Post Earnings Announcement Drift (PEAD) Algorithm Design Document

Group 10

Archit Sharma

Bhuvesh Chopra

Braedon Kwan

Capstone 4ZP6

Version 0

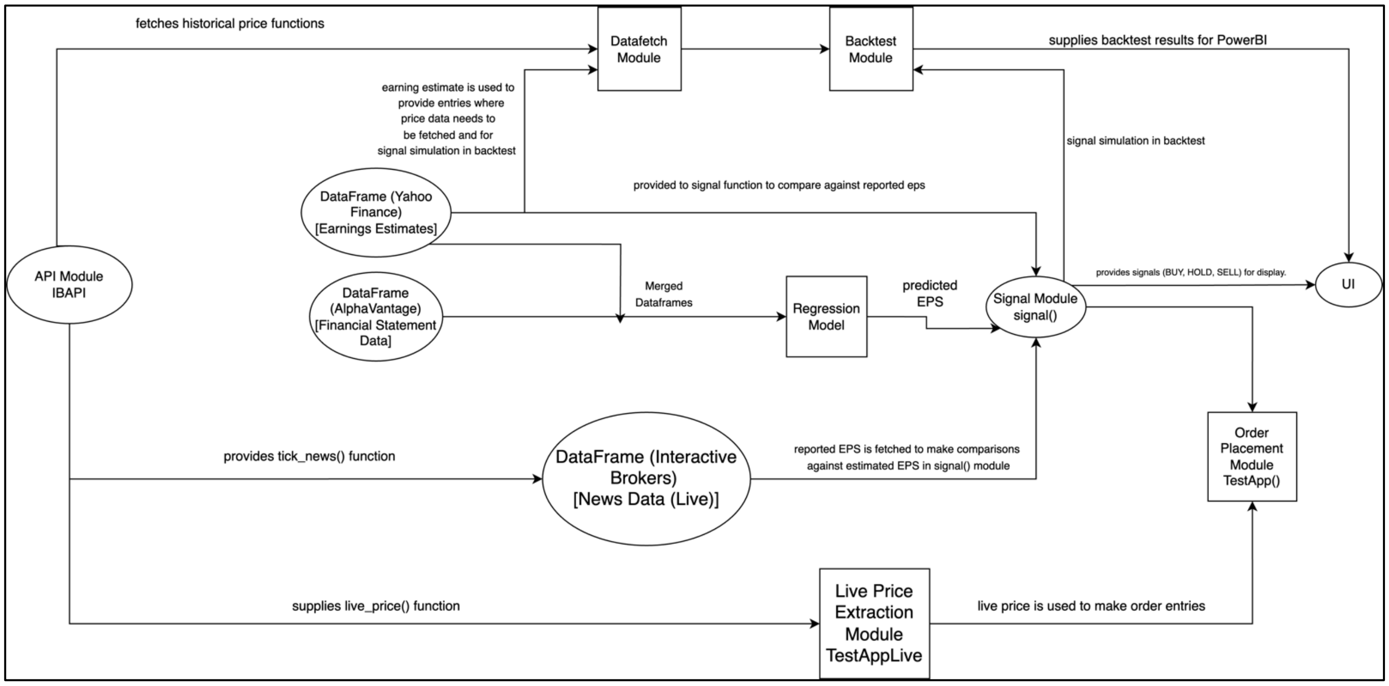
January 24, 2024

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 effective trading strategies.

Diagram of Components


fetches historical price functions
Datafetch Module
Backtest Module
supplies backtest results for PowerBI
earning estimate is used to provide entries where price data needs to be fetched and for signal simulation in backtest
signal simulation in backtest
provided to signal function to compare against reported eps
DataFrame (Yahoo Finance) [Earnings Estimates]
API Module IBAPI
DataFrame (AlphaVantage) [Financial Statement Data]
Merged Dataframes
Regression Model
predicted EPS
Signal Module signal()
provides signals (BUY, HOLD, SELL) for display.
UI
provides tick_news() function
DataFrame (Interactive Brokers) [News Data (Live)]
reported EPS is fetched to make comparisons against estimated EPS in signal() module
Order Placement Module TestApp()
supplies live_price() function
Live Price Extraction Module TestAppLive
live price is used to make order entries

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.

	P0: conducting comparison between predicted EPS using regression model and actual EPS reported to test model accuracy.
DataFrame (Interactive Brokers) [News Data (Live)]	**P0** : Fetches Reported EPS number from the tickNews() function which provides news in real time
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 using the signal() function)	**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 we are considering to generate a trading signal between buy, sell or do nothing.
Order Placement Module (implemented using the TestApp() function)	**P0**: using the algorithm to make automated trades. **Non-Functional**: generating and executing trades within a reasonable timeframe after detecting earnings anomalies.
Live Price Data Extraction Module (implemented using the TestAppLive() function)	**P0**: connecting to Interactive Brokers livestream API to retrieve live price data to make real time trades.
API Module (Interactive Brokers API)	**P0:** interactive brokers api library to communicate with their platform to fetch information or send orders
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	**P2:** provides an overview of financial metrics and actionable recommendations.

(implemented using PowerBI for reporting and React, Node for webpage)	**P2:** provides historical performance charts (backtest visualization)

Project Components

DataFrame (YahooFinance) [Earnings Estimates]

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.

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

Normal Behaviour: Fetches the news in headline format and provides a constant feed of live information as soon as it's released in parsable string format.

API and Structure: We use the app.reqMktData() function which requires a request ID and a contract object (contract() object specifies the stock and its exchange) and ticknews() function to receive the news headlines as soon as they are released, we need to pass in the reqID parameters which is an integer and the contract.

Implementation: We set the contract ID and the request ID and we call this function under the nextOrderID() function provided by the Interactive Brokers API, and then we use the variable "headline" extracted from this function in real time and pass it on to NLP functions or regular expressions for further use.

Potential Undesired Behaviours: Sometimes there is a possibility that the news feed does not update instantly or has a significant latency which would make it hard to replicate the results.

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 OrderPlacement module to generate a trade.

API and Structure: The component consists of two methods, being signal(stock, vec,estimate) and flipsignal(signal). 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: *C*alculates the surprise by dividing actual earnings*(vec)* by estimated earnings*(estimate)*, generating a "BUY" if it exceeds 1.22, a "SELL" if below 1.04, or "NONE" otherwise. The flipsignal function reverses a signal, allowing flexibility for changing market conditions or adjustments.

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.

Order Placement Module

Normal Behaviour: Places the orders based on the desired signal generated from the signal module. Orders are expected to be placed with minimum latency with no particular conflicts and correct order quantities as specified.

API and Structure: This module extends the Interactive Brokers API by implementing key methods like nextValidId, based on a trading signal, and calls the TestAppLive instance to update live market data and close positions with flipsignal. Order lifecycle management is handled through methods such as openOrder for logging margin details, orderStatus for tracking order progress, and execDetails for execution information.

Implementation: A TestApp which is an instance of an IBAPI is used in which we call the app.placeOrder() for the equity we are trading, we simultaneously call the flipsignal() function in the signal module to reverse the position and close it out within a fixed amount of seconds.

Potenial Undesired Behaviours: If there is a conflict in the orderIds there could be issues and if there is latency in the order placement or if a fill is not ensured outside RTH.

Live Price Data Extraction Module

Normal Behaviour: This component interacts with a broker's API to fetch live stock price data, set it to a global variable named trigger, and store the trading signal and related financial information in a JSON file.

API and Structure: This component uses the Interactive Brokers API module (ibapi) to connect to a broker, fetch live stock price data via reqRealTimeBars, and store the closing price in a global variable, trigger, through the realtimeBar method. The class TestAppLive handles the API requests, such as initializing stock contracts in nextValidId, and processes responses for real-time updates. The fetched price is used to calculate the trading quantity based on preset capital, and trading data (signal, reported EPS, and estimated EPS) is saved in a JSON file for record-keeping

Implementation: The program begins by defining a stock contract (mycontract) and initiating a real-time bar request using reqRealTimeBars. When data is received, the realtimeBar function processes it, extracting the close price and storing it in the global trigger variable. The app instance of TestAppLive connects to the broker's server, runs the event loop, and disconnects after receiving the required data. The live price is then used to calculate the trade quantity (capital // trigger), while the trading signal and related EPS data are stored in an output.json file for further analysis or auditing.

Potenial Undesired Behaviour: The code assumes a stable connection to the brokers API. Incase the connection is not stable it could lead to latency issues and feed disruptions which might not allow us to get the price as quickly as possible.

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 reqRealTImeBars() for the live component , reqMktData() and tickNews() for live news component, PlaceOrder() for order placement component and 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 - 6pm sampled at 15-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 different 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.

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 15 minutes, we would subtract price at 16:20 from 16:05 price and label that as a row in the returns column for this trade instance.

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

User Interface

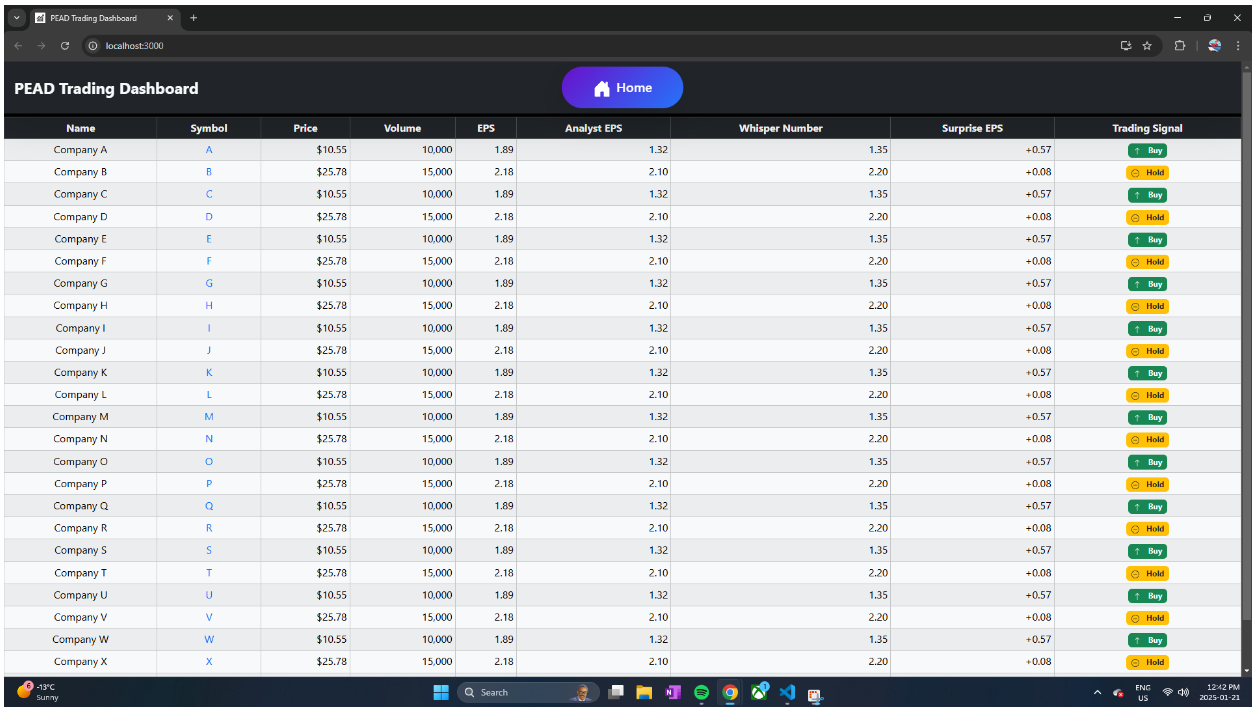
The UI offers two main views: a Dashboard View for real-time trading data and signals, and a Performance Analysis View for historical performance metrics and backtesting. It retrieves real-time data through REST APIs and WebSocket feeds, providing interactive elements for filtering data, adjusting parameters, and visualizing results. Built with React.js, styled using Bootstrap CSS for a responsive design, and featuring Chart.js for dynamic visualizations, the UI includes color-coded signals (Buy, Hold, Sell) and interactive performance charts to enhance user experience. More 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 designed to facilitate seamless interaction with the trading algorithm while ensuring clarity and ease of use for traders. The UI is divided into two main views: the Dashboard View, which provides real-time trading data and actionable signals, and the Performance Analysis View, which offers detailed historical performance metrics.

Dashboard View



PEAD Trading Dashboard

Name	Symbol	Price	Volume	EPS	Analyst EPS	Whisper Number	Surprise EPS	Trading Signal
Company A	A	$10.55	10,000	1.89	1.32	1.35	+0.57	↑ Buy
Company B	B	$25.78	15,000	2.18	2.10	2.20	+0.08	⊖ Hold
Company C	C	$10.55	10,000	1.89	1.32	1.35	+0.57	↑ Buy
Company D	D	$25.78	15,000	2.18	2.10	2.20	+0.08	⊖ Hold
Company E	E	$10.55	10,000	1.89	1.32	1.35	+0.57	↑ Buy
Company F	F	$25.78	15,000	2.18	2.10	2.20	+0.08	⊖ Hold
Company G	G	$10.55	10,000	1.89	1.32	1.35	+0.57	↑ Buy
Company H	H	$25.78	15,000	2.18	2.10	2.20	+0.08	⊖ Hold
Company I	I	$10.55	10,000	1.89	1.32	1.35	+0.57	↑ Buy
Company J	J	$25.78	15,000	2.18	2.10	2.20	+0.08	⊖ Hold
Company K	K	$10.55	10,000	1.89	1.32	1.35	+0.57	↑ Buy
Company L	L	$25.78	15,000	2.18	2.10	2.20	+0.08	⊖ Hold
Company M	M	$10.55	10,000	1.89	1.32	1.35	+0.57	↑ Buy
Company N	N	$25.78	15,000	2.18	2.10	2.20	+0.08	⊖ Hold
Company O	O	$10.55	10,000	1.89	1.32	1.35	+0.57	↑ Buy
Company P	P	$25.78	15,000	2.18	2.10	2.20	+0.08	⊖ Hold
Company Q	Q	$10.55	10,000	1.89	1.32	1.35	+0.57	↑ Buy
Company R	R	$25.78	15,000	2.18	2.10	2.20	+0.08	⊖ Hold
Company S	S	$10.55	10,000	1.89	1.32	1.35	+0.57	↑ Buy
Company T	T	$25.78	15,000	2.18	2.10	2.20	+0.08	⊖ Hold
Company U	U	$10.55	10,000	1.89	1.32	1.35	+0.57	↑ Buy
Company V	V	$25.78	15,000	2.18	2.10	2.20	+0.08	⊖ Hold
Company W	W	$10.55	10,000	1.89	1.32	1.35	+0.57	↑ Buy
Company X	X	$25.78	15,000	2.18	2.10	2.20	+0.08	⊖ Hold

The Dashboard View presents a table-based layout that displays critical trading information in a structured and easily scannable format. The table columns include:

- **Company Name**: Identifies the company associated with the displayed data.
- **Symbol**: The stock ticker symbol for quick reference.
- **Price**: The most recent or real-time price of the stock.
- **Volume**: Trading volume, providing insights into market activity.
- **EPS (Earnings Per Share)**: The latest reported EPS value.
- **Analyst EPS**: Consensus EPS from analysts.
- **Whisper Number**: Predicted EPS based on custom machine learning algorithm.
- **Surprise EPS**: Calculated as the difference between Analyst EPS and EPS Number.

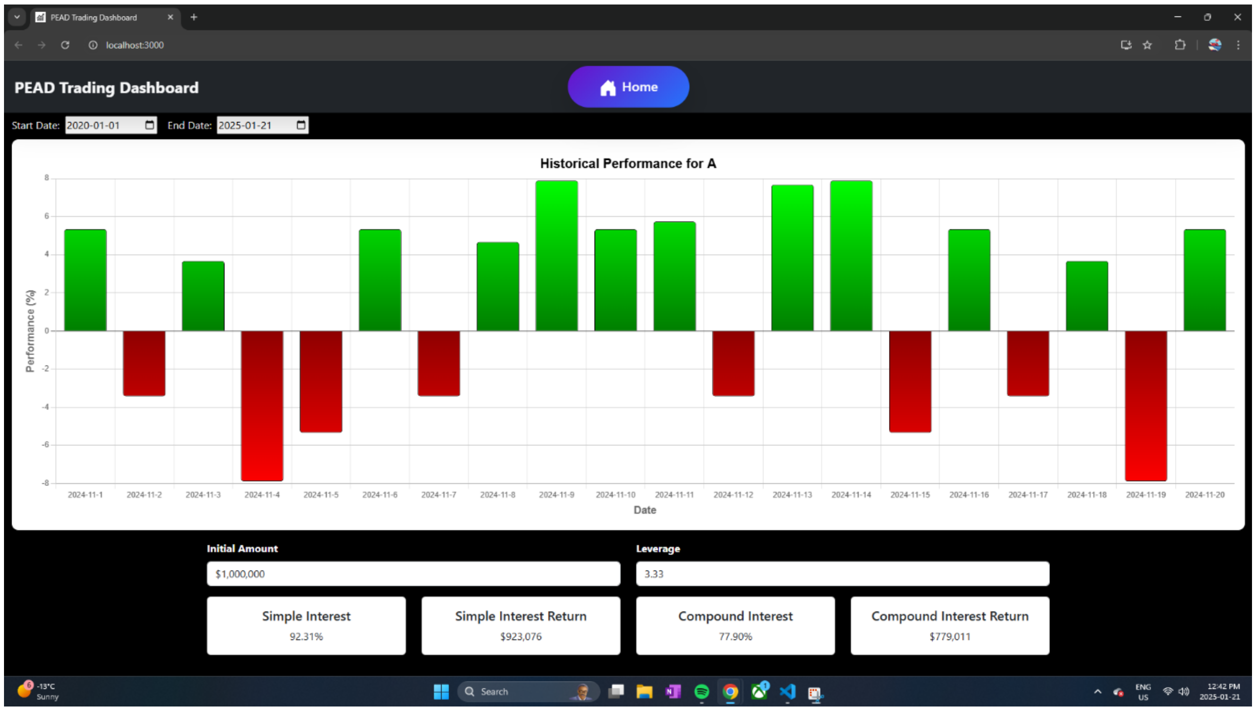
- **Trading Signal**: Displays actionable recommendations (e.g., Buy, Hold, Sell) with visual cues.

The trading signals are highlighted using color-coded buttons for quick decision-making:

- **Green (Buy)**: Indicates a positive trading recommendation.
- **Yellow (Hold)**: Suggests maintaining the current position without immediate action.
- **Red (Sell)**: Advises closing the current position to minimize potential losses or secure profits.

This view prioritizes a simple yet effective structure to minimize information overload and enhance decision-making efficiency.

Performance Analysis View



PEAD Trading Dashboard

Start Date: 2020-01-01 End Date: 2025-01-21

Initial Amount: $1,000,000

Leverage: 3.33

Simple Interest	Simple Interest Return	Compound Interest	Compound Interest Return
92.31%	$923,076	77.90%	$779,011

The Performance Analysis View provides a detailed breakdown of historical trading performance. This view includes:

1. **Performance Chart**:
    - A bar chart visually represents historical trading outcomes for selected companies.
    - **Green bars** indicate days with positive returns, while **red bars** highlight losses.
    - Users can adjust the date range for analysis via interactive selectors.
2. **Metrics and Interactions**:

    - Below the chart, the interface displays:
        - **Initial Investment**: The starting amount for performance calculations.
        - **Leverage**: A multiplier for potential returns or losses.
        - **Simple Interest Return**: A calculation of returns without compounding.
        - **Compound Interest Return**: Reflects returns with compounding included.
    - Inputs for the initial amount and leverage are dynamically adjustable, allowing users to explore different scenarios.

Design and Implementation Details

The UI is implemented using modern web technologies to ensure responsiveness and cross-device compatibility. Key features include:

- **Dark Theme**: Enhances readability and reduces strain during prolonged use.
- **Consistent Layout**: The structured table and chart maintain alignment and clarity.
- **Navigation**: A prominent "Home" button provides quick access to the main page.