Hull Tactical Market Prediction

Zehra Marziya Cengiz, Erik Papernuik, Elias Saker (**Team 13**)
GitHub

1. INTRODUCTION

Predicting stock market returns is one of the main problems in finance. Since markets are impacted by many factors, predicting them is a challenging task. For this reason, Kaggle hosted the Hull Tactical Market Prediction competition. In this project, we joined this competition and tried to predict daily excess returns of the S&P 500 index and provide a trading strategy that outperforms the benchmark. The following report describes our approach and results.

2. DATASET AND TASK DESCRIPTION

2.1 Dataset structure and available features

The dataset is provided as part of the Hull Tactical Market Prediction Kaggle competition and contains daily financial and macroeconomic indicators used to predict the forward excess returns of the S&P 500 index.
Each row corresponds to a single trading day.
The dataset includes (features):

- Market dynamics (M*): technical indicators and market conditions.
- Economic indicators (E*): macroeconomic data (inflation, industrial activity, growth).
- Interest rate variables (I*): short- and long-term rate levels.
- Price/valuation metrics (P*).
- Volatility indicators (V*).
- Sentiment data (S*).
- Momentum features (MOM*).
- Binary/dummy indicators (D*).

Target variables:

- forward_returns: next-day S&P 500 returns.
- risk_free_rate: daily US Fed funds rate.
- market_forward_excess_returns: adjusted excess return used as the main target.

The dataset contains more than 20 years of daily data (≈7000–8000 rows), although early time periods include many missing values due to feature availability.

2.2 Time periods covered and train/test split

The competition is split into two phases:

- Training phase: historical data up to mid-2024 (public).
- Evaluation phase: rolling future data streamed through the Kaggle API during inference (private).

In our workflow:

- We use the full public dataset for model training and validation.
- A time-series split (walk-forward validation) ensures that validation samples always occur after the training samples, preventing look-ahead bias.
- Final testing occurs on the private dataset via the evaluation server, meaning no access to future features is possible.

2.3 Exploratory Data Analysis Findings

Before building the model, we conducted an EDA. The training set contains 98 features which are grouped into seven categories.. Unfortunately, the dataset has a lot of missing values, with 12 features having more than 30% missing. The target variable (forward excess returns) has a mean close to zero and

standard deviation of 0.0105, which indicates that the data has an approximately normal distribution. We analyzed feature correlations with the target. Overall, all correlations were weak (below 0.06), confirming the difficulty of predicting market returns.

2.4 Data preprocessing steps

Several preprocessing steps were required:

- Features with excessive missingness (>30%) were removed and remaining NaNs were forward-filled, backward-filled and median imputed as a last resort.
- Feature filtering relied on both a correlation-over-time analysis and a LightGBM-based feature-importance ranking. Throughout the workflow, we maintained two parallel versions of the feature-selection process:
    1. A **quota-based version**, where each feature category was assigned a predefined number of allowed features (ex: minimum 4 volatility features)
    2. A **non-quota version**, where all features competed globally without category restrictions.

3. BASELINE METHODS

3.1 Simple Regression

In the baseline model, we used a simple Linear Regression method to establish a starting point before implementing more complex methods. The main goal of this step was to verify our submission pipeline.

Before training the model, we first preprocessed the data. We removed any feature that had more than 30% missing values. The remaining missing values were first filled with forward filling, then backward filling. Before fitting a Linear Regression model, we normalized all features with standard scaling. The predictions were clipped to the range of [0, 2] as indicated by the competition format.

We got a score of 0.817 in our Kaggle submission with this approach. This score showed us that a simple regression method is not enough to explain complicated market prediction, which aligned with our initial expectations. In this step, our main challenge was figuring out the right submission format for the Kaggle competition, which differed from our previous assignments. After that, the pipeline worked flawlessly.

4. MODEL DEVELOPMENT AND KEY FEATURES

4.1 Feature Engineering

We explored several categories of time-dependent features:

- **Momentum / Lag Features**: 1-day, 3-day, and 5-day lags for selected base features.
- **Moving averages**: Rolling mean over 5 and 10 days to capture short-term trends.
- **Volatility indicators**: Rolling standard deviations (5–10 days) and an additional volatility-regime indicator approximating high/low volatility market states.
- **Drawdown statistics**: Rolling minimums were tested but removed due to high noise contributions.
- **Calendar/Regime indicators**: Day-of-week proxies and a volatility-regime index were added to reflect structural market changes

After testing several feature-engineering approaches (momentum indicators, interaction terms, EWMs, z-scores, long rolling windows…), we found that most transformations added noise without improving walk-forward performance.
We therefore kept only a minimal and stable FE set, which consistently improved the Sharpe-variant score.
For each selected base feature, we kept:

- **Lag features**: 1-day, 3-day, and 5-day lags
- **Rolling means**: 5-day and 10-day moving averages
- **Rolling volatility**: 5-day and 10-day rolling standard deviations

This lightweight FE pipeline captures short-term momentum and local volatility regimes while avoiding overfitting.

4.2 Model Selection and Implementation

We experimented with several model families:

- **Random Forest**: Fast but underperformed due to its inability to capture subtle temporal patterns.
- **LSTM**: Tested early but discarded due to low performance.
- **LightGBM**: Demonstrated the best performance consistently and our final model choice. Handles non-linear interactions, missing values, and high-dimensional engineered features efficiently.
  A hand-crafted grid search around the best baseline configuration was used, testing variations in hyperparameters.

4.3 Time-Series Cross-Validation Strategy

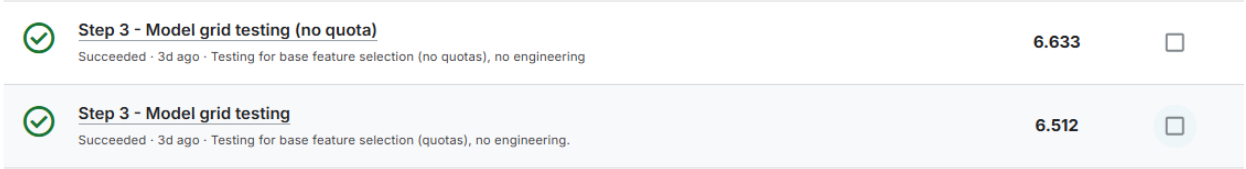
To avoid look-ahead bias in financial data, we used walk-forward validation:

- Each fold trains on a contiguous block of past data.
- Validation always occurs on future (never past) days.
- Rolling windows ensure temporal integrity of engineered features.

This method simulates realistic trading conditions and avoids data leakage caused by shuffling.

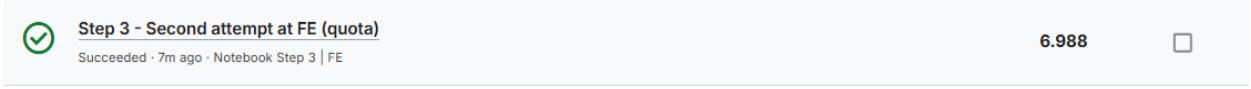
4.4 Results

With **only feature selection** (no engineering yet), we get these results using a LightGBM from a grid :



The results show that the unconstrained feature-selection strategy achieves a modest performance advantage over the quota-based method.

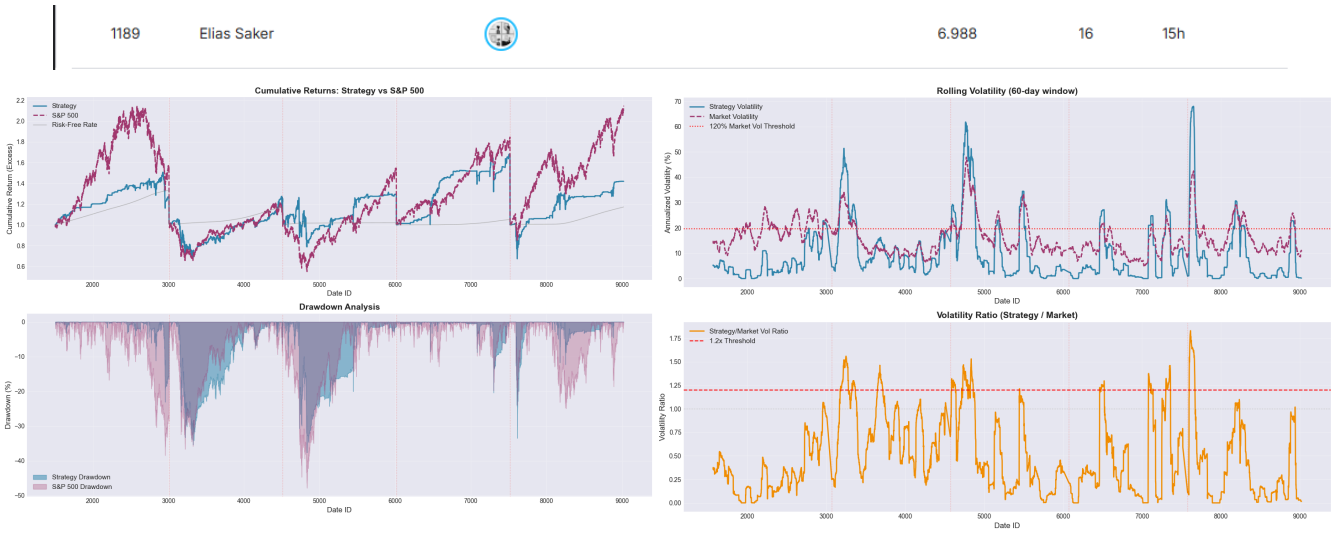
Our **best attempt** at feature engineering gives us the following results:



We can see that our feature engineering allowed us to increase our performances by a decent amount, adding more temporal context into our data and making our model reason on increasing periods of time.
It is also worth noting that this time we actually achieved better performance with the constrained feature-selection as the engineered features took advantage of a more global context with diverse features from all types compared to those from the unconstrained strategy.

5. RESULTS AND BACKTESTING ANALYSIS

Using a 5-fold walk-forward evaluation, our final LightGBM model achieves a mean adjusted Sharpe score of ~0.38. Over the same periods, cumulative returns reach ~38%, compared with ~64% for the S&P 500. The strategy stays comfortably within the 120% volatility limit (volatility ratio ≈ 0.83) and shows drawdowns similar to the benchmark (≈ –30%).
Performance is penalized mainly by limited upside capture during strong rallies, not by excessive volatility. Nevertheless, the model produces stable and risk-controlled allocations consistent with the competition's requirements.
Our final model achieve the following performances :



6. LIMITATIONS AND FUTURE DIRECTIONS

A central limitation in performance for this problem is the instability of feature importance across time. Market regimes shift, and a single global model cannot adapt equally well to all environments—leading to diluted performance when conditions change.
Another constraint is the simple linear mapping from predictions to allocations. More flexible mappings or volatility-aware position sizing could better balance risk and reward.
We have tried ensemble learning but it did not present an upgrade over our model alone, thus for future work, a more promising direction is training this ensemble of models on different temporal horizons (e.g., fast daily signals + slower weekly trends). Since financial behavior unfolds over multiple time scales, such a multi-resolution ensemble could capture more persistent patterns.
Regime-specific models and richer feature representations (e.g., neural embeddings) also offer potential improvements, provided temporal leakage is avoided.

7. CONCLUSION

This project shows both the potential and difficulty of using machine learning to predict market movements. Indeed, despite careful designing, the model performs below the S&P 500 benchmark, although relatively close. It controls risk well but struggles to consistently outperform in an efficient and rapidly adapting market.
This outcome echoes the principles of the Efficient Market Hypothesis (EMH)—the idea that asset prices already reflect most available information, making systematic outperformance inherently difficult.

Appendix: Cross-Market Extension

1. BTC - Market Selection

As our market, we selected Bitcoin (BTC/USDT) for the following reasons.

1. Unlike the S&P 500, Bitcoin is available for trading 24/7. This removes the market closure problem.
2. Bitcoin has higher volatility than the traditional market. This makes it less affected by macro financial events.

These features create a better environment for testing ML-based forecasting models. We collected our data from the Binance public API which covers January 2015 to December 2025.

2. Model Architecture

We used a Long Short-Term Memory (LSTM) neural network in our prediction model which is implemented in PyTorch. We decided to use LSTM because it is good at finding temporal dependencies in sequential financial data. The LSTM network takes 20-day sequences of 14 features. Then, it processes these features in a LSTM layer. After these steps, it gives a next-day log-return prediction as its output.

We preferred LSTM instead of tree-based models like LightGBM because crypto returns show time-dependent patterns. Recurrent networks handle time-dependent patterns better. Also, 20-days was chosen as sequence length because we wanted to find an equilibrium between having enough context and limiting overfitting.

3. Hyperparameters

Category	Parameters
Data	Sequence: 20 days | Train/Val/Test: 70/15/15% (2015–2022 / 2022–2024 / 2024–2025)
Model	LSTM: 64 hidden, 1 layer | Adam (lr=0.001) | MSE loss | Early stop patience: 5
Strategy	Threshold: 0.5% | Weights: [0, 2] | Vol constraint: ≤1.5× benchmark

We had a 0.5% threshold on validation data to reduce the noise without losing any meaningful signals.

4. Feature Engineering

Our features consists of

1. 1 - to 10-day lagged daily returns
2. 7-day rolling volatility

3. 7-day rolling mean return
4. 7-day moving average of price
5. Log-transformed volume

We normalized all features with a Z-score normalization. To avoid data leakage, we only used training set statistics.

5. Strategy Logic

Prediction-to-Weight Mapping: We used continuous weight scaling. In this method, if we had a negative or zero predicted return, the weight stays at 0, meaning 100% cash. Otherwise, the weight increases linearly according to the following formula: Weight = min(prediction / 0.005, 2).

Volatility Constraint: To control the risk, we put a $\leq 1.5\times$ benchmark volatility limit on strategy volatility. If this limit is exceeded, we scaled down the weight proportionally. However, in our backtest, we realized that volatility never exceeded this limit. It was only 0.33× the benchmark.

6. Results Summary

Metrics (Between April 2024 – December 2025)

Metric	LSTM Strategy	Buy & Hold BTC	Difference
Total Return	33.34%	23.26%	+10.08 pp
Annualized Volatility	15.06%	45.74%	-30.68 pp
Sharpe Ratio	1.279	0.517	+0.762
Maximum Drawdown	-3.23%	-33.12%	+29.89 pp

Key Observations

1. The system preferred to keep funds in cash about 97% of the time. This strategy successfully avoided a 33% drop in value. This result is important because it shows our model overcomes a risk seen in a buy and hold strategy.
2. The model captured upward moves using only 19 trading days from mid-August 2024 into early March 2025.
3. The strategy has a 2.5× better Sharpe ratio and approximately 67% lower volatility.
4. We faced several limitations. One limitation we faced was that the participation rate was low. Another one was that we only had a small number of active trading days to analyze so our sample size was limited. These limitations make it hard to draw conclusions on the edge cases.