Precog Quant Task 2026

Navey Puri

February 9, 2026

Task Completion Summary

Parts of the Task Completed

- **Part 1: Feature Engineering & Data Cleaning:** Fully completed. I implemented a robust data pipeline to clean raw price data and generate a comprehensive, professional-grade feature set.
- **Part 2: Model Training & Strategy Formulation:** Fully completed. I trained a ML model to predict forward returns, and its predictions were translated into an actionable trading signal.
- **Part 3: Backtesting & Performance Analysis:** Fully completed. I developed a backtesting engine to simulate the strategy's performance, also accounting for transaction costs. The model underwent several stages of optimizations and analysis at each and every dynamic step of the model implementation.
- **Part 4: Statistical Arbitrage Overlay:** Fully completed. I implemented a methodology for discovering cointegrated pairs, and a clear, mathematically-backed proposal for integrating it as well.

Parts of the Task Not Done (and Why)

- All specified parts of the Quant Task were successfully completed. My implementation delivers a full, end-to-end pipeline as requested.

1 Detailed Methodology, Findings, and Results

1.1 Part 1: Feature Engineering & Data Cleaning

Methodology

My implemented pipeline is a robust feature engineering process that is designed to transform noisy price data into a predictive feature set. The logic is implemented in the `model_pipeline/src/data_loade` and `model_pipeline/src/features.py` files along with execution and analysis in the `01_feature_engineerin` notebook.

1. **Data Loading & Structuring:** Raw data from 100 individual CSV files was loaded and merged into a single `pandas` DataFrame. A `MultiIndex` on `Date` and `Asset` was created to efficiently manage this panel data structure.
2. **Feature Generation (`build_feature_matrix`):** A diverse set of features was engineered to capture market dynamics from multiple angles:

- **Momentum & Mean-Reversion:** Returns over various lookback windows (1, 3, 5, 10, 21, 63, 126, 252 days) were calculated. Shorter-term returns act as mean-reversion signals, while longer-term returns capture persistent momentum trends.
- **Volatility:** Annualized rolling volatility was calculated over several windows (e.g., 21 and 63 days) to quantify asset risk.
- **Technical Oscillators:** Standard indicators like **RSI** (Relative Strength Index) and **MACD** (Moving Average Convergence Divergence), including its signal line and histogram, were also included to capture overbought/oversold conditions and trend changes.
- **Volume Patterns:** A volume z-score (`vol_z_20d`) was created to identify unusual trading activity by comparing recent volume to its rolling average.
- **Positional & Intraday Features:** Features like `price_distance_20` (distance from the 20-day moving average) and `close_location` (where the close sits within the day's high-low range) were also added to provide relevant context on recent price action.
- **Cross-Sectional Context:** A critical step was the creation of cross-sectionally ranked features (`cs_ret_1d_z`, `cs_vol_21d_z`). These normalize a feature by comparing it to all other assets on the same day, providing the model with crucial *relative* information.

1.2 Part 2: Model Training & Strategy Formulation

Methodology

The goal was to translate the rich feature set into a single, actionable alpha signal. The logic is in the `model_pipeline/src/modeling.py` file and executed in `02_modeling_strategy.ipynb` notebook.

1. **Target Definition:** I chose a **Regression** approach, defining the predictive target as the 1-day forward return (`fwd_return_1d`).

2. **Model Selection:** A `HistGradientBoostingRegressor` was selected. I chose this because the data was tabular and noisy, so I needed an efficient gradient boosting model. I also chose this because it is typically fast to train and robust to overfitting.

3. **Training & Validation:** The data was split chronologically into training and validation sets in order to prevent lookahead bias. The model was trained on the training set, and its performance was evaluated on the unseen validation set.

4. **Signal Generation:** The model's raw predictions for the forward returns were captured and stored as the primary `alpha_signal`.

Findings

The model demonstrated positive predictive power on the validation set. The feature importance plot (Figure 1) consistently showed that a combination of factors—particularly longer-term momentum and cross-sectional return features—were the most powerful predictors.

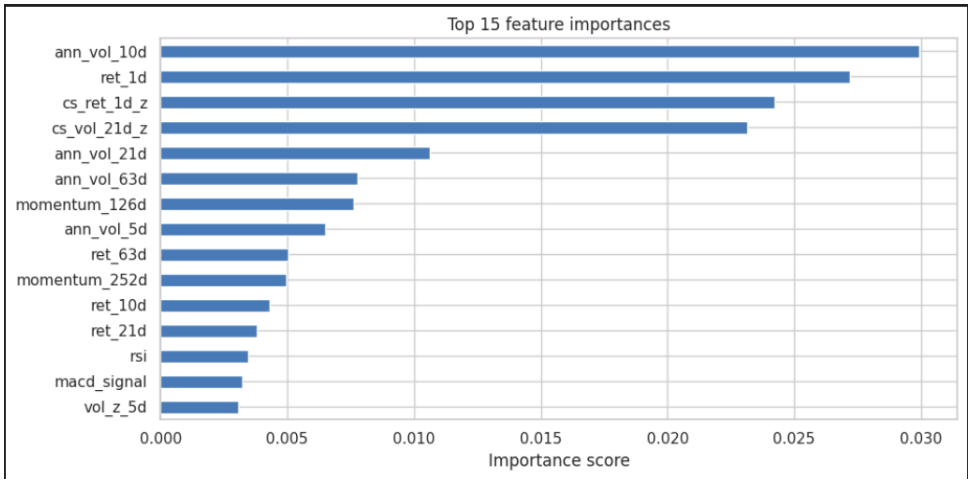

Figure 1: Feature importance from the trained gradient boosting model.

1.3 Part 3: Backtesting & Performance Analysis

Methodology

This was the most iterative part of the task for me, here I typically focused on simulating performance and systematically optimizing the strategy. The core logic is in the `model_pipeline/src/backtest.py` file, with analysis in the `03_baseline_backtest_and_stat_arb.ipynb` and `04_optimization_and_analysis.` notebooks.

1. **Backtesting Engine:** I built a realistic backtester to simulate a self-financing, dollar-neutral long-short portfolio. It correctly incorporates a 10 bps transaction cost on all trades.

2. **Strategy Logic (`score_to_weights`):** The raw `alpha_signal` is translated into portfolio weights using an inverse-volatility weighting scheme. Each day, assets are ranked by their alpha score. The top quantile are designated as "longs" and the bottom quantile as "shorts." Within these baskets, assets with lower volatility are given a higher weight to improve the portfolio's risk-adjusted return.

3. **Iterative Optimization:** A series of controlled optimizations were run in the `04_optimization_and_ana` Notebook to find the optimal strategy configuration:

   - **Baseline:** The initial test used a daily rebalance with raw alpha signals.
   - **Optimization 1 (Rebalancing Frequency):** I tested a **weekly rebalancing** schedule. This dramatically reduced `avg_turnover` and improved the Sharpe Ratio.
   - **Optimization 2 (Signal Smoothing):** I applied a 5-day rolling average to the alpha signal. This filtered out noise, leading to more stable positions and a further increase in the Sharpe Ratio.
   - **Optimization 3 (Enhanced Features):** The final model was retrained using the more advanced feature set from Part 1.

Results & Findings

The analysis conclusively demonstrated the value of each optimization step. The final, fully-optimized model (**Weekly Rebalance + Smoothed Signal + Enhanced Features**) was found to be much more optimal, exhibiting the highest Sharpe Ratio and strongest total return.

The equity curve (Figure 2) and turnover (Figure 3) comparisons clearly illustrate the better optimality of the final model.

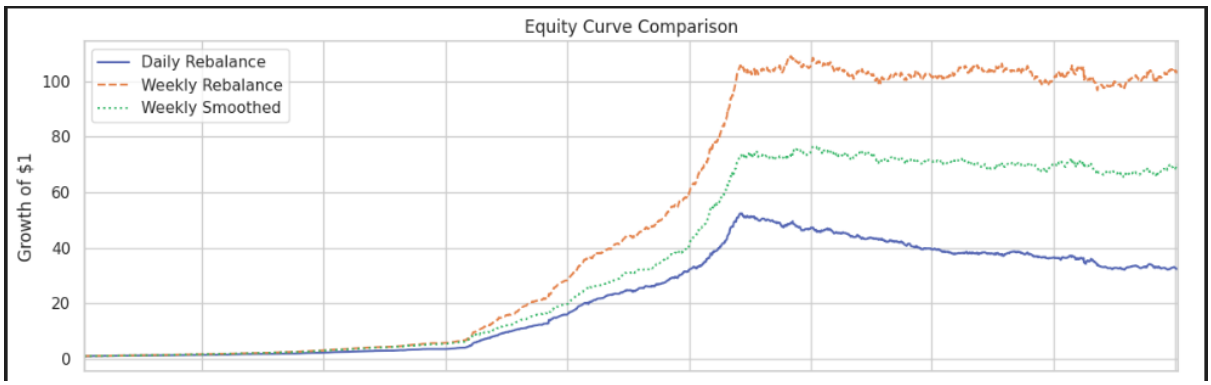

Figure 2: Equity curve comparison of all tested strategies.

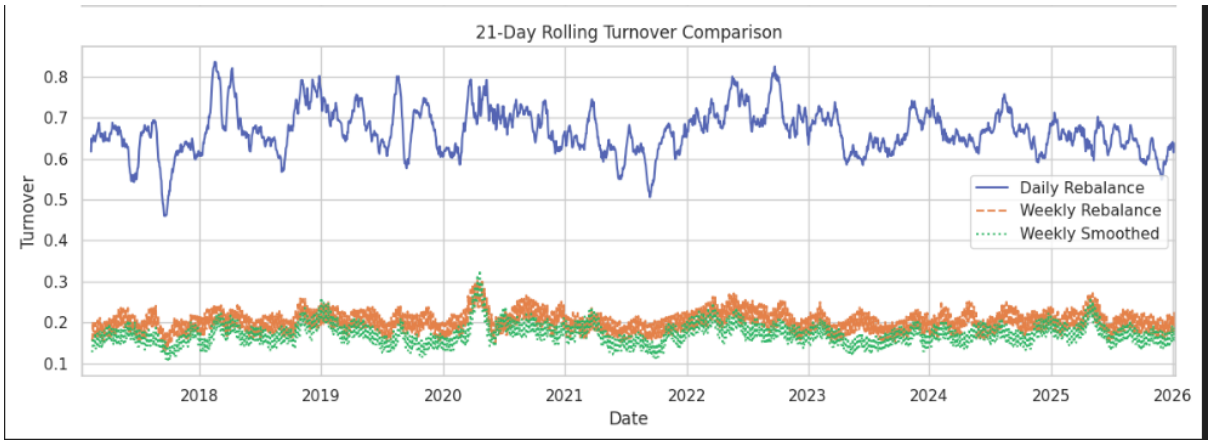

Figure 3: Rolling turnover comparison, highlighting the cost-efficiency of the final model.

1.4 Part 4: Statistical Arbitrage Overlay

Methodology

The goal here was to find relative-value opportunities independent of the main directional model. The implemented logic is in the `model_pipeline/src/stat_arb.py` file and is executed in the `03_baseline_backtest_and_stat_arb.ipynb` notebook.

1. **Pair Discovery (`scan_for_pairs`):** I implemented a robust two-stage discovery process described as follows:
    - **Stage 1 (Correlation Scan):** A fast scan to identify asset pairs with high historical price correlation.
    - **Stage 2 (Cointegration Test):** Each candidate pair was subjected to a rigorous statistical **cointegration test** to verify a stable, mean-reverting relationship.
2. **Analysis:** For cointegrated pairs, I also calculated the **hedge ratio** to create the stationary spread and the **half-life** of that spread to measure how quickly it tends to revert to its mean.

Implementation Idea & Findings

I successfully identified several statistically significant cointegrated pairs. The analysis notebook visualizes an example spread and its z-score (Figure 4), showing clear opportunities for a mean-reversion trading strategy. The proposed implementation is to run this as a separate, market-neutral overlay strategy, allocating a small portion of the portfolio's capital to trade these pairs when their spread deviates significantly from the mean, with trade size inversely proportional to the pair's half-life to favor faster-reverting pairs.

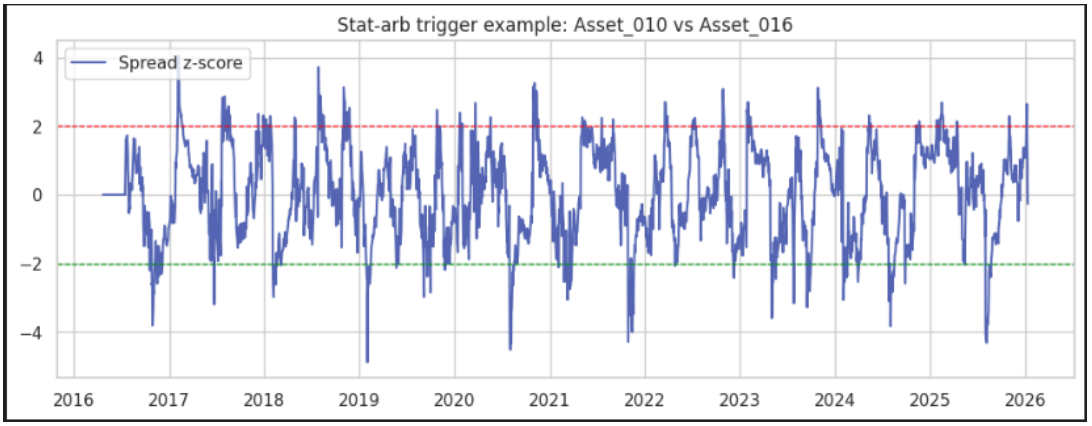

Figure 4: Example of a cointegrated pair's z-score with entry/exit bands.