ISYE 6767 – Homework-4 Report

Trading Strategy method:

The trading_strategy function simulates a trading strategy on historical stock price data read from a CSV file. Here is a summary of how the method works:

1. Reading Data: The function checks if the filename is "aapl.csv". If it is, pandas is used to read the file and convert the 'Close' column into a numpy array. For other filenames, the CSV module reads the file and converts each row's first value to a float, skipping empty rows.
2. Initialization: It then initialises the length of the trading period, arrays for signals, positions, and account values with zeros. The starting account value is set to $10,000, and a fixed number of shares (10) is determined for transactions.
3. Trading Loop: The function iterates over each trading day and looks for signals:
    a. Buy Signal: If there is an increasing trend for three consecutive days and the current position plus the intended purchase does not exceed twice the fixed share number, a buy signal is set, the position is increased, and the purchase amount decreases the account value.
    b. Sell Signal: If there is a decreasing trend for two consecutive days and shares are in the position, a sell signal is set, the position is liquidated, and the sale amount increases the account value.
    c. End of Period: On the last day, if there are any shares left, they are sold off.
    d. Hold: On days without a clear buy or sell signal, the previous day's position is carried over.
4. Recording Results: After processing each day, the function writes the prices, signals, positions, and account values to a new CSV file, with its name derived from the input filename.
5. Profit or Loss Calculation: The function calculates the profit or loss by subtracting the initial account value from the final account value and prints the cumulative profit and loss.
6. Output: The function returns the arrays of signals, positions, and account values, which could be used for further analysis or visualization.

Outputs of the function:

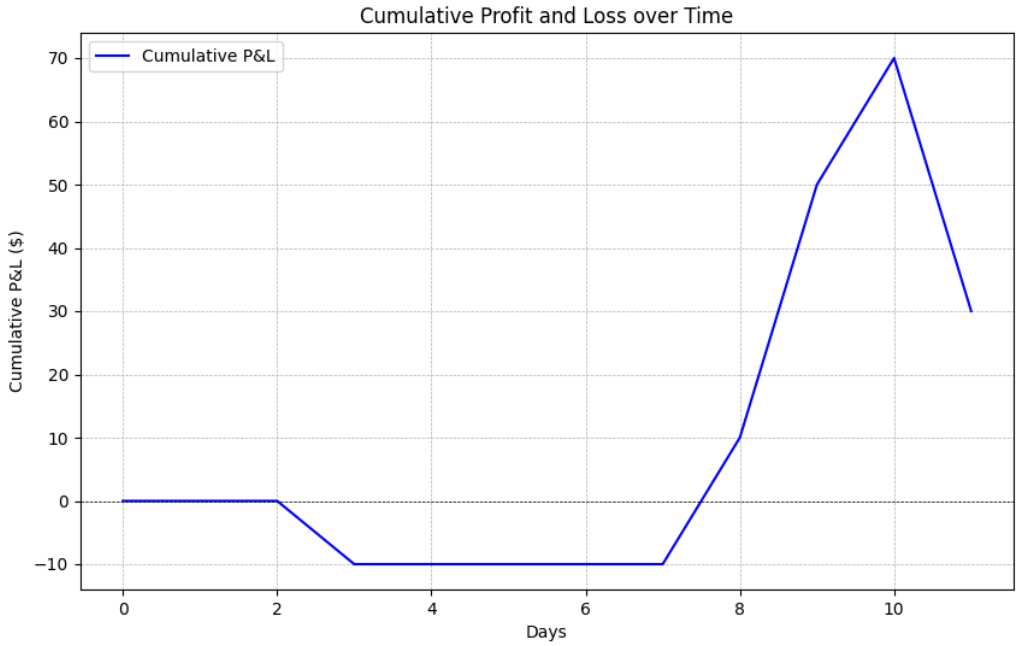
1. The trading strategy method returns the correct output for the input provided in the pdf i.e.

    For prices = [100, 102, 104, 103, 101, 99, 100, 102, 104, 106, 107, 105]

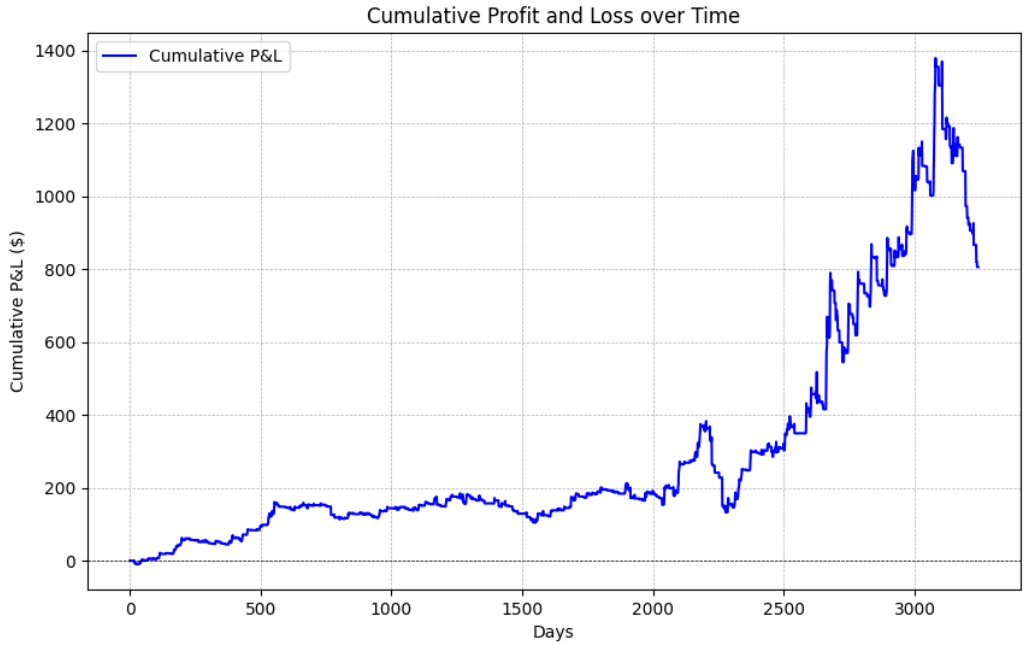
    signals = [0, 0, 1, 0, 0,−1, 0, 0, 1, 1, 0,−1]

    positions = [0, 0, 10, 0, 0, 0, 0, 0, 10, 20, 20, 0]

It returns the proper P&L and other outputs and the cumulative P&L for this example is found in this plot, the cumulative P&L for this price vector is Cumulative Trading Profit-and-Loss: $30.00



For other input of prices using the aapl.csv, I have considered the Close price as the price and computed the P&L per the trading strategy and for the prices of aapl the cumulative P&L is as follows

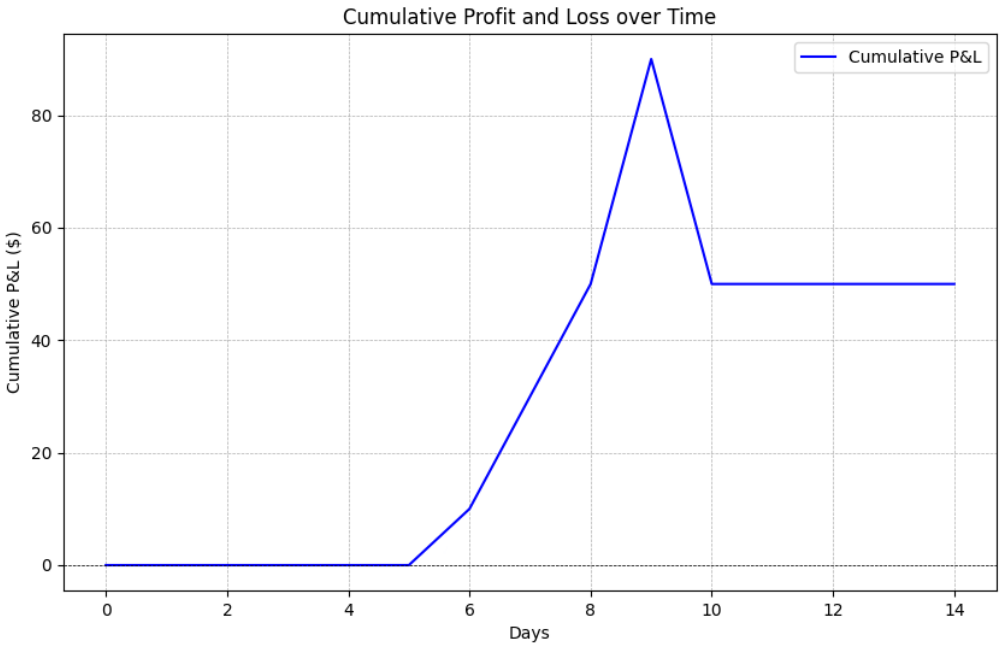


The cumulative P&L for appl close prices as input is Cumulative Trading Profit-and-Loss: $-707.30

For yet another concrete unit test prices are as follows

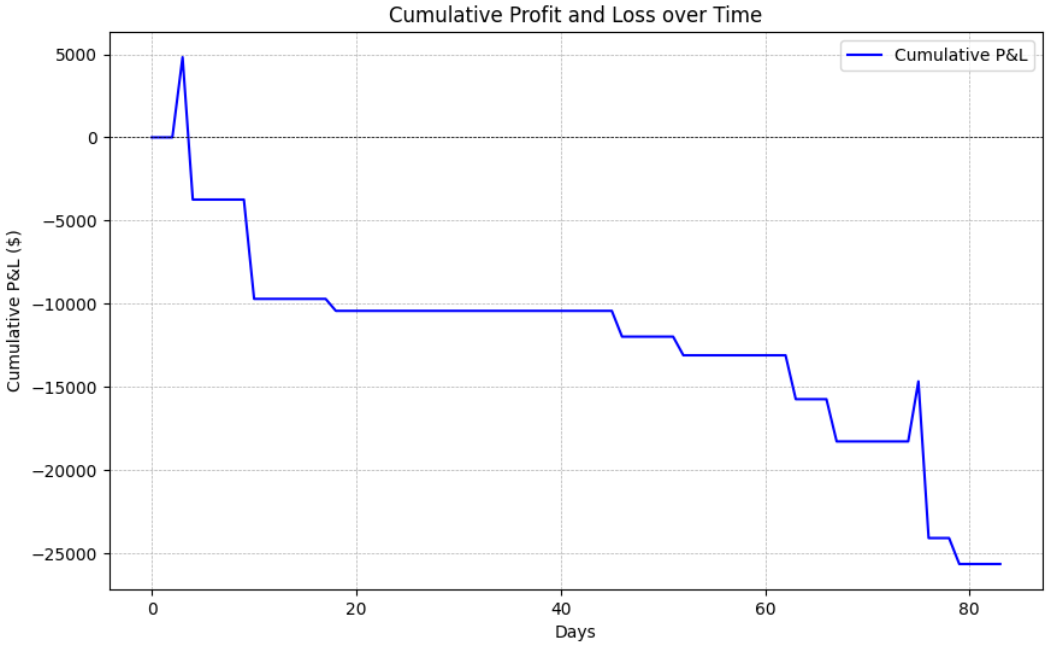
prices = [105, 103, 104, 103, 105, 107, 108, 109, 110, 112, 110, 108, 107, 106, 108]

The P&L is as follows



The cumulative P&L for the concrete unit test case-1 is Cumulative Trading Profit-and-Loss: $50.00

Lastly, for random prices that are generated using my gtid as a seed, the P&L is as follows



The Cumulative P&L for concrete test case-2 with random prices is Cumulative Trading Profit-and-Loss: $-25650.00

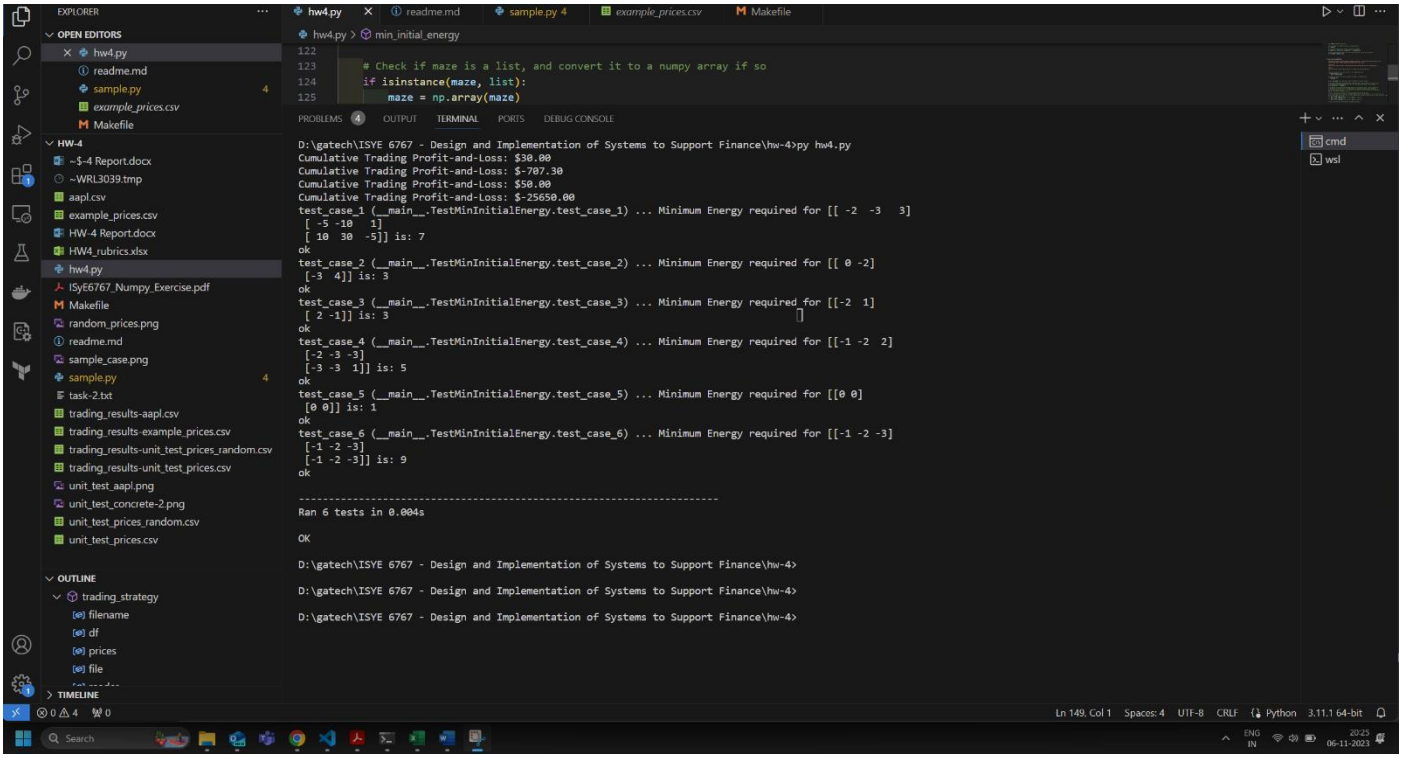
Note: For all the test cases for this function appropriate CSV files are generated with the output fields which can be viewed after execution

Min_initial_energy method:

- Purpose: The method calculates the minimum initial energy required to pass through a maze from the top-left to the bottom-right corner without the energy level dropping below 1 at any point.
- Input Conversion: If the input maze is a list, it is first converted to a numpy array to utilize array operations efficiently.
- Empty Maze Handling: The method immediately returns 0 if the maze is empty, as no energy is needed in this case.
- Dynamic Programming Approach: It uses a dynamic programming (DP) table to store the minimum energy required at each step when traversing from the destination to the starting point.
- Bottom-Right Cell Initialization: The energy level in the bottom-right cell (destination) is set to ensure the player has at least 1 energy point after accounting for the cell's energy value.
- Base Cases: The last row and last column of the DP table are filled first, representing the energy required when there is only one direction to move (right or down, respectively).
- DP Table Filling: The method fills the DP table in a bottom-up manner, ensuring that each cell's value is enough to move to the next cell (either to the right or below) while maintaining at least 1 energy point.
- Energy Calculation Logic: For each cell, the required energy is the energy needed to reach the next cell minus the energy value of the current cell, with a minimum of 1 energy point at every cell.
- Result: The value in the top-left cell of the DP table after filling indicates the minimum initial energy needed to start the maze and guarantees survival to the end.
- This method ensures an optimal solution due to its dynamic programming nature, considering all possible paths and energy configurations to find the minimum initial energy required.

Unit tests and outputs

This method is tested with 6 unit test cases the first of which is the example from the pdf and the remaining 5 are randomly generated, the method passes all the test cases and here’s a screenshot of the code passing all the test cases and the inputs can be seen in the unit test class that has been written. For the sample test case, the function returns 7 which is the correct expected output


EXPLORER
OPEN EDITORS
hw4.py
readme.md
sample.py 4
example_prices.csv
Makefile
HW-4
~$-4 Report.docx
~WRL3039.tmp
aapl.csv
example_prices.csv
HW-4 Report.docx
HW4_rubrics.xlsx
hw4.py
ISyE6767_Numpy_Exercise.pdf
Makefile
random_prices.png
readme.md
sample_case.png
sample.py 4
task-2.txt
trading_results-aapl.csv
trading_results-example_prices.csv
trading_results-unit_test_prices_random.csv
trading_results-unit_test_prices.csv
unit_test_aapl.png
unit_test_concrete-2.png
unit_test_prices_random.csv
unit_test_prices.csv
OUTLINE
trading_strategy
filename
df
prices
file
TIMELINE
hw4.py
readme.md
sample.py 4
example_prices.csv
Makefile
hw4.py > min_initial_energy
# Check if maze is a list, and convert it to a numpy array if so
if isinstance(maze, list):
maze = np.array(maze)
PROBLEMS 4
OUTPUT
TERMINAL
PORTS
DEBUG CONSOLE
D:\gatech\ISYE 6767 - Design and Implementation of Systems to Support Finance\hw-4>py hw4.py
Cumulative Trading Profit-and-Loss: $30.00
Cumulative Trading Profit-and-Loss: $-707.30
Cumulative Trading Profit-and-Loss: $50.00
Cumulative Trading Profit-and-Loss: $-25650.00
test_case_1 (__main__.TestMinInitialEnergy.test_case_1) ... Minimum Energy required for [[-2 -3 3]
[-5 -10 1]
[10 30 -5]] is: 7
ok
test_case_2 (__main__.TestMinInitialEnergy.test_case_2) ... Minimum Energy required for [[0 -2]
[-3 4]] is: 3
ok
test_case_3 (__main__.TestMinInitialEnergy.test_case_3) ... Minimum Energy required for [[-2 1]
[2 -1]] is: 3
ok
test_case_4 (__main__.TestMinInitialEnergy.test_case_4) ... Minimum Energy required for [[-1 -2 2]
[-2 -3 -3]
[-3 -3 1]] is: 5
ok
test_case_5 (__main__.TestMinInitialEnergy.test_case_5) ... Minimum Energy required for [[0 0]
[0 0]] is: 1
ok
test_case_6 (__main__.TestMinInitialEnergy.test_case_6) ... Minimum Energy required for [[-1 -2 -3]
[-1 -2 -3]
[-1 -2 -3]] is: 9
ok
Ran 6 tests in 0.004s
OK
D:\gatech\ISYE 6767 - Design and Implementation of Systems to Support Finance\hw-4>
D:\gatech\ISYE 6767 - Design and Implementation of Systems to Support Finance\hw-4>
D:\gatech\ISYE 6767 - Design and Implementation of Systems to Support Finance\hw-4>
cmd
wsl
Ln 149, Col 1
Spaces: 4
UTF-8
CRLF
Python
3.11.1 64-bit
Search
ENG
IN
20:25
06-11-2023