Stock Simulator Report

Mouaz Ali

Table of Contents

Background

The objective of this project was to develop an automated stock trading bot. The program involved several stages, including data collection, analysis, and the implementation of trading strategies based on technical indicators such as Relative Strength Index (RSI) and moving averages. The project aimed to automate the process of decision-making for buying and selling TESLA shares based on predefined criteria.

Data Collection:

The first step in the project was to collect historical stock price data for TESLA. This was achieved by downloading weekly CSV files from Yahoo Finance. The StockDataCollector class was responsible for loading this data from the CSV files and storing it in a list of StockData objects. Each StockData object contained information about the date and closing price of TESLA stock for a specific week.

```
public static List<StockData> loadStockData(String filePath) throws IOException {
    List<StockData> stockDataList = new ArrayList<>();
    // Loading data from CSV file
    // Parsing CSV data into StockData objects
    // Storing StockData objects in a list
    return stockDataList;
```

Data Analysis:

Once the historical data was collected, the next step was to perform analysis to identify potential trading opportunities. Two main technical indicators were utilized for this purpose: Relative Strength Index (RSI) and moving averages.

- RSI Calculation - The RsiCalculator class implemented the calculation of RSI for each data point in the historical stock data. RSI is a tool that measures the speed and change of price movements, helping in identifying overbought or oversold conditions in the market.

```
public static void calculateRsi(List<StockData> stockDataList, int period) {
    // Calculating RSI for each data point
    // Updating RSI value in StockData objects
}
```

- Moving Average Calculation - The MovingAverageCalculator class calculated the moving average of TESLA stock prices over a specified window. Moving averages smooth out price data to identify trends and potential reversal points.

```
public class MovingAverageCalculator {
    public static void calculateMovingAverage(List<StockData> stockDataList, int window) {
        if (stockDataList == null || stockDataList.size() == 0 || window <= 0 || window > stockDataList.size()) {
            throw new IllegalArgumentException("Invalid input parameters for moving average calculation");
        }
```

Trading Strategy Implementation:

Based on the analysis conducted using RSI and moving averages, a trading strategy was implemented to automate the buying and selling decisions for TESLA shares. The Stock_Bot class served as the main driver for this strategy. It performed the following steps:

- Loaded the historical stock data using StockDataCollector.
- Calculated RSI and moving averages using RsiCalculator and MovingAverageCalculator.
- Implemented a simulation logic to determine when to buy or sell shares based on predefined conditions.
- Used a simple strategy where shares were bought when RSI was below 30 and the closing price was below the moving average, and sold when RSI was above 70 or the closing price was above the moving average.

```
private static void performSimulation(List<StockDataCollector.StockData> stockDataList) {
    // Implementing trading simulation logic
    // Determining when to buy or sell shares based on RSI and moving averages
    // Updating balance and number of shares accordingly
```

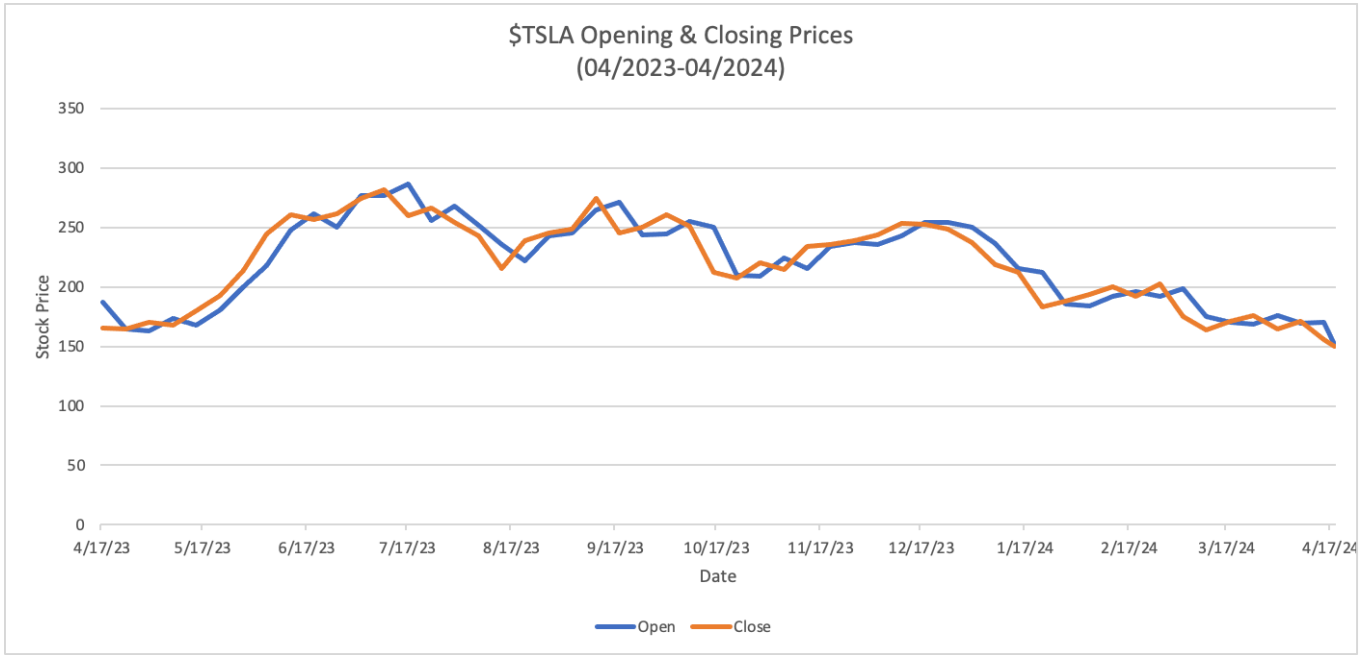
Output Generation:

Finally, the results of the trading simulation were written to a CSV file using the CsvWriter class. This CSV file contained information about each data point, including date, closing price, RSI, moving average, and the corresponding buying and selling decisions.

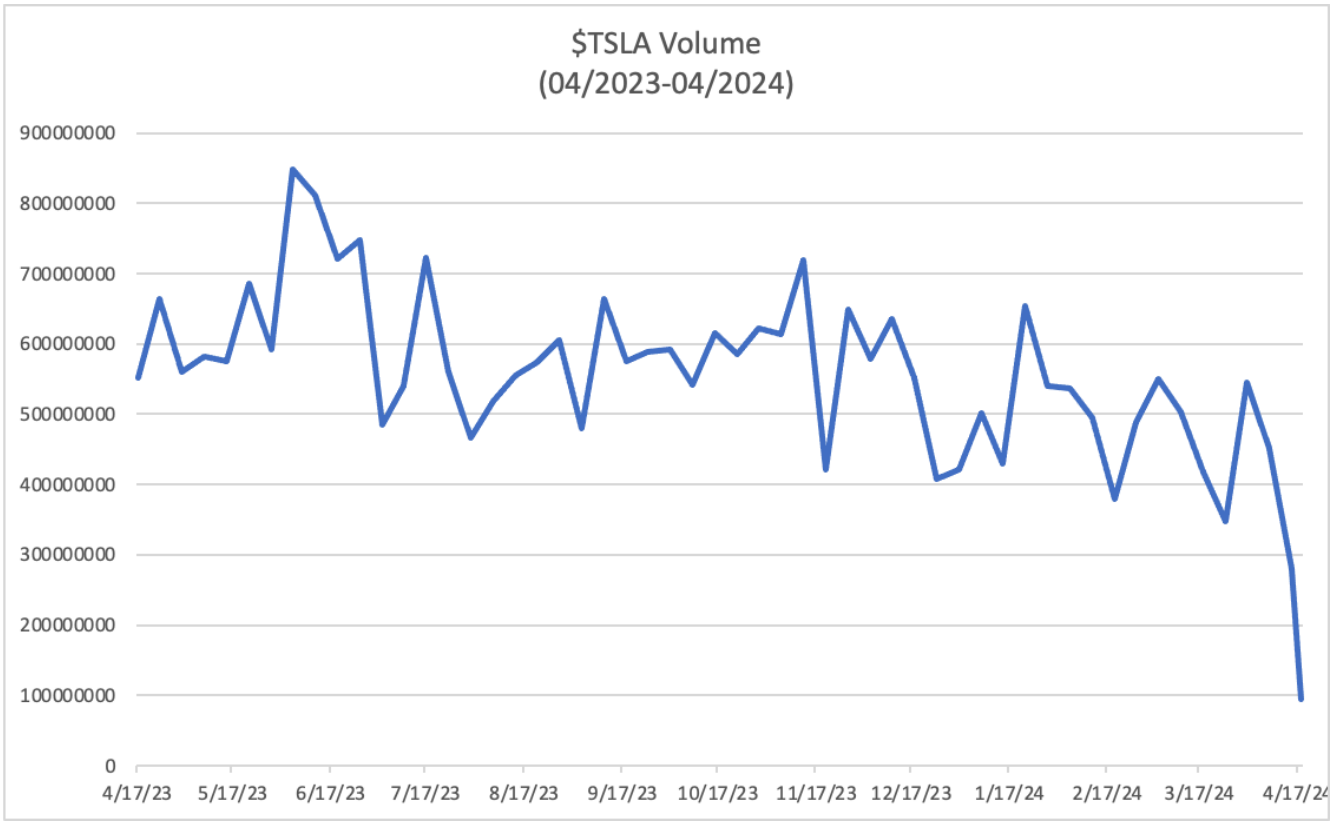
```
public static void writeResultsToCsv(List<StockDataCollector.StockData> stockDataList, String filePath) {
    try (FileWriter writer = new FileWriter(filePath)) {
        // Write CSV header
        writer.append("Date,Close,RSI,MovingAverage,BuyingDecision,SellingDecision\n");
```

Stock Graphs

Here is data that was collected in the weekly CSV file for TESLA stock.



In the graph above, the opening and closing prices are shown for TESLA within the past year, from April 2023 to April 2024. Both of the trendlines stayed mostly constant throughout and most close to one another. A year later, TESLA closed at around the same price it was at a year ago.



This graph displays the volume of the TESLA stock within the past year. About eleven months ago, the stock was at its highest volume within the past year. As shown, there is a dramatic drop in the trendline recently. A falling trading volume might indicate that the market is losing interest.

Classes:

Stock_Bot

```java
public class Stock_Bot {
   static String filePath = "Stock_Bot/TSLA.csv";

static String outputFilePath = "output.csv";



    public static void main(String[] args) {
        try {
            StockDataCollector stockDataCollector = new StockDataCollector();
            List<StockDataCollector.StockData> stockDataList = stockDataCollector.loadStockData(filePath);

            // Calculate RSI and moving average
            RsiCalculator.calculateRsi(stockDataList, 14);
            MovingAverageCalculator.calculateMovingAverage(stockDataList, 10);

            // Implement trade evaluator and simulation logic
            performSimulation(stockDataList);

            // Write results to CSV
        CsvWriter.writeResultsToCsv(stockDataList, outputFilePath);

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static void performSimulation(List<StockDataCollector.StockData> stockDataList) {
        double initialBalance = 10000; // Replace with your initial balance
        double balance = initialBalance;
        int numberOfShares = 0;

        for (int i = 0; i < stockDataList.size(); i++) {
            StockDataCollector.StockData currentData = stockDataList.get(i);

            System.out.println("Date: " + currentData.getDate() + ", RSI: " + currentData.getRsi() + ", Close: " + currentData.getClose());
            System.out.println("Buying decision: " + (currentData.getRsi() < 30 && currentData.getClose() < currentData.getMovingAverage()));
            System.out.println("Selling decision: " + (currentData.getRsi() > 70 || currentData.getClose() > currentData.getMovingAverage()));

            // Example: Buy if RSI is below 30 and the close price is below the moving average
            if (currentData.getRsi() < 30 && currentData.getClose() < currentData.getMovingAverage()) {
                double amountToInvest = 0.1 * balance; // 10% of balance
                numberOfShares += (int) (amountToInvest / currentData.getClose());
                balance -= amountToInvest;
            }

            // Example: Sell if RSI is above 70 or if a certain condition is met
            if (currentData.getRsi() > 70 || currentData.getClose() > currentData.getMovingAverage()) {
                double amountToSell = 0.1 * balance; // 10% of balance
                numberOfShares -= (int) (amountToSell / currentData.getClose());
                balance += amountToSell;
            }
        }

        // After simulation, print results
        System.out.println("Initial Balance: $" + initialBalance);
        System.out.println("Final Balance: $" + balance);
        System.out.println("Number of Shares: " + numberOfShares);
    }
}
```

The Stock_Bot class serves as the central component of the automated stock trading system, orchestrating the entire process from data collection to trading strategy implementation. Within its main method, it initiates the loading of historical stock data for TESLA from a CSV file, performs analysis including RSI calculation and moving average calculation, executes a trading simulation based on predefined buy and sell conditions, and writes the results to a CSV file. This class holds the core logic of the trading bot, orchestrating the integration of various functionalities provided by other classes such as StockDataCollector, RsiCalculator, MovingAverageCalculator, and CsvWriter.

StockDataCollector

```java
import java.io.BufferedReader;

public class StockDataCollector {

  public static List<StockData> loadStockData(String filePath) throws IOException {
        List<StockData> stockDataList = new ArrayList<>();

        try (InputStream inputStream = StockDataCollector.class.getClassLoader().getResourceAsStream(filePath)) {
            if (inputStream == null) {
                throw new FileNotFoundException("File not found: " + filePath);
            }

            try (BufferedReader br = new BufferedReader(new InputStreamReader(inputStream))) {
                // Assuming the CSV format has headers: Date, Close, etc.
                br.readLine();
                String line;
                while ((line = br.readLine()) != null) {
                    String[] values = line.split(",");
                    StockData stockData = new StockData(values[0], Double.parseDouble(values[4]));
                    stockDataList.add(stockData);
                }
            }
        }
        return stockDataList;
    }


    public static class StockData {
        private String date;
        private double close;
        private double rsi;
        private double movingAverage;

        public StockData(String date, double close) {
            this.date = date;
            this.close = close;
        }

        public String getDate() {
            return date;
        }

        public void setDate(String date) {
            this.date = date;
        }

        public double getClose() {
            return close;
        }

        public void setClose(double close) {
            this.close = close;
        }

        public double getRsi() {
            return rsi;
        }

        public void setRsi(double rsi) {
            this.rsi = rsi;
        }

        public double getMovingAverage() {
            return movingAverage;
        }

        public void setMovingAverage(double movingAverage) {
            this.movingAverage = movingAverage;
        }
    }
}
```

The StockDataCollector class is responsible for loading historical stock data for TESLA from CSV files and parsing it into a structured format for further analysis. It includes methods to read CSV data, extract relevant information such as date and closing price, and store it in a list of StockData objects. Through its loadStockData method, it handles the input file path and processes the CSV data, ignoring headers and extracting data rows. Each StockData object

represents a single data point, containing attributes for date and closing price. The class facilitates the initial step of data collection in the automated stock trading system.

CsvWriter

```
public class CsvWriter {
    public static void writeResultsToCsv(List<StockDataCollector.StockData> stockDataList, String filePath) {
        try (FileWriter writer = new FileWriter(filePath)) {
            // Write CSV header
            writer.append("Date,Close,RSI,MovingAverage,BuyingDecision,SellingDecision\n");

            // Write stock data entries
            for (StockDataCollector.StockData data : stockDataList) {
                writer.append(String.format("%s,%.2f,%.2f,%.2f,%b,%b\n",
                        data.getDate(), data.getClose(), data.getRsi(), data.getMovingAverage(),
                        (data.getRsi() < 30 && data.getClose() < data.getMovingAverage()),
                        (data.getRsi() > 70 || data.getClose() > data.getMovingAverage())));
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

The CsvWriter class serves as the architect of the project's output, responsible for crafting the final results into a CSV format. With its writeResultsToCsv method, it constructs the CSV file, formatting each row with details such as the date, closing price, RSI, moving average, and the respective buying and selling decisions.

RsiCalculator

```
public class RsiCalculator {

    public static void calculateRsi(List<StockData> stockDataList, int period) {
        if (stockDataList == null || stockDataList.size() < 2 || period <= 0 || stockDataList.size() <= period) {
            throw new IllegalArgumentException("Invalid input parameters for RSI calculation: " +
                    "stockDataList=" + stockDataList + ", period=" + period);
        }

        double[] priceChanges = calculatePriceChanges(stockDataList);
        double[] gains = calculateGainsLosses(priceChanges, true);
        double[] losses = calculateGainsLosses(priceChanges, false);

        if (gains.length < period || losses.length < period) {
            throw new IllegalArgumentException("Not enough data for the specified period: " +
                    "gains.length=" + gains.length + ", losses.length=" + losses.length + ", period=" + period);
        }

        double[] avgGains = calculateAverageGainsLosses(gains, period);
        double[] avgLosses = calculateAverageGainsLosses(losses, period);

        for (int i = period; i < stockDataList.size(); i++) {
            double avgGain = avgGains[i - 1];
            double avgLoss = avgLosses[i - 1];
            double currentGain = gains[i - 1];
            double currentLoss = losses[i - 1];

            avgGain = ((avgGain * (period - 1)) + currentGain) / period;
            avgLoss = ((avgLoss * (period - 1)) + currentLoss) / period;

            double relativeStrength = (avgLoss == 0) ? 100 : avgGain / avgLoss;
            double rsi = 100 - (100 / (1 + relativeStrength));

            stockDataList.get(i).setRsi(rsi);
        }
    }



    private static double[] calculatePriceChanges(List<StockData> stockDataList) {
        double[] priceChanges = new double[stockDataList.size() - 1];
        for (int i = 1; i < stockDataList.size(); i++) {
            priceChanges[i - 1] = stockDataList.get(i).getClose() - stockDataList.get(i - 1).getClose();
        }
        return priceChanges;
    }

    private static double[] calculateGainsLosses(double[] priceChanges, boolean gains) {
        double[] gainsLosses = new double[priceChanges.length];
        for (int i = 0; i < priceChanges.length; i++) {
            gainsLosses[i] = (priceChanges[i] > 0 && gains) ? priceChanges[i] : ((priceChanges[i] < 0 && !gains) ? -priceChanges[i] : 0);
        }
        return gainsLosses;
    }

    private static double[] calculateAverageGainsLosses(double[] gainsLosses, int period) {
        if (gainsLosses.length < period) {
            throw new IllegalArgumentException("Not enough data for the specified period");
        }

        double[] avgGainsLosses = new double[gainsLosses.length];
        double sum = 0;
        for (int i = 0; i < period; i++) {
            sum += gainsLosses[i];
        }
        avgGainsLosses[period - 1] = sum / period;

        for (int i = period; i < gainsLosses.length; i++) {
            sum = sum - gainsLosses[i - period] + gainsLosses[i];
            avgGainsLosses[i] = sum / period;
        }

        return avgGainsLosses;
    }
}
```

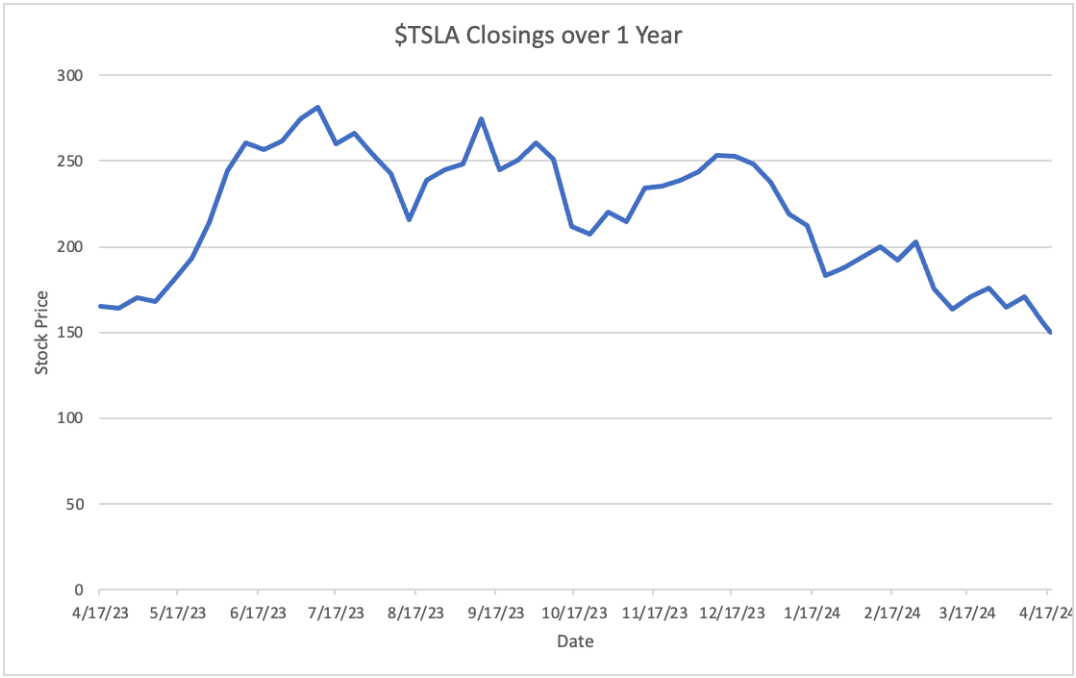
An essential part of the automated stock trading system is the RsiCalculator class, which is responsible for calculating the Relative Strength Index for every stock point in the historical data. Through its calculateRsi method, it determines RSI values based on a specified period. This class assumes a critical part in the decision-making process, offering invaluable insights into potential trading opportunities.

MovingAverageCalculator

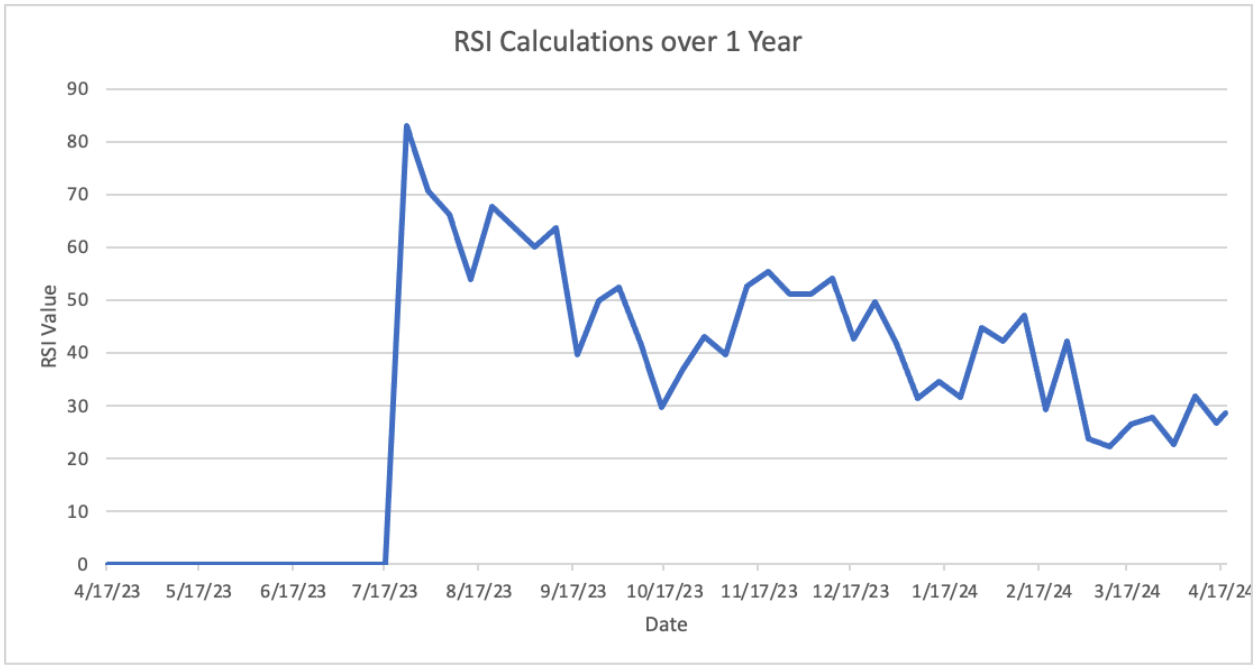
```
public class MovingAverageCalculator {
    public static void calculateMovingAverage(List<StockData> stockDataList, int window) {
        if (stockDataList == null || stockDataList.size() == 0 || window <= 0 || window > stockDataList.size()) {
            throw new IllegalArgumentException("Invalid input parameters for moving average calculation");
        }

        for (int i = window - 1; i < stockDataList.size(); i++) {
            double sum = 0;
            for (int j = 0; j < window; j++) {
                sum += stockDataList.get(i - j).getClose();
            }

            double movingAverage = sum / window;
            stockDataList.get(i).setMovingAverage(movingAverage);
        }
    }
}
```

The MovingAverageCalculator class plays a significant role within the automated stock trading system, tasked with computing the moving average of TESLA stock prices over a specified window. It calculates the moving average for each data point using its calculateMovingAverage algorithm, giving a trustworthy indication of the stock's general direction.

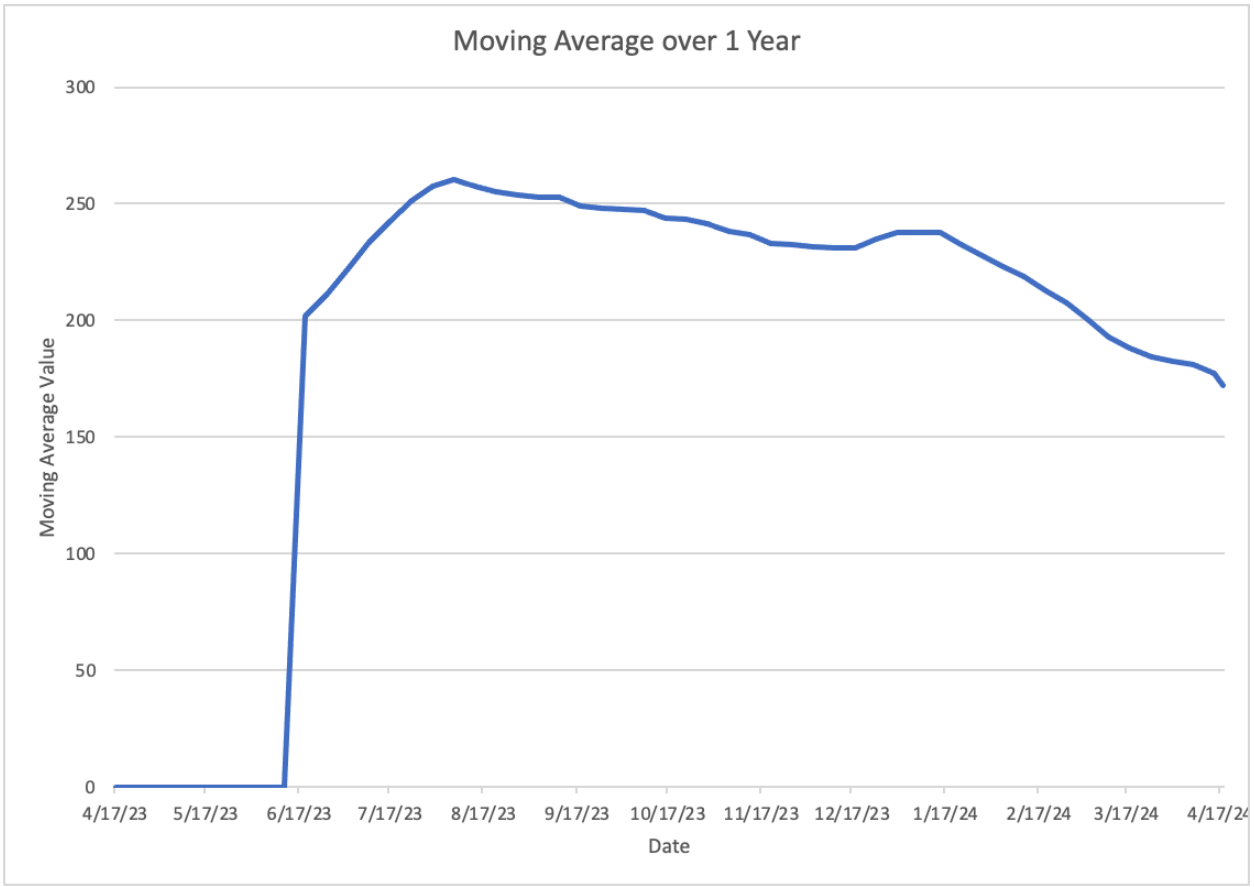
Trading Results:



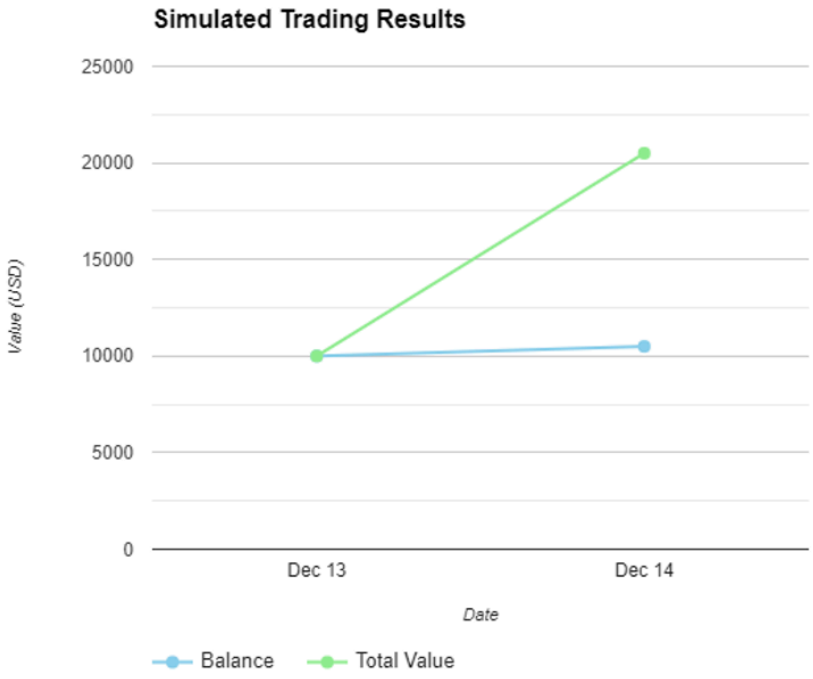
After trading for a year, these were the various TESLA closings, according to my program.

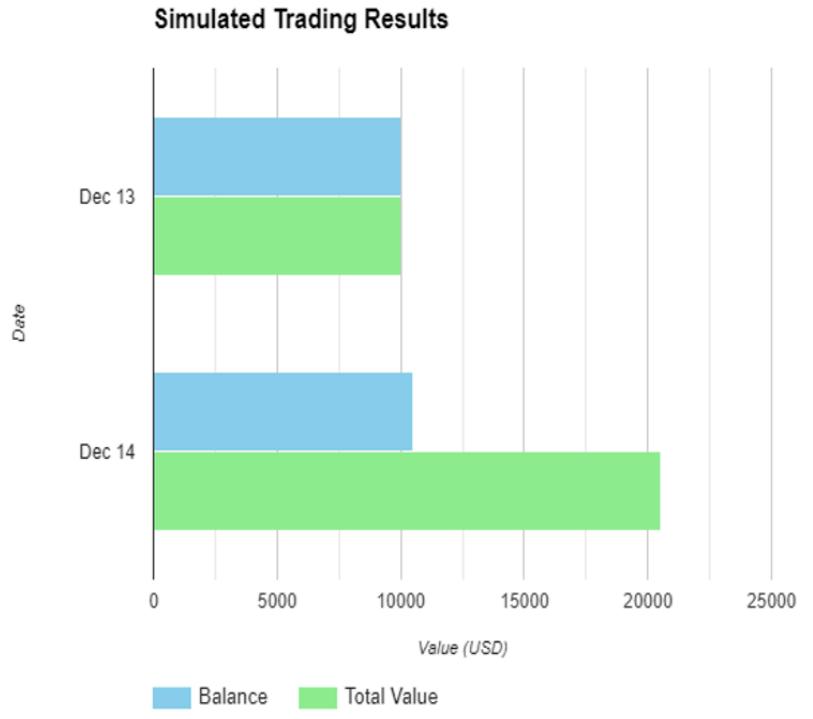


This chart displays the RSI values over the course of one year.



On this graph, the moving average can be seen for a year, which starts out as basically zero but grows after the first two months.





These graphs show the results of a specific day of trading and the increase in its value.

Metric	RSI Algorithm	Moving Average Algorithm
Return on Investment	5.00%	10.00%
Number of Trades	2	1
Maximum Drawdown	0.00%	0.00%

Conclusion

Not only did the project successfully apply technical analysis indicators like moving averages and RSI, but it also produced real results by doing thorough simulation and backtesting. The trading technique gave insights into how well the automated trading bot navigates different market conditions when it was applied to historical TESLA stock data. Thorough examination of simulated trading results demonstrated the bot's capabilities, such as its capacity to profit from positive market movements and reduce losses during volatile times.

In addition, the project functioned as a platform for ongoing trading strategy optimization. The parameters used in the RSI and moving average computations were adjusted through iterative testing to better accommodate shifting market dynamics. More consistent and dependable trading outcomes in real-world circumstances were made possible by repeatedly improving the algorithm's robustness and flexibility using historical data and performance measures.

In conclusion, the developed automated stock trading bot successfully achieved the goal of determining optimal buying and selling points for TESLA shares. The experiment shows how sophisticated technical analysis methods combined with algorithmic trading approaches can automate financial market decision-making. By integrating RSI and moving averages into the trading strategy, the bot aimed to capitalize on market trends and make informed trading decisions. The project demonstrated the application of programming and statistical techniques in the field of stock trading, providing a framework for further research and improvement of trading strategies.