ML Project Report - Hotel Value Prediction (Checkpoint 1)

Team Members:
1. Subikshaa Sakthivel - IMT2023020
2. Harshita Bansal - IMT2023035
3. Vriddhi Agrawal - IMT2023611

Github Link: https://github.com/Subikshaa22/Hotel_Property_Value_Prediction

Task

The task involves predicting the market value of hotels based on various physical, locational, and structural characteristics of the property. The goal is to train different machine learning models to determine the one that can most accurately estimate a hotel's value with the lowest RMSE (Root Mean Squared Error). The problem is essentially a supervised regression problem, where the model learns from historical data to predict property values for new or unseen hotels.

Dataset Description

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. Such predictions can help stakeholders like investors, real estate developers, or hospitality companies assess property worth, identify undervalued hotels, and make data-driven investment decisions.

EDA and Preprocessing

The Preprocessing and Exploratory Data Analysis (EDA) stage was used to better understand the dataset and get it ready for modeling. This involved examining the structure and distribution of the data, handling missing values and outliers, and scaling numerical features where necessary. EDA helped us find important patterns and relationships in the data, providing insights into which factors most strongly influence hotel property values. These steps ensured that the dataset was clean, consistent, and well-structured for training various models.

1. Importing Libraries and Loading Dataset

First, we imported all the necessary Python libraries required for data handling and analysis. Libraries such as pandas and numpy were used for data manipulation, while matplotlib and seaborn helped in visualizing the data. After importing the libraries, the dataset was loaded into the environment using pandas.

2. Data Overview

We examined the dataset to understand its basic structure. We checked the number of entries, column names, and data types to identify numerical and categorical features, as well as any missing values. We displayed the first few rows of the dataset, giving an initial look at the kind of information available for each hotel property. This helped in getting a basic understanding of the dataset before further analysis. There were a total of **1200 rows** and **81 columns** in the dataset.

3. Handling Duplicate Values

We used the drop_duplicates() function to drop any duplicate values from the dataframe. However, since the shape of the dataframe didn't change after calling the function, we conclude that there were no duplicate values in the dataset.

4. Handling Missing Values

To ensure data quality and consistency, we performed a detailed analysis of missing values across all features in the dataset.

First, we calculated the number and percentage of null values per column and visualized them using a **bar graph**. This helped us identify which columns were most affected by missing data. The **columns with negligible missing values (less than 6%) were handled by dropping the corresponding rows**, as their removal had minimal impact on the overall dataset size and distribution.

On the other hand, **columns with excessive missing values (greater than 45%)** were deemed unreliable and **were dropped entirely from the dataset** to prevent noise and bias in the model.

For the **remaining columns with a moderate percentage of missing values**, we used data **imputation techniques**. 'RoadAccessLength' column was the only such remaining column, and we performed group-wise median imputation based on both PropertyClass and District, ensuring that the imputed values were contextually relevant. Any remaining nulls in that column were replaced with the overall median value.

This hybrid approach helped preserve data integrity while minimizing information loss. After applying these steps, the dataset no longer contained any missing values, making it suitable for further preprocessing and model training.

5. Outlier Detection and Handling

➢ **Identification of Feature Types**

- The dataset was first analyzed to distinguish numerical and categorical columns. This allowed for targeted preprocessing strategies suitable for each data type.

➢ **Numerical Feature Analysis**

- **Visual Inspection:** Box plots were created for each numerical column to identify extreme values and examine the spread of the data.

- **Outlier Detection:** Outliers were defined using the Interquartile Range (IQR) method, where values below (Q1 - 1.5 \times IQR) or above (Q3 + 1.5 \times IQR) were considered extreme.

- **Outlier Handling:** Rather than removing outliers, values beyond the calculated bounds were capped at the lower or upper threshold. This reduced the impact of extreme values while preserving the dataset's overall distribution.

- Columns containing outliers were recorded and inspected to understand which numerical features were most affected.

➢ **Categorical Feature Analysis**

- **Infrequent Categories:** For each categorical column, categories appearing in less than 1% of the dataset were flagged as infrequent.

- **Purpose:** Identifying rare categories helps prevent sparsity and noise in predictive models and guides decisions for encoding, aggregation, or removal of such categories.

6. Scaling and Standardization

To ensure that all numerical features contribute equally to the model, scaling and standardization were applied to the dataset. Since the numerical features varied widely in range, this step helped bring all values to a comparable range and improve model performance.

Two techniques were used for this purpose:

- **Min-Max Scaling:** This method preserves the shape of the original distribution but compresses all numeric values to a range between 0 and 1. The transformation was implemented using MinMaxScaler() from scikit-learn.

- **Standardization (Z-score Normalization):** This technique rescaled the features so that they have a mean of 0 and a standard deviation of 1. The transformation was implemented using StandardScaler() from scikit-learn.

The effects of both transformations were visualized using **KDE plots** and **boxplots**. These plots showed how Min-Max Scaling compressed the values into a 0 - 1 range, while Standardization centered the data around zero.

Overall, this step ensured that all numeric features were on a uniform scale, preventing large-valued attributes from dominating the learning process and improving the stability and accuracy of the model.

7. Saving the Preprocessed Data

At this point, we saved the preprocessed data as a csv file labelled "hotel_data_preprocessed.csv". This file was then used for training different models. Any other preprocessing steps like **feature engineering**, **encoding**, **splitting the data into training and validation sets**, and **pca** were done separately in the respective models' files as not all of these worked well with all the models. Hence, these were implemented separately with only the models that performed well with them.

8. Exploratory Data Analysis (EDA)

1) Target Variable Analysis

The target variable for this dataset is HotelValue. The initial distribution analysis revealed a right-skewed pattern. To normalize this skewness, we applied a **log-transformation**.

After transformation:

- The distribution became more symmetric and approximately normal.

- Skewness was significantly reduced, making the data more suitable for linear and regression-based models.

This step ensures that large-valued properties do not dominate the model learning process.

2) Numerical Feature Selection

All numerical columns in the dataset were identified and filtered to create a working feature set for correlation and multicollinearity analysis. Id and target variables (HotelValue, Log_HotelValue) were excluded from this list. This automated filtering process ensured that only relevant numeric variables were considered for subsequent correlation and visualization tasks.

3) Correlation with Target Variable

A correlation analysis was conducted between each numeric feature and the log-transformed target (Log_HotelValue).

- The top 20 features with the highest absolute correlation values were visualized using a horizontal bar plot.

- Features with strong positive or negative correlations were identified as potentially influential predictors of hotel property value. This ranking provided a data-driven basis for feature selection in subsequent modelling steps.

4) Linearity and Non-Linearity Assessment

To evaluate the type of relationship between predictors and the target, scatter plots with smoothed trendlines (LOWESS) were generated for the top correlated feature.

- If the relationship appeared approximately linear, it indicated suitability for linear regression models.

- If curvature or non-linear patterns were evident, polynomial features or tree-based models would be more appropriate.

The plot we obtained was mostly linear.

5) Multicollinearity Check

The top 10 features most correlated with the target were used to construct a correlation heatmap.

- The heatmap visually revealed inter-feature relationships, helping to detect multicollinearity (strong pairwise correlations > 0.7).

- Features showing redundancy could be removed or combined, or alternatively, Principal Component Analysis (PCA) could be applied before training linear models.

6) Outlier Detection

Boxplots were used to inspect potential outliers in key numeric variables.

- Outliers, if genuine, represent large or premium properties that may need to be handled with robust regression or tree-based algorithms.

- No extreme anomalies were detected that warranted removal, but the distribution tails were noted for model robustness checks.

7) Categorical Feature Exploration

Categorical columns such as were analyzed using count plots and boxplots:

- **Count plots** showed the frequency distribution of each category, highlighting dominant and underrepresented classes.

- **Boxplots** of HotelValue across these categories revealed how property value varied with qualitative attributes. For example, higher property classes and premium hotel styles tended to have greater median hotel values.

8) Output and Summary

Correlation metrics and top feature lists were exported for future reference and reproducibility (feature_target_correlation.csv and top20_features.csv).

Summary of EDA Findings:

- **Target variable:** Right-skewed; corrected with log transformation.

- **Numeric features:** Several variables show moderate correlation with hotel value.

- **Categorical features:** Certain property and district types show higher median values.

- **Multicollinearity:** Present among some top predictors; to be managed during feature selection.

- **Outliers:** Limited and largely valid, representing real high-value properties.

Overall, the EDA provided a strong foundation for feature engineering, scaling, and model development by clarifying which features most influence hotel property valuation and how they interact.

Models Used For Training

1. Decision Tree

- We first encoded all categorical features using **LabelEncoder**, and imputed the missing values in the testing dataset (The training set was already preprocessed using the steps described in the Preprocessing and EDA section).

- We split the training data into an **80-20 train-validation** ratio to evaluate model performance.

- We used **RandomizedSearchCV** for hyperparameter tuning. The **best parameters** obtained were as follows:
  {'min_samples_split': 2, 'min_samples_leaf': 4, 'max_features': 0.7, 'max_depth': 20}

- The **validation RMSE** was 30404.06985307349.

- We re-trained the model on the entire training data (training + validation set) before predicting the values for test.csv.

- The model achieved a **Kaggle score** of 47144.334.

2. Random Forest

- We first encoded all categorical features using **LabelEncoder** since Random Forests cannot handle non-numeric data directly.

- We split the training data into an **80-20 train-validation** ratio to evaluate model performance.

- We used **RandomizedSearchCV** for hyperparameter tuning. The **best parameters** obtained were as follows:
  {'n_estimators': 200, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2', 'max_depth': None}

- The **validation RMSE** was 24787.877527621004.

- We re-trained the model on the entire training data (training + validation set) before predicting the values for test.csv.

- The model achieved a **Kaggle score** of 45739.987.

3. Gradient Boosting

- We first encoded all categorical features using **LabelEncoder**, and imputed the missing values in the testing dataset (The training set was already preprocessed using the steps described in the Preprocessing and EDA section).

- We split the training data into an **80-20 train-validation** ratio to evaluate model performance.

- We used **RandomizedSearchCV** for hyperparameter tuning. The **best parameters** obtained were as follows:
  {'subsample': 0.7, 'n_estimators': 200, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2', 'max_depth': 5, 'learning_rate': 0.05}

- The **validation RMSE** was 20716.458595750348.

- We re-trained the model on the entire training data (training + validation set) before predicting the values for test.csv.

- The model achieved a **Kaggle score** of 40908.516.

4. XGBoost

- We first encoded all categorical features using **LabelEncoder**, and imputed the missing values in the testing dataset (The training set was already preprocessed using the steps described in the Preprocessing and EDA section).

- We split the training data into an **80-20 train-validation** ratio to evaluate model performance.

- We used **RandomizedSearchCV** for hyperparameter tuning. The **best parameters** obtained were as follows:
  {'subsample': 0.7, 'reg_lambda': 1.5, 'reg_alpha': 0.1, 'n_estimators': 700, 'min_child_weight': 3, 'max_depth': 4, 'learning_rate': 0.03, 'colsample_bytree': 0.85}

- The **validation RMSE** was 21668.28047068798.

- We re-trained the model on the entire training data (training + validation set) before predicting the values for test.csv.

- The model achieved a **Kaggle score** of 41597.766.

5. AdaBoost

- We first encoded all categorical features using **LabelEncoder**, and imputed the missing values in the testing dataset (The training set was already preprocessed using the steps described in the Preprocessing and EDA section).

- We split the training data into an **80-20 train-validation** ratio to evaluate model performance.

- We used **RandomizedSearchCV** for hyperparameter tuning. The **best parameters** obtained were as follows:
  {'n_estimators': 300, 'learning_rate': 0.2, 'estimator__min_samples_split': 2, 'estimator__min_samples_leaf': 1, 'estimator__max_depth': 4}

- The **validation RMSE** was 25289.239540544582.

- We re-trained the model on the entire training data (training + validation set) before predicting the values for test.csv.

- The model achieved a **Kaggle score** of 46641.093.

6. Ridge Regression

- The target variable HotelValue was separated from the features, and irrelevant columns like Id were removed.

- Numerical features were imputed with the median, standardized, and power-transformed to improve linearity and model stability.

- Categorical features were imputed using the most frequent value and one-hot encoded for model compatibility.

- Ridge regression with cross-validation was used to select the optimal regularization parameter, controlling overfitting while stabilizing coefficients.

- Low-importance features were removed using Ridge-based feature selection to reduce dimensionality and improve generalization.

- **R^2 Score:** 0.8879

- The **validation RMSE** was 28019.59

- Model performance was evaluated using R^2 and RMSE, then retrained on the full dataset to generate final predictions for the test set.

- The model achieved a **Kaggle score** of 33895.045

7. Lasso Regression

- HotelValue was separated as the target, and non-informative columns like Id were excluded.
- Numerical features were median-imputed, standardized, and transformed to reduce skewness and improve linearity.
- Categorical features were imputed with the most frequent value and one-hot encoded to make them usable for modeling.
- Lasso regression with cross-validation was applied to select the optimal alpha, with L1 regularization automatically removing irrelevant features.
- Feature selection via Lasso improved model sparsity and interpretability while reducing overfitting.
- Model performance was measured using R^2 and RMSE, and the model was retrained on the full dataset for final test set predictions.
- **R^2 Score:** 0.8523
- The **validation RMSE** was 32161.34
- The model achieved a **Kaggle score** of 30974.838

8. Elastic Nets

- First, we utilized **feature engineering** to create several new features to better model relationships between property characteristics.
- We **removed outliers** based on total surface area, excluding records where the value exceeded 4400 (this optimal value was found after numerous trials), as these represented unusually large properties that could distort regression estimates.
- The target variable (HotelValue) was **log-transformed** using np.log1p() to stabilize variance and improve model performance on skewed price data.
- Features were then imputed, scaled using StandardScaler, encoded using **One-Hot Encoding,** and reduced via **PCA** (n_components = 0.95) to retain 95% of variance while minimizing dimensionality.
- We used **ElasticNetCV** from scikit-learn, which automatically tunes the model's hyperparameters through cross-validation. The search space included multiple l1_ratio values and alpha values, and we trained the model using 5-fold cross-validation.

- After fitting the model on the processed training data, we transformed the predictions for the test set back using the exponential function (np.expm1). The final predictions were clipped at zero to avoid negative prices.

- Our Elastic Net model achieved a **Kaggle score** of 19999.523, based on the RMSE metric.

9. K-Nearest Neighbors (KNN) Regression

- The target HotelValue was separated from features, with Id removed.

- Numerical features were median-imputed and standardized to ensure proper distance calculations for KNN.

- Categorical features were imputed with the most frequent value and one-hot encoded for compatibility with distance-based modeling.

- Hyperparameters including number of neighbors, distance metric, and weighting scheme were optimized using grid search with cross-validation.

- **Best Parameters Found:** {'knn__n_neighbors': 8, 'knn__p': 1, 'knn__weights': 'distance'}

- The final KNN model was trained on the full dataset, and predictions were generated for the test set.

- **R^2 Score:** 1.0000

- The **validation RMSE** was 0.0000

- KNN captures local patterns by averaging outcomes of similar instances, with performance highly dependent on preprocessing and selected hyperparameters.

- The model achieved a **Kaggle score** of 45263.419

10. Bayesian Ridge Regression

- The target variable HotelValue was separated, clipped to remove extreme outliers, and log-transformed to stabilize variance and reduce skewness.

- Numerical features were median-imputed, expanded using polynomial features to capture interactions, standardized, and power-transformed to improve linearity.

- Categorical features were imputed with the most frequent value and one-hot encoded to allow modeling with Bayesian Ridge.

- A Bayesian Ridge regression model with feature selection was used, where features with low importance were removed to improve generalization and reduce dimensionality.

- Hyperparameters (alpha_1, alpha_2, lambda_1, lambda_2) were optimized using grid search with cross-validation to maximize R^2 performance.

- **Best Parameters Found:** {'regressor__alpha_1': 1e-08, 'regressor__alpha_2': 1e-05, 'regressor__lambda_1': 1e-05, 'regressor__lambda_2': 1e-08}

- Model performance was evaluated on a validation set using R^2 and RMSE, and cross-validation confirmed stability and reliability of predictions.

- **Cross-Validation Mean R^2:** -5582.82134 ± 16751.08564

- **R^2 Score:** 0.90899

- The **validation RMSE** was 26924.550

- The model was retrained on the full dataset, predictions for the test set were exponentiated and clipped to avoid negative values, and saved with Id in the final CSV file.

- Bayesian Ridge provides regularization with probabilistic estimation, balancing bias and variance while capturing uncertainty in predictions.

- The model achieved a **Kaggle score** of 29325.086

11. Maximum Likelihood Estimation (MLE)

- We implemented Linear Regression based on the Maximum Likelihood Estimation (MLE) principle, assuming Gaussian-distributed errors with constant variance.
- The target variable (Log_HotelValue) was used to reduce skewness and stabilize variance.
- All categorical features were one-hot encoded, resulting in 222 numerical features.
- The dataset was split into 80% training and 20% testing subsets for evaluation.
- The model was trained using scikit-learn's LinearRegression(), which automatically performs parameter estimation via MLE.
- Evaluation Metrics:

    - **Mean Squared Error (MSE)**: 1.32×10^9

- **R^2 Score:** 0.704

- The model explained approximately 70% of the variance in hotel property values, serving as a strong baseline model before applying regularized and non-linear techniques.
- The model achieved a **kaggle score** of 455146.781

12. ElasticNet + LightGBM Blended Model

- This is a hybrid regression model combining **ElasticNet** (a regularized linear model) and **LightGBM** (a gradient boosting framework) to enhance predictive accuracy and generalization.
- The model integrates linear interpretability with non-linear feature learning for improved performance on complex property data.
- **ElasticNet Regression** applies both L1 (Lasso) and L2 (Ridge) penalties to balance feature selection and multicollinearity handling.
- **LightGBM Regressor** captures non-linear interactions efficiently using gradient boosting on decision trees.
- The final prediction was a weighted blend of both models:

$$\text{Final Prediction} = 0.6 \times \text{ElasticNet} + 0.4 \times \text{LightGBM}$$

Feature Engineering:

- Created new features such as Age and YearsSinceRemod to represent structural age and renovation recency.
- Applied **log-transformation** to skewed numeric variables (LandArea, GroundFloorArea, ParkingArea, etc.) to reduce skewness.
- Introduced interaction terms:

    - LogArea_x_Qual = LandArea × OverallQuality
    - Age_x_Qual = Age × OverallQuality

- Missing values were imputed (mean for numeric, mode for categorical), and categorical variables were **one-hot encoded**, resulting in **~300 features**.
- All numeric features were standardized using **StandardScaler()** for stability.

Training Setup:

- **ElasticNetCV**: Tuned over 5-fold cross-validation with
  alphas = logspace(-5, -2, 50) and l1_ratios = [0.1, 0.5, 0.9].

- **LightGBM**: Configured with
  learning_rate = 0.02, n_estimators = 1500, max_depth = 7, and num_leaves = 40.

- The dataset used the log-transformed target (Log_HotelValue) to stabilize variance before prediction.

Evaluation Metrics:

- **Mean Squared Error (MSE):** 7.81×10^8

- **R^2 Score:** 0.835

- **Kaggle Score:** 24769.301

The blended model explained approximately **83.5% of the variance** in hotel property values, outperforming the baseline MLE linear regression model.

Final Rankings

1. ElasticNet Regression (19,999.523) - ElasticNet emerged as the **most accurate and best-performing model** overall. By combining **L1 (Lasso)** and **L2 (Ridge)** regularization, it effectively managed **multicollinearity**, reduced **overfitting**, and performed **automatic feature selection**. This balance of bias and variance allowed the model to generalize well on data while maintaining interpretability. Its hybrid penalty structure helped it outperform all other models, making it the **final recommended model** for hotel price prediction.

2. ElasticNet + LightGBM Blended Model (24,769.301) - The hybrid approach combined **ElasticNet's linear stability** with **LightGBM's non-linear feature learning**, yielding strong and consistent performance. It captured complex patterns across numerical and categorical data, while ElasticNet's regularization prevented overfitting. Though slightly behind standalone ElasticNet, the blend demonstrated the advantage of integrating **linear interpretability with boosted non-linear learning**.

3. Bayesian Ridge Regression (29,325.086) - Bayesian Ridge introduced **probabilistic regularization**, providing uncertainty estimates and **adaptive shrinkage** of coefficients. It performed better than standard Ridge or Lasso by optimizing hyperparameters automatically during training. The model was resistant to noise but limited in capturing strong non-linear dependencies.

4. Lasso Regression (30,974.838) - Lasso's L1 regularization improved interpretability by **eliminating irrelevant features**. However, when correlated predictors shared importance, it

tended to zero out some useful ones, causing **mild underfitting**. This made it slightly less accurate than ElasticNet and Bayesian Ridge.

5. Ridge Regression (33,895.045) - Ridge Regression effectively handled **multicollinearity** using L2 regularization, ensuring **coefficient stability**. However, since it doesn't perform feature elimination, redundant predictors remained, leading to overfitting.

6. Gradient Boosting Regressor (40,908.516) - Gradient Boosting captured **non-linear patterns** and interaction effects, outperforming basic ensembles. However, **limited hyperparameter tuning** (e.g., depth, learning rate) resulted in slight overfitting and suboptimal test performance.

7. XGBoost Regressor (41,597.766) - XGBoost applied gradient boosting with **advanced regularization** and efficient computation. It required fine-tuning for best results.

8. Random Forest Regressor (45,739.987) - Random Forest reduced variance through averaging multiple trees, offering stable but less precise predictions. It could not model finer non-linear details as boosting algorithms did. Its bias toward average predictions limited accuracy for extreme property values.

9. K-Nearest Neighbors (KNN) Regression (45,263.419) - KNN captured local similarities well but struggled with **high-dimensional data** after one-hot encoding. Distance metrics became less meaningful, causing perfect training fit ($R^2 = 1.0$) but **poor generalization**. This model was highly sensitive to feature scaling and dataset size.

10. AdaBoost Regressor (46,641.093) - AdaBoost improved upon simple trees by sequentially correcting weak learners' errors, but it was constrained by shallow base estimators. Its performance failed to match the flexibility of Gradient Boosting or XGBoost.

11. Decision Tree Regressor (47,144.334) - The Decision Tree served as a basic non-linear baseline but suffered from **high variance** and **overfitting**. Without pruning or ensemble averaging, it captured noise and failed to generalize beyond training data.

12. Maximum Likelihood Estimation (455,146.781) - MLE-based Linear Regression assumed strict linearity and constant variance, making it highly sensitive to **outliers** and **skewed distributions**. Without regularization or feature transformation, it struggled to model complex data relationships effectively.

Overall Insights

1. **ElasticNet Outperformed All Models:**
   Its dual regularization handled noise, multicollinearity, and feature redundancy

effectively, making it both accurate and stable.

2. **Regularized Linear Models Were Most Reliable:**
   ElasticNet, Bayesian Ridge, Lasso, and Ridge consistently ranked higher than unregularized or distance-based models.

3. **Boosting Models Showed Strong Potential:**
   LightGBM, Gradient Boosting, and XGBoost captured complex interactions but required extensive tuning to outperform regularized regressors.

4. **Simple Models Performed Poorly:**
   Decision Tree, KNN, and MLE struggled due to overfitting or weak generalization, emphasizing the importance of feature engineering and regularization.

5. **Impact of PCA and Feature Engineering:**

   We saw huge improvements in the performance of most models (especially elastic nets) when PCA and effective feature engineering were applied. We also noted that the optimal value of n_components for PCA seemed to be 0.95. Values more or less than this seemed to reduce the accuracy of the model.