Post Earnings Announcement Drift (PEAD) Algorithm V&V

Group 10

Archit Sharma (shara96@mcmaster.ca)

Bhuvesh Chopra (choprab@mcmaster.ca)

Braedon Kwan (kwanb5@mcmaster.ca)

Capstone 4ZP6

Version 1

April 4, 2025

Table of Contents

PROJECT DESCRIPTION 3
COMPONENT TEST PLAN 3
DataFrame (YahooFinance) [Earnings Estimates] *3*
DataFrame (AlphaVantage) *4*
DataFrame (Interactive Brokers) [News Data (Live)] *4*
Regression Module *5*
Signal Module *5*
API Module *5*
Datafetch Module *6*
Backtest Module *6*
User Interface *7*
TESTING RESULTS (ADDED TO VNV) 7

Project Description

This project aims to develop an automated trading algorithm that exploits the Post Earnings Announcement Drift (PEAD) anomaly, where stock prices drift following earnings surprises. By analyzing discrepancies between expected and actual earnings using key financial metrics such as Earnings Per Share (EPS), revenue, and gross profit margins, the system generates trade recommendations ("buy," "sell," or "hold"). These recommendations are displayed through a web-based interface, while the backtrader library is used for backtesting and analyzing historical performance. The algorithm integrates real-time data ingestion, machine learning models, and decision-making components to ensure efficient and systematic trading execution.

SRS: SRS Design-Doc: DesignDoc

Component Test Plan

DataFrame (YahooFinance) [Earnings Estimates]

Unit Tests: Our goal is to ensure that the Yahoo Finance DataFrame component accurately retrieves, processes, and integrates earnings estimate data for backtesting and trade signal generation. To achieve this, we implement multiple unit tests focusing on data retrieval accuracy, filtering correctness, and merging consistency. The primary test verifies that the get_earnings_dates(64) function successfully fetches earnings estimates and reported dates for a given stock, checking for completeness and format consistency. Additionally, we validate that the filtering mechanism correctly eliminates future earnings data by ensuring that no records with dates beyond pd.Timestamp.now() remain. To handle undesired behaviors, we include tests for invalid tickers, where non-existent stock symbols should return an empty DataFrame without causing execution errors. We also test excessive data requests, ensuring that requesting more than the available historical records does not break functionality. Furthermore, merging tests validate that earnings data aligns with financial indicators without introducing NaN values or date mismatches. By incorporating these edge cases, our unit tests provide robust verification of data integrity and pipeline stability.

Performance Test and Metrics: One of the key performance metrics for this component is the API response time when retrieving earnings data. To ensure efficiency, we measure the execution time of the get_earnings_dates(64) function, with an expected completion threshold of under 2 seconds. Another crucial metric is the speed of filtering operations, where eliminating future earnings data must be completed within 50 milliseconds to support real-time data processing. Additionally, we assess memory usage during the DataFrame merging process, as excessive memory consumption can impact overall system performance. Using memory_usage(deep=True), we ensure that merging does not increase memory consumption beyond 10% of the original dataset size.

DataFrame (AlphaVantage)

Unit Tests: Unit testing for the AlphaVantage DataFrame component ensures accurate retrieval, processing, and storage of financial data. We validate that API responses contain the "quarterlyReports" key and that extracted data aligns consistently across income_df, balance_df, and cashflow_df. Additionally, tests ensure dynamic stock symbol input works without modifications and that missing or invalid API responses generate appropriate error messages. To address undesired behaviors, we simulate API rate limits to verify graceful failure handling. Tests also confirm that excessive requests do not crash the system, and that the data extraction process maintains structural integrity. These validations help maintain reliable data ingestion for further financial analysis.

Performance Test and Metrics: To ensure efficiency, we evaluate API response time, with requests.get() expected to complete within 1.5 seconds per request. Data extraction speed, measured from JSON conversion to DataFrame storage, should remain under 100 milliseconds to prevent bottlenecks. Memory usage is another key metric; using memory_usage(deep=True), we ensure that DataFrame operations do not exceed 10% additional memory overhead. Stress tests simulate high-frequency data requests to assess system scalability under concurrent retrieval. These performance measures ensure real-time financial data processing remains optimized for trading decisions.

DataFrame (Interactive Brokers) [News Data (Live)]

Unit Tests: Unit testing for the Interactive Brokers News DataFrame (Live Data) component ensures accurate retrieval, parsing, and sentiment analysis of earnings-related news. The primary test verifies that the tickNews method correctly fetches earnings headlines for the selected ticker, ensuring data consistency. Additionally, regex-based extraction of Adjusted EPS and earnings surprise keywords (e.g., “beat”) is validated to confirm accurate parsing of financial statements from news articles. To handle undesired behaviors, tests verify system response when encountering missing or malformed headlines, ensuring robust error handling. We also validate that multiple headlines for the same earnings event are correctly appended to a Pandas DataFrame without duplication. Finally, unit tests ensure that sentiment aggregation across all headlines properly influences the final trade signal, preventing misleading sentiment classification.

Performance Tests and Metrics: Performance evaluation of the Interactive Brokers News DataFrame focuses on latency, execution speed, and sentiment accuracy. We measure the delay between earnings release and system recognition, ensuring that signals are generated within milliseconds of news retrieval. Additionally, we analyze trade execution speed by comparing signal generation timestamps with earnings announcement times to assess latency impact on trading decisions. To ensure scalability, tests simulate high-frequency news inflows, validating that the system remains responsive under rapid updates. We also benchmark regex extraction performance on large textual datasets, ensuring efficient parsing without delays. Finally, sentiment analysis accuracy is tested to confirm correct classification into positive, negative, or neutral, and signal validation is verified to ensure correct buy/sell decisions when earnings exceed market expectations.

Regression Module

Unit Tests: Unit testing for the Regression Module ensures accurate feature selection, data alignment, and prediction consistency. Tests verify that correlations are correctly computed and that only features with correlation above 0.20 are included in reduced_data, ensuring proper filtering. Additionally, we check that shifting Reported EPS aligns historical earnings data correctly, preventing misalignment errors in training. Further tests validate that train_test_split() correctly partitions data with a 70/30 split and that LinearRegression.fit() properly trains the model. Edge cases include handling missing or collinear features, ensuring they do not degrade prediction accuracy. Finally, tests confirm that predict() generates valid numerical outputs without unexpected NaNs or extreme values.

Performance Tests and Metrics: Performance evaluation for this component focuses on model accuracy, computational efficiency, and stability. We use R^2 (coefficient of determination) to measure prediction accuracy, expecting values above 0.75 for reliable forecasting. Additionally, Mean Squared Error (MSE) quantifies deviations between actual and predicted earnings, ensuring precision. For efficiency, we monitor training time, ensuring model fitting completes within 500ms for standard datasets. To assess robustness, we track variance inflation factor (VIF) for selected features, preventing excessive multicollinearity. These tests ensure the regression model remains performant, interpretable, and scalable for trading applications.

Signal Module

Unit Tests: Unit testing for the Signal Module ensures accurate trade signal generation based on earnings data. The signal() function must return "BUY" for a surprise ratio ≥ 1.22, "SELL" for ≤ 1.04, and "NONE" otherwise. Tests validate correct computation of the ratio and handling of edge cases, such as division by zero, extreme values, and floating-point precision errors. The flipsignal() function is tested to ensure correct signal reversals, flipping "BUY" to "SELL" and vice versa, while "NONE" remains unchanged. Additional tests verify that invalid inputs trigger appropriate error handling or default "NONE" outputs.

Performance Tests and Metrics: Performance evaluation ensures low latency and scalability in signal processing. The signal() function must execute within 1ms, supporting real-time trading. Scalability tests assess performance under bulk processing of high-frequency earnings data across multiple stocks. Backtesting against historical earnings data measures prediction accuracy, refining static thresholds (1.22, 1.04) for better trade alignment. The flipsignal() function is tested for execution under 0.5ms, ensuring reliable signal reversals without unnecessary delays.

API Module

Unit Tests: Unit testing for the API Module (Interactive Brokers API) ensures correct data retrieval, order execution, and news processing. reqRealTimeBars () is tested for accurate price updates, while reqMktData () and tickNews () validate real-time news integration. PlaceOrder () is checked for correct execution parameters, and reqHistoricalData () is

tested for timestamp alignment in past data retrieval. Edge cases like API rate limits, missing responses, and connection failures are handled to ensure seamless operation.

Performance Tests and Metrics: Performance evaluation focuses on latency, reliability, and fault tolerance. reqRealTimeBars () must update prices within 500ms, and reqMktData ()/tickNews () must deliver real-time updates in under 1 second. Order execution latency is measured, with expected completion in <100ms for liquid stocks. Resilience tests simulate network disruptions and API downtime, ensuring smooth recovery and retry mechanisms. These optimizations guarantee low-latency, high-reliability interactions with Interactive Brokers.

Datafetch Module

Unit Tests: Unit testing for the Datafetch Module ensures accurate retrieval, storage, and organization of historical stock data. The primary test verifies that each stock's data is stored separately, preventing overwriting or data leakage between tickers. Additionally, we validate timestamp alignment across all extracted tickers, ensuring that historical periods remain consistent without misalignment. To maintain data integrity, tests check for duplicate or missing entries, confirming that historical data remains complete and structured properly. Another crucial validation ensures that API responses return correct stock identifiers, preventing datasets from being mixed due to mislabeling or incorrect ticker handling.

Performance Tests and Metrics: Performance evaluation of the Datafetch Module focuses on latency, scalability, and data integrity. We measure latency when retrieving large datasets across multiple stocks, ensuring that multi-ticker historical extraction remains efficient. Additionally, we assess memory usage and storage efficiency, confirming that high-frequency financial data does not cause excessive resource consumption. To ensure robustness, we evaluate API rate-limit handling, ensuring that automated retries and delays prevent disruptions in large-scale extractions. Lastly, extracted stock values are cross-referenced with external market data sources, verifying that fetched historical prices and metrics remain accurate and reliable.

Backtest Module

Unit Tests: Unit testing for the Backtesting Module ensures accurate historical data extraction, signal validation, and profit/loss computation. The primary test verifies that price data between 16:05 PM and 16:40 PM is correctly extracted from the GOOGL_Earnings_Data(5M).csv file, ensuring completeness. Additionally, merging with historical earnings data and regression estimates is tested to confirm proper alignment across all datasets. The signal() function is tested to ensure it correctly classifies BUY, SELL, or NONE based on earnings surprise calculations. Edge cases such as missing earnings estimates, division by zero errors, and extreme price fluctuations are handled to prevent erroneous trade signals. Further, PnL calculations are validated to ensure price movements before and after signal execution are correctly accounted for, with no data leakage or misalignment.

Performance Tests and Metrics: Performance evaluation focuses on backtesting efficiency, accuracy, and scalability. Data extraction time is measured to ensure historical prices are retrieved and processed within 500ms per stock, allowing for quick batch analysis. Additionally, trade signal execution time is benchmarked to confirm that signals are generated in under 1ms per entry, ensuring real-time responsiveness. To validate backtest accuracy, historical trade signals are compared against actual stock price movements to measure PnL deviation from expected returns. Further, scalability tests are conducted to evaluate system performance when backtesting multiple stocks simultaneously, ensuring consistent performance under large datasets. By integrating these tests, the backtest module is optimized for realistic and reliable trading strategy evaluation.

User Interface

Unit Tests: The UI offers five main views: an Overview Dashboard for summary-level performance insights, a Stock Details View for ticker-specific analysis, a Comparison View for metric benchmarking across stocks, a Trade History View for reviewing individual trade logs, and an About View for methodology and metric definitions.Component Rendering and Display tests verify that all components render properly on various devices and screen sizes, ensuring a consistent user experience. Navigation tests confirm that the search button and symbol links function as intended, preventing broken routes or dead links. In addition, the equity graphs and associated performance metrics ensure a dynamic and flexible backtest experience.
Performance Tests and Metrics: To maintain a responsive user experience, Latency Measurement tests track the time taken to load the two views, Dashboard and Performance Analysis, ensuring transitions remain smooth (e.g., under two seconds). Data Retrieval Performance tests measure how quickly the system can fetch backtest data, trading signals, and other financial information from the server, confirming that dynamic updates do not block or degrade user interactions.

Testing Results (Added to VnV)

A combination of regression modeling, backtesting, and thorough module-level tests was performed to validate the solution's accuracy, reliability, and resilience.

Regression Modeling
Ordinary Least Squares (OLS) regression models were developed to forecast Earnings Per Share (EPS) for GME, GS, MSFT, NVDA, and GOOGL. GME's model (R^2 = 0.676) predicted 0.18 EPS (actual 0.01), with significant coefficients for other current assets and change in operating liabilities. GS (R^2 = 0.747) and MSFT (R^2 = 0.956) predictions closely matched their reported EPS, underscoring strong model performance. NVDA (R^2 = 0.813) showed a larger gap between predicted (2.26) and actual (0.81), while GOOGL (R^2 = 0.914) saw near parity between predicted (2.13) and actual (2.12). Large condition numbers in some models suggest multicollinearity, which will be examined further.

Backtest and Other Component Testing

Extensive backtesting under varied market conditions was performed, with results presented on our trading dashboard. Users can review performance metrics in real time, confirming strategy resilience to both typical and stress scenarios. Beyond backtesting, targeted unit tests were conducted for the project's dataframes, signal module, and API module. Dataframe tests verified data integrity, ensuring valid columns, correct data types, and proper handling of missing records. The signal module's unit tests confirmed that buy/sell signals were generated accurately based on defined thresholds and edge conditions. In parallel, API module tests validated endpoint reliability and response handling under simulated production loads. All tests passed consistently, exhibiting no significant performance regressions or failures. Integration tests further confirmed that the data pipelines and user-facing components communicate effectively and maintain stable throughput.