Internship Report - Backtesting Platform

1. Introduction

Backtesting is a vital process in algorithmic trading, where historical data is used to simulate the performance of a trading strategy. This project focuses on building a platform that enables users to input their trading strategies, test them against historical data, and analyze the results to determine the effectiveness of those strategies. The primary goal is to provide a web-based tool that can handle multiple strategies, fetch data efficiently, and provide detailed analysis.

2. Project Objectives

The project was designed with specific goals in mind:

- **Create a Web-Based Platform:** The aim was to develop an interactive web application where users can easily input parameters for their trading strategies. The platform needed to be user-friendly, enabling traders of all skill levels to use it effectively.
- **Implement Trading Strategies:** The platform had to support multiple popular trading strategies like Golden Cross, MACD. Each strategy required careful implementation to accurately replicate how it would perform in real-world scenarios.
- **Analyze Historical Data:** To ensure that the backtesting is realistic, the platform needed to fetch and process historical stock market data. This data serves as the basis for simulating the trades and analyzing the performance of the strategies.
- **Evaluate Strategy Performance:** After simulating the trades, the platform needed to calculate key performance metrics such as total return and annualized return. These metrics help users assess how well a strategy would have performed historically.
- **Store Results:** The platform needed to log the results of each backtest, enabling users to review their strategies, adjust parameters, and refine their approach.

3. Technologies and Libraries Used

The choice of technologies and libraries was crucial to meet the project objectives:

- **Flask:** Flask is a lightweight and flexible web framework in Python that is ideal for small to medium-sized projects. It provides the necessary tools for routing, template rendering, and handling HTTP requests and responses.
- **SQLite:** SQLite is a self-contained, serverless database engine. It was chosen for this project because it is easy to set up and provides sufficient functionality for storing the strategies and their corresponding parameters.
- **Pandas:** Pandas is a powerful data analysis and manipulation library. It is used to handle time series data, calculate moving averages, and perform other data-centric operations crucial to backtesting.

- **yFinance:** yFinance is a Python library that simplifies access to historical market data from Yahoo Finance. It was chosen for its ease of use and ability to fetch accurate and up-to-date financial data for various instruments.
- **NumPy:** NumPy is a fundamental package for scientific computing in Python. It provides support for arrays, matrices, and a collection of mathematical functions that are useful in handling large datasets and performing complex calculations.

4. Implementation Details

4.1 Flask Application Structure

The platform's architecture is centered around Flask, which organizes the application into different routes corresponding to various functionalities:

- **Home Route (`/`)**: This is the entry point of the application. It displays the home page, where users can select the trading strategy they want to test and enter the relevant parameters.
- **Strategy Routes**: Each trading strategy has its own route, such as `/golden_cross`, `/macd`, etc. When a user selects a strategy, they are redirected to the corresponding route, where the strategy-specific logic is executed. The results are then displayed on the same page.

Flask's template rendering system (using Jinja2) is employed to dynamically generate HTML pages based on user inputs and strategy results. The HTML templates are designed to be responsive and user-friendly.

4.2 Database Schema

The database (`strategies.db`) uses SQLite to store information about the strategies and their parameters. The schema was designed to ensure that all necessary details for each backtest are stored:

- **`strategy_name`**: The name of the strategy being tested.
- **`instrument_name`**: The financial instrument (e.g., stock ticker) on which the strategy is applied.
- **`start_date`** and **`end_date`**: The time period over which the backtest is conducted.
- **`time_frame`**: The frequency of the data used (e.g., daily, hourly).
- **`expiry`, `strike`, `strike_length`, `option_type`, `lot_size`**: These fields store options-specific data, relevant to strategies like Iron Condor.
- **`spread`, `stop_loss`, `book_profit`, `action`**: Parameters related to risk management and trade execution.

This database schema allows for comprehensive logging and easy retrieval of backtest results for review and analysis.

4.3 Data Fetching and Preparation

The platform uses `yfinance` to fetch historical market data. This data is essential for simulating trades. The process involves:

- **Fetching Data**: Using `yfinance`, historical price data for the selected instrument is downloaded. This includes open, high, low, close, and volume data.
- **Data Cleaning**: The data is cleaned to ensure there are no missing values, which could distort the backtesting results. If necessary, missing data points are filled using forward or backward filling methods.
- **Data Preparation**: The data is rounded to meaningful values (e.g., rounding stock prices to the nearest integer or the nearest 50, as applicable). The time series is also adjusted to match the specified timeframe (daily, hourly, etc.).

4.4 Trading Strategies Implemented

Each trading strategy has unique logic and conditions for generating buy/sell signals:

4.4.1 Golden Cross Strategy

- **Concept**: The Golden Cross strategy is a trend-following strategy that uses moving averages to identify potential buy and sell opportunities.
- **Implementation**: The strategy computes two moving averages, one short-term (e.g., 50-day) and one long-term (e.g., 200-day). When the short-term moving average crosses above the long-term average, it generates a buy signal. Conversely, when the short-term average crosses below the long-term average, it generates a sell signal.
- **Backtest Logic**: The backtesting process simulates trading based on these signals, iterating through the historical data and recording the entry and exit points, along with the profit or loss from each trade.

4.4.2 MACD Strategy

- **Concept**: The MACD (Moving Average Convergence Divergence) strategy is based on the difference between two exponential moving averages (EMAs) of different lengths.
- **Implementation**: The MACD is calculated by subtracting the long-term EMA from the short-term EMA. A signal line EMA of the MACD, known as the signal line, is also plotted to identify buy/sell signals. A buy signal is generated when the MACD crosses above the signal line, and a sell signal is generated when it crosses below.
- **Backtest Logic**: Similar to the Golden Cross strategy, the platform iterates through the historical data, applying the MACD and signal line to generate and execute trades.

4.5 Strategy Backtesting

The core of the platform is the backtesting engine, which executes trades based on the signals generated by the selected strategy. The backtesting process includes:

- **Daily Returns**: Calculating the daily return for each trading day in the backtesting period.
- **Strategy Return**: Measuring the return generated by the strategy over the entire backtesting period.
- **Cumulative Returns**: Calculating the cumulative return, which shows the growth of an initial investment over time, assuming all profits were reinvested.

The results are displayed on the web interface, providing users with a clear overview of how the strategy performed historically.

4.6 Result Logging

The results of the backtest are meticulously saved to an Excel file named `trades.xlsx`. This Excel file logs all the trades executed during the backtest, providing an in-depth record for further analysis. Key details captured include:

- **Trade Timestamps**: The exact date and time when each trade was executed, which is crucial for understanding the context and timing of each action.
- **Buy/Sell Prices**: The prices at which trades were made, allowing for an accurate calculation of profits or losses.
- **Trading Signals**: The indicators or triggers that prompted the buy or sell actions, helping users to understand the rationale behind each trade.
- **Performance Metrics**: Additional metrics like the profit or loss from each trade, the cumulative return, and other relevant financial ratios.

This comprehensive logging enables users to perform a detailed review of their strategies, refine their parameters, and improve future performance.

5. Results and Analysis

After running a backtest, the platform provides users with a detailed summary of the results, allowing for a thorough analysis of the strategy's performance. The key components of the results include:

- **Total Return**: The overall percentage return generated by the strategy during the backtest period. This metric provides a snapshot of the strategy's success.

- **Annualized Return**: This is the return adjusted to account for the number of trading days in a year (typically 252 days). It helps users understand how the strategy might perform on an annual basis, making it easier to compare with other strategies or benchmarks.

- **Buy/Sell Signals**: A detailed log of every trade executed by the strategy, including the dates and prices. This log helps users verify the accuracy of the strategy and gain insights into its trading behavior.

5.1 Example of Results Display (HTML)

The platform generates an HTML report that provides a clear and visually appealing presentation of the backtest results. This report includes:

- **Strategy Name and Parameters**: The report begins with a summary of the strategy tested and the parameters used during the backtest. This context is crucial for understanding the results.
- **Trade Log Table**: A table listing each trade signal, the corresponding date, the price at which the trade was executed, and any resulting profit or loss. This table offers a detailed view of the strategy's trading activity.

5.2 To run the project:

1) **Install required dependencies from requirements.txt file**

2) **Run on terminal - “python main.py”**

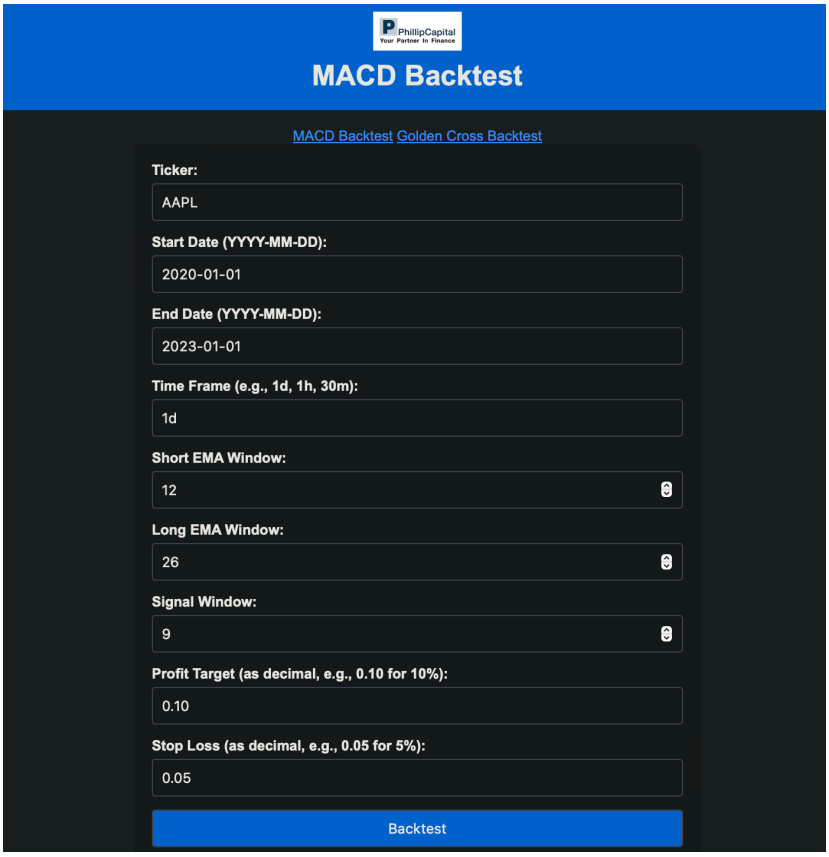
3) **Go to “http://localhost:8000/macd” to access MACD strategy or “http://localhost:8000/golden_cross” to access Golden Cross strategy**

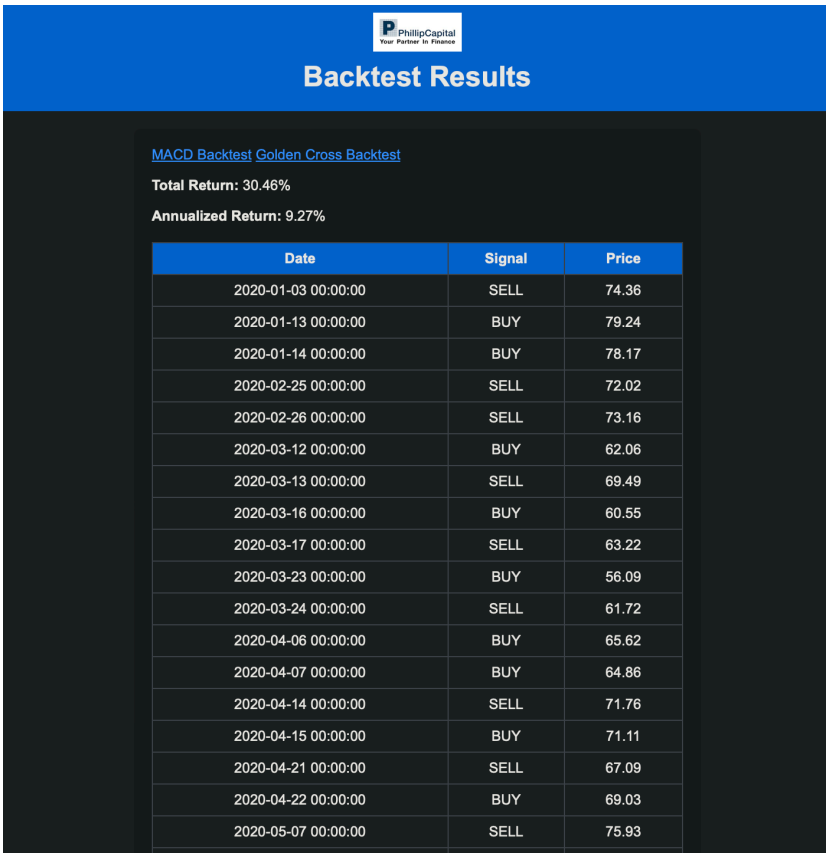
6. Challenges and Solutions

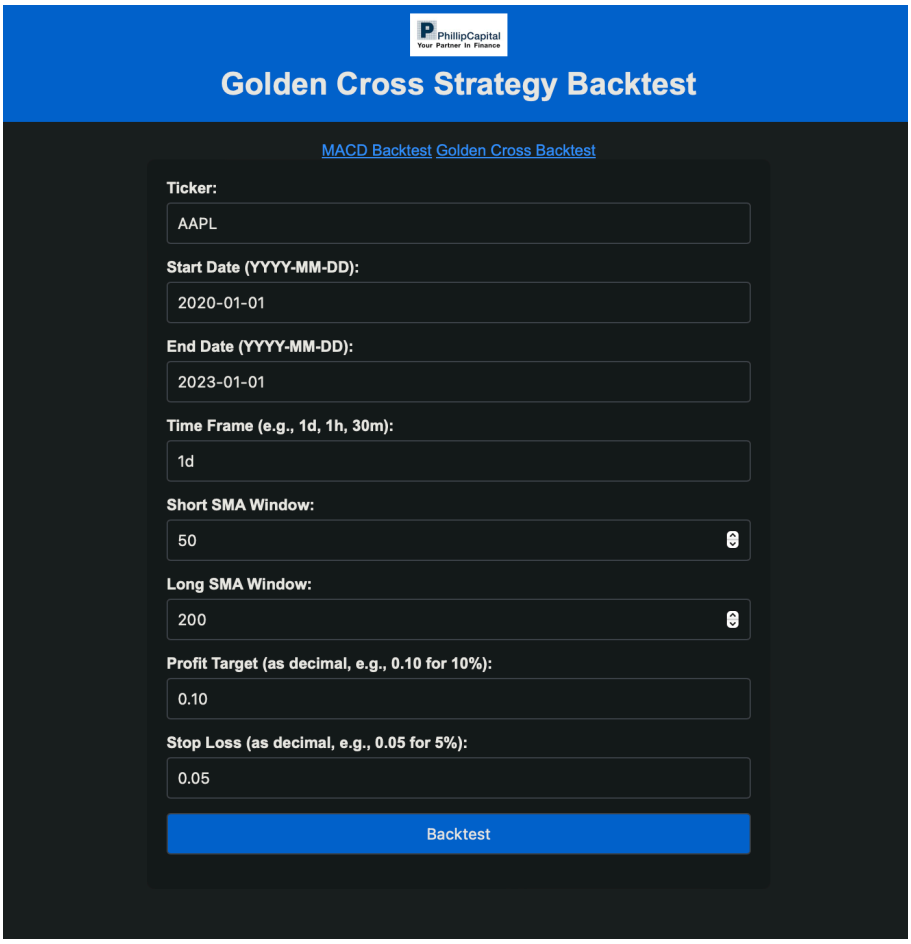
During the development and implementation of the platform, several challenges were encountered:

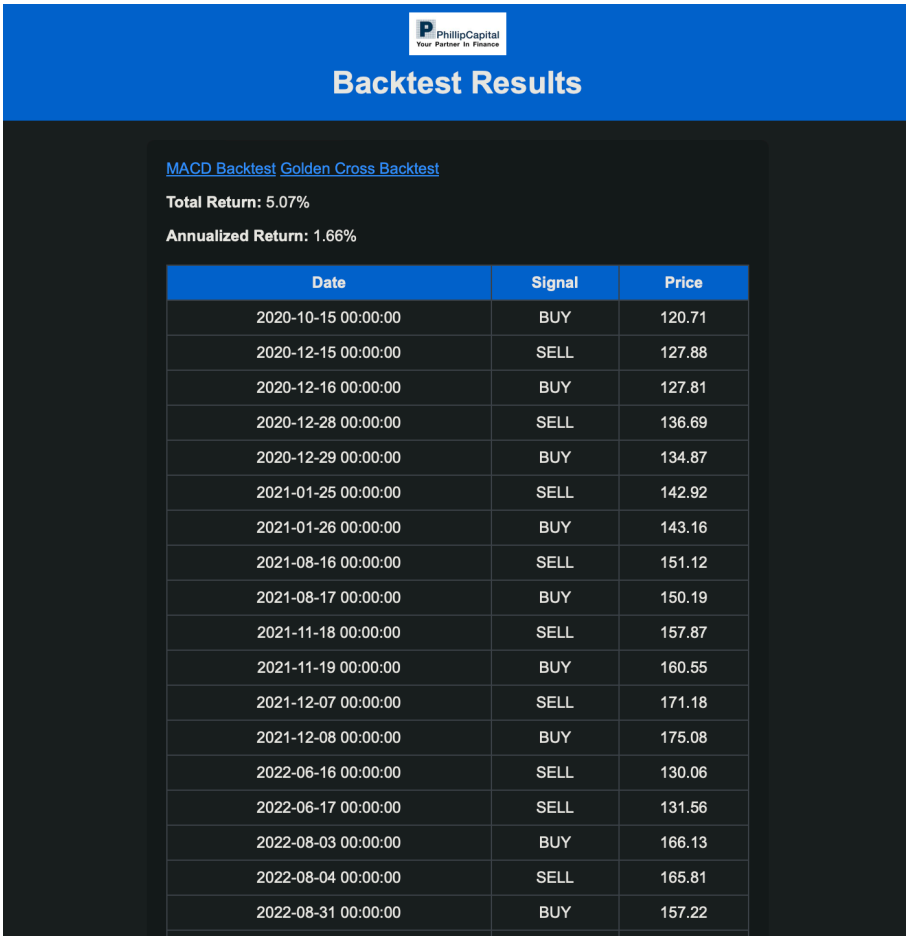
- **Data Integrity**: Ensuring that the historical data fetched from Yahoo Finance was clean and complete was a major challenge. This was addressed by implementing rigorous data validation checks and handling any missing values through imputation or exclusion, ensuring the backtest results were accurate.
- **Performance Optimization**: The backtest loop initially had performance bottlenecks due to redundant calculations and excessive data processing. These were resolved by optimizing the code, streamlining data handling processes, and reducing the amount of data processed in each iteration. This optimization significantly improved the platform's speed and responsiveness.

Project Results:









Date	Signal	Price
2020-10-15 00:00:00	BUY	120.71
2020-12-15 00:00:00	SELL	127.88
2020-12-16 00:00:00	BUY	127.81
2020-12-28 00:00:00	SELL	136.69
2020-12-29 00:00:00	BUY	134.87
2021-01-25 00:00:00	SELL	142.92
2021-01-26 00:00:00	BUY	143.16
2021-08-16 00:00:00	SELL	151.12
2021-08-17 00:00:00	BUY	150.19
2021-11-18 00:00:00	SELL	157.87
2021-11-19 00:00:00	BUY	160.55
2021-12-07 00:00:00	SELL	171.18
2021-12-08 00:00:00	BUY	175.08
2022-06-16 00:00:00	SELL	130.06
2022-06-17 00:00:00	SELL	131.56
2022-08-03 00:00:00	BUY	166.13
2022-08-04 00:00:00	SELL	165.81
2022-08-31 00:00:00	BUY	157.22

7. Conclusion and Future Work

The backtesting platform developed in this project offers a robust environment for simulating and evaluating various trading strategies. The current implementation successfully covers several widely-used strategies, providing valuable insights into their historical performance.

However, the platform has the potential for further enhancements:

- **Additional Strategies**: Expanding the platform to include more complex trading strategies, including those based on machine learning models, would broaden its applicability and provide users with more options for testing and analysis.
- **Real-Time Data Integration**: Incorporating real-time data would allow users to perform forward testing, where strategies are tested in live market conditions. This would add a new dimension to the platform, making it more relevant for active traders.
- **Portfolio Backtesting**: Extending the platform to support portfolio-level backtesting, where multiple assets are tested together, would enable users to evaluate the performance of diversified investment strategies. This would be particularly useful for portfolio managers and institutional investors.

8. References

The development of this platform was guided by several key resources:

- **Flask Documentation**: Provided essential information on building and deploying web applications using Flask.(https://flask.palletsprojects.com/en/3.0.x/)
- **Pandas Documentation**: Served as a reference for data manipulation and analysis, particularly in handling time series data.(https://pandas.pydata.org/docs/)
- **yFinance Library**: Used for fetching historical market data, with its documentation guiding the integration and use of its features.(https://pypi.org/project/yfinance/)