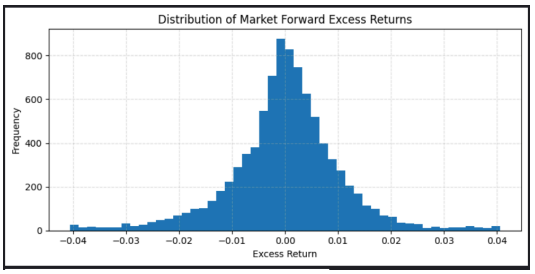
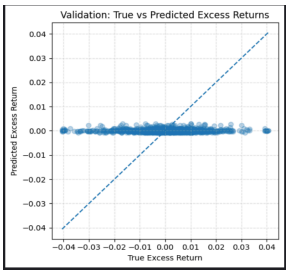
Time-series prediction problem: Hull Tactical Market Prediction

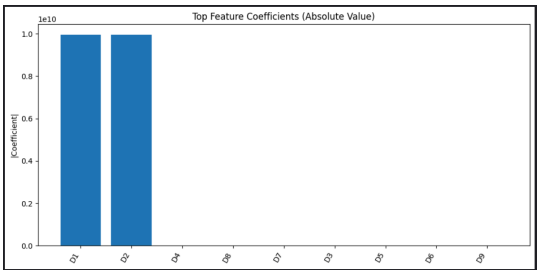
Step 1. Baseline Model(Public score: 0.425/-0.120)

First are two simplest baseline models to predict market excess returns. The former is a linear approach that directly predicts future excess returns by training a simple regression on all numerical variables. The latter is a momentum-based approach that estimates returns by using the previous day's return directly without prediction. Neither model uses distinctive indicators or considers nonlinear patterns or volatility structures.

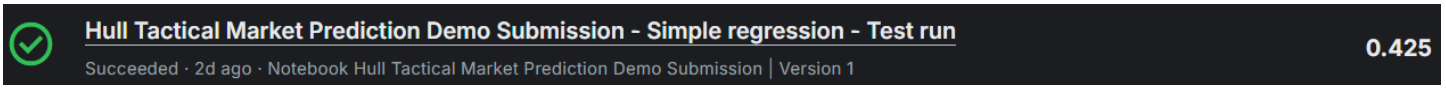
The linear regression model set market_forward_excess_returns as the target during training and used all numerical variables as input features, excluding ID-related columns and the target column. Preprocessing step applied scale normalization to prevent regression coefficient distortion caused by differences in variable units. The momentum model does not perform training. It directly uses the simple momentum value calculated from lagged_market_forward_excess_returns or the (previous day's return)-(risk-free rate) as the prediction value. Both models linearly scaled the predicted excess returns to convert them into allocations between 0 and 2. This ensured that the position changed based on the direction of the return relative to a neutral weight of 1.



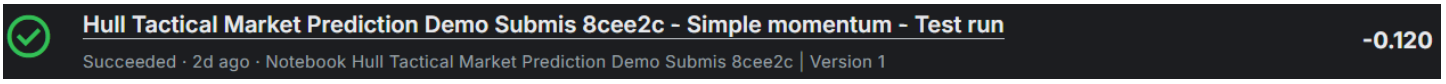




The linear regression model saw most predicted excess returns clustered near zero, causing allocation values to remain largely centered around 1. Overall profit-generating opportunities were limited with a score of 0.425.



The momentum-based model failed to capture market direction effectively. Instead, it tended to react to noise, causing excessive shifts in investment weight and amplifying losses. Its final score was very low at -0.120.



This showed that the absence of a nonlinear structure in market data, time-series characteristics, investment suppression intervals, and risk control mechanisms were the primary causes of performance degradation.

Step 2. Model Development (Public score: 5.567)

The XGBoost regression model was applied to the baseline model. Data from recent 800 days in train.csv was used to ensure the model focuses on the latest information. Input variables features were extracted from columns starting with M, E, I, P, V, S(representing financial and factor characteristics), using only those with a missing rate below 50%. StandardScaler was applied to reduce scale differences between variables and ensure regression prediction stability.

During model training, a Walk-forward approach using TimeSeriesSplit cross-validation was applied. This structure, training on past data and validating on future intervals, prevented data leakage and simulated the predictive flow of a real financial investment environment. The model with the highest correlation coefficient across all 5 folds was selected as the Best Model. XGBoost regression model was used with tree-based structure and low learning rate, applied to ensure smooth function estimation and market sensitivity.

Prediction results were calculated based on the regression model's excess return forecast values. The position = clip(pred * 400, 0, 2) rule was applied to dynamically generate investment weights between 0 and 2. To limit risk, VOL_SCALE was applied by comparing the strategy's return volatility with the benchmark's volatility within the in-sample interval based on the model predictions, ensuring the strategy's volatility did not exceed

120% of the benchmark's. The final submission score was 5.567, achieving reasonably meaningful performance compared to a simple regression model baseline.

Step 3. Feature Engineering

To better predict the daily returns of the S&P 500 Index, we added the following features:

day_of_cycle, {base feature}_lag_1, {base feature}_lag_5, {base feature}_roll_mean_{window}, {base feature}_roll_std_{window}, {base feature}_ema_{window}, feature_div, feature_minus lagged_forward_returns, lagged_risk_free_rate, lagged_market_forward_excess_returns

{base feature} refers to the first features M1, E1, P1, V1, S1, and D1 in the default M*, E*, P*, V*, S*, and D* provided in the train.csv file. Only the first feature was used because, in anonymized financial data, the first variable, such as 1 or 0, is often the principal component. Furthermore, calculating lag_1, lag_5, roll_mean, roll_std, and ema for all base features would add hundreds of variables, potentially leading to overfitting. So only feature 1 of each base feature was used. {window} indicates how many days of data will be used to calculate the mean, standard deviation, and moving average of each feature.

- day_of_cycle: The stock market is basically open for five days from Monday to Friday. To make data_id more meaningful, values divided by 5 were added as features.

- {base feature}_lag_1, {base feature}_lag_5: lag_1 represents the base feature values from one day ago, and lag_5 represents the base feature values from one week ago (since the stock market is open five days a week). These values were added because yesterday's and last week's features can influence today's values.

- {base feature}_roll_mean_{window}, {base feature}_roll_std_{window}: The mean and standard deviation were added as features for each window size (5, 10, 20).

- {base feature}_ema_{window}: The exponential moving average was calculated for each window size (10, 30, 60) and added as a feature. Since it's widely used as a supplementary indicator alongside the mean and standard deviation in real-world stock market applications, we believed it would be helpful in market prediction.

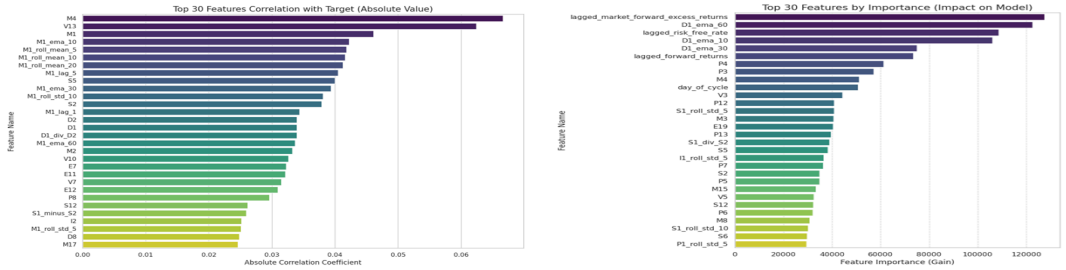
feature_div, feature_minus: Since most feature engineering uses only the first feature, we added feature_div and feature_minus by performing divide and subtract operations between feature1 and feature2 to demonstrate the correlation between other features of the same type.

- lagged_forward_returns, lagged_risk_free_rate, lagged_market_forward_excess_returns: The forward_returns, risk_free_rate, and market_forward_excess_returns features, which were only present in the train dataset, were added using shift(1) to align them with the test dataset.

The following features were excluded from learning.

data_id, forward_returns, risk_free_rate, market_forward_excess_returns feature, is_scored

Date_id was removed because it had already been converted to features including day_of_cycle. Similarly, the **forward_returns, risk_free_rate, and market_forward_excess_returns** features were also removed because they were added with the suffix aged_. Finally, **is_scored**, a feature that only exists in the test dataset, was removed because it was not present in the train data.

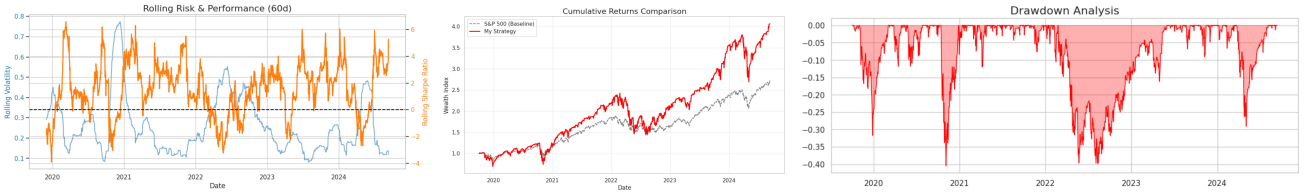


Left graph: We performed EDA on the correlation between each feature, including the added features and market_forward_excess_returns. The added features such as ema, roll_mean, lag, roll_std, and div, have a significant impact.

Right graph: We visualized how each feature influences the results (accuracy) in the LightGBM model. We can also see that the added features have a significant impact on the model's results.

Step 4. Evaluation and Backtesting(Public score: 9.781)

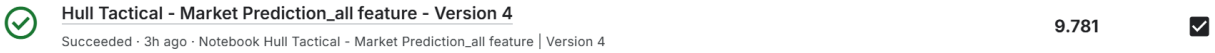
The results evaluated with the valid set (20% of train.csv) are as follows.



Metric	Benchmark (S&P500)	My Strategy
Sharpe Ratio	0.8841	0.8151
CAGR (Return)	15.05%	21.63%
Volatility (Risk)	17.62%	29.33%
Max Drawdown	-40.47%	-24.13%

During the validation period, the S&P500(Benchmark) recorded a Sharpe ratio of 0.8841. In contrast, the implemented model's Sharpe ratio was 0.8151. The model's Sharpe ratio is slightly lower than the benchmark because it employed an aggressive leverage strategy to generate profits. Although the risk-adjusted return was slightly lower, this is interpreted as a structural result of the aggressive investment allocation.

Compared to S&P 500, the CAGR(Compound Annual Growth Rate) increased by about 6%. This shows the effectiveness of the model's strategy of accurately capturing upward trends & increasing allocation.
The values for risk indicators, Volatility and Max Drawdown, demonstrate an aggressive, high-risk, high-reward approach. Since each value is 1.5 times higher than the S&P 500, this suggests that there were instances of failed bets during periods of high volatility.

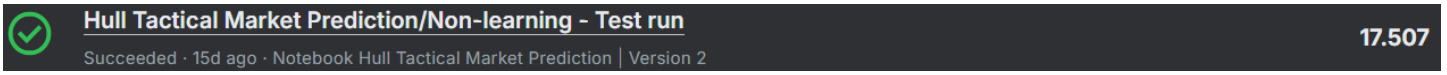


In conclusion, we were able to confirm that the model implemented through each evaluation index is optimized for maximizing profit rather than stability. The Kaggle public score also reached 9.781, indicating that feature engineering increased the public score by 1.75 times compared to Step 2's public score of 5.567.

Step 5. Final Model & Report(Public score: 10.109)

This model employs an ensemble structure where a single input passes through three distinct prediction modules: XGBoost, LGBM, and GBM. Results are weighted averaged to produce the final value. The first block generates various derived features for time-series and indicator-type variables, such as time lags, rolling averages/standard deviations, EMA, ratios, and differences. It then trains a binary classification model and converts prediction probabilities into weights. The second block is an auxiliary module that uses the same preprocessing results to train a regression model specifically on the most recent period. It then passes the predicted signal through a simple threshold-based postprocessing function to determine its weight. The third block trains another regression model with dedicated preprocessing and scaling. It similarly converts the output into a weight between 0 and 2 via a postprocessing function, creating an independent signal. These three signals are then combined according to predefined weights and used as the final output.

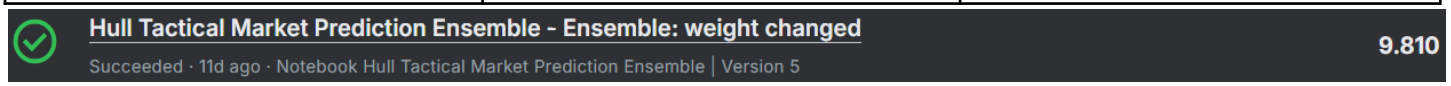
Previously, Time Series Cross Validation was applied but models without TSCV achieved better scores than those with it, so it is omitted in the final code. This likely indicates that by avoiding TSCV(which prevents overfitting), the one-shot single training captured even the subtle patterns optimized during the test period. This is supported by evidence that an external non-learning model, which directly optimizes the prediction values of the most recent 180-day samples, achieved an exceptionally high score of 17.507.



This model demonstrated higher scores than other models that utilized different off-the-shelf (pre-)trained models. This is likely because general pre-trained models are designed to reflect broad market characteristics, but the public data in this competition is optimized for the limited purpose of predicting time-series local volatility and short-term excess returns. Also, reflecting a highly specific factor structure (M/E/I/P/V/S, etc.) and optimized for forecasting short-term returns.

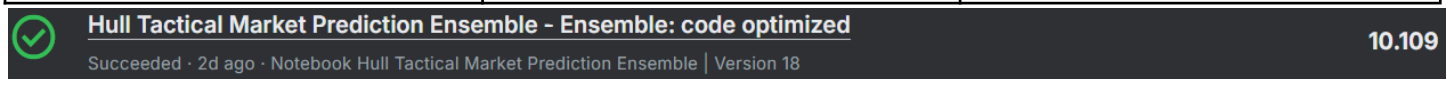
The initial model weights were [0.6, 0.2, 0.2], and the test result for that model was 9.675. For this, we first assumed that the weights for each model would be equally important, based on the finding from a previous task that weights for each model are crucial when performing ensemble tasks. We then modified these weights and conducted experiments for each case. The scores for each and the best result are as follows.

Intent to Change Weights	Weight Value	Results
Initial Weight	[0.6, 0.2, 0.2]	9.675
XGBoost Centric	[0.4, 0.5, 0.1]	9.740
Equal Weights	[0.4, 0.3, 0.3]	9.628
XGBoost-LGBM Centric	[0.5, 0.5, 0.0]	9.696
LGBM Centric	[0.7, 0.3, 0.0]	**9.810**



Additional techniques were introduced to enhance performance. First, the feature preprocessing implemented in LGBM was applied to XGBoost, and parameters such as XGBoost's target scaling and process, along with LGBM's threshold values, were adjusted. Attempts were also made to increase scores by adjusting global parameters like convert_prediction_to_allocation. The highest score achieved was 10.109, and despite the parameter and code adjustments, the score showed almost no change.

Intention	Final Changes	Results
LGBM features to XGBoost	Features of LGBM are applied	10.005
LGBM edge threshold changes	0.02 → 0.015	10.005
Confidence Multiplier changes	5.0 → 6.0	10.024
convert_prediction_to_allocation change 1	Added edge	10.005
convert_prediction_to_allocation change 2	Changed edge parameters	10.109
XGB target scaling, post-process	Scale parameters changed, etc	**10.109**



This model provides implications for the Efficient Market Hypothesis by predicting future excess returns based on past market indicators, volatility, trends, etc, and utilizing these predictions for asset allocation adjustments. Attempts to predict future return directions from past data challenge the weak form EMH(the hypothesis that past information cannot be used to predict excess returns). Moving averages, EMA, time lags, and rolling statistics suggest that the market may exhibit some trend persistence or non-stationary patterns. Combining publicly available economic/psychological indicators and detecting hidden relationships through nonlinear ML models, questioning the semi-strong EMH(the hypothesis that publicly available information is immediately reflected in prices). But it does not utilize insider or non-public information and its predictive performance is limited, so it does not disprove the strong form EMH. So this model demonstrates the possibility of a realistic ‘partially efficient market’ where the market is not completely unpredictable, but it is also limited and temporary.

However, we think that integrating external information such as text-based Soft Features (market sentiment/big events/sector outlook) generated by LLMs into the model inputs could give further performance improvements.

Extra goal: Cross-Market Extension - Bitcoin

1. Reasons for Selecting BTC

a. Similarities to the S&P 500 Index
Both Bitcoin and the S&P 500 Index serve as benchmarks for their respective markets, measure and represent the overall price trends and performance of the cryptocurrency market and the U.S. large-cap stock market. Also, the Bitcoin index adopts the same market capitalization-weighted pricing method as the S&P 500.

b. Differences from the S&P 500 Index
The most significant difference is that the S&P 500 Index represents 500 large-cap stocks listed on the U.S. stock market, whereas Bitcoin has a single underlying asset. Unlike stocks, Bitcoin is traded across multiple cryptocurrency exchanges, giving it a decentralized nature. There are also differences in their detailed composition. Unlike the S&P 500, which selects stocks based on various characteristics indicating a company's financial health, such as liquidity and financial soundness, Bitcoin's price is determined solely by aggregating real-time trading data from multiple exchanges.

c. Unique Characteristics of BTC
Due to the differences described above, the Bitcoin market reacts more sensitively to investor sentiment compared to the stock market. Consequently, indicators quantifying ‘greed’ and ‘fear’ emotions play a crucial role in investment decision-making.

2. Building the dataset through APIs and feature engineering

a. Collecting BTC data using APIs
For obtaining Bitcoin index data, we selected Upbit, the most well-known and accessible exchange in Korea. Data was extracted using the daily candle endpoint of the Upbit Quotation API, and some preprocessing, such as sorting by price, was performed.

b. Feature Engineering
- Basic Metric: We created metrics that intuitively show daily returns and log returns for profit verification, along with several representative features used in stock markets: high price, low price range, closing price, and volume-related indicators.
- Time Series-Based: We created indicators showing returns and volume changes from a certain period ago using lag features, as well as indicators representing moving statistics.
- Labels: Two labels were defined. The first is a regression label, aiming to predict the actual next-day return. The second is a classification label, aiming to predict whether the next-day return will be positive or negative.

3. Learning Process

a. Model Selection
Select either HistGradientBoostingClassifier or HistGradientBoostingRegressor; the default is Classifier. Set parameters as follows: learning_rate=0.7, max_depth=4, max_iter=300.

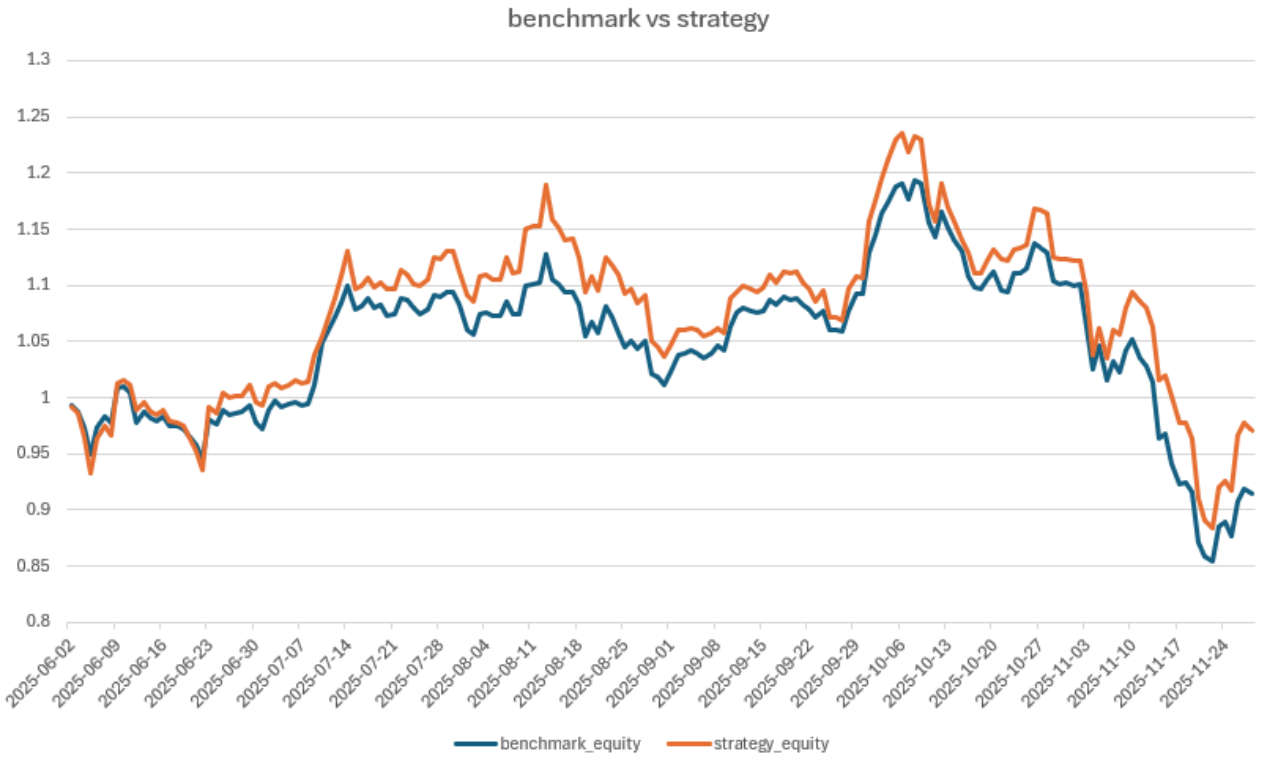
b. Pipeline Configuration
Separate the final test_days from the entire dataset to create test and backtest sets, using the preceding data as the training set. Preprocess the data by replacing any NaN values that may occur when calculating lag features or moving statistics features during the feature engineering step with the median value, making it suitable for training.

c. Differences from Existing Models
Compared to ensembles of LGBM, xgBoost, and gb models used in the previous task, Hull Tactical Predict, the HistGradientBoosting model aims to improve single-model performance through structural simplification by

implementing a single-model pipeline. This approach is efficient in terms of training speed and memory usage when handling large datasets.

d. Setting Constraints on Volatility via Allocation
After making predictions, these are converted into allocation values to set investment positions. Values closer to 0 indicate a sell position, while values closer to 2 indicate a buy position. Volatility in the allocation is constrained to prevent overly aggressive investment positions.

4. Test - Mock test execution and visualization
After training, tests were set to run automatically. We visualized the results graphically by comparing the allocation figures with the benchmark value of the executed tests—specifically, the profit achieved when no additional buys or sells occurred after the initial purchase—against the profit achieved by following the model's predictions. We observed that the strategy following the predictions recorded a slightly higher profit than the benchmark graph over time.



5. Conclusion
Visualizing the results of the preceding tests confirmed that machine learning models can provide advice leading to tangible profits in Bitcoin investments. Further performance improvements can be expected through additional model refinements and the use of diverse models.