Week 13 — Bring it all together

1. Model Development Life Cycle for REIT Total Shareholder Return Prediction

Our project followed a complete model development life cycle (MDLC) to build a predictive model for Real Estate Investment Trust (REIT) total shareholder returns. The workflow spanned from raw data ingestion through deployment preparation, encompassing seven major phases that align with industry best practices for machine learning systems.

1.1 Data Ingestion and Integration

The first step involved collecting and merging multiple data sources into a unified monthly panel dataset. We gathered stock price and dividend data for six publicly traded REITs spanning major U.S. metropolitan areas: Boston Properties (BXP), SL Green Realty (SLG), Elme Communities (ELME), Equity Residential (EQR), Rexford Industrial (REXR), and Terreno Realty (TRNO). These companies operate in Boston, New York, Washington D.C., Chicago, Los Angeles, and Miami respectively.

In addition to firm-level data, we integrated macroeconomic indicators to capture regional and national economic conditions. This included metro-specific Consumer Price Index (CPI) data from the Bureau of Labor Statistics for all six markets, Metropolitan Statistical Area (MSA) unemployment rates, and the 10-year U.S. Treasury yield as a proxy for risk-free rates and broader market sentiment.

The data ingestion process involved several critical transformations. Daily stock prices were aggregated to month-end closing values to align with the monthly frequency of macroeconomic indicators. Dividend payments were summed by ex-dividend month and converted into trailing twelve-month (TTM) totals to capture the full income component of shareholder returns. All date fields were standardized to month-end format to ensure proper temporal alignment across data sources. The final merged dataset contained one observation per company per month, with features spanning stock prices, dividend metrics, inflation measures, unemployment levels, and Treasury yield movements.

1.2 Exploratory Data Analysis and Temporal Splitting

The second phase focused on understanding data characteristics and establishing proper train-validation-test splits that respect the temporal nature of financial time series. Our prediction target was monthly Total Shareholder Return (TSR), defined as:

TSR = (Price_next - Price_current) / Price_current + Dividend_next / Price_current

This formulation captures both capital appreciation and dividend income, representing the complete return an investor would realize by holding the stock for one month. The target variable was constructed by computing next month's price and dividend for each company-month observation, ensuring predictions would not suffer from look-ahead bias.

We implemented a chronological split strategy, allocating 70% of observations to the training set, 15% to validation, and 15% to the final test set. This time-based partitioning is crucial in financial forecasting to prevent information leakage from future periods into model training. Exploratory data analysis revealed that TSR exhibited moderate volatility with occasional extreme values during market stress periods. CPI showed consistent upward trends across most metros with notable acceleration in recent years. Correlation analysis highlighted expected relationships such as negative correlation between TSR and unemployment. Missing value analysis identified sparse gaps in macroeconomic series, primarily due to reporting lags, which informed our subsequent data cleaning strategy.

1.3 Data Cleaning and Preprocessing

The third phase addressed data quality issues and transformed raw features into model-ready inputs. Missing value imputation followed a two-stage approach. For macroeconomic variables, we first applied forward-fill within each metro area to propagate the most recent valid observation. Any remaining missing values were filled with the median across all observations for that feature, providing a conservative central estimate.

Outlier treatment employed multiple strategies depending on feature characteristics. For bounded variables like unemployment rates, we applied hard clipping to ensure values stayed within plausible ranges (0-25% for unemployment, 0-15% for Treasury yields). For momentum and change variables such as month-over-month CPI growth and Treasury yield changes, we used Interquartile Range (IQR) based winsorization, which clips values falling outside 1.5 times the IQR beyond the first and third quartiles.

Feature transformation included logarithmic scaling for skewed variables. Stock prices and trailing dividend amounts were transformed using log(1+x) to compress their dynamic range and reduce the impact of extreme values. Standardization was applied to all numeric features using scikit-learn's StandardScaler, critically fitted only on the training set to prevent test set information leakage. Categorical variables (company ticker and metro area) were one-hot encoded, resulting in six company indicators and six metro indicators. The final preprocessing step removed redundant raw columns that had been transformed to avoid multicollinearity.

1.4 Feature Engineering

Feature engineering translated the cleaned dataset into informative predictors that capture temporal patterns and economic relationships. This phase generated 73 total features through several transformation families.

Lag features captured historical information by creating time-shifted versions of key variables. For the target variable TSR and all macroeconomic indicators (CPI year-over-year, CPI month-over-month, unemployment level, unemployment change, Treasury yield, and Treasury yield change), we generated 1-month, 3-month, and 6-month lags. These lags allow the model to learn autoregressive patterns and delayed effects of economic conditions on stock returns.

Rolling window statistics quantified recent trends and volatility. For TSR, CPI month-over-month change, unemployment, and Treasury yield, we computed 3-month and 6-month rolling means and standard deviations. Rolling means capture momentum and trending behavior, while rolling standard deviations measure volatility and uncertainty. These features help the model distinguish between stable and turbulent market conditions.

Interaction features explicitly encoded domain knowledge about economic relationships. We created three key interactions: (1) inflation-unemployment interaction (CPI growth × unemployment rate) to capture stagflation or Phillips curve dynamics, (2) interest rate-dividend interaction (Treasury yield × log dividend) to represent the trade-off between fixed income yields and equity income, and (3) dividend yield ratio (log dividend / log price) to measure the income component of returns relative to valuation.

Time-based features captured seasonal patterns through month-of-year indicators. We one-hot encoded the calendar month (1-12) to allow the model to learn seasonal effects in real estate returns. After generating all features, we removed rows containing NaN values introduced by lag and rolling window operations. The feature engineering process was applied identically to training, validation, and test sets, with lag and rolling calculations computed separately within each company's time series to prevent cross-contamination between firms.

1.5 Model Training, Hyperparameter Tuning, and Selection

The fifth phase implemented a systematic model comparison across three complexity levels. We trained nine models spanning three model families: simple linear models with regularization (Lasso), medium-complexity non-linear ensembles (Random Forest), and complex boosted tree ensembles (Gradient Boosting).

For the Lasso family, we tested three L1 regularization strengths (alpha = 0.0005, 0.001, 0.005). For Random Forest, we varied the number of trees (200, 300, 500) and maximum tree depth (5, 8, 10). For Gradient Boosting, we tuned the number of estimators (200, 300, 300), learning rate (0.05, 0.05, 0.03), and tree depth (3, 3, 4).

All models were trained on the 70% training set and evaluated on the 15% validation set using four metrics: Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and R-squared (R^2). Model comparison results showed that Gradient Boosting consistently outperformed other families. The best-performing configuration—Gradient Boosting with 300 estimators, learning rate 0.03, and maximum depth 4—achieved a validation RMSE of 0.018 and R^2 of 0.969.

The selected model was then evaluated on the held-out test set to estimate true out-of-sample performance. Test set results confirmed robustness: RMSE of 0.018, MAE of 0.014, and R^2 of 0.969. These metrics indicate that the model explains approximately 97% of the variance in monthly TSR and maintains an average prediction error of just 1.4 percentage points—a strong result given the inherent volatility of equity returns.

1.6 Data-Centric Model Refinement

Following initial model selection, we applied data-centric AI principles to improve performance through data quality enhancements rather than algorithmic changes. Error analysis revealed that the top 10% of training samples by absolute error exhibited significantly larger residuals. Many of these high-error cases corresponded to extreme market events or data quality issues.

We implemented two data refinement strategies. First, we removed the top 5% most extreme outlier samples from the training set—those with prediction errors exceeding the 95th percentile. Second, we applied RobustScaler instead of StandardScaler to all numeric features. RobustScaler uses median and interquartile range rather than mean and standard deviation, making it less sensitive to outliers.

The refined dataset was used to retrain the exact same Gradient Boosting configuration selected in Phase 5, isolating the impact of data improvements from model architecture changes. Performance comparison showed minimal improvement on validation (RMSE changed from 0.0182 to 0.0180), indicating that the original data quality was already sufficient. Despite modest metric improvements, this exercise yielded valuable insights: prediction failures were not random but concentrated in specific market regimes (high volatility periods) and certain companies (those with thinner trading volumes).

1.7 Bias Analysis and Model Explainability

The seventh phase examined model fairness, interpretability, and performance disparities across companies. Feature importance analysis identified the top three most important features: (1) TSR lagged 1 month (autoregressive momentum), (2) 6-month rolling mean of TSR (medium-term trend), and (3) CPI year-over-year change (inflation environment). These results align with domain intuition—recent return history and inflation expectations are well-known drivers of equity returns.

Company-level bias analysis evaluated whether the model performed equitably across all six REITs. We computed MAE, RMSE, and R^2 separately for each company. Results showed modest performance variation: MAE ranged from 0.012 to 0.017 across companies, with a spread of 0.005. Companies with higher MAE tended to be those with more volatile return patterns or those operating in metros with more economic instability. The observed MAE range represents approximately 30% of the overall MAE, indicating moderate but not extreme bias.

Individual prediction analysis examined five diverse cases spanning the prediction quality spectrum. The best predictions typically occurred during stable market periods with consistent momentum. The worst predictions often coincided with abrupt market reversals—months where a company's return sharply diverged from its recent trend due to idiosyncratic news events. These cases highlight the model's limitation in capturing unexpected discrete events that lack clear antecedent signals in the feature set.

1.8 Model Deployment Preparation

The final phase prepared the trained model for production deployment. Model serialization used Python's pickle protocol to save the trained Gradient Boosting model object to disk. The serialized file contains all learned parameters—tree structures, split thresholds, leaf values—enabling exact reconstruction of predictions without retraining. Alongside the model, we saved a feature schema JSON file documenting the 73 required input features.

We developed a standardized inference pipeline consisting of three functions: load_model_for_inference(), load_data_for_inference(), and make_predictions(). Deployment testing verified the complete inference pipeline end-to-end, confirming that serialized predictions exactly matched the original model's outputs. Performance benchmarks showed that inference latency was negligible (under 10 milliseconds for a batch of 100 predictions).

Documentation deliverables included a dependency specification listing all required Python packages with minimum version numbers and a monitoring plan for production deployment. The plan specifies data drift detection (tracking whether input feature distributions shift significantly from training data), concept drift detection (monitoring whether the relationship between features and target changes over time), and retraining triggers (performance degradation exceeding 15% or quarterly scheduled updates). At this point, the full life cycle is complete, and the project can be handed to a model governance officer or engineering team as a coherent, end-to-end artifact.

2.In the second phase of Week 13, we systematically organized and restructured the entire project. Our goal was to establish a final model development workflow that is clearly structured, logically consistent, and capable of running independently from start to finish, all while adhering to the MDLC framework. Given that the code accumulated over the first 12 weeks evolved incrementally during the learning process, issues such as redundancy, fragmentation, high coupling, and messy dependencies inevitably arose. Consequently, this week's primary task was not introducing new content, but rather determining which components should be retained, which should be discarded, and how to distill a truly standardized pipeline from the multitude of code fragments.

During this reorganization, we first eliminated numerous redundant model training scripts generated in earlier stages. For instance, Weeks 8 and 9 repeatedly trained various models—including ElasticNet, Random Forest, and HGBR—but a production-grade ML pipeline requires only one validated optimal model. Consequently, we retained only Week 9's Lasso model as the final model, removing all training logic related to other models. This reduced code complexity and prevented dependency conflicts among redundant models. Similarly, extensive visualization code, debugging outputs, temporary file writes, and other elements generated throughout the exploration phase were removed during this consolidation. These steps are meaningful only during learning and exploration phases, are unnecessary for the final pipeline, and disrupt the overall structural coherence.

Simultaneously, we organized all data processing steps to ensure a natural progression from Week 5 through Week 6, Week 9, Week 10, Week 11, and finally to Week 12. For instance: - Week5's feature engineering and target variable construction form the foundation for all subsequent model training and must be retained; - Week6's data partitioning and schema output determine the training set structure, being central to model reproducibility; Week 9's optimal model training, as the final outcome of the model selection phase, also requires stable retention; Week 10's data governance steps represent a critical data quality component in real MDLC scenarios and are thus retained in the final version; Week 11 handles model interpretation and bias analysis, while Week 12 focuses on deployment preparation. By reconnecting these key modules, I ensured the final code flows seamlessly from raw features through training, evaluation, interpretation, and deployment.

The rationale behind these omissions is not to reduce content but to enhance professionalism in the process. The final delivery version retains only essential steps that genuinely impact model behavior, eliminating all experimental, redundant, or non-reproducible components. This streamlines the pipeline, making it clearer and more engineering-ready. This version now functions as a standard ML pipeline, aligning with the MDLC specifications required by the course and embodying the maintainability and interpretability sought in industrial model development.

3.

Following the creation of this streamlined version, our team further discussed that while concise and suitable as a final production-grade pipeline, it might have been trimmed too rigorously. We concluded that elements removed in the second version, specifically the Week 9 multi-model comparison, explanations in the model selection section, and Week 8/9 hyperparameter tuning records (including charts generated during model comparison and validation), all serve to demonstrate model justification within the strict MDLC documentation framework. Omitting these components prevented the document from fully showcasing our exploratory efforts during model selection. Consequently, we later reintroduced some content to generate a more comprehensive final version. This revised version reinstated the multi-model comparison, benchmark model performance table, concise model selection rationale, and key charts (e.g., bias plots and feature importance). This approach provides instructors with a more complete presentation of our learning journey and thought process, demonstrating that our final model selection was a deliberate choice among multiple models rather than arbitrary.