INF1002 Python Project Report

Stalking Stocks

Team ID: P1-08

Name	Student ID	Email
Mohammed Aamir	2500933	2500933@sit.singaporetech.edu.sg
Muhammad Hasif Bin Mohd Faisal	2500619	2500619@sit.singaporetech.edu.sg
Timothy Chia Kai Lun	2501530	2501530@sit.singaporetech.edu.sg
Low Gin Lim	2501267	2501267@sit.singaporetech.edu.sg
Dalton Chng Cheng Hao	2504003	2504003@sit.singaporetech.edu.sg

Course: INF1002 – Programming Fundamentals with Python

Institution: Singapore Institute of Technology

Date of Submission: 13th October, 2025

Abstract

Stalking Stocks addresses the need for beginner-friendly, interpretable stock analysis. It implements a compact, transparent pipeline that applies four core functions to validated OHLCV data: (1) Simple Moving Average, (2) Trend Runs, (3) Daily Returns, and (4) Maximum Profit and present outputs in a Streamlit dashboard. The system separates data handling from computation and visualisation, enabling clear explanations of each step. Using representative tickers across major sectors, the dashboard overlays moving averages, marks streaks on price charts, and provides candlestick and sector views to support guided exploration. In our evaluations, the tools consistently picked out clear patterns, such as prolonged trend runs in growth-oriented sectors and higher single-window profit opportunities in more volatile names. These results suggest that a small set of manual, explainable functions can make market structure legible for first-time learners without requiring prior financial expertise.

1. Introduction

The accessibility of open-source libraries and financial data platforms has made stock market analysis increasingly approachable. Within a Python programming module, this project applies key programming principles such as modular design, data management, and collaborative development to a practical, real-world problem. The resulting system, *Stalking Stocks*, was created to make financial analysis more approachable for beginners through an interactive dashboard that encourages learning by exploration.

1.1 Problem Statement & Objectives

For new investors, understanding stock market behaviour can be challenging because of the large amount of data available and the technical nature of most analytical tools. *Stalking Stocks* addresses this challenge by transforming raw financial data into clear and interactive visual insights that promote intuitive learning.

The project aims to create a beginner-friendly environment where users can explore market trends, interpret financial indicators, and gain confidence in analysing stock movements without financial risk. To achieve this, the system includes four analytical functions: the Simple Moving Average (SMA), Trend Runs, Daily Returns, and Maximum Profit. These functions are applied to live market data retrieved from Yahoo Finance and visualised through a Streamlit interface that enables comparison and exploration across sectors. The system is developed in Python using libraries such as *pandas* and *yfinance*. It follows a modular structure that separates data handling, computation, and presentation to ensure clarity, maintainability, and scalability.

1.2 Scope

Stalking Stocks focuses on developing a web-based application that performs essential technical analyses and presents sector-level insights through an interactive dashboard. It is designed for educational and exploratory purposes rather than professional trading.

The system analyses data from sectors including, but are not limited to, technology, finance, healthcare, consumer goods, and energy. Since users may be unfamiliar with ticker symbols, the application provides a curated list of representative companies within each sector, as defined by Yahoo Finance's categorisation. This approach supports a guided and simplified exploration experience while ensuring that the dataset remains relevant and efficient to process. More advanced analytical methods such as predictive modelling and machine learning are intentionally excluded to maintain the project's focus on transparency, interpretability, and educational clarity.

2. Related Works / Literature Review

Several platforms currently provide stock analysis and visualisation tools, yet each targets different audiences and presents challenges for beginners.

StockAnalysis.com offers free access to financial data such as income statements, ratios, and analyst insights. However, its interface prioritises completeness over usability, meaning that the large number of metrics may overwhelm new users. Similarly, Yahoo Finance consolidates financial news, stock quotes, and performance indicators but assumes users can interpret advanced measures such as price-to-earnings (P/E) ratios, beta coefficients, and earnings per share. This creates a steep learning curve that often discourages beginners.

TradingView delivers professional-grade charting with custom technical indicators and advanced candlestick visualisations. Although highly capable, its terminology and feature depth cater primarily to experienced traders. The interface's complexity and scripting environment (Pine Script) can be intimidating to users seeking understanding rather than execution.

In contrast, Simply Wall St was designed to simplify investing through visual reports and infographics. Its "snowflake" charts translate company fundamentals into intuitive graphics, making data interpretation easier. Yet, its visuals are sometimes superficial, focusing on aesthetics rather than hands-on learning or analysis depth.

Beyond commercial tools, algorithmic approaches such as the Simple Moving Average (SMA) and trend analysis (Hayes, 2025a; 2025b) provide structured methods for studying market patterns. Similarly, the maximum profit algorithm (AlgoMonster, n.d.) illustrates how computational logic can identify optimal buy–sell intervals, although such reasoning is rarely made transparent in public-facing tools.

Stalking Stocks bridges these gaps by combining intuitive visualisation with algorithmic transparency in an open-source, beginner-oriented environment. Unlike StockAnalysis.com or Yahoo Finance, it narrows its dataset to key sectors and essential indicators to reduce cognitive overload. Compared to TradingView, it prioritises conceptual clarity rather than technical customisation, and unlike Simply Wall St, it aims for genuine analytical understanding rather than aesthetic simplification. By integrating SMA, trend analysis, and maximum profit computations, *Stalking Stocks* enables users not only to view data but also to comprehend the analytical processes behind it, promoting confidence and understanding before engaging with real markets.

3. Methodology

3.1 Design Philosophy

We designed Stalking Stocks around Separation of Concerns and abstraction: distinct packages (models, services, UI, utilities, constants) each own a single, well-defined behaviour. The UI never fetches data; services never plot; and an adapter that translates models to UI formats so internal schema changes stay isolated.

We apply Single Responsibility rigorously. Data models validate and coerce data; services perform domain computations; UI modules handle rendering and interaction; utilities provide narrow helpers making each component easier to reason about and test.

Guided by the Don't Repeat Yourself (DRY) principle, shared logic, constants, and mappings are centralised in utilities and common modules. Domain constants are maintained in a single location, and shared test fixtures prevent duplication. Together, these practices keep the codebase clean, robust, and easy to extend.

3.2 Dataset

Market data is obtained dynamically from *Yahoo Finance* through the *yfinance* library, accessed via the service layer in `src/services/finance.py`. The application retrieves daily; Open, High, Low, Close, Adjusted Close, and Volume (OHLCV) data for user-selected symbols and outputs structured tables for analysis. Within the current scope, the interface processes one ticker at a time, although the service is designed to accept multiple symbols for future development. User selections for time horizon and sampling intervals are passed directly to `yfinance.download()` as keyword arguments, ensuring that the dashboard reflects the chosen parameters. These retrieval steps are combined into a single helper function for streamlined access (see Appendix A, Figure A.3).

3.2.1 Cleaning

Data pre-processing is intentionally minimal and easy to interpret, as implemented in `src/services/data.py`. `fill_gaps()` performs a forward fill followed by a backward fill to address short gaps, ensuring continuity during temporary API interruptions. No interpolation beyond this is applied. Trading-day integrity is maintained by filtering weekends to prevent unintended rows caused by external joins, while daily *Yahoo Finance* data already exclude exchange holidays. Potential outliers are detected but not removed, using Tukey's interquartile range (IQR) rule. A Boolean mask is exposed to the user interface, allowing users to include or exclude flagged points in their analyses while keeping raw values visible. This approach stabilises indicators such as the Simple Moving Average, streaks, daily returns, and maximum profit while maintaining full transparency. Examples of the cleaning process are shown in Appendix A, Figure A.3 (GME dataset) and Appendix A, Figure A.4 (IQR mask on volume, k = 1.5).

3.2.2 Sectors & Industries

Sector and industry information is standardised using custom data models rather than direct API payloads. The function `get_sectors()` provides a fixed sector list stored as a constant in the project codebase, ensuring consistent naming across the interface without additional network calls. When live details are required, such as a sector's industries or company profile, functions including `get_sector_info()`, `get_industry_info()`, and `get_ticker_info()` query *yfinance* and map the results to the project's internal models. This design maintains a stable taxonomy while still allowing retrieval of up-to-date market information when needed. Since many users may be unfamiliar with ticker symbols, the system provides a curated list of top-performing or representative companies within each sector to enable simple and guided exploration.

3.2.3 Table shape & Validation

Because *yfinance* returns different table structures for single and multi-ticker requests, table schemas are standardised and validated before analysis. For single-ticker inputs, OHLCV data are stored in a flat DataFrame, and rows missing key fields are dropped to prevent charting errors. Multi-ticker inputs, which are mainly used for testing or planned future features, are reformatted into a long-form table with an additional Ticker column for clarity. Validation is performed using Pydantic models for metadata (`src/models/base.py`) and schema-based checks for price tables (`src/schemas/dataframe.py`). These measures ensure schema integrity, prevent downstream errors, and simplify debugging when API structures change. This dataset design ensures that the data remain live through dynamic retrieval, predictable through standardised and validated structures, and transparent through light cleaning and user-controlled outlier masking. It supports the project's educational focus while providing a solid foundation for future extensions, including additional indicators, longer time horizons, and multi-ticker analysis.

3.3 System Architecture

Our application adopts a three-layer architecture with distinct roles: Data → Core Algorithms (Domain) → Streamlit UI (Visualisation) with `app.py` as the entry point. Dependencies flow strictly downward, keeping data and business logic decoupled from presentation (see Appendix A, Figure A.2). At the foundation, the Data layer acquires and prepares market data. The module `src/services/finance.py` is the sole interaction point with the external *yfinance* API, encapsulating necessary provider-specific behavior. Raw responses are subsequently checked and cleaned by `src/services/data.py`, which provides post-processing utilities to resolve common *`yfinance`* irregularities (such as missing values, type inconsistencies, and index gaps). Together, these components establish a stable and validated input surface for the remainder of the system.

After data access, the Domain layer (`src/services/core.py`) encapsulates the application's financial logic. It operates on the cleaned datasets supplied by `finance.py` and cleaned by `data.py`, implementing time-series computations while remaining independent of both the user interface and external I/O. This separation keeps the domain code deterministic and straightforward to unit-test and it ensures that changes in data providers or presentation do not propagate the computation pipeline.

The high-level dependency graph (see Appendix A, Figure A.2) illustrates only the primary imports governing the project's core logic. The structure depicts a clear layered flow with one-way dependencies: Data → Domain → Visualisation. In the Data layer, `finance.py` is the single entry for external communication with *yfinance* and `data.py` cleans the retrieved data. The Domain layer (`core.py`) encapsulates financial computations and business rules, assuming cleaned data. The Visualisation layer is split by role: `ui.adapters.py` orchestrates the end-to-end flow of: select ticker → fetch & clean → compute algorithms → shape UI-ready structures. `ui.overview.py` wires controls to views and `ui.charts.py` renders figures only (no data fetching). By funnelling all services through `src/ui/adapters.py` before presentation, the design enforces separation of concerns, reduces coupling, and preserves clear testing boundaries.

3.4 Core Functions

Stalking Stocks implements four core computational algorithms using native Python to ensure transparency, reproducibility, and educational value. *Pandas* (Pandas Development Team, 2024) and *NumPy* (Harris et al., 2020) were used only for data handling, while all numerical computations were implemented manually rather than through vectorised methods. The four primary functions, Simple Moving Average, Trend Runs, Daily Returns, and Maximum Profit, form the analytical foundation of the system. Each function was designed for efficiency and interpretability and includes safeguards for conditions such as empty inputs or invalid parameters. A comparison with equivalent module-based implementations was also conducted to evaluate performance and structural differences.

A detailed summary of these computational functions, including their file locations, signatures, and concise analytical roles, is presented in Appendix B, Table B.1. This table serves as a quick technical reference for understanding how each core algorithm contributes to data processing and analysis in *Stalking Stocks*.

3.4.1 Simple Moving Average (SMA)

The SMA is defined as the arithmetic mean of the last N closing prices:

$$\mathrm{SMA}_t = \frac{P_t + P_{t-1} + \cdots + P_{t-N+1}}{N}$$

Typically using 5, 20 or 50-day windows that correspond to standard trading conventions (Hayes, 2025b). These windows correspond to standard trading conventions, enabling meaningful capture of market trends across different time horizons and aligning the analysis with commonly accepted practices.

The Simple Moving Average was implemented using a sliding window algorithm that maintains a running sum to avoid repeated calculations. Each new price is added to the total, and once the window reaches size N, the oldest value is subtracted as the window moves forward. This approach enables linear-time computation since each observation is processed once. Observations with fewer than N data points are assigned null values to preserve alignment, and input validation ensures stability by handling empty datasets or invalid parameters such as non-positive window sizes. For example, with $N = 5$ and price data, the SMA at the fifth data point is:

$$\mathrm{SMA}_4 = \frac{10 + 12 + 11 + 13 + 14}{5} = 12$$

When the next price $P_5 = 15$ arrives, the sliding window moves forward, updating the sum by adding 15 and subtracting 10, resulting in:

$$\text{SMA}_5 = \frac{(10+12+11+13+14)-10+15}{5} = 12.8$$

The algorithm operates in linear time, $O(n)$, since every data point is processed once, and uses proportional space, $O(n)$, as the output series matches the input length. Only a few supporting variables, such as the running total and current index, are retained throughout execution, ensuring efficiency and clarity. The resulting SMA values and their visual output are presented in Appendix B, Figures B.1–B.3, where the smoothed trend line is compared with the original closing prices. A performance comparison with module-based approaches using *NumPy*'s `np.convolve` and `np.correlate` showed that these vectorised methods are faster (approximately 0.0005 seconds per 100 iterations) due to C-level optimisation. However, the manually implemented version (approximately 0.008 seconds) provides greater transparency and educational value by clearly demonstrating the underlying logic of the calculation.

3.4.2 Trend Runs (Streaks)

The Trend Runs function identifies consecutive sequences of price movements:

$$S_t = \begin{cases} +1, & \text{if } \Delta P_t > 0, \\ -1, & \text{if } \Delta P_t < 0, \\ 0, & \text{if } \Delta P_t = 0, \end{cases}$$

Where ($\Delta P_t = P_t - P_{t=1}$) is the price difference between times t and $t - 1$. A run R_i is a maximal consecutive subsequence of identical nonzero signs in the sequence S_t :

$$R_i = \{S_k, S_{k+1}, \ldots, S_{k+m} \mid S_j = S_k, S_j \neq 0, \quad \forall j = k, \ldots, k+m\},$$

where the length of the run is given by : $|R_i| = m + 1.$ The formulation follows the Wald and Wolfowitz Runs Test (1940), applying its principles to financial time-series data to measure trend persistence. By classifying daily changes as upward, downward or flat, the function captures continuous streaks of similar movements. The resulting sequence S_t is used to construct a trend mask for visualising market phases, as shown in Appendix B, Figures B.4 and B.5.

TThe algorithm performs a single traversal of the entire price series, resulting in a linear time complexity of $O(n)$. Each counter and mask update is processed in constant time, while auxiliary storage remains minimal, $O(1)$. The direction mask itself scales linearly with the number of observations because it forms part of the intended output rather than additional memory usage. Edge cases, such as datasets with one or zero valid observations, are handled safely by returning neutral results, ensuring robustness across all dataset sizes.A comparative test using *Pandas* and *NumPy* methods was conducted for reference. Directional changes were calculated using `df["Close"].diff()` and classified with `np.sign()`. While the vectorised approach offers higher computational efficiency, it abstracts the intermediate logic that the manual implementation exposes. The manual Python function therefore provides a clearer educational demonstration of how trend streaks are computed and tracked step by step.

3.4.3 Daily Returns

Following the convention used in `pandas.Series.pct_change()` (Pandas Documentation, 2024), the daily return r_t is defined as:

$$r_t = \frac{P_t - P_{t-1}}{P_{t-1}} = \frac{P_t}{P_{t-1}} - 1$$

where P_t and P_{t-1} denote the closing prices on consecutive trading days t and $t - 1$. This formulation ensures comparability between assets regardless of price level and allows consistent evaluation of proportional changes over time.

The function was implemented as a single-pass algorithm that iterates through the price series, computing returns for each valid observation. The first value is set to null since no prior price exists for comparison, and division-by-zero cases return None when the previous closing price equals zero. Implementing the procedure manually enhances transparency by exposing each computation step and verifying edge-case handling. The output is returned as a Pandas Series aligned with the original index.

The computation operates in linear time, $O(n)$, and uses proportional space, $O(n)$, since each input is processed once and mirrored in the output. Only a few supporting variables are maintained, ensuring efficiency and scalability. The resulting return series and its descriptive statistics are presented in Appendix B, Figures B.6–B.8. A comparative version using the `pandas.Series.pct_change()` method achieves faster execution due to vectorised processing and automatic index alignment. However, the manual implementation preserves pedagogical value by exposing the underlying arithmetic and providing clearer insight into how daily returns are computed.

3.4.4 Maximum Profit

The algorithm was conceptually inspired by the *Best Time to Buy and Sell Stock II* problem (LeetCode, 2024), which presents the idea of capturing all profitable price movements. In this project, the implementation was independently developed in Python using a greedy accumulation approach that sums all positive day-to-day price differences. The logic assumes that each local price increase contributes to the total achievable profit. Whenever $P_t > P_{t-1}$, the difference is added to the cumulative total, representing the maximum potential profit achievable through unconstrained sequential trading.

Formally, the total profit Π can be expressed as the sum of all positive price differences between consecutive trading days:

$$\Pi = \sum_{t=1}^{n-1} \max(0, P_t - P_{t-1})$$

where P_t and P_{t-1} represent closing prices on consecutive days t and $t - 1$. For example, with prices $[10, 12, 11, 13, 14, 12]$ the algorithm captures each gain as, $(12 - 10) + (13 - 11) + (14 - 13) = 2 + 2 + 1 = 5$, yielding a total profit of five units. The algorithm iterates once through the entire price series, comparing each pair of consecutive observations. When the next price exceeds the current one, their difference is added to a cumulative profit total. This single-pass structure ensures that each observation is processed exactly once, avoiding redundant rescans and the need for auxiliary data structures.

The method operates in linear time $O(n)$ and constant auxiliary space $O(1)$, since only a small number of variables, including the current price and cumulative profit, are maintained. This efficient design enables the computation to scale effectively to large datasets without increased memory usage or computational overhead. Although the function models an ideal scenario, it excludes transaction costs, slippage, and liquidity constraints. The resulting value should therefore be interpreted as a theoretical maximum rather than a realistic measure of attainable returns. The resulting cumulative profit and its visual representation are illustrated in Appendix B, Figures B.9–B.11. A comparative version using *NumPy* and *Pandas* calculated positive price differences through vectorised functions such as `np.diff()` and `np.maximum()`. These methods improved performance but concealed the explicit stepwise trading logic, making the manual algorithm preferable for transparency and instructional clarity.

3.5 Visualisation Layer (Streamlit)

The visualisation layer serves as the presentation component of *Stalking Stocks*, responsible for transforming processed financial data into clear, interactive visuals. It was developed using *Streamlit*, a Python framework that enables the rapid creation of web-based interfaces directly from scripts. This layer functions as the user's main point of interaction, connecting the analytical modules to intuitive, data-driven insights.

The dashboard prioritises simplicity, speed, and interpretability. Through the *Streamlit* interface, users can explore sectors, select tickers, and view multiple visualisation types. These include a line chart displaying price trends, a trend-marked chart indicating upward and downward runs, and a candlestick chart showing open, high, low, and close prices (see Appendix D, Figures D.2–D.4). Each visualisation updates automatically in response to user selections, allowing comparisons across different indicators or sectors.

The interface also includes a sector overview panel that summarises industries within the selected sector (see Appendix D, Figure D.1). It displays daily changes, industry weights, market capitalisation, and the number of listed companies. Streamlit layout elements such as columns and containers ensure that information is presented clearly and in an organised manner. Technically, the visualisation layer integrates with the analytical and data-handling modules contained in `core.py`, `finance.py`, and `data.py`. When a user makes a selection, *Streamlit* retrieves and processes data through these modules to compute indicators such as the Simple Moving Average (SMA), daily returns, trend runs, and maximum profit. The processed outputs are visualised through helper functions from `ui/charts.py`, including `set_linechart()`, `set_candlechart()`, and `set_treemap()`, which generate corresponding *Plotly* figures.

This layered architecture ensures modularity: the visualisation layer manages presentation logic only, while analytical and data-processing tasks remain independent. By combining *Streamlit* interactivity with reusable chart functions, the system provides an accessible and educational experience that supports the project's objective of helping beginners understand financial data through exploration and visual interpretation.

3.6 Testing & Validation

This project uses *pytest* as the main testing framework because it is lightweight, readable, and follows Python's natural syntax. Compared with traditional frameworks such as *unittest*, *pytest* simplifies test discovery, fixture creation, and debugging.

3.6.1 Unit Testing

Unit tests were designed to confirm that each computation and data-handling component behaved correctly before integration.

Test File	Functions Tested	Purpose / Focus	Expected Outcome
`test_core.py`	`compute_sma()`, `compute_streak()`, `compute_sdr()`, `compute_max_profit()`	Verify mathematical consistency of SMA, streak, and profit calculations against manual and pandas references	Outputs align with trusted libraries; handles edge datasets gracefully
`test_finance.py`	`get_ticker_info()`, `get_ticker_data()`, `get_sector_data()`, `get_industry_info()`	Validate retrieval and transformation of financial data and metadata across tickers	Correctly structured DataFrames and metadata; schema integrity maintained
`test_models.py`	`Ticker` model validation	Confirm *Pydantic* schema raises errors for invalid data and accepts valid fields	`ValidationError` raised for invalid types; passes for valid data
`conftest.py`	Dataset and sector fixtures	Provide consistent mock data for tests	Fixtures load successfully and remain immutable during execution
`test_edge_cases.py`	Edge conditions for `compute_sma()` and `validate_ticker()` using `xfail` / `xpass`	Demonstrate controlled handling of known failures and success cases	`xfail` tests fail safely without breaking suite; `xpass` confirms resolved conditions

Table 3.1 *Summary table for unit tests, function and uses*

3.6.2 Integration and System Testing

Integration testing ensured that validated modules operated smoothly as a cohesive system. The simulated workflow began with data retrieval through `get_ticker_data()`, followed by analytical processing using `compute_sma()` and `compute_streak()`, and schema validation via `src/models/base.py` before final display in `src/ui/dashboard.py`. This verified that each transformation preserved data structure and numerical integrity when passed between services and user-interface components.

Automated testing is integrated through GitHub Actions, using the workflow file `tests.yml` located in `.github/workflows/`, a screenshot of the workflow file is shown in Appendix C, Figure C.3. Each time a commit or pull request is made, the pipeline automatically sets up the Python environment, installs dependencies from `requirements.txt`, and executes `python -m pytest`. Results are reported directly in the repository's Actions tab. This integration not only maintained consistency between local and remote testing environments but also provided immediate visibility of code health to all team members. Screenshot of the workflow is shown in Appendix C, Figure C.4.

3.6.3 Validation Results and Analysis

Validation compared algorithmic outputs against reliable references to confirm analytical accuracy. The results of `compute_sma()` matched `pandas.Series.rolling().mean()` outputs precisely, while streak counts from `compute_streak()` were verified manually using small datasets. Model tests confirmed that invalid fields raised the appropriate exceptions, demonstrating the system's defensive programming design.

The inclusion of `xfail` and `xpass` markers provided additional evidence of robustness, showing that even deliberate error scenarios were managed without crashes. Edge cases such as invalid data formats, empty inputs, or oversized computation windows were all handled gracefully, preventing unhandled exceptions. The single `xpass` case highlighted a resolved limitation that had previously been expected to fail, underscoring ongoing improvement in the implementation.

4. Results and Insights

The results evaluate the analytical performance and educational impact of the *Stalking Stocks* system. Quantitative outputs from the four analytical functions, the Simple Moving Average (SMA), Trend Runs, Daily Returns, and Maximum Profit, are presented alongside qualitative findings from a user study. Together, they demonstrate the system's effectiveness in helping beginner users interpret stock market behaviour through structured and transparent visual analysis.

4.1 Output Results

Each analytical module operated correctly, producing clear and interpretable outputs consistent with expected reference values.

The Simple Moving Average (SMA) generated precise trend overlays, as shown in Appendix D, Figure D.1. The shorter SMA(5) reflected short-term price variation, while the longer SMA(20) revealed broader market direction. Crossover points between the two averages aligned with observed shifts in trend, confirming the accuracy of the implementation. The Trend Runs function, presented in Appendix D, Figure D.2, detected consecutive sequences of upward and downward price movement. Green and red markers represented the direction of each run, and the longest upward and downward streaks were correctly identified, demonstrating consistency and robustness in tracking momentum. The Daily Returns, also shown in Appendix D, Figure D.3, calculated the percentage change between consecutive trading days. These results served as the foundation for subsequent trend and profit analyses, confirming the reliability of the underlying data-handling process. The Maximum Profit output, illustrated in Appendix D, Figure D.4 and D.5, identified the most profitable buy and sell interval within the one-year dataset for NVIDIA. The function validated the theoretical model of maximum attainable profit and confirmed its precision across the examined data range.

4.2 User Study and Interpretation

A user study of ten participants aged 21–27 from diverse academic and professional backgrounds evaluated *Stalking Stocks* as a learning tool for understanding financial indicators and market behaviour. Feedback was largely positive, with participants finding the interface intuitive and the visual explanations clear and engaging. The Simple Moving Average and Maximum Profit functions were rated as the easiest to interpret, while Trend Runs and Daily Returns occasionally required clarification. Users appreciated the clarity of charts and smooth navigation but suggested adding clearer date markers, additional indicators, and improved chart scaling. Overall, the findings indicate that Stalking Stocks effectively supports financial learning through visual exploration and enhances user comprehension of key analytical concepts, aligning with the project's educational objectives.

	Age	Background	Familiarity	SMA	Trend Runs	Daily Returns	Max Profit	Navigation	Chart Clarity
P01	27	Information Security	None	Significant	Significant	Significant	Significant	Easy	Good
P02	25	Business	Intermediate	Moderate	Slight	None	Slight	Neutral	Poor
P03	22	Accountancy	Intermediate	Significant	Significant	Significant	Significant	Easy	Good
P04	22	Finance	Intermediate	Significant	Significant	Slight	Moderate	Very Easy	Excellent
P05	21	Computer Science	Intermediate	Significant	Moderate	Slight	Moderate	Neutral	Good
P06	23	Student	Beginner	Significant	Moderate	Significant	Significant	Easy	Good
P07	24	Social Work	Beginner	Moderate	Significant	Significant	Significant	Easy	Good
P08	24	Information Technology	Beginner	Moderate	Slight	Moderate	Significant	Easy	Good
P09	22	Civil Engineering	None	Significant	Moderate	Significant	Significant	Very Easy	Excellent
P10	24	Civil Engineering	Beginner	Slight	Significant	Significant	Significant	Easy	Good

Table E.1 presents participants' understanding ratings for the four core indicators.

4.3 Discussion

The findings confirm that all analytical modules performed accurately and integrated smoothly within the Streamlit interface. Visual components such as SMA overlays, streak indicators, and candlestick charts responded dynamically to user input, demonstrating efficient real-time functionality. Cross-referencing with Yahoo Finance data validated the accuracy of computed results, confirming the reliability of the implemented algorithms. Overall, *Stalking Stocks* effectively transformed quantitative financial data into clear and interpretable visual outputs, achieving its objective of making market analysis accessible to beginner users.

5. Conclusion

The Stalking Stocks project aimed to help beginners understand stock market behaviour through a simple, educational dashboard built in Streamlit. The system successfully implemented four analytical modules — Simple Moving Average, Trend Runs, Daily Returns, and Maximum Profit — all written in pure Python for transparency and instructional clarity. Each function performed correctly, producing accurate and interpretable results that aligned with theoretical expectations. User feedback confirmed that the interface was clean, intuitive, and easy to navigate, demonstrating that the project achieved its core goal of making financial analysis accessible for novice users.

5.1 Limitations

The application was designed primarily for learning, not trading. It operates using historical end-of-day data and excludes factors such as transaction costs, slippage, and liquidity, meaning the Maximum Profit function represents a theoretical upper limit. The system currently supports single-ticker analysis and a limited set of indicators, focusing on clarity rather than analytical depth. Feedback from the user study highlighted minor usability gaps, such as unclear date ranges, limited timeframes, and occasional slow loading. These issues, while not critical, indicate areas for refinement to enhance user experience and engagement.

5.2 Future Works

Future enhancements will focus on improving analytical scope, usability, and visual clarity, guided directly by user feedback:

Expanded indicators: Introduce additional metrics such as RSI, MACD, and Bollinger Bands. One participant noted, *"Include better trend indicators like moving average... and more timeframes for investors to have more data."* **Improved timeframe visualisation:** Add clear start–end markers on graphs and monthly separators to improve readability, as suggested by users who wanted *"indicators to show when months begin and end."* **Navigation and layout:** Simplify the sector selection process and prioritise the chart display for faster access, responding to feedback on *"getting what I want in the least number of clicks."* **Performance and clarity:** Implement more responsive chart loading indicators and optimise dark mode readability. **Enhanced Maximum Profit visualisation:** Add buy/sell markers and brief explanatory captions to clarify how the *"best trading sequence"* is defined.

In summary, Stalking Stocks delivered a functional and beginner-friendly educational tool that converts complex price data into clear, meaningful insights. With targeted enhancements in interactivity, performance, and analytical flexibility, *Stalking Stocks* can evolve into a more comprehensive learning platform for financial literacy and applied market analysis.

References

122. *Best Time to Buy and Sell Stock II.* (n.d.). AlgoMonster. https://algo.monster/liteproblems/122

Free portfolio tracker, stock insights and community - simply Wall St. (n.d.). https://simplywall.st/

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). *Array programming with NumPy. Nature, 585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hayes, A. (2025, April 24). *Trend analysis & trading strategies: Predict market movements. Investopedia*; Dotdash Meredith. https://www.investopedia.com/terms/t/trendanalysis.asp

Hayes, A. (2025b, August 5). *Simple moving average (SMA) explained: Definition and calculation formula. Investopedia*; Dotdash Meredith. https://www.investopedia.com/terms/s/sma.asp

LeetCode. (2024). *Best time to buy and sell stock II.* Retrieved 10 October 2025, from https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/

Pandas Development Team. (2024). *pandas: Powerful Python data analysis toolkit* (Version 2.2.2). *Zenodo.* https://doi.org/10.5281/zenodo.3509134

Stock Analysis. (n.d.). *Stock Analysis - Free online stock information for investors. StockAnalysis.* https://stockanalysis.com/

Streamlit Inc. (2024). *Streamlit documentation.* Retrieved 10 October 2025, from https://docs.streamlit.io

TradingView. (n.d.). *TradingView — Track all markets.* https://www.tradingview.com/

Wald, A., & Wolfowitz, J. (1940). *On a test whether two samples are from the same population. Annals of Mathematical Statistics, 11*(2), 147–162. https://doi.org/10.1214/aoms/1177731909

Yahoo! Finance. (n.d.). *Yahoo! Finance.* https://finance.yahoo.com

Appendix A – Datasets Cleaning

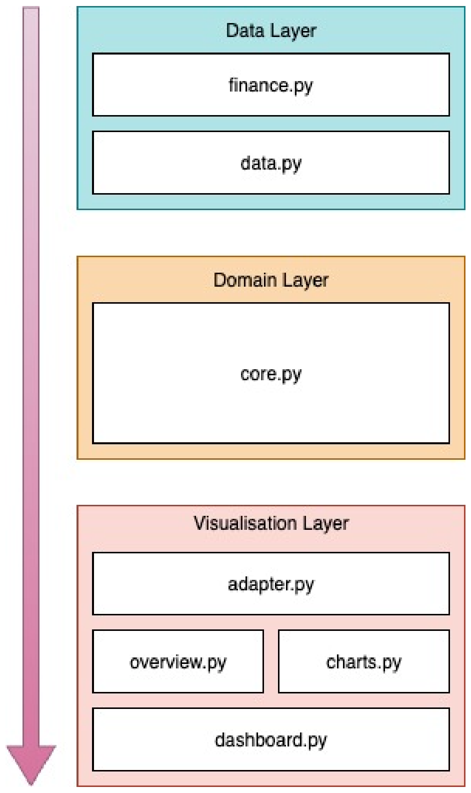


Figure A.1 *Architectural layers and data flow*

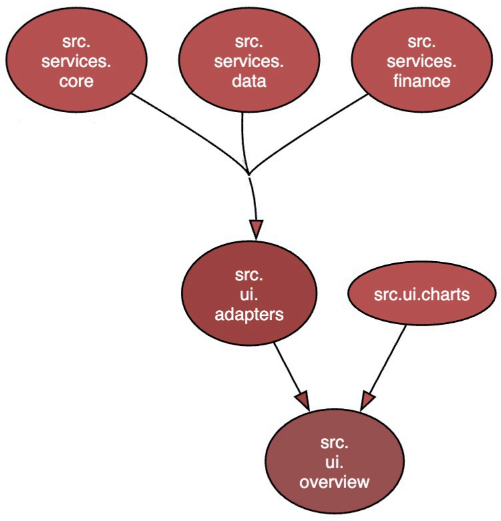


Figure A.2 *Core logic dependency graph (generated with pydeps). Arrows indicate import direction. Ancillary and utility imports are omitted for clarity.*

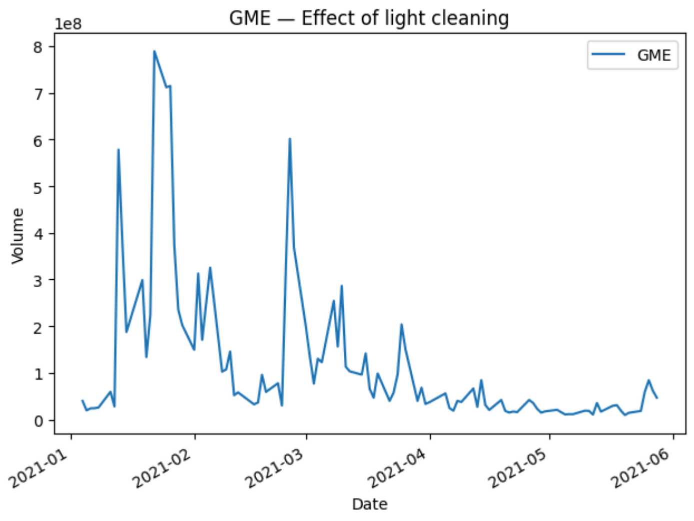


Figure A.3

Effect of light cleaning on GME volume data, showing continuity restored through forward- and backward-fill without altering overall trends.

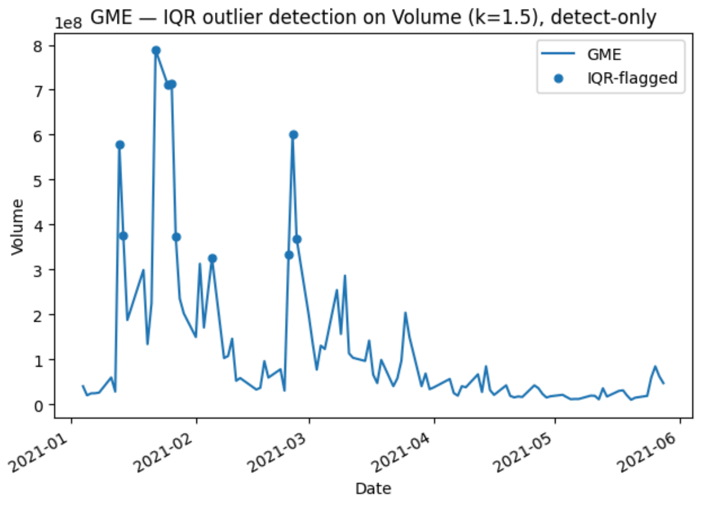
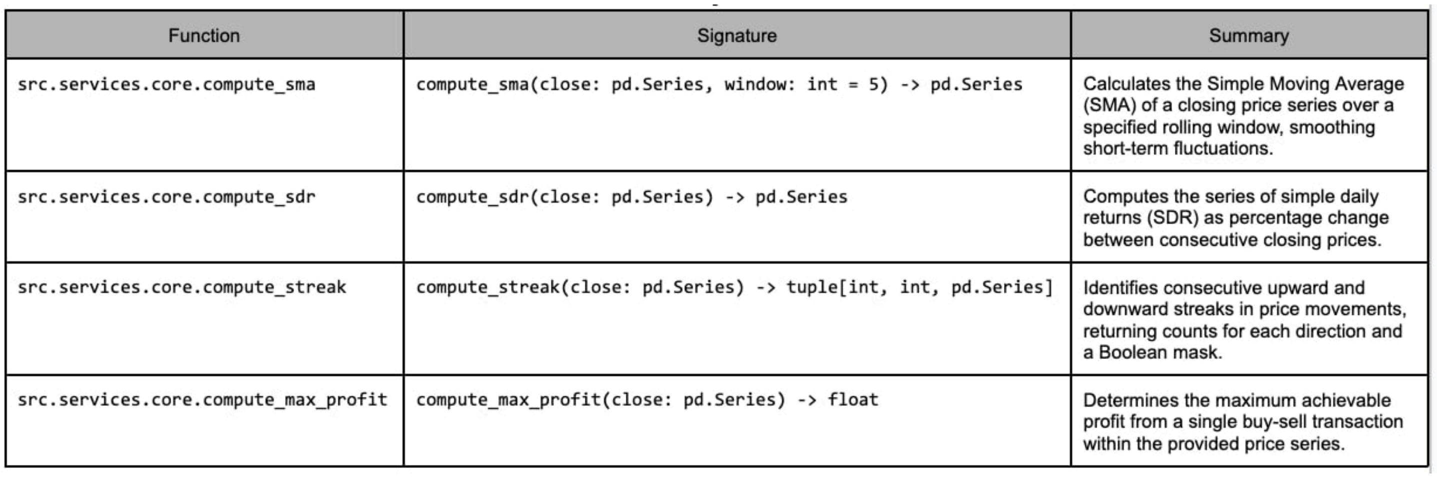


Figure A.4

IQR-based outlier detection (k = 1.5) on GME volume, highlighting flagged anomalies while preserving raw data transparency.

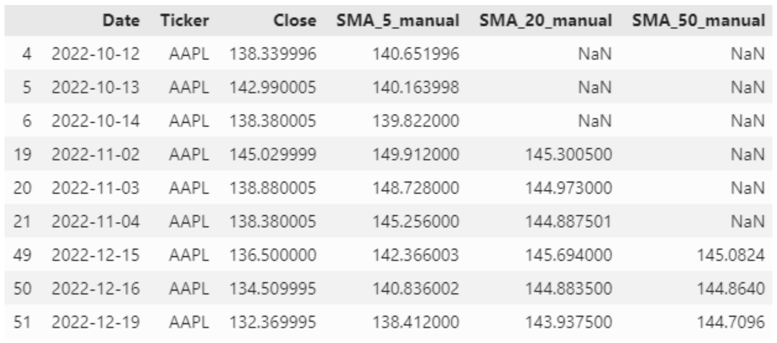
Appendix B – Core Computational Functions



Function	Signature	Summary
`src.services.core.compute_sma`	`compute_sma(close: pd.Series, window: int = 5) -> pd.Series`	Calculates the Simple Moving Average (SMA) of a closing price series over a specified rolling window, smoothing short-term fluctuations.
`src.services.core.compute_sdr`	`compute_sdr(close: pd.Series) -> pd.Series`	Computes the series of simple daily returns (SDR) as percentage change between consecutive closing prices.
`src.services.core.compute_streak`	`compute_streak(close: pd.Series) -> tuple[int, int, pd.Series]`	Identifies consecutive upward and downward streaks in price movements, returning counts for each direction and a Boolean mask.
`src.services.core.compute_max_profit`	`compute_max_profit(close: pd.Series) -> float`	Determines the maximum achievable profit from a single buy-sell transaction within the provided price series.

Table B.1

summarises the function signatures and their analytical roles within the system.



	Date	Ticker	Close	SMA_5_manual	SMA_20_manual	SMA_50_manual
4	2022-10-12	AAPL	138.339996	140.651996	NaN	NaN
5	2022-10-13	AAPL	142.990005	140.163998	NaN	NaN
6	2022-10-14	AAPL	138.380005	139.822000	NaN	NaN
19	2022-11-02	AAPL	145.029999	149.912000	145.300500	NaN
20	2022-11-03	AAPL	138.880005	148.728000	144.973000	NaN
21	2022-11-04	AAPL	138.380005	145.256000	144.887501	NaN
49	2022-12-15	AAPL	136.500000	142.366003	145.694000	145.0824
50	2022-12-16	AAPL	134.509995	140.836002	144.883500	144.8640
51	2022-12-19	AAPL	132.369995	138.412000	143.937500	144.7096

Figure B.1

displays the tabular output showing SMA values aligned with closing prices

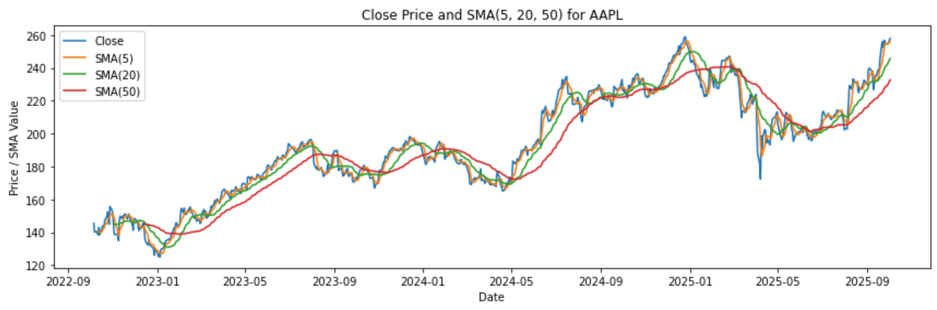


Figure B.2

visualises the corresponding price chart with the SMA overlays

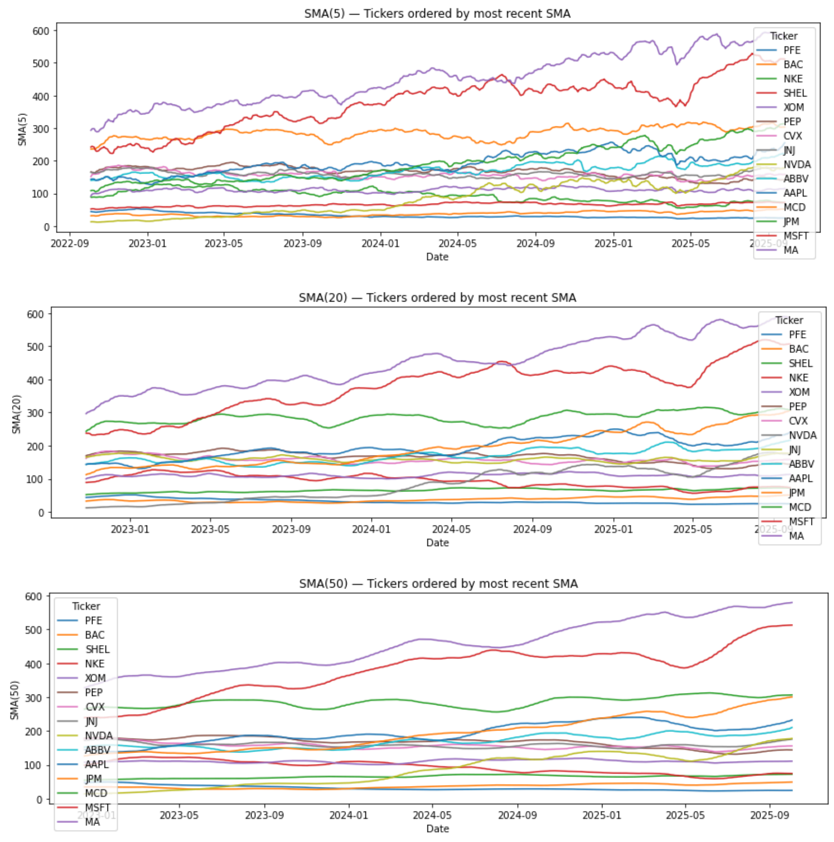
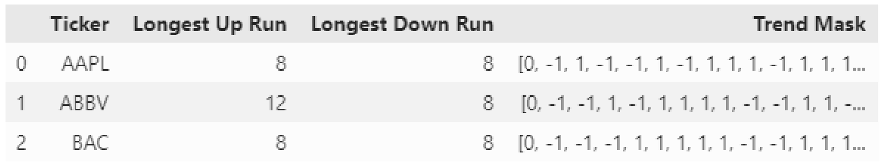


Figure B.3

visualises these aggregated SMA trends, highlighting how longer windows reduce volatility and reveal broader market direction across sectors.



	Ticker	Longest Up Run	Longest Down Run	Trend Mask
0	AAPL	8	8	[0, -1, 1, -1, -1, 1, -1, 1, 1, 1, -1, 1, 1, 1...
1	ABBV	12	8	[0, -1, -1, 1, -1, 1, 1, 1, 1, -1, -1, 1, 1, -...
2	BAC	8	8	[0, -1, -1, -1, 1, 1, 1, 1, 1, -1, -1, 1, 1, 1...

Figure B.4

shows tabular output listing the longest upward and downward runs per ticker

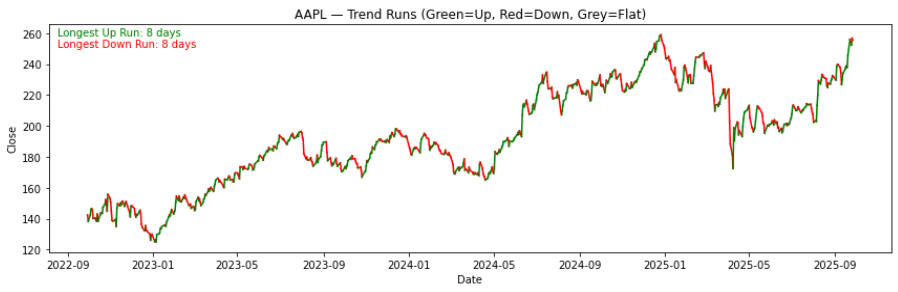
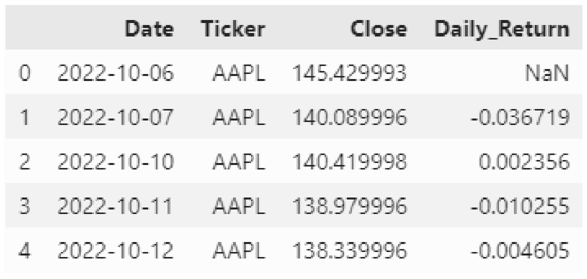


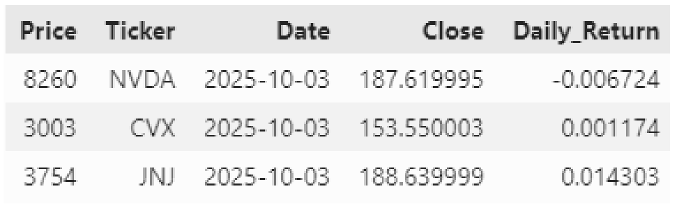
Figure B.5

depicts the corresponding price chart with trend segments highlighted to indicate sustained upward or downward momentum



	Date	Ticker	Close	Daily_Return
0	2022-10-06	AAPL	145.429993	NaN
1	2022-10-07	AAPL	140.089996	-0.036719
2	2022-10-10	AAPL	140.419998	0.002356
3	2022-10-11	AAPL	138.979996	-0.010255
4	2022-10-12	AAPL	138.339996	-0.004605

Figure B.6



Price	Ticker	Date	Close	Daily_Return
8260	NVDA	2025-10-03	187.619995	-0.006724
3003	CVX	2025-10-03	153.550003	0.001174
3754	JNJ	2025-10-03	188.639999	0.014303

Figure B.7

Figures B.6 and B.7 display the calculated daily returns for individual and multiple tickers, respectively.

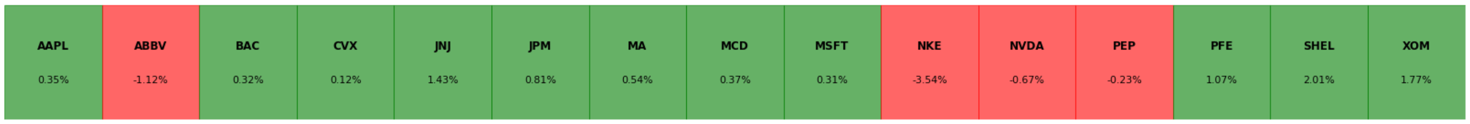
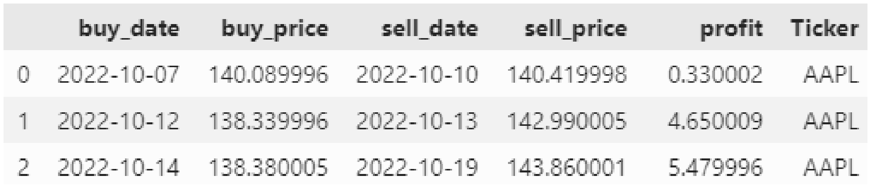


Figure B.8

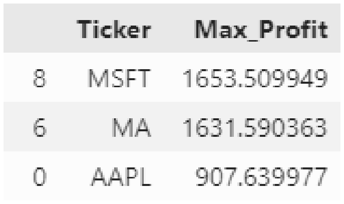
visualises sector-wide performance through a gridmap, providing a snapshot of relative daily gains and losses across tickers.



	buy_date	buy_price	sell_date	sell_price	profit	Ticker
0	2022-10-07	140.089996	2022-10-10	140.419998	0.330002	AAPL
1	2022-10-12	138.339996	2022-10-13	142.990005	4.650009	AAPL
2	2022-10-14	138.380005	2022-10-19	143.860001	5.479996	AAPL

Figure B.9

shows the detailed transaction-level results of the Maximum Profit computation, including optimal buy–sell pairs and associated gains



	Ticker	Max_Profit
8	MSFT	1653.509949
6	MA	1631.590363
0	AAPL	907.639977

Figure B.10

summarises the overall maximum profit achievable for each ticker.

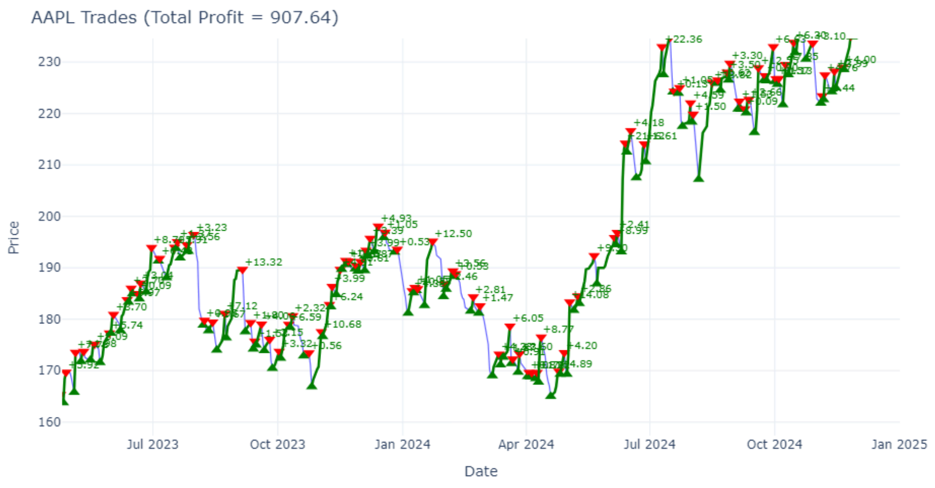
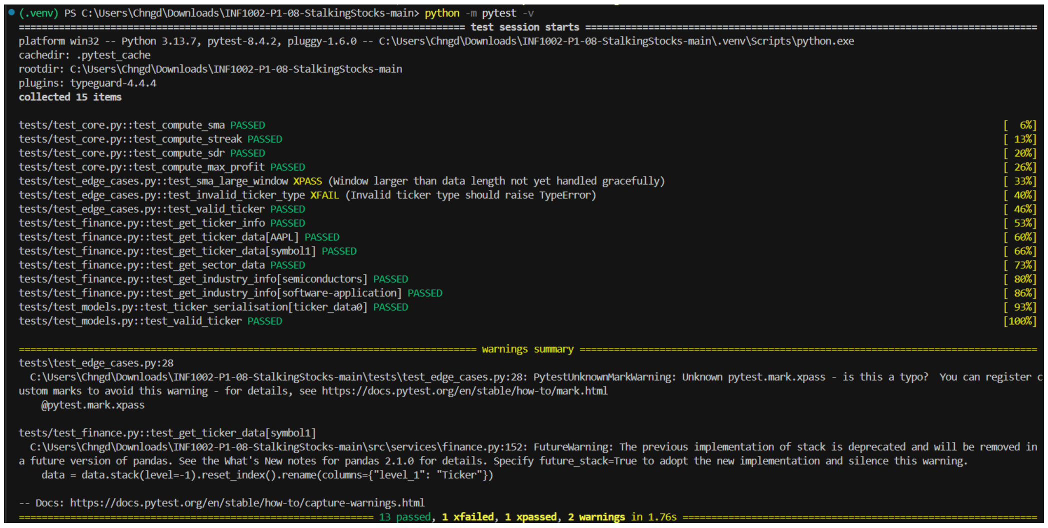


Figure B.11

visualises these profit intervals on a price chart, highlighting theoretical buy and sell markers that represent the most profitable trading sequence over the selected period.

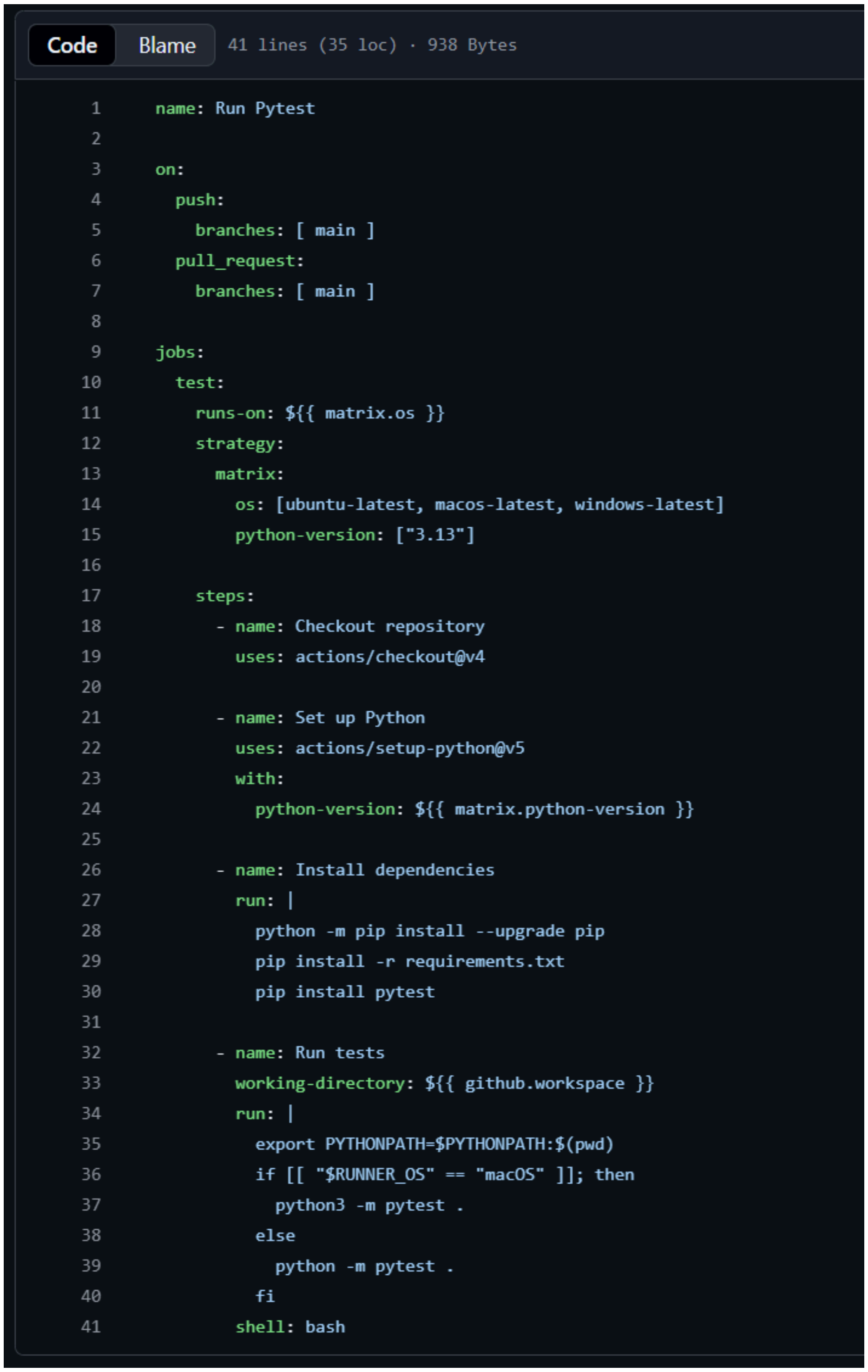
Appendix C – Testing Summaries



```
(.venv) PS C:\Users\Chngd\Downloads\INF1002-P1-08-StalkingStocks-main> python -m pytest -v
======================================================================== test session starts ========================================================================
platform win32 -- Python 3.13.7, pytest-8.4.2, pluggy-1.6.0 -- C:\Users\Chngd\Downloads\INF1002-P1-08-StalkingStocks-main\.venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\Chngd\Downloads\INF1002-P1-08-StalkingStocks-main
plugins: typeguard-4.4.4
collected 15 items

tests/test_core.py::test_compute_sma PASSED                                                                                                                   [  6%]
tests/test_core.py::test_compute_streak PASSED                                                                                                                [ 13%]
tests/test_core.py::test_compute_sdr PASSED                                                                                                                   [ 20%]
tests/test_core.py::test_compute_max_profit PASSED                                                                                                            [ 26%]
tests/test_edge_cases.py::test_sma_large_window XPASS (Window larger than data length not yet handled gracefully)                                             [ 33%]
tests/test_edge_cases.py::test_invalid_ticker_type XFAIL (Invalid ticker type should raise TypeError)                                                         [ 40%]
tests/test_edge_cases.py::test_valid_ticker PASSED                                                                                                            [ 46%]
tests/test_finance.py::test_get_ticker_info PASSED                                                                                                            [ 53%]
tests/test_finance.py::test_get_ticker_data[AAPL] PASSED                                                                                                      [ 60%]
tests/test_finance.py::test_get_ticker_data[symbol1] PASSED                                                                                                   [ 66%]
tests/test_finance.py::test_get_sector_data PASSED                                                                                                            [ 73%]
tests/test_finance.py::test_get_industry_info[semiconductors] PASSED                                                                                          [ 80%]
tests/test_finance.py::test_get_industry_info[software-application] PASSED                                                                                    [ 86%]
tests/test_models.py::test_ticker_serialisation[ticker_data0] PASSED                                                                                          [ 93%]
tests/test_models.py::test_valid_ticker PASSED                                                                                                                [100%]

========================================================================= warnings summary ==========================================================================
tests\test_edge_cases.py:28
  C:\Users\Chngd\Downloads\INF1002-P1-08-StalkingStocks-main\tests\test_edge_cases.py:28: PytestUnknownMarkWarning: Unknown pytest.mark.xpass - is this a typo?  You can register c
ustom marks to avoid this warning - for details, see https://docs.pytest.org/en/stable/how-to/mark.html
    @pytest.mark.xpass

tests/test_finance.py::test_get_ticker_data[symbol1]
  C:\Users\Chngd\Downloads\INF1002-P1-08-StalkingStocks-main\src\services\finance.py:152: FutureWarning: The previous implementation of stack is deprecated and will be removed in
a future version of pandas. See the What's New notes for pandas 2.1.0 for details. Specify future_stack=True to adopt the new implementation and silence this warning.
    data = data.stack(level=-1).reset_index().rename(columns={"level_1": "Ticker"})

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
============================================================== 13 passed, 1 xfailed, 1 xpassed, 2 warnings in 1.76s ==============================================================
```

Figure C.1

Output of pytest showing all 15 tests running successfully, including 1 expected fail and 1 expected pass scenarion

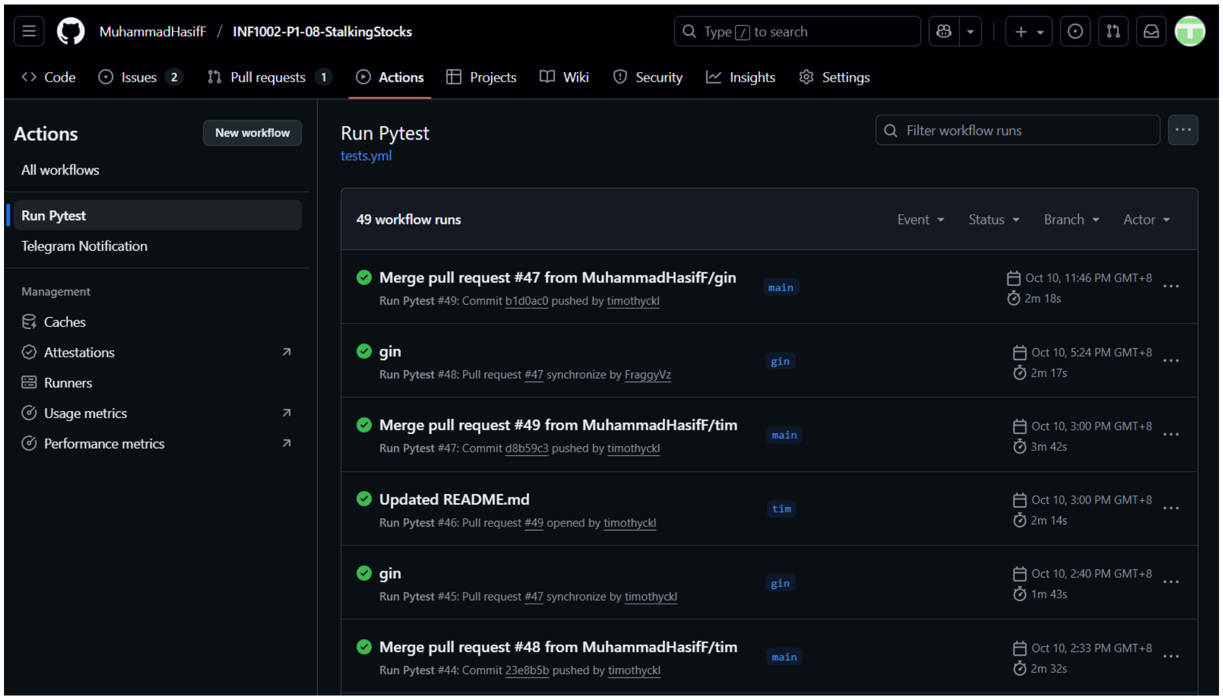


41 lines (35 loc) · 938 Bytes

```yaml
name: Run Pytest

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  test:
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, macos-latest, windows-latest]
        python-version: ["3.13"]

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: ${{ matrix.python-version }}

      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt
          pip install pytest

      - name: Run tests
        working-directory: ${{ github.workspace }}
        run: |
          export PYTHONPATH=$PYTHONPATH:$(pwd)
          if [[ "$RUNNER_OS" == "macOS" ]]; then
            python3 -m pytest .
          else
            python -m pytest .
          fi
        shell: bash
```

Figure C.3

Workflow of test.yml used for GitHub Actions


MuhammadHasifF / INF1002-P1-08-StalkingStocks

Code | Issues 2 | Pull requests 1 | Actions | Projects | Wiki | Security | Insights | Settings

Actions — New workflow

- All workflows
- Run Pytest
- Telegram Notification

Management

- Caches
- Attestations
- Runners
- Usage metrics
- Performance metrics

Run Pytest
tests.yml

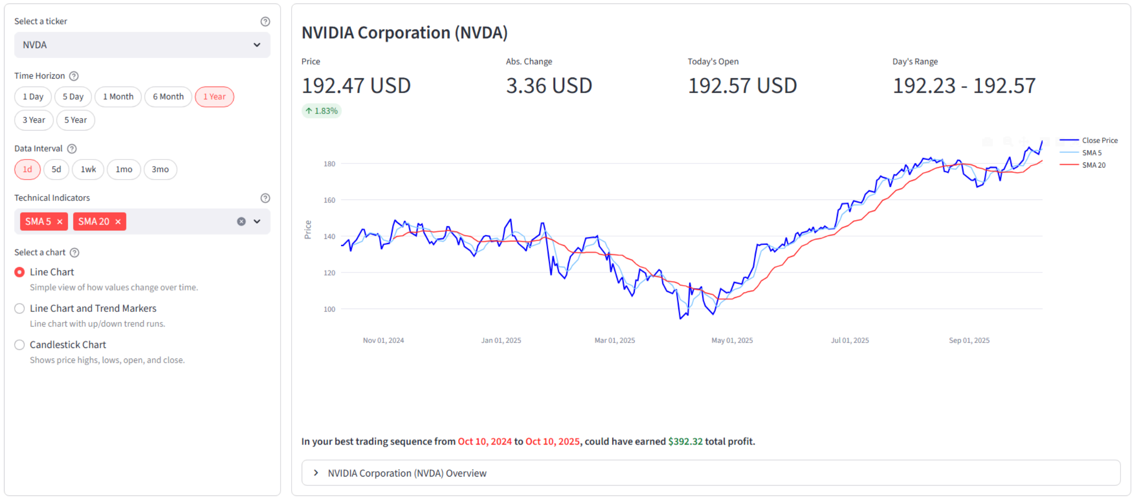
49 workflow runs

Merge pull request #47 from MuhammadHasifF/gin Run Pytest #49: Commit b1d0ac0 pushed by timothyckl	main	Oct 10, 11:46 PM GMT+8	2m 18s
gin Run Pytest #48: Pull request #47 synchronize by FraggyVz	gin	Oct 10, 5:24 PM GMT+8	2m 17s
Merge pull request #49 from MuhammadHasifF/tim Run Pytest #47: Commit d8b59c3 pushed by timothyckl	main	Oct 10, 3:00 PM GMT+8	3m 42s
Updated README.md Run Pytest #46: Pull request #49 opened by timothyckl	tim	Oct 10, 3:00 PM GMT+8	2m 14s
gin Run Pytest #45: Pull request #47 synchronize by timothyckl	gin	Oct 10, 2:40 PM GMT+8	1m 43s
Merge pull request #48 from MuhammadHasifF/tim Run Pytest #44: Commit 23e8b5b pushed by timothyckl	main	Oct 10, 2:33 PM GMT+8	2m 32s

Figure C.4

Screenshot of the actions tab in GitHub showing test.yml successfully running on every push/pull

Appendix D - Visualisations Layer


Select a ticker: NVDA

Time Horizon: 1 Day | 5 Day | 1 Month | 6 Month | [✓ 1 Year] | 3 Year | 5 Year

Data Interval: [✓ 1d] | 5d | 1wk | 1mo | 3mo

Technical Indicators: SMA 5, SMA 20

Select a chart:
- [✓ Line Chart] — Simple view of how values change over time.
- Line Chart and Trend Markers — Line chart with up/down trend runs.
- Candlestick Chart — Shows price highs, lows, open, and close.

NVIDIA Corporation (NVDA)

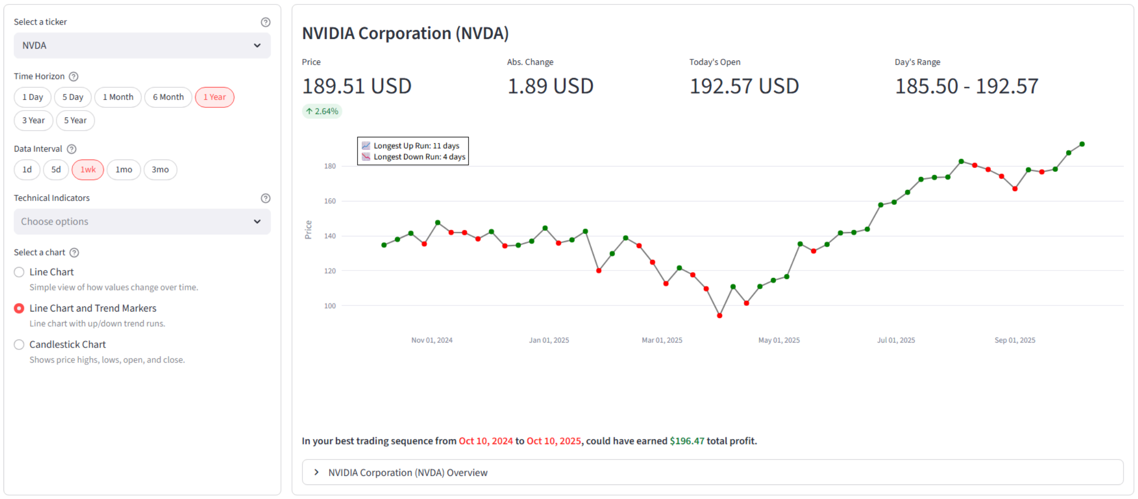
Price	Abs. Change	Today's Open	Day's Range
192.47 USD ↑ 1.83%	3.36 USD	192.57 USD	192.23 - 192.57

In your best trading sequence from Oct 10, 2024 to Oct 10, 2025, could have earned $392.32 total profit.

NVIDIA Corporation (NVDA) Overview

Figure D.1: SMA chart (1-year, 1-day)

SMA chart showing one year of daily prices with SMA(5) and SMA(20) overlays; the short window tracks fluctuations closely while the long window smooths broader trends.



Select a ticker

NVDA

Time Horizon: 1 Day | 5 Day | 1 Month | 6 Month | [✓ 1 Year] | 3 Year | 5 Year

Data Interval: 1d | 5d | [✓ 1wk] | 1mo | 3mo

Technical Indicators

Choose options

Select a chart

Line Chart
Simple view of how values change over time.

[✓ Line Chart and Trend Markers]
Line chart with up/down trend runs.

Candlestick Chart
Shows price highs, lows, open, and close.

NVIDIA Corporation (NVDA)

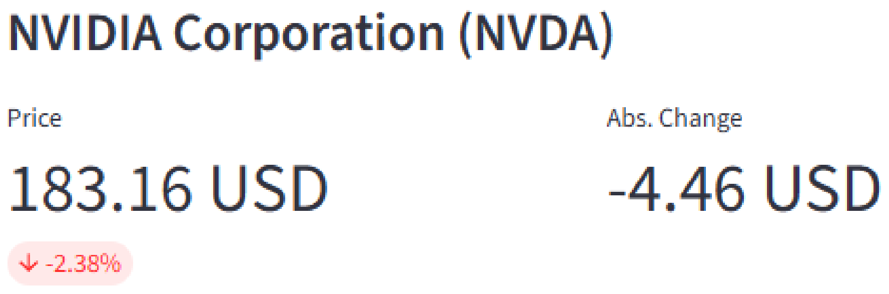
Price	Abs. Change	Today's Open	Day's Range
189.51 USD	1.89 USD	192.57 USD	185.50 - 192.57
↑ 2.64%			

In your best trading sequence from Oct 10, 2024 to Oct 10, 2025, could have earned $196.47 total profit.

› NVIDIA Corporation (NVDA) Overview

Figure D.2

Line chart with up/down trends (1-year, 1-week), Trend-marked line chart where green segments denote upward streaks and red denote downward streaks; legend shows the longest up/down runs.

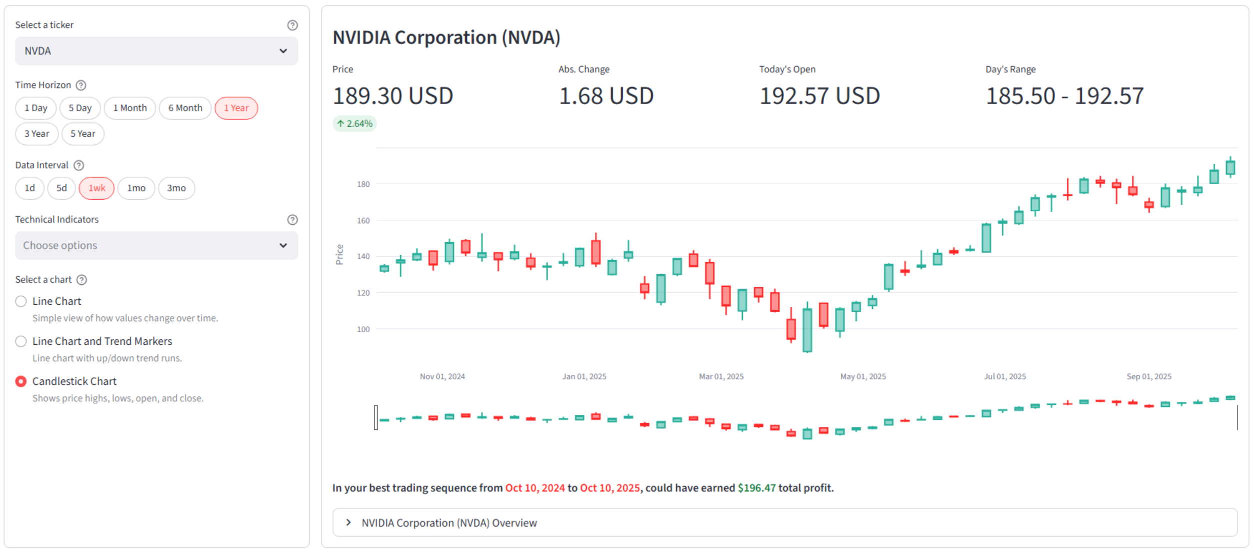


NVIDIA Corporation (NVDA)

Price	Abs. Change
183.16 USD	-4.46 USD
↓ -2.38%	

Figure D.3

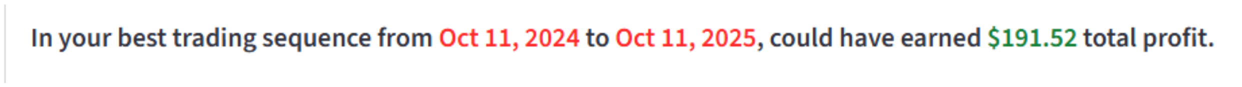
Daily Returns (1-year, 1-week), Daily stock return measures a stock's day-to-day performance



In your best trading sequence from Oct 10, 2024 to Oct 10, 2025, could have earned $196.47 total profit.

Figure D.4

Candlestick Chart (1-year, 1-week), Candlestick chart for the selected horizon with the best buy–sell sequence summary displayed for the period.



In your best trading sequence from Oct 11, 2024 to Oct 11, 2025, could have earned $191.52 total profit.

Figure D.5

Max Profit Output (1-year, 1-week), best buy–sell sequence summary displayed for the period.

Appendix E - Reflections

Muhammad Hasif

Time management was one of the most important and ongoing lessons throughout this project. From the beginning, I approached my responsibilities by dividing the work into manageable phases: research and planning, core feature development, integration and validation, and finally documentation and refinement. Since I was responsible for designing and implementing the analytical core, I began development early to allow enough time for verification and troubleshooting before integration. Alongside the technical workload, I also took the initiative to coordinate meetings, facilitate communication, and ensure progress remained steady across team members. Balancing the technical and organisational aspects was demanding, but it developed my ability to prioritise effectively and to stay adaptable when schedules shifted.

In retrospect, I learned that time management in a team environment extends beyond individual planning — it involves helping others align their timelines too. Every member worked differently, with some preferring structured routines and others working better under flexibility. This diversity required patience and a degree of gentle coordination to keep the project cohesive. In future group projects, I would introduce earlier internal checkpoints and clearly defined micro-deadlines to ensure that testing, validation, and feedback cycles are distributed more evenly. I also learned that planning buffers for integration and unexpected challenges is essential, as late refinements often take longer than anticipated.

The technical challenges in this project were equally formative. Developing the system's analytical functions — including calculations for moving averages, daily returns, trend sequences, and maximum profit — entirely in native Python was an instructive experience. The task required efficiency, mathematical accuracy, and interpretability, as these algorithms were meant to serve an educational purpose rather than pure performance. Implementing them in linear time while avoiding vectorised shortcuts deepened my understanding of algorithm design, data structures, and complexity management. Benchmarking the manually implemented logic against established Pandas and NumPy methods helped validate both correctness and efficiency.

Through this process, I strengthened several professional programming habits. Writing modular, reusable functions with explicit type hints made the system easier to maintain and test. I developed a more disciplined approach to debugging, using pytest to build confidence in each unit's reliability rather than relying on repetitive print statements. I also maintained version control rigorously — writing meaningful commit messages, syncing branches frequently, and reviewing code before merges to prevent regressions. This taught me how disciplined version control supports teamwork by creating shared visibility over progress and ensuring traceability.

The project's collaborative nature offered lessons beyond coding. Coordinating work within a diverse team meant engaging with different communication styles, work habits, and technical backgrounds. Early on, I learned that not everyone approaches problem-solving in the same way or shares the same comfort level with programming tools. Some preferred visualising progress verbally, while others focused on execution. Recognising this, I adapted my approach by creating small task guides, sharing references, and encouraging open discussions so that everyone could contribute meaningfully. This experience taught me that leadership is not only about directing work — it's about fostering an environment where each person feels supported and capable of growth.

There were moments when maintaining consistent progress required extra initiative, particularly when balancing varying levels of readiness and engagement. Instead of viewing this as a setback, I treated it as a learning opportunity to develop patience, communication, and empathy. I found that steady, constructive communication — asking questions, clarifying goals, and reaffirming shared objectives — was far more effective than rigid enforcement. Over time, this helped the team regain alignment and work more collaboratively toward milestones. This is something I would carry into future professional settings, where balancing differing personalities and priorities is an inevitable part of teamwork.

One of the more valuable takeaways from this experience was realising the difference between taking responsibility and taking control. At times, I found myself stepping in to solve tasks outside my main scope to maintain overall momentum. While this helped in the short term, it also reminded me that sustainable teamwork depends on distributed ownership. A good team leader provides structure and guidance but also trusts others to find their own solutions. This awareness will shape how I approach future collaborations — focusing more on enabling others rather than absorbing additional work.

Working collaboratively also revealed the importance of clear communication channels. Early in the project, I noticed that misunderstandings often arose when progress updates were fragmented or when feedback loops were delayed. Establishing consistent meeting schedules, documenting action points, and maintaining shared notes improved overall transparency. I also learned the importance of tone — constructive and encouraging communication fosters motivation, whereas overly direct feedback, even if well-intentioned, can create tension. These experiences built my understanding of emotional intelligence in teamwork, especially when navigating tight deadlines or differing expectations.

On a personal level, this project helped refine my leadership and technical identity. I learned to translate complex financial logic into accessible educational tools and to communicate programming concepts in ways that supported my teammates' learning. My involvement in structuring the system architecture, documenting technical sections, and interpreting the user study results allowed me to see the project holistically — from concept to delivery. This broadened my perspective beyond coding into areas of project management, user experience, and data communication.

More importantly, I realised that successful projects are built not just on technical accuracy but also on collaboration, trust, and adaptability. Working within a team environment with different personalities and levels of expertise mirrored real workplace conditions, where balancing efficiency, inclusivity, and communication is essential. I learned to view differences not as friction but as complementary strengths — opportunities to understand how others think and work.

From a professional standpoint, this experience reinforced several key lessons. First, preparation and planning form the backbone of any project, but flexibility is equally vital. Second, consistent documentation and testing save time and reduce confusion later. Third, clear communication — especially in distributed teams — is fundamental to avoiding duplication of effort and misalignment. Lastly, leadership involves emotional awareness just as much as technical oversight.

Overall, the *Stalking Stocks* project was more than an academic exercise; it was a microcosm of professional collaboration. It allowed me to apply technical theory to real-world data, strengthen my analytical and programming abilities, and gain first-hand experience in team coordination and communication under realistic constraints. I leave this project with a stronger appreciation of both the technical and human sides of software development. It has shaped my confidence not just as a programmer but as a communicator and collaborator — someone capable of balancing precision with empathy, structure with adaptability, and leadership with humility.

Timothy Chia Kai Lun

This project challenged me to integrate programming basics into an effective solution that was functional and modular, targeting the solution to a concrete real-world problem. My goal from the start was to strengthen my understanding of data structures, modular design, and collaborative version control while gaining hands-on experience in developing production-ready code. I was especially motivated to explore financial data analytics and apply proper software engineering principles and philosophies to our project architecture. I hoped this experience would provide a way to integrate the conceptual classroom ideas with their implementation, giving me a deeper sense of structured design and the nature of group-based development.

My team and I developed an interactive financial stock-analysis dashboard aimed at lowering the barrier to entry for beginner investors via a simulated and risk-free educational platform. My primary task was frontend development using Streamlit, ensuring that each member's work integrated smoothly into a cohesive user interface. Beyond that, I often assisted teammates when they encountered technical issues, as we had differing levels of programming experience.

Early in the process, I noticed an over-reliance on generative AI among my groupmates, like using it for meeting minutes, documentation, and even core programming tasks. While I value GenAI as a learning aid, I believed it should be applied sensibly, as a tool rather than a crutch. I expressed these concerns during a group meeting, which led to partial improvement and more mindful AI use. Around the same time, some members began experimenting directly with code before any planning or architectural design. Although I appreciated their proactiveness, I recognised the risk of accumulating technical debt without a clear structure.

The task delegation process had given its fair share of problems, with most of us still new to workload projections and deadlines. During an in-person discussion with one teammate, they admitted that they were unfamiliar with this level of project management and that the structured approach came as a surprise. That openness made me understand that, while planning was key to me, others were still getting used to that mindset. It made me plan to take things one step before the next, allowing everyone's learning curve to process.

To solve coordination issues, during the meetings I tried to guide the group toward increased systematic behaviour by walking them through concepts like programming standards, modularisation, and dependency management. This put me under strain to balance leadership with cooperation; though I had wanted to ensure the quality of the project, I realised that taking all the responsibilities prevented the others from getting experience. In hindsight, this tension became one of my most important lessons on teamwork and self-awareness.

This project tested not only my technical abilities but also my patience and adaptability. Initially, I believed that progress depended on maintaining tight control over every detail. However, as the project evolved, I learned that genuine teamwork involves trust because trusting that others will grow through their own trial and error, and that progress sometimes comes from letting go rather than stepping in.

One recurring challenge was communication. Some members preferred discussing ideas privately instead of using our group chat, which made it harder to maintain transparency and shared understanding. At first, I found this frustrating, but later realised it reflected different comfort levels with online collaboration rather than disinterest. By addressing this gently and encouraging more open discussion channels, I learned how inclusive communication directly influences efficiency and team morale.

In retrospect, leadership was something less management-like and something more growth-enabling. I learned to slow down, explain my reasoning, and invite feedback instead of simply correcting issues. Over time, my teammates became more engaged and confident in their roles. This experience also reshaped how I view GenAI, not as something to resist, but as a tool that demands human judgment and accountability.

The lessons from this project will shape how I work for the rest of the term and beyond. In the near term, I will formalise lightweight planning routines (brief scoping notes, task breakdowns, and clear "definition of done") before writing code, so that modules in upcoming assignments remain cohesive and low-debt. I will also adopt transparent communication habits to reduce information gaps and invite timely help. My stance on GenAI has shifted from convenience to critical use: I will treat outputs as drafts to be verified, annotated, and refactored for clarity, with citations where appropriate. Longer-term, these practices support my goals in the following trimesters with modules involving similar technical knowledge.

Overall, this project grew my appreciation of the balance between technical rigour and collaboration. It reinforced that software development is as much about people and process as it is about code. I now approach future group projects with greater empathy, encouraging thoughtful GenAI use and open communication. This mindset, I believe, will serve me well as I progress toward a career in AI, software development, and research.

Low Gin Lim

Throughout the project, I managed my time by planning tasks around our weekly meetings and shared progress trackers. Each meeting ended with clear deliverables, which helped me stay organised and accountable for the documentation and presentation sections I was responsible for. I used GitHub commits as a record of progress and worked in short, focused sessions to update code comments and refine written sections. If I were to do this again, I would begin report writing earlier in parallel with coding to avoid last-minute editing and formatting work.

The main technical challenge for me was developing my own version of the four analytical functions—SMA, Trend Runs, Daily Returns, and Maximum Profit—and comparing them with my teammate's implementation. Although we eventually used his version, the process deepened my understanding of how algorithms are structured and validated.

Beyond that, most of my learning came from improving our software architecture and documentation practices. I learned the importance of maintaining a modular file structure, where each component handles a single responsibility, and how this makes collaboration and debugging much easier.

I also gained practical experience with Git and GitHub, including creating branches, submitting pull requests, resolving merge conflicts, and keeping documentation consistent across the repository. These tools made teamwork smoother and reduced versioning errors.

Good programming practices I learned include keeping functions short and reusable, writing meaningful docstrings, and using consistent naming conventions. Poor practices to avoid are hard-coding values, leaving ambiguous comments, and editing shared files without proper version control.

This project gave me a clearer picture of what real collaborative software development feels like. I learned that good communication and clear documentation are just as vital as writing correct code. Working closely with my teammates taught me to appreciate version control discipline and the need for readable, maintainable code over quick fixes. Preparing the Results & Insights section and contributing to the final presentation also helped me connect technical outputs to meaningful explanations for users. Overall, I became more confident in using professional workflows, structuring Python projects correctly, and collaborating effectively on a shared codebase.

Mohammed Aamir

This project was a valuable learning experience that provided a glimpse into what teamwork and project-based learning will be like throughout my university journey. Being in my first trimester and handling my first major assignment, I could already feel the intensity and expectations of a real collaborative environment. Coming into this module, I lacked confidence since I had no prior background in Python, unlike most of my teammates who were already familiar with programming. Despite that, I was entrusted with the role of data validation and cleaning, which quickly became both a challenge and an opportunity for growth.

Initially, I worked with my teammate to design and implement the data-cleaning and validation functions. However, our early approach was later replaced when we realised that the methods we used did not align with the project's main processing pipeline. This experience taught me the importance of clear communication and alignment among team members before diving into implementation. It reinforced that good teamwork is not just about dividing work but ensuring everyone understands the system as a whole.

To strengthen my technical understanding, I spent additional time watching YouTube tutorials, exploring sites like W3Schools to review Python fundamentals, and using ChatGPT for clarification when I encountered difficult concepts. Still, the level of coding required for this project went beyond the basics. Since we relied heavily on Pandas and the Yahoo Finance library, I had to research how to structure functions that could effectively interact with these libraries. This helped me gain a deeper appreciation for proper function design and modularity in programming.

I also gained hands-on experience with GitHub, learning how to manage branches, create pull requests, merge updates, and write meaningful commit messages. Through guidance from my teammates, I picked up valuable programming practices such as writing docstrings, following consistent naming conventions, and including clear comments for readability. These habits will be essential for future coding projects, both in university and beyond.

Time management was another area I learned from. Although I began work early, unexpected changes like having to rewrite parts of the code, required extra time that had originally been planned for other tasks. This experience taught me the importance of flexibility and building buffer time into my schedule to handle unforeseen adjustments.

Overall, I am grateful that we had this project early in the trimester, as it helped set realistic expectations for the rest of the course. Working with a supportive and skilled team made the experience especially rewarding. I have learned not only new technical skills but also the value of teamwork, adaptability, and proactive communication, all of which will guide me in future projects.

Dalton Chng

Our team divided tasks based on individual strengths and prior experience. I was responsible for data cleaning and validation during the coding phase, and later, for testing, validation, and the results section of the report. As data cleaning was the foundation for the project, my partner and I aimed to complete it early to give the rest of the team a head start. We maintained regular meetings with recorded minutes and clear deliverables, but I think having smaller checkpoints between each meeting would have helped us stay more accountable. Balancing deadlines and returning to student life after a break was difficult, and I often felt overwhelmed. However, I learnt that proper time management and pacing myself were essential to keeping stress under control. If I were to do this again, I would start documenting my report alongside development, instead of leaving writing to the end.

At the start, everything felt foreign. Before this module, I had only used Python's IDLE and Jupyter Notebook, so working with GitHub, branches, and pull requests was completely new to me. I initially struggled to keep up with the team's technical proficiency and often felt intimidated by the complexity of the repository and workflow. Over time, I adapted by asking questions, observing my teammates' code, and using AI tools more tactfully, to learn rather than to rely on. The project taught me how modular programming works in practice. The separation of code into different files and functions with defined inputs and outputs made sense once I saw how easily our modules integrated. It also showed me the importance of version control, documentation, and reviewing pull requests. If there were mistakes, they were caught early, and everyone could learn from them.

For this project, I took a more backseat role than I was used to in polytechnic. It was a big shift from leading to learning how to listen, follow, and adapt. At first, I found it hard to share ideas because I lacked confidence, but my teammates were encouraging and never made me feel out of place. I realised that it's alright to still be learning as long as you are willing to improve. I also learnt that being proactive must be balanced with allowing others the space to grow. After discussing it as a group, we agreed that sharing responsibilities equally was important for everyone's development.

Working styles and expectations varied across the team, and the initial project plan was overly ambitious for an introductory module. Through consultations with our lecturer and other teams, we refined the scope to something more achievable. Our weekly meetings helped us align progress, and though communication across different platforms could be overwhelming, it ensured everyone stayed informed.

This project was my first real experience with professional collaboration and software development. I now feel much more confident in taking on technical roles and contributing meaningfully to group projects. It has strengthened my appreciation for teamwork, patience, and clear communication. Looking ahead, I am excited to apply what I have learnt as I continue my journey in AI and programming, particularly as I move on to learning C with a stronger foundation and a better mindset.