Interpretable Machine Learning for NYC Airbnb Price Prediction

Samuel Orellana Mateo*

November 26th, 2024

Abstract

This project tackles the prediction of Airbnb rental prices in New York City using a dataset of 15,696 training points and 6,727 prediction instances, encompassing 51 diverse features such as property type, location, host information, and amenities. We begin with comprehensive exploratory data analysis to understand feature distributions, correlations, and data quality. Based on these insights, we perform data processing steps including one-hot and ordinal encoding, date transformations, and custom feature engineering (e.g., neighborhood mean pricing, amenities count, host responsiveness). Principal Component Analysis (PCA) reduces multicollinearity, while outlier handling stabilizes model training.

We employ Random Forest and XGBoost regression models, optimizing their hyperparameters through Bayesian optimization with Optuna. This automated tuning significantly enhances predictive accuracy, with our best XGBoost model achieving an RMSE of approximately 0.79 on the competition leaderboard, surpassing simpler models.

Beyond accuracy, we prioritize model interpretability by engineering intuitive composite features and conducting feature importance analyses. These reveal that room type, neighborhood price indicators, and listing capacity are key drivers of price variation. Our approach not only improves predictive performance but also provides clear insights into the factors influencing Airbnb prices in NYC.

* Duke University, `samuel.orellanamateo@duke.edu`

1 Introduction

This project was undertaken as part of the Computer Science 671 course at Duke University during the Fall semester of 2024, under the instruction of Professor Cynthia Rudin, a distinguished expert in interpretable machine learning. The primary objective was to participate in a Kaggle competition focused on predicting Airbnb rental prices in New York City. The dataset utilized for this competition is accessible at `https://insideairbnb.com/new-york-city/`.

The dataset provided in the competition had two primary components: a training set containing labeled instances with price categories and a test set requiring price predictions without provided labels. The price labels are integer values ranging from 0 to 5, each representing predefined price intervals rather than exact monetary amounts. This categorization transforms the problem into a regression task where the goal is to accurately predict the appropriate price range for each listing based on its features.

Participants were supplied with the training dataset to develop and validate their models, while the test dataset was used for generating predictions to be submitted to Kaggle. To ensure the integrity of the competition and prevent overfitting to the final evaluation metric, Kaggle employed a hidden subset of the test data to calculate the Root Mean Squared Error (RMSE) score. This approach required participants to optimize their models using only the provided training data, with the RMSE score reflecting performance on an unseen portion of the dataset.

Our project focused on achieving high predictive accuracy while maintaining model interpretability to better understand the factors influencing Airbnb pricing in New York City.

2 Exploratory Analysis

Initially, we visualized non-numerical features to understand the number of unique values they possess. The summary is presented in Table 1.

Index	Feature	Unique Count
0	`name`	15,189
1	`amenities`	13,314
2	`description`	12,687
3	`reviews`	11,215
4	`host_since`	4,037
5	`first_review`	3,261
6	`last_review`	1,390
7	`neighbourhood_cleansed`	217
8	`property_type`	59
9	`bathrooms_text`	30
10	`host_verifications`	6
11	`neighbourhood_group_cleansed`	5
12	`room_type`	4
13	`host_response_time`	4
14	`host_is_superhost`	2
15	`host_identity_verified`	2
16	`host_has_profile_pic`	2
17	`instant_bookable`	2
18	`has_availability`	1

Table 1: Unique Counts of Non-Numerical Features

This prompted us to start by exploring the features with a low number of unique values, and then process the remaining features accordingly by analyzing their distributions and relationships with the target variable in detail.

Subsequently, through the Data Processing phase, we employed various visualization techniques to understand the dataset and modify it for optimal model performance. Furthermore, we utilized visualization techniques during the model selection process. Some of the techniques used were:

- **Histograms:** Used to assess the distribution of numerical features and identify outliers.
- **Scatter Plots:** Utilized to explore relationships between pairs of features, particularly those identified as highly correlated.
- **Correlation Heatmaps:** Generated to visualize the strength and direction of relationships between features, aiding in the identification of groups for PCA.

- **Confusion Matrices:** Constructed to evaluate model performance on binary classification tasks.
- **Hyperparameter Tuning Plots:** Visualized the relationship between predictive accuracy on a subset of the training data and varying hyperparameters.
- **Feature Importance Plots:** Created to assess the relative importance of features in the model's decision-making process.

This exploratory process informed subsequent feature engineering and selection steps, ensuring that the most relevant and non-redundant features were utilized in the modeling phase.

3 Data Processing

3.1 One Hot Encoding

For features with fewer than 10 unique values, we applied one-hot encoding:

- `host_verifications` was one-hot encoded into `phone`, `email`, and `work_email`.
- `neighbourhood_group_cleansed` was one-hot encoded into its 5 distinct neighbourhoods.

3.2 Ordinal Encoding

Certain categorical features were ordinally encoded based on their qualitative assessment:

- `room_type` was rated from 0 to 3 based on quality:

$$\begin{aligned}\texttt{Shared room} &: 0,\\ \texttt{Private room} &: 1,\\ \texttt{Hotel room} &: 2,\\ \texttt{Entire home/apt} &: 3.\end{aligned}$$

  This mapping was determined empirically to optimize model performance, with higher values representing better room quality.

- `host_response_time` was similarly rated:

$$\begin{aligned}\texttt{a few days or more} &: 0,\\ \texttt{within a day} &: 1,\\ \texttt{within a few hours} &: 2,\\ \texttt{within an hour} &: 3.\end{aligned}$$

  This scoring reflects the responsiveness of the host, with higher scores indicating quicker responses.

3.3 Binary Encoding

The following binary features were one-hot encoded into 0 and 1, where 0 represents `false` and 1 represents `true`. In cases of missing values, we assumed a default value of 0 (`false`):

- `host_is_superhost`
- `host_identity_verified`
- `host_has_profile_pic`
- `instant_bookable`
- `has_availability`

For example, if there is no information about the host having a profile picture, it is assumed that the host does not have one.

3.4 Date Features

The features `last_review`, `first_review`, and `host_since` were originally in date format. We converted these into the number of days since the initial submission date of this project, November 24, 2024. This transformation allows these features to be normalized effectively, as the specific reference date does not impact the relative differences between data points.

3.5 Sentiment Analysis

For textual features `reviews`, `name`, and `description`, we performed sentiment analysis using NLTK's `SentimentIntensityAnalyzer`:

- For `reviews`, each review string was converted into an array. Using the `langdetect` library, we detected the language of each review. Non-English reviews were translated to English using the OpenAI GPT-4 API before applying sentiment analysis. The sentiment scores, ranging from 0 to 1, were averaged across all reviews.

- For `name` and `description`, we concatenated both strings and performed sentiment analysis on the combined text, resulting in a sentiment score between 0 and 1.

However, the sentiment analyzer exhibited non-deterministic behavior, producing varying results upon each run. Additionally, the sentiment scores sometimes appeared arbitrary. This adversely affected model performance. Consequently, we decided to exclude these sentiment features from the final model.

3.6 Neighborhood Features

For `neighbourhood_cleansed`, we calculated the mean price per neighborhood by grouping the data accordingly. This mean price was then mapped back to each listing to provide an indicator of the neighborhood's expense level. In cases where the test set included neighborhoods not present in the training set, we employed a K-Nearest Neighbors (KNN) approach with $k = 5$ to estimate the mean price based on the nearest neighbors from the training set, thereby avoiding data leakage.

3.7 Amenities Encoding

We manually selected specific amenities to one-hot encode by reviewing unique amenity values and identifying relevant matches. The mapping is as follows:

$$
\begin{aligned}
\texttt{amenity\_targets} &= \{ \\
\texttt{air\_conditioning} &: [\text{'air conditioning', 'AC'}], \\
\texttt{tv} &: [\text{'TV', 'television'}], \\
\texttt{streaming\_services} &: [\text{'Apple TV', 'Disney+', 'HBO Max',} \\
&\quad \text{'Hulu', 'Netflix', 'Roku', 'Amazon Prime'}], \\
\texttt{refrigerator} &: [\text{'refrigerator', 'fridge'}], \\
\texttt{microwave} &: [\text{'microwave'}], \\
\texttt{wifi} &: [\text{'wifi'}], \\
\texttt{parking} &: [\text{'parking', 'free parking', 'street parking',} \\
&\quad \text{'paid parking', 'driveway parking'}], \\
\texttt{gym} &: [\text{'gym', 'workout', 'exercise'}], \\
\texttt{water\_view} &: [\text{'ocean view', 'sea view', 'water view',} \\
&\quad \text{'river view'}], \\
\texttt{kitchen} &: [\text{'kitchen', 'oven', 'stove'}] \\
&\}
\end{aligned}
$$

For each amenity, a binary feature is created that returns 1 if any of the specified keywords are found in the `amenities` list, and 0 otherwise.

3.8 Principal Component Analysis

After processing all features, we plotted a correlation heatmap (Figure 1) and identified blocks of highly correlated features. To reduce data complexity and improve model generalization, we decided to perform Principal Component Analysis (PCA) on these correlated groups.

PCA is a dimensionality reduction technique that transforms a large set of variables into a smaller one while retaining most of the original data's variability. Mathematically, PCA involves projecting the data into a lower-dimensional space by identifying the principal components—orthogonal directions that capture the maximum variance in the data.

For implementing PCA, we standardized the selected feature groups to have a mean of zero and a standard deviation of one. This standardization is crucial because PCA is sensitive to the variances of the original variables. After standardization, PCA was applied to each group of highly correlated features to extract the principal components that encapsulate the majority of the variance within each group.

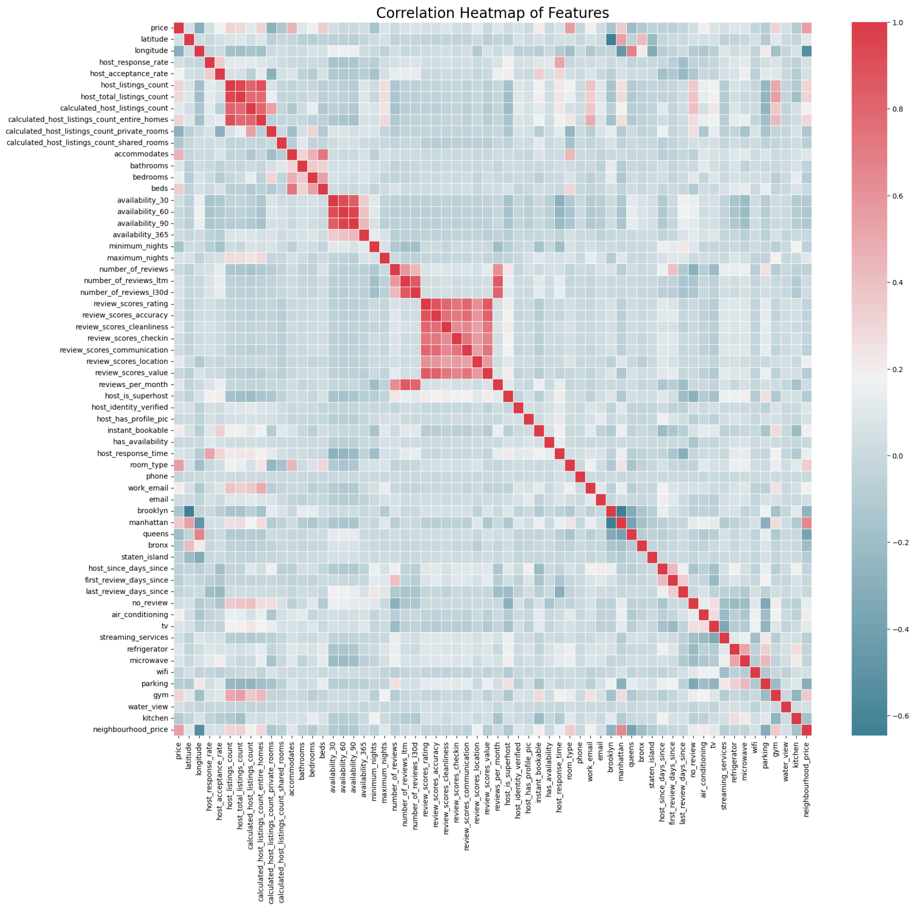


Figure 1: Correlation Heatmap Before PCA

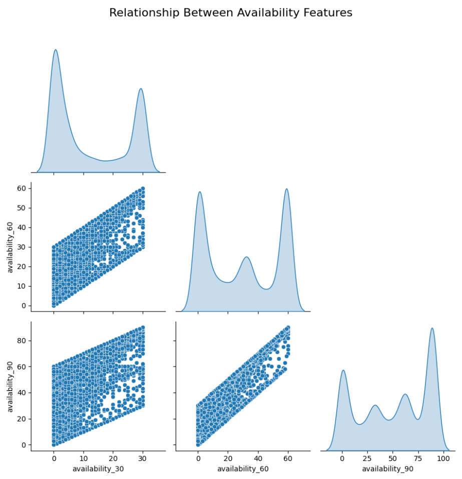
To reduce data complexity and improve model generalization, we applied PCA to the following feature groups, which showed high correlation in the previous matrix. Then, we generated all pairwise plots generated by each block, as shown in Figure 2.

1. **Availability Features:** `availability_30`, `availability_60`, and `availability_90` were highly correlated (Figure 2a) as they represent Airbnb availability in the next 30, 60, and 90 days, respectively. PCA reduced these to a single feature, `availability_short`.

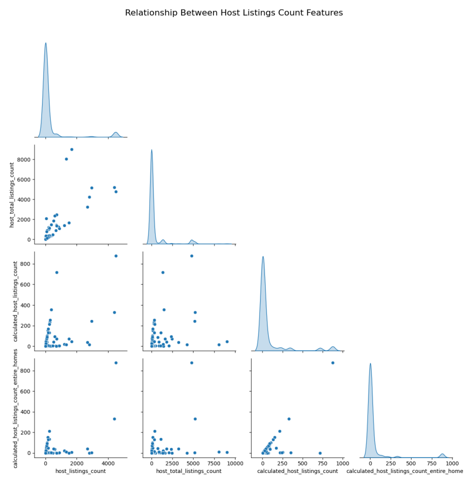
2. **Host Listings Features:** `host_listings_count`, `host_total_listings_count`, `calculated_host_listings_count`, and `calculated_host_listings_count_entire_homes` were highly correlated (Figure 2b), representing various aspects of the host's listing count. PCA reduced these to `hlc_short`.

3. **Number of Reviews Features:** `number_of_reviews`, `number_of_reviews_ltm`, `number_of_reviews_l30d`, and `reviews_per_month` were highly correlated (Figure 2c), indicating overall review activity. PCA reduced these to `nor_short`.

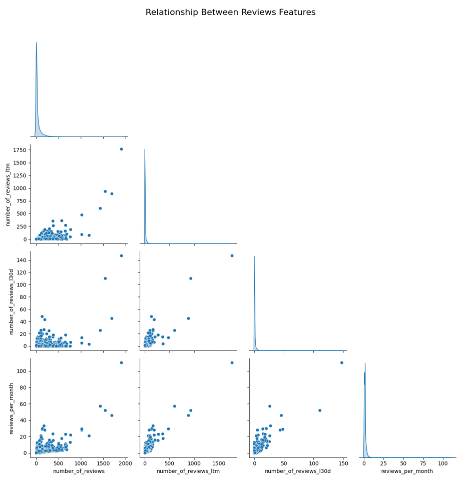
4. **Review Scores Features:** `review_scores_rating`, `review_scores_accuracy`, `review_scores_cleanliness`, `review_scores_checkin`, `review_scores_communication`, `review_scores_location`, and `review_scores_value` were highly correlated (Figure 2d), reflecting overall review quality. PCA reduced these to `rs_short`.



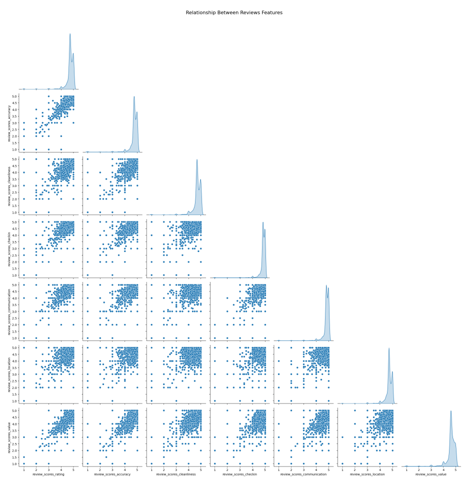
(a) Availability Features



(b) Host Listings Features



(c) Number of Reviews Features



(d) Review Scores Features

Figure 2: Pairwise Relationships of Feature Groups

After applying PCA and dropping the original correlated variables, the correlation heatmap (Figure 3) shows a reduction in multicollinearity, facilitating better model generalization and reduced data complexity.

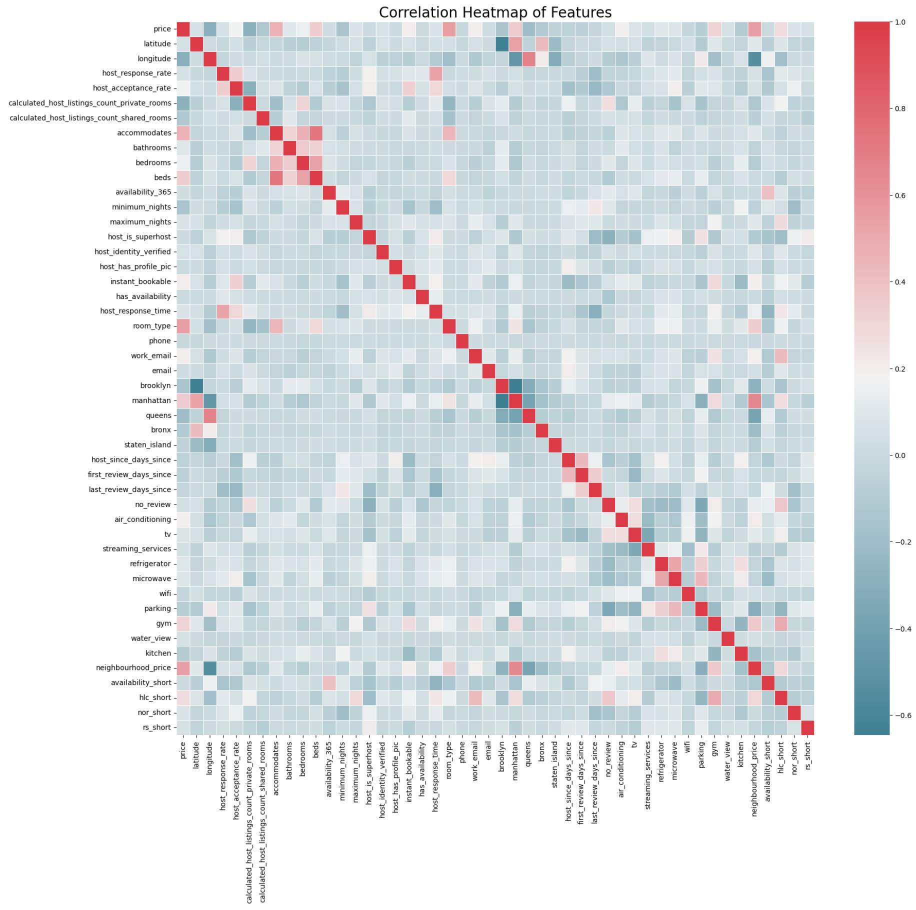
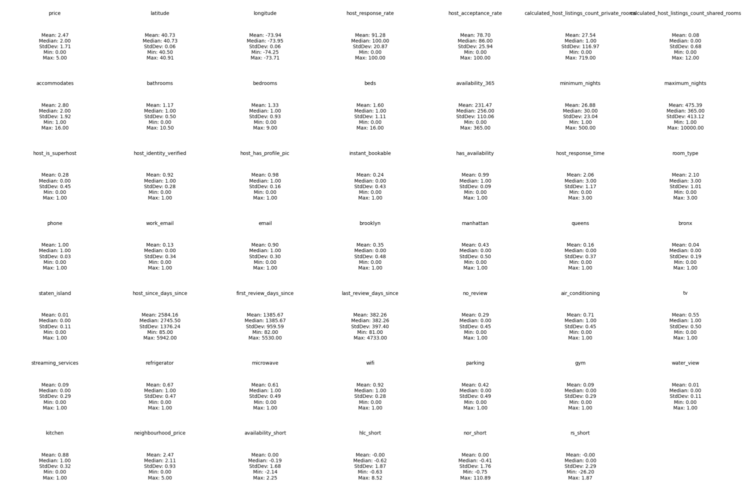


Figure 3: Correlation Heatmap After PCA

3.9 Outliers and Distribution Normalization

We examined the distribution of each feature at this point to identify outliers and non-standard distributions that could impact model performance. Figures 4 and 5 show the summary statistics and distributions of the features, respectively.



Feature	Mean	Median	StdDev	Min	Max
price	2.47	2.00	1.71	0.00	5.00
latitude	40.73	40.73	0.06	40.50	40.91
longitude	-73.94	-73.95	0.06	-74.25	-73.71
host_response_rate	91.28	100.00	20.87	0.00	100.00
host_acceptance_rate	78.70	86.00	25.94	0.00	100.00
calculated_host_listings_count_private_rooms	27.54	1.00	116.97	0.00	719.00
calculated_host_listings_count_shared_rooms	0.08	0.00	0.68	0.00	12.00
accommodates	2.80	2.00	1.92	1.00	16.00
bathrooms	1.17	1.00	0.50	0.00	10.50
bedrooms	1.33	1.00	0.93	0.00	9.00
beds	1.60	1.00	1.11	0.00	16.00
availability_365	231.47	256.00	110.06	0.00	365.00
minimum_nights	26.88	30.00	23.04	1.00	500.00
maximum_nights	475.39	365.00	413.12	1.00	10000.00
host_is_superhost	0.28	0.00	0.45	0.00	1.00
host_identity_verified	0.92	1.00	0.28	0.00	1.00
host_has_profile_pic	0.98	1.00	0.16	0.00	1.00
instant_bookable	0.24	0.00	0.43	0.00	1.00
has_availability	0.99	1.00	0.09	0.00	1.00
host_response_time	2.06	3.00	1.17	0.00	3.00
room_type	2.10	3.00	1.01	0.00	3.00
phone	1.00	1.00	0.03	0.00	1.00
work_email	0.13	0.00	0.34	0.00	1.00
email	0.90	1.00	0.30	0.00	1.00
brooklyn	0.35	0.00	0.48	0.00	1.00
manhattan	0.43	0.00	0.50	0.00	1.00
queens	0.16	0.00	0.37	0.00	1.00
bronx	0.04	0.00	0.19	0.00	1.00
staten_island	0.01	0.00	0.11	0.00	1.00
host_since_days_since	2584.16	2745.50	1376.24	85.00	5942.00
first_review_days_since	1385.67	1385.67	959.59	82.00	5530.00
last_review_days_since	382.26	382.26	397.40	81.00	4733.00
no_review	0.29	0.00	0.45	0.00	1.00
air_conditioning	0.71	1.00	0.45	0.00	1.00
tv	0.55	1.00	0.50	0.00	1.00
streaming_services	0.09	0.00	0.29	0.00	1.00
refrigerator	0.67	1.00	0.47	0.00	1.00
microwave	0.61	1.00	0.49	0.00	1.00
wifi	0.92	1.00	0.28	0.00	1.00
parking	0.42	0.00	0.49	0.00	1.00
gym	0.09	0.00	0.29	0.00	1.00
water_view	0.01	0.00	0.11	0.00	1.00
kitchen	0.88	1.00	0.32	0.00	1.00
neighbourhood_price	2.47	2.11	0.93	0.00	5.00
availability_short	0.00	-0.19	1.68	-2.14	2.25
hlc_short	-0.00	-0.62	1.87	-0.63	8.52
nor_short	0.00	-0.41	1.76	-0.75	110.89
rs_short	-0.00	0.00	2.29	-26.20	1.87

Figure 4: Summary Statistics of Features

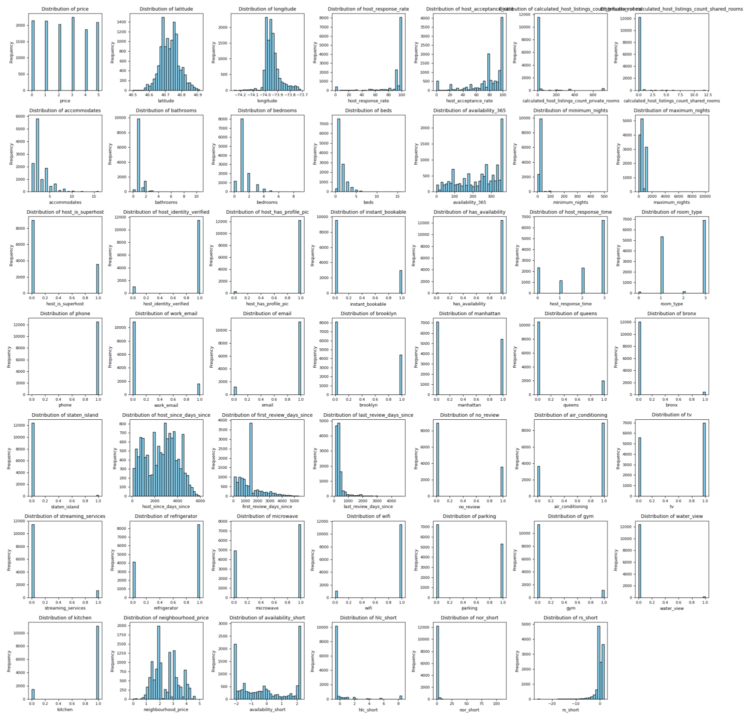
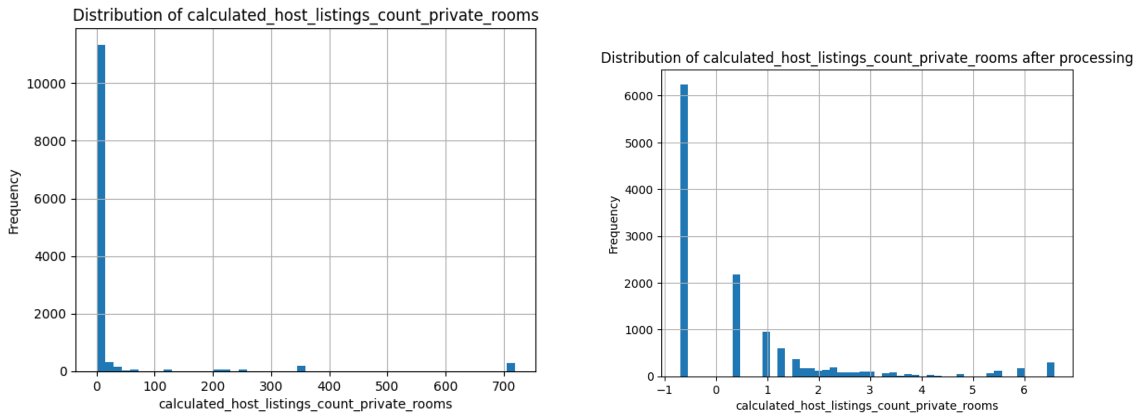


Figure 5: Distribution of Features

3.9.1 `calculated_host_listings_count_private_rooms`



(a) Before Processing (b) After Processing

Figure 6: Distribution of `calculated_host_listings_count_private_rooms`

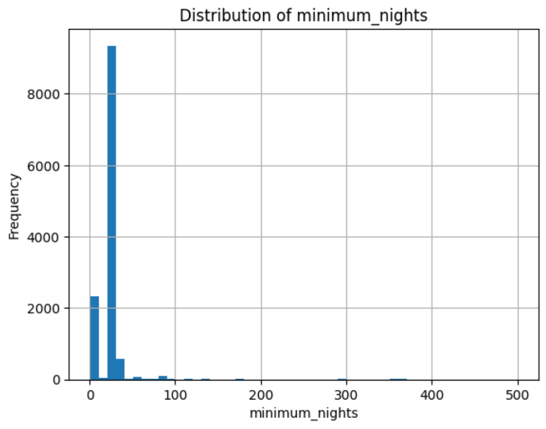
The feature `calculated_host_listings_count_private_rooms` exhibited an exponential-like distribution. Hence, we applied the logarithmic transformation:

$$x \to \log(x + 0.5)$$

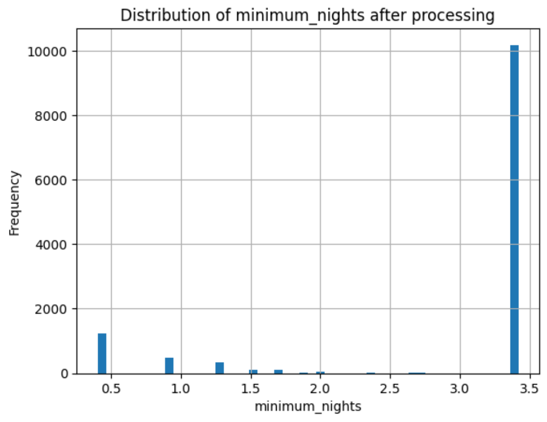
Adding 0.5 avoids taking the logarithm of zero and ensures that a value of 0 maps close to 1. This transformation normalizes the distribution, making it more suitable for machine learning models. However, since decision tree-based models like Random Forests and XGBoost are insensitive to monotonic transformations, this change did not impact their performance. Conversely, for models like Logistic Regression, the transformation improved performance.

The distribution before this transformation is shown in Figure 6a, while the distribution after the transformation is shown in Figure 6b.

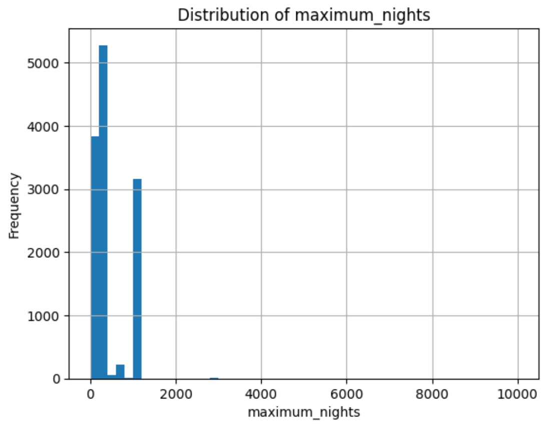
3.9.2 `minimum_nights` and `maximum_nights`



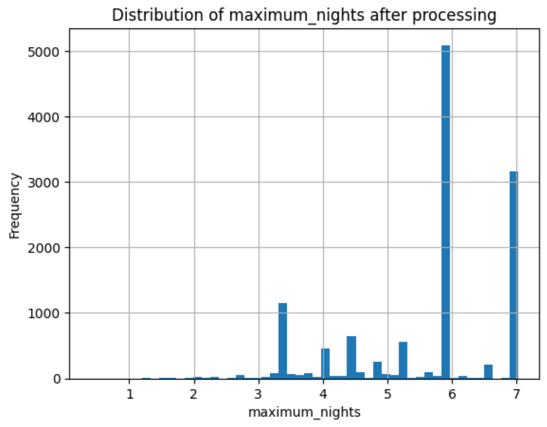
(a) Minimum Nights Before Processing



(b) Minimum Nights After Processing



(c) Maximum Nights Before Processing



(d) Maximum Nights After Processing

Figure 7: Distributions of `minimum_nights` and `maximum_nights`

We observed outliers in `minimum_nights` and `maximum_nights`. To address this:

- For `minimum_nights`, we capped values above 30 to 30, as the 99th percentile was 30. This decision was based on the rationale that requiring more than 30 nights is uncommon and likely represents long-term rentals, which should be treated uniformly.

- For `maximum_nights`, we capped values above 1,124 to 1,124, as this represented an extreme outlier (approximately 4 years). This capping prevents undue influence from these rare cases.

- As both distributions exhibited exponential-like behavior, we transformed both features using:

$$x \to \log(x + 0.5)$$

The distributions before these transformations are shown in Figures 7a and 7c, while the distributions after the transformations are shown in Figures 7b and 7d.

3.10 Feature Dropping

We excluded the following features due to minimal informational value:

- `calculated_host_listings_count_shared_rooms`
- `has_availability`
- `work_email`

3.11 Feature Engineering

To enhance model interpretability, we created new features:

- **Amenities Count:** Defined as the sum of all one-hot encoded amenity features.

$$\begin{aligned}\texttt{amenities\_count} = {} & \texttt{air\_conditioning} + \texttt{tv} + \texttt{streaming\_services} \\ & + \texttt{refrigerator} + \texttt{microwave} + \texttt{wifi} \\ & + \texttt{parking} + \texttt{gym} + \texttt{water\_view} + \texttt{kitchen}\end{aligned}$$

  This aggregated count allows decision trees to evaluate the number of amenities more efficiently, without incurring a regularization penalty.

- **Host Responsiveness:** Defined as the product of `host_response_time` and `host_response_rate`.

$$\texttt{host\_responsiveness} = \texttt{host\_response\_time} \times \texttt{host\_response\_rate}$$

  This composite score ranges from 0 to 300, representing overall host responsiveness after normalization.

4 Models

In this project, we employed two different algorithms for generating predictions: Random Forests and XGBoost. The selection of these models was motivated by their ability to handle complex datasets with numerous features, their robustness to overfitting, and the availability of high-quality implementations in popular machine learning libraries.

Initially, we experimented with various classification and regression models, including Linear Regression, Logistic Regression, Decision Trees, and Neural Networks. However, these models did not yield satisfactory performance on the dataset.

4.1 Ineffective Models

Linear Regression and Logistic Regression were not complex enough to capture the intricate relationships within the data. With approximately 50 features, these models struggled to model the non-linear interactions between variables effectively. This limitation was evident as both training and validation RMSE scores were high and similar, indicating underfitting.

Decision Trees, while capable of modeling non-linear patterns, tended to overfit the training data when not properly regularized. Even with techniques like pruning, individual decision trees did not perform well due to the high dimensionality and complexity of the dataset.

Neural Networks are powerful models known for their ability to learn complex patterns. However, they typically require large amounts of data to perform optimally. In this case, the dataset contained around 15,000 data points, which was insufficient for training an effective neural network without encountering overfitting or underfitting issues. Additionally, neural networks are computationally intensive and require careful tuning of numerous hyperparameters, which was not practical.

4.2 Rationale Behind Model Selection

A common motivation for both models is feature importance: both Random Forests and XGBoost provide tools for feature importance analysis, enabling a better understanding of the models' decision-making processes.

4.2.1 Random Forest

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mean prediction of the individual trees. This approach reduces variance and mitigates overfitting by averaging out the biases of individual trees.

A key motivation for choosing Random Forests is their computational efficiency. While more complex than individual decision trees, Random Forests are parallelizable and computationally efficient, making them suitable for datasets of this size.

4.2.2 XGBoost

XGBoost (Extreme Gradient Boosting) is an optimized gradient boosting framework that uses tree-based models. It builds models sequentially, where each new tree focuses on correcting the errors of the previous ones.

Key motivations for choosing XGBoost include flexibility and efficiency. XGBoost offers extensive hyperparameter tuning options to optimize model performance, making it suitable for complex datasets. Additionally, XGBoost is designed for speed and performance, handling large datasets efficiently.

4.3 Rationale for Regression Models

We employed the `XGBRegressor` from the `xgboost` library, which offers an efficient and scalable implementation of the algorithm, and the `RandomForestRegressor` from `scikit-learn`, a robust implementation of Random Forests.

Although this project was framed as a classification problem, we discovered that using regression models and rounding their outputs provided better results than the classification versions of these models. By leveraging the regression capabilities of XGBoost and applying post-processing techniques, we were able to enhance the predictive performance, achieving more accurate and reliable outcomes.

4.4 External Libraries

Both models were implemented using `scikit-learn` and `xgboost`, which provide robust implementations of Random Forests and XGBoost, respectively. These libraries are open-source and widely used in the machine learning community and offer extensive functionality for model training, evaluation, and hyperparameter tuning.

5 Hyperparameter Selection

To achieve optimal performance for both the `XGBRegressor` and `RandomForestRegressor` models, careful tuning of hyperparameters was essential. Hyperparameters, such as the number of trees in a Random Forest or the learning rate in XGBoost, significantly influence model performance. Selecting appropriate hyperparameter values ensures that the models generalize well to unseen data without overfitting.

5.1 Optuna for Hyperparameter Optimization

We utilized Optuna, a hyperparameter optimization framework in Python, to automate the search for optimal hyperparameter configurations. Optuna employs Bayesian Optimization, specifically using the Tree-structured Parzen Estimator (TPE) as its surrogate model. This method models the objective function $f(\mathbf{x})$, where $\mathbf{x}$ represents the hyperparameters, by estimating two probability densities:

$$l(\mathbf{x}) = p(\mathbf{x} \mid f(\mathbf{x}) < y^*)$$

for promising hyperparameters and

$$g(\mathbf{x}) = p(\mathbf{x})$$

for all possible hyperparameters. The TPE approach optimizes the selection of the next set of hyperparameters by maximizing the ratio:

$$\frac{l(\mathbf{x})}{g(\mathbf{x})}$$

This ratio prioritizes hyperparameters that are more likely to yield better performance based on past evaluations, effectively balancing exploration and exploitation in the search space.

5.2 Hyperparameter Optimization Process

5.2.1 XGBoost Regressor

For the `XGBRegressor`, the hyperparameter optimization involved tuning parameters such as the learning rate (η), maximum tree depth (d), and the number of estimators (n). We also optimized hyperparameters like `subsample`, `colsample_bytree`, `gamma`, `min_child_weight`, `reg_alpha`, and `reg_lambda`.

Initially, we conducted a preliminary search with a narrow range of hyperparameters, which allowed us to train a reasonably good model in approximately 10 minutes on our computer. This initial model provided a baseline for performance.

To further enhance the model's accuracy, we extended the search by allowing Optuna to explore a wider range of hyperparameters over a duration of 4 hours. This extensive search enabled the identification of the best-performing hyperparameter configuration, which significantly improved the model's predictive capabilities.

5.2.2 Random Forest Regressor

Similarly, for the `RandomForestRegressor`, hyperparameter tuning was crucial due to the model's inherent complexity and sensitivity to parameter settings. Key hyperparameters included the number of trees (`n_estimators`), maximum tree depth (`max_depth`), and the minimum number of samples required to split an internal node (`min_samples_split`). We also optimized hyperparameters like `min_samples_leaf`, `max_features`, and `bootstrap`.

Given that training a single Random Forest model is computationally more intensive than XGBoost, the hyperparameter optimization process for Random Forests was more time-consuming. Optuna was run for 6 hours, allowing it to thoroughly explore the hyperparameter space and identify the optimal configuration that minimized the RMSE on the validation set.

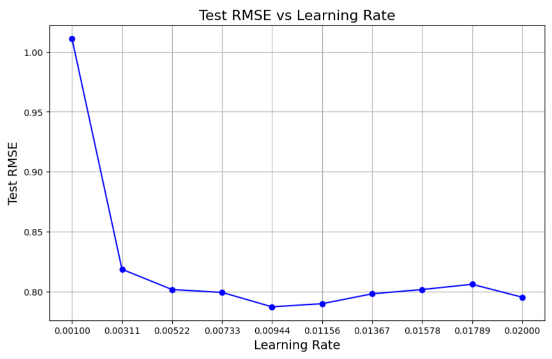
5.3 Advantages of Using Optuna

The hyperparameter optimization was implemented using Optuna's `study.optimize` function, which iteratively evaluates different hyperparameter configurations based on the defined objective function.

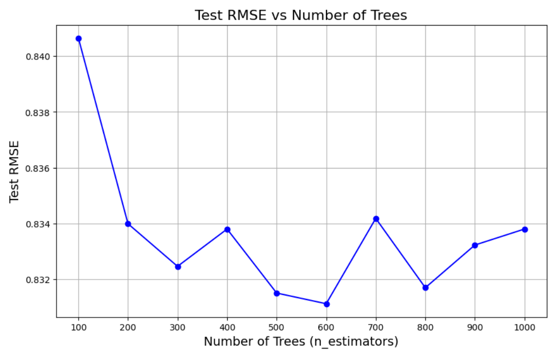
Optuna is highly effective in optimizing hyperparameters due to its ability to handle a vast search space efficiently. Compared to manual hyperparameter tuning or simpler search methods like grid search, Optuna's Bayesian Optimization approach with TPE significantly outperformed in both speed and the quality of the resulting models. By automatically focusing on the most promising regions of the hyperparameter space, Optuna reduced the computational resources required and accelerated the discovery of optimal hyperparameter settings.

5.4 Plotting

To illustrate the impact of hyperparameter variations on model performance, we generated plots showing the relationship between specific hyperparameters and the RMSE on a subset of the training data.



(a) XGB Learning Rate vs. RMSE



(b) RF Number of Trees vs. RMSE

Figure 8: Hyperparameter Tuning Plots

Figure 8a depicts how different learning rates (η) affect the RMSE of the `XGBRegressor`. This plot helped identify the optimal learning rate that minimized the RMSE on the validation set. Similarly, Figure 8b illustrates the effect of varying the number of trees (`n_estimators`) in the `RandomForestRegressor` on the RMSE. This plot helped identify the optimal number of trees that minimized the RMSE on the validation set.

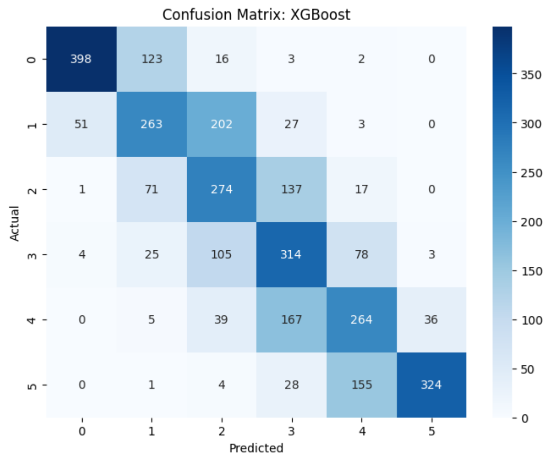
6 Data Splits

To ensure robust model evaluation and prevent overfitting, we split the dataset as follows:

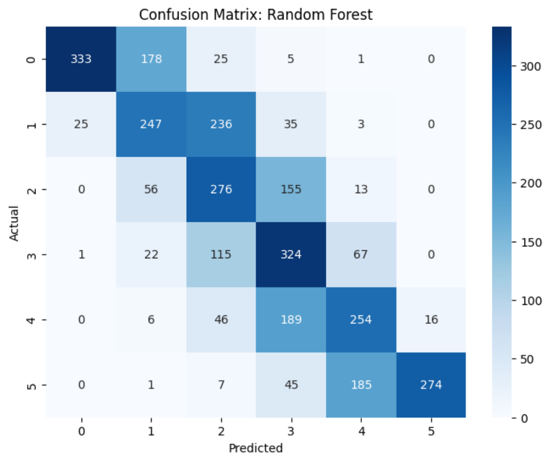
- The original `train.csv` dataset, referred to as the final train set, was deterministically split into an 80% training subset and a 20% validation subset. This allowed us to train the models on a majority of the data while retaining a separate portion for evaluation.
- All preprocessing steps—including missing value imputation, normalization, and neighborhood processing using KNN—were performed exclusively on the training subset to prevent data leakage. These transformations were then applied to the validation subset to maintain consistency across datasets.
- The test set was processed using the preprocessing parameters derived from the final train set, allowing the final model to be trained on all available data.

7 Predictive Performance

We submitted predictions from both the Random Forest and XGBoost models to Kaggle. The performance of these models was evaluated using the RMSE metric.



(a) Best XGB Confusion Matrix



(b) Best RF Confusion Matrix

Figure 9: Confusion Matrices for Best Models

Random Forest Performance:

- Train RMSE: 0.1056
- Test RMSE: 0.8353

XGBoost Performance:

- Train RMSE: 0.2268
- Test RMSE: 0.7917

7.1 Analysis of Performance

Although both the Random Forest and XGBoost models exhibited a noticeable gap between training and testing RMSEs, suggesting potential overfitting, these specifically tuned versions outperformed other more generalized configurations of the same models in the Kaggle competition. By optimizing hyperparameters, the tuned Random Forest and XGBoost models were able to effectively capture complex relationships and handle a large number of features, resulting in superior predictive accuracy.

8 Interpretable Modeling Approaches

To enhance the interpretability of the models, we employed feature engineering techniques that resulted in easily understandable and logically intuitive features. This approach allows for a clearer understanding of how different variables influence the Airbnb listing prices.

8.1 Interpretable Feature Engineering Pipeline

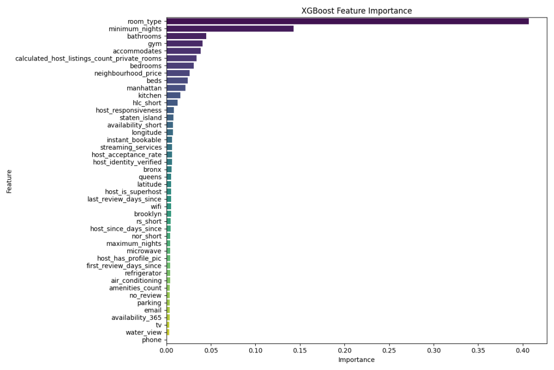
As detailed in the Feature Engineering section, we created new features that encapsulate important aspects of the listings in a manner that is straightforward to interpret. These features were specifically designed to enhance the models' transparency and make the predictive factors easily understandable.

The `amenities_count` feature represents the total number of amenities offered by a listing. This aggregated count is highly interpretable, as it directly reflects the number of desirable amenities provided. By summarizing multiple binary features into a single numerical value, it simplifies the evaluation process for decision trees, allowing for a clearer understanding of how the quantity of amenities influences listing prices without introducing unnecessary complexity.

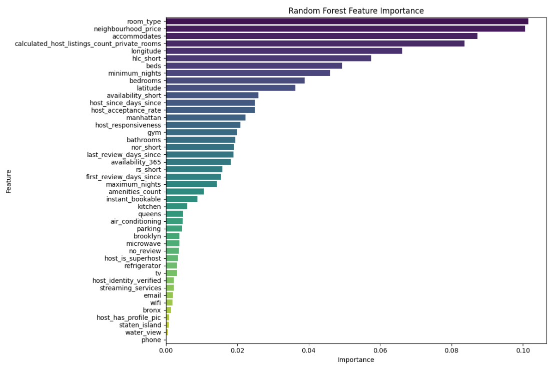
The `host_responsiveness` feature combines the host's response time and response rate into a single metric, providing an easily interpretable measure of the host's engagement level. This composite score effectively captures the overall responsiveness of the host, which is a crucial factor for guests and can logically influence the price. By consolidating these two related aspects into one feature, the model can better assess the impact of host behavior on listing prices, enhancing the interpretability of the models' predictions.

8.2 Feature Importance Analysis

To identify and explain key predictive factors, we performed a feature importance analysis using both the Random Forest and XGBoost models. This analysis reveals which features have the most significant impact on the models' predictions.



(a) XGBoost Feature Importance



(b) Random Forest Feature Importance

Figure 10: Feature Importance Plots

8.2.1 Key Predictive Factors

The feature importance plots in Figure 10 highlight the most influential features in predicting Airbnb listing prices.

The top features identified by the Random Forest model include:

- `room_type` (Importance: 0.1015)
- `neighbourhood_price` (Importance: 0.1006)
- `accommodates` (Importance: 0.0873)

Similarly, the XGBoost model identified the following top features:

- `room_type` (Importance: 0.4067)
- `minimum_nights` (Importance: 0.1430)
- `bathrooms` (Importance: 0.0449)

8.2.2 Observations on Feature Importance

It is notable that the `room_type` feature is highly influential in the XGBoost model, with an importance score of 0.4067. This indicates that the type of accommodation (e.g., entire home, private room) is a key determinant of listing prices, aligning with intuitive expectations as different room types naturally command varying price ranges.

Conversely, engineered features such as `water_view` demonstrated low importance, scoring 0.0005 in the Random Forest model and even lower in XGBoost. Similarly, other amenities like `wifi` and `parking` showed minimal impact, suggesting that while these features enhance the overall appeal of a listing, their direct effect on price prediction is overshadowed by more dominant factors.

8.2.3 Interpretation of Engineered Features

The engineered features, such as `amenities_count` and `host_responsiveness`, also appeared in the feature importance rankings, demonstrating their relevance:

- `amenities_count` had an importance of 0.0107 in the Random Forest model and 0.0041 in the XGBoost model, lower than expected.
- `host_responsiveness` had an importance of 0.0209 in the Random Forest model and 0.0083 in the XGBoost model, a good result, even outperforming other `host` features in some cases.

These importance scores indicate that the number of amenities and host responsiveness are significant predictors of listing prices. Their interpretability makes it easier to understand how improvements in these areas could potentially increase a property's price.

8.2.4 Insights Gained

By analyzing feature importance, we gain valuable insights into what drives the pricing of Airbnb listings:

- **Room Type:** The most critical feature, indicating that the type of accommodation (e.g., entire home, private room) greatly affects the price.
- **Location:** Features like `neighbourhood_price`, `longitude`, and borough indicators (e.g., `manhattan`, `brooklyn`) highlight the impact of location.
- **Capacity and Amenities:** Features such as `accommodates`, `bathrooms`, and `amenities_count` show that larger properties with more amenities command higher prices.
- **Host Factors:** The `host_responsiveness` feature underscores the importance of host engagement in pricing.

9 Conclusion

In conclusion, this project demonstrates the successful application of interpretable machine learning techniques to predict Airbnb rental prices in New York City. By combining robust feature engineering, advanced preprocessing, and hyperparameter optimization using Optuna, we achieved high predictive accuracy, with XGBoost outperforming Random Forest in RMSE on the test set. The analysis also provided clear insights into key drivers of rental pricing, such as room type, neighborhood price indicators, and listing capacity, while engineered features like host responsiveness and amenities count offered added interpretability.

References

[1] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," *CoRR*, vol. abs/1907.10902, 2019. arXiv: `1907.10902`. [Online]. Available: `http://arxiv.org/abs/1907.10902`.

[2] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. DOI: `10.1023/A:1010933404324`. [Online]. Available: `https://doi.org/10.1023/A:1010933404324`.

[3] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2016)*, New York, NY, USA: ACM, 2016, pp. 785–794. DOI: `10.1145/2939672.2939785`. [Online]. Available: `https://doi.org/10.1145/2939672.2939785`.

[4] I. T. Jolliffe, *Principal Component Analysis* (Springer Series in Statistics), 2nd ed. New York, NY, USA: Springer Science+Business Media, 2002, p. 488, Springer Book Archive, ISBN: 978-0-387-95442-4. DOI: `10.1007/b98835`. [Online]. Available: `https://doi.org/10.1007/b98835`.