Week 8 — Develop Third modeling approach

1.Model training

We built a histogram-based gradient boosting regressor as the Week 8 baseline and wrapped it in a small, reproducible training cell. The choice matches the data and the weekly goal. Our matrix is fully numeric and mixes macro levels, lags, and engineered changes; interactions and thresholds matter, while strict scaling is not. Tree ensembles handle that pattern naturally. The histogram variant bins features once and grows trees on the binned representation, which reduces split-search cost and keeps training fast and memory-efficient while preserving the ability to learn non-linear effects. Under squared loss this learner is fairly resilient to moderate outliers, needs little preprocessing, and behaves well when features have different ranges. These properties let us keep the pipeline short and stable so others can pick up the artifact and continue with tuning and evaluation.

The code starts by resolving input names defensively. In shared notebooks the training matrices are not always called the same thing, so we search a small set of common variable names and grab the first that exists. Next, we align features and target by index when both carry labeled rows; otherwise we trim to the shorter length. This guards against off-by-k mistakes after filtering or joins. We normalize the output directory as well: if a global MODELS path exists we write there, else we create a local “Models” folder. With paths set, we define a plain baseline model with a learning rate of 0.1, no explicit depth cap, and 300 boosting iterations, and we fix the random state for reproducibility. We fit on NumPy arrays to avoid surprises from exotic index types, then persist three artifacts: a serialized model, a JSON file that records the exact feature column order, and a compact “training card” with model family, key hyperparameters , row counts, and the seed. The final print lines echo the file locations so downstream steps can load the right objects.

This approach fits the prediction task because it balances expressiveness and discipline. Gradient boosting can learn curved effects and two-way interactions typical in macro-financial data, yet it remains controllable through a few levers. In scikit-learn’s histogram engine, each feature is bucketed into a modest number of bins, so evaluating candidate splits uses aggregated statistics over bins rather than scanning all unique values; this substantially lowers constants in runtime and memory. Regularization together with shrinkage via the learning rate provides the main bias–variance control, and early-stopping hooks can be enabled during tuning if desired. Because the training recipe avoids scaling and encoding, it also avoids drift from stateful transformers; the model consumes the raw numeric matrix directly.

The complexity of this modeling approach appears in two layers: algorithmic cost inside the ensemble and operational choices that make the training cell repeatable. On algorithmic cost, four factors dominate. The number of boosting iterations governs how many stage-wise trees are added; runtime grows roughly linearly with this count. The learning rate scales each tree’s contribution; smaller values typically require more iterations but generalize better. Tree capacity (depth or leaves) sets per-iteration cost and flexibility; deeper trees capture finer local structure at the expense of computation and overfit risk. Finally, effective dimensionality matters: highly correlated features increase the number of near-duplicate split candidates, so regularization and conservative tree sizes are helpful. In the baseline we fix a moderate learning rate and a reasonable iteration budget, deferring stronger capacity changes to tuning so training is quick while leaving headroom for improvement.

Operationally, robustness and traceability are prioritized. Flexible name resolution reduces friction when the notebook is reused across sections; index-aware alignment prevents silent mislabels when features and targets were filtered differently; persisting the feature list is essential because tree models are column-order sensitive; and the training card serves as a minimal experiment ledger. The artifacts are small and portable via joblib, which makes hand-offs cheap and lowers environment risk as long as the scikit-learn version is consistent.

In short, a histogram-based gradient boosting regressor gives a strong, pragmatic baseline for our numeric, macro-focused matrix. The code avoids brittle preprocessing, guards against common notebook pitfalls, and produces artifacts that are easy for the team to tune and evaluate without refactoring the workflow.

2. Hyperparameter tuning

This week, we performed hyperparameter tuning on the previously trained HistGradientBoostingRegressor model. This step aimed to optimize model parameters to enhance prediction accuracy and generalization capabilities on both the training and validation sets.

We focused on three key hyperparameters significantly impacting model performance. The first parameter, learning_rate, controls the step size during each iteration when updating predictions. An excessively high learning rate may cause model oscillations and prevent convergence, while an overly low value significantly slows training speed. We tested values of 0.05, 0.1, and 0.2 to balance convergence speed with model stability. The second parameter, max_depth, limits the growth depth of decision trees, preventing overfitting by controlling model complexity. We experimented with three depth settings: 3 layers, 5 layers, and None, where None allows trees to grow automatically until no further improvement is possible. The third parameter, max_iter, determines the number of boosting iterations or the number of trees in the ensemble. Too few iterations may result in underfitting, while too many may cause overfitting. We tested 200, 300, and 500 iterations to identify the optimal balance.

We employed GridSearchCV for hyperparameter tuning, combined with three-fold cross-validation to ensure robustness in evaluation. Cross-validation measures model performance across different data subsets, reducing bias risks from single-split data. The optimization metric used was negative root mean squared error (neg_root_mean_squared_error), which quantifies the average deviation between predicted and actual values. The grid search process automatically tested all 27 parameter combinations and selected the configuration with the lowest root mean square error as the optimal solution.

The final optimal parameter set is: learning rate 0.05, maximum depth 5, maximum iterations 300. After retraining the model on the training data using this parameter set, the training root mean square error (RMSE) was approximately 0.0604. This indicates a small average prediction error and acceptable numerical precision. A lower RMSE means the predicted values are closer to the actual values. The R^2 value is approximately 0.3115, indicating that the model explains about 31% of the variability in the target variable, suggesting it captures some structural information within the data. However, approximately 69% of the variability remains unexplained. These results indicate the model possesses predictive capability, capable of identifying local patterns within the training data. The relatively low R^2 value may suggest: insufficient feature information or weak correlations, constrained model complexity (e.g., maximum layer limit), excessive data noise, or significant difficulty in predicting the target variable. Nevertheless, its stability is good and no overfitting occurred, making it suitable as the optimal model. Subsequently, we saved this model and evaluated it on an independent validation dataset.

To ensure reproducibility and facilitate subsequent analysis, we saved the tuned model as a .joblib file named “week8_best_hgbr.joblib”. This allows the model to be directly loaded and used in subsequent steps without retraining.

In summary, the hyperparameter tuning process fully demonstrates the significant impact of parameter selection on model performance. Compared to the baseline version, this tuning simultaneously enhances both the model's accuracy and interpretability. Through this process, we mastered how to balance learning rate, model complexity, and iteration depth, and learned to systematically identify the optimal model using cross-validation and root mean square error. This step lays a solid foundation for further evaluation and comparison in subsequent phases.

3. Model evaluation

Following the model tuning phase, this week focused on conducting a comprehensive evaluation of the tuned histogram-based gradient boosting regressor. The objective was to measure its predictive accuracy, interpret its generalization ability, and visualize the relationship between predicted and actual values. Evaluation was carried out using the same engineered macro-financial feature matrix constructed earlier, ensuring consistency across feature definitions and scaling. The model loaded for testing was the optimized version saved as *week8_best_hgbr.joblib*, and the evaluation dataset consisted of 741 monthly observations with 74 numeric features across six real estate companies.

The evaluation metrics were chosen to capture different aspects of model performance: mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), mean absolute percentage error (MAPE), the coefficient of determination (R^2), and the adjusted R^2. These metrics jointly assess the model's bias, variance, and overall explanatory power. The results obtained were as follows: MSE = 0.0076, RMSE = 0.0871, MAE = 0.0641, MAPE = 2.0531, $R^2 = -0.1476$, and adjusted $R^2 = -0.1828$.

At first glance, the magnitude of the RMSE and MAE suggests that the model's numerical predictions stay within a relatively small absolute error range (roughly 8–9% on average). However, the negative R^2 values indicate that the model explains less variance than a simple mean predictor, implying a very weak linear fit between predicted and true returns. This outcome suggests that the model captures little systematic information in the evaluation sample, and its predictions tend to regress toward the mean, yielding an almost flat output surface.

Visual inspection further supports this diagnosis. In the *Actual vs Predicted TSR* scatter plot, the predicted points concentrate tightly around the zero line with minimal spread. The fitted red dashed diagonal—representing the perfect 1:1 relationship—passes through the dense center of the cloud, yet the surrounding dispersion shows no strong directional trend. This pattern implies that the model largely predicts small positive or near-zero returns for all companies, regardless of the actual outcome. The *Residual Distribution* plot presents a near-Gaussian shape centered at zero, confirming that residuals are symmetric but narrow. Together, these visuals highlight a model that is stable but underpowered: it produces consistent yet uninformative predictions, failing to react strongly to variations in the explanatory features.

Several factors may explain this weak performance. First, the evaluation dataset differed subtly from the training and validation splits in both time span and company coverage. Fourteen feature columns recorded during training—mostly company dummy variables and city-level identifiers—were missing in the test file and had to be filled with zeros before prediction. Because tree-based models rely on these encoded categories to partition data, their absence effectively removed key segmentation cues. Second, the overall variation in Total Shareholder Return (TSR) across months was low, with most values clustered near zero, leaving little dynamic range for the model to learn meaningful gradients. Third, despite prior tuning, the chosen parameter combination (learning rate = 0.05, max depth = 5, max iterations = 300) may still be too conservative for a highly non-linear target such as next-month TSR, leading to underfitting. Finally, macro-financial environments can change structurally between training and testing periods; regime shifts in interest rates or post-pandemic volatility could render past patterns obsolete.

Nevertheless, the evaluation remains valuable because it clarifies the current model's limitations and establishes a baseline for future refinement. The low explanatory power, combined with stable residual behavior, suggests that model errors are random rather than systematically biased. This stability indicates that the preprocessing pipeline and feature alignment work as intended, even though the signal itself is weak. The diagnostic plots also confirm the absence of severe outliers or asymmetric tails, meaning the model predictions are statistically well-behaved.

To improve performance in subsequent iterations, several adjustments are proposed. Re-introducing the missing categorical dummy variables with consistent encoding across training and testing stages is a priority, as categorical splits often drive much of the predictive gain in boosting models. Expanding the feature set to include lagged macro indicators (for instance, prior-month inflation changes or yield curve differentials) could also enhance temporal sensitivity. From a modeling perspective, increasing tree depth or number of iterations, combined with stronger learning-rate regularization, may help the ensemble capture subtler interactions without overfitting. Alternatively, gradient-boosted decision trees could be complemented with linear terms or simple autoregressive structures to capture both cross-sectional and temporal dependencies. Finally, evaluating the model under rolling-window or time-series cross-validation would provide a more realistic assessment of forward-looking stability.

In summary, while the tuned histogram-based gradient boosting model demonstrates robustness and computational efficiency, its explanatory strength remains limited when applied to the held-out test period. The quantitative metrics show small absolute errors but minimal correlation with true TSR values, and the diagnostic plots reveal a consistent yet uninformative prediction pattern. This week's evaluation highlights the need for stricter feature alignment and possibly richer temporal dynamics in the input space. The exercise successfully validated the reproducibility of the modeling pipeline and clarified the next directions for improvement, marking an essential checkpoint before advancing to integrated ensemble or hybrid forecasting approaches in the following phase.