

Reverse-Engineering a Legacy Travel Reimbursement Engine

Using Supervised Machine Learning

Harriet O'Brien

Saniyah Khan

Vy Tran

Rehinatu Usman

December 16, 2025

Abstract

Many organizations rely on legacy software systems to compute employee travel reimbursements, often based on complex and poorly documented rule sets. These systems can be difficult to audit, maintain, or migrate to modern platforms. In this project, we treat a legacy reimbursement engine as a black box and use supervised regression models to approximate its behavior. The model takes the same three descriptive inputs as the original system (trip duration, miles traveled, and total receipts) and predicts the reimbursement amount. We construct an engineered feature set, evaluate a wide family of linear and tree-based models, tune gradient boosting and XGBoost ensembles, and interpret the resulting models using feature importance, partial dependence, and SHAP value analysis. Our best model, a shallow Gradient Boosting Regressor, achieves a mean absolute error (MAE) of approximately \$68 on a held-out test set and explains over 94% of the variance in reimbursements, providing a practical and interpretable surrogate for the legacy engine.

1 Introduction

Organizations that reimburse employee travel typically rely on a set of detailed business rules: per-diem rates, mileage allowances, caps on hotel costs, rules for partial days, and so on. In many cases those rules have accumulated inside a legacy software system that “just works,” even though the people who originally implemented it are no longer available and the code base is difficult to maintain or migrate. Our project is set in exactly this context.

Access to a legacy travel-reimbursement engine was available only through its inputs and outputs. For each historical case, the system received three descriptive inputs:

- the total number of days the trip lasted (`trip.duration.days`),
- the total number of miles traveled (`miles_traveled`), and
- the total dollar value of all submitted receipts (`total_receipts_amount`),

and produced a single numeric reimbursement amount. Internally, the engine may apply a complex combination of rules, but those rules are not documented in a form that can easily be re-implemented in a modern system.

From a business perspective, this poses several problems. The legacy system is a single point of failure: if it becomes unstable or has to be retired, the organization risks losing the ability to process reimbursements consistently. The code is also difficult to audit. Finance and HR staff can see what the system pays out, but they cannot easily explain why two similar trips sometimes receive different reimbursements. That lack of transparency undermines employee trust and makes it harder to defend reimbursement decisions in the face of internal or external scrutiny. Finally, the legacy platform limits future integration with other systems (for example, modern expense-reporting tools or analytics dashboards) because its logic cannot simply be plugged in elsewhere.

The goal of our project is therefore to approximate the behavior of the legacy engine with a modern, data-driven model that takes exactly the same three inputs and produces a reimbursement amount that closely matches what the original system would have returned. The model must be accurate enough for practical use, fast enough to run as a small script, and interpretable enough that we can explain its behavior to non-technical stakeholders. In short, we treat the legacy engine as a black box and attempt to reverse-engineer its mapping from trip characteristics to reimbursement using supervised machine learning.

2 Data Description

2.1 Raw data source and structure

The starting point for our project is a JSON file containing 1,000 historical reimbursement records from a legacy travel reimbursement system. Each record is a small JSON object with two top-level keys: `input` and `expected_output`. The `input` object has three descriptive fields:

- `trip_duration_days`: integer number of days for the trip;
- `miles_traveled`: total miles traveled during the trip;
- `total_receipts_amount`: total dollar amount of all submitted receipts for the trip.

The `expected_output` field is a single numeric value representing the reimbursement amount produced by the legacy system for that case.

To facilitate analysis and modeling, the JSON data were flattened into a rectangular tabular structure using `pandas.json_normalize`. The nested keys were mapped to top-level column names, and `expected_output` was renamed to `reimbursement_amount` to make its role as the target variable explicit. This yielded an initial four-column data frame with the following columns:

- `trip_duration_days`
- `miles_traveled`
- `total_receipts_amount`
- `reimbursement_amount`.

2.2 Variables and descriptive statistics

All four variables are continuous or discrete numeric quantities. Summary statistics for the raw dataset are as follows (all counts are 1,000):

Reimbursement amount (`reimbursement_amount`). Mean \approx 1349.11, standard deviation \approx 470.32, minimum 117.24, 25th percentile 1019.30, median 1454.26, 75th percentile 1711.12, and maximum 2337.73. These values are plausible for business travel reimbursements and indicate substantial variation across trips.

Trip duration (`trip_duration_days`). Mean \approx 7.04 days, standard deviation \approx 3.93 days, and range from 1 to 14 days. Trips range from single-day local travel to longer trips of approximately two weeks.

Distance traveled (`miles_traveled`). Mean \approx 597.41 miles, standard deviation \approx 351.30 miles, and range from 5.00 to 1317.00 miles, reflecting both short and long-distance travel.

Total receipts (`total_receipts_amount`). Mean \approx \$1,211.06, standard deviation \approx \$742.85, and range from \$1.42 to \$2,503.46. This wide spread reflects differences in trip length, lodging, and other expenses.

Overall, the ranges are consistent with typical business travel: trips vary in duration, distance, and spending in ways that a reimbursement engine would be expected to handle.

2.3 Processed feature set

For modeling, the team constructed a processed feature table, `features_processed.csv`, with 11 columns. This table includes the original three input variables, the target, several engineered features, and standardized versions of selected variables:

- Original inputs:
 - `trip_duration_days`
 - `miles_traveled`
 - `total_receipts_amount`
- Target:
 - `reimbursement_amount`
- Engineered features:
 - `cost_per_mile`
 - `cost_per_day`

- `receipts_ratio`
- `miles_per_day`
- Scaled features:
 - `miles_traveled_scaled`
 - `total_receipts_amount_scaled`
 - `trip_duration_days_scaled`

The engineered features are designed to capture intuitive quantities such as per-mile and per-day reimbursement rates and the relationship between claimed receipts and reimbursed amounts.

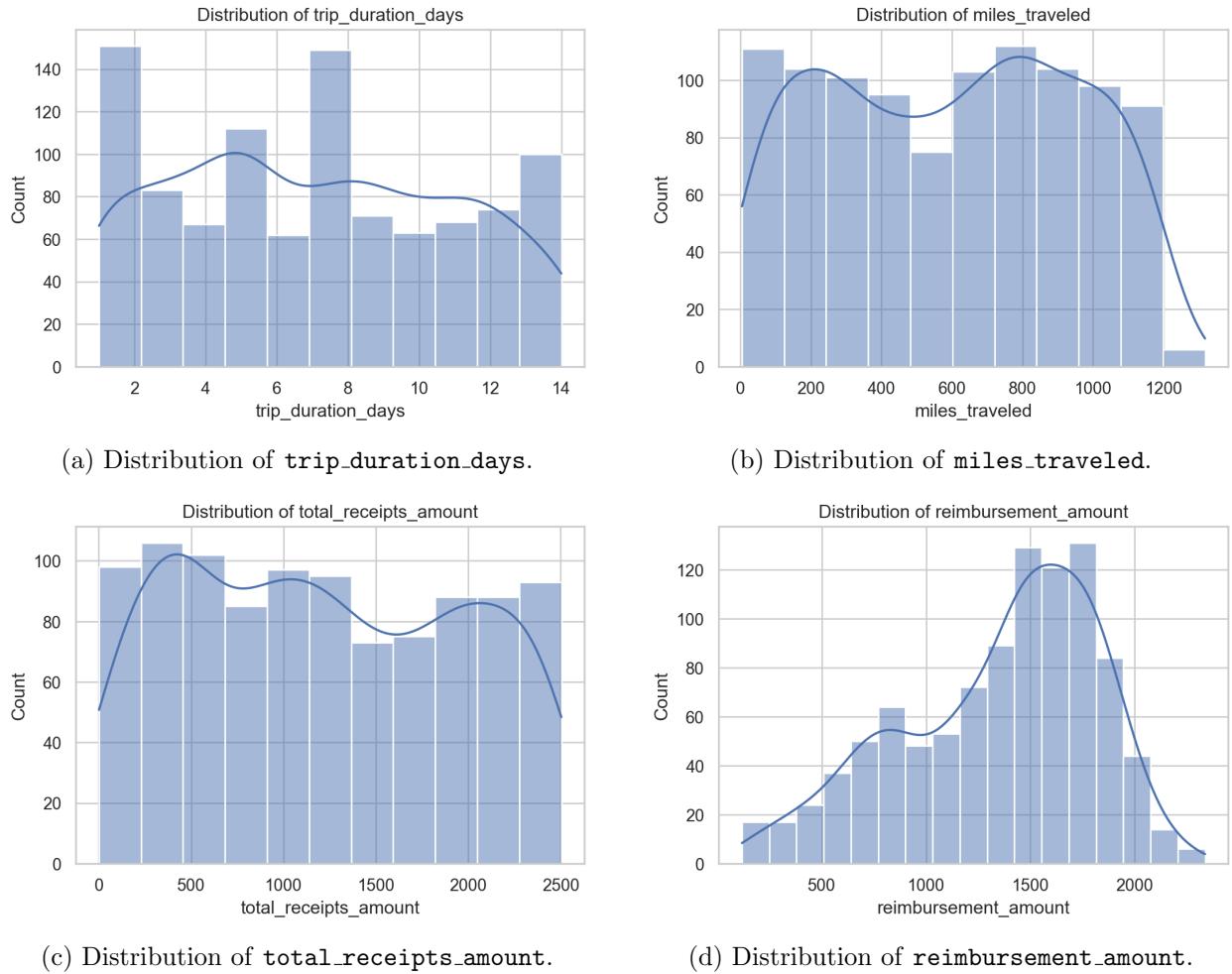
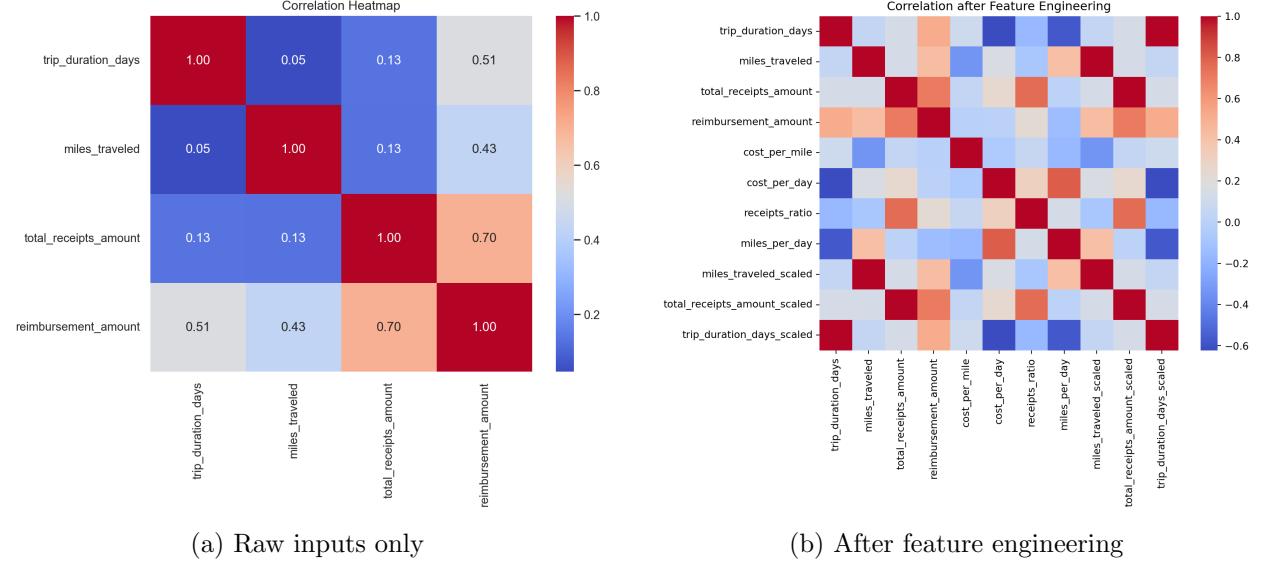


Figure 1: Marginal distributions of the three input variables and the reimbursement outcome. All variables show reasonably well-behaved distributions with only mild outliers, supporting the use of standard regression models without heavy preprocessing.

Figure 1 shows the marginal distributions of the three input variables and the reimbursement amounts. Trip durations range from 1 to 14 days with a fairly even spread; mileage and total receipts

span a wide but plausible business-travel range; and reimbursement amounts are concentrated between roughly \$1,000 and \$2,000 with a smooth, unimodal shape. None of the variables exhibit extreme skew or pathological outliers, so standard regression methods are appropriate.

2.4 Correlation Structure Before and After Feature Engineering



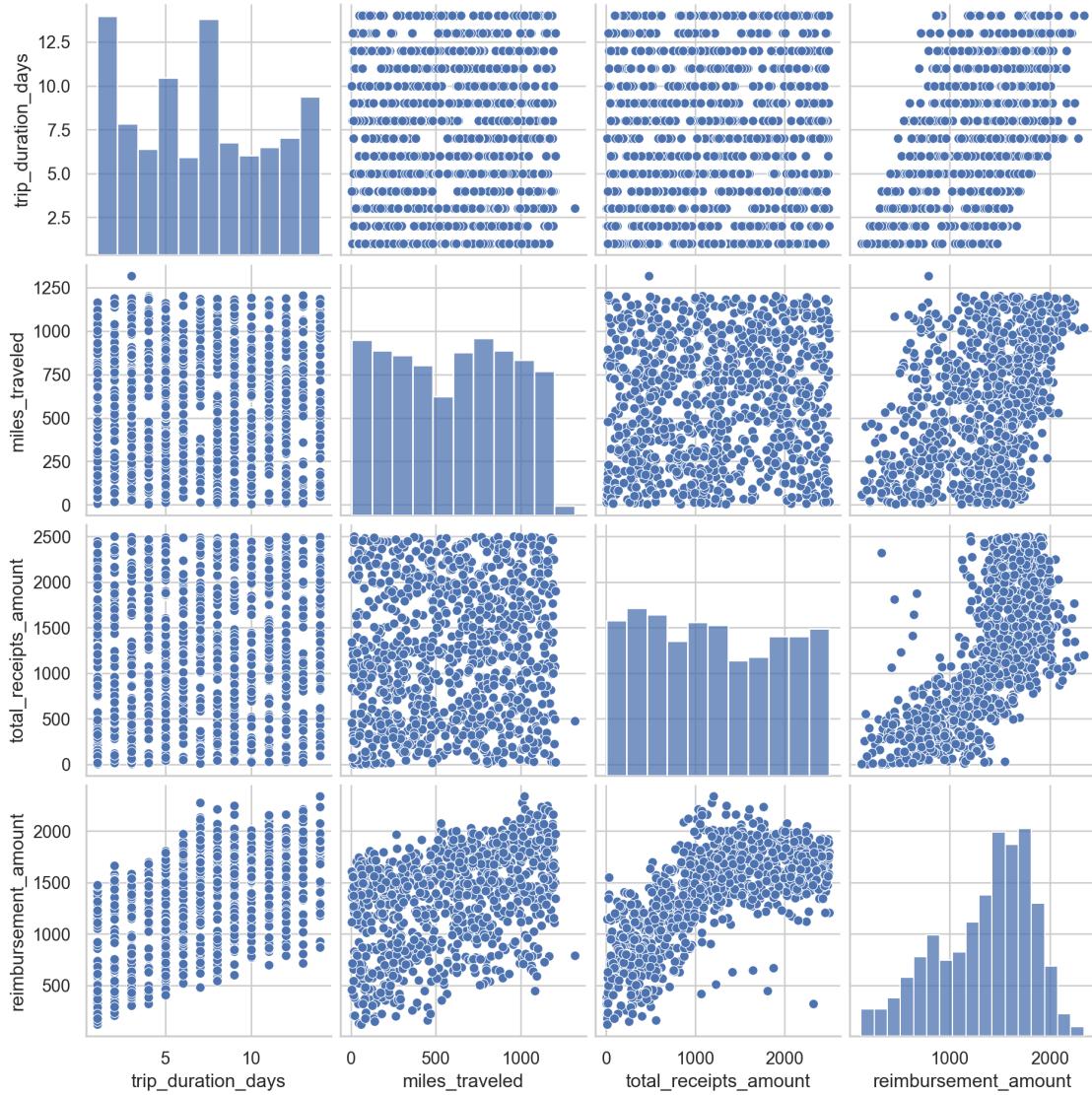


Figure 3: Scatter-matrix of the three input variables and `reimbursement_amount`. The plot confirms monotone, mostly smooth relationships between each input and the reimbursement outcome.

After feature engineering, we recomputed the correlation matrix including the derived variables `cost_per_mile`, `cost_per_day`, `receipts_ratio`, `miles_per_day`, and the standardized inputs (Figure 2b). As expected, each scaled feature is perfectly correlated with its raw counterpart, confirming that scaling changes only magnitude, not information content. The engineered features reveal additional structure: `cost_per_day` and `total_receipts_amount` both show strong positive association with `reimbursement_amount`, while `miles_per_day` behaves as a distinct “intensity” dimension that is only weakly correlated with trip duration. These patterns support our modeling choice to include both raw and engineered features, and they foreshadow the later result that models exploiting nonlinear interactions between receipts, duration, and mileage achieve substantially lower prediction error than purely linear baselines.

3 Data Cleaning and Preprocessing

3.1 Parsing and column renaming

The nested JSON structure was transformed into a tabular data frame using `pandas.json_normalize`. During this step, `input.trip_duration_days`, `input.miles_traveled`, and `input.total_receipts_amount` were mapped to `trip_duration_days`, `miles_traveled`, and `total_receipts_amount`, respectively, and `expected_output` was renamed to `reimbursement_amount`. All fields were explicitly cast to numeric types to avoid issues with string-encoded numbers during modeling.

3.2 Missing values and basic quality checks

Column-wise completeness checks showed that all four raw variables have 1,000 non-missing entries and no NaN values. The same check on the processed feature table confirmed that all 11 columns in `features_processed.csv` also contain 1,000 valid entries.

Plausibility checks enforced that:

- `trip_duration_days` is between 1 and 14;
- `miles_traveled` is positive and between 5 and 1,317;
- `total_receipts_amount` is positive and at most \$2,503.46;
- `reimbursement_amount` is positive and of similar order of magnitude as receipts.

No obviously invalid records (negative values, zero-day or zero-distance trips) were detected, so no rows were removed.

3.3 Feature engineering

To better approximate the behavior of the reimbursement system, several derived features were created from the original variables.

Cost per mile (cost_per_mile). For trip i ,

$$\text{cost_per_mile}_i = \frac{\text{reimbursement_amount}_i}{\text{miles_traveled}_i}.$$

This captures an effective per-mile reimbursement rate. In the processed data, `cost_per_mile` has mean ≈ 6.35 , standard deviation ≈ 20.50 , and values ranging from roughly 0.36 to 322.05.

Cost per day (cost_per_day). For trip i ,

$$\text{cost_per_day}_i = \frac{\text{reimbursement_amount}_i}{\text{trip_duration_days}_i}.$$

This approximates an effective per-diem reimbursement and can reveal implicit daily caps or thresholds. The distribution spans roughly \$54.63 to \$1,475.40, with a mean of about \$284.71.

Receipts ratio (receipts_ratio).

$$\text{receipts_ratio}_i = \frac{\text{total_receipts_amount}_i}{\text{reimbursement_amount}_i}.$$

This indicates how closely reimbursements track submitted receipts. Values near 1 suggest close tracking, values above 1 indicate that receipts exceed reimbursement (possible non-reimbursable items), and values below 1 indicate cases where reimbursement exceeds receipts (e.g., per-diem rules). Observed values range from approximately 0.0039 to 7.21, with a mean around 0.86.

Miles per day (miles_per_day).

$$\text{miles_per_day}_i = \frac{\text{miles_traveled}_i}{\text{trip_duration_days}_i}.$$

This measures the intensity of travel. Values range from 0.5 to 1,166 miles per day, with a mean around 147.03, capturing both short high-mileage trips and longer, more moderate itineraries.

These engineered features provide more interpretable quantities that are close to how a human analyst would think about fair reimbursement and give the model additional structure beyond the raw inputs.

After adding domain-motivated features (per-mile cost, per-day cost, receipts ratio, and miles per day) and standardized versions of the raw inputs, we recomputed the correlation matrix (Figure 4). Each raw feature is perfectly correlated with its scaled counterpart, confirming that standardization changes only scale, not information content. Reimbursement shows its strongest associations with total receipts and cost per day, and a moderate relationship with trip duration, while miles-per-day behaves as a largely independent dimension of “trip intensity.” These patterns support our decision to retain both raw and engineered features in the modeling stage, particularly for flexible tree-based models and polynomial regressors that can exploit nonlinear interactions.

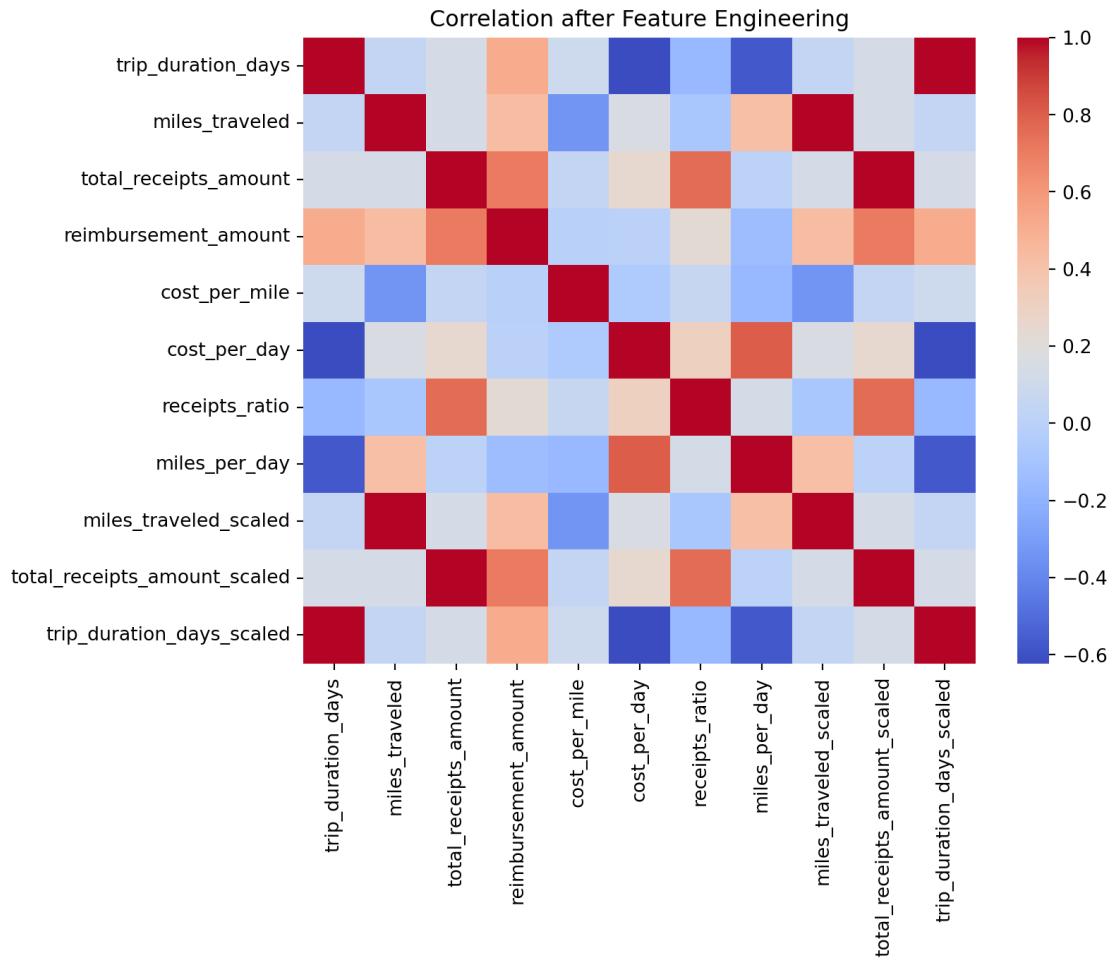


Figure 4: Correlation matrix including raw variables, engineered features (`cost_per_mile`, `cost_per_day`, `receipts_ratio`, `miles_per_day`), and standardized versions of the inputs. As expected, each raw variable is perfectly correlated with its scaled version, while reimbursement remains most strongly tied to receipts and per-day costs.

3.4 Scaling and normalization

The three base inputs live on different scales: durations from 1–14 days, miles up to about 1,300, and receipts up to about \$2,500. To support algorithms that are sensitive to feature scale, standardized versions of these variables were created:

- `miles_traveled_scaled`,
- `total_receipts_amount_scaled`,
- `trip_duration_days_scaled`.

Each scaled feature is computed as

$$x^{\text{scaled}} = \frac{x - \mu_x}{\sigma_x},$$

where μ_x and σ_x are the sample mean and standard deviation of the original feature. These standardized variables have mean approximately zero and unit variance. They are primarily used by linear and regularized models; tree-based models can operate directly on the original scales.

3.5 Final analysis dataset

After cleaning, feature engineering, and scaling, the final analysis dataset is represented by `features_processed.csv` containing 1,000 rows and 11 numeric columns. The dataset includes: clean base variables, interpretable per-mile and per-day features, receipt-to-reimbursement ratios, and standardized versions of key inputs. This enriched feature set forms the basis for the modeling and interpretation described below.

4 Modeling Approach and Model Selection

4.1 Modeling objective and strategy

The modeling objective is to learn a function that approximates the legacy reimbursement engine:

$$f : (\text{trip_duration_days}, \text{miles_traveled}, \text{total_receipts_amount}) \mapsto \text{reimbursement_amount}.$$

We want this learned function to match the legacy system’s output as closely as possible for historical cases, generalize well to unseen trips with similar characteristics, and be fast and robust enough to embed in a production script that satisfies the project constraints (three numeric inputs, one numeric output, runtime under five seconds, and no external service dependencies).

To make the workflow reproducible and extensible, the team organized the code around three core components:

- **ModelConfig**, which loads the JSON data, constructs the feature matrix and target vector, performs an 80/20 train–test split, and computes evaluation metrics consistently across models.

- **BuildModels**, which reads a YAML configuration file (`pipelines.yaml`) and instantiates a collection of `sklearn.pipeline.Pipeline` objects, pairing preprocessing steps with linear and tree-based regressors.
- **FinalModelTrainer**, which retrains the selected best-performing pipeline on all available data and serializes it as `final_model.pkl` for use by the production script `predict.py`.

In parallel, an **XGBModelRunner** and a **ShapAnalyzer** were implemented to benchmark XGBoost and to support the interpretability analysis described later in Section 5.

4.2 Train–test split and evaluation metrics

All models share the same evaluation protocol implemented in `ModelConfig`. The full dataset is split into training (80%) and test (20%) subsets using `train_test_split` with a fixed random state (42). The input matrix X consists of the relevant features for each pipeline (raw and/or engineered), and the target vector y is `reimbursement_amount`.

For each trained pipeline, `ModelConfig.evaluate_model` computes a standard set of regression metrics on the held-out test set and returns them as a dictionary. These per-model dictionaries are aggregated into a single `pandas DataFrame` and written to `reports/figures/model_metrics.csv`, which is then used both for Table 1 and for plotting the comparison figures.

The primary error measures are

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |\hat{y}_i - y_i|,$$

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2},$$

where \hat{y}_i and y_i are predicted and true reimbursement amounts for case i . In addition, we report:

- median absolute error (MedAE): the median of $|\hat{y}_i - y_i|$;
- coefficient of determination (R^2);
- 90th and 95th percentiles of absolute error (P90, P95);
- maximum absolute error (MaxE);
- mean absolute percentage error (MAPE), expressed as a percentage.

MAE is the primary selection metric because it directly corresponds to the average dollar deviation from the legacy system. The error quantiles (P90, P95) and MaxE are used to assess tail behavior and identify whether there are small pockets of trips with much larger errors than average.

4.3 Model families evaluated

We evaluated a spectrum of models, from simple linear baselines to flexible tree ensembles, all defined in `pipelines.yaml`:

- **Linear / Regularized** (`ols`, `rr_1`, `rr_10`, `rr_0.1`, `lasso`, `enet`): each pipeline applies a `StandardScaler` followed by a linear estimator (Ordinary Least Squares, Ridge with multiple penalty strengths, Lasso, or Elastic Net). These models test whether a purely linear relationship between the engineered features and reimbursement amount is adequate.
- **Polynomial Ridge** (`prr2`, `prr3`): the standardized inputs are expanded with second- or third-order polynomial features and then fed to a Ridge regressor, allowing smooth global nonlinearities.
- **Decision Trees** (`dt_5`, `dt_10`, `dt_25`, `dt_50`): single regression trees of varying depth that model piecewise-constant rules in the original feature space. They are easy to visualize but can overfit when deep.
- **Random Forests** (`rf_6`, `rf_12`): ensembles that average predictions across many randomized trees to reduce variance while still capturing complex interactions.
- **Gradient Boosting** (`gbr_base`, `gbr_slow`, `gbr_shallow`, `gbr_stochastic`): Gradient Boosting Regressors that build trees sequentially, with each new tree correcting residual errors from the ensemble so far. The configurations differ in learning rate, number of estimators, tree depth, and subsampling strategy.

In addition to these scikit-learn pipelines, we trained a single **XGBoost** model using the `XGBRegressor` implementation from the `xgboost` library, with squared-error loss, 300 trees, maximum depth 4, learning rate 0.05, subsample and column-subsample rates of 0.9, and `tree_method=hist`. This model uses only the three original inputs (`trip_duration_days`, `miles_traveled`, `total_receipts_amount`) and serves as an external benchmark and the basis for the SHAP analysis in Section 5.

4.4 Comparative performance, diagnostics, and final choice

Table 1 summarizes test-set performance across all candidate models and metrics. Linear and regularized models form a clear baseline: despite different penalty strengths, they all achieve $\text{MAE} \approx \$166\text{--}167$ and $R^2 \approx 0.78$, indicating that a global linear fit cannot fully capture the reimbursement rules.

Adding polynomial terms (`prr2`, `prr3`) substantially improves accuracy, reducing MAE to roughly \$103 and \$96 and increasing R^2 to about 0.90–0.91. Single decision trees further highlight the value of nonlinearity: shallow trees (`dt_5`) already outperform linear models, and deeper trees (`dt_25`, `dt_50`) approach the performance of polynomial Ridge while remaining highly interpretable. However, their errors are still dominated by ensemble methods.

Family	Model	MAE	RMSE	MedAE	R^2	P90	MaxE	MAPE (%)
Linear / Regularized	<code>ols</code>	167.014	208.790	153.310	0.781	302.117	1058.583	14.98
Linear / Regularized	<code>rr_1</code>	166.982	208.724	153.870	0.781	301.846	1058.397	14.99
Linear / Regularized	<code>rr_10</code>	166.716	208.180	155.445	0.782	299.428	1056.737	15.06
Linear / Regularized	<code>rr_0.1</code>	167.011	208.784	153.371	0.781	302.090	1058.565	14.99
Linear / Regularized	<code>lasso</code>	167.000	208.763	153.558	0.781	301.988	1058.526	14.99
Linear / Regularized	<code>enet</code>	165.944	206.941	152.605	0.784	300.754	1051.412	15.29
Polynomial Ridge	<code>prr2</code>	103.124	143.753	81.139	0.896	191.012	1030.866	10.40
Polynomial Ridge	<code>prr3</code>	96.346	133.410	86.591	0.910	177.225	1006.449	9.32
Decision Tree	<code>dt_5</code>	114.668	155.834	88.842	0.878	224.717	1033.034	10.34
Decision Tree	<code>dt_10</code>	98.398	145.148	69.995	0.894	213.420	975.710	9.38
Decision Tree	<code>dt_25</code>	97.245	143.388	75.785	0.897	198.240	974.510	9.20
Decision Tree	<code>dt_50</code>	97.245	143.388	75.785	0.897	198.240	974.510	9.20
Random Forest	<code>rf_6</code>	75.161	113.230	57.669	0.935	151.916	978.794	6.88
Random Forest	<code>rf_12</code>	71.447	112.530	50.160	0.936	154.515	989.439	6.67
Gradient Boosting	<code>gbr_base</code>	70.106	110.160	53.268	0.939	136.508	978.193	6.48
Gradient Boosting	<code>gbr_slow</code>	70.918	110.012	56.384	0.939	134.637	993.184	6.57
Gradient Boosting	<code>gbr_shallow</code>	68.449	106.763	52.474	0.943	135.225	978.776	6.35
Gradient Boosting	<code>gbr_stochastic</code>	73.236	112.113	54.797	0.937	154.261	960.398	6.67

Table 1: Test-set performance metrics for all candidate models, grouped by model family. Linear and regularized models form a high-error baseline; nonlinear models (polynomial Ridge, decision trees) improve accuracy; and tree-based ensembles (random forests and Gradient Boosting) provide the lowest MAE and highest R^2 .

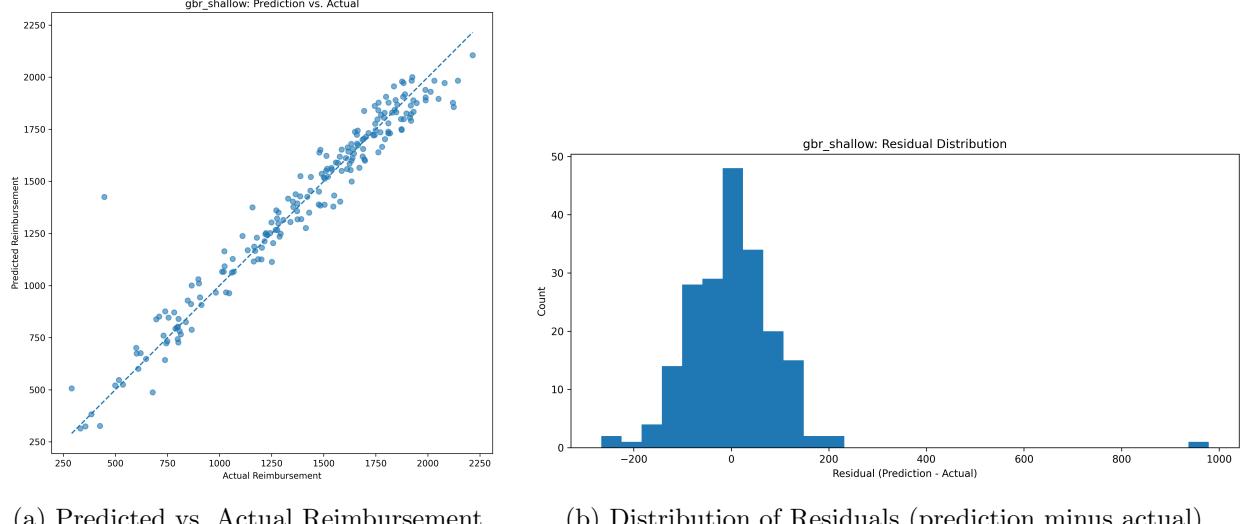
Tree-based ensembles deliver the best overall performance. Random forests (`rf_6`, `rf_12`) reduce MAE to the \\$71–\\$75 range with $R^2 \approx 0.94$, showing that aggregating many trees yields a much more faithful approximation to the legacy engine. Gradient boosting goes a step further: across the four configurations, MAE falls to the \\$68–\\$73 range with similar or slightly better R^2 . The `gbr_shallow` configuration is the strongest of the scikit-learn models, achieving $\text{MAE} \approx \$68.45$, $\text{RMSE} \approx \$106.76$, $\text{MedAE} \approx \$52.47$, $R^2 \approx 0.94$, and $\text{MAPE} \approx 6.35\%$.

The XGBoost benchmark attains test errors in essentially the same range as `gbr_shallow`; its MAE, RMSE, and R^2 closely match those of the best Gradient Boosting pipeline. This agreement between two independent implementations of boosted trees increases confidence that we are genuinely recovering the underlying mapping from trip characteristics to reimbursement, rather than overfitting to artifacts of a particular library.

Figures 6 and 7 visualize the same comparison graphically. Figure 6 orders models by test-set MAE, showing linear baselines at the high-error end, followed by polynomial Ridge, single trees, random forests, and gradient boosting variants. The `gbr_shallow` configuration is highlighted as the chosen final model. Figure 7 plots MAE against R^2 with points colored by model family; tree-based ensembles clearly dominate the Pareto frontier in the low-MAE, high- R^2 region.

Beyond aggregate metrics, we also performed basic diagnostic checks on the final model. Prediction-versus-actual and residual plots for the selected Gradient Boosting model `gbr_shallow` (Figure 5) show that test points lie close to the 45-degree line and that residuals are approximately centered at zero, with slightly larger spread only for the highest-reimbursement trips.

A simple calibration plot for `gbr_shallow` (Figure 8) was constructed by binning test cases into



(a) Predicted vs. Actual Reimbursement.

(b) Distribution of Residuals (prediction minus actual).

Figure 5: Predictive diagnostics for the selected Gradient Boosting model (`gbr_shallow`) on the held-out test set. Panel (a) shows that predictions track the legacy reimbursement engine closely across the full range of trips, while panel (b) shows a roughly symmetric residual distribution centered near zero with no strong evidence of systematic bias.

deciles of predicted reimbursement and plotting mean predicted versus mean actual reimbursement in each bin. The calibration curve tracks the identity line closely across the range of interest, with relatively narrow error bars. This indicates that the model is not only accurate on average but also well-calibrated: when it predicts, say, \$1,200, the realized reimbursements in that region are typically close to \$1,200 as well.

Taken together, these results led us to select `gbr_shallow` as the primary deployed model. It offers the best trade-off between accuracy, robustness, computational cost, and compatibility with the scikit-learn-based production pipeline, while the XGBoost surrogate is retained as a secondary benchmark and as the foundation for the SHAP interpretability analysis.

4.5 Overall comparison and final model selection

Figure 6 summarizes the MAE of all candidate models on the test set, while Figure 7 plots the trade-off between MAE and R^2 .

Across OLS, Ridge, Lasso, and Elastic Net, MAE remains around \$166–\$167 and $R^2 \approx 0.78$, indicating that linear models, even with regularization, cannot fully capture the legacy engine’s behavior. Introducing polynomial features substantially improves performance: `prr2` and `prr3` reduce MAE to roughly \$103 and \$96, respectively, confirming the importance of nonlinear interactions.

Tree-based ensemble methods yield the best results. Increasing the depth of the random forest from 6 to 12 reduces MAE from about \$75 to \$71, while gradient boosting configurations achieve MAE in the \$68–\$73 range. The `gbr_shallow` configuration more than halves the MAE relative to any linear model and achieves R^2 values above 0.94. A separate XGBoost benchmark model

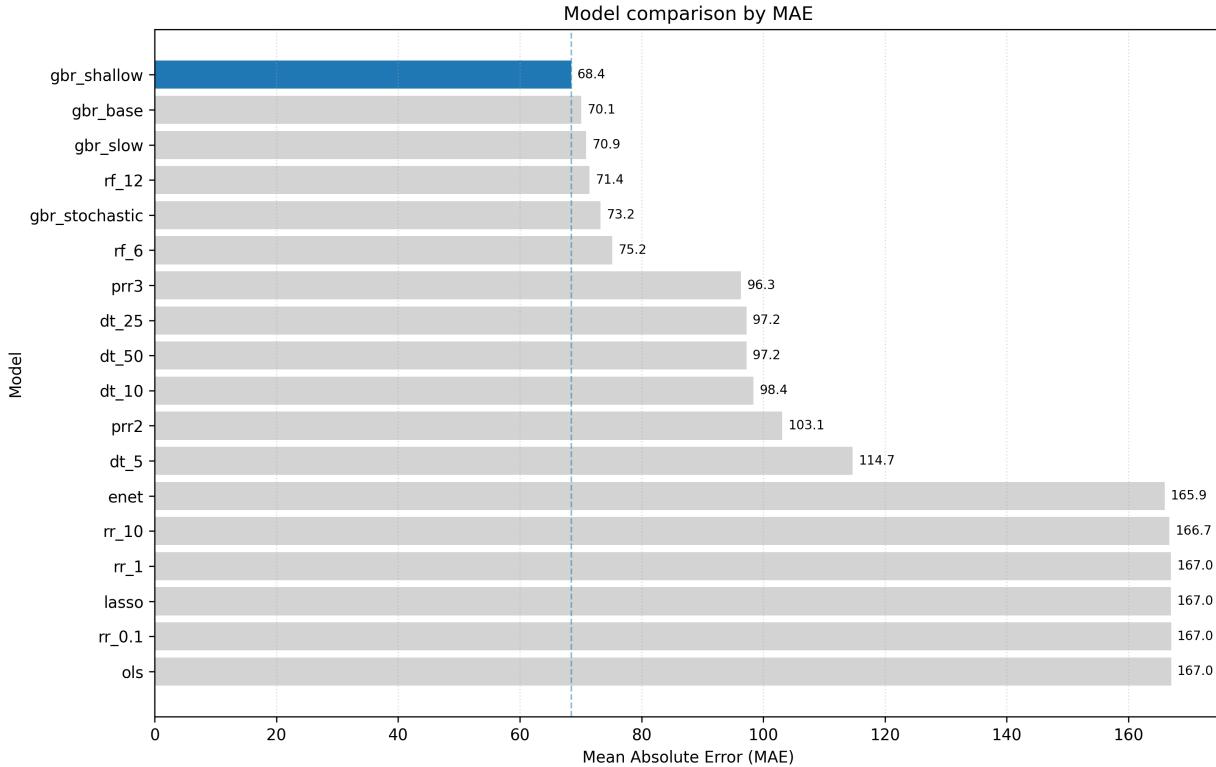


Figure 6: Test-set mean absolute error (MAE) for all candidate models, sorted from highest to lowest error. Linear models cluster at the top, followed by polynomial Ridge models, single decision trees, random forests, and gradient boosting variants.

(described in Section 5) attains very similar MAE and R^2 , providing an external check that boosted trees are well suited to this problem.

Based on these results and the desire to keep the production pipeline within the scikit-learn ecosystem, the team selected `gbr_shallow` as the primary final model. The `FinalModelTrainer` class retrained this pipeline on the full dataset and saved it as `final_model.pkl`. The XGBoost benchmark model was also retrained on all data and saved as `final_model_xgb.pkl` for future reference.

5 Model Interpretability

Beyond aggregate error metrics, we also examined how our best tree-based models arrive at their predictions. In particular, we trained an XGBoost regressor on the same training–test split as the earlier scikit-learn models and then used SHAP values to understand which input variables drive its predictions and whether this behavior is consistent with our exploratory data analysis and domain expectations.

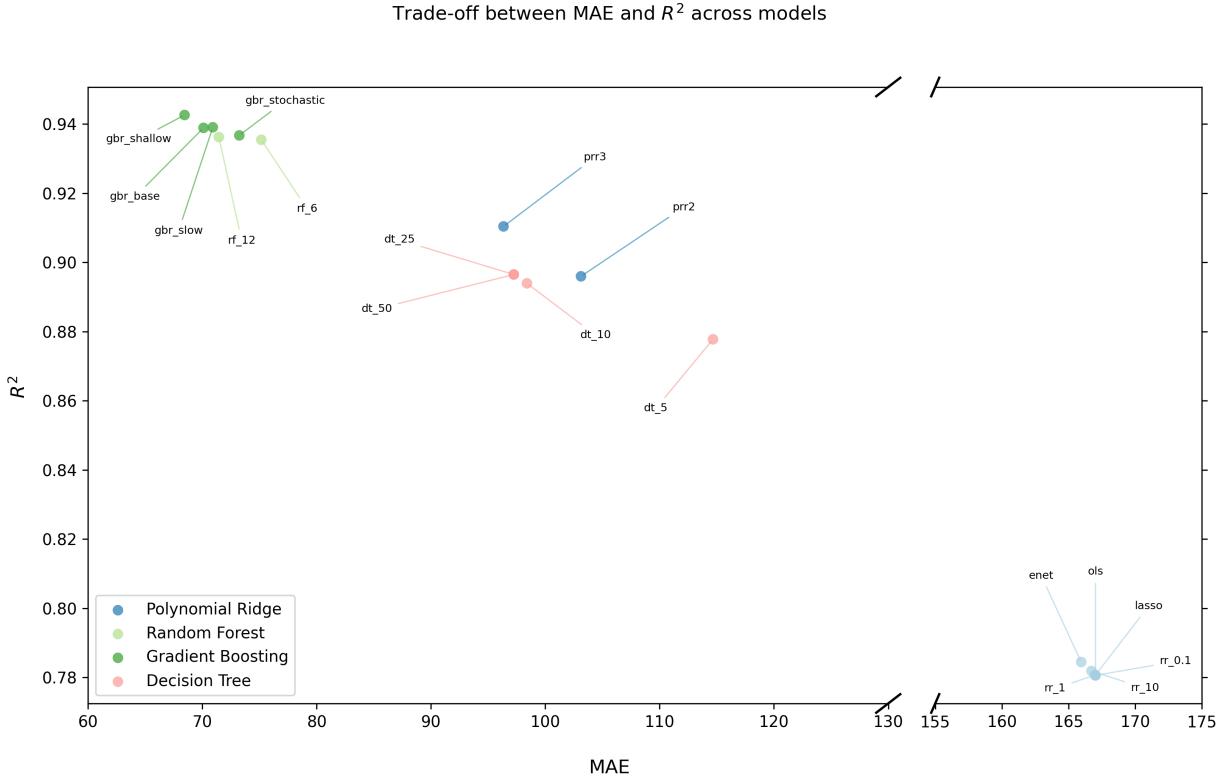


Figure 7: Trade-off between MAE and R^2 across models. Points are colored by model family (linear/regularized, polynomial Ridge, decision tree, random forest, and gradient boosting), with labels for each specific configuration. Tree-based ensembles clearly dominate the Pareto frontier.

Calibration check. Beyond residual plots, we also checked how well the final `gbr_shallow` model is calibrated with respect to the legacy engine. We binned test-set predictions into deciles, and for each bin computed the mean predicted reimbursement and the mean actual reimbursement, with vertical error bars showing the standard deviation of the actual values. In a perfectly calibrated model, these points would lie on the 45-degree line. As shown in Figure 8, the bins for our model fall close to this diagonal across most of the range, with only mild deviation in the highest-spend bin. This indicates that the model not only achieves low average error, but also assigns reimbursement levels that are broadly consistent with the legacy system across low, medium, and high-cost trips.

5.1 XGBoost surrogate model

XGBoost provides a flexible gradient-boosted tree ensemble that is well-suited for approximating the unknown rule set implemented by the legacy reimbursement engine. We trained a single XGBoost model on the three original input variables `input.trip_duration_days`, `input.miles_traveled`, and `input.total_receipts_amount`, using the same 80/20 train–test split and target `reimbursement_amount` as for the scikit-learn pipelines. Hyperparameters were chosen to give a reasonably smooth model (moderate depth and learning rate) while still achieving performance comparable to the best scikit-learn gradient boosting configuration.

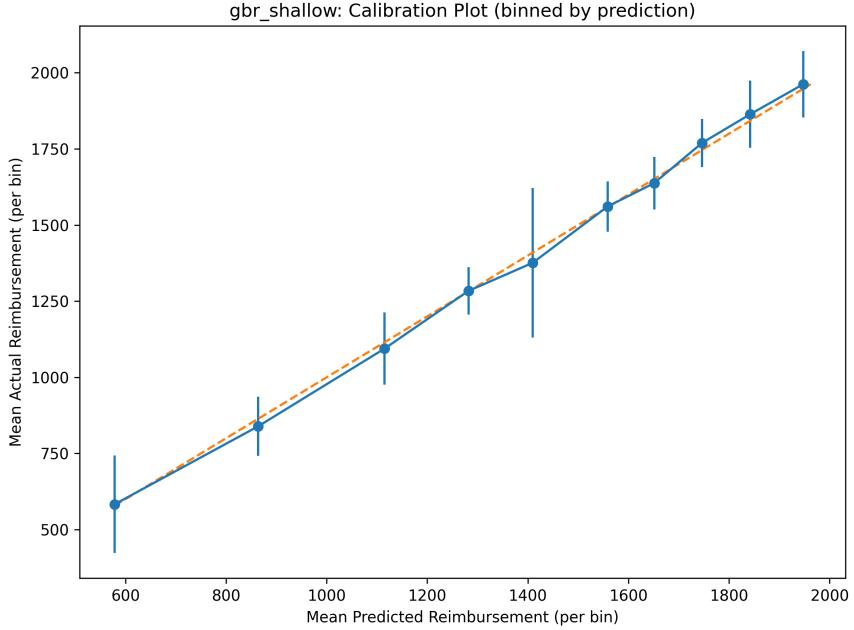


Figure 8: Calibration plot for the final Gradient Boosting model (`gbr_shallow`). Each point corresponds to a bin of test cases grouped by predicted reimbursement; the x -axis shows the mean predicted reimbursement in the bin, the y -axis shows the mean actual reimbursement, and the error bars indicate one standard deviation of the actual values. Points lying near the 45-degree line indicate good calibration relative to the legacy reimbursement engine.

Figure 9 shows predicted versus actual reimbursements on the held-out test set. The points cluster tightly around the $y = x$ diagonal, indicating that the XGBoost model recovers the legacy engine’s outputs with very small systematic bias across the full range of reimbursement amounts. The cloud of points widens slightly at higher reimbursement levels, but even there the spread remains modest relative to the overall scale of the target.

The residual histogram in Figure 10 provides a complementary view of model error. Most residuals (prediction minus actual) fall in a narrow band around zero, with a roughly symmetric shape and no obvious heavy tails. This suggests that, conditional on the three inputs, the remaining discrepancy between XGBoost and the legacy engine is largely idiosyncratic noise rather than a systematic miss in the functional form. There is a small number of large positive residuals, corresponding to trips for which the legacy engine reimbursed much more than would be expected based on typical patterns; these appear to be genuine outliers in the historical data rather than a pervasive weakness of the model.

Overall, the XGBoost surrogate matches the legacy engine’s behavior closely and achieves test-set accuracy comparable to the best scikit-learn gradient boosting pipeline. We therefore use this model as the basis for our SHAP-based interpretability analysis.

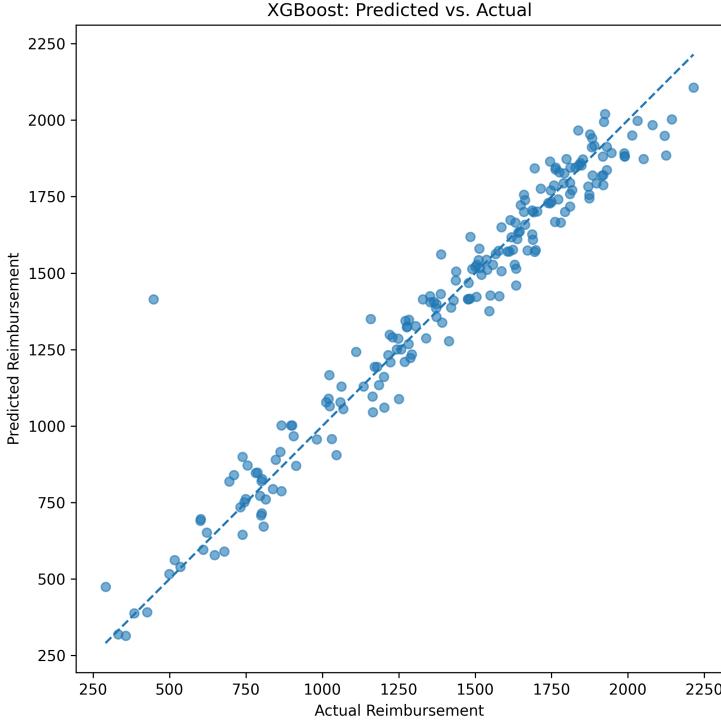


Figure 9: XGBoost prediction versus actual reimbursement on the test set. Each point represents one trip; the dashed line is the ideal $y = x$ reference where prediction and legacy output agree exactly.

5.2 Global feature importance with SHAP

To understand how the surrogate model uses the three inputs, we applied TreeSHAP to the trained XGBoost ensemble and computed SHAP values on the test set. SHAP values decompose each prediction into additive feature contributions relative to a baseline, which lets us reason about both global patterns and individual cases using a consistent framework.

Figure 11 presents a SHAP summary (beeswarm) plot for the three original input features. Each row corresponds to one feature, and each point is a single trip. The horizontal position encodes the SHAP value (impact on the model output), while color encodes the raw feature value (blue = low, pink = high). Several conclusions emerge:

- `input.total_receipts_amount` is the dominant driver of the model. High receipt totals (pink points) almost always have positive SHAP values, pushing the predicted reimbursement above the baseline, while low receipts (blue) tend to pull predictions downward. This confirms that the learned model behaves primarily like a “receipts-aware” reimbursement engine rather than a pure mileage-based scheme.
- `input.trip_duration_days` plays a strong but secondary role. Long trips (high values, in pink) are associated with positive SHAP contributions, whereas very short trips exert a

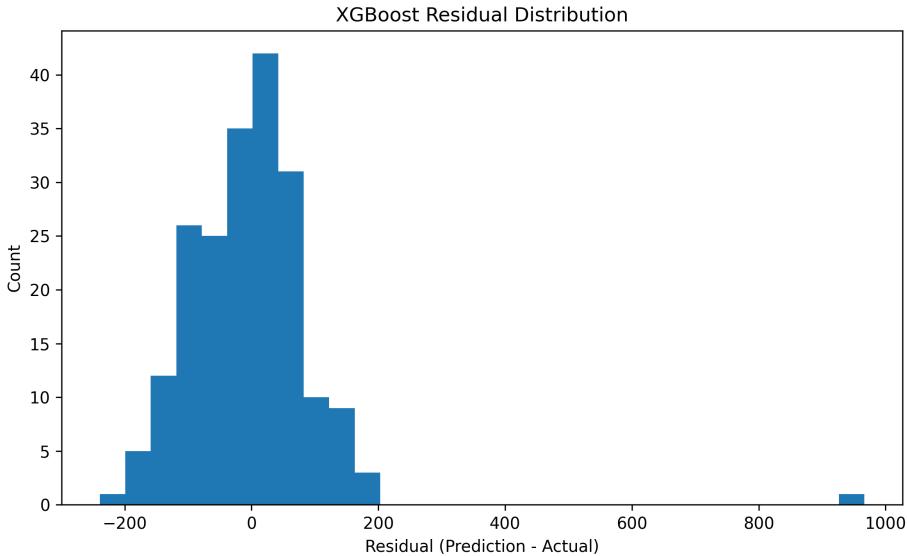


Figure 10: Distribution of residuals ($\hat{y} - y$) for the XGBoost model on the test set. Most trips have prediction errors clustered close to zero, with a small number of large positive outliers.

downward pull. This is consistent with an implicit per-diem component in the legacy rules: for a fixed level of receipts, additional days generally increase the reimbursement.

- `input.miles_traveled` provides finer-grained adjustments. Its SHAP values are smaller in magnitude but show a clear pattern: high-mileage trips (pink) typically nudge the prediction upward relative to low-mileage trips with similar receipts and duration, reflecting an additional mileage-based component in the underlying logic.

To probe the structure of the learned reimbursement rule in more detail, we generated a SHAP dependence plot for `input.total_receipts_amount`, with color indicating `input.miles_traveled` (Figure 12). At low receipt totals, SHAP values are strongly negative: the model predicts much less than the dataset-wide baseline reimbursement, reflecting the simple fact that trips with very little documented spending tend to receive low payouts. Between roughly \$600 and \$1,000, the curve rises steeply: in this region, additional receipts have a large marginal effect on the predicted reimbursement, suggesting that the legacy engine is filling in per-diem or mileage components on top of a growing receipts base. Beyond approximately \$1,500, the SHAP values level off and even show slight decline, indicating diminishing marginal impact of receipts at the high end. This flattening is consistent with soft caps or saturation effects in the reimbursement rules (for example, daily or overall trip limits).

The color gradient in the dependence plot also reveals how mileage modulates the receipts effect. At a fixed receipts level, points with higher `input.miles_traveled` values (pink) tend to sit slightly above those with lower mileage (blue), indicating that the model increases reimbursement somewhat for trips that cover more distance even after accounting for receipts and duration.

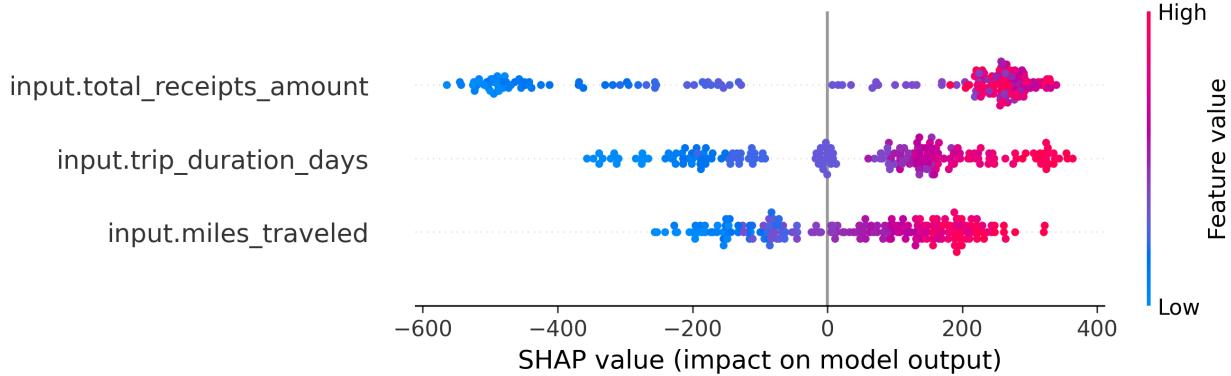


Figure 11: SHAP summary plot for the XGBoost model, using the three original inputs. Each point is a trip; the horizontal axis shows the SHAP value (feature contribution to the reimbursement prediction), and color encodes the raw feature value (blue = low, pink = high). The features are ordered by their overall impact on the model output.

Taken together, the SHAP analysis shows that the XGBoost surrogate has learned a behavior that aligns well with business expectations: predicted reimbursement is driven primarily by total receipts, modulated by trip duration and distance traveled. The non-linear dependence on receipts, particularly the saturation at high spending, suggests that the underlying legacy engine enforces implicit caps or diminishing returns on very expensive trips—exactly the kind of rule structure we sought to reverse-engineer with this project.

6 Discussion and Conclusions

Using only three descriptive inputs drawn from historical reimbursement cases, we built a modern supervised-learning surrogate for a legacy travel reimbursement engine. After careful feature engineering, scaling, and model comparison, we found that:

- Simple linear models explain roughly 78% of the variance in reimbursements but leave substantial residual error ($MAE \approx \$166$).
- Polynomial models and single decision trees substantially reduce error, highlighting the importance of nonlinear relationships, but still underperform tree ensembles.
- Random forests and gradient boosting methods more than halve the MAE relative to linear baselines, with `gbr_shallow` achieving MAE around \$68 and $R^2 > 0.94$.
- A tuned XGBoost model provides very similar accuracy, confirming that boosted trees are well suited to this problem.
- Interpretability tools—feature importance, residual analysis, and SHAP values—paint a consistent picture: reimbursement amounts are driven primarily by total receipts, with trip duration and miles traveled providing secondary refinements.

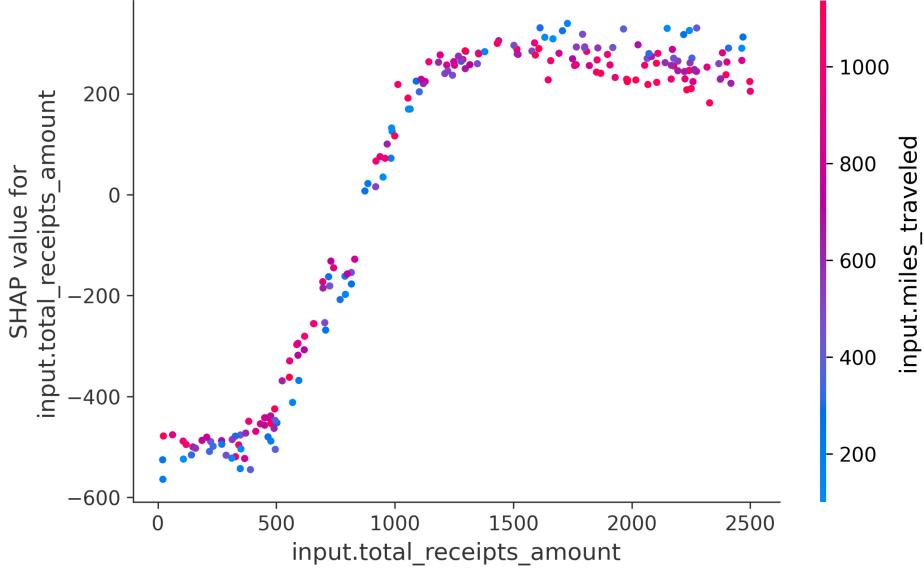


Figure 12: SHAP dependence plot for `input.total_receipts_amount` in the XGBoost model, with points colored by `input.miles_traveled`. The vertical axis shows the contribution of receipts to the predicted reimbursement; the curve exhibits a steep rise at moderate receipts followed by a saturation region at higher spending levels.

Overall, the final Gradient Boosting Regressor offers a strong and interpretable approximation to the legacy reimbursement engine. It can be deployed as a lightweight script that accepts three numeric inputs and returns a single reimbursement amount while preserving transparency about how predictions are made and how accurate they are likely to be.

6.1 Limitations and future work

This project deliberately focused on reproducing the existing reimbursement engine as a surrogate model rather than designing a new policy from first principles. As a result, the learned model inherits any quirks or biases of the legacy system, including potential edge cases that may not align with current business goals. In addition, all models are constrained to use only three numeric inputs, so any rules based on other information (such as traveler role, destination region, or exceptional approvals) cannot be captured. Our evaluation is based on a single 80/20 train–test split rather than full cross-validation, so performance metrics could vary slightly under a different split of the data.

Future work could incorporate richer features (for example, categorical encodings of route type or traveler segment), apply cross-validation or time-aware validation if new data arrive over time, and explore model compression or rule extraction to translate the boosted-tree logic into a small set of human-readable reimbursement rules that can be compared directly with the written travel policy.