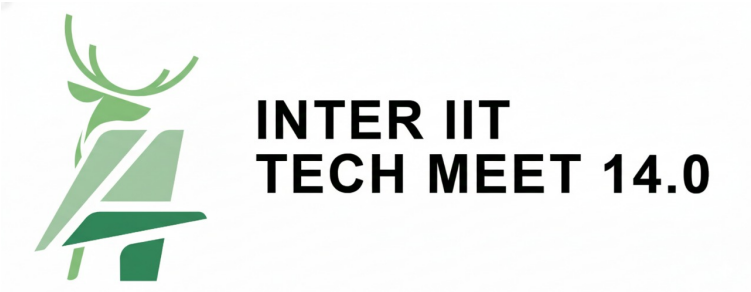
Algorithmic Strategy Development on Multi-Feature Time Series

INTER IIT TECH MEET 14.0

Mid Term Submission Report

Team 33



Contents

1 Introduction **2**

2 Problem Description **2**

3 Experimented Approaches **2**
3.1 Mean Reversion of Residual White Noise 2
3.2 Feature-Based Threshold Strategy 2
3.3 Band-4 PV3 Causal Slope Gate (with Adaptive Smoothing) 3
3.4 Opening Range Breakout (ORB) with Volatility Confirmed Whipsaw Filter 3
3.5 PB10/11 and CUSUM-Based Regime Signal 3

4 Current Strategy **4**
4.1 Data Preparation and Label Generation 4
4.1.1 Kalman Filter Labeling 4
4.1.2 Regime-Based Feature (CUSUM) & Other Feature Combinations 4
4.1.3 Training Data Construction 5
4.2 Model Training 5
4.3 Candle-Based Consensus Strategy 6
4.3.1 Entry and Exit Logic 6
4.4 Performance Metrics 7

5 Code Guide **7**

1 Introduction

In modern financial markets, speed and precision determine profitability. Algorithmic trading allows systematic strategies to process high-frequency, multi-feature time series data and respond intelligently to shifting market conditions. This project, "Algorithmic Strategy Development on Multi-Feature Time Series," aims to design a robust, data-driven trading framework capable of adapting to volatility and maintaining consistency across different instruments. Rather than predicting price movements, our approach focuses on quantifying and exploiting repeatable intraday patterns to generate sustainable returns.

2 Problem Description

The project focuses on developing a systematic intraday trading strategy using two anonymized datasets, EBX and EBY, each representing a distinct but related market instrument. Both datasets contain second-level time-series data with a wide range of masked features covering price, volatility, volume, and alternate signals.

Our aim is to design a strategy that identifies and exploits meaningful intraday patterns while ensuring all trades are opened and closed within the same trading day. The approach emphasizes adaptability across both datasets, maintaining consistent performance even when market structures or data sequences vary, in order to build a framework that remains reliable and robust under realistic market conditions.

3 Experimented Approaches

3.1 Mean Reversion of Residual White Noise

This strategy assumes that the residual noise obtained by finding the difference between the actual price and predicted price by a machine learning model shows a mean-reverting behavior, as discussed in the study by Jung, Jaehyun, et al. Within each rolling window, the linear regression model is fitted on the Principal Components, to predict the current price. We define residual noise as the deviation between actual and predicted values. The **residual noise** $\mathbf{r(t) = y(t) - \hat{y}(t)}$, where $y(t)$ is the price at time t and $\hat{y}(t)$ is the predicted price at time t. This residual captures short-lived mispricings that tend to mean-revert intraday. A smoothed noise signal (via EMA) triggers trades — long when strongly negative, short when positive, and exits as it reverts toward zero.

Basic risk controls such as transaction costs, daily stop-loss, and cool-downs are applied. The model adapts through periodic refits and is evaluated on intraday EBX and EBY data, showing that structured residual noise can uncover consistent mean-reverting opportunities. However, the system generated few trades, mostly early in the prediction window; frequent PCA and Linear regression refits reduced residual variance, aligning predicted and actual prices too closely and limiting exploitable deviations during the day.

3.2 Feature-Based Threshold Strategy

This approach explored engineered feature interactions combining Band-Based **(BB9_T6)**, Volatility-Based **(VB4_T6)**, and Price Based **(PB4_T8)** indicators. The composite feature is defined as:

$$\mathrm{BB9}_{T6} \times \mathrm{VB4}_{T6} \times \mathrm{PB4}_{T8}$$

It showed strong spike-based reactions to intraday trend shifts, producing clear but dataset-sensitive entry signals. Dynamic thresholds and stop-loss variations were tested, but performance suffered due to inconsistent spike density across sessions. To stabilize signal behavior, the feature was modified as:

$$\sqrt{\mathrm{BB9}_{T6} \times \mathrm{VB4}_{T6}} \times \mathrm{PB4}_{T6}$$

This scaling reduced volatility and improved threshold uniformity. The modified version achieved around a 70% win rate during stable market periods, though performance degraded on strongly trending days where noise increased and the stop-loss logic cut several profitable trades.

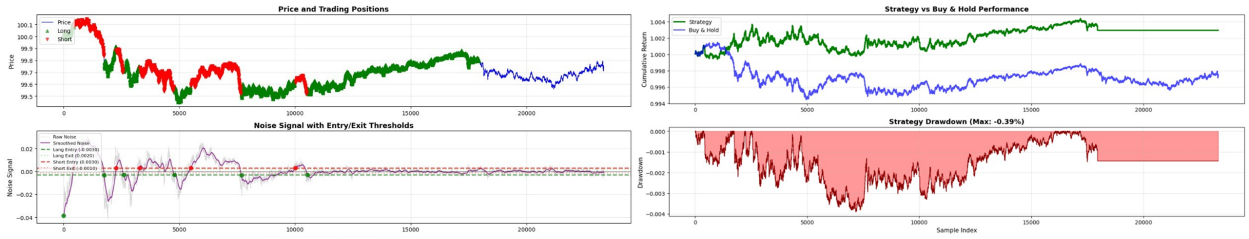


Figure 1: Example showing mean-reverting behavior of the residual noise. The noise tends to be high during the initial phase due to low linear model accuracy and noise becomes very low as the accuracy increases.

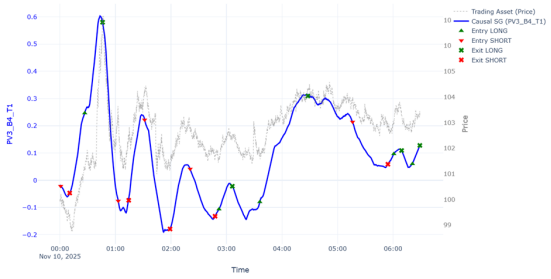


Figure 2: PV3_B4 Trading Signals

Figure 3: Composite feature discussed in Section 3.2

3.3 Band-4 PV3 Causal Slope Gate (with Adaptive Smoothing)

This strategy exploits the slope of a causally smoothed **PV3_B4_T1** signal, with Band-4 chosen for its consistent positioning in the middle of the price range on most days. The approach captures sustained impulsive movements, where the smoothed PV3 signal tends to lead price action.

We use an adaptive threshold based on the rolling standard deviation of the signal, requiring a confirmation streak for valid entries and exits, while incorporating **hysteresis, deadband arming, minimum spacing, cooldown,** and minimum holding periods to reduce churn and false signals.

The strategy strikes a balance between smoothing and lag: higher smoothing reduces noise but introduces some delay, while lighter smoothing reacts more quickly but may allow noise.

3.4 Opening Range Breakout (ORB) with Volatility Confirmed Whipsaw Filter

This strategy builds on a Opening Range Breakout (ORB) framework discussed in Tarsen, Arjun, et al., where the high and low from the first 15 minutes define a key intraday range. After a 60-minute observation period, a breakout beyond this range signals a directional trend, while days that break both extremes are classified as "whipsaw" and avoided.

A volatility-based filter using the VB1_{T3} feature identifies these whipsaws - sharp, directionless movements marked by spikes in volatility. By skipping such days, the system aims to avoid unpredictable market phases and improve its risk-adjusted returns. Strict intraday risk controls include take-profit, trailing stop loss and daily loss cap, acting as a circuit breaker.

This method enhances the classic ORB's robustness by adapting to volatility and focusing on high-confidence, directional setups, aligning with the goal of building a systematic, risk-aware intraday strategy.

3.5 PB10/11 and CUSUM-Based Regime Signal

The features $\textbf{PB10}_{T7}$ and $\textbf{PB11}_{T7}$ were identified as key indicators representing the maximum and minimum price levels over specific lookback periods, making them ideal for detecting regime shifts. To quantify the directional strength of price movements and generate trading signals, a **CUSUM (Cumulative Sum)**-based

model was implemented. This model uses cumulative sums of indicator functions, which track price movements relative to predefined thresholds, to generate buy, sell, or hold signals.

The indicator functions at time t, $h_{10,t} = \mathbb{1}(P_t \geq \text{PB10}_{T7,t})$ and $h_{11,t} = \mathbb{1}(P_t \leq \text{PB11}_{T7,t})$, where $\mathbb{1}(\cdot)$ is the indicator function, track price movements relative to thresholds $\text{PB10}_{T7,t}$ and $\text{PB11}_{T7,t}$, respectively. Then the cumulative sums for positive (γ_t) and negative (ζ_t) price movements are computed as $\gamma_t = \max(0, \gamma_{t-1} + h_{10,t})$ and $\zeta_t = \min(0, \zeta_{t-1} - h_{11,t})$. The regime signal is then generated as:

$$\text{Regime}_t = \begin{cases} 1 \text{ (bullish)}, & \text{if } \gamma_t \geq 2 \\ -1 \text{ (bearish)}, & \text{if } \zeta_t \leq -2 \\ 0 \text{ (neutral)}, & \text{otherwise} \end{cases}$$

Although effective in detecting regime shifts, the **CUSUM** model experienced a delay in regime transitions during strong one-sided trends. This was due to the cumulative sum accumulating large values over extended periods, causing the model to lag in detecting regime reversals. To address this issue, a **CUSUM limit** was introduced to cap extreme values, preventing the model from over-accumulating during persistent trends. Furthermore, a **longer lookback-based reset mechanism** using PB10_{T8} and PB11_{T8} was incorporated, which resets the cumulative sum to zero when the threshold is crossed in the opposite direction.

4 Current Strategy

Our current strategy leverages feature engineering, machine learning, and a novel candle-based execution system to trade intraday. The core pipeline consists of four main stages:

- **Data-Driven Labeling:** A Kalman Filter is used to smooth the price data and generate Long/Short-/Hold labels, which serve as the ground truth for the machine learning model.
- **Feature Engineering:** A rich feature set is created, combining time-lagged market data, CUSUM-based regime detection, cyclical time features, and custom indicators.
- **Machine Learning Prediction:** An XGBoost classifier is trained on time-series sequences of these features to predict the Long/Short/Hold signals.
- **Candle-Based Execution:** Model predictions are aggregated over fixed-time candles (e.g., 20-25 minutes). Trades are executed based on the consensus of predictions within each candle, subject to a strict set of risk management rules.

4.1 Data Preparation and Label Generation

4.1.1 Kalman Filter Labeling

The primary purpose of the Kalman filter is to estimate the underlying price trend by filtering out noise in the observed price data. We use this estimation to compute the slope of the trend, which is then used to generate buy and sell signals. The Kalman filter smooths the price at any time t as $\hat{P}_t = \hat{P}_{t-1} + K_t(P_t - \hat{P}_{t-1})$, where K_t is the Kalman gain. The slope is computed as $\Delta P_t = \hat{P}_t - \hat{P}_{t-T}$, and the standard deviation of the slope is $\sigma_{\Delta P} = \sqrt{\frac{1}{T}\sum_{i=1}^{T}(\Delta P_i - \bar{\Delta P})^2}$.

The thresholds are calculated as Upper Threshold (α) = ($\sigma_{\Delta P} \times \mu$) and Lower Threshold ($\beta$) = $-(\sigma_{\Delta P} \times \mu)$, where μ is a multiplier. Signals are generated as Buy Signal = 1 if $P_t > \alpha$ and Sell Signal = -1 if $P_t < \beta$

4.1.2 Regime-Based Feature (CUSUM) & Other Feature Combinations

The **CUSUM (Cumulative Sum) regime signal,** Regime_t, discussed in Section 3.5, was incorporated as one of the features for training, as it tracks short-term price fluctuations accurately and adapts to broader market shifts promptly, thereby improving overall trading decision-making.

To account for the cyclical nature of time, we introduce sinusoidal transformations. These features capture daily and hourly cyclic patterns in the data, encoding the time of day in a way that the model can learn from. For hours h and minutes m, we compute the following time features:

$$H_{\sin} = \sin\left(\frac{2\pi h}{24}\right), \quad H_{\cos} = \cos\left(\frac{2\pi h}{24}\right), \quad M_{\sin} = \sin\left(\frac{2\pi m}{60}\right), \quad M_{\cos} = \cos\left(\frac{2\pi m}{60}\right)$$

In addition to the basic features, we add features based on interactions between multiple existing features, as discussed in Section 3.2. The following custom features are introduced:

$$F_1 = \sqrt{\text{BB9}_{T6} \times \text{VB4}_{T6}} \times \text{PB4}_{T6}, \quad F_2 = \frac{\text{BB4}_{T3}}{\text{BB4}_{T6} + \epsilon}, \quad F_3 = \frac{\text{BB4}_{T2}}{\text{BB4}_{T6} + \epsilon}$$

4.1.3 Training Data Construction

The feature vector $\mathbf{X}_t$ at each time step t consists of engineered features discussed in Section 4.1.2, as well as other price, band, and volatility-based features (including volume and price-volume in EBX) up to the horizon level $T6$.

The feature vector $\mathbf{X}_t$ incorporates data from the past T time steps (lookback period). The shape of $\mathbf{X}$ is $(n_{\text{samples}}, T \times n_{\text{features}})$, where n_{samples} is the number of training examples, T is the lookback period (which is 30 seconds in our case), and n_{features} is the number of features at each time step. The target vector $\mathbf{y}$ contains trading signals, encoded as (+1/-1/0) corresponding to Long, Short, and Hold at time step t.

To handle outliers and maintain meaningful consistency across features, the **RobustScaler** is employed. This method scales data based on the Interquartile Range (IQR), making it resistant to extreme values. For an unscaled feature $\mathbf{X}$, the scaled value $\mathbf{X}'$ is calculated as:

$$\mathbf{X}' = \frac{\mathbf{X} - Q_2(X)}{Q_3(X) - Q_1(X)}$$

where $Q_1(X)$, $Q_2(X)$, and $Q_3(X)$ are the 25th, 50th, and 75th percentiles (median) of the feature $\mathbf{X}$, respectively. The denominator $Q_3(X) - Q_1(X)$ is the Interquartile Range (IQR).

Table 1: Model Performance Metrics by Class (Prediction on Test Set)

Metric	**EBX Model**			**EBY Model**		
	Buy Long	**Buy Short**	**Hold**	**Buy Long**	**Buy Short**	**Hold**
Precision	0.68	0.64	0.68	0.60	0.60	0.61
Recall	0.64	0.65	0.73	0.75	0.48	0.44
F1-Score	0.63	0.65	0.70	0.67	0.53	0.51
Overall Accuracy		0.6604			0.6036	

4.2 Model Training

The model is trained using the **XGBoost** algorithm with the following key parameters: the objective function is set to **multi:softprob** for multi-class classification, with 3 classes (Buy, Sell, Hold). The evaluation metric used is mlogloss, and the maximum depth of trees is set to 3. The learning rate is 0.04, with 1200 boosting rounds, and both subsample and colsample by tree are set to 0.7. The gamma parameter is set to 0.1 to control tree pruning. Table 1 summarizes the model's performance. Figures 4 and 5 show the curves for the training and validation loss over the boosting iterations.

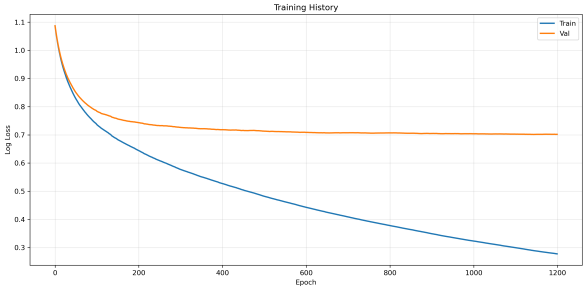


Figure 4: EBX Model Training Curve

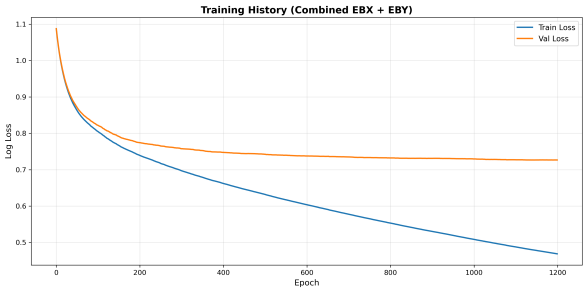


Figure 5: EBY Model Training Curve

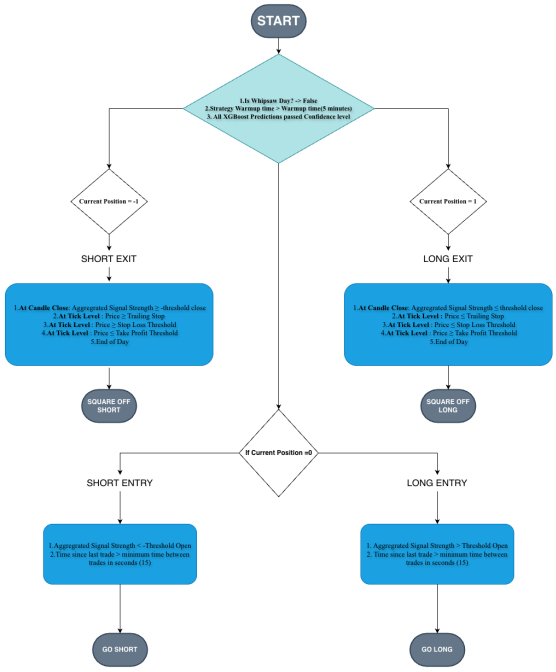


Figure 6: The flowchart explains the Entry and Exit logic of our candle-based consensus strategy

4.3 Candle-Based Consensus Strategy

Initially, the model made high-frequency predictions every 4 seconds, which led to frequent entries and exits. At such short time intervals (4 seconds), the market can exhibit small fluctuations that do not always reflect the true trend. Since the model made too many trades, significant transaction costs eroded the potential profits. To address this, we define a candle-based strategy for entry and exit logic, which aggregates predictions over a predefined time period (or "candle" period) rather than making decisions based on individual predictions. We define an implicit candlestick with the opening price as Price_{t-T} and the closing price as Price_t, where T is the candle period (for EBX, we set $T = 20$ minutes, and for EBY, we set $T = 25$ minutes).

4.3.1 Entry and Exit Logic

The entry decision is based on the aggregation of the long and short signals. We calculate the signal strength of a candlestick as the difference between the count of long and short predictions: $\Delta = \mathbb{1}(\hat{y}_k = 1) - \mathbb{1}(\hat{y}_k = -1)$, where $\hat{y}_k$ is the model's prediction at time k within the duration $[t-T, t]$, and $\mathbb{1}$ is the indicator function. We Enter Long Position if $(\Delta > 10)$ and Enter Short Position if $(\Delta < -10)$. The entry is always taken at the closing price of the candlestick to avoid false signals. Once Position $\neq 0$, we check the exit condition: we Exit Long Position if $(\Delta < 15)$ and Exit Short Position if $(\Delta > -15)$. This condition ensures that once we lose confidence in a position during a trade, we exit to avoid losses. The exit rule is further improved by incorporating a take profit of 0.4% and a trailing stop loss of 0.35%.

The Entry and Exit Logic is described in Figure 6. Figure 7 demonstrates that the Candle-Based Consensus Strategy reduces noise and transaction costs by aggregating predictions into candles, leading to more robust decision-making and improved performance compared to trading with individual predictions.

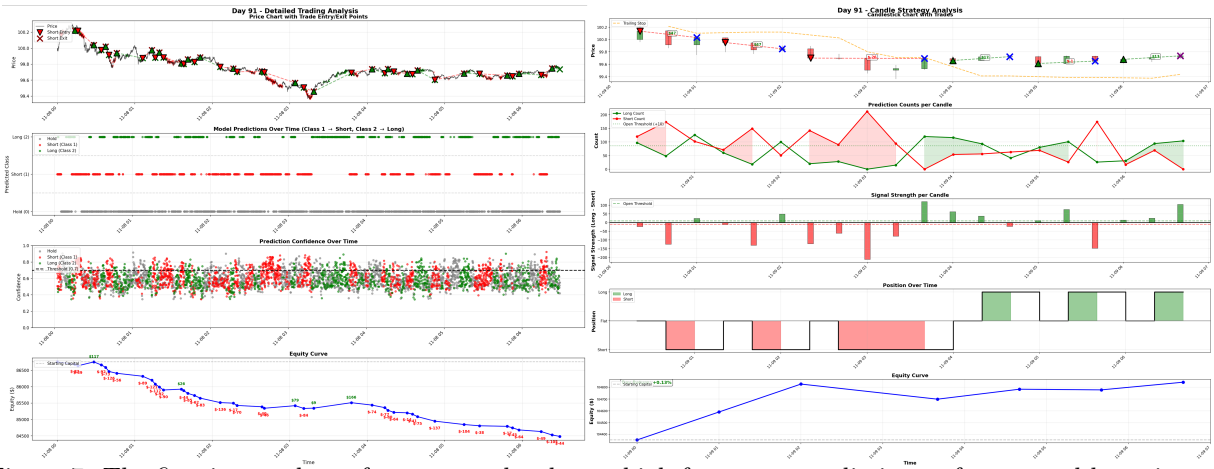


Figure 7: The first image shows frequent trades due to high-frequency predictions, often caused by noise and rapid regime shifts. The second plot, using candle-based aggregation, smooths the signals over a larger time window, reducing the number of trades.

4.4 Performance Metrics

Out of the given 510 days for EBX and 279 days for EBY, the strategy was only traded on 252 days for EBX and 126 days for EBY. These trading days were selected at random, and the XGBoost model was trained on the remaining days to ensure a generalized model. Table 2 is the table summarizing the performance metrics for EBX and EBY.

Table 2: Performance Metrics for EBX and EBY Models

Ticker	Days	Final Equity	PnL (%)	Max DD (%)	Sharpe	Calmar	Ann. PnL (%)
EBX	252	126 004.783	26.004	2.047	4.718	12.698	26.00
EBY	126	114 082.885	14.082	1.122	4.075	12.543	28.17

5 Code Guide

The strategy's codebase is designed with modularity, efficiency, and command-line execution in mind to ensure reproducibility and robustness. The architecture relies on three distinct operational commands—train, test, and backtest—all executed via the Command Line Interface (CLI). This modular approach ensures that feature calculation, model training, signal generation, and backtesting are handled separately, promoting a clean separation of concerns. Additionally, external configuration files centralize hyperparameters, ensuring that each run is fully documented and auditable.

- **Train the model for EBX and EBY:** Prepares data, applies feature engineering, and trains an XGBoost model.

```
python <ticker>.py train --days 250 --config config_<ticker>.json
python <ticker>.py train --days MODELS/train_days_<ticker>.txt
                  --config config_<ticker>.json
```

- **Generate signals on the test set:** Generates trading signals based on the trained model.

```
python <ticker>.py test --config config_<ticker>.json
python <ticker>.py test --days MODELS/test_days_<ticker>.txt
                   --config config_<ticker>.json
```

- **Run backtests for EBX, EBY, or Both:** Backtests on the provided backtester module using the generated signals, calculates metrics, and outputs results.

```
python <ticker>.py backtest_ebullient --config config_<ticker>.json
```