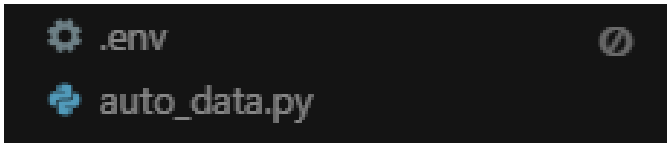
Background

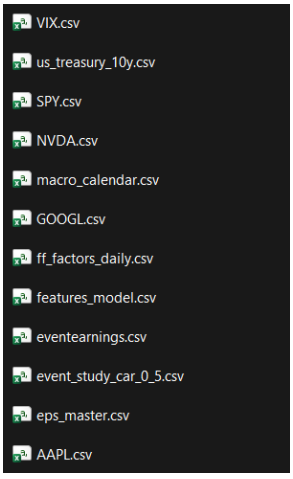
VP Analytics provides data and modelling services to hedge funds that trade around major events. Earnings announcements for large technology companies often lead to sharp moves in prices and trading volumes. Prior research on post-earnings drift shows prices do not always adjust fully on the announcement day. Our project supports VP Analytics by testing these ideas on three large and liquid companies, Apple, Nvidia and Google. We study how prices move before and after earnings, find patterns that repeat across time and names, and turn them into a simple, explainable trading rule for short term post earnings returns.

Project Development Process

Data Acquisition and Cleaning Approach

Our development process began with the construction of a robust, automated data pipeline, auto_data.py, designed to scrape and harmonise data from disparate sources, including Yahoo Finance for price action and Fama-French factors for market benchmarking.





A significant challenge encountered during data cleaning was the temporal misalignment between earnings announcements and trade execution. As earnings are typically reported "After Market Close" (AMC)—roughly 5 PM ET or 10 PM UK time—immediate trading is impossible. To address this, we engineered a specific "Day 0" variable, defined as the first trading session following the announcement. This required a complex merge logic with other csv files: aligning data not just by date, but by unique (Ticker, Day 0) tuples to resolve instances where different companies reported on the same day.

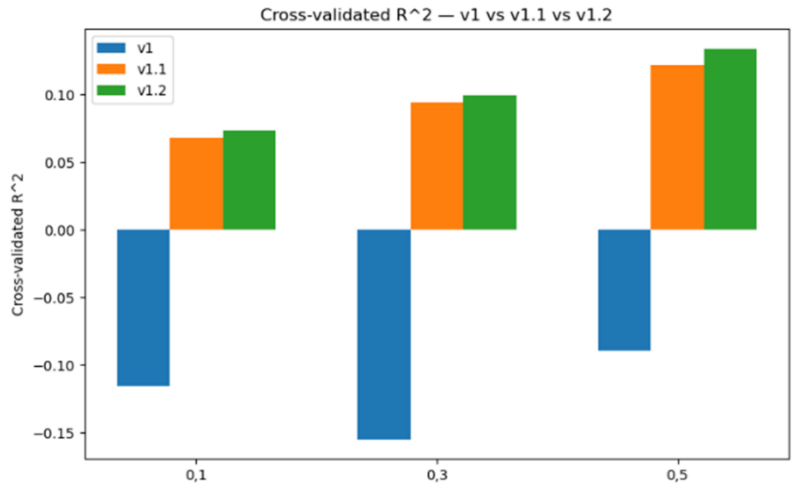
Furthermore, to ensure our model responded to genuine market signals rather than data errors or extreme outliers, we applied a winsorization technique at the 1% level (WINSOR_P = 0.01). This clipped extreme tail values in the return data, ensuring that our cumulative abnormal return (CAR) calculations were robust and not skewed by single anomalous ticks.

```
# Event study settings
EVENT_WINDOW = (0, 5)              # returns at day0..day5
ESTIMATION_LOOKBACK = 120          # -120..-20 trading days
ESTIMATION_GAP = 20                # gap from day0 back to end of estimation
WINSOR_P = 0.01                    # 1% tails for returns
```

Analytical Decisions and Feature Selection

Our initial exploratory analysis identified 44 potential "drivers" of price action, ranging from macroeconomic indicators to idiosyncratic volatility metrics. To prevent overfitting, we employed a rigorous reduction funnel. We established a baseline of 16 features but found they yielded poor predictive power on unseen data.

To refine this, we utilised Recursive Feature Elimination (RFE) and Permutation Tests. These methods allowed us to isolate the "signal from the noise," systematically removing weak features that contributed to model complexity without adding predictive value. This process narrowed our inputs to seven high-impact features.

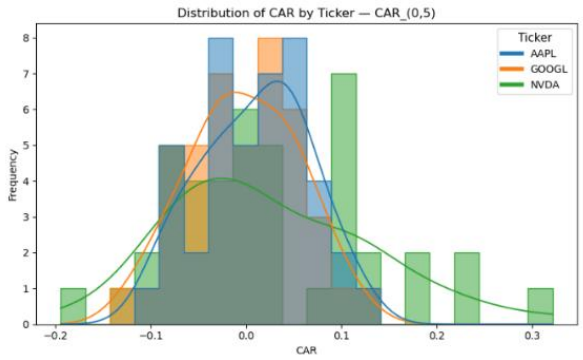


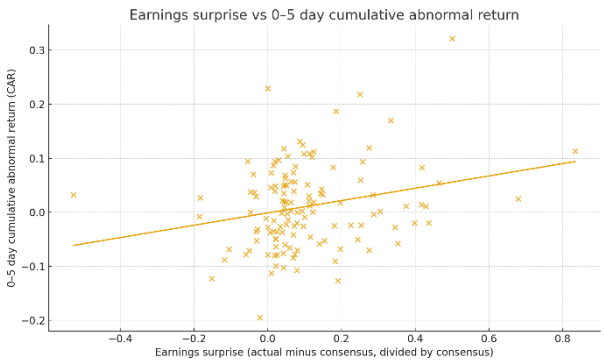
A critical decision in our analysis was the selection of Cross-Validated r2 (CV r2) as our primary success metric, rather than standard r2. Standard r2 often increases artificially as more features are added, creating a false sense of accuracy. In contrast, CV r2 penalizes overfitting and measures how well the model generalizes to new, unseen data. For a hedge fund client, this "future realism" is the "North Star"—a model that looks good on paper but fails in live back testing is a liability; CV r2 protects against this risk.

Visualisation Strategy

Our visualisation choices were driven by the need to identify tradeable patterns and communicate a clear data story to the client.

- Scatter Plots: We utilised scatter plots with trend lines to explicitly map the correlation between independent variables and market reaction. This allowed us to visually confirm linear relationships, giving us the confidence to rely on these patterns for trade execution.
- Histograms: To understand risk, we used histograms to display the distribution of Cumulative Abnormal Returns (CAR). This highlighted the "fat tails" and volatility differences between tickers—specifically showing that Nvidia has a wider, more volatile spread compared to the centred, predictable distributions of Google and Apple.





Technical Overview of the Code

Rationale for Tools and Technologies

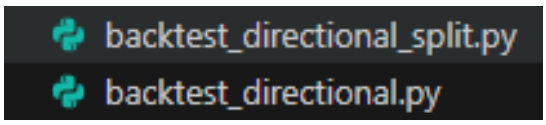
The technical foundation of this project is built on a standard Python data science stack—primarily pandas for data manipulation, numpy for numerical vectorisation, and yfinance for data acquisition.

We deliberately chose to build a custom backtesting engine from scratch rather than utilising established backtesting frameworks like *Backtrader* or *Zipline*. While these frameworks are powerful, they often come with steep learning curves and rigid "opinionated" structures regarding data ingestion and trade execution. Given the specific nuances of our "After Market Close" (AMC) strategy—where the "Day 0" trade entry occurs a full trading day after the announcement—we determined that a custom script would offer superior flexibility. It allowed us to implement bespoke logic, such as the dynamic training window (requiring a minimum of 80 historical events before trading), without fighting against the constraints of a pre-packaged library.

Architectural Design

Modularity and Scalability Our codebase follows a strict "Separation of Concerns" philosophy, divided into two distinct functional blocks:

1. Data Pipeline (auto_data.py): This module handles the Extract, Transform, Load (ETL) process. It manages API connections, cleanses the raw data (handling the complex date alignment for earnings), and calculates the 7 technical features.
2. Strategy Engine (backtest_directional.py): This module is purely responsible for logic execution. It consumes the clean data, runs the regression models, and generates trade logs.



We adopted this modular approach to ensure the solution is future-proof for VP Analytics. By decoupling the data fetching from the strategy logic, the client can easily swap out data sources (e.g., replacing Yahoo Finance with a Bloomberg terminal feed) or adjust strategy parameters (e.g., changing the lookback window or risk thresholds) without risking the integrity of the entire pipeline.

Security and Reproducibility

To ensure the code is both secure and reproducible, we utilised a .env file structure for managing sensitive API keys (AlphaVantage, FRED). This design choice serves a dual purpose: it prevents sensitive credentials from being hardcoded into the script (a security best practice), and it allows the client to seamlessly plug in their own paid data subscriptions by simply updating a local configuration file, ensuring the product remains usable long after hand-off.

Critical Reflection: How Choices Helped and Hindered

The decision to use a custom "vectorised-style" script rather than an event-driven framework significantly helped our development speed. It reduced the debugging time often associated with complex object-oriented frameworks and allowed us to "keep it simple." We could inspect intermediate DataFrames at any step to verify that our winsorization or feature selection was behaving exactly as intended.

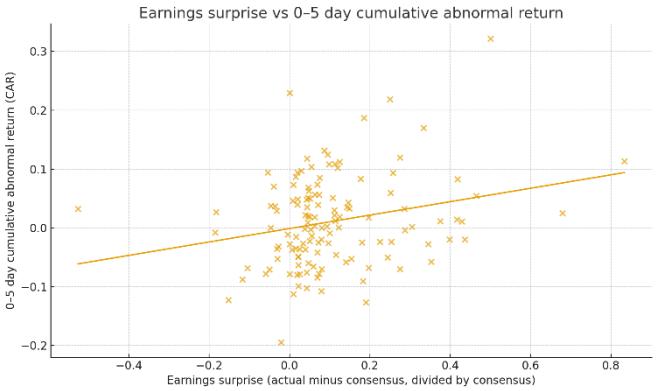
However, this choice also presented specific hindrances. By writing a backtester from scratch, we had to manually implement standard performance metrics that off-the-shelf libraries handle automatically. We had to write our own code to track the equity curve, calculate the Sharpe ratio, and log trade-level PnL. This manual "boilerplate" work was time-consuming and introduced a higher risk of calculation errors compared

to using a battle-tested library like *Pyfolio*. Additionally, ensuring there was no "look-ahead bias" (peeking at future data) placed the burden entirely on our code logic, whereas established frameworks often have built-in safeguards against this. Despite these trade-offs, the transparency and control gained by owning the entire stack were ultimately vital for explaining the precise mechanics of the strategy to the client.

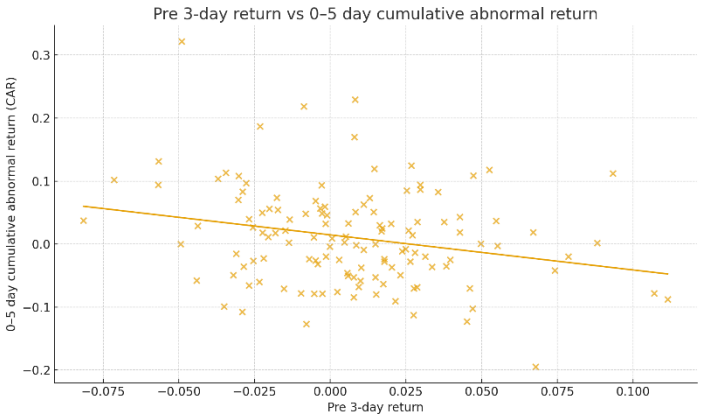
Patterns, Trends, and Insights

Market Dynamics and Predictive Patterns

Our analysis uncovered two primary "drivers" of post-earnings price action. First, we observed a distinct linear relationship between Earnings Surprise and subsequent returns: as expected, companies that beat consensus estimates tend to drift higher over the following five days.



However, a more nuanced insight emerged from the Pre-3-Day Return feature. We identified a negative slope, suggesting a market "correction" mechanism . When a stock rallies aggressively *before* an earnings announcement, it often underperforms *after* the event, likely due to traders "selling the news" or profit-taking. Conversely, stocks that are beaten down prior to earnings often experience a mean-reversion bounce. This confirms that pre-event positioning is just as critical as the event itself.



Strategic Implications: Precision vs. Volume

We developed two distinct strategies to capitalise on these patterns, each serving a different function for a hedge fund portfolio:

- The "Sniper" (Strategy 2): Utilising only the two strongest signals (Surprise & 3 day Pre-Return), this model achieved an exceptional 85% hit rate and a Sharpe ratio of 1.57. However, it is low-frequency, triggering only 14 trades in 7 years. It offers precision and capital preservation with virtually zero drawdown.
- The "Workhorse" (Strategy 1): By including volatility and macro factors (7 features), this model trades more frequently (42 trades). While the Sharpe ratio drops to ~1.0, it generates significantly higher total PnL ($522k vs $261k) .

```
Number of trades: 42
Total PnL: $522,119.48
Average PnL per trade: $12,431.42
Median PnL per trade: $8,842.86
Std dev PnL per trade: $25,449.63
Hit rate: 0.667
Worst trade: $-37,050.10
Best trade: $92,406.74

Trades by size:
  $200,000: 17 trades
  $400,000: 3 trades
  $600,000: 22 trades

Return per trade as % of position:
Average: 2.32%
Median:  2.25%
Std dev: 6.26%
Worst:   -12.27%
Best:    18.01%

Hit rate: 66.7% of trades are profitable

Portfolio view assuming $10,000,000 allocated to this strategy:
Total PnL as % of book: 5.22%
Trading period: 2016-11-11 to 2025-07-31 (~8.72 years)
Trades per year: 4.82
Approx annualised return on the book: 0.59%

Equity curve on 10m book:
Final capital (starting from 1.0): 1.053426
Maximum drawdown: -0.004010
Approx annual volatility: 0.005586
Approx annual Sharpe: 1.048

Per-tier average return (as % of position):
  Size $200,000: n=17, avg = 0.71%
  Size $400,000: n=3, avg = 2.24%
  Size $600,000: n=22, avg = 3.57%
```

```
Number of trades: 14
Total PnL: $261,017.71
Average PnL per trade: $18,644.12
Median PnL per trade: $13,982.38
Std dev PnL per trade: $16,700.70
Hit rate: 0.857
Worst trade: $-1,500.84
Best trade: $51,908.25

Trades by size:
  $200,000: 9 trades
  $400,000: 2 trades
  $600,000: 3 trades

Return per trade as % of position:
Average: 5.72%
Median:  5.90%
Std dev: 4.67%
Worst:   -0.75%
Best:    15.40%

Hit rate: 85.7% of trades are profitable

Portfolio view assuming $10,000,000 allocated to this strategy:
Total PnL as % of book: 2.61%
Trading period: 2016-11-11 to 2024-02-29 (~7.30 years)
Trades per year: 1.92
Approx annualised return on the book: 0.35%

Equity curve on 10m book:
Final capital (starting from 1.0): 1.026402
Maximum drawdown: -0.000296
Approx annual volatility: 0.002313
Approx annual Sharpe: 1.529

Per-tier average return (as % of position):
  Size $200,000: n=9, avg = 5.72%
  Size $400,000: n=2, avg = 3.49%
  Size $600,000: n=3, avg = 7.24%
```

Recommendations and Next Steps

We recommend implementing this system as a specialised "earnings sleeve"—a satellite strategy rather than a core driver.

1. Deploy Both Models: Use Strategy 2 for high-conviction, levered allocation, and Strategy 1 to capture broader volume.

2. Optimise the Universe: The model naturally avoided Apple (only 1 trade), identifying it as less reactive compared to the high-volatility opportunity in Nvidia. We recommend replacing stable tickers like Apple with more volatile names to increase trade frequency.
3. Forward Testing: Immediate next steps are to run the model live on small size for 12–18 months. If the "correction" behaviour holds in real-time, capital should be scaled gradually .