Post Earnings Announcement Drift (PEAD) Algorithm Design Document

Group 10

Archit Sharma

Bhuvesh Chopra

Braedon Kwan

Capstone 4ZP6

Version 1

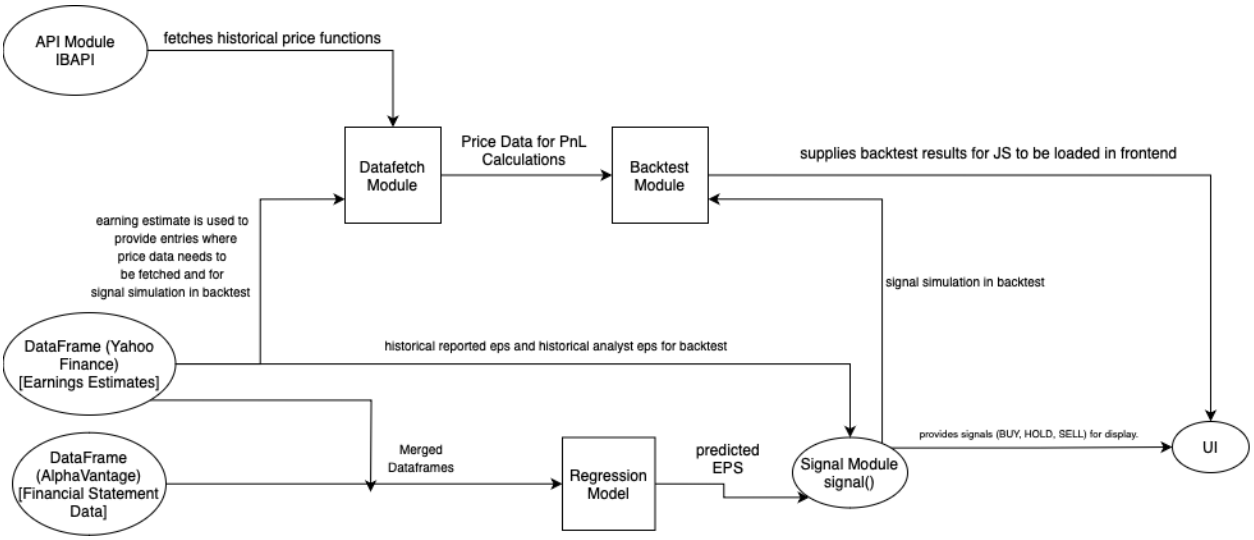
April 3, 2024

Table of Contents

Purpose Statement

This project aims to develop an automated trading algorithm that leverages the Post Earnings Announcement Drift (PEAD) anomaly, a phenomenon where stock prices drift following significant earnings surprises. The system integrates real-time data ingestion, machine learning models, and decision-making components to analyze earnings discrepancies and generate actionable trade recommendations ("buy," "sell," or "hold"). These recommendations are displayed via a user-friendly web interface, while PowerBI is used for backtesting and analyzing historical performance metrics. This document outlines the system's architecture, functionality, and implementation details, ensuring alignment with technical requirements and business goals for eTective trading strategies.

Diagram of Components



Relationship between Project Components and Requirements

System Component	Requirement Covered
DataFrame (YahooFinance) [Earnings Estimates]	**P0**: Historical EPS Data over a span of 10 years for multiple companies in similar sectors.
DataFrame (AlphaVantage) [Financial Statement Data]	**P0**: Parsing past finanical reports including cashflow statements, income statements, and balance sheets to derive features such as Accounts

	Receivables, COGS and many more for EPS prediction.
Regression Model	**P1**: predicting the next whisper number by analyzing multiple features from earnings reports and live data. **P1**: algorithm would be trained using deltas (margins) based on actual, required and whisper EPS. **P1**: training the model on a broader dataset from similar companies and sectors
Signal Module (implemented inside the backtest module)	**P0**: Action Determination, comparing the predicted and reported EPS to output a trading decision. **P1**: The algorithm would hard code the required margins/thresholds between these three EPS's metrics which is a weighted average of these surprises, which we are considering to generate a trading signal between buy, sell or do nothing and sectors
Datafetch Module (implemented to conduct backtesting)	**P1:** fetches historical price data (5 min frequency) from 4pm – 6pm for the assets in a sequential manner from IBAPI() for every earnings date in the last 10 years.
Backtest Module	P0: Simulate signals and returns for historical data and assess the viability of the product
UI	P0: provides an overview of financial metrics and actionable recommendations and provides historical performance charts (visualization)

Project Components

Dataframe [Yahoo Finance] [Earnings Estimate]

Normal Behaviour: The dataframe components are used to store different levels of information. We decided to include them as a component as each dataframe provides important inputs and are integral for the workflow. The DataFrame (YahooFinance) stores earnings estimates, reported estimates and dates in (YYYY-MM-DD HH:SS) format for the last 10 years.

API and Structure: Input to the get_earnings_dates() is the stock that we want to request the data for and the number of data entries we want which is 64 in our case.

Implementation: To make the calls generalizable we make a dictionary of tickers which we loop through and call the get_earnings_dates(64) API function to get the desired data entries. The call then returns the future earning estimates and dates which are stored in the dataframe.

Potential Undesired Behaviour: As mentioned above the call returns future earning estimates and dates which might not be appropriate for the backtest or the regression so we have cleaned it before using it further.

Dataframe [AlphaVantage]

Normal Behaviour: For storing and retrieval of income statement, balance sheet and cashflow data for each quarter for the past 10 years for a particular company/stock.

API and Structure: This includes making an API call to the AlphaVantage API using requests.get (), three such calls are made, one for each financial statement with the parameters in the call set as the name of the financial statement for which data is required, the unique API key and the name of the company.

Implementation: The response is converted to a Python dictionary using response.json(). If the key "quarterlyReports" exists, the data is extracted into a Pandas DataFrame for analysis, with the first few rows displayed using .head(). Otherwise, an error message and the raw response are printed for debugging.

Potential Undesired Behaviour: The code may exceed Alpha Vantage API rate limits if pre-saved CSV files aren't used, and the hardcoded stock symbol limits flexibility for analyzing multiple stocks.

Regression Module

Normal Behaviour: This component is designed to filter financial statement data based on correlation with a specific target variable, 'Surprise(%)', and to use multivariate regression to predict the next quarter's earnings based on the selected features.

API and Structure: The financial data is provided in a Pandas DataFrame (finance_df). The Reported EPS is shifted by one quarter to use historical data for prediction. A multivariate regression model is trained on the reduced dataset using Scikit-Learn's LinearRegression, with a portion reserved for testing, and predictions for the next quarter's earnings are made based on the regression model.

Implementation: The component calculates correlations of all columns with the target column, filters out those with a correlation below a specified threshold (0.20), and creates a reduced dataset containing the most relevant features along with Reported EPS which is present in the reduced_data dataframe. The data is then divided into training and testing sets using train_test_split() with a test_size of 30%. Finally the model is fitted using the .fit() function and can be used to make predictions once the r^2 is calculated.

Potenial Undesired Behaviour: Using a fixed correlation threshold might exclude significant features or include collinear ones, impacting model accuracy. The alignment of Reported EPS through shifting assumes that quarters are properly synchronized, misalignment could lead to erroneous predictions.

Signal Module

Normal Behaviour: Generates the signal for the particular equity to be traded and is called by the Backtest module to simulate a trade.

API and Structure: The component is embedded in the next() function for the being signal(stock, vec,estimate) and the former takes the actual earning (vec) and the estimated earning being estimate (outputted by the regression module), and the latter just reverses the decision of the signal method.

Implementation: Calculates the surprise by using a weighted average between the regression prediction surprise and the analyst prediction surprise, generating a "BUY" if it exceeds an established margin such as 1.22 or ‘SELL’ if its below the established margin. If it is in between then we do not make any trades.

Potential Undesired Behaviour: Relies on hardcoded threshold values such as 1.22 and 1.04, which could be estimated more robustly with change in basket of stocks and sector of industry.

API Module (Interactive Brokers API)

Normal Behaviour: We have mentioned the API module as a component as it acts as the orchestrator and sends inputs to the Datafetch module and the dataframes. Additionally, the built in functions which are used in this project maintain robust functionality.

API and Structure: Major functions used are reqHistoricalData() for the data fetch module.

Implementation: The implementation is done by Interactive Brokers. Link for the same is

as follows: https://ibkrcampus.com/campus/ibkr-api-page/twsapi-doc/#bp-reauthenticate

Potenial Undesired Behaviour: Syntactic Inconsistencies, Functional issues, Increased latency, and Maintenance shutdowns.

DataFetch Module

Normal Behaviour: Extracts Price Data for the specified tickers from 4pm - 7pm sampled at -minute frequency for the last 40 earning dates.

API and Structure: Configured as a script, requires the list of tickers, the time frame(hours) and the frequency (15 minutes) , and the earnings_dataframe. The major ibapi() function used here is reqHistoricalData() which requires the timeframe, frequency, and the reqIds along with the ticker information and the earnings dates which is derived from earnings_df.

Implementation: The prices are fetched through the TestApp() class which loops through the tickers available and then the earning dates provided. Each Call requires a unique reqID so we first make a reqId dictionary with each set of reqIds linked to a specific ticker. We retrieve the historical data for the provided tickers and output them in a csv file which is stored for further use. Also note that there is an onlyRTH flag which is set to false so that we can extract outside regular trading hours data.

Potenial Undesired Behaviour: Concurrency issues from the broker which might cause the data to be redundant or inconsistent across diTerent stock tickers (we are currently facing this issue because of which we need to do this process individually for each ticker).

Backtest Module

Normal Behaviour: Simulates trading activity over historical data and prices and generates a Profit and Loss report which would help us understand the profitability of this strategy.

API and Structure: A script would be made which would accept the historical price data frame from the Datafetch module and would manipulate the data frame based on

the signal function implemented along with the earnings_df to work.This functionality is achieved using the Backtrader Library in Python

Implementation: EPS surprises would be calculated historically using the regression module and would be passed into the signal function, once a signal is established("BUY,SELL,NONE"), prices would be subtracted from each other according to the signal and the holding period. For example, if there was a BUY signal and the holding period was 120 minutes, we would subtract price at 18:20 from 16:05 price and label that as a row in the returns column for th5is trade instance.

Potenial Undesired Behaviour: Backtest results are overly optimistic due to strong assumptions which might not hold in real time trading.

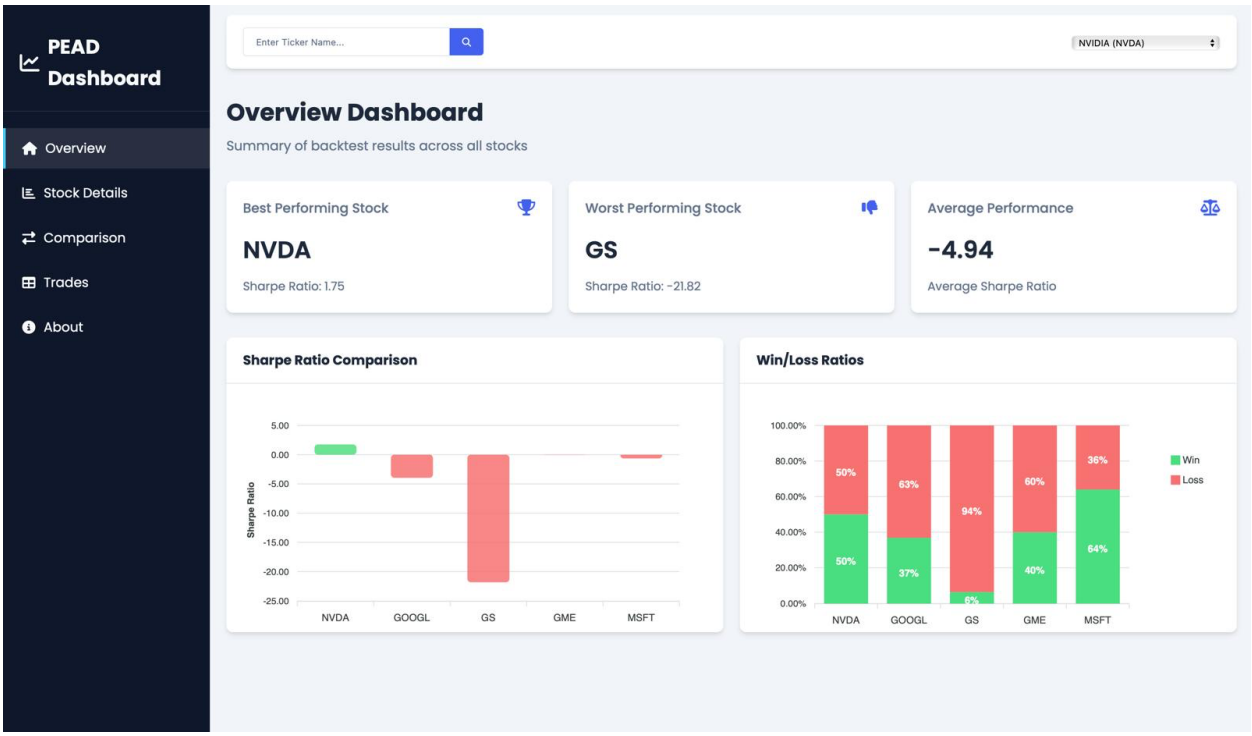
User Interface

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. It retrieves real-time and historical data through REST APIs and WebSocket feeds, allowing for interactive filtering, metric selection, and date-based parameter tuning.
Built with React.js, styled using Bootstrap CSS for a responsive layout, and featuring Chart.js for dynamic charting, the UI offers a dark theme and modular card-based components to enhance usability and visual clarity. Elements such as equity curves, performance comparisons, and win/loss breakdowns are designed to guide both individual stock evaluation and overall strategy review. Color-coded signals, responsive input forms, and chart interactivity improve navigation and user experience. Further implementation details are shared in the Appendix.

Appendix

The appendix provides more detailed information about the User Interface
The User Interface (UI) for the PEAD Trading Dashboard has been redesigned to improve visual clarity and user interaction while retaining the analytical depth of the original layout. The new UI introduces multiple modular views aimed at simplifying data interpretation for both real-time trading logic and backtest evaluation. The interface is now divided into five main views: the Overview Dashboard, the Stock Details View, the Comparison View, the Trade History View, and the About section.

Overview Dashboard



The Overview Dashboard provides a high-level summary of the backtesting results across all stocks. The layout displays comparative performance metrics and identifies top and bottom performers to assist in portfolio-level decision-making.
Key elements on this view include:

- Best Performing Stock: Highlights the stock with the highest Sharpe Ratio from the backtest results.
- Worst Performing Stock: Displays the lowest Sharpe Ratio recorded.
- Average Performance: Shows the mean Sharpe Ratio across all tested stocks.

Visual charts supplement the overview:

- Sharpe Ratio Comparison: A bar chart showing the Sharpe Ratios of each stock. Green bars denote positive risk-adjusted returns, while red bars indicate negative performance.
- Win/Loss Ratios: A stacked bar chart visualizing the percentage of winning and losing trades per stock.

This view is designed for quick performance benchmarking across the portfolio.

Stock Details View

The Stock Details View provides a detailed breakdown of a single stock's performance during the backtest period. The information is displayed in a dashboard-style layout for ease of analysis.
Metrics on display include:

- **Sharpe Ratio**: Risk-adjusted return of the trading strategy on the selected stock.
- **Total Trades**: Total number of trades executed for this stock.

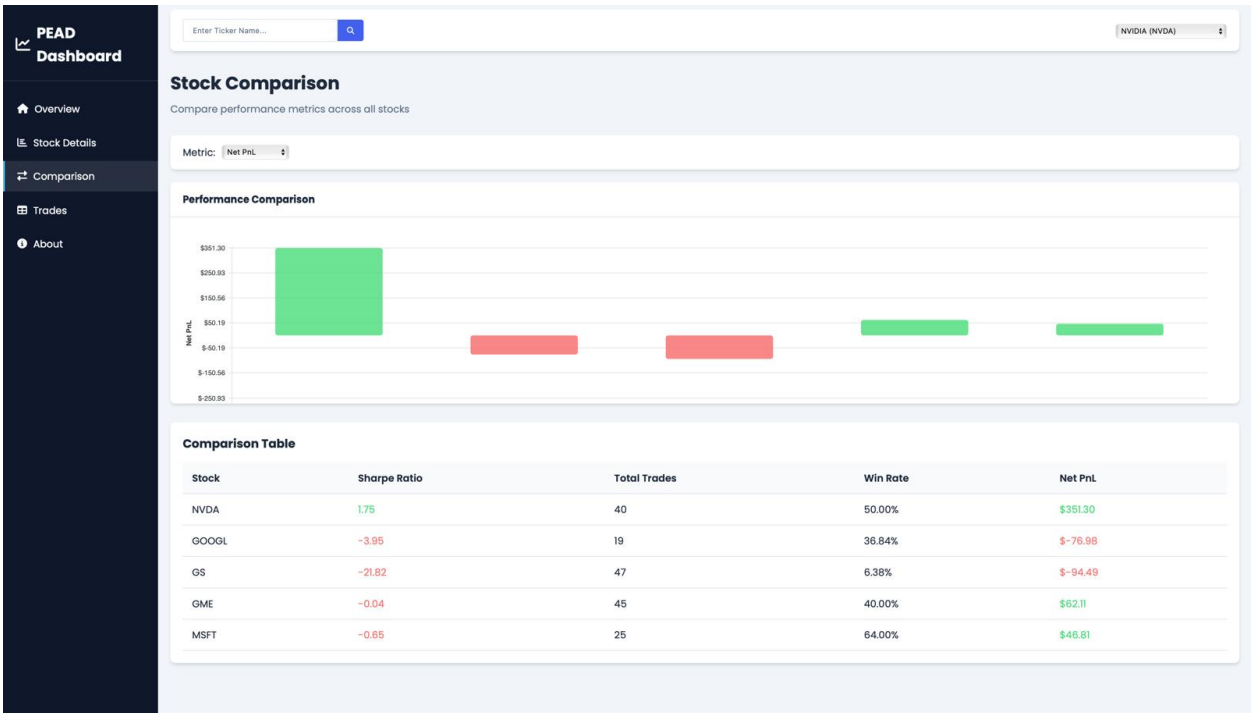
- **Win Rate**: The percentage of profitable trades.
- **Net PnL**: Net profit or loss after accounting for trading costs.

Three charts are featured in this view:

- **Equity Curve**: A line graph tracking the evolution of the trading account's value over time.
- **Trade Outcomes**: A pie chart showing the proportion of winning vs. losing trades.
- **Trade Streaks**: A bar chart depicting the current and longest win/loss streaks.

This view enables deeper inspection of trading behavior at the individual stock level.

Stock Comparison View



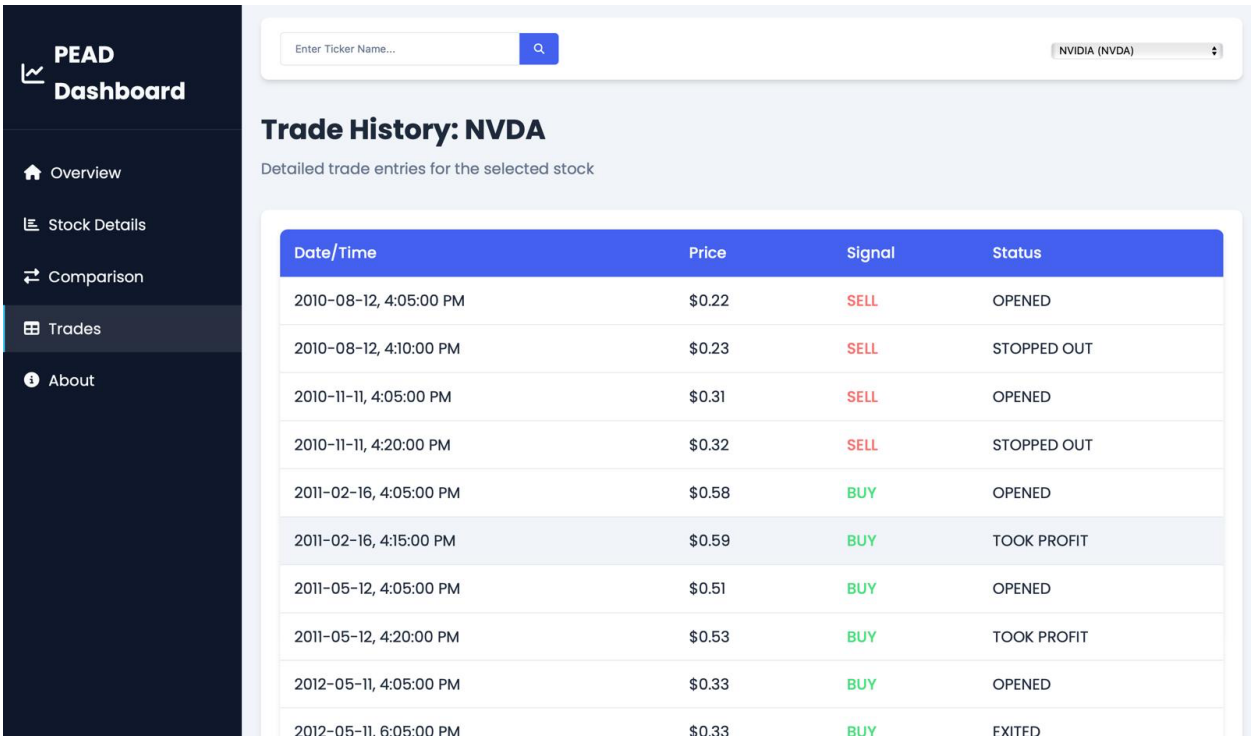
The Stock Comparison View allows users to compare the strategy's performance across multiple stocks using the charts and tables we provide.

The view includes:

- **Metric Selector**: A dropdown menu that lets users choose the comparison metric (e.g., Net PnL, Sharpe Ratio)
- **Performance Comparison**: A bar chart displaying the selected metric for each stock. Green bars represent positive results, while red bars indicate losses.
- **Comparison Table**: A table that includes:
    - **Sharpe Ratio**
    - **Total Trades**
    - **Win Rate**
    - **Net PnL**

This view helps in identifying which stocks are consistently profitable and which may require strategy tuning or exclusion.

Trade History View



This view compiles all the trades that were made in the simulation for a particular simulation and includes all the important details about the trades such as the timestamp, order Type, Position, and Exit Type of the Trade.

About View

The About section describes the strategy's foundation and explains the key performance metrics used across the dashboard and the methodologies.

We discuss the purpose and the the core topics behind PEAD which the user might want to know about here.

Some of the Main Topics covered so far :

- Sharpe Ratio
- Win Rate
- PnL
- Streaks