COS30018: Intelligent Systems Option B: Stock Prediction

Task B.8 – Extend Researching

Name: Duc Thuan Tran
Student ID: 104330455
Tutor: Dr. Ru Jia
Tutorial: Friday 2:30 – 4:30

Table of Contents

Introduction

After completing seven key enhancements (from B1 to B7) to the stock price prediction code, reaching its most complete and refined state, I believe there are still additional opportunities to further improve decision-making beyond relying solely on models and external data sources. This report marks the release of version v07 of the code, which focuses on optimizing stock price predictions through improved strategy decision-making. The notebook, titled trading_strategies.ipynb, demonstrates the backtesting environment I've established and outlines the strategic decisions implemented for better stock trading outcomes. The following sections will dive into the enhancements made and how they contribute to more effective decision-making in stock price prediction.

Backtesting Environment

The provided code sets up a backtesting environment for a trading strategy that uses Simple Moving Averages (SMA) on historical stock price data from 2015 to 2019. Below is a breakdown of the code and its purpose:

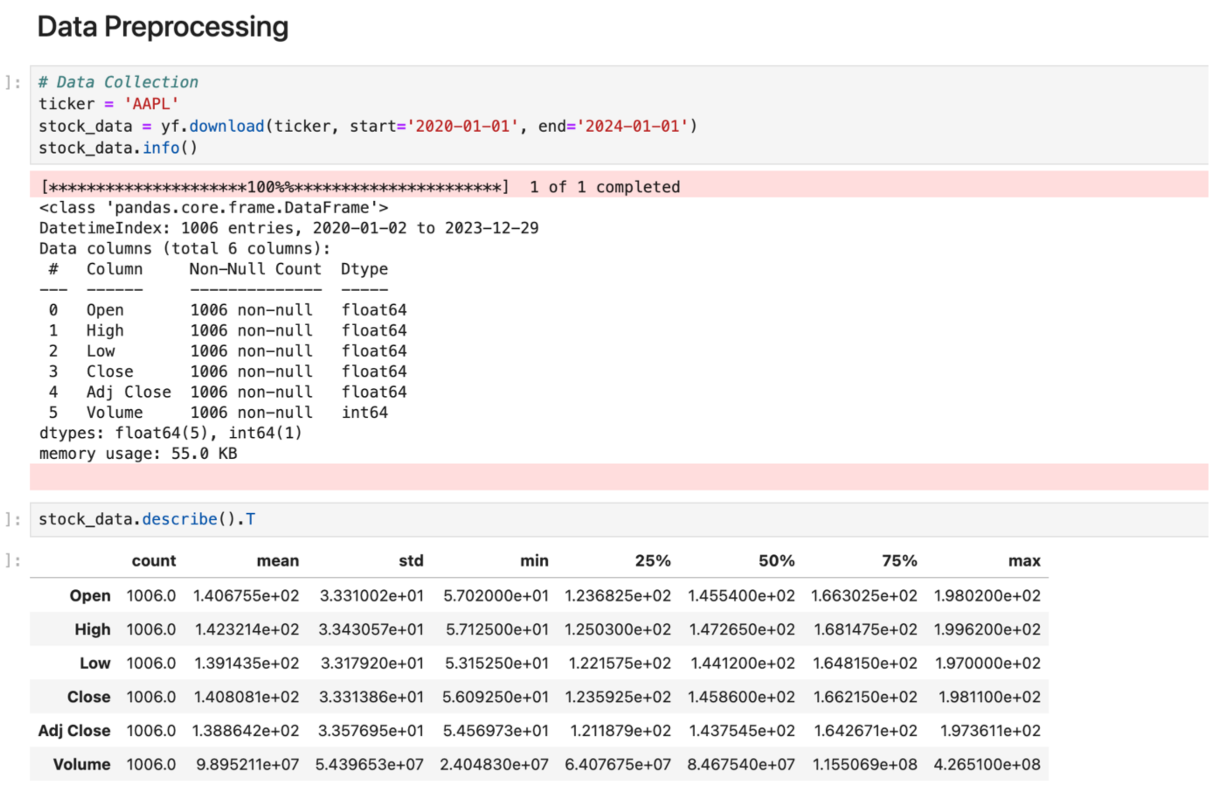
Code Explanation:

1. **SmaCross Class (Strategy):**
   - This class inherits from a Strategy base class and implements the logic for trading based on SMA crossovers.
   - The constructor (__init__) initializes two moving averages (ma1 and ma2):
     - ma1 is a short-term SMA (with a window of 10 periods).
     - ma2 is a long-term SMA (with a window of 20 periods).
   - The next method contains the logic that runs on each period:
     - If ma1 (short SMA) crosses above ma2 (long SMA), a **buy** signal is triggered.
     - If ma1 crosses below ma2, a **sell** signal is triggered.
2. **Data Loading:**
   - The historical data for the Apple (AAPL) stock is downloaded using the yfinance library for the period between January 1, 2015, and January 1, 2019.
3. **Backtest Setup:**
   - The backtesting environment is set up using the Backtest class, which takes in the downloaded historical data, the SmaCross strategy, and certain parameters like commission fees (0.002) and exclusive order handling.
4. **Running the Backtest:**
   - The backtest.run() command executes the backtest simulation, applying the strategy to the historical data.
   - A final plot of the results is generated and saved as backtest_results.html, which opens in the browser to visually show the strategy's performance.

Purpose of the Code for Stock Price Prediction:

The purpose of this code is to simulate the performance of a basic moving average crossover trading strategy on historical stock price data. By backtesting the strategy on past data, we can analyze its potential profitability and evaluate how it would have performed in real-world conditions. This approach helps in making informed decisions about whether the strategy is viable for predicting stock price trends and timing market entries and exits.

The backtest environment allows us to tweak the parameters, such as the length of the moving averages, and assess how these changes impact the overall results. In the broader context of stock price prediction, this type of backtesting provides a systematic way to test hypotheses about market behavior and to refine trading models before applying them to live trading.



Data Preprocessing

```
# Data Collection
ticker = 'AAPL'
stock_data = yf.download(ticker, start='2020-01-01', end='2024-01-01')
stock_data.info()
```

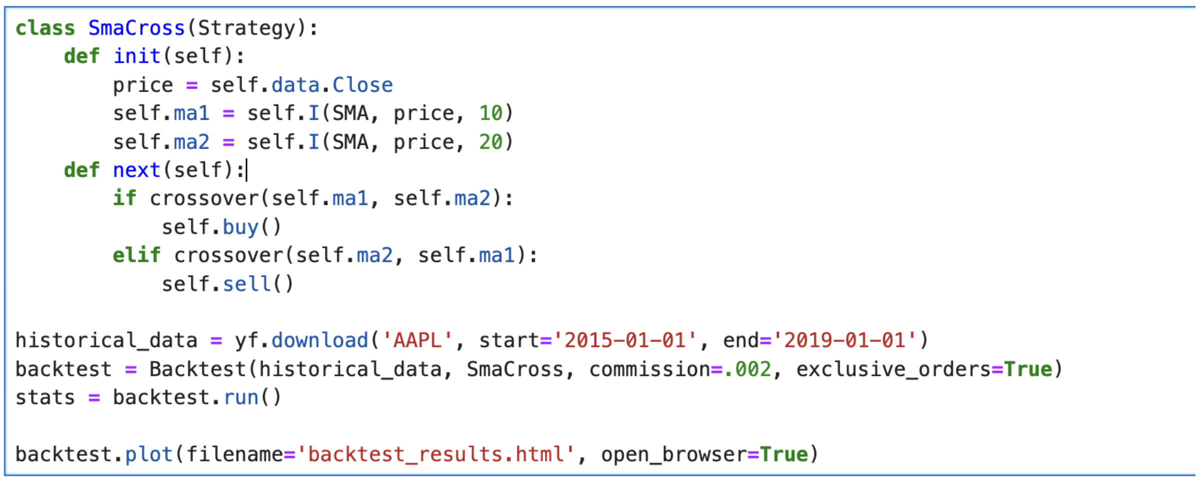
```
[*********************100%***********************]  1 of 1 completed
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1006 entries, 2020-01-02 to 2023-12-29
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Open       1006 non-null   float64
 1   High       1006 non-null   float64
 2   Low        1006 non-null   float64
 3   Close      1006 non-null   float64
 4   Adj Close  1006 non-null   float64
 5   Volume     1006 non-null   int64
dtypes: float64(5), int64(1)
memory usage: 55.0 KB
```

```
stock_data.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Open	1006.0	1.406755e+02	3.331002e+01	5.702000e+01	1.236825e+02	1.455400e+02	1.663025e+02	1.980200e+02
High	1006.0	1.423214e+02	3.343057e+01	5.712500e+01	1.250300e+02	1.472650e+02	1.681475e+02	1.996200e+02
Low	1006.0	1.391435e+02	3.317920e+01	5.315250e+01	1.221575e+02	1.441200e+02	1.648150e+02	1.970000e+02
Close	1006.0	1.408081e+02	3.331386e+01	5.609250e+01	1.235925e+02	1.458600e+02	1.662150e+02	1.981100e+02
Adj Close	1006.0	1.388642e+02	3.357695e+01	5.456973e+01	1.211879e+02	1.437545e+02	1.642671e+02	1.973611e+02
Volume	1006.0	9.895211e+07	5.439653e+07	2.404830e+07	6.407675e+07	8.467540e+07	1.155069e+08	4.265100e+08

Image 1. Data preprocessing for Stock Price Prediction (AAPL)

BackTesting Environment (2015 - 2019)



```
class SmaCross(Strategy):
    def init(self):
        price = self.data.Close
        self.ma1 = self.I(SMA, price, 10)
        self.ma2 = self.I(SMA, price, 20)
    def next(self):
        if crossover(self.ma1, self.ma2):
            self.buy()
        elif crossover(self.ma2, self.ma1):
            self.sell()

historical_data = yf.download('AAPL', start='2015-01-01', end='2019-01-01')
backtest = Backtest(historical_data, SmaCross, commission=.002, exclusive_orders=True)
stats = backtest.run()

backtest.plot(filename='backtest_results.html', open_browser=True)
```

Image 2. Backtesting Environment with Sma Strategy (2015 - 2019)

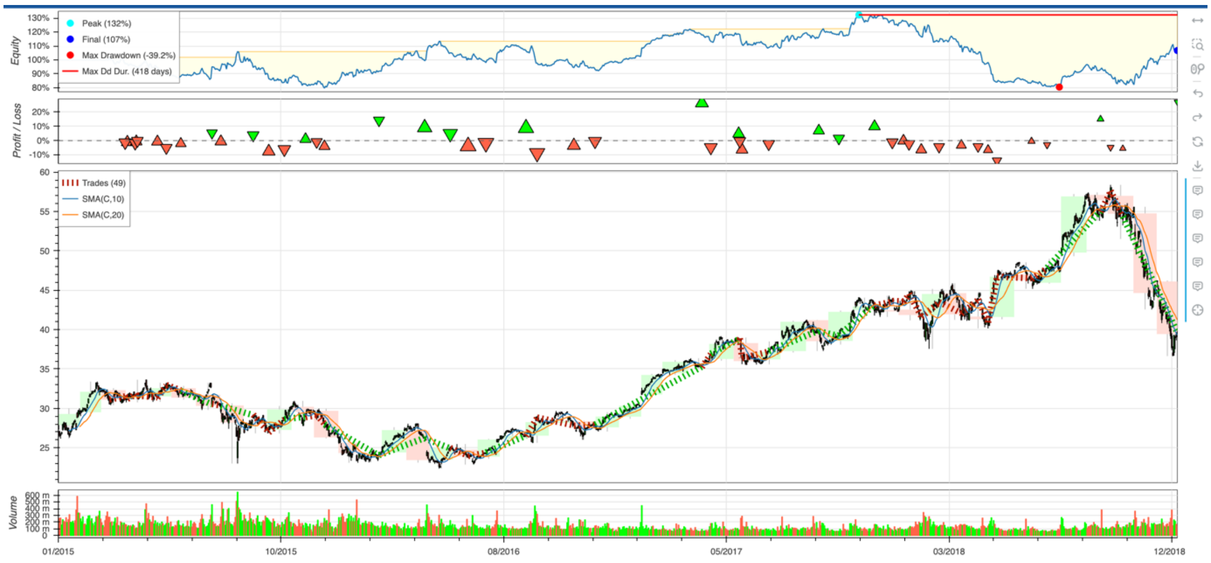


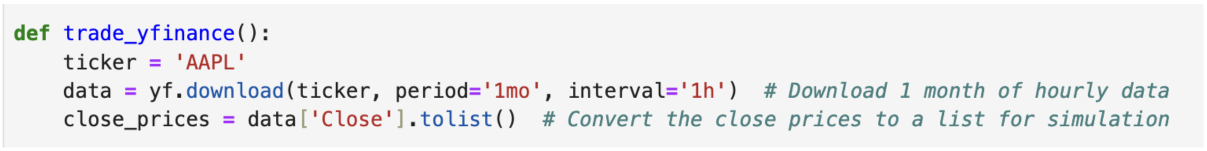
Image 3. SMA Crossover Strategy Backtest Results (2015-2019)

Strategy Decision-Making: Using 'Close' Column

This section introduces the implementation of a trading strategy that combines the Relative Strength Index (RSI) and Simple Moving Averages (SMA) for decision-making. The code uses hourly price data for the stock ticker 'AAPL' over the last month, fetched from yfinance, and simulates trades based on specific conditions like SMA crossovers, RSI thresholds, stop-loss, and take-profit mechanisms. Below is a breakdown of the code and its purpose.

Code Breakdown:

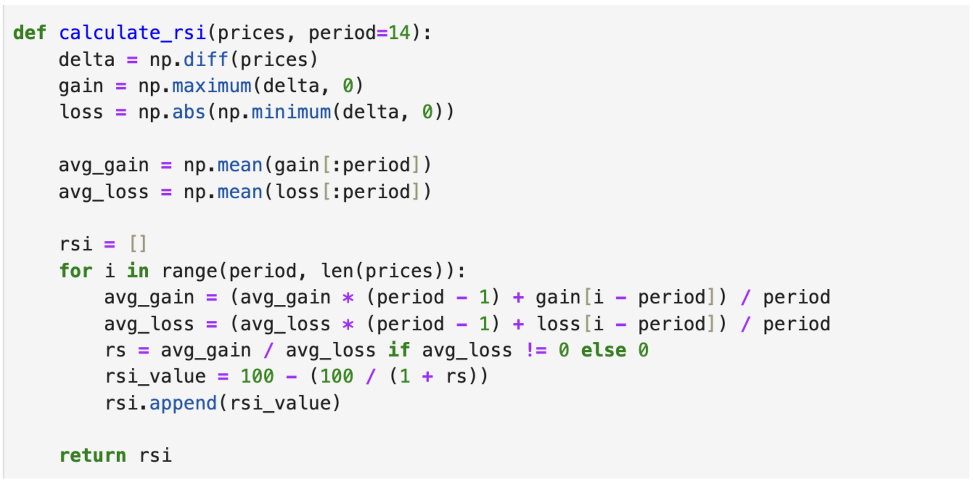
1. **Data Collection:**
   - The stock data for 'AAPL' is downloaded from yfinance for a one-month period, with an interval of one hour. The close prices are extracted to be used in the simulation.



```python
def trade_yfinance():
    ticker = 'AAPL'
    data = yf.download(ticker, period='1mo', interval='1h')  # Download 1 month of hourly data
    close_prices = data['Close'].tolist()  # Convert the close prices to a list for simulation
```

Image 4. Downloading Hourly data for AAPL using yfinance

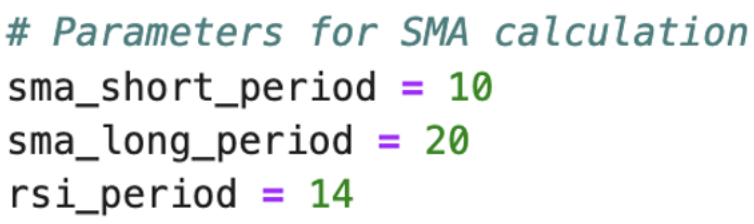
2. **RSI Calculation:**
   - A custom calculate_rsi function computes the Relative Strength Index (RSI) over a 14-period window. RSI is a momentum indicator that compares the magnitude of recent gains to recent losses and provides a value ranging from 0 to 100. It helps determine whether a stock is overbought or oversold, aiding in buy/sell decisions.



```python
def calculate_rsi(prices, period=14):
    delta = np.diff(prices)
    gain = np.maximum(delta, 0)
    loss = np.abs(np.minimum(delta, 0))

    avg_gain = np.mean(gain[:period])
    avg_loss = np.mean(loss[:period])

    rsi = []
    for i in range(period, len(prices)):
        avg_gain = (avg_gain * (period - 1) + gain[i - period]) / period
        avg_loss = (avg_loss * (period - 1) + loss[i - period]) / period
        rs = avg_gain / avg_loss if avg_loss != 0 else 0
        rsi_value = 100 - (100 / (1 + rs))
        rsi.append(rsi_value)

    return rsi
```

Image 5. Caculate rsi

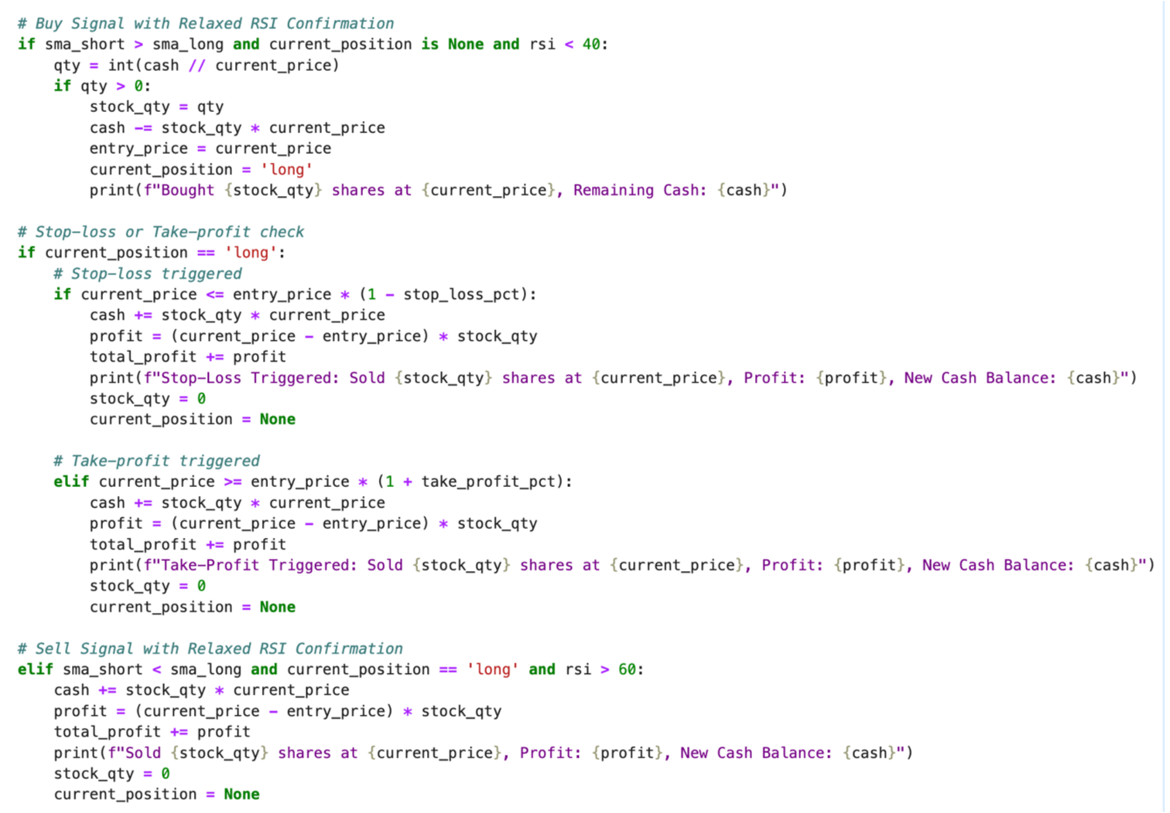
3. **Strategy Logic:**
   - **SMA Crossover:** The strategy calculates a short-term (10-period) and a long-term (20-period) SMA. A buy signal is triggered when the short-term SMA crosses above the long-term SMA, and a sell signal is triggered when the opposite occurs.



```python
# Parameters for SMA calculation
sma_short_period = 10
sma_long_period = 20
rsi_period = 14
```

Image 6. Sma parameter

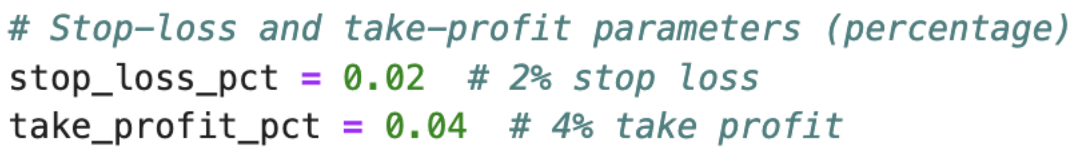
   - **RSI Confirmation:** Buy and sell signals are further confirmed based on the RSI value. The strategy enters a "long" position if the short SMA crosses above the long SMA and the RSI is below 40 (indicating oversold conditions). Conversely, it exits a long position if the short SMA crosses below the long SMA and the RSI exceeds 60 (indicating overbought conditions).



```python
# Buy Signal with Relaxed RSI Confirmation
if sma_short > sma_long and current_position is None and rsi < 40:
    qty = int(cash // current_price)
    if qty > 0:
        stock_qty = qty
        cash -= stock_qty * current_price
        entry_price = current_price
        current_position = 'long'
        print(f"Bought {stock_qty} shares at {current_price}, Remaining Cash: {cash}")

# Stop-loss or Take-profit check
if current_position == 'long':
    # Stop-loss triggered
    if current_price <= entry_price * (1 - stop_loss_pct):
        cash += stock_qty * current_price
        profit = (current_price - entry_price) * stock_qty
        total_profit += profit
        print(f"Stop-Loss Triggered: Sold {stock_qty} shares at {current_price}, Profit: {profit}, New Cash Balance: {cash}")
        stock_qty = 0
        current_position = None

    # Take-profit triggered
    elif current_price >= entry_price * (1 + take_profit_pct):
        cash += stock_qty * current_price
        profit = (current_price - entry_price) * stock_qty
        total_profit += profit
        print(f"Take-Profit Triggered: Sold {stock_qty} shares at {current_price}, Profit: {profit}, New Cash Balance: {cash}")
        stock_qty = 0
        current_position = None

# Sell Signal with Relaxed RSI Confirmation
elif sma_short < sma_long and current_position == 'long' and rsi > 60:
    cash += stock_qty * current_price
    profit = (current_price - entry_price) * stock_qty
    total_profit += profit
    print(f"Sold {stock_qty} shares at {current_price}, Profit: {profit}, New Cash Balance: {cash}")
    stock_qty = 0
    current_position = None
```

Image 7. Buy, Sell, Stop-Loss, and Take-Profit Logic with RSI Confirmation

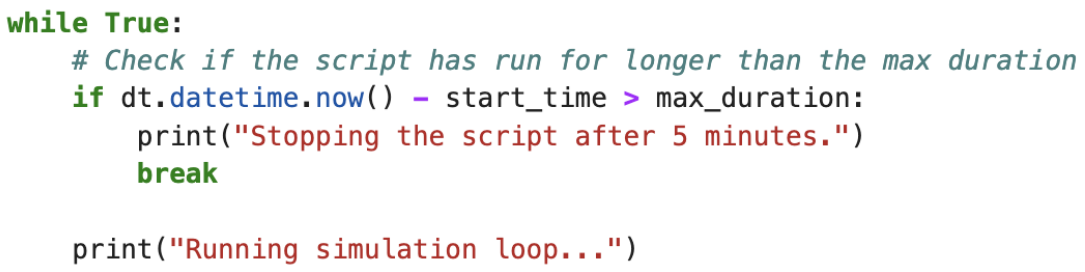
4. **Risk Management:**
   - **Stop-Loss and Take-Profit:** To limit risk, the strategy employs a 2% stop-loss and a 4% take-profit mechanism. If the stock price drops by more than 2% from the entry price, the position is closed to prevent further losses. Similarly, if the price increases by 4%, the position is closed to secure profits.



```python
# Stop-loss and take-profit parameters (percentage)
stop_loss_pct = 0.02   # 2% stop loss
take_profit_pct = 0.04   # 4% take profit
```

Image 8. Stop-loss, take-profit parameters

5. **Simulation Control:**
   - The simulation runs for a maximum duration of 5 minutes in a loop, with the code checking the stock price at regular intervals. If the simulation duration exceeds 5 minutes or the end of the historical price data is reached, the loop exit.

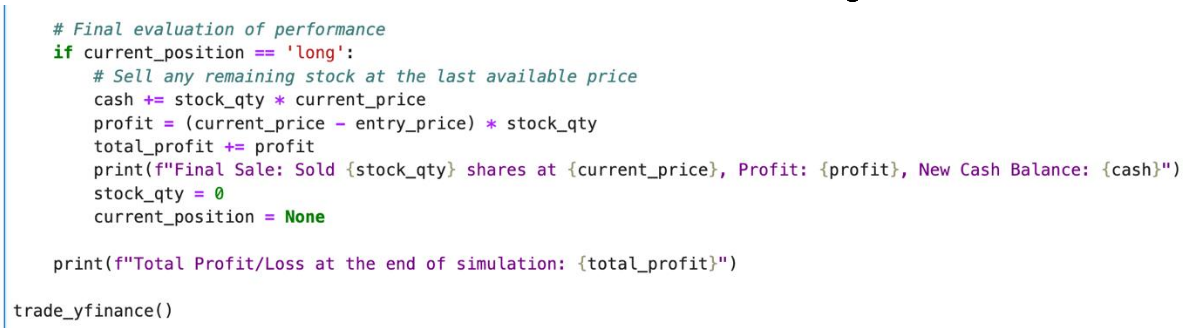


```python
while True:
    # Check if the script has run for longer than the max duration
    if dt.datetime.now() - start_time > max_duration:
        print("Stopping the script after 5 minutes.")
        break

    print("Running simulation loop...")
```

Image 9. Duration checking every 5 minutes

6. **Final Profit/Loss Calculation:**

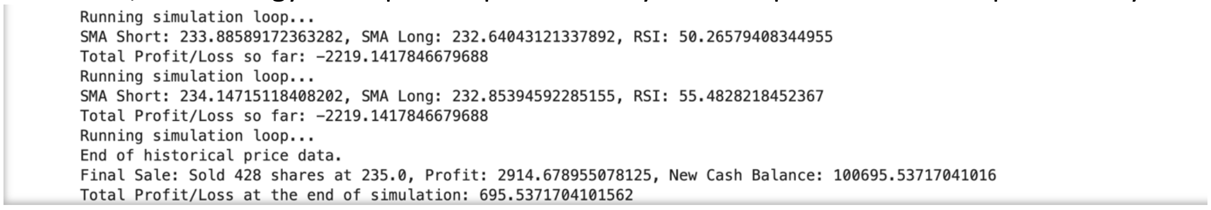
   - At the end of the simulation, any open positions are closed, and the final profit or loss is reported. The total profit or loss is accumulated throughout the simulation based on the number of shares bought and sold.



```
    # Final evaluation of performance
    if current_position == 'long':
        # Sell any remaining stock at the last available price
        cash += stock_qty * current_price
        profit = (current_price - entry_price) * stock_qty
        total_profit += profit
        print(f"Final Sale: Sold {stock_qty} shares at {current_price}, Profit: {profit}, New Cash Balance: {cash}")
        stock_qty = 0
        current_position = None

    print(f"Total Profit/Loss at the end of simulation: {total_profit}")

trade_yfinance()
```

Image 10. Final evaluation of performance

This simulation helps visualize how the combined RSI and SMA strategy performs in a short-term scenario using historical data. By leveraging moving averages and momentum indicators, the strategy attempts to optimize entry and exit points for better profitability.



```
Running simulation loop...
SMA Short: 233.88589172363282, SMA Long: 232.64043121337892, RSI: 50.26579408344955
Total Profit/Loss so far: -2219.1417846679688
Running simulation loop...
SMA Short: 234.14715118408202, SMA Long: 232.85394592285155, RSI: 55.4828218452367
Total Profit/Loss so far: -2219.1417846679688
Running simulation loop...
End of historical price data.
Final Sale: Sold 428 shares at 235.0, Profit: 2914.678955078125, New Cash Balance: 100695.53717041016
Total Profit/Loss at the end of simulation: 695.5371704101562
```

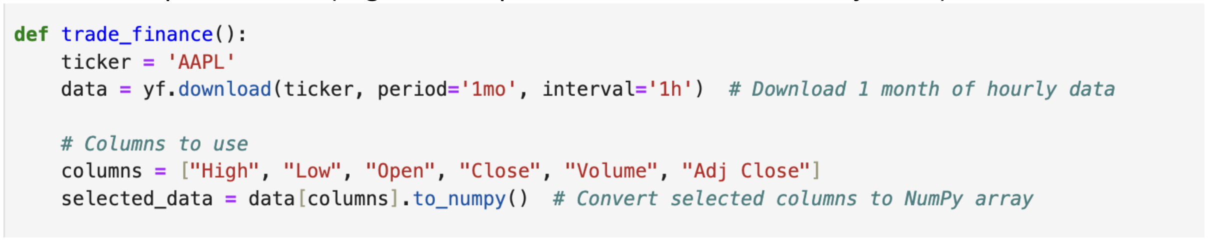
Image 11. Simulation output

Strategy Decision-Making: Using All Column

The key difference in this version of the trading simulation is that, instead of relying solely on the 'Close' prices for decision-making, it now incorporates additional columns such as **High**, **Low**, **Open**, **Close**, **Volume**, and **Adjusted Close**. This allows for more comprehensive analysis and decision-making based on more granular data points.

Differences from the Previous Version:

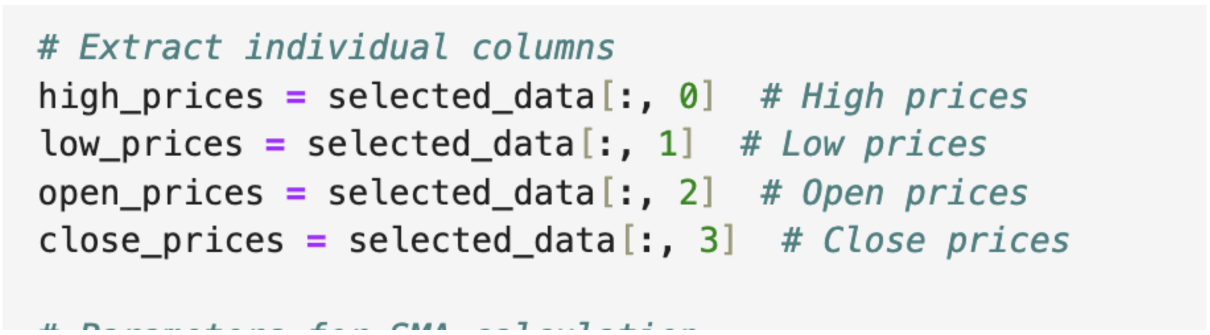
- **Columns Used:** Instead of just the 'Close' column, this version extracts data from multiple columns (High, Low, Open, Close, Volume, and Adj Close).



```
def trade_finance():
    ticker = 'AAPL'
    data = yf.download(ticker, period='1mo', interval='1h')  # Download 1 month of hourly data

    # Columns to use
    columns = ["High", "Low", "Open", "Close", "Volume", "Adj Close"]
    selected_data = data[columns].to_numpy()  # Convert selected columns to NumPy array
```

Image 12. Downloading Hourly data for AAPL using yfinance

- **Stop-Loss and Take-Profit Logic:** In this version, the **Low** price is used for triggering the stop-loss, while the **High** price is used for triggering the take-profit. This improves the accuracy of risk management as it accounts for intra-period price volatility.

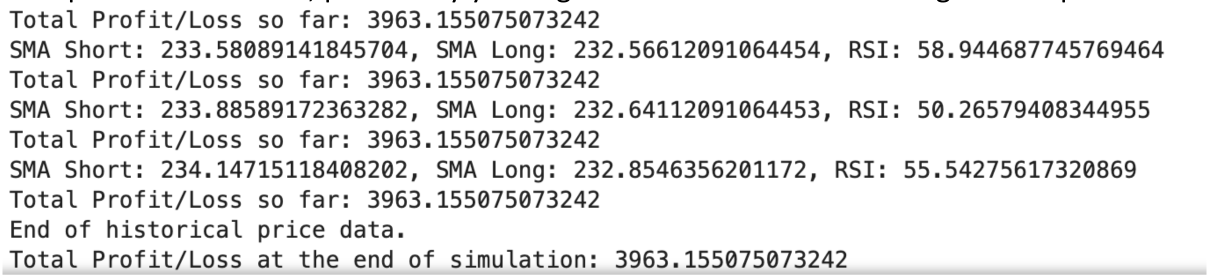


```python
# Extract individual columns
high_prices = selected_data[:, 0]  # High prices
low_prices = selected_data[:, 1]  # Low prices
open_prices = selected_data[:, 2]  # Open prices
close_prices = selected_data[:, 3]  # Close prices
```

Image 13. Extracting Individual Price Columns

- **Multiple Price Points for Decisions:** High, Low, and Close prices are all factored into buy/sell decisions, providing a more robust analysis of market movements.

By leveraging these additional data points, this approach allows the strategy to be more responsive to real-time market dynamics and improves the precision of the stop-loss and take-profit mechanisms, potentially yielding more accurate results during volatile periods.



```
Total Profit/Loss so far: 3963.155075073242
SMA Short: 233.58089141845704, SMA Long: 232.56612091064454, RSI: 58.944687745769464
Total Profit/Loss so far: 3963.155075073242
SMA Short: 233.88589172363282, SMA Long: 232.64112091064453, RSI: 50.26579408344955
Total Profit/Loss so far: 3963.155075073242
SMA Short: 234.14715118408202, SMA Long: 232.8546356201172, RSI: 55.54275617320869
Total Profit/Loss so far: 3963.155075073242
End of historical price data.
Total Profit/Loss at the end of simulation: 3963.155075073242
```

Image 14. Simulation output

Backtest Comparison and Results

The following comparison highlights the difference in total profit/loss between the two approaches:

1. **Using Only 'Close' Column:**
   - The final simulation results show a **Total Profit/Loss** of **695.54** at the end of the simulation.
   - The strategy was less responsive to intra-period price movements, focusing solely on the closing price to determine buy/sell signals, stop-loss, and take-profit levels.
2. **Using All Columns (High, Low, Open, Close):**
   - The final simulation results show a significantly higher **Total Profit/Loss** of **3963.16**.
   - By incorporating High and Low prices, the strategy was able to make more informed decisions, especially regarding stop-loss and take-profit, resulting in better profitability.

Github Repository

The project's code base is hosted on GitHub for version control and review. Everyone can access the repository via the following link: GitHub Repository. This repository contains all necessary files for the project and is available for the tutor to review the work.

References

https://corporatefinanceinstitute.com/resources/data-science/backtesting/#:~:text=Summary-,Backtesting%20involves%20applying%20a%20strategy%20or%20predictive%20model%20to%20historical,%2C%20market%20exposure%2C%20and%20volatility.

https://pypi.org/project/Backtesting/

https://pypi.org/project/yfinance/

https://www.investopedia.com/ask/answers/122214/what-are-main-differences-between-moving-average-convergence-divergence-macd-relative-strength-index.asp

https://www.investopedia.com/terms/s/stock-analysis.asp

https://www.youtube.com/watch?v=ErGU9sHO8wY

https://www.youtube.com/watch?v=f2gxz8JbrlQ