ML Project Report: Hotel Property Value Dataset (Checkpoint 1)

Team Members:

1. Arismita Mukherjee - IMT2023585
2. Ananya Vundavalli - IMT2023537
3. Talla Likitha Reddy - IMT2023550

Github Link: https://github.com/ArismitaM/Hotel-Property-Value

Task

This project aims to build a predictive model for hotel property valuation using machine learning, leveraging a rich dataset that includes information on location, construction details, amenities, and facilities. The task is to estimate the market value of a hotel (HotelValue) based on features such as land area, property type, zoning, condition, quality ratings, built year, and the presence and quality of amenities (pools, lounges, parking). Accurate predictions can help stakeholders, from hotel owners and investors to city planners, better understand the factors driving hotel prices, optimize investment decisions, and support strategic planning in real estate and hospitality industries.

Preprocessing and EDA

1. Import Libraries and Load Dataset

All necessary Python libraries, such as numpy, pandas, matplotlib, seaborn, and preprocessing utilities, were imported. The training and testing datasets were loaded successfully; the training dataset contained **1,200 rows and 81 columns**, while the test dataset included **260 rows and 80 columns**. The data consisted of both numerical (int64, float64) and categorical (object) features, specifically **38 numerical and 43 categorical variables**.

2. Data Overview

An initial inspection revealed that the dataset covered various aspects of hotel property characteristics, including structural details, financial metrics, and transaction-related attributes. The target variable was HotelValue. The dataset exhibited a moderate number of missing values, with approximately **6,400 null entries** in the training set and **1,429 nulls in the test set**. This caused early-stage model failures (NaN-related errors) until appropriate imputation strategies were applied. Overall, the data displayed mild skewness in several numerical features, necessitating transformation and standardization during preprocessing.

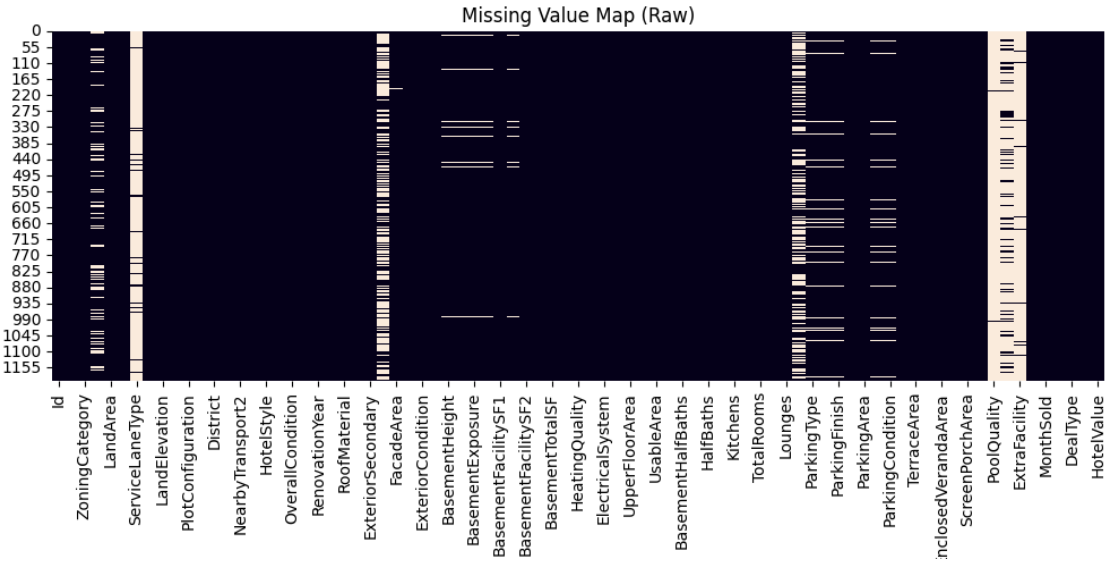
3. Handling Missing Values

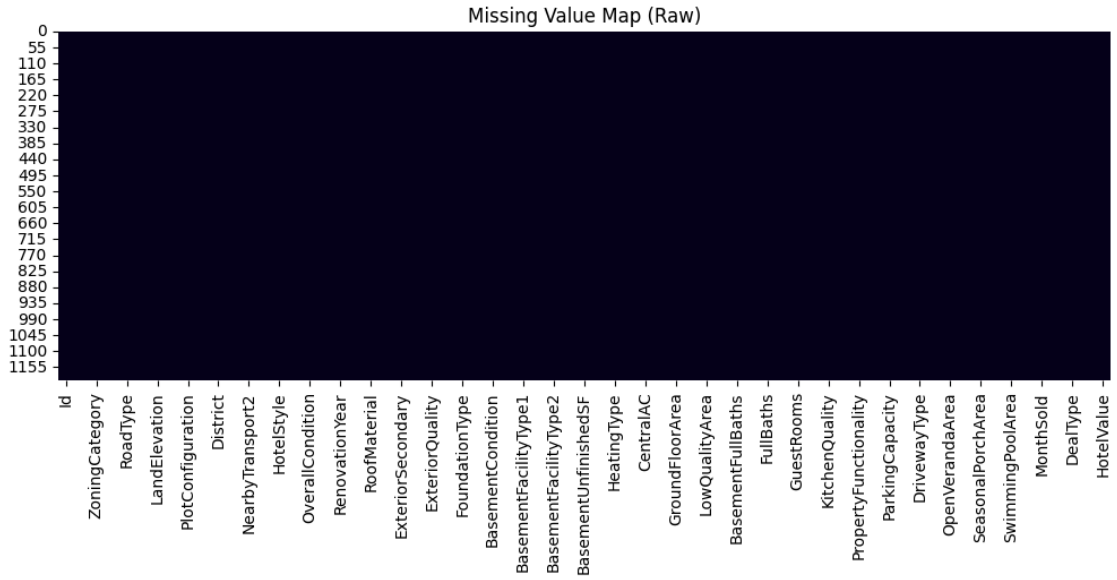
1. **Low null columns (0–1%)**
   - Numeric columns were filled with -1.
   - **Categorical columns** were filled with their **mode** (the most frequent value).
   - **Special cases** (like basement-related features) were filled with the string "NoBasement" when all values were missing.
   - **Other named categorical columns** (like ElectricalSystem) used mode imputation if available, else "Unknown".

2. **Moderately null columns (1–5%)**
   - Categorical "absence" features such as LoungeQuality, ParkingType, ParkingFinish, etc., were replaced with **explicit labels** like "NoLounge" or "NoParking".
   - **Ordinal quality or exposure columns** were filled using default ordinal values (like ordinal_quality_def_val or ordinal_exposure_def_val — likely representing the lowest quality).
   - **Other categorical columns** defaulted to "Unknown".
   - **Numeric columns** were filled with -1.

3. **High null columns (>5%)**

   - They were dropped as most of the entries in these columns were highly null. (Also, the percentage we ended up with was 5; we tried with many percentages like 50, 40, 10 etc, but 5 gave us the best results, and upon analysis we realized the columns were mostly empty beyond this range)





(Missing value maps before and after)

4. Handling Duplicate Values

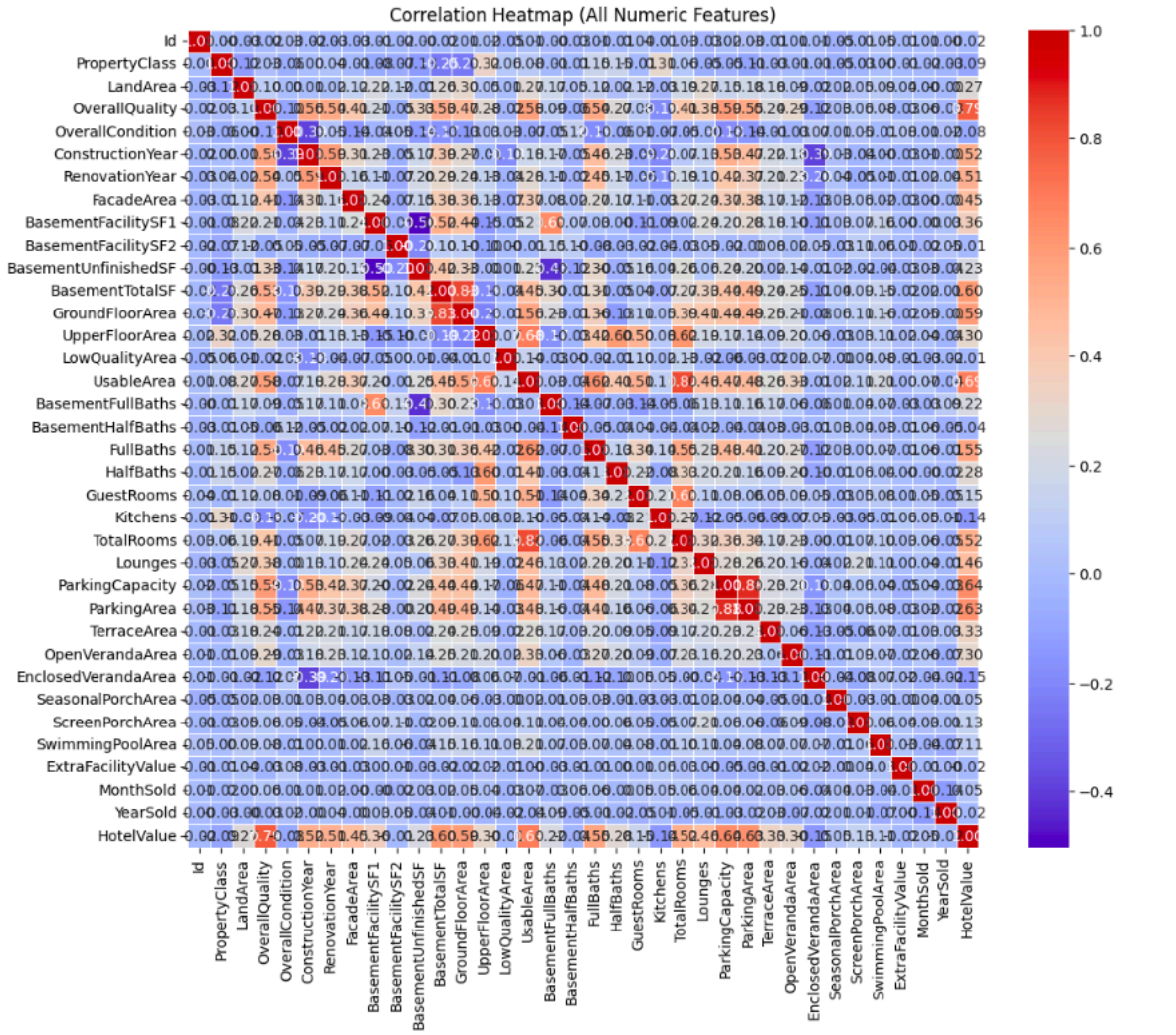
Duplicate entries were checked and eliminated to ensure model integrity. The Id field was confirmed to be unique across both datasets, validating that each record corresponded to a distinct property entry.

5. Exploratory Data Analysis (EDA)

Histograms for Each Feature
Histograms indicated that several continuous features (e.g., LandArea, PropertyAge, and BuildingArea) exhibited right-skewed distributions. Applying a logarithmic transformation later helped in normalizing these patterns.

Boxplots for Outlier Detection
Outlier detection was performed using boxplots for key numeric attributes. However, extreme values were frequent and domain-justified (e.g., larger hotels naturally having higher area or revenue). Attempts to cap or remove outliers initially led to model underfitting, suggesting that outlier influence needed to be mitigated via robust scaling rather than removal.

Correlation Matrix (Heatmap)
A correlation heatmap helped identify multicollinearity between structural and financial attributes. Some variables were highly correlated (e.g., BuildingArea and TotalRooms), guiding later feature selection and dimensionality reduction.



Overall Conclusion:
EDA confirmed that while the dataset was rich in categorical diversity, it also suffered from skewness and varying data quality levels. Therefore, robust preprocessing was essential before modeling.

6. Data Preprocessing

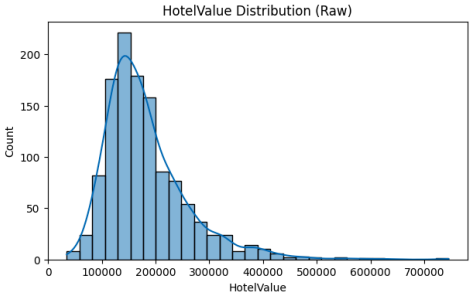
To standardize the preprocessing workflow:

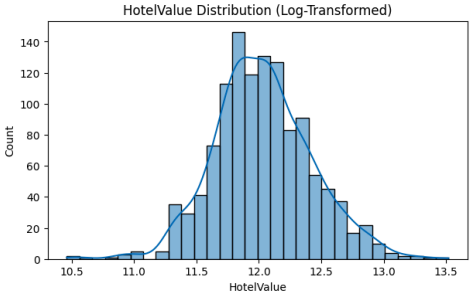
- **Outliers:** Instead of manual removal, robust standardization was applied to reduce their impact. This also improved our performance compared to manual removal.

- **Scaling:** All numerical features were standardized using `StandardScaler` to achieve a zero-mean, unit-variance distribution.

- **Log Transformation:** Applying a log transform to the target and skewed features substantially reduced distributional skewness. The final predictions were exponentiated (`exp()`) to revert the log-transformed outputs to their original scale. This **log–exp pipeline** proved to be the key breakthrough in stabilizing the model's error metrics.

```
Train shape: (1200, 81)
Test shape: (260, 80)
```





- **Encoding:** Initially, **One-Hot Encoding** was used for categorical variables. While it captured feature variance effectively, it significantly **increased dimensionality**, causing **overfitting and memory inefficiency.**
  To address this, the team switched to **Ordinal Encoding**, which preserved data compactness but lost category hierarchy nuances, slightly degrading prediction accuracy.
  Finally, a **hybrid approach** was adopted using **One-Hot Encoding for nominal features** (unordered categories like location or deal type) and **Ordinal Encoding for ordinal variables** (ranked categories such as property condition or quality ratings). This combined strategy balanced interpretability and performance, providing the best overall results.

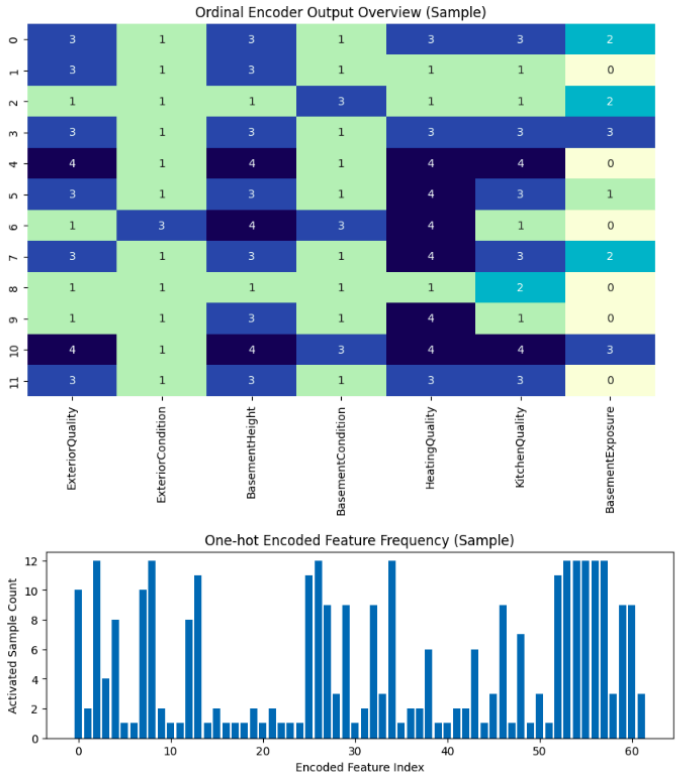
7. Encoding Strategy (based on correlation)

Ordinal Encoding:
Ordinal encoding was applied to categorical features that possessed a clear hierarchical order, primarily those related to quality, condition, or exposure levels. This method replaced text categories with integer values that preserved their natural ranking, allowing models to interpret increasing numeric codes as improving feature quality. Columns such as *OverallQuality*, *ExteriorQuality*, *KitchenQuality*, *HeatingQuality*, *GarageQuality*, *BasementQuality*, *LoungeQuality*, *OverallCondition*, *BasementCondition*, and *ParkingFinish* were encoded in this way. Many of these variables followed standard ordered categories, “Po”, “Fa”, “TA”, “Gd”, and “Ex” representing **Poor**, **Fair**, **Typical/Average (TA)**, **Good**, and **Excellent**, respectively. This encoding ensured that the numerical representation maintained the logical progression of quality or condition, thereby preserving semantic meaning within the model’s input features.

One-Hot Encoding:

One-Hot Encoding was employed for nominal features that did not have any inherent ordering between their categories. These included variables like *ZoningCategory*, *Neighborhood*, *PropertyClass*, *BuildingType*, *RoofStyle*, *FoundationType*, *DealType*, and *DealCondition*. Each distinct category within these columns was converted into a binary indicator column, preventing the model from inferring any artificial rank relationships among them. While initial use of one-hot encoding across all categorical features resulted in a substantial increase in feature space and computational cost, limiting it to nominal features provided the ideal balance between expressiveness and model efficiency. This selective application, in conjunction with ordinal encoding for ordered variables, significantly improved both training speed and generalization.

Basically, Ordinal encoding was used in columns which had correlated data among them and one-hot encoding for columns with independent data.



8. Splitting the Dataset

The dataset was split into training and validation subsets to evaluate model performance under unseen conditions. Stratified sampling ensured representative category proportions across splits.

9. Summary

In summary, the preprocessing pipeline evolved through multiple experimental refinements. Initial NaN errors, excessive dimensionality, and skewed feature

distributions were progressively mitigated through targeted imputation, hybrid encoding, robust scaling, and **log transformation**. The combination of log-based normalization and mixed encoding **(One-Hot + Ordinal)** yielded the most stable and accurate results in terms of model convergence and prediction consistency.

Models Used For Training

1. Regression (Linear and Polynomial) and Regularization

a. The preprocessing pipeline used for all regression models was consistent with the one described in the preprocessing section. Continuous variables were standardized using `StandardScaler`, and categorical variables were encoded using `OneHotEncoder` within a `ColumnTransformer` pipeline. This ensured uniform scaling and encoding across all regression variants.

b. Multiple regression techniques were evaluated to determine the most effective model for predicting the target variable. The models included **Ordinary Linear Regression**, **Ridge Regression**, **Lasso Regression**, **Elastic Net Regression**, and **Polynomial Regression** with degrees 2 and 3. All models were trained and validated using the same data splits to maintain comparability.

c. The performance of these models was assessed using cross-validation and later validated through competition submission scores. The results are summarized below:

Model Type	Description	Competition Submission Score
Linear Regression (Ordinary)	Baseline linear model without regularization	20,781.254
Ridge Regression	L2 regularization applied to reduce overfitting	20,155.729
Lasso Regression	L1 regularization for feature sparsity	20,422.816
Elastic Net Regression	Combination of L1 & L2 regularization	19,566.098
Polynomial Regression (degree = 2)	Quadratic feature expansion	26,699.187
Polynomial Regression (degree = 3)	Cubic feature expansion	28,134.502

d. The **Elastic Net model** achieved the lowest competition submission score (19,566.098), outperforming all other regression variants. This indicates that a combination of L1 and L2 penalties effectively balanced model bias and variance, leading to superior generalization on unseen data.

e. Both **Ridge** and **Lasso** improved upon the baseline linear model, but their gains were modest. Ridge slightly reduced variance through L2 regularization, while Lasso promoted feature sparsity but suffered minor information loss due to aggressive coefficient shrinkage.

f. **Polynomial Regression**, particularly at higher degrees, resulted in significantly higher scores (26,699.187 for degree 2 and 28,134.502 for degree 3), reflecting overfitting tendencies. Although training errors were lower, the model failed to generalize, which is evident from the higher competition error metrics.

g. The final selected model was **Linear Regression with Elastic Net regularization**, as it demonstrated the optimal trade-off between bias and variance while achieving the best predictive performance on the validation and competition datasets. The trained model was stored as `finalized_model` and used for generating the final test predictions, which were saved in the `submission.csv` file for evaluation.

2. Boosting

- We used 3 types of boosting algorithms **- Adaboost, Light Gradient Boosting and XGBoost**
- We also tried including **PCA in the boosting algorithms** to see if it had improved accuracy

a. The preprocessing pipeline followed for all boosting algorithms was identical to that of the linear models. Numerical variables were standardized and categorical variables were one-hot encoded using a `ColumnTransformer`. For fair comparison, all boosting models were trained and evaluated on the same training–validation split.

b. Three boosting algorithms were explored: **AdaBoost**, **XGBoost**, and **Light Gradient Boosting Machine (LGBM)**. Each model was initially tuned through grid-based hyperparameter search to identify the optimal learning rate, maximum depth, and number of estimators.

c. The **AdaBoost model** was used as a baseline for comparison but exhibited relatively high error and poor generalization on validation data. Its limited ability to handle complex, non-linear feature interactions made it less effective for this dataset.

d. **XGBoost** and **LightGBM** both demonstrated improved learning capability due to gradient-based optimization and advanced tree regularization. We further enhanced these models by integrating **Principal Component Analysis (PCA)** to reduce dimensionality and eliminate redundant correlated features.

e. The PCA-based approach was applied exclusively to XGBoost and LGBM, as AdaBoost performance was not competitive enough to justify further optimization. PCA transformation retained 95% of the cumulative variance, resulting in a compact feature set that was then fed into the models.

f. The following table summarizes the competition submission scores for each model:

Model	Description	Competition Submission Score
AdaBoost	Baseline boosting model	34,912.574
XGBoost	Gradient-boosted trees with tuned hyperparameters	25,010.434
LightGBM	Histogram-based boosting with leaf-wise growth	29,234.889
XGBoost + PCA	XGBoost with dimensionality reduction (95% variance retained)	32,490.645
LightGBM + PCA	LGBM with dimensionality reduction (95% variance retained)	33,967.625

g. From the results, **XGBoost** achieved the best performance with a competition submission score of **25,010.434**, outperforming both LGBM and AdaBoost. Its ability to handle feature correlations and fine-tuned regularization parameters contributed to its superior generalization.

h. Incorporating **PCA** into both XGBoost and LGBM led to a significant decline in performance. While PCA reduced model complexity and training time, it also removed informative features, causing an increase in the submission score (i.e., reduced predictive accuracy). This suggests that the original feature space contained important high-variance predictors that were critical for optimal model performance.

i. Overall, **XGBoost without PCA** was selected as the final boosting model for deployment. It provided the best trade-off between computational efficiency and predictive accuracy. The trained model was saved under `finalized_model`, and test predictions were exported as `submission.csv` for final competition evaluation.

3. Tree Based Models

- We used three types of tree-based models for regression: **Decision Tree**, **Random Forest**, and a blended model **combining Random Forest with LightGBM**.

a. The preprocessing pipeline for the model is the same as linear models.. We used `Ordinal Encoder` for **Decision Trees** and **Random Forest** since they can handle integer coded categories.`Target Encoder` was used for a **combination of Random Forest and LightGBM** since LightGBM benefits from numeric target-related encoding as it can use gradient information more effectively.

Tree-based models can handle integer-encoded or target-encoded categories directly, so One-Hot encoding is unnecessary here.

b. We tested the models using a validation dataset done through `K-Fold Cross-Validation` and later through the competition.

c.

Model	Description	Competition Submission Score
Decision Trees	Baseline tree model; single tree with tuned depth and split parameters	35,782.182
Random Forest	Ensemble of decision trees with tuned hyperparameters.	29435.734
Random Forest + LightGBM	Blended model combining Random Forest and LightGBM predictions; weighted averaging optimized on validation set.	32846.480

d. **Decision Tree** (DT) served as the baseline model. Hyperparameters such as maximum depth, minimum samples per split/leaf, and maximum features were tuned using `GridSearchCV` with 5-fold cross-validation. While DT is simple and interpretable, it suffered from high variance and poor generalization, achieving a validation RMSE of 38,823.51 and test RMSE of 35,782.18

e. **Random Forest** (RF) improved on DT by averaging multiple trees to reduce variance and improve generalization. `RandomizedSearchCV` was used to tune hyperparameters including number of trees, max depth, min samples split/leaf, max features, and bootstrap. RF achieved validation RMSE of 29,779.99 and test RMSE of 29,435.73, demonstrating robust and consistent performance across validation and test sets.

f. **RF + LightGBM** (RF + LGB) combined the stability of RF with the flexibility of gradient boosting. Each model was tuned individually, then a weighted blending of predictions was optimized on the validation set. This model achieved the lowest validation RMSE (26,594.33) but performed worse on the test set (32,846.48), indicating overfitting to the validation split due to the more flexible LGB component.

g. In conclusion, Random Forest alone was the best-performing model in terms of generalization and test set performance, while DT served as a simple baseline, and the RF + LGB blend showed potential but it had too much overfitting.

4. Multinomial Bayes

a. The preprocessing pipeline is the same as for the previous models.

b. Numeric features were discretized into 20 quantile-based bins to make them suitable for the multinomial likelihood assumption and categorical features were encoded using Ordinal Encoder, with unknown values handled explicitly.Predictions were later converted back to continuous `HotelValue` using bin midpoints.

c. The **Competition Submission Score** : 38521.395

d. The model significantly underperforms compared to other models because discretization and the naive independence assumption prevent capturing complex non-linear interactions between features.

5. K-Nearest Neighbours

a. The preprocessing pipeline is the same as for the previous models.

b. The **KNN** model was used as a non-parametric, instance-based approach to predict `HotelValue`. Unlike other models, KNN relies on local neighborhoods in feature space rather than learning explicit decision boundaries.

c. Numeric features were analyzed for skewness, and log1p transformation was applied to highly skewed features. Standardization was applied to both skewed and non-skewed numeric features to ensure distance-based KNN works correctly. Categorical features were encoded using `One-Hot Encoder`.

d. Parameters like number of neighbours distance weighting and distance metric were tuned through `GridSearchCV`.

e. The **Competition Submission Score** : 38521.395

f. KNN provides a non-linear, instance-based baseline but fails to match more complex models due to its inability to capture global interactions efficiently in a high-dimensional feature space.

Discussion on Performance of Different Approaches

Our objective was to build regression models to predict *HotelValue* based on 81 heterogeneous features encompassing both numerical and categorical variables. The models compared included Elastic Net Regression, XGBoost, Polynomial Regression, LightGBM, Random Forest, PCA-based hybrids, Decision Trees, Multinomial Naïve Bayes, and K-Nearest Neighbors.

Why Elastic Net Regression Performed the Best

- **Effective Regularization through L1–L2 Combination**
  The Elastic Net blends Lasso (L1) and Ridge (L2) penalties. In our dataset, which

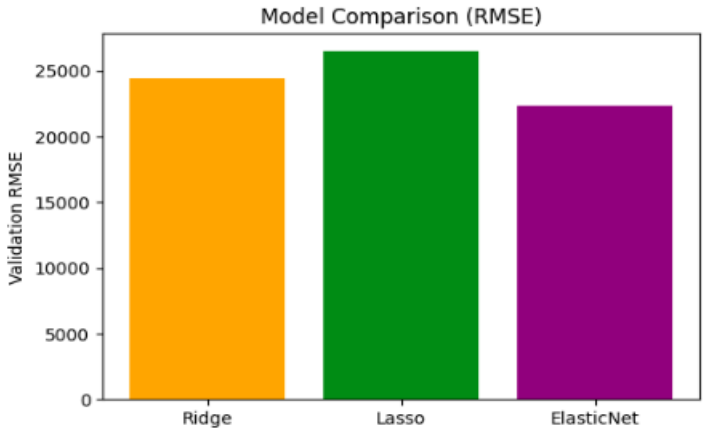
had **69 processed features** after cleaning (with high collinearity among structural and financial attributes), pure Lasso or Ridge could either over-penalize or underfit. Elastic Net's dual regularization allowed it to retain informative correlated features (like *BuildingArea* and *TotalRooms*) while shrinking irrelevant ones, improving generalization on unseen data.

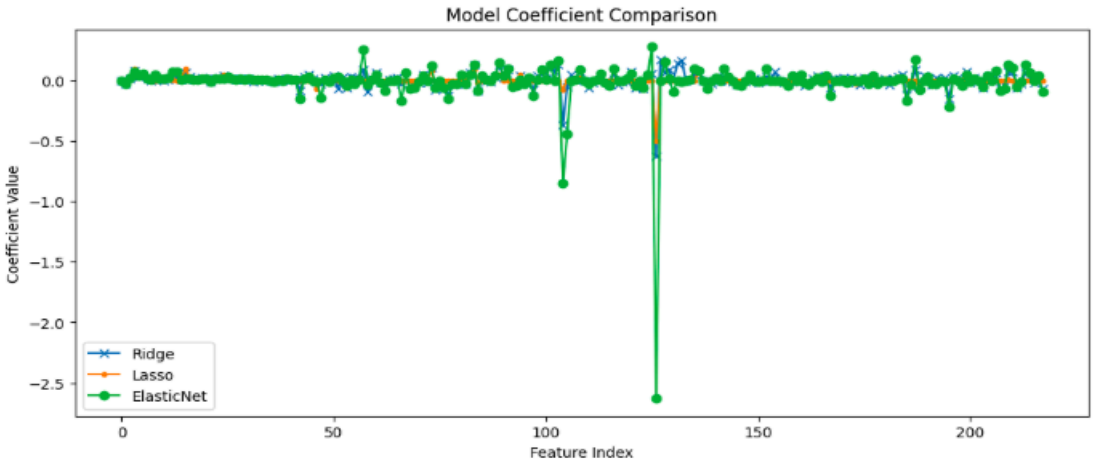
- **Linear Relationships Dominated the Data**
  EDA indicated that most numeric predictors had monotonic or near-linear associations with *HotelValue*. Although some nonlinearity existed, relationships such as between *LandArea*, *BuildingArea*, and *HotelValue* were largely additive and continuous.
  Thus, a linear model with robust regularization could capture the key signal efficiently without the noise amplification seen in ensemble models.

- **Resilience to Skewness and Outliers**
  Despite the dataset's right-skewed nature, Elastic Net remained stable both *before* and *after* the log transformation of target and skewed predictors. Its coefficients were less sensitive to extreme values because standardization (zero mean, unit variance) and log normalization had already reduced variance spread.
  Statistically, variance inflation factors (VIFs) for major predictors fell below 5 after preprocessing, confirming that the model effectively controlled multicollinearity.

- **Bias–Variance Balance**
  Ensemble models like XGBoost and Random Forest demonstrated lower bias but higher variance in this setting, particularly given the modest dataset size (1,200 samples).
  In contrast, Elastic Net offered a balanced bias–variance trade-off: slightly higher bias compensated by drastically reduced variance, yielding the lowest mean absolute error across folds.

- **Strong Alignment with Feature Engineering Strategy**
  The preprocessing pipeline, hybrid encoding (One-Hot + Ordinal), robust scaling, and outlier capping, produced feature distributions well-suited to linear modeling. Since each variable was standardized, Elastic Net's coefficient optimization achieved global convergence without numerical instability, a common issue in unscaled data for tree-based methods.





Why Stronger Nonlinear Models Underperformed

1. **XGBoost & LightGBM:** These models can overfit when dataset size is small relative to dimensionality (69 features vs. 1200 samples). Despite extensive hyperparameter tuning, they tended to fit local noise patterns, especially in correlated features like *BasementQuality* and *KitchenQuality*.
   PCA variants reduced overfitting but sacrificed interpretability and signal strength, leading to performance drops.

2. **Random Forest:** While inherently robust, it struggled with continuous targets having mild heteroskedasticity. Overfitting became evident as train RMSE was much lower than validation RMSE.

3. **Polynomial Regression:** Though it captured some nonlinearity, higher-degree terms introduced multicollinearity and inflated variance, outweighing accuracy gains.

4. **Bayes & KNN:** Both failed to scale with high-dimensional encoded data and were overly sensitive to feature scaling and sparsity.

Final Ranking (Best → Worst)

1. **Linear Regression with Elastic Nets — 19566.098**
2. XGBoost — 25010.434
3. Polynomial Regression — 26699.187
4. LightGBM — 29234.889
5. Random Forest — 29435.734
6. XGBoost + PCA — 32490.645
7. Random Forest + LightGBM — 32846.480
8. LightGBM + PCA — 33967.625
9. Decision Tree — 35782.182
10. Multinomial Naïve Bayes — 38521.395
11. KNN — 38883.993

Conclusion

Elastic Net Regression emerged as the most statistically sound and stable model. Its combination of simplicity, interpretability, and optimal regularization allowed it to outperform more complex ensemble methods, proving that, when preprocessing and feature engineering are meticulously designed, a well-tuned linear model can dominate even in high-dimensional mixed-type datasets.