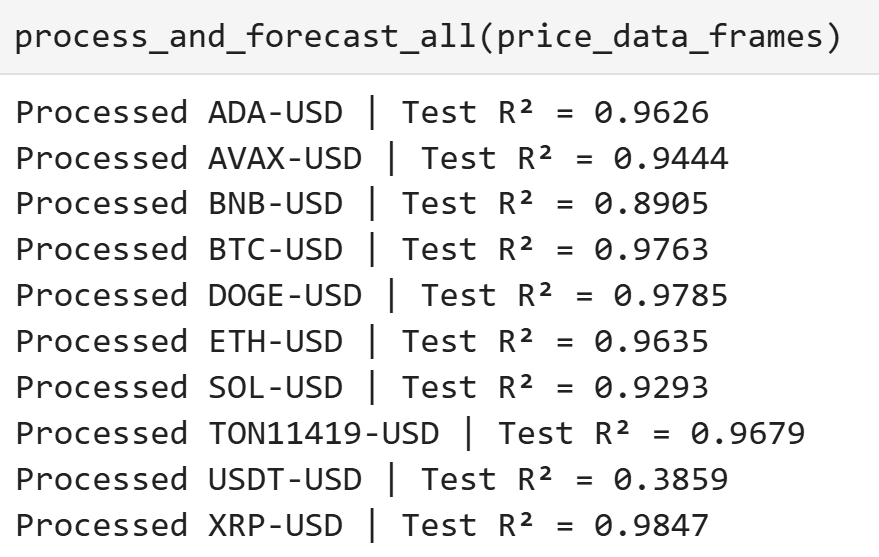
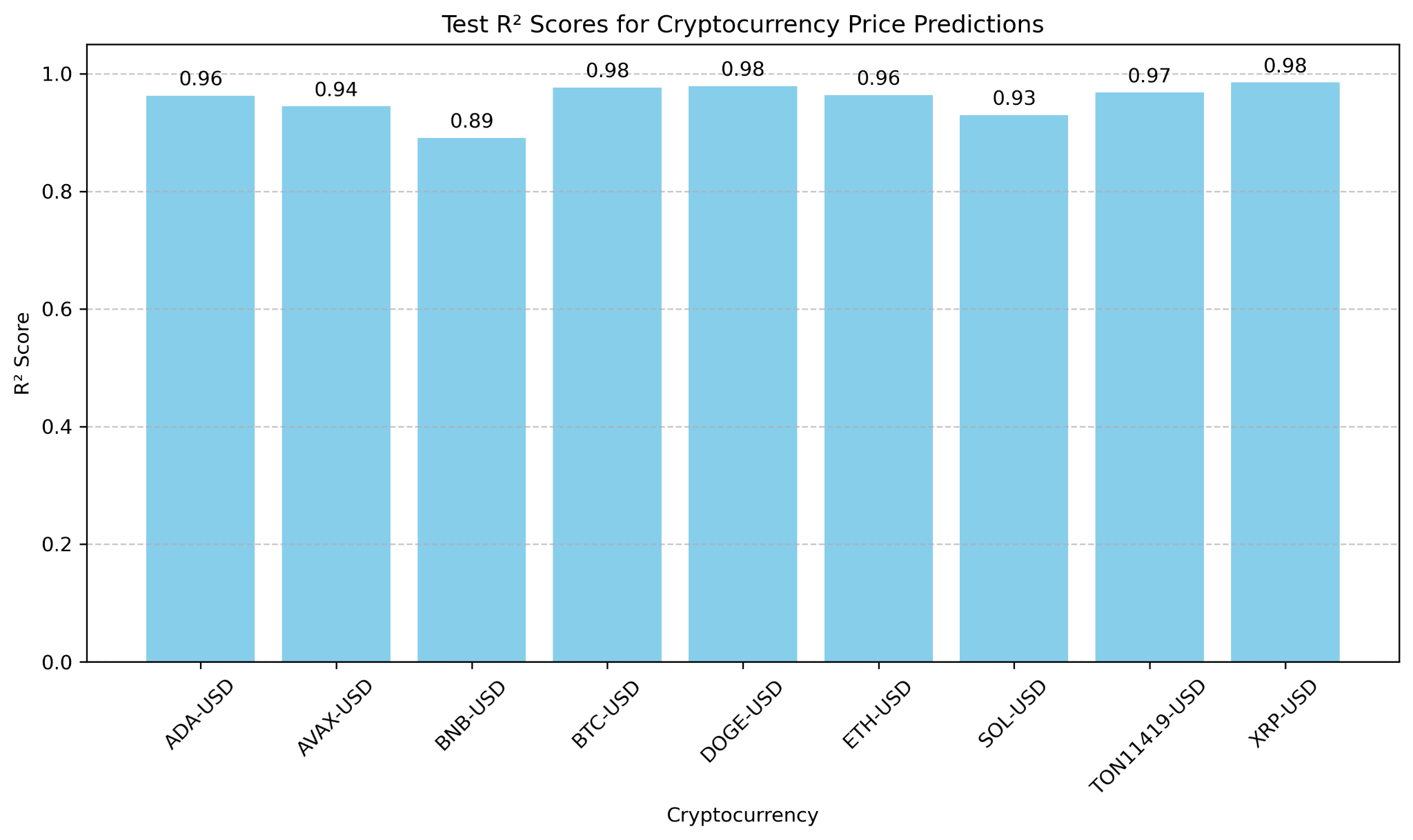
First, I try to predict next-day cryptocurrency prices based on technical indicators, including Simple Moving Averages (SMA) and Exponential Moving Averages (EMA) over 5-day and 20-day windows; Bollinger Bands; and 20-day high and low values. After computing these technical indicators, the next step is sliding window forecast. In this project, we set the sliding window size to 14 days. We loop through the data frame in windows of 14 days, use the first 13 days to train the model, and make a prediction for the last day. The input matrix is the technical indicators we calculated, and the target is the next-day closing price. Our goal is to train the model to predict the closing price of the next day based on various technical indicators calculated on the previous day. We test this forecasting method on different assets, record the actual and predicted prices, then use the actual and predicted prices to calculate R^2 to measure performance of each model.

The following graph shows Bitcoin prices from May 2024 to May 2025 and the technical indicators.

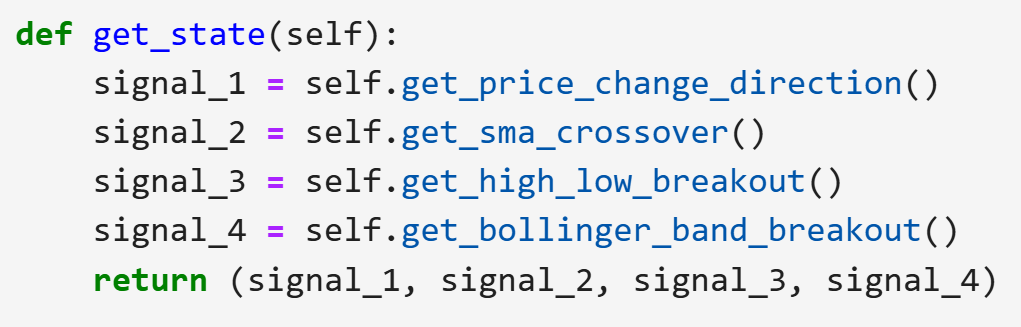


The following graph shows the actual and predicted prices for Bitcoin, with a R^2 score of 0.97.



The model performed very well on most Bitcoins, achieving an R^2 score of 0.97, indicating that the model captured most of the price variance effectively. We later found out that it worked relatively well for other assets as well.

Next , we want to test if this predicted next-day price is actually helpful in real-life trading: We want to use this predicted price as part of an actual trading strategy.

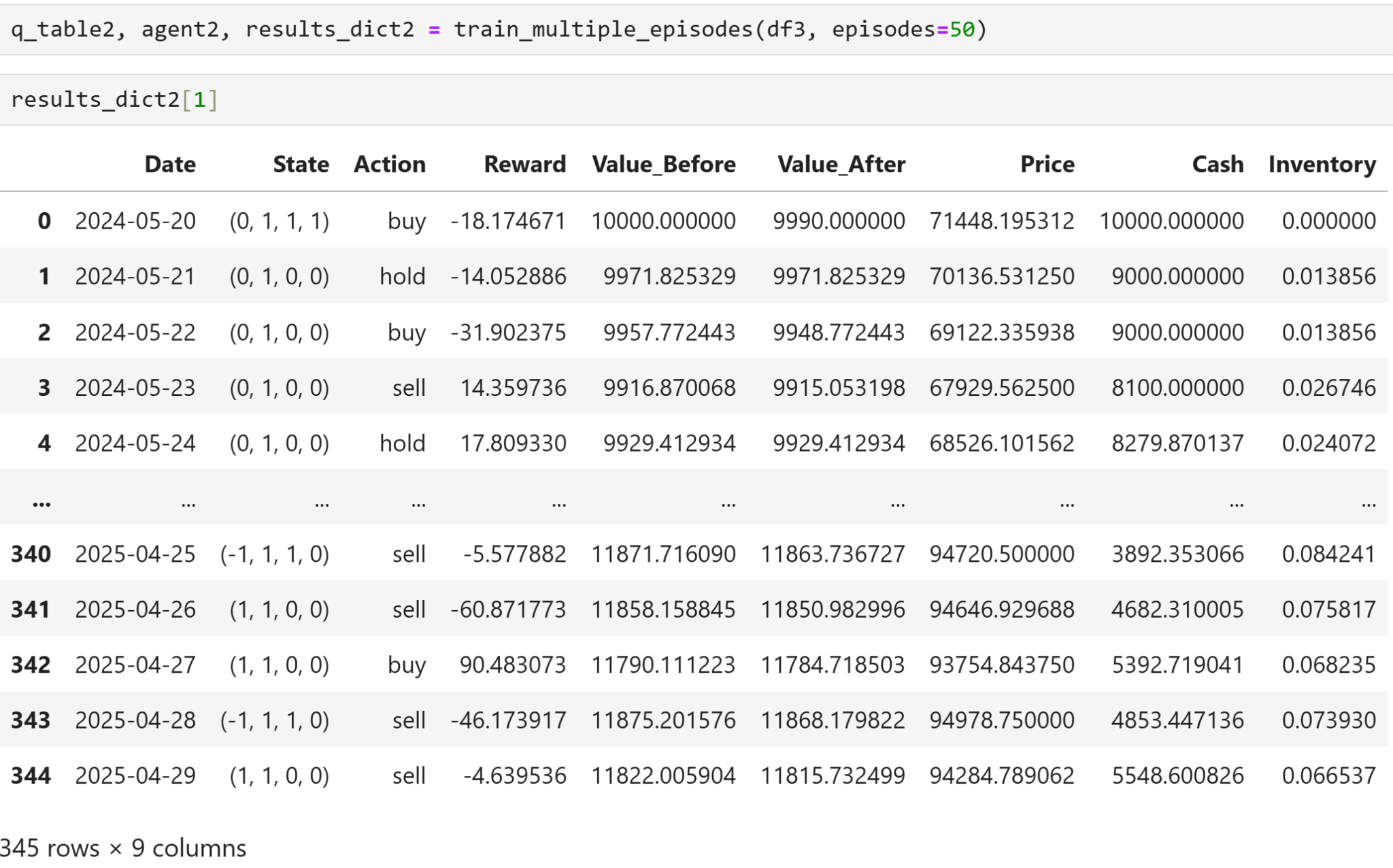


We start by defining the rules of our trading game: We have an initial cash balance of $10,000 and no assets. On each day, we decide whether to “buy”, “hold”, or “sell”. Since the price of Bitcoins can be up to $100,000, we define “buy” as using 10% of our current cash balance to buy whatever amount of asset we can afford; “hold” means that we do nothing, and “sell” means that we sell 10% of all asset we currently own; each transaction has a transaction cost of 1%. We simulate our trading environment using a class called “Portfolio”, which keeps track of the current stock price, current cash balance, and current stock position. This class also calculate the current portfolio value (cash + asset \* price) and current state. We define “state” using 4 signals: (1) price change direction: +1 when the predicted price is greater than the current price, -1 when the predicted price is less than the current price; (2) SMA crossover: +1 when the short term SMA is greater than the long term SMA (bullish trend), -1 when the short term SMA is lower than the long term SMA (bearish trend); (3) High-low breakout: +1 when the price is equal to the 20-day high, -1 when the price is equal to the 20-day low, 0 otherwise; (4) Bollinger band breakout: +1 when the current price is higher than the top bollinger band; -1 when the current price is lower than the bottom band. Each signal has three values: 0, 1, and -1, so there are 3\*\*4 = 81 possible states in total. We use this action space (“buy”, “hold”, “sell”) and signal-states to construct a Q-table, which tells us the expected reward for each action at any given state. Initially, the Q-table is empty, so we select actions at random. We calculate the reward from the difference between the portfolio value after we take an action and the portfolio value on the next day (before we take any action), and we use this reward to update the Q-table (via a class called “Agent”, which stands for the Q-Learning Agent). We use the SARSA on-policy update rule:

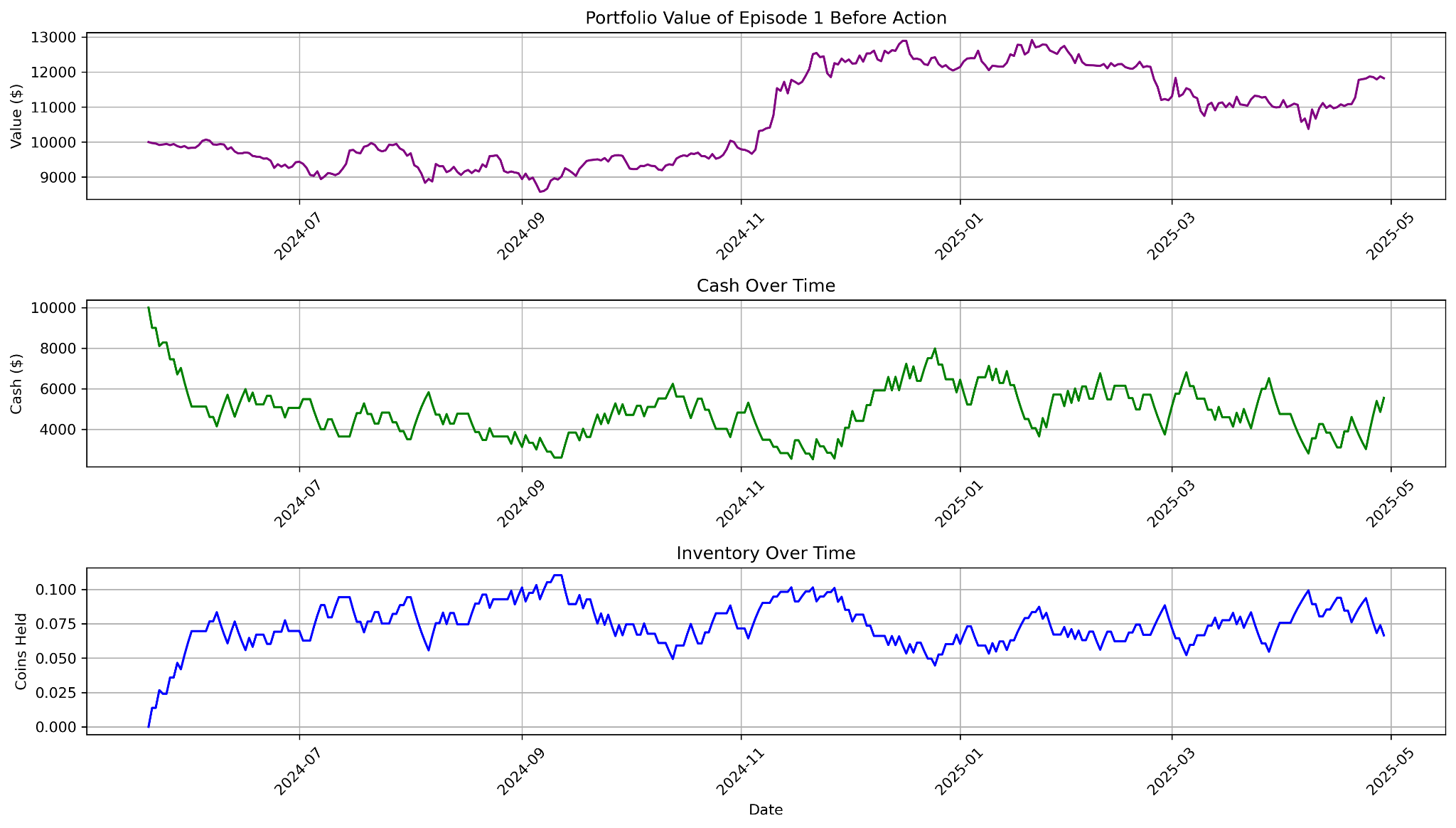
Q(state, action) = Q(state, action) + α[reward + γ·Q(next\_state, next\_actions) - Q(state, action)]

SARSA is an on-policy update method that updates the Q-table based on the next action. The Agent also chooses actions using an ε-greedy strategy: it randomly selects actions with ε probability; otherwise, it decides the next action based on the current state and the reward (Q-table).

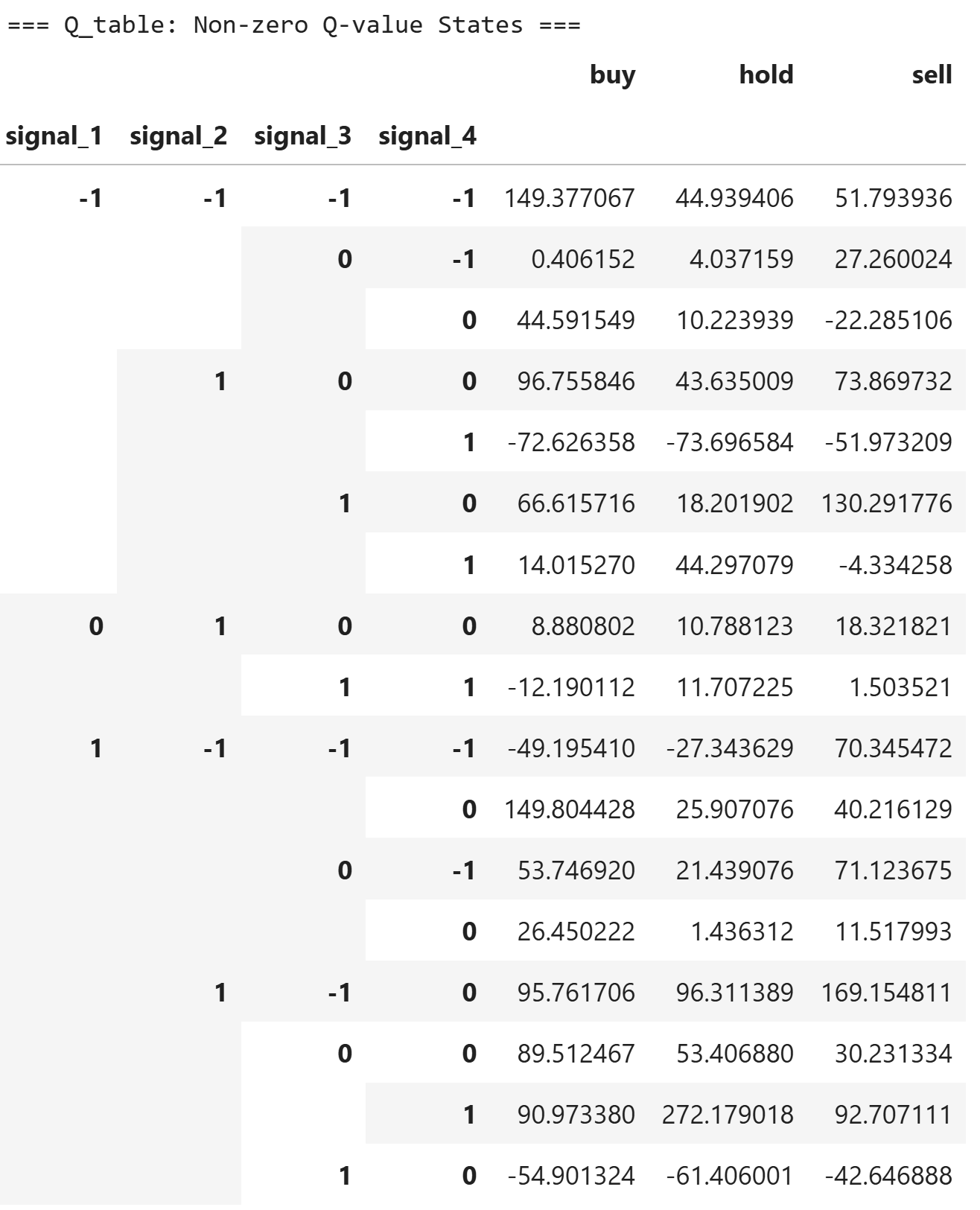
We loop over the entire dataset 50 times (50 training episodes), at each step, we record the current date, state, action, reward, portfolio value before and after the action, as well as current cryptocurrency price, cash, and inventory. The result table for a single training episode looks like this.



We can visualize the actions and results of our trading strategy by plotting the portfolio value, cash, and asset over time. Here is what is happening during the first training episode.



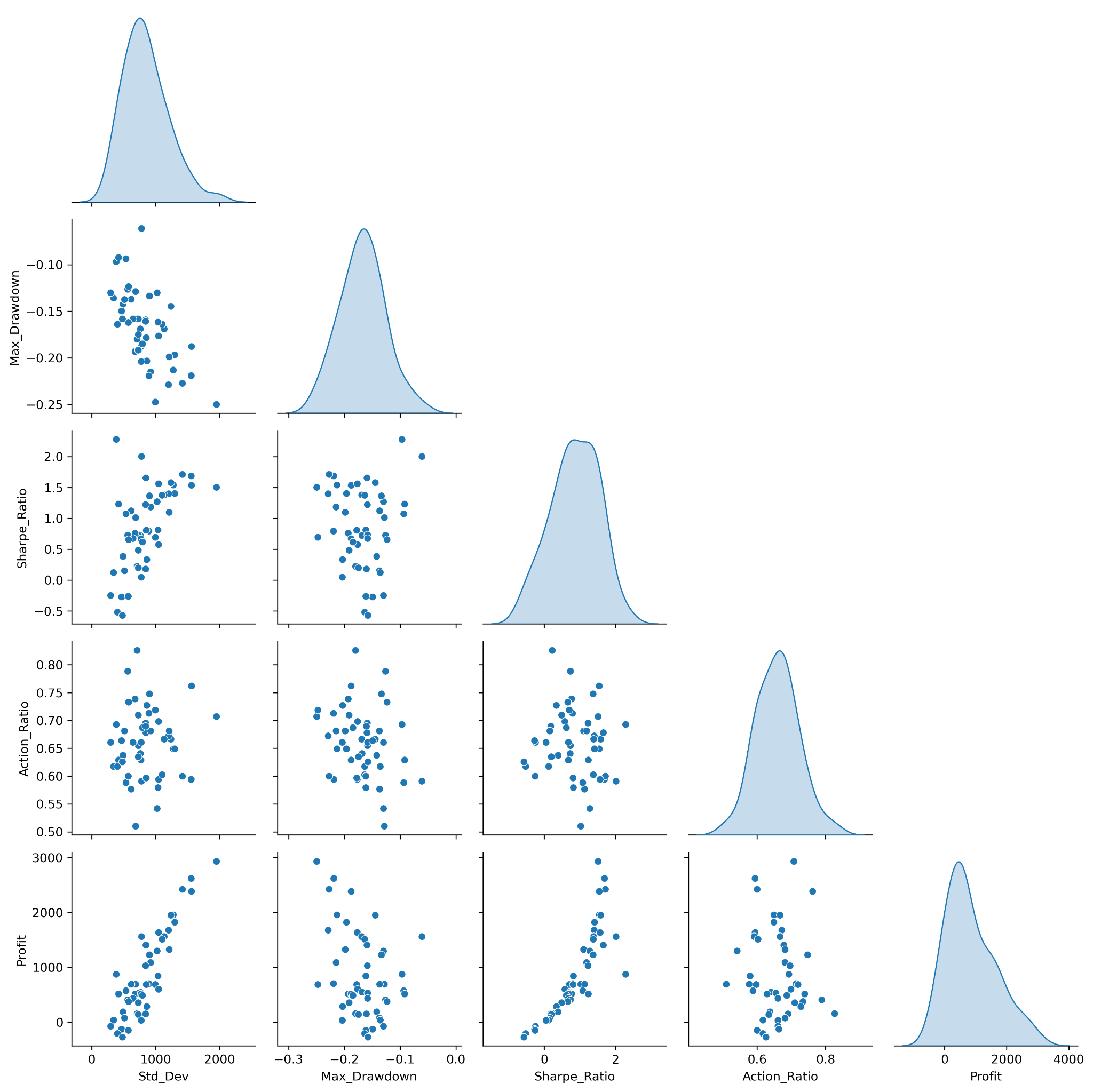
We also want to visualize the non-zero rows of the trained Q-Table to make sure that the Q-table is indeed being updated during the training process.



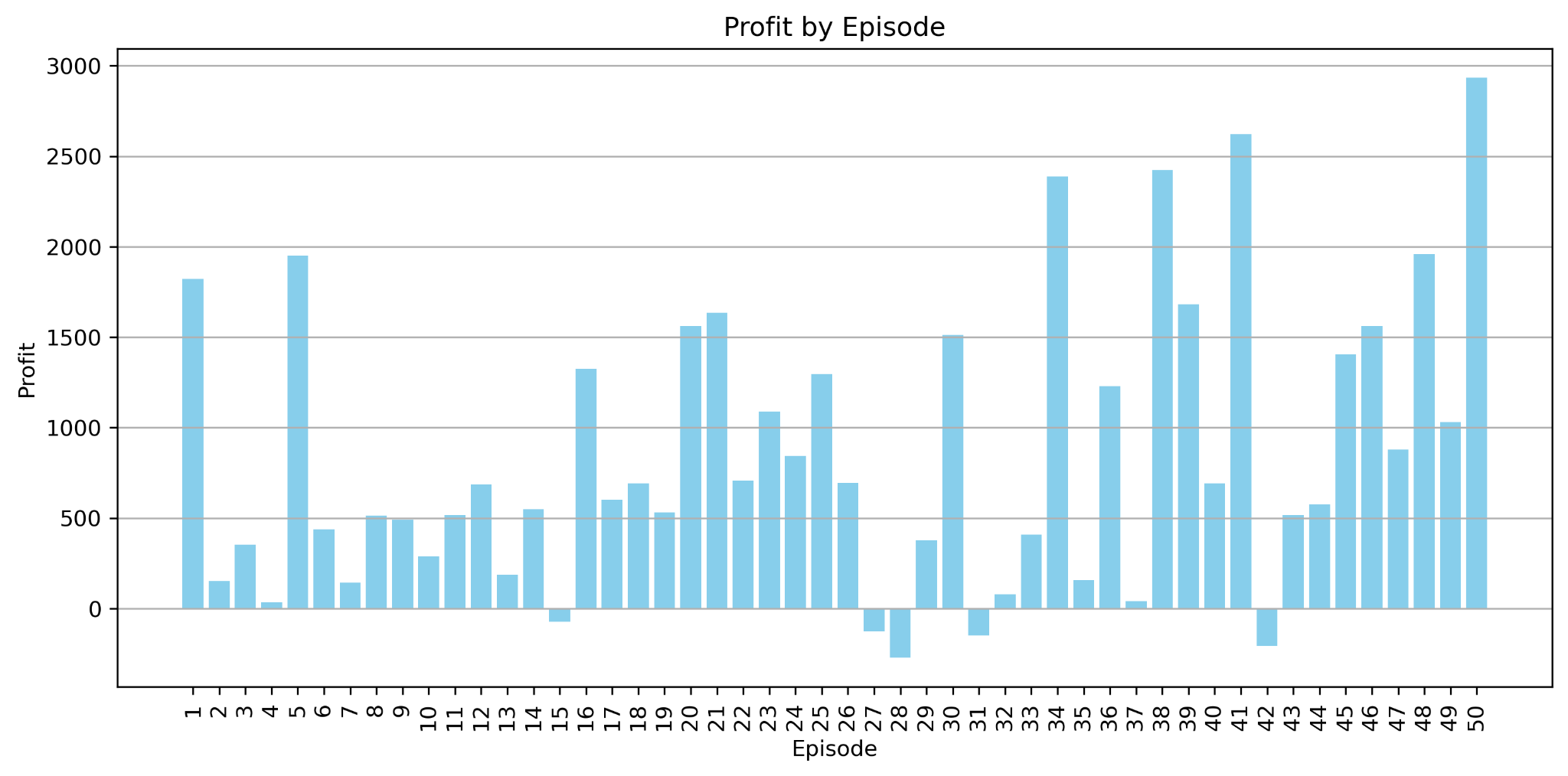
We computed some basic statistics for each training episode: including standard deviation of portfolio value (“Std\_Dev”), Maximum drawdown of portfolio value (“Max\_Drawdown”), Profit calculated from the difference between initial and final portfolio (“Profit”), and Sharpe Ratio calculated from “Profit” divided by “Std\_Dev” (“Sharpe\_Ratio”)



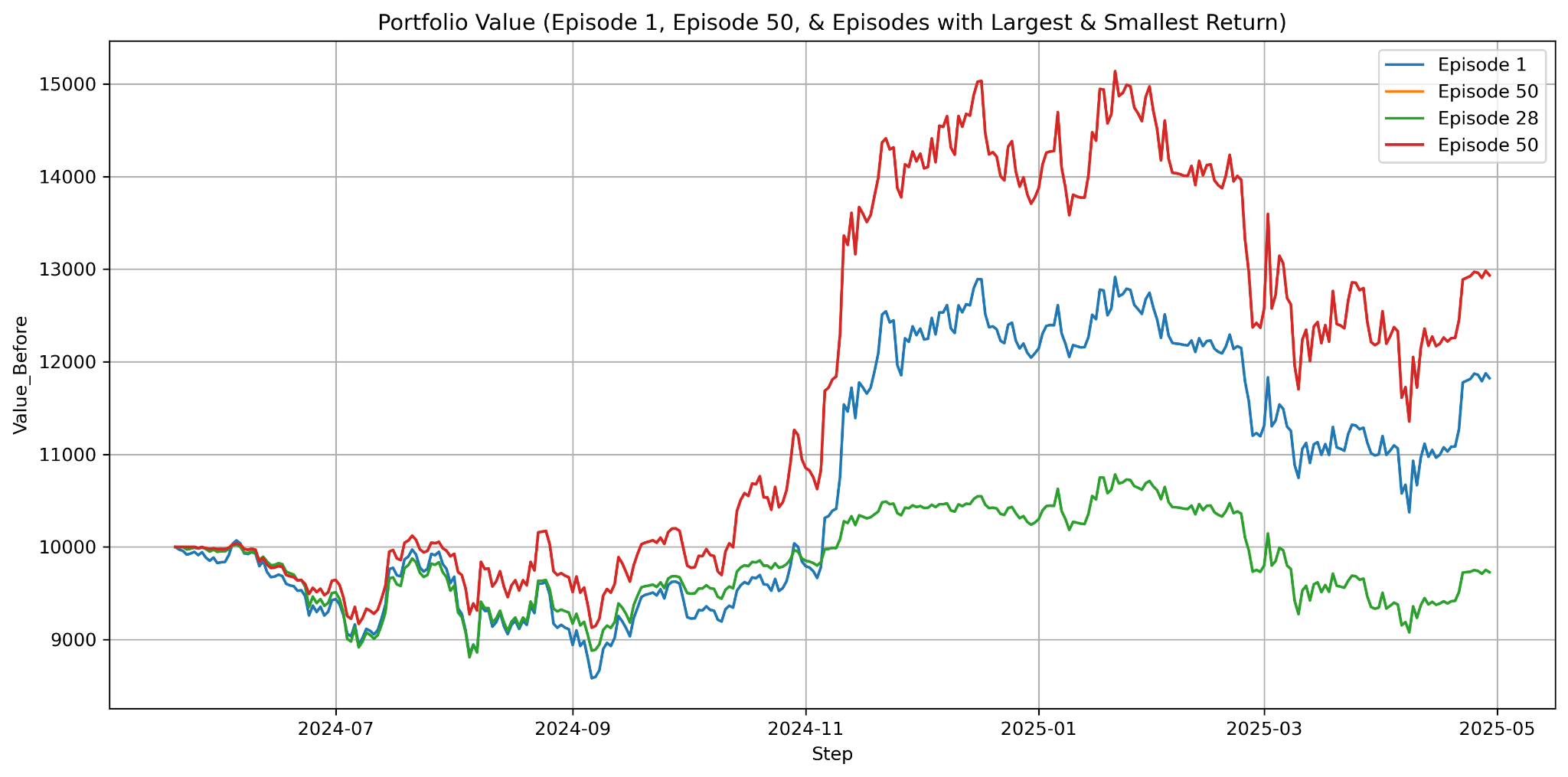
Now, we can visualize the pairwise relationship between these metrics using pairwise scatterplots. We observe that “Std\_Dev” has positive correlation with “Profit” and “Sharpe\_Ratio”, this seems to imply that high return is associated with higher risk.



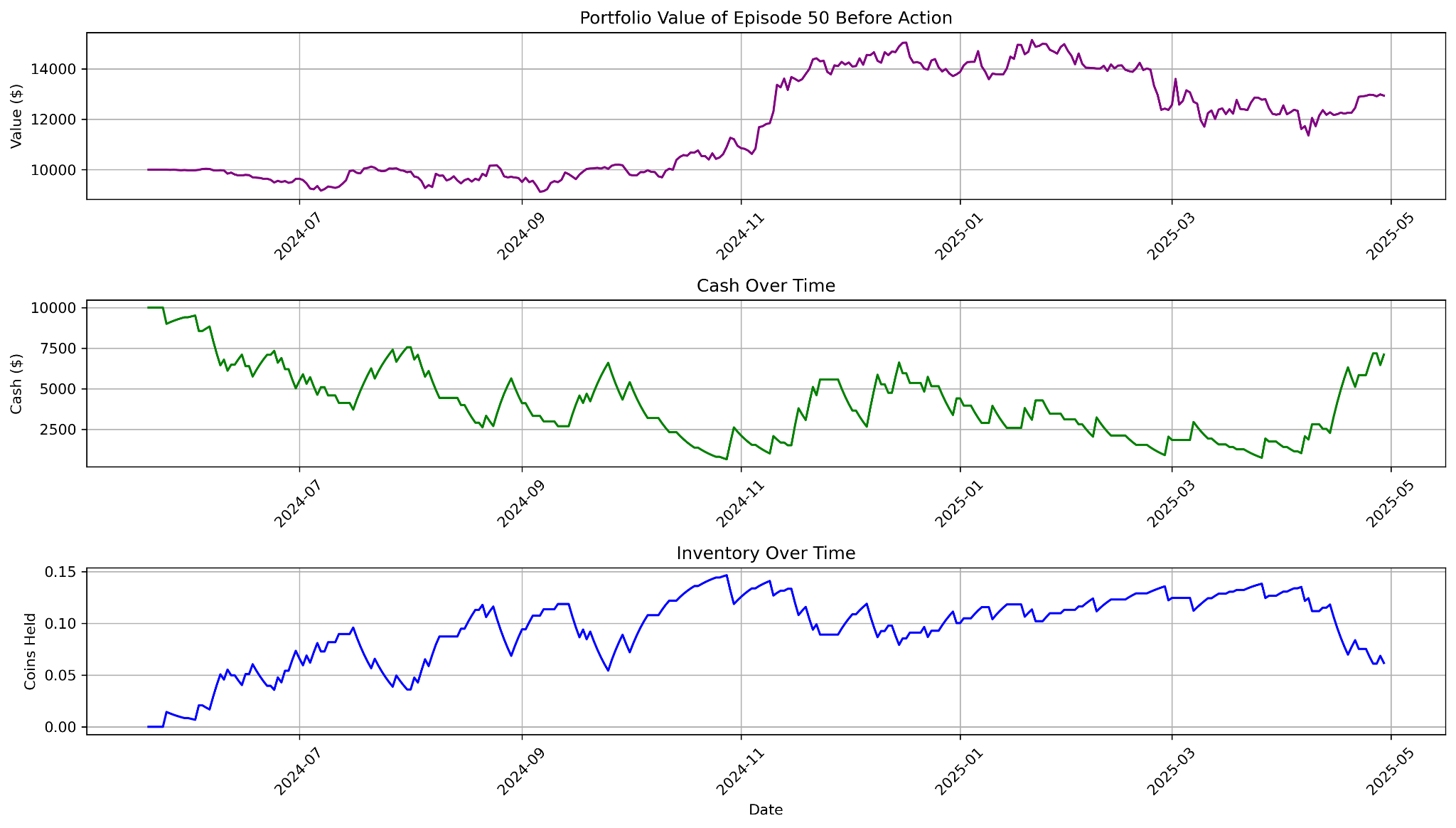
Our learning agent dictates that we start with complete randomness (epsilon = 1.0), our actions are selected at random and use the observed result to update the Q-table. The decay rate is 0.95, so by the end of the 50 training episodes, the probability that each action is chosen at random instead of based on the Q-table is about 8%. Ideally, the Profit should stabilize as the episode number increases, but we observed that this is not always the case when we plot the profit for each episode.



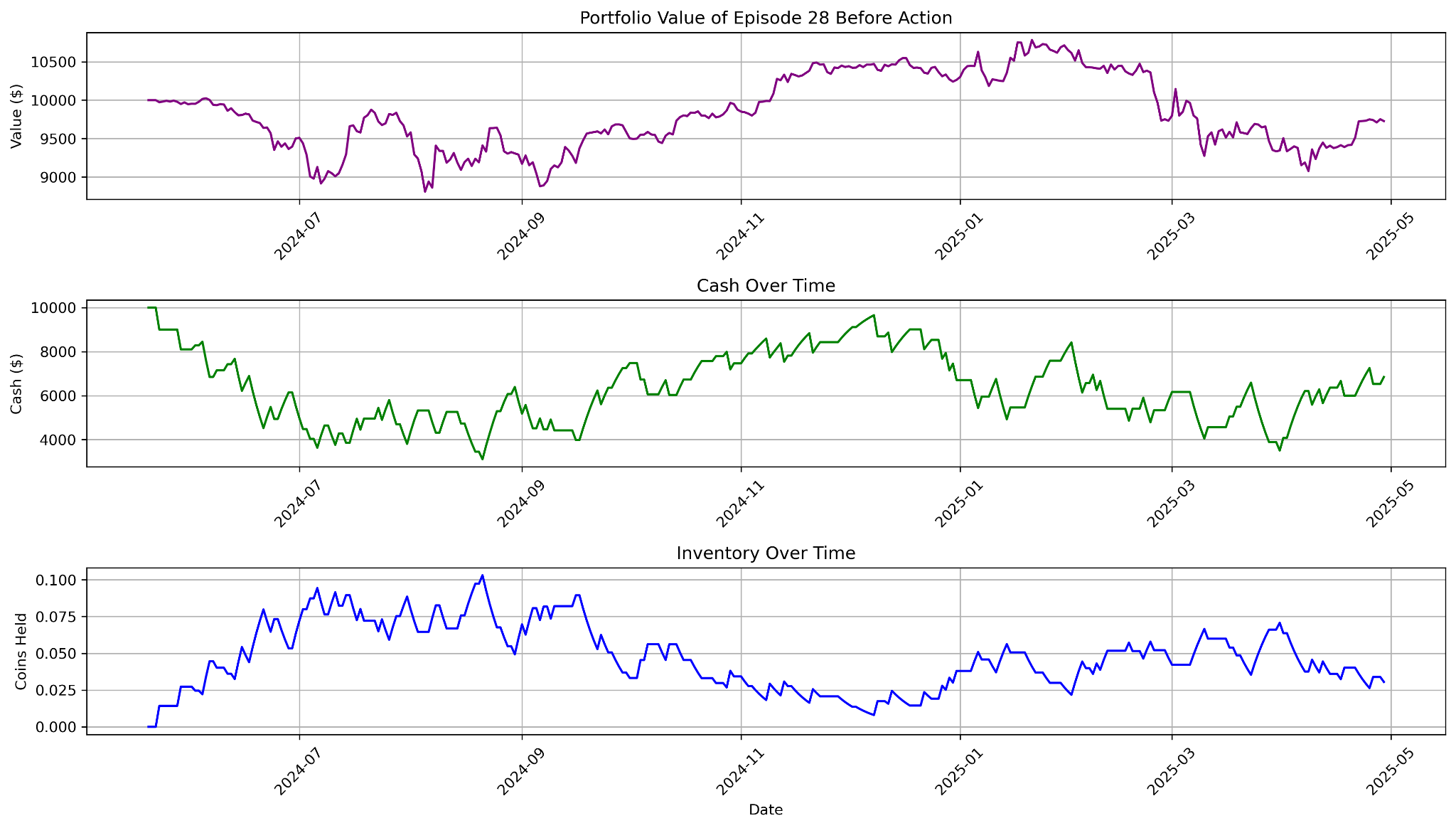
By comparing the portfolio over time for the portfolios with highest and lowest returns, we can observe that the portfolio with highest return also has the highest volatility , and the episode with the lowest return also has lower volatility.



The following graph visualizes the change in stock position, cash balance, and portfolio value for the episode that yielded the highest return.

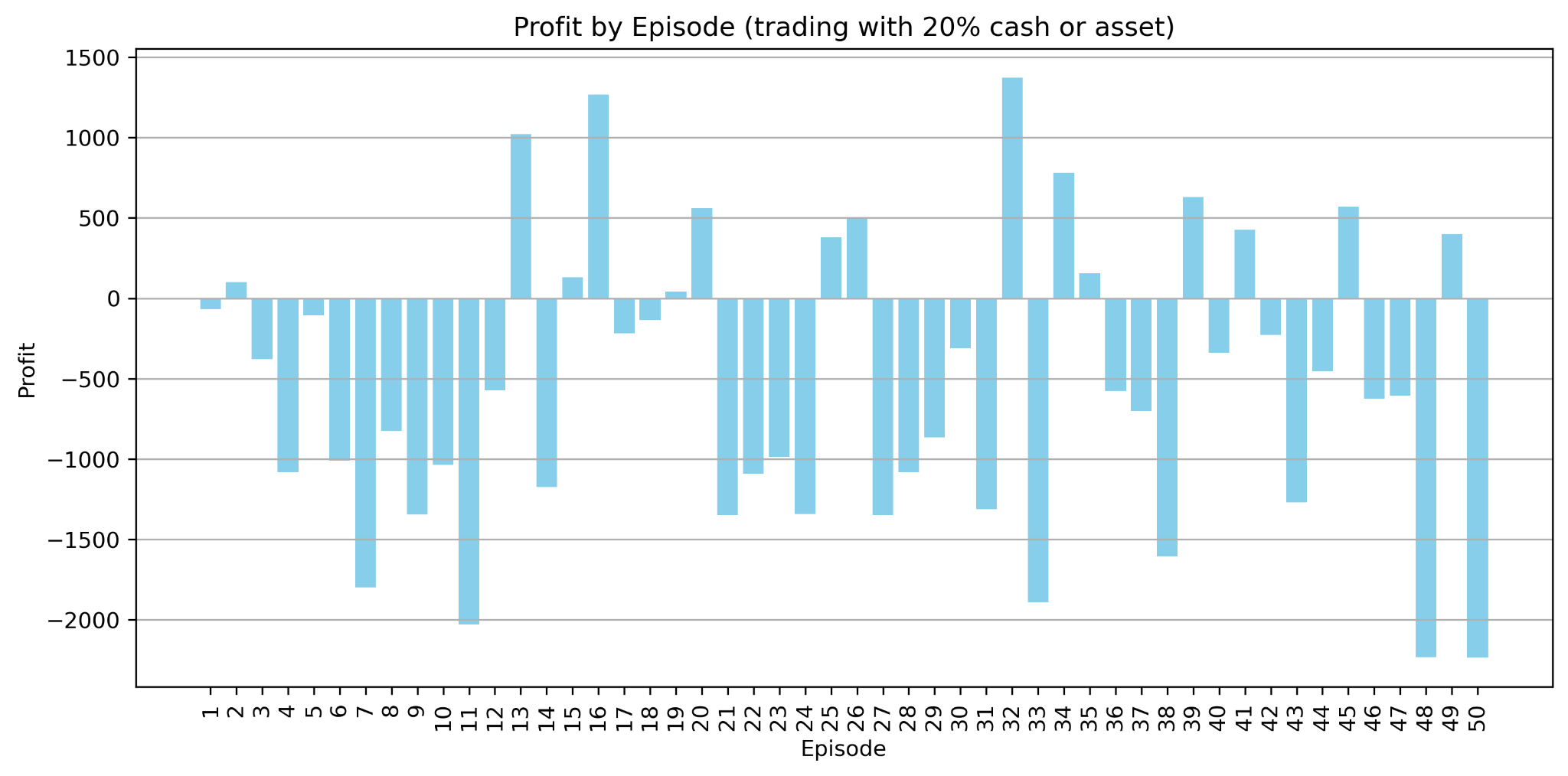


The following graph visualizes the change in stock position, cash balance, and portfolio value for the episode that resulted in the lowest return.



In general, if we want lower risk (lower standard deviation over time), we should sell cryptocurrencies and convert them to cash. If we want higher return (higher standard deviation), we should buy cryptocurrencies and decrease cash balance.

Next, we try buying with 20% cash or selling 20% assets at each transaction.



We observe that the overall profit over the entire period becomes more volatile as we increase the trading volume. Losses occur more often.

## **Part 4: Model Results**

We test the model by applying the same steps on different assets:

1. Compute the technical indicators (SMA, EMA, Bollinger Bands) based on price of each asset;
2. Make predictions based on the technical indicators, and store the “price\_change\_direction” (+1 or -1) in the same data frame as the technical indicators and price — this forms the input data frame, which we store as CSV files in a folder called “input”. There should be a CSV file for each asset.
3. Loop through the “input” folder, load the csv files into data frames. Divide each input data frame into 80% training set and 20% test set.
4. Train a Q-table using the function “train\_multiple\_episodes”, complete 50 training episodes for each training set.
5. Test the training result using the function “test\_with\_greedy\_policy”, which would take the test set as input, choose actions entirely based on the trained Q-table (chose the action that has the greatest reward given the current state), and store the result table as a csv file in the “training\_result” folder.
6. Each csv file in the “training\_result” folder should contain the dates, chosen action on each date, portfolio value before and after the action, price of the asset, cash, and inventory. We plot them and analyze the result.

After computing some basic statistics for each of the training results, we noticed that most of the assets did not result in positive return (“Profit”) during this period under this trading strategy. This is likely because most of the assets experience a negative price change during this period. We observe that BTC resulted in the highest profit and ETH resulted in the lowest profit.

## **Part 5: Conclusion & Recommendations**

Our results show that the Q-learning framework is capable of learning meaningful trading policies through repeated interactions with a simulated environment. The agent successfully updated its Q-table based on daily portfolio returns and demonstrated improved decision-making across episodes. Nevertheless, the profitability of the strategy varied significantly depending on the asset, the market trend, and the time frame. Assets with declining prices generally produced negative profits. We tested larger trade volume in hope for greater profit, but they only led to higher volatility and more frequent losses.

Overall, this project confirms that reinforcement learning can be a powerful tool in financial trading. However, it also reveals the limitations of applying a fixed-state, discrete-action Q-learning model in a highly dynamic, noisy market like cryptocurrency. The agent's performance is heavily dependent on the quality of the input signals, the stability of market trends, and the design of the reward structure.

## **Part 7: Future Work**

Several directions can be explored to improve the performance, robustness, and generalizability of this approach:

1. Experiment with different signal-states, try to figure out which signals can affect profit the most.
2. Update the Q-table during the testing period: Update the Q-table dynamically during the test period, try to assign different weights to rewards during the testing period.
3. Try to build the Q-table only on episodes with the most overall profit.
4. Try different combinations of assets instead of one asset at a time.