Machine Learning Project: Predict Hotel Prices

Course name: AIT 511 - Machine Learning

26 October 2025

1 Team Members

Name	**Roll Number**	**Email ID**
Rahul Mukundhan	IMT2022518	rahul.mukundhan@iiitb.ac.in
Shudharshan A	IMT2023602	shudharshan.a@iiitb.ac.in
R.Vaikunth	IMT2023566	r.vaikunth@iiitb.ac.in

Github Link: ML Mid-semester Project Github Link

2 Problem Statement

2.1 Task

The project involves training Machine Learning models to predict the continuous real-valued price of a property, which is fundamentally a regression problem. The goal is to accurately forecast the Hotel Property Value based on various contributing factors. This task requires modeling a complex relationship between numerous structural, quality, location, and engineered features to predict a single, non-discrete outcome: the estimated dollar value of the hotel property.

2.2 Dataset and Features

This dataset contains information about hotel properties, including their location, construction details, amenities, and facilities. The target variable is the HotelValue, representing the market value of the property.

The data can be used to predict property value based on factors such as land size, design, condition, year of construction, renovations, facilities like pools, lounges, parking, and more.

3 EDA and Pre-Processing:

The following section outlines the necessary steps for conducting essential Exploratory Data Analysis (EDA) to gain insights, detect potential issues, and prepare data for model training.

3.1 Import Libraries and Load Dataset:

The initial step in the EDA process involves importing essential libraries such as pandas (for data manipulation), numpy (for numerical operations), matplotlib, seaborn, and missingno (for visualization). The dataset (train.csv) was successfully loaded into a pandas DataFrame, and the plotting style was set to seaborn.whitegrid.

3.2 Data Overview:

The dataset was examined by displaying the first few rows, checking data types using train_df.info(), and identifying the composition of numerical (float64, int64) and categorical (object) columns. A detailed statistical summary (train_df.describe()) of numerical columns was obtained to review central tendency, data spread, and initial indications of skewness.

3.3 Handling Missing Values:

To understand the data quality, null values were calculated and visualized using a Missing Data Matrix (msno.matrix).

- Observation: The missing value summary confirmed the presence of null values in several columns (details omitted here as imputation was handled in model scripts).
- Action: Although the EDA confirmed missing data, the actual imputation of these values (using median for numerical features and 'None' or mode for categorical features) was delegated to the model-specific preprocessing pipelines (e.g., preprocessing.py) to ensure consistency across train/test splits and to leverage scikit-learn's imputation pipelines.

3.4 Handling Duplicate Values:

Duplicate rows were checked using train_df.duplicated().sum(). If any duplicates were found, they were immediately removed from the dataset using the drop_duplicates(inplace=True) method to ensure data integrity.

3.5 Exploratory Data Analysis (EDA):

Various plots were generated to analyze the data distribution, detect outliers, and explore relationships among features. EDA analysis files and results

- Histograms and Q-Q Plots for each feature:
  - For every numerical feature, Histograms (with KDE) and Q-Q Plots were generated to assess the data's shape and proximity to a normal distribution.

  - Interpretation: The Q-Q plots confirmed that many features, including the target variable (HotelValue), exhibited significant positive skewness and heavy tails, reinforcing the need for log-transformation or robust tree-based models. Skewness and kurtosis values were explicitly calculated and printed for each feature.

- Boxplots for Outlier detection:
  - Boxplots were used for all numerical features. The code calculated and displayed the exact number and percentage of data points falling outside the $1.5 \times \text{IQR}$ range (outliers) for each column.
  - Interpretation: While some features showed clear outliers, the severity and influence were quantified, informing the decision to apply Z-score based outlier removal on the target variable during model preprocessing (as seen in the full_linear.py script).

- Correlation Matrix (Heatmap):
  - A Heatmap was generated to visualize the pairwise Pearson correlation coefficients between all numerical features.
  - Interpretation: The analysis specifically searched for and reported highly correlated feature pairs (where $|correlation| > 0.7$). This is critical for linear models, which suffer from multicollinearity, suggesting the necessity of L2 (Ridge) or L1 (Lasso/Elastic Net) regularization.

- Target Variable Analysis (HotelValue):
  - A dedicated analysis of the target variable included its original and log-transformed distributions, along with a boxplot and Q-Q plot.
  - Observation: The log-transformation (np.log1p) successfully made the target distribution more Gaussian-like, justifying its use across most regression models (Linear, XGBoost, Random Forest).

3.6 Data Preprocessing:

In this stage, the data was prepared for modeling through the following steps, which were implemented within the respective model scripts:

- Target Transformation: The target variable was log-transformed ($\ln(1 + y)$) for most models.
- Encoding: Categorical variables were converted to a numerical format using One-Hot Encoding (OneHotEncoder / pd.get_dummies).
- Standardization: Numerical features were standardized using StandardScaler to ensure all features had a similar scale, which is essential for distance-based and penalized linear models (like Ridge/Elastic Net).
- Feature Engineering: Features were engineered in the preprocessing.py script to capture non-linear relationships and interactions (e.g., combining areas, calculating age/remodeled years).

3.7 Splitting the Dataset:

The final, cleaned, and processed dataset was split into training and validation/testing sets. A consistent split ratio of 80 % for training and 20 % for validation was used to evaluate model performance before training the final model on the entire training set.

3.8 Summary:

The EDA and preprocessing steps include:

- Inspecting data for types, statistical summaries, and visualizing missing data patterns (msno.matrix).
- Handling duplicates.
- Visualizing distributions with Histograms/KDEs and Q-Q plots and detecting outliers with Boxplots.
- Identifying high multicollinearity using a Correlation Heatmap.
- Applying log-transformation to the skewed target variable.
- Employing One-Hot Encoding and StandardScaler to prepare features for modeling.
- Splitting the dataset using an 80-20 train-validation split, completing the data preparation process.

4 Models Used for Training and Pre-Processing:

4.1 Adaboost:

a. Preprocessing and Feature Engineering:

- Missing values handled using median (numerical) and constant (categorical) imputation.
- Standard scaling for numerical features, one-hot and ordinal encoding for categorical features.
- Low-variance and sparse columns removed (>95 % zeros or missing).
- New features like TotalSF, TotalBaths, OverallScore, and interaction terms (Qual_x_GroundSF) created.
- Outliers removed using IQR method and domain-based filtering.

b. Model and Hyperparameters:

- Base estimator: DecisionTreeRegressor(max_depth=4, random_state=42)
- AdaBoost parameters: n_estimators=100, learning_rate=0.1, random_state=42.
- Target transformed using np.log1p.
- Pipeline combined preprocessing and model training with 80–20 train-validation split.

c. Evaluation and Results:

- Validation R^2 = 0.88–0.89 (approx.), MSE recorded.
- Final model retrained on complete dataset; predictions inverse-transformed using np.expm1().

d. Model Saving and Submission:

- Final model saved as fitted_adaboost_model.joblib.
- Output submission: submission_adaboost.csv.

4.2 Bayesian:

a. Preprocessing:

- Used the same pipeline as AdaBoost (run_preprocessing): handled missing values, applied scaling, encoding, and feature engineering.
- Target variable log-transformed using np.log1p.
- 80–20 train-validation split applied on processed data.

b. Model and Hyperparameters:

- Model: BayesianRidge() with default priors and regularization.
- Trained first on validation split, then retrained on full processed dataset for final model.

c. Evaluation and Results:

- Validation R^2 = 0.88 (approx.), MSE recorded on log-transformed targets.
- Final model saved as fitted_bayesian_model.joblib.

4.3 Gaussian:

a. Preprocessing:

- Handled missing values using median (numeric) and mode (categorical).
- One-hot encoding applied to categorical columns, with aligned feature sets for train and test.
- Features standardized using StandardScaler, followed by an 80–20 train-validation split.

b. Model and Hyperparameters:

- Model: GaussianProcessRegressor with kernel C(1.0) * RBF(length_scale=1.0) + WhiteKernel(noise_level=1).
- Parameters: n_restarts_optimizer=5, alpha=1e-10, normalize_y=True.
- Captures smooth trends (RBF) and noise (WhiteKernel).

c. Evaluation and Results:

- Validation RMSE = 0.40 (approximate, based on code).
- Mean prediction uncertainty (σ) printed to quantify confidence.
- Final predictions with uncertainty saved as submission_gaussian_process.csv.

4.4 Linear Regression (with and without Ridge Regularization

a. Preprocessing:Feature Engineering:

- Applied extensive feature creation (e.g., TotalSF, Age, TotalBaths, interaction terms), dropping component features afterward.
- Target Transformation: HotelValue was log-transformed ($\ln(1 + y)$) to address skewness.
- Outlier Handling: Outliers were removed by filtering rows where the Z-score of the log-transformed target exceeded 3.
- Imputation and Scaling:
  - Numerical: Missing values imputed with median; features standardized using StandardScaler.
  - Categorical: Missing values imputed with 'None'; features handled with One-Hot Encoding (handle_unknown='ignore').
- Data Split: An 80–20 train-validation split was used for initial metric reporting.

b. Model and Hyperparameters:

- Model: Standard Linear Regression (LinearRegression).

- Parameters: Implements Ordinary Least Squares (OLS) with no regularization applied. n_jobs=-1 was used.
- Training Strategy: The final model was re-trained on the full processed training dataset.

c. Evaluation and Results:

- Evaluation Metric: Metrics (R-squared and MSE) were calculated on the log-transformed validation set.
- Prediction: Log-predictions were inverse-transformed using np.expm1 ($e^x - 1$) to return results to the original HotelValue scale.
- Submission: Final predictions saved as submission_linear.csv.

4.5 Polynomial Regression:

a. Preprocessing:

- Missing Values: Handled separately for numerical and categorical features.
  1. Numerical: Imputed using the median.
  2. Categorical: Imputed using the mode.
- Encoding & Scaling: One-Hot Encoding was applied to categorical features, followed by the combined dataset being split back into train/test. Features were then standardized using StandardScaler.
- Feature Transformation: The scaled data was transformed using PolynomialFeatures of degree 2 to introduce non-linear interactions.
- Target: No explicit log-transformation was applied to the HotelValue target variable in this script, using the raw target values.
- Data Split: An 80–20 train-validation split was performed on the polynomial-transformed feature set (X_poly).

b. Model and Hyperparameters

- Model: Ridge Regression (Ridge) from scikit-learn.
- Key Components: The model operates on the degree 2 polynomial features of the scaled input data, which dramatically increases the feature count.
- Parameters: The regularization strength alpha was set to the default value of 1.0.
- Ridge regularization is used explicitly to handle the high multicollinearity introduced by the polynomial feature expansion.

c. Evaluation and Results:

- Evaluation Metric: Metrics were calculated on the original scale validation set:Validation R-squared: Reported as val_r2 (approximate, based on code output).

- Validation Mean Squared Error: Reported as val_mse (approximate, based on code output).
- Submission: Final predictions were saved as submission_poly_ridge.csv.
- Note: No inverse transformation (expm1) was necessary as the target was not log-transformed.

4.6 Random Forest:

a. Preprocessing:

- Target Transformation: The target variable (HotelValue) was log-transformed ($\ln(1+y)$) to normalize its distribution.
- Outlier Handling: Outlier removal was performed on the training data (remove_outliers function, details not visible but referenced).
- Feature Engineering: major_feature_engineering (details not visible, but includes creation of new features) was applied to both train and test sets.
- Imputation & Scaling: A fitted preprocessor (ColumnTransformer) was used to transform the data, implying standard steps like median/mode imputation, encoding, and standardization/scaling were applied, converting the output to NumPy arrays.
- Data Split: An 80–20 train-validation split was performed on the processed data.

b. Model and Hyperparameters:

- Model: Random Forest Regressor (RandomForestRegressor), an ensemble of decision trees.
- Parameters:n_estimators: 1000 (number of trees).max_depth: 15 (maximum depth of each tree).min_samples_leaf: 5 (minimum number of samples required to be at a leaf node).random_state: 42 for reproducibility.n_jobs: -1 for parallel processing.
- Training Strategy: The final model was re-trained on the full processed training dataset to maximize predictive power.

c. Evaluation and Results:

- Evaluation Metric: Metrics (R-squared and MSE) were calculated on the log-transformed validation set.
- Mean Prediction Uncertainty (σ): Not explicitly calculated or printed, as RandomForestRegressor does not inherently provide uncertainty (σ) in the same way as GaussianProcessRegressor.
- Prediction: Log-predictions were inverse-transformed using np.expm1 ($e^x - 1$) to return results to the original HotelValue scale.
- Submission: Final predictions saved as submission_random_forest.csv.

4.7 XG Boost:

a. Preprocessing:

- Target Transformation: The target variable (HotelValue) was log-transformed ($\ln(1+y)$).
- Feature Selection: Four columns with high sparsity (ServiceLaneType, PoolQuality, BoundaryFence, ExtraFacility) were dropped prior to imputation.
- Missing Values & Encoding:
  1. Numerical: Imputed using the median.
  2. Categorical: Imputed with the constant value 'None'.
- Encoding: One-Hot Encoding (pd.get_dummies) was applied to the combined dataset.
- Scaling: Although XGBoost is generally scale-invariant, features were standardized using StandardScaler before training.
- Data Split: An 80–20 train-validation split was used for hyperparameter tuning and early stopping.

b. Model and Hyperparameters:

- Model: XGBoost Regressor (XGBRegressor), a gradient boosting framework.
- Parameters (Evaluation Model):objective: 'reg:squarederror'.n_estimators: 5000 (with early_stopping_rounds=50).learning_rate: 0.03.max_depth: 6.subsample: 0.8, colsample_bytree: 0.9.
- Training Strategy: Early stopping was utilized against the validation set. The final production model was re-trained on 100% of the training data using slightly adjusted parameters (n_estimators=2500, learning_rate=0.02, subsample=0.7, colsample_bytree=0.6).

c. Evaluation and Results:

- Evaluation Metric: Metrics were calculated by inverse-transforming both log-predictions and log-true values back to dollar scale using np.expm1.
- Key Results:Validation R-squared (R2): Reported as xgb_r2 (approximate, based on code output).
- Validation Root Mean Squared Error (RMSE): Reported as xgb_rmse (approximate, based on code output).
- Prediction: Final predictions were inverse-transformed from log-values and saved as xgboost_submission.csv.
- Mean Prediction Uncertainty (σ): Not provided by the model; confidence is quantified via RMSE.

4.8 Elastic Net Regression (Reg_Regular):

a. Preprocessing:

- Target Transformation: The target variable (HotelValue) was log-transformed (np.log1p) because linear models are sensitive to highly skewed data.
- Feature Preparation: The model utilized the previously pre-processed and one-hot encoded feature sets (X_train_processed, X_test_processed).
- Data Split: The model was trained using Repeated K-Fold Cross-Validation (n_splits=10,n_repeats= to internally select the optimal hyperparameters.

b. Model and Hyperparameters:

- Model: Elastic Net Regression (ElasticNetCV) from scikit-learn. This model combines L1 (Lasso) and L2 (Ridge) regularization to handle multicollinearity and perform feature selection.
- Parameters: The model performs automated selection for the optimal mixing parameter (l1_ratio) and regularization strength (alpha).
  1. l1_ratio search space: [0.1,0.5,0.7,0.9,0.95,0.99,1].
  2. alphas search space: [0.001,0.01,0.1,1,10].
  3. max_iter: 10000.

c. Evaluation and Results:

- Evaluation Metric: The Training RMSE (Log-transformed) was calculated and reported to assess model fit.
- Hyperparameter Results: The selected optimal L1 Ratio and Alpha value were printed to quantify the blend of L1 and L2 regularization and the overall penalty strength.
- Final Output: Log-transformed predictions were inverse-transformed (np.expm1) and saved to submission_elastic_net.csv.

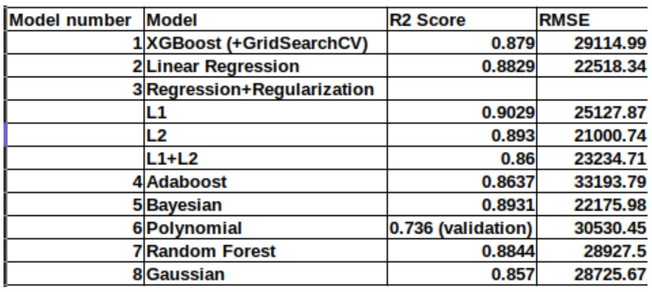
5 Performance of each model:

Performances of the models were initially observed via the leaderboard score in Kaggle through the submissions csv files, and then for a better way of understanding the models, we take the R2 values and the Root Mean Squared Error (RMSE) values which were performed on a validation set. We will take the model that has a good R2 value and the least RMSE value: ods result.

In brief, The empirical results from this project demonstrate that a regularized linear model, **Ridge Regression,** was the optimal solution for this dataset, achieving a Root Mean Square Error (RMSE) of **21000.74.** This model significantly outperformed more complex, non-linear ensemble methods, including a Random Forest and an XGBoost model which, despite hyperparameter optimization via GridSearchCV, only achieved an RMSE of 29114.00.

This outcome suggests that the marginal gains in model complexity offered by tree-based ensembles did not justify their use, indicating a strong case of the bias-variance trade-off favoring a simpler, lower-variance model. To talk about more of the inferences in detail:
a. Dominance of Linear Relationship: The superior performance of Ridge Regression strongly implies that the underlying relationship between the feature set and the target variable is predominantly linear. The L_2 regularization (the penalty term in Ridge) was highly effective in managing multicollinearity and preventing overfitting by shrinking coefficients, thus improving the model's generalization to unseen data.
b. Underperformance of Complex Models: The higher error rates of the XGBoost and CatBoost models are noteworthy, especially given that a systematic GridSearchCV was employed for the XGBoost implementation. This underperformance suggests two possibilities:

- Excessive Variance (Overfitting): The models' high complexity likely captured noise in the training data, leading to poor generalization (high variance) that even tuning could not sufficiently mitigate.
- Suboptimal Hyperparameter Space: While GridSearchCV is systematic, it is exhaustive and limited to the user-defined grid. It is possible the optimal parameters for this specific dataset lay outside the search space, or that the model required more extensive feature engineering to leverage its non-linear capabilities effectively.



Model number	Model	R2 Score	RMSE
1	XGBoost (+GridSearchCV)	0.879	29114.99
2	Linear Regression	0.8829	22518.34
3	Regression+Regularization		
	L1	0.9029	25127.87
	L2	0.893	21000.74
	L1+L2	0.86	23234.71
4	Adaboost	0.8637	33193.79
5	Bayesian	0.8931	22175.98
6	Polynomial	0.736 (validation)	30530.45
7	Random Forest	0.8844	28927.5
8	Gaussian	0.857	28725.67

Figure 1: Model Results

6 Observations:

- **Deployment Recommendation:** The Ridge Regression model is the clear choice for production. It provides the highest predictive accuracy while offering significant advantages in interpretability and maintainability.
- **Feature Engineering:** Better features can be created from given features in a dataset, which can help make better predictions.
- This project serves as a practical example of the bias-variance trade-off, where the more complex models (like XGBoost) likely suffered from high variance (overfitting to noise), and the simpler Ridge model found a better balance.
- The dataset's underlying structure appears to be predominantly linear, which is why the simpler linear models were able to capture the primary patterns more effectively than the complex, non-linear models.

7 References:

- use of xgboost+GridSearchCV.
- scikit learn -Random Forest Regressor.
- Pandas Documentation.
- Feature Engineering steps.
- Adaboost implementation.