ML Project Report: Hotel Value Prediction (Checkpoint 1)

Team Members

1. Priyanshu Tiwari (IMT2023010)
2. Siddharth Kini (IMT2023026)

Github Link: https://github.com/BookwormBoy/HotelValuePrediction

Project Task

The primary objective of this project was to develop and evaluate various machine learning models for accurately predicting the value of hotels based on their features, making it a regression problem. To achieve this, we did:

- **Data Pre-processing:** This involved cleaning the dataset, handling missing values, encoding categorical features, and scaling numerical features to prepare the data for model training.
- **Exploratory Data Analysis (EDA):** We conducted a thorough EDA to understand the distribution of features, identify correlations, and uncover insights into factors influencing hotel value.
- **Model Training and Evaluation:** We trained and evaluated various machine learning models suitable for regression tasks, including linear regression, decision trees, random forests, and gradient boosting models. We also explored hyperparameter tuning to optimise model performance.

Dataset and Features Description

The evaluation of a hotel is one of the important tasks undertaken by stakeholders, such as shareholders and banks, which helps them make informed decisions regarding their investments and loans. Therefore, predicting hotel value becomes of utmost importance. To achieve the most accurate and efficient prediction, we need to examine past data, i.e., the already evaluated hotels, along with the features of the hotel that support the evaluation. We start the process by understanding the dataset, which consists of 1200 rows and 81 columns. The columns comprise the physical features, ranging from the hotel's parking area to the presence of air conditioners.

EDA and Preprocessing

Before training our models and making predictions, we need to analyse, understand and perform a sanity check of the training and testing data. We begin by conducting Exploratory Data Analysis (EDA), which helps us understand the quality of the features and their relationship to the target value.

Importing Libraries and Data Overview

We begin our process by importing the necessary libraries, including NumPy, Pandas, Matplotlib, and Seaborn. These libraries are useful for loading the dataset into our program (using a Pandas DataFrame) and manipulating it for cleaning and evaluating the quality of the dataset and its features.

Data Overview

After loading the dataset, we use the .info() method to get an overview of the training data:

- Dimensions: 1200 rows and 81 columns.
- Data Types: A mix of 4 float64, 34 int64, and 43 object (categorical) columns.

Data Cleaning

We analysed categorical columns with missing values to understand their pattern. We performed domain-driven imputations, where columns with NaN values due to non-existent features were imputed with “None”. The parking features were only missing for rows with parking area=0, so they can be imputed with None too. In general, the basement features were missing only for those rows with a total basement area = 0. These can be imputed with None too. ‘PoolQuality’ was dropped entirely, as most rows were missing it. The ’ Electrical System’ had only one missing row and was imputed using the mode.

Features such as 'UtilityAccess', 'RoadType', ‘MonthSold', 'BasementHalfBaths', and 'YearSold' showed minimal variation across categories, as these are unlikely to be strong predictors of the target variable. Hence, they were dropped from the training and test datasets.

Feature Engineering

We perform feature engineering to create new features and transform existing ones, thereby enhancing the model’s predictive accuracy. We did so by using different encoding schemes.

We performed Ordinal Encoding for the following ordered categories and mapping.

```
ordinal_cols = ['ExteriorQuality', 'ExteriorCondition', 'BasementHeight',
'BasementCondition', 'BasementExposure',  'HeatingQuality', 'KitchenQuality',
'LoungeQuality', 'ParkingQuality', 'ParkingCondition']

ordinal_mappings = {'Ex': 5, 'Gd': 4, 'TA': 3, 'Av': 3, 'Fa': 2, 'Mn': 2, 'Po':
1, 'No': 0, 'None': 0}
```

We also performed Target Encoding on high-cardinality categorical columns, where we replaced each categorical value with a numerical statistic derived from the target variable (HotelValue). It helps us efficiently capture the relationship between the categorical columns and the target variable. For low-cardinality categorical columns, we performed One-Hot Encoding, which creates binary indicators for each category, helping us avoid artificial ordering and expand categorical data.

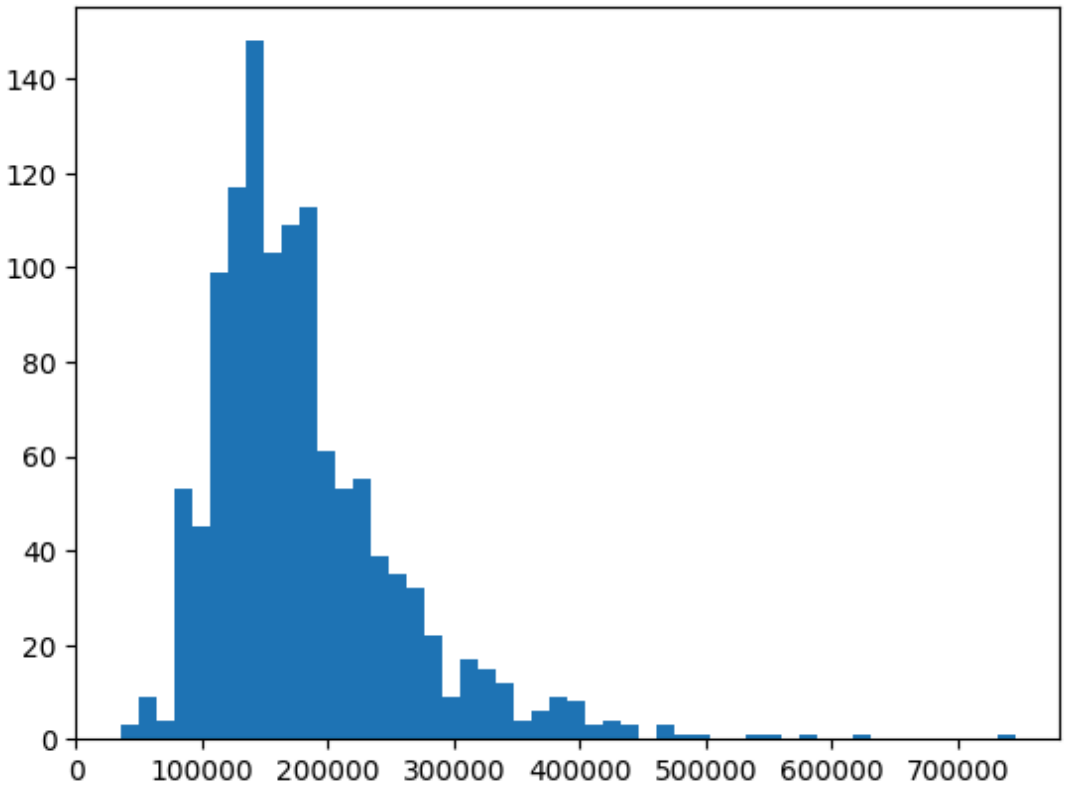
Exploratory Data Analysis (EDA)

Various plots, including histograms, correlation heatmaps, scatterplots, and box plots, were generated to analyse the data distribution, detect outliers, and explore relationships among features. These visualisations provided critical insights that informed subsequent preprocessing and modelling decisions.

Target Variable Histogram

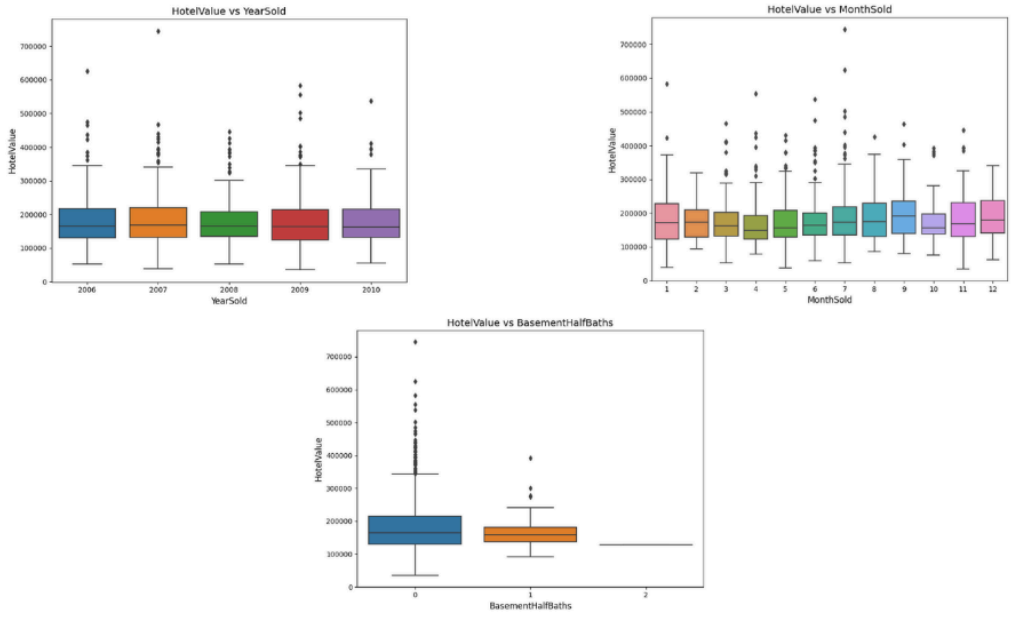
The target variable, HotelValue, exhibited a **right-skewed distribution**, i.e., most hostels in the dataset have moderate values, while a smaller number of properties have exceptionally high values. Upon closer look, most of the hotel values fall between $100,000 and $200,000.

Due to the skewed nature of the histogram, we decided to **log-transform** the target variable before training, which reduces the effect of outliers. Before generating the final predictions, we applied an inverse log transform to return the values to their original scale. This resulted in a 4-5% improvement in the boosting models.



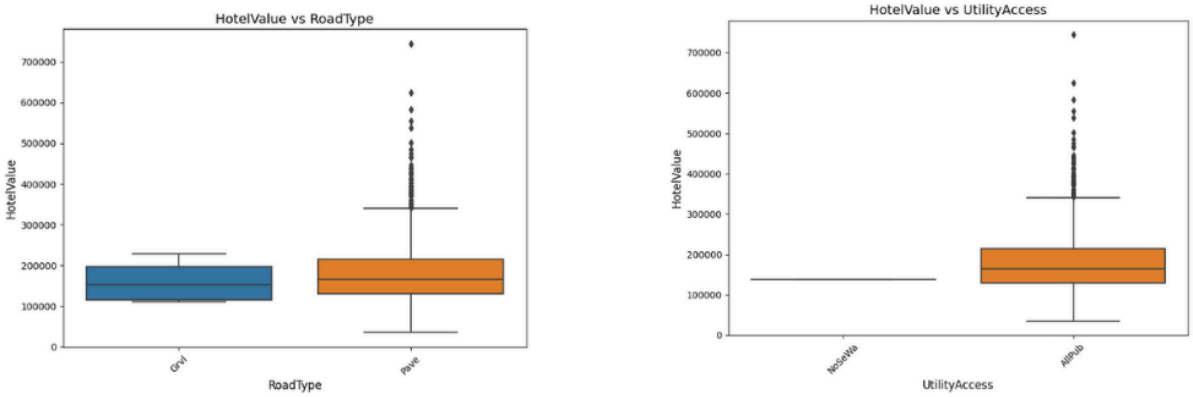
Numeric Features vs Target Value (Boxplots and Scatterplots)

Boxplots were used to visualise numeric features with low cardinality, while scatterplots were employed to analyse numeric features with high cardinality. While most of the features looked consistent without extreme values, features like ‘MonthSold’, ‘BasementHalfBaths’ and ‘YearSold’ showed minimal variation, and were unlikely to be strong predictors of the target variables. Therefore, we dropped these columns.



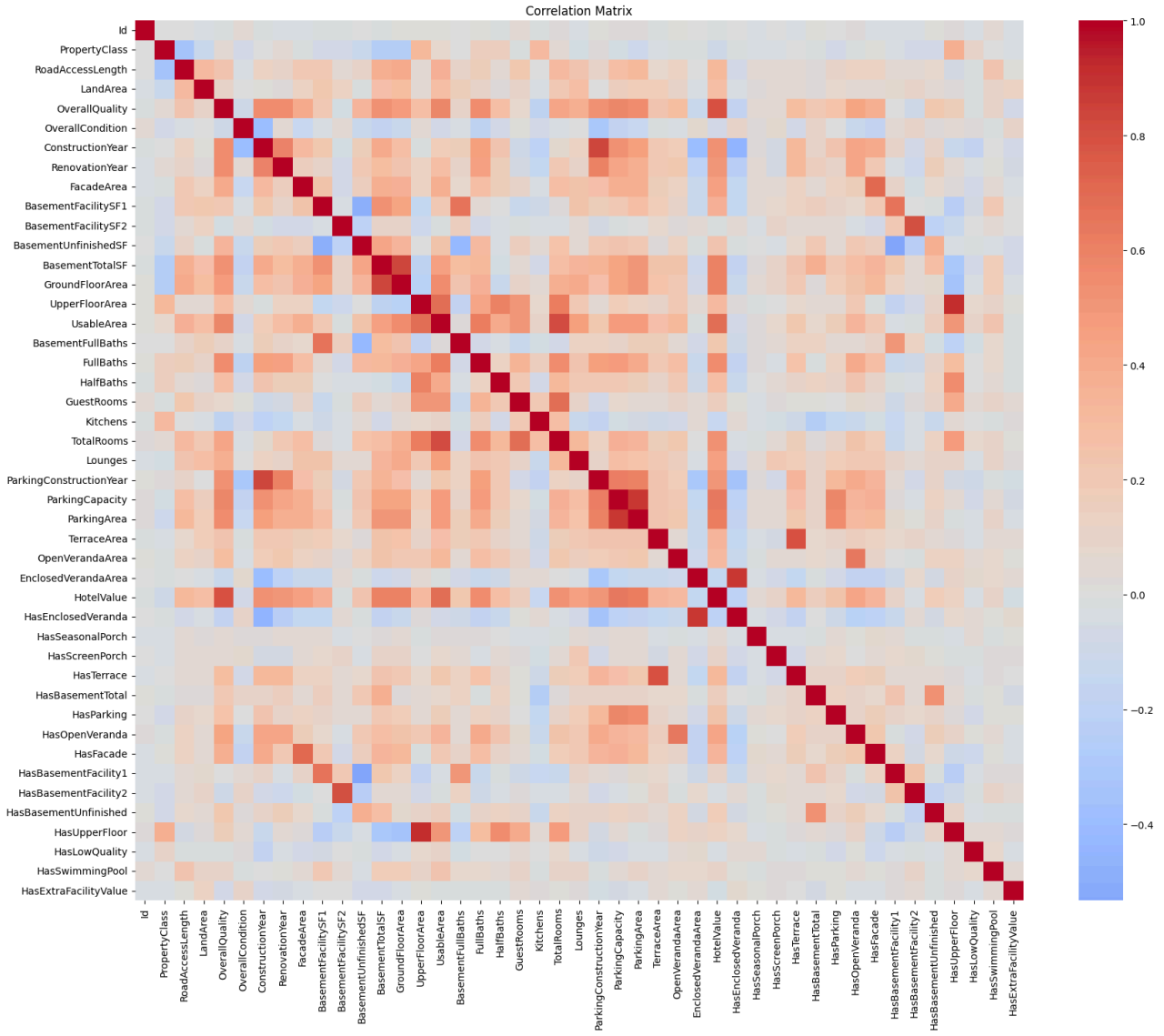
Categorical Features vs Target Value (Boxplots)

For categorical features, we use boxplots to visualise how HotelValue varied across category levels. Features like UtilityAccess and RoadType showed minimal variation and were unlikely to be strong predictors of the target variables. Therefore, we dropped these columns.



Correlation Matrix (Heatmap)

We generated a correlation heatmap to visualise the relationship among numerical features and with the target variable ‘HotelValue’.



- Overall Quality and Hotel Value had the **highest correlation of 0.788**, indicating that Overall Quality has a significant influence on hotel valuation.
- Features related to parking, like ParkingConstructionYear, ParkingCapacity, and ParkingArea, showed a significant correlation with HotelValue.
- HotelValue had the lowest correlation with features like HasEnclosedVeranda and EnclosedVerandaArea, indicating features related to Veranda aren’t helpful in predicting the value of a hotel.
- **Facility-based features**, such as SwimmingPoolArea and ExtraFacilityValue, showed very weak or negligible correlations with HotelValue. It was also evident from the scatterplot

constructed between them and the target value. Such features are dropped during our preprocessing, as they added little predictive power.

Splitting The Dataset

For training our machine learning models, we split our dataset into training and testing sets. The split ratio is 80% for training and 20% for testing. We do this to evaluate the model's performance on unseen data and tune it accordingly.

Summary

During our preprocessing and EDA phase, we:

- Handled missing values using domain-specific replacements, mode imputation, and dropped features with high null values.
- Dropped low-variance and low-correlation features to reduce noise.
- Applied ordinal encoding for quality-based features, target encoding for high-cardinality categorical columns, and one-hot encoding for low-cardinality ones.
- Performed EDA to understand that OverallQuality and parking-related features were highly related, whereas facility-based ones weren't.
- Split the dataset into training and testing sets, and complete the data preparation process.

Models Used For Prediction

Linear and Polynomial Regression (Ridge, Lasso, and ElasticNets)

- For linear models, such as Linear Regression, we handled preprocessing differently. We perform thorough **feature engineering** by creating features such as TotalSD, which is calculated by summing BasementTotalSD, GroundFloorArea, and UpperFloorArea.
- We only performed one-hot encoding as it creates independent binary features for each category. We also performed comprehensive imputation.
- We also use **Principal Component Analysis (PCA)** for dimensionality reduction. With one-hot encoding, we got a vast number of features. PCA reduces this dimensionality, which helps eliminate noise. After one-hot encoding, we had 235 features, and the PCA-reduced features were 171.
- With the above preprocessing, we trained vanilla linear regression, Ridge Regression, Lasso Regression, and Elastic Net Regression. The RMSE and hyperparameters for them are:
    - **Vanilla linear regression:**
        - RSME on test.csv: 19537.931
    - **Ridge Regression:**
        - Best Ridge Alpha: 100.0
        - Best Ridge RMSE (negated): -0.1449726879752317
        - RMSE on test.csv: 19485.601
    - **Lasso Regression:**

        - Best Lasso Alpha: 0.0033932217718953264
        - Best Lasso RMSE (negated): -0.1477009340930391
        - RMSE on test.csv: 21504.308
    - **Elastic Net Regression:**
        - Best ElasticNet Params: {'alpha': 0.0307029062975785, 'l1_ratio': 0.1}
        - Best ElasticNet RMSE (negated): -0.14669784029324753
        - RMSE on test.csv: 19487.840
- When we attempted **polynomial regression**, the high number of features resulting from one-hot encoding led to poorer results. When we tried quadratic regression, the RMSE on test.csv crossed 35k. Therefore, we decided to stick with linear regression.

Regression Tree

- The preprocessing differs from that in the previous section. We noticed that with one-hot encoding, we got poor RMSE, and realised that creating new features was inefficient for trees. We found LabelEncoder to be more memory-efficient and to avoid misinterpretation of the numerical relationship between categories.
- We used GridSearchCV for hyperparameter tuning and got the following results:
    - Best Hyperparameters: {'max_depth': 10, 'max_features': None, 'min_samples_leaf': 3, 'min_samples_split': 10}
- Upon testing the model with the test.csv dataset, we got the RMSE of 27696.249.

Random Forest

- The preprocessing differs from that in the previous section. We noticed that with one-hot encoding, we got poor RMSE, and realised that creating new features was inefficient for trees. We found LabelEncoder to be more memory-efficient and to avoid misinterpretation of the numerical relationship between categories.
- We used GridSearchCV for hyperparameter tuning and got the following results:
    - Best Hyperparameters: {'n_estimators': 1500, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_features': 0.5, 'max_depth': None, 'bootstrap': False}
- Upon testing the model with the test.csv dataset, we obtained an RMSE of 24,634.113.

XGBoost

- The preprocessing done on the training and testing data was the same as previously mentioned (in the EDA and Preprocessing section).
- We used GridSearchCV for hyperparameter tuning and got the following results:
    - Best parameters found: {'colsample_bytree': 0.7, 'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 500, 'reg_alpha': 0, 'reg_lambda': 1, 'subsample': 0.7}
    - Best cross-validation RMSE: 0.1193
- Upon testing the model with the test.csv dataset, we got the RMSE of 22801.

Gradient Boosting

- The preprocessing done on the training and testing data was the same as previously mentioned (in the EDA and Preprocessing section).
- We used GridSearchCV for hyperparameter tuning and got the following results:
    - Best parameters found: {'learning_rate': 0.05, 'loss': 'squared_error', 'max_depth': 3, 'n_estimators': 1000, 'subsample': 0.8}
    - Best cross-validated RMSE from GridSearch: 0.1186
- Upon testing the model with the test.csv dataset, we got the RMSE of 21973.

Performance and Interpretation of Different Models

- **Linear Regression** (including Ridge, Lasso, and Elastic Nets) outperformed other models, such as random forests and XGBoost. This win is primarily attributed to its specialised pre-processing pipeline, which normalised the target variable and effectively managed high-dimensional categorical features through one-hot encoding and PCA.
- After linear regression, **Gradient Boosting** gave us the best prediction on the test dataset. It succeeded where other tree models failed because it used the same superior one-hot encoding-based pre-processing as the linear models, and its boosting algorithm effectively learned from the errors of sequential weak learners.
- **The regression tree** proved to be the poorest model, with a 27.7k RMSE score, failing to compete effectively with the simplest linear regression model. This poor performance highlights the model's high variance as single trees are highly prone to overfitting and are generally weak predictors.

Interesting Observations

- **Preprocessing was more critical than model selection:** We expected popular and sophisticated models, such as XGBoost, Gradient Boosting, and Random Forests, to outperform Linear Regression, but that wasn't the case. Preprocessing created a cleaner dataset and played a crucial role in the success of linear regressions. The similar preprocessing gave poorer results on the well-known ensemble methods.
- **Regularisation offered almost no benefit:** Ridge and ElasticNet (RMSE 19.4k) barely outperformed vanilla Linear Regression (RMSE 19.5k). This suggests the PCA step was so effective at "eliminating noise" and handling multicollinearity that there was very little overfitting left for the regularisation to fix.