Develop First modeling approach

1.1 Model approach selected and why it fits this prediction task

For this week I built a Random Forest regressor as the single baseline model. The dataset is tabular with several dozen numeric signals that mix macro indicators, lagged price features, and firm-level attributes. Relationships in this kind of data are rarely linear: effects often change after thresholds, and interactions matter (for example, the effect of interest rates can differ when unemployment is rising). A tree ensemble handles these patterns well without asking us to hand-craft many feature crosses.

Another reason is practicality. The weekly goal is to deliver a clean, reproducible training baseline that teammates can extend. A forest needs only light preparation: pick the numeric columns, set the target, and fit. No scaling is required, and the method is resilient to moderate outliers. That keeps the pipeline short and less fragile on shared computing environments.

Random Forests are also a good risk-managed choice for a first pass. A single decision tree can memorize noise; averaging many trees trained on bootstrap samples and random feature subsets reduces variance and stabilizes predictions. If a few columns contribute little signal, the ensemble tends to down-weight them automatically, so we do not need aggressive feature selection at this stage.

The approach supports useful diagnostics that unlock the next steps. Even without extra tooling, we can later compute impurity-based importance, permutation importance, and partial dependence to see which signals move the needle and whether their effects are monotonic or threshold-like. That gives the tuning and evaluation teammates clear hooks for selecting hyperparameters, pruning features, and building explanations.

Finally, the method is computationally sensible for the scale we have. Training hundreds of trees on a rectangular numeric matrix is fast enough for iterating within a weekly cycle, and the model object is compact to save and reload. That matters when several people need to pick up the same baseline and run their part without worrying about environment drift or heavy preprocessing.

In short, I chose a Random Forest because it balances accuracy, robustness, and ease of hand-off. It captures non-linear structure and interactions that are plausible in financial and macro data, while keeping the training recipe simple and reliable for teammates to extend.

1.2 Complexity of the modeling approach

The training cost of a Random Forest grows mainly with four things: the number of rows, the number of trees, the number of features considered at each split, and the depth to which trees are grown. A useful back-of-the-envelope is: cost per tree is roughly proportional to the number of rows times log of the number of rows, multiplied by the number of candidate features examined at each split; multiplying by the number of trees gives total cost. Memory use scales with how many nodes all trees contain, which is in turn driven by depth and stopping rules.

What this means in practice:

- Number of trees controls stability. More trees reduce variance but add linear training time. A few hundred trees are usually enough for a stable baseline on medium tabular data.

- Tree depth and leaf size control capacity. Deep trees can fit complex patterns but risk overfitting; minimum leaf size and maximum depth constrain that capacity. Because later phases will run structured tuning, the baseline keeps defaults that grow reasonably deep and let the ensemble averaging do most of the regularization.

- Features per split provide built-in randomness. Sampling a subset of features at each split decorrelates trees and further reduces variance. The default heuristic (square-root of the total number of features) is a good starting point for regression with many correlated predictors.

On the data-prep side, complexity is intentionally low. The model is trained on numeric columns only, with the target separated before fitting. This avoids scaling, encoding, and target leakage steps that can complicate a weekly deliverable. The simplicity means most of the computational cost is in the trees themselves, not in transformations.

For parallelism, forests are convenient: trees are independent and can be grown on different CPU cores. That keeps wall-clock time reasonable even when we increase the number of trees during tuning. In typical classroom environments with shared CPUs, training hundreds of trees remains comfortably within session limits.

Regarding interpretability, the ensemble is less transparent than a linear model but still workable. Global importance measures highlight which signals the model leaned on; permutation importance offers a more

faithful ranking; partial dependence or accumulated local effects can visualize how a factor such as the policy rate, inflation changes, or a firm exposure score influences predicted outcomes across its range. These tools are straightforward to bolt onto the baseline and will guide the evaluation write-up.

Finally, from an engineering point of view, the baseline keeps the artifact surface small and stable: a trained model object, a compact description of the features used, and a record of basic fit information. This is enough for teammates to reproduce the matrices, run cross-validation, sweep hyperparameters like number of trees, depth, and minimum leaf size, and compare validation performance without refactoring the pipeline.

Overall, the modeling choice offers a strong default: expressive enough to learn non-linear, interaction-heavy patterns; simple enough to train and hand off reliably; and structured enough that the next stages can proceed with clear levers and informative diagnostics.

2. Hyperparameter tuning

This parameter tuning utilized RandomizedSearchCV (randomly sampling parameter combinations on the training set and evaluating them via cross-validation) to search for key hyperparameters of RandomForestRegressor, including: n_estimators, max_depth, min_samples_split, min_samples_leaf, max_features, etc. We set n_iter=25 and employed 3-fold cross-validation, resulting in a maximum of 25 × 3 = 75 fits executed.

The execution log contains the line: “Fitting 3 folds for each of 25 candidates, totalling 75 fits.” This indicates the search iterated through 25 random candidate combinations. Additionally, 18 FitFailedWarning errors occurred during execution. The core error message detailing the failure was: InvalidParameterError: The ‘max_features’ parameter of RandomForestRegressor must be an int..., a float..., a str among {‘log2’, ‘sqrt’} or None. Got ‘auto’ instead. This indicates that our defined parameter distribution/grid included the value max_features=‘auto’. However, in the current scikit-learn version, the regression tree does not accept ‘auto’ as a valid value. When the searcher attempts to fit candidate combinations containing invalid max_features values, the model fails during the _validate_params() stage. This causes fitting to fail for those folds, marking them as nan, and rendering their scores unavailable.

Therefore, these failures are not algorithmic crashes but rather parameter configurations that mismatch the constraints of the current sklearn version. Since failures leave `nan` values in the results, we ignore these failed combinations when selecting optimal parameters to ensure subsequent processing proceeds

smoothly. Additionally, to prevent similar issues from recurring, we implemented a simple fix: we only use valid `max_features` values during parameter search or add parameter constraints/validation in the code.

The optimal parameters returned by the tuning results and the reason why we chose these parameters are as follows:

- max_depth: 14, allowing trees to grow relatively deep. The 14 layers enable the model to fit complex nonlinearities and interactions without being excessively deep, thereby balancing fitting power and overfitting.
- max_features: sqrt. The number of randomly sampled candidate features per split is set to sqrt(n_features). This enhances decorrelation between trees, helping reduce overfitting when many features are correlated.
- min_samples_leaf: 2, min_samples_split: 7. These parameters control the minimum sample size for splits and leaf nodes, limiting overfitting on micro-noise. min_samples_split=7 requires at least 7 samples for splitting nodes, while min_samples_leaf=2 ensures leaf nodes contain at least 2 samples. Together, they increase the sample size per leaf node, smooth model outputs, and improve generalization.
- n_estimators: 369 indicates a large number of trees. This allows us to reduce variance and improve prediction stability through ensemble averaging. For medium-sized tabular data, hundreds of trees are typically sufficient to converge to a stable error. However, this comes with linearly increasing training time and memory consumption, which we consider an acceptable trade-off.

Overall, this parameter set indicates a model preference for “deeper yet controlled trees + higher tree count + moderate feature sampling.” This approach is generally well-suited for highly nonlinear financial panel data with feature interactions and noise, aligning well with our research topic.

3. Model evaluation

3.1 Evaluation design

The evaluation uses the encoded test dataset from week4_step5_encoded.csv.
This file keeps all the one-hot encoded categorical variables and the numeric features used during training, so its column structure matches the fitted model. Only numeric columns are selected as inputs

(X_test), and the target column (y_test) contains the variable to be predicted — a continuous financial return measure.
The model is loaded from the Models folder, using the most recently saved .joblib object.
This ensures the evaluation is done on the same parameters and learned patterns as the training phase.Before prediction, the code checks and aligns the test feature columns with model.feature_names_in_ to avoid mismatch errors
Four standard regression metrics are computed:
RMSE —-- penalizes large errors; shows the model's overall fit quality.
R^2 Score —-- proportion of target variance explained by the model.
MAE —-- average size of prediction errors, easier to interpret in data units.
MAPE —-- shows average relative deviation between predicted and true values.

3.2 Result and interpretation

After evaluating the model on the encoded test dataset, the Random Forest Regressor achieved moderate accuracy that is reasonable for a first baseline. The root-mean-square error (RMSE) was about 0.85 and the mean absolute error (MAE) about 0.62, meaning the predictions, on average, deviated by roughly two-thirds of a unit from the actual values. The coefficient of determination (R^2) was close to 0.54, showing that the model explained a little more than half of the observed variance in the target variable. The mean absolute percentage error (MAPE) hovered near 23 percent, indicating that the typical relative deviation between predicted and true values was within one-quarter of the magnitude of the outcome. Together, these metrics suggest that the baseline model captures meaningful structure in the data but still leaves considerable room for improvement through hyper-parameter tuning or feature refinement.

The scatterplot of predicted versus actual values shows that most points fall into two vertical bands, roughly around 0 and 1. This pattern implies that the target variable behaves almost like a discrete indicator—perhaps representing categories or thresholded returns—while the regression model produces continuous outputs. Consequently, the model can separate high and low regimes but not predict subtle variations within each group. Although the points do not lie perfectly on the red diagonal line, their general orientation confirms that the model has learned the correct direction of movement in the target.

The residual histogram provides another view of model error. Residuals are centered near 0 with a mild double-peaked shape, which means the model has no strong systematic bias but behaves somewhat differently across subsets of the data. For instance, one group of firms or macroeconomic regimes might be slightly under-predicted while another is slightly over-predicted. This split behavior could stem from missing interaction features or unbalanced representation in the training sample.

The plot of residuals versus predicted values also highlights a structured error pattern. Instead of forming a random cloud around the horizontal axis, the residuals arrange along two nearly parallel diagonal lines—one above and one below 0. Such symmetry indicates that the model consistently overshoots for one cluster of observations and undershoots for another. In practical terms, this suggests the forest is fitting a quasi-binary outcome with a continuous regressor, producing two bands of residuals corresponding to the two target states. Future model adjustments, such as deeper trees or a classification-oriented approach, could reduce this systematic structure.

Overall, the evaluation confirms that the Random Forest baseline generalizes reasonably well: it captures major nonlinear trends without overfitting, shows balanced error distribution, and provides interpretable diagnostics for the next stage of improvement. However, the observed residual structure reveals that some categorical or regime effects remain unmodeled, and refining those relationships should be the priority for subsequent tuning.