Report on Kalman Filter applied on Neural Network for Finance

August 13, 2025

Mathys VINATIER - GitHub Project page
Supervisor:
Pr Kim Tae-Wan

Mathys VINATIER

Contents

Chapter 10

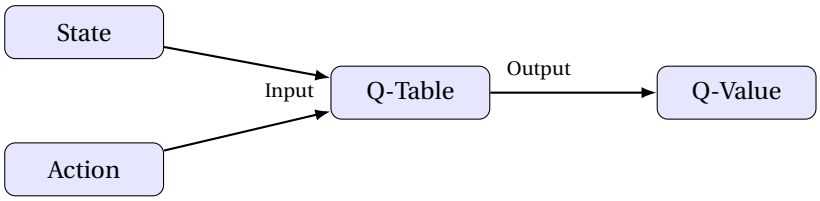
Week 10

Contents

10.1 Reminder on Greedy Policy and QLearning

Q-Learning is a **model-free, off-policy, value-based reinforcement learning algorithm** that trains an action-value function (Q-function) to find the optimal policy indirectly. The Q-function estimates the expected cumulative reward of taking a certain action in a given state and following the best policy thereafter.

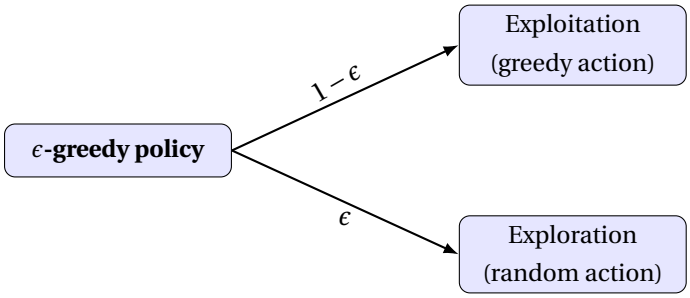
- **Q-table:** Internally, Q-Learning uses a Q-table storing values for each state-action pair. Initially, all values are zero, and the table is updated iteratively during training.



- **Epsilon-Greedy Strategy:** To balance exploration and exploitation, actions are chosen using an epsilon-greedy policy:

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

  - With probability $1-\epsilon$, exploit by selecting the action with the highest Q-value.
  - With probability ϵ, explore by selecting a random action.
  - ϵ decays over time to shift from exploration to exploitation.



- **TD Update:** At each step, the Q-value for the current state-action pair $Q(s_t, a_t)$ is updated based on the immediate reward r_{t+1} plus the discounted maximum Q-value of the next state s_{t+1}:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t) \right]$$

- **Off-policy Learning:** The policy used to select actions (epsilon-greedy) differs from the policy used to compute the target update (greedy). This distinction allows Q-Learning to learn optimal policies even when exploring randomly.

After enough training episodes, the Q-table converges to the optimal Q-function, which directly yields the optimal policy by choosing actions with the highest Q-values in each state.

10.2 Implementation Progress and Experiments

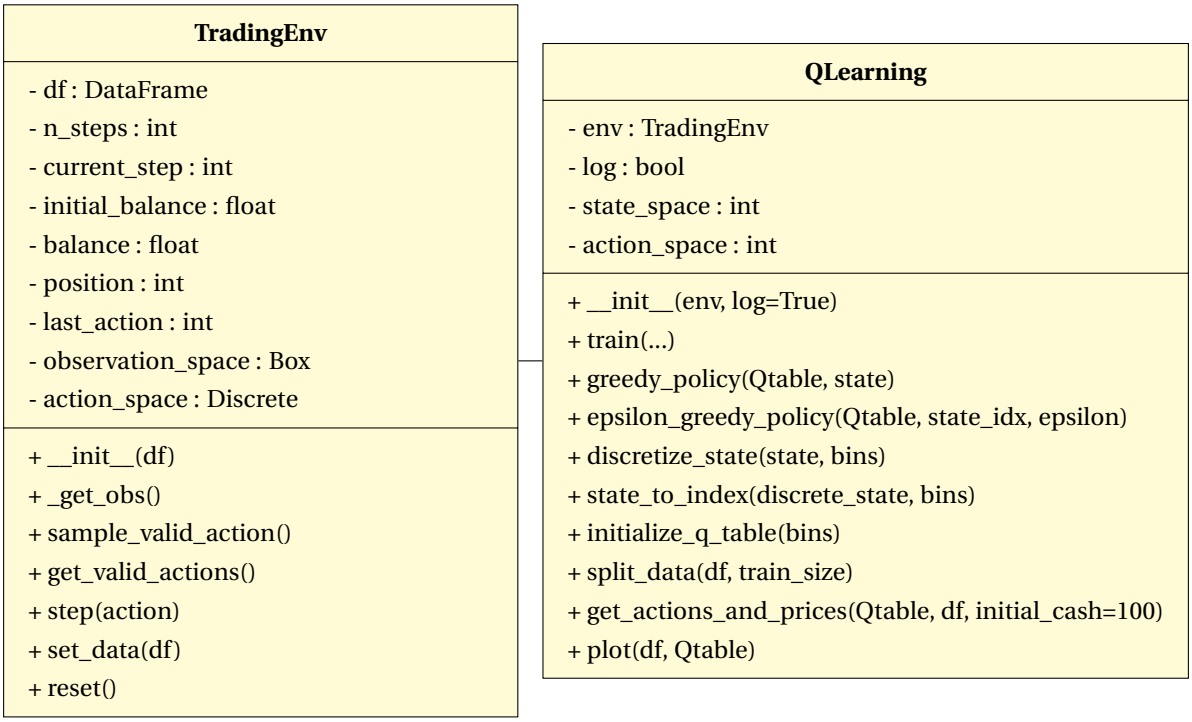
This week was dedicated to the initial implementation and experimentation phase of the project, focusing primarily on the Proximal Policy Optimization (PPO) basis algorithms. Using the Hugging Face classes, we developed a prototype with greedy policy agent and integrated it into a custom trading environment inspired by the concepts discussed in previous weeks.

10.2.1 Environment Setup

The custom environment was designed to simulate stock market trading dynamics with the following key elements :

- **State representation** including historical price data, technical indicators (moving averages, RSI) and Kalman filtered signals.
- **Action space** consisting of discrete trading actions: Buy, Hold or Sell.
- **Reward structure** based on portfolio value change and risk-adjusted metrics to incentivize both profit and stability.
- **Episode design** with fixed-length trading periods to allow episodic evaluation of agent performance.

The environment was implemented adhering to OpenAI Gym interfaces, allowing smooth integration with standard RL training pipelines. Here is the UML of the created environement :



Algorithm 1: Trading Environment Behavior

Input: Environment env, number of episodes N, exploration rate ϵ

Output: Episode value and rewards

for $episode \leftarrow 1$ **to** N
- $s \leftarrow env.reset()$;
- total_reward $\leftarrow 0$;
- **while** *True*
  - Get valid actions $A_{valid} \leftarrow env.get\_valid\_actions()$;
  - Select $a \leftarrow$ random choice from A_{valid};
  - Execute $(s', r, done, info) \leftarrow env.step(a)$;
  - total_reward $\leftarrow$ total_reward $+ r$;
  - Log $(episode, s, a, r, info["portfolio\_value"])$;
  - $s \leftarrow s'$;
  - **if** $done$ **then**
    - break
- Print episode summary : total_reward, final portfolio value;

Algorithm 2: Q-Learning Training

Input: Environment env, data frame df, training size $train\_size$, episode N

Output: Trained Q-table Q

Split data: $df_{train} \leftarrow train\_size$ of df;

Calculate bins for discretization : $bins \leftarrow$ compute bins from df_{train};

Initialize Q-table

for $episode \leftarrow 1$ **to** N
- Exploration rate : $\epsilon \leftarrow \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) \cdot \exp(-decay\_rate \times episode)$;
- Reset environment : $state_{cont} \leftarrow env.reset()$;
- Discretize initial state : $state_{disc} \leftarrow discretize(state_{cont}, bins)$;
- Convert to index : $state_{idx} \leftarrow state\_to\_index(state_{disc}, bins)$;
- **for** $step \leftarrow 1$ **to** max_steps
  - Choose action a using epsilon-greedy policy :

$$a \leftarrow \begin{cases} \arg\max_{a'} Q[state_{idx}, a'] & \text{with probability } 1-\epsilon \\ \text{random valid action} & \text{with probability } \epsilon \end{cases}$$

  - Take action : $(next_{state_{cont}}, r, done, info) \leftarrow env.step(a)$;
  - Discretize next state : $next_{state_{disc}} \leftarrow discretize(next_{state_{cont}}, bins)$;
  - Convert to index : $next_{state_{idx}} \leftarrow state\_to\_index(next_{state_{disc}}, bins)$;
  - Update Q-value :$Q[state_{idx}, a] \leftarrow Q[state_{idx}, a] + \alpha\left(r + \gamma \max_{a'} Q[next_{state_{idx}}, a'] - Q[state_{idx}, a]\right)$;
  - $state_{idx} \leftarrow next_{state_{idx}}$;
  - **if** $done$ **then**
    - **break**;

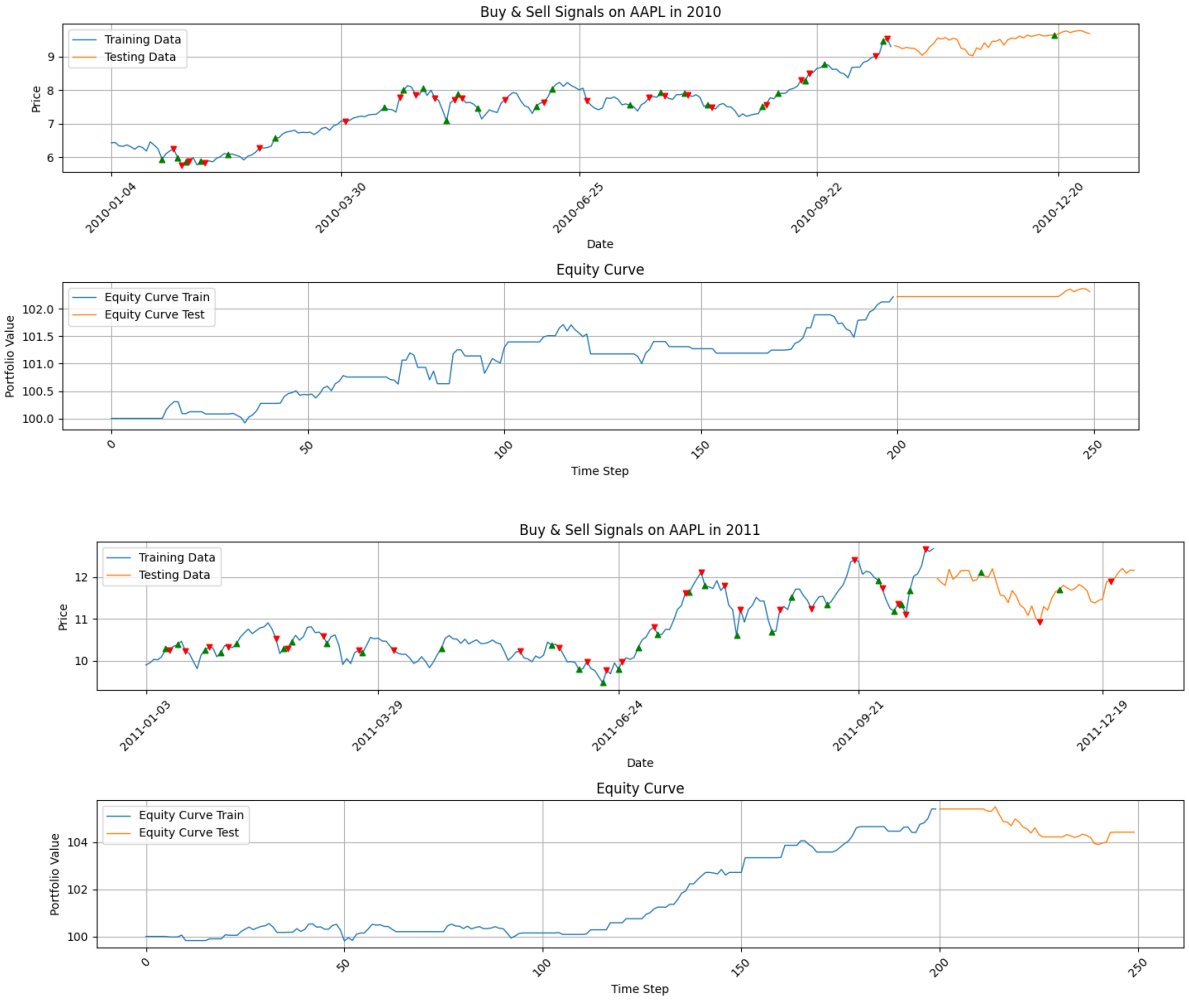
return Q

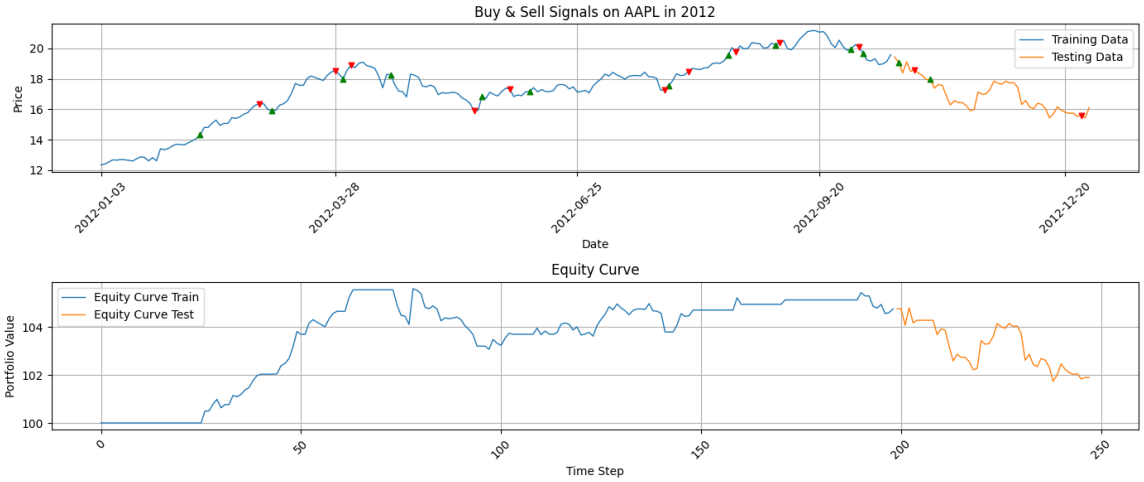
In the context of Q-learning for trading environments, bins are used to discretize the continuous state space into a finite number of discrete states. This discretization is essential because the Q-table requires a finite and manageable number of states to store and update the expected rewards for each state-action pair. Continuous variables, such as stock prices or technical indicators, have infinite possible values, which makes it impractical to represent them directly in a Q-table. To address this, we divide the range of each continuous feature into a fixed number of intervals called bins. Each bin represents a segment of the feature's value range, allowing us to map any continuous observation into a discrete category. For implementation, we analyze the training dataset and create evenly spaced bins (10 bins) for each feature by computing linearly spaced intervals between the minimum and maximum values of that feature. Then, during training or evaluation, the observed continuous state is converted into a discrete state by determining which bin each feature value falls into. This discretized state is subsequently converted into a unique index to access and update the Q-table. The use of bins thus enables effective Q-learning in continuous state environments by simplifying the state representation while preserving essential information.

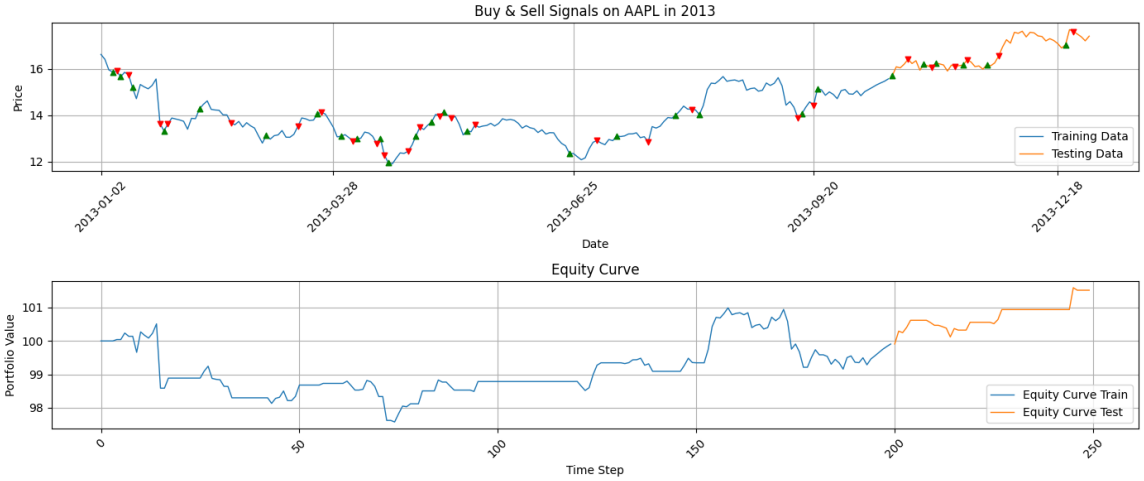
10.2.2 Preliminary Results for Q-Learning Greedy Policy

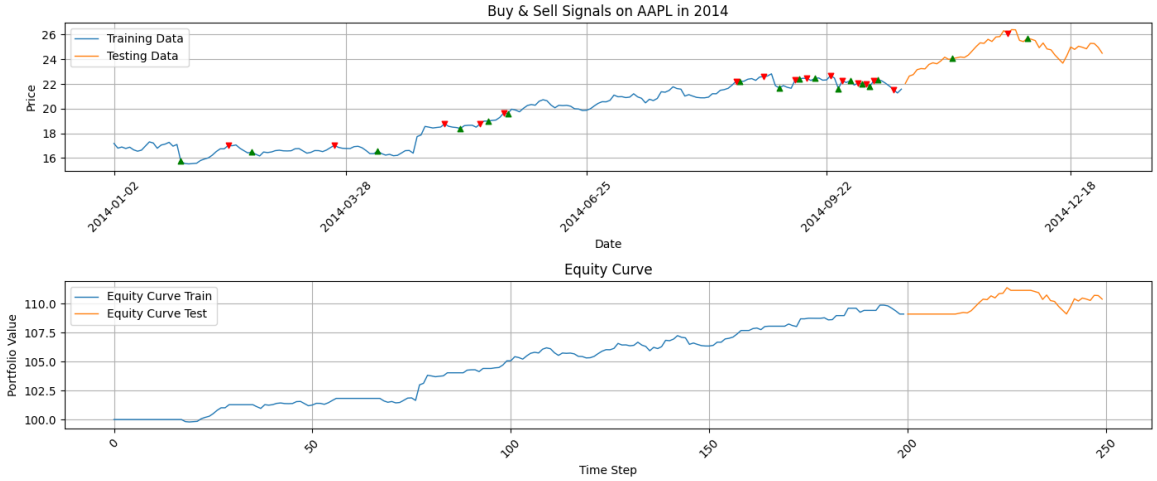
Initial experiments aimed to evaluate the training behavior and reward progression of the Q-Learning agent following a greedy policy. Results were compared over two time horizons : a shorter training period of 100 episodes and an extended training of 1000 episodes.

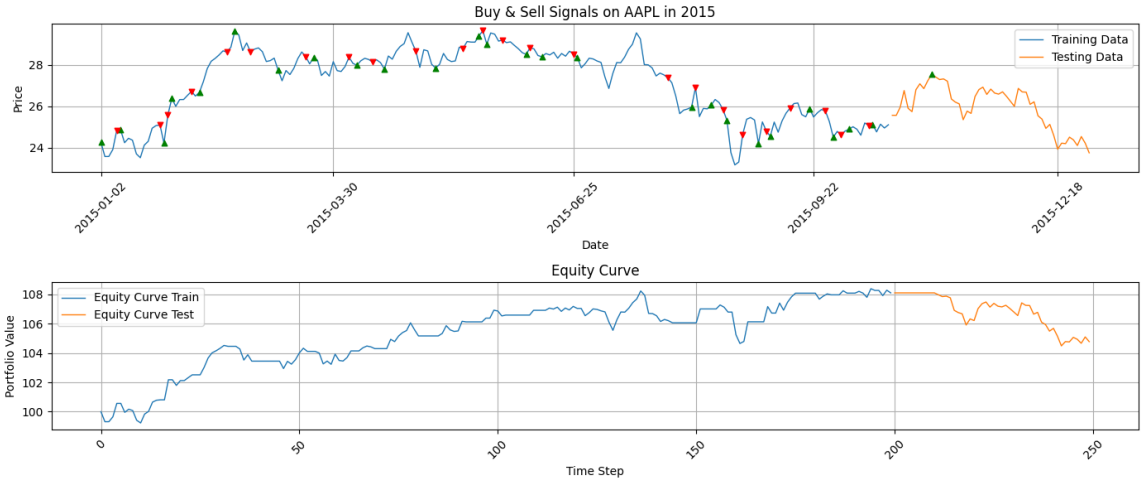
100 episodes

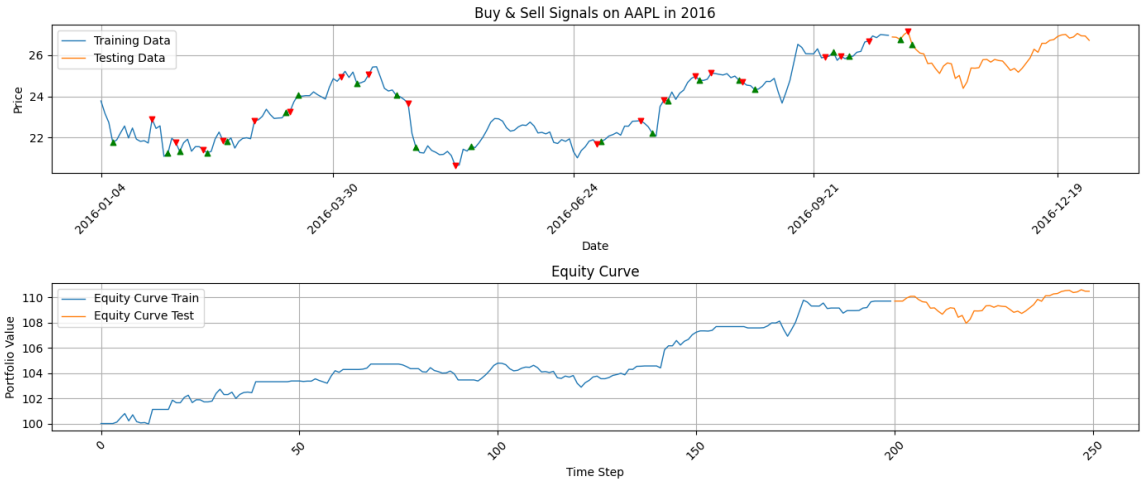


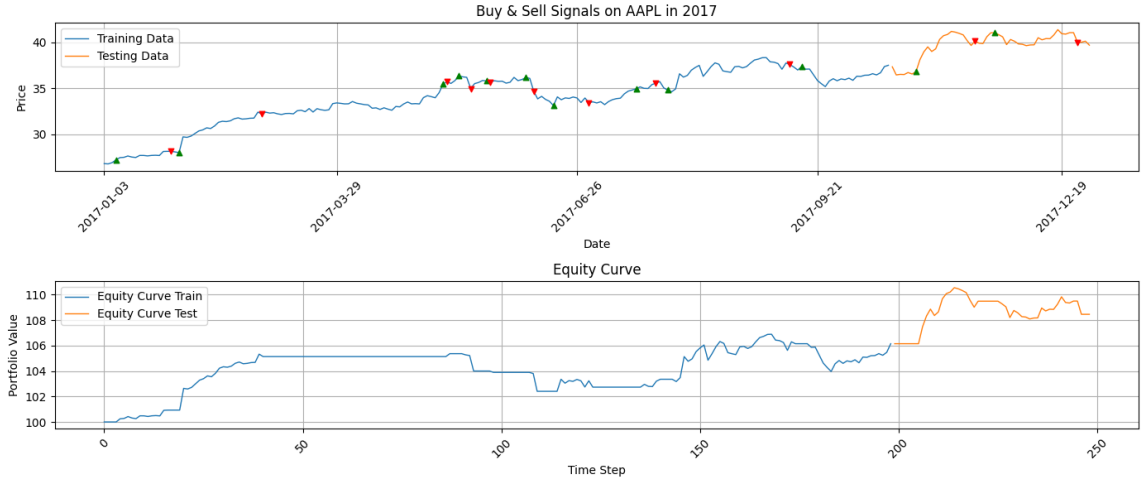

Buy & Sell Signals on AAPL in 2012
Training Data
Testing Data
Price
Date
2012-01-03
2012-03-28
2012-06-25
2012-09-20
2012-12-20
Equity Curve
Equity Curve Train
Equity Curve Test
Portfolio Value
Time Step

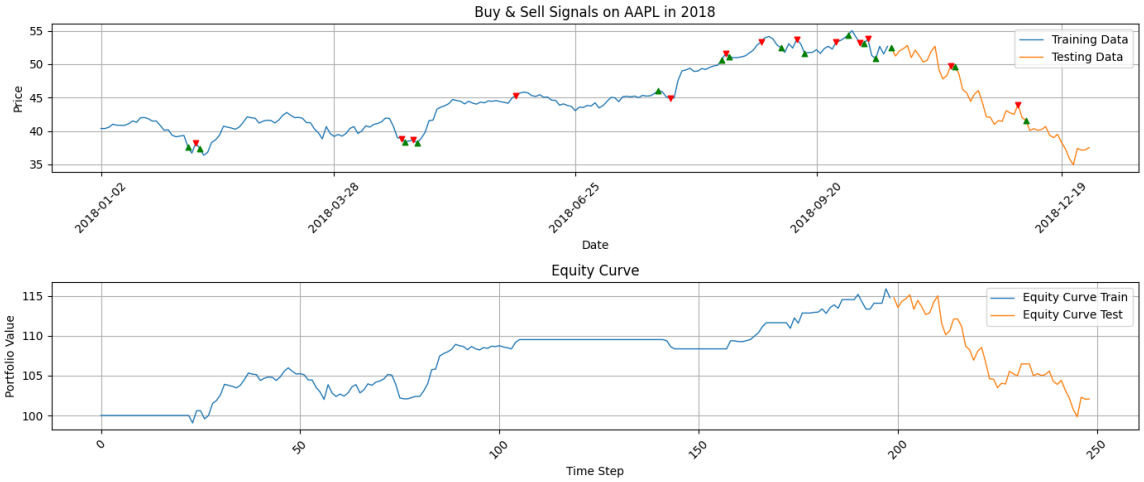

Buy & Sell Signals on AAPL in 2013
Training Data
Testing Data
Price
Date
2013-01-02
2013-03-28
2013-06-25
2013-09-20
2013-12-18
Equity Curve
Equity Curve Train
Equity Curve Test
Portfolio Value
Time Step

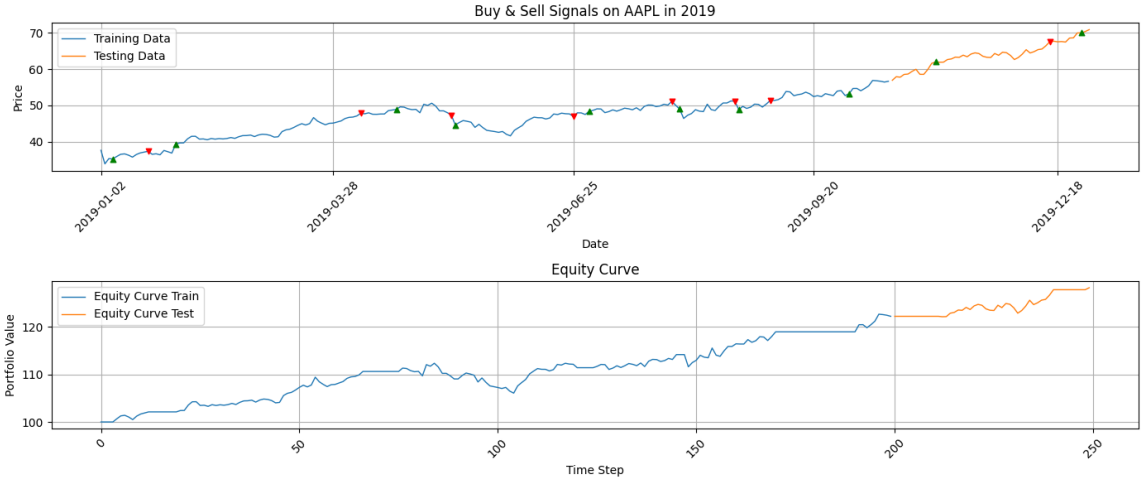

Buy & Sell Signals on AAPL in 2014
Training Data
Testing Data
Price
Date
2014-01-02
2014-03-28
2014-06-25
2014-09-22
2014-12-18
Equity Curve
Equity Curve Train
Equity Curve Test
Portfolio Value
Time Step

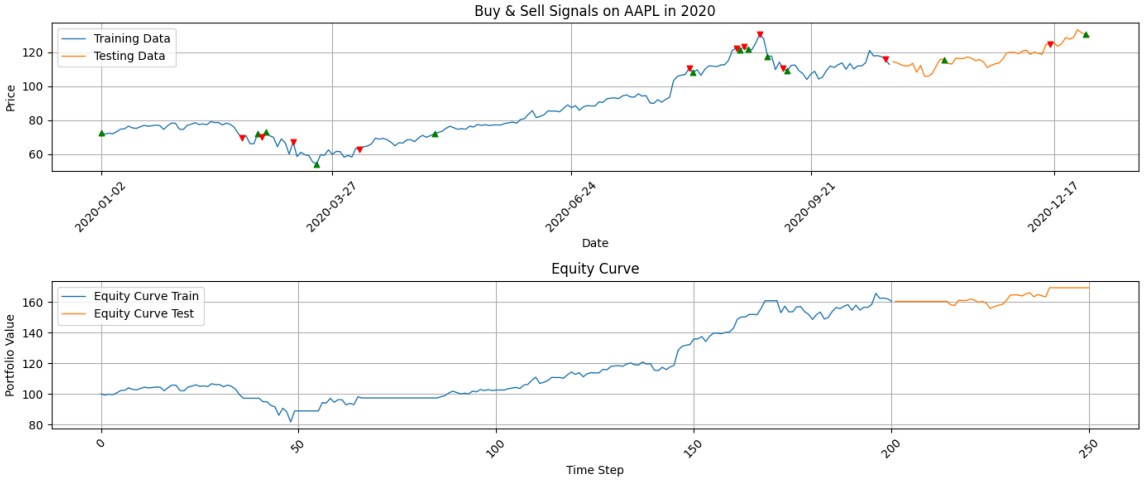

Buy & Sell Signals on AAPL in 2015
Training Data
Testing Data
Price
Date
2015-01-02
2015-03-30
2015-06-25
2015-09-22
2015-12-18
Equity Curve
Equity Curve Train
Equity Curve Test
Portfolio Value
Time Step

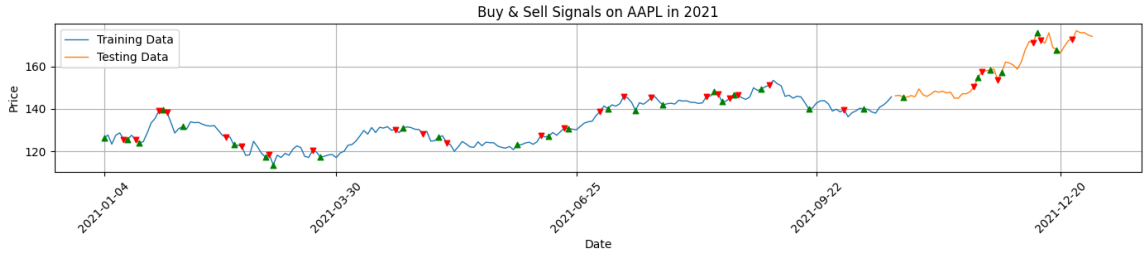

Buy & Sell Signals on AAPL in 2016
Training Data
Testing Data
Price
Date
2016-01-04
2016-03-30
2016-06-24
2016-09-21
2016-12-19
Equity Curve
Equity Curve Train
Equity Curve Test
Portfolio Value
Time Step

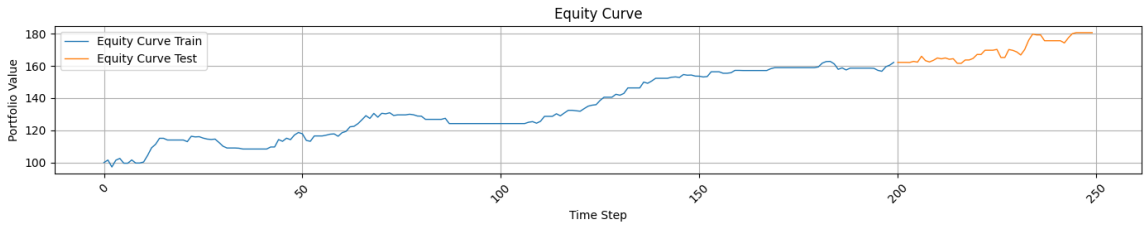

Buy & Sell Signals on AAPL in 2017
Training Data
Testing Data
Price
Date
2017-01-03
2017-03-29
2017-06-26
2017-09-21
2017-12-19
Equity Curve
Equity Curve Train
Equity Curve Test
Portfolio Value
Time Step

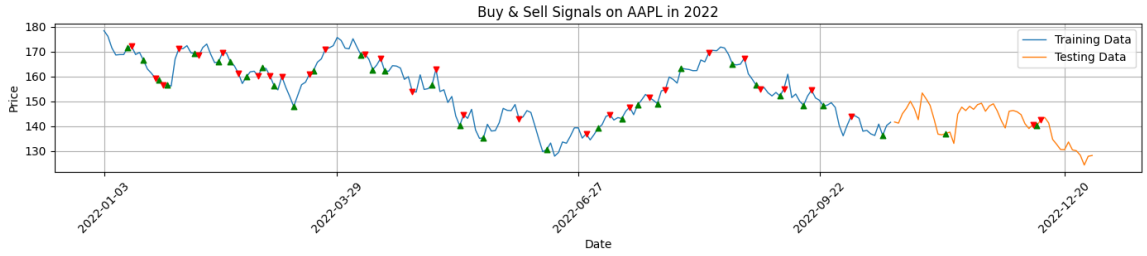

Buy & Sell Signals on AAPL in 2018
Training Data
Testing Data
Price
Date
2018-01-02
2018-03-28
2018-06-25
2018-09-20
2018-12-19
Equity Curve
Equity Curve Train
Equity Curve Test
Portfolio Value
Time Step

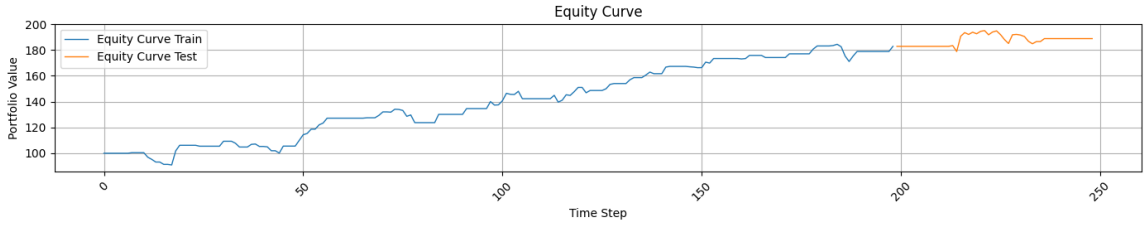

Buy & Sell Signals on AAPL in 2019
Training Data
Testing Data
Price
Date
2019-01-02
2019-03-28
2019-06-25
2019-09-20
2019-12-18
Equity Curve
Equity Curve Train
Equity Curve Test
Portfolio Value
Time Step


Buy & Sell Signals on AAPL in 2020
Training Data
Testing Data
Price
Date
2020-01-02
2020-03-27
2020-06-24
2020-09-21
2020-12-17
Equity Curve
Equity Curve Train
Equity Curve Test
Portfolio Value
Time Step

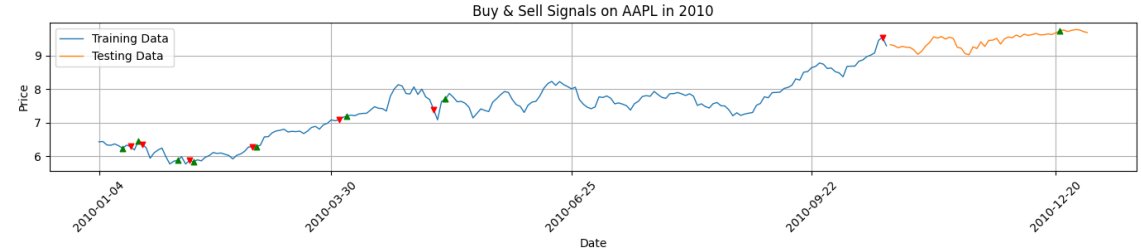


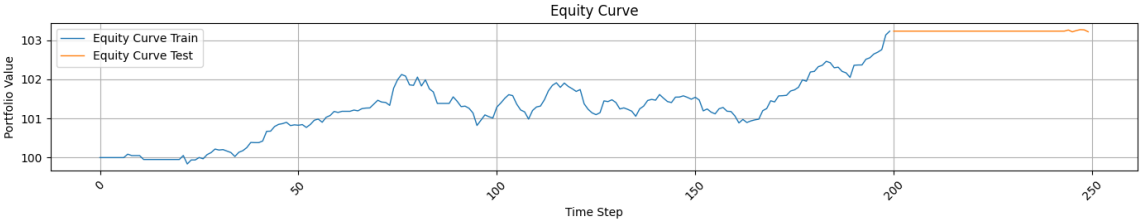


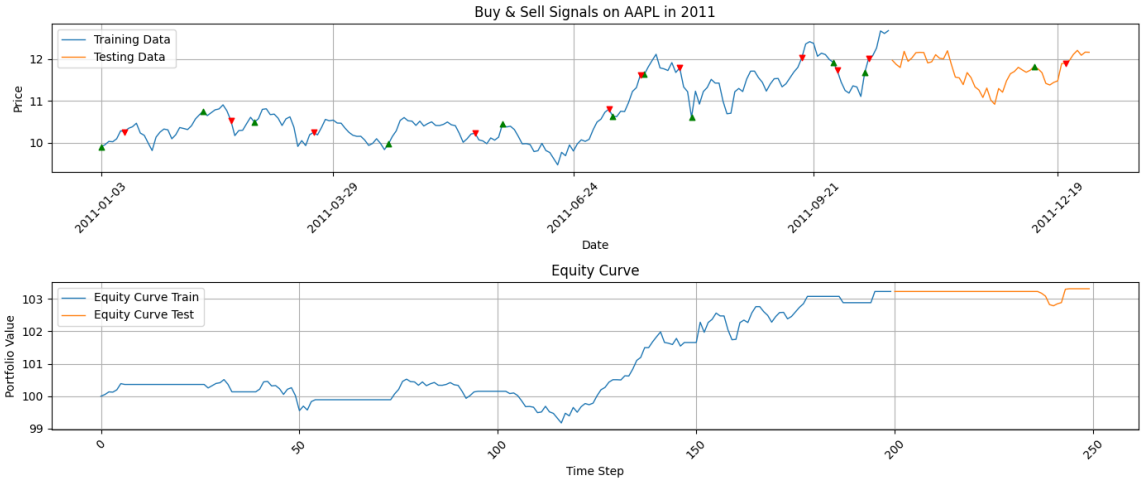


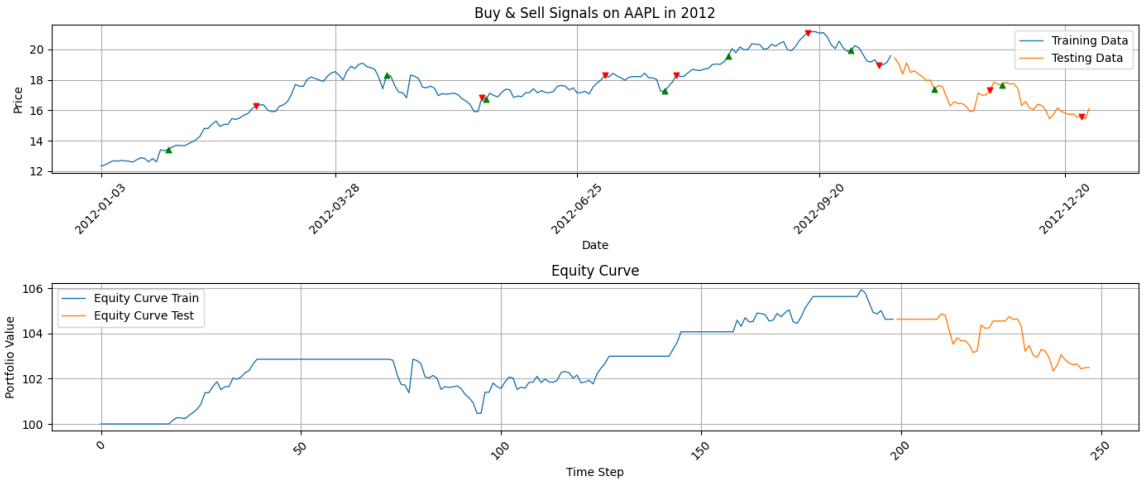


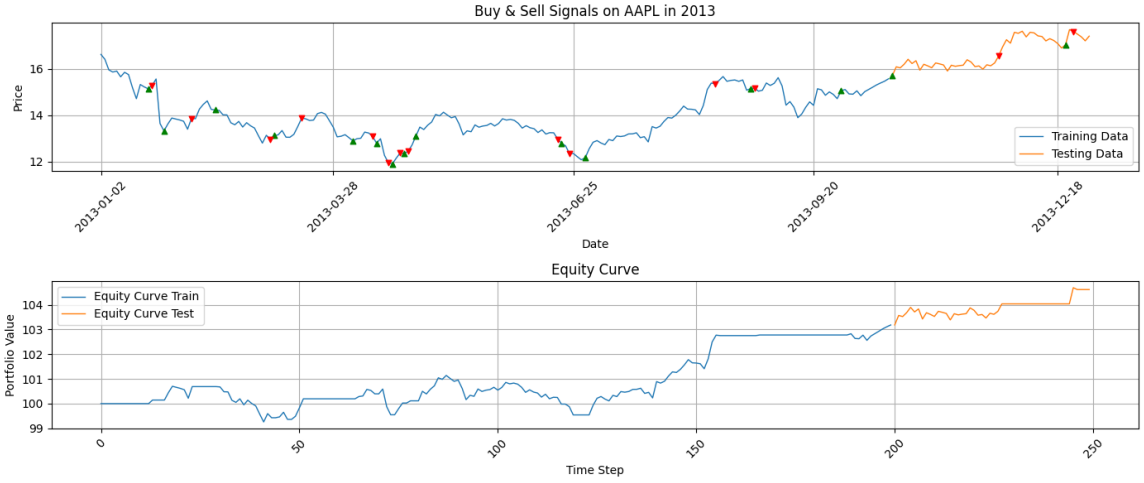
1000 episodes

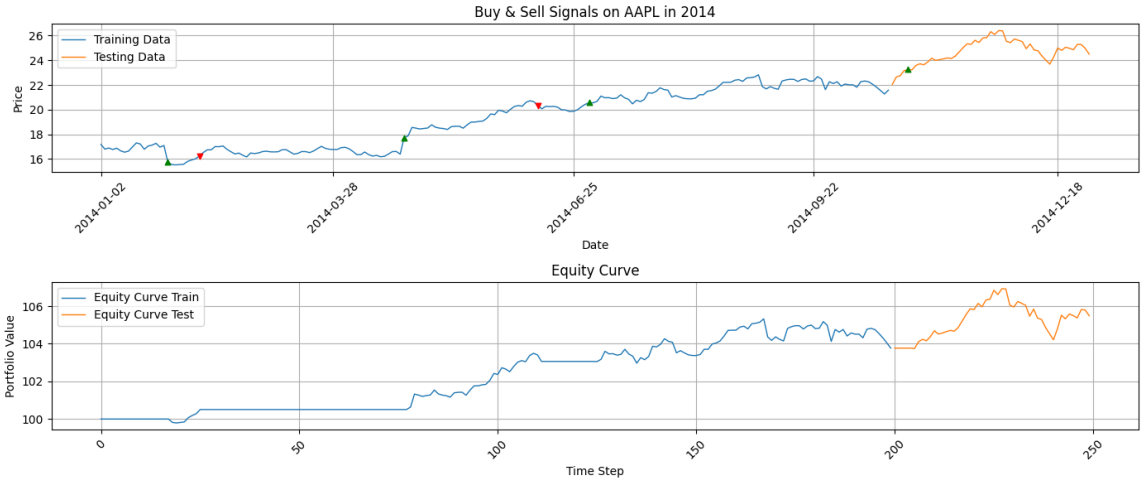


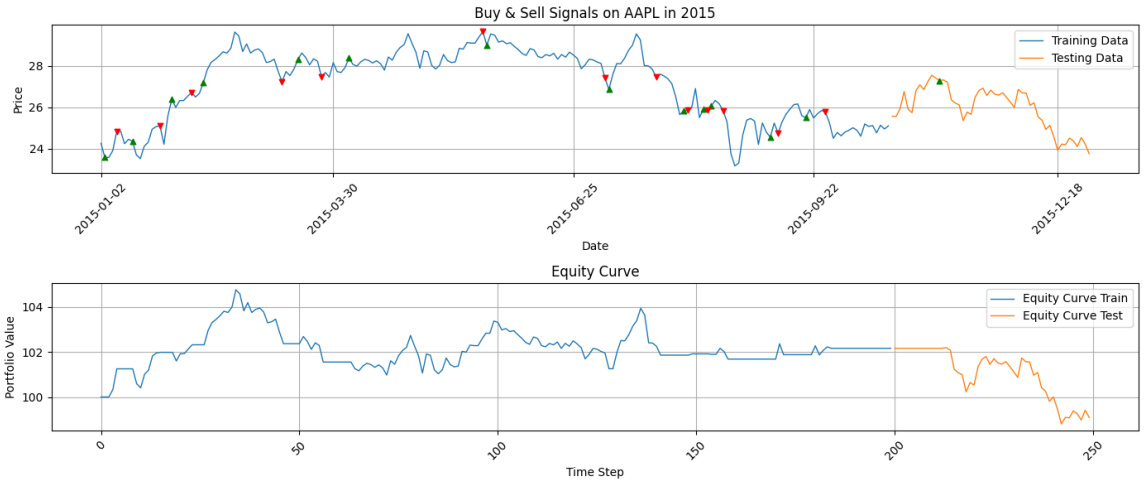


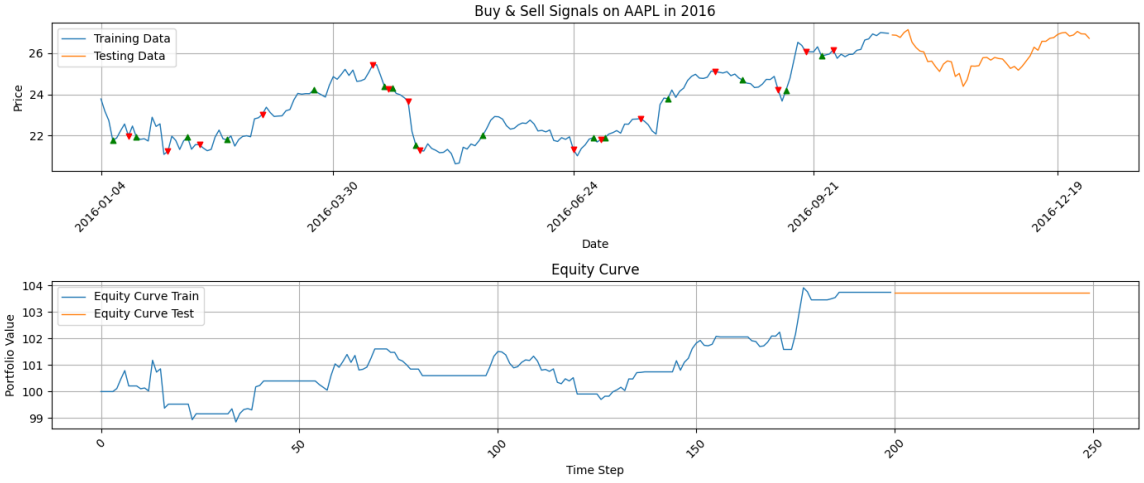

Buy & Sell Signals on AAPL in 2011
Training Data
Testing Data
Price
Date
2011-01-03
2011-03-29
2011-06-24
2011-09-21
2011-12-19
Equity Curve
Equity Curve Train
Equity Curve Test
Portfolio Value
Time Step

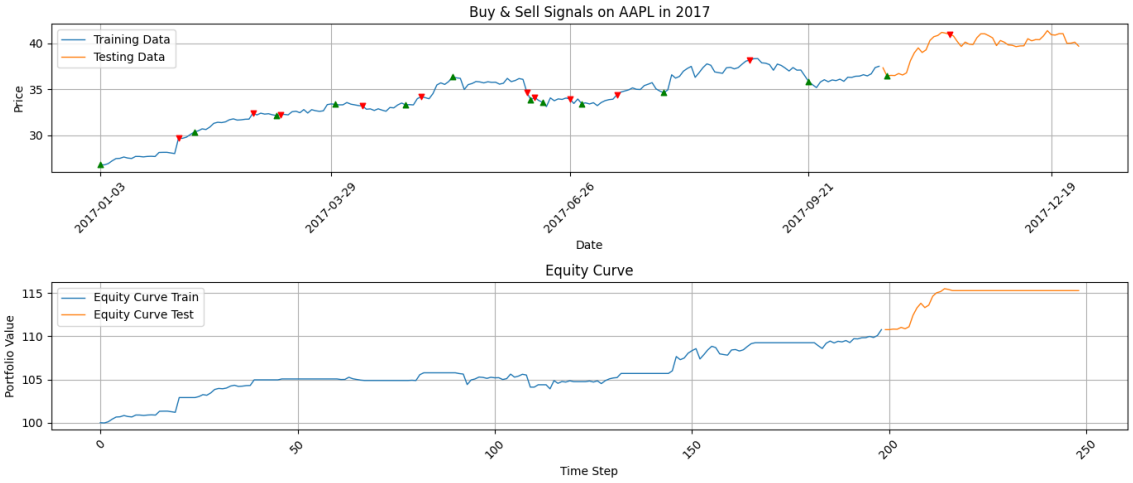

Buy & Sell Signals on AAPL in 2012
Training Data
Testing Data
Price
Date
2012-01-03
2012-03-28
2012-06-25
2012-09-20
2012-12-20
Equity Curve
Equity Curve Train
Equity Curve Test
Portfolio Value
Time Step

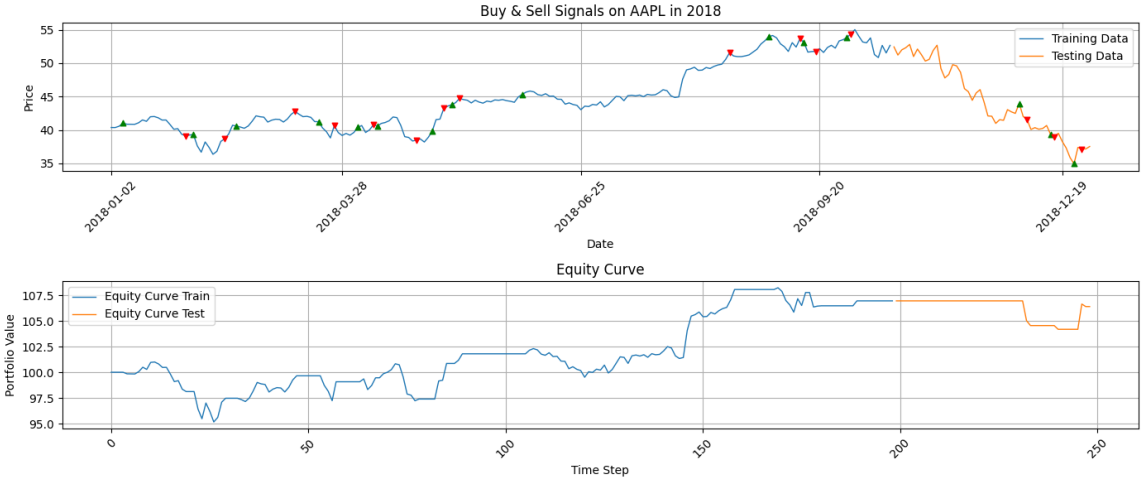

Buy & Sell Signals on AAPL in 2013
Training Data
Testing Data
Price
Date
2013-01-02
2013-03-28
2013-06-25
2013-09-20
2013-12-18
Equity Curve
Equity Curve Train
Equity Curve Test
Portfolio Value
Time Step

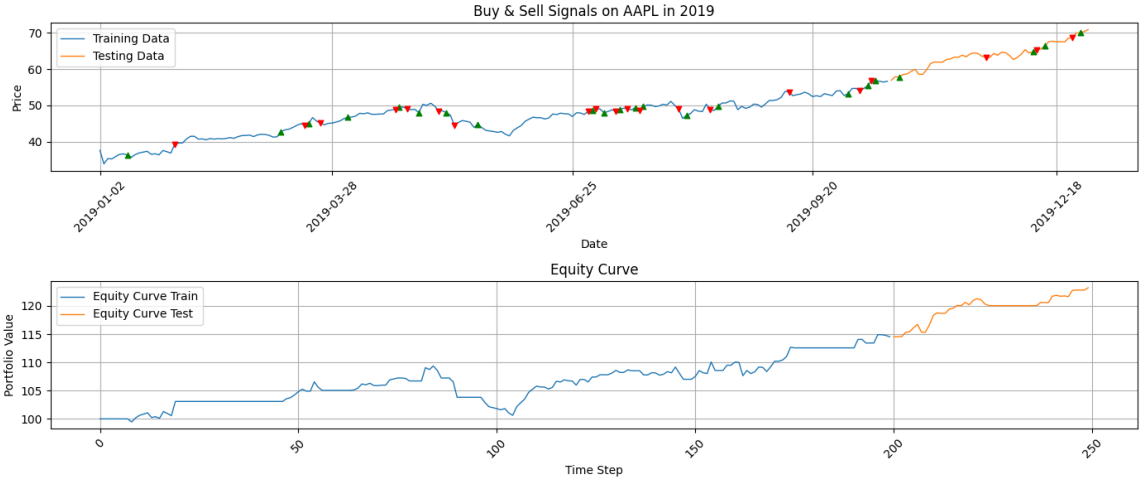

Buy & Sell Signals on AAPL in 2014
Training Data
Testing Data
Price
Date
2014-01-02
2014-03-28
2014-06-25
2014-09-22
2014-12-18
Equity Curve
Equity Curve Train
Equity Curve Test
Portfolio Value
Time Step

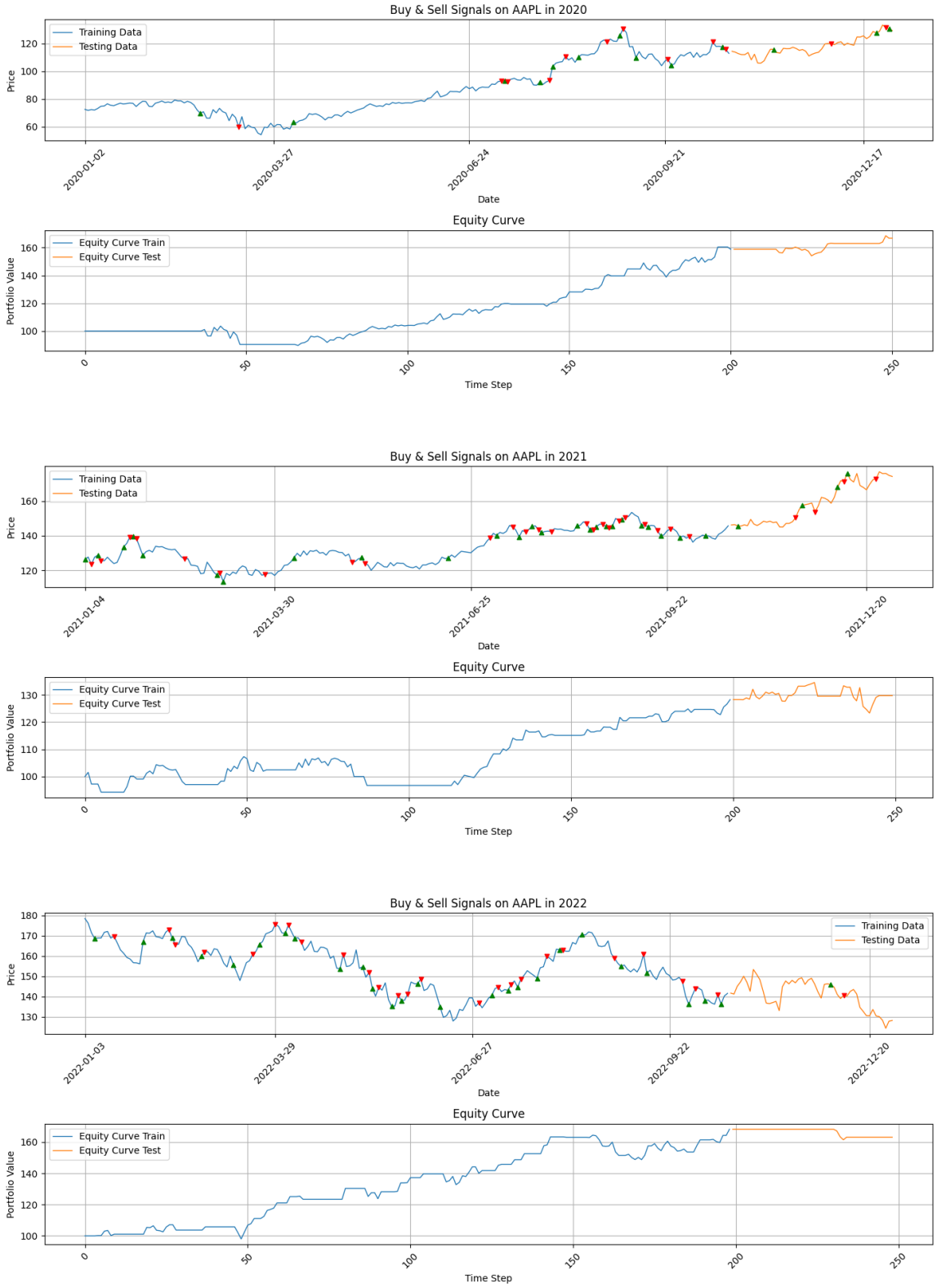

Buy & Sell Signals on AAPL in 2015
Training Data
Testing Data
Price
Date
2015-01-02
2015-03-30
2015-06-25
2015-09-22
2015-12-18
Equity Curve
Equity Curve Train
Equity Curve Test
Portfolio Value
Time Step


Buy & Sell Signals on AAPL in 2016
Training Data
Testing Data
Price
Date
2016-01-04
2016-03-30
2016-06-24
2016-09-21
2016-12-19
Equity Curve
Equity Curve Train
Equity Curve Test
Portfolio Value
Time Step


Buy & Sell Signals on AAPL in 2017
Training Data
Testing Data
Price
Date
2017-01-03
2017-03-29
2017-06-26
2017-09-21
2017-12-19
Equity Curve
Equity Curve Train
Equity Curve Test
Portfolio Value
Time Step


Buy & Sell Signals on AAPL in 2018
Training Data
Testing Data
Price
Date
2018-01-02
2018-03-28
2018-06-25
2018-09-20
2018-12-19
Equity Curve
Equity Curve Train
Equity Curve Test
Portfolio Value
Time Step


Buy & Sell Signals on AAPL in 2019
Training Data
Testing Data
Price
Date
2019-01-02
2019-03-28
2019-06-25
2019-09-20
2019-12-18
Equity Curve
Equity Curve Train
Equity Curve Test
Portfolio Value
Time Step



- Over 100 episodes, the cumulative rewards exhibited a modest upward trend, indicating initial learning progress, though with significant variability.
- Extending to 1000 episodes resulted showed that the agent is not taking position anymore, a convergence of the hold position shouyld be evaluated
- Value estimates became more consistent over time, though some fluctuations persisted due to the noisy environment dynamics (should add filtering or smoothing).

- The results highlight the trade-off between exploration and exploitation inherent in the greedy approach and motivate future exploration of strategies with controlled exploration. Noticed that the agent is not learning from the path but more on the direct previous value.

These preliminary findings establish a baseline for Q-Learning performance and set the stage for comparisons with other reinforcement learning methods such as Deep Q-Networks (DQN).

10.3 Challenges and Solutions

The Q-Learning experiments faced several challenges specific to the greedy policy and the trading environment :

- **Limited exploration due to greediness** : The strictly greedy policy limited the agent's ability to discover better actions, particularly early in training. Future work will include incorporating greedy or softmax action selection to balance exploration.
- **Reward noise and sparsity** : The stochastic environment produced noisy and sparse rewards, complicating learning.
- **Hyperparameter sensitivity** : Learning rate and discount factor tuning were critical, especially over longer training horizons. We will use optuna to get better hyperparameter in the future on more complexe models.
- **Computational efficiency** : Although less demanding than deep RL methods, Q-Learning with large state spaces required careful management of updates. We leveraged experience replay buffers and batch updates to accelerate convergence.

10.4 Next Steps

Building on the current progress with Q-Learning and the greedy policy, the immediate next steps focus on advancing the model complexity and preparing for PPO :

- Designing and integrating a deep neural network architecture to implement Deep Q-Learning (DQN), enabling function approximation for larger and continuous state spaces.
- Conducting experiments to compare the performance of DQN against the baseline Q-Learning greedy policy.
- Following the DQN implementation, proceeding with the development and fine-tuning of the PPO agent, including enhanced reward shaping and hyperparameter optimization.
- Expanding the trading environment to model more realistic market conditions such as transaction costs, slippage, and possibly varying liquidity.
- Performing ablation studies to isolate the effects of PPO-specific components like clipping, value function loss weighting, and entropy regularization.
- Enhancing visualization tools to track detailed performance metrics such as Sharpe ratio, maximum drawdown, and other risk-adjusted return measures.

These steps will establish a robust foundation for evaluating advanced reinforcement learning algorithms in the trading domain and inform subsequent research and publication efforts.