TABLE OF CONTENTS

Executive Summary:

Motivated by the pricing prediction problem for Airbnb hosts, we implemented a project to build a dynamic machine-learning tool to support decision-making. This project involves preprocessing and data cleaning, exploratory data analysis, feature selection, model fitting and comparison, as well as recommendations and business insights. The different algorithms applied and assessed for model-building include multiple linear regression, cross-validation, Lasso regression, CART (classification and regression tree), and Random Forest. In all models, it is found that using logged price (instead of price) as the response variable improves the prediction power significantly. Based on two major performance metrics, we picked Random Forest Model as the final model for pricing suggestions. The result is beneficial for both the Airbnb website and potential hosts, as they can utilize this technique to rationally determine an appropriate price in the changing market conditions.

I. Project Background

Airbnb is a well-known online marketplace that shares information on short-term home and apartment rentals. The platform connects hosts who have vacant properties and guests who need a comfortable living place during their travels. Given the various kinds of listings, it is not easy to determine the optimal rental price that hosts want to offer. There are lots of aspects that have an influence, such as location, room type, variety and quality of home facilities, etc. The complex combination of all relevant information needs to be filtered and assessed in order to determine the appropriate price.

Meanwhile, the impact of each property feature varies from listing to listing. Such impact also depends on the demands or interests of coming guests. For example, one listing might charge a higher price due to its unique convenience to a nearby grocery store, compared to another listing with similar conditions. In other cases, some guests might prefer the second listing, as they do not need the extra benefit of the grocery store.

The difficulty of setting a price has become a problem from the hosts' perspective, since a listing with an unreasonably high price has little chance of attracting guests, while one with an unreasonably low price does hurt the rental revenue earned. Especially for new hosts who do not have sufficient experience in practice, they might be discouraged to join the platform when facing this difficulty.

II. Traditional Solutions to the problem

A common way of estimating price is to compare with other listings in the same residential area. For example, hosts can browse through the website based on filtering conditions, finding other listings that share similar characteristics and then referring to their prices. The Airbnb website also provides hosts with general guidelines on how to rent out a space in the marketplace. In addition to this, some third-party dynamic pricing tools are easily found online, such as Host Tools, Beyond, PriceLabs, and so on.

While Airbnb does provide its own solution called Airbnb Smart Pricing, it is arguable that the tool automatically underprices the rentals, which is against the interests of hosts. The setting of minimum and maximum prices also allows little flexibility for customization, so the tool is likely to end up suggesting the same daily price even when market conditions are changing. Thus, hosts may not choose to use the built-in tool in making price adjustments, and they need to switch to other solutions if possible.

III. Project Goal

Given such context, a price prediction mechanism will benefit Airbnb hosts by estimating a suitable range of offering price, which is associated with the features of the property being provided. It also helps the website to consistently attract new hosts to join, thus increasing the total amount and variety of listings for Airbnb guests. A dynamic tool that balances all possible factors and comes up with an accurate estimate of the market price is necessary for Airbnb's business.

This project aims to research different features of listing properties and come up with acceptable price levels for Airbnb hosts. We will dig into existing website data of listing prices and apply different machine learning algorithms to build a model based on analysis. The final model will serve as a

suggestion tool for the Airbnb website to assist potential hosts to set an appropriate price. By dynamically adjusting the feature inputs of the model, hosts are able to obtain a realistic market value for the rental property. Additionally, it also helps them to find out the most important features to focus on when trying to self-improve and provide better living services at a higher price.

IV. Data wrangling: data cleaning and transformation

While Airbnb does not release data publicly, we located a resource called 'Inside Airbnb' that compiles information available for different cities. The listings in our dataset are all located in San Francisco, with data points compiled on December 4th, 2022. The dataset comprises 6789 rows and 75 columns. Each row represents one listing and each column contains the property's corresponding features. We divided the dataset into two buckets: numeric and categorical for ease of data wrangling and exploration, post which we combined two subsets together to obtain the final cleaned-up version.

For the 40 numerical columns, we started with checking for the number of NaN/null values. We addressed null values within different columns in the following ways:

- Dropped columns that have all values as NaN such as 'bathrooms' and 'calendar_updated';
- Dropped all reviews-related columns (except for 'reviews_per_month') that have almost one-third of NaN values (1227-1248), as these columns were also highly correlated;
- Replaced the NaN's with 0's in the 'reviews_per_month' column to indicate no reviews;
- Replaced the NaN's with the mode value for columns with a smaller number of NaN values including 'bedrooms', 'beds', and all combinations of 'minimum/maximum nights', as 0 beds or bedrooms in listings do not make sense in reality.

We also checked for the number of unique values within each column and dropped the 'scrape_id' column that only included 1 value throughout the whole dataset.

Next, we checked for correlations between combinations of columns that seemed to be derived or calculated from each other. We retained only 1-2 columns from each combination: retained 2 base-line

columns ‘minimum_nights’ and ‘maximum_nights’ and dropped the rest 6; retained ‘availability_30’, and ‘availability_365’ and dropped the rest 2; retained ‘host_listings_count’ and dropped the rest 6.

After that, we moved on to the 35 non-numeric columns. We started by dropping the columns that were cleaned-up versions of other columns such as ‘neighbourhood_overview’, ‘host_neighbourhood’, etc., all of which were captured by ‘neighbourhood_cleansed’, which we retained. We also dropped columns that contained the same value for all rows such as ‘last_scraped’ and ‘source’ as well as columns that didn't seem useful for pricing prediction, such as ‘host_name’, ‘ host_url’, etc.

Next, we encoded the categorical variables to fit them into regression models. For ‘host_response_time’, we converted 5 statements to 3 categories. ‘Within a few hours’/ ‘within an hour’ were categorized as “quick responses” represented by 1; ‘within a few days’/ ‘a few days or more’ as “late responses” represented by 2; 1044 NaN entries as “no responses” represented by 3. Similarly, the ‘t’/ ‘f’ values within columns including ‘host_is_superhost’, ‘host_has_profile_pic, ‘host_identity_verified’, 'has_availability' and ‘instant_bookable’ were all converted to 0 or 1 numeric values respectively.

We converted some columns from strings to floats and cleaned them for extra symbols ‘,’ and ‘$’ signs contained within the values such as ‘host_response_rate’, ‘host_acceptance’ and ‘price’.

For the ‘amenities’ column, since we have different amenities available within a property such as ‘hot water kettle’, ‘coffee-maker’ and ‘portable fans’, we converted the information to a numeric by counting the total number of amenities offered by each of the properties, named as ‘amenities_variety’.

Finally, we concatenated the two separate sets into one final dataset called ‘airbnb_concat’ comprising 6789 rows and 29 columns. It is seen that the number of columns has significantly reduced.

V. Exploratory Data Analysis

On plotting the relationship between ‘id’ and ‘price’, we noticed that the scatter points are clustered on the y-axis. We applied log transformation on the price column to scale the data, which helped stretch out the price values and exhibit a better pattern for observation (see the comparison of two plots in Appendix). Hence, we decided to proceed with ‘log_price’ as the outcome variable for later analyses.

We explored the data based on four aspects: (1) Location of the property, (2) Features of the property, (3) Reviews from Airbnb guests, and (4) Booking convenience and host performance.

To have a basic understanding of the common features of host properties, we first looked at the distribution of property features against the corresponding count of listings:

- Bedrooms and bed counts: Most listings have only 1 bedroom available. The majority of listings offer no more than 3~4 bedrooms, while 1-3 beds seem to be the popular trend.
- Room Types: "Entire home/apt" and "Private room" listings are prevalent in San Francisco, indicating that guests prioritize personal space, comprehensive facilities and services.
- Amenities: Most listings have 20~40 different amenities in the room/apartment, while the entire scope ranges from as low as 0 to as high as 80+. The distribution is right-skewed.

Accordingly, we created scatterplots and box plots to explore various aspects vs. log_price:

- Room_type vs. log_price: The highest median price is observed in "Entire home/apt", followed by "Hotel room" type, with "Shared room" and "Private room" having similar median values.
- Bedrooms vs. log_price: Listings with 6 or 8 bedrooms have the highest price, while the price of listings with less than 6 bedrooms increases with the number of bedrooms.
- Number_of_reviews vs. log_price: The fewer reviews a listing has, the wider its price range becomes, with the highest/lowest extremes at both ends. Listings with the fewest reviews tend to demonstrate the most instability in prices.
- Has_availability vs. log_price: Properties with booking availability have a slightly higher median price than those without the availability.
- Instant_bookable vs. log_price: This field seems to have little impact on prices. The median value and price ranges for both groups are similar, with comparable numbers of outliers in both.
- Availability_30/availability_365 vs. log_price: The number of days available within a month or a year doesn't seem to have a strong impact, prices of all groups remain in the same range.
- Host_is_superhost/host_verifications vs. log_price: Same median price, no significant trend.

- Host_response_time vs. log_price: Hosts who don't respond have a higher median price compared to those with quick /late responses, which is against our expectations. Meanwhile, hosts with quick responses have more outliers toward higher prices, which is a positive signal.
- Host_response_rate/host_acceptance_rate vs. log_price: Hosts with the lowest or highest response/acceptance rates seem to have higher price variations than those in the middle range.

VI. Analysis of Machine learning algorithms

To proceed with regression algorithms, we split our dataset "airbnb_concat" into airbnb_x = dependent (x covariates) and airbnb_log_y = independent (y variable, log_price). We further split the two subsets airbnb_x and airbnb_log_y into train and test subsets. Based on these, we then proceeded from the most basic to more complex algorithms for price prediction as follows.

Linear Regression Model:

We first made an attempt with the simplest machine learning model: linear regression algorithm. Using logged price as the response variable, we found the relationship between price outcome and host features in the training dataset. The trained model is then tested on the test set, **which yielded an OOS R-squared value of 0.4079**. Compared with the **R-squared value of 0.4077 in the training set**, our linear model performed exceptionally well regarding new data. The variables representing property features help to explain 40.79% of the variation in the logged listing price. While an R-squared above 0.7 or more is regarded as good enough, our linear model does not seem to have good prediction power based on the general criteria.

The importance of features is ranked from the largest to smallest according to their coefficient values. The top attributes include 'has_availability', 'accommodates', 'room_type_cat', 'bedrooms', and 'instant_bookable'. In our linear regression model, these are the important variables that have the largest impact on pricing. For instance, 'has_availability' (0 or 1) indicates whether a listing is currently available for booking and has the largest influence on price: if available, then higher price.

Other attributes that follow are interpreted in similar ways. The coefficient of 'accommodates' is 1.212, showing that for every additional number of guests accepted, the listing price (after exponentiating) increases by 1.2 dollars on average, keeping all other variables constant. Likewise, we can make an inference on 'room_type_cat' (factor variable; 0 = Private room, 1 = Entire home/apt, 2 = Shared room, 3 = Hotel room) with a coefficient of 1.184, 'bedrooms' (number of bedrooms) with a coefficient of 1.16, and 'instant_bookable' (whether the listing is instantly bookable = 0 or 1) with a coefficient of 1.1.

Cross Validation for feature selection:

Based on the linear regression model, we then conducted k-fold cross-validation using the GridSearchCV function in Python. We first shuffled the data and created a 10-fold CV scheme. We then specified the range of hyperparameters to tune within the range of (1, 24) using input from an RFE (Recursive Feature Elimination) object. Finally, the grid search was performed on a linear model to find out the optimal number of features based on the R-squared evaluation metric. **The optimal subset from feature selection is 22 variables, and the resulting model yields an OOS R-squared of 0.4079 on the validation data.** In the case of Airbnb pricing, it is shown that if a linear regression model is to be chosen, 22 features of host listings should be selected to maximize its predictive power. For better visualization, we drew a plot showing the number of features vs. the value of test/train scores (see Appendix).

Standard Lasso Regression Model:

Lasso regression can help identify the most important features by performing a type of regularization that encourages sparse solutions, by setting the coefficients of the features to zero. This can help eliminate irrelevant features and reduce the complexity of the model, which in turn can lead to better generalization performance and avoid overfitting.

We used GridSearchCV after scaling the data to optimize our hypertuning parameter alpha, which controls the strength of L1 regularization and affects the sparsity of coefficients in the model. We tested values from 0.01 to 1 with a step of 0.01 and used cross-validation to evaluate the performance of the model. Specifically, we calculated the average value of the MSE for each fold and used the negative of

this value since GridSearchCV minimizes the score. **We then fit a lasso regression model on the train and test datasets using the optimum alpha to get an R-squared value of 38.94% (train) and 39.81%(test) and RMSE of 0.62 (train) and 0.58 (test).** This means that the model did not overfit or underfit and generalizes well. Yet the value of 39.81 is quite lower than an expected value of 60% and needs improvement. In the process, there were 15 features that survived the penalty including accommodates (with a coefficient of 0.3) followed by mimum_nights, bedrooms, room_type_cat and host_acceptance_rate in the top 5.

CART Decision Tree Model:

CART decision trees could be a more suitable choice over lasso regression when the relationship between the features is complex, there are significant outliers and interpretability is important. Once again, we used the GridSearchCV function to find the optimal values for the hyperparameters max_depth and min_samples_split. We evaluated the model using cross-validation with 8 folds and a negative mean squared error score. After identifying the optimal values of max_depth as 8 and min_samples_split as 50, we fit the tree on the training and test datasets **to get an R-squared value of 63.73% (train) and 49.34%(test) and RMSE of 0.46 (train) and 0.53 (test).**

The results indicated a potential overfitting problem. However, an R-squared value of 49.34% on the test set is still a reasonably good result, indicating that the model is able to explain a significant portion of the variation in the data. Additionally, the RMSE values are relatively low, which means that the model's predictions are on average close to the true values. Yet we felt that further optimization and refinement may be necessary to improve its generalization performance on unseen data.

Random Forest Model:

We tried a random forest algorithm that combines multiple decision trees to improve prediction accuracy and reduce overfitting. We went for the Bagging method that builds multiple decision trees using random samples of the training data with replacement. Each tree in the forest is trained on a different bootstrap sample, and at each split, a random subset of features is considered for splitting. The final

prediction is then made by aggregating the predictions of all the trees in the forest. In our code, we trained multiple instances of the model, with different numbers of trees (n_estimators), ranging from 20 to 1000 with an interval of 20.

For each model, the Out-of-Bag (OOB) score was calculated using the oob_score parameter. We then obtained the n_estimators value of 940 which gives us the maximum OOB_score, computed as the number of correctly predicted rows from the OOB sample. While the construction of a test set might not be necessary in this case, as random forests protect against overfitting by construction through bootstrapping, we wanted to get an estimate on the model's capability to generalize by testing the final outcome on a completely unseen test set to get an unbiased estimate. We obtained **an R-squared value of 95.59% (train) and 64.43%(test) and RMSE of 0.16 (train) and 0.44 (test).** The model has a high degree of accuracy during the training phase given the extremely high R squared value and low RMSE. However, the model's performance during the test phase is not as good implying that it does not generalize well to new data, and there is some overfitting during the training phase. Yet, the model has the highest R2 compared to all the models we ran so far. The model also provides us with the features essential in price prediction including accommodates, bedrooms, minimum_nights, host_listings_count, beds and amenities_variety as the top 6 variables with coefficients > 0.6.

Summary Table: Model Performance Metrics

Regression	Error Type	Error Value
Linear Regression (Test Set)	R Squared (R^2)	40.79%
Linear Regression (Test Set)	Root Mean Squared Error (RMSE)	0.33
Lasso Regression (Test Set)	R Squared (R^2)	39.81%
Lasso Regression (Test Set)	Root Mean Squared Error (RMSE)	0.58
CART Regressor Decision Tree (Test Set)	R Squared (R^2)	49.34%
CART Regressor Decision Tree (Test Set)	Root Mean Squared Error (RMSE)	0.53
Random Forest Regression (Test Set)	R Squared (R^2)	64.43%
Random Forest Regression (Test Set)	Root Mean Squared Error (RMSE)	0.44

VII. Recommendations

The summary table shows that the Random Forest Regression Model has the highest R^2 value and the second-lowest RMSE value among all models. Additionally, the graph comparing the true values vs. predicted values also demonstrates that many of the points are situated near the diagonal line, verifying that Random Forest gives an accurate result in most cases.

The CART decision tree model has the second-highest R^2 value, which is within our expectation because the Random Forest algorithm usually performs better than single tree predictions by averaging the results of all tree samples. It is also noted that the R^2 values of linear regression and Lasso regression models are very close to each other, which is probably due to the similarity between the two algorithms, as Lasso regression functions as an upgraded version of linear regression by selecting only a subset of the variables.

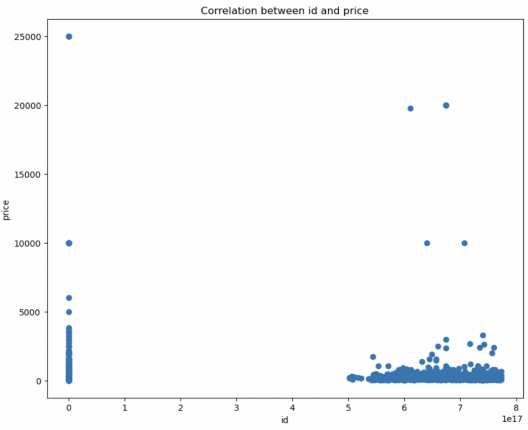
We choose Random Forest Model as our final price prediction model for Airbnb host listings. The higher the R^2 is, and the lower the RMSE is, the better an ML model is likely to perform. Among all models, RF shows the best balancing of both evaluation metrics and thus is suggested to have the highest predictive power. In practice, Airbnb hosts can utilize our Random Forest Model to assist their pricing decision-making, which is calculated from various feature inputs of the properties they own. They can also explore the most important features as provided by the algorithm, focusing on certain aspects to make improvements and thus providing better services that naturally charge a higher rental price.

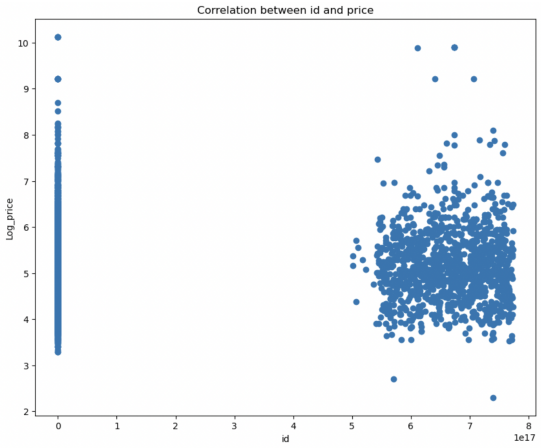
VIII. Conclusion

This project made an attempt to find the optimal machine learning algorithm that provides pricing prediction to facilitate Airbnb hosts' decision-making. Among all candidates: linear regression, standard Lasso regression, CART decision tree, and Random Forest, we finally select Random Forest Model with the best overall OOS performance on the test dataset. Combining this dynamic technique with traditional solutions to the problem can assist hosts in estimating a realistic market value of rental properties.

Appendix

Exploratory Data Analysis: Comparison of plotting Price vs. Log_price on the y-axis

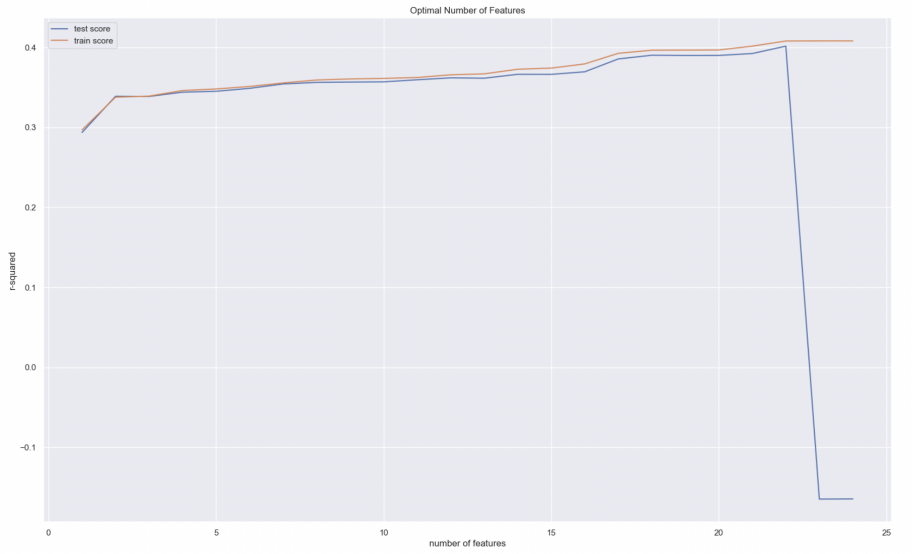




Linear Regression: Feature Importance Ranking

	Attribute	Importance
19	has_availability	1.278810
1	accommodates	1.211872
22	room_type_cat	1.184005
2	bedrooms	1.160115
20	instant_bookable	1.100709
18	host_identity_verified	1.099046
15	host_is_superhost	1.093062
16	host_verifications	1.021991
21	amenities_variety	1.004673
6	availability_30	1.002450
9	number_of_reviews_ltm	1.000484
0	host_listings_count	1.000127
5	maximum_nights	1.000000
8	number_of_reviews	0.999960
7	availability_365	0.999849
13	host_response_rate	0.998437
4	minimum_nights	0.997969
12	host_response_time	0.996871
14	host_acceptance_rate	0.996827
23	neighbourhood_cleansed_cat	0.996006
10	number_of_reviews_l30d	0.989119
11	reviews_per_month	0.974630
17	host_has_profile_pic	0.941009
3	beds	0.919380

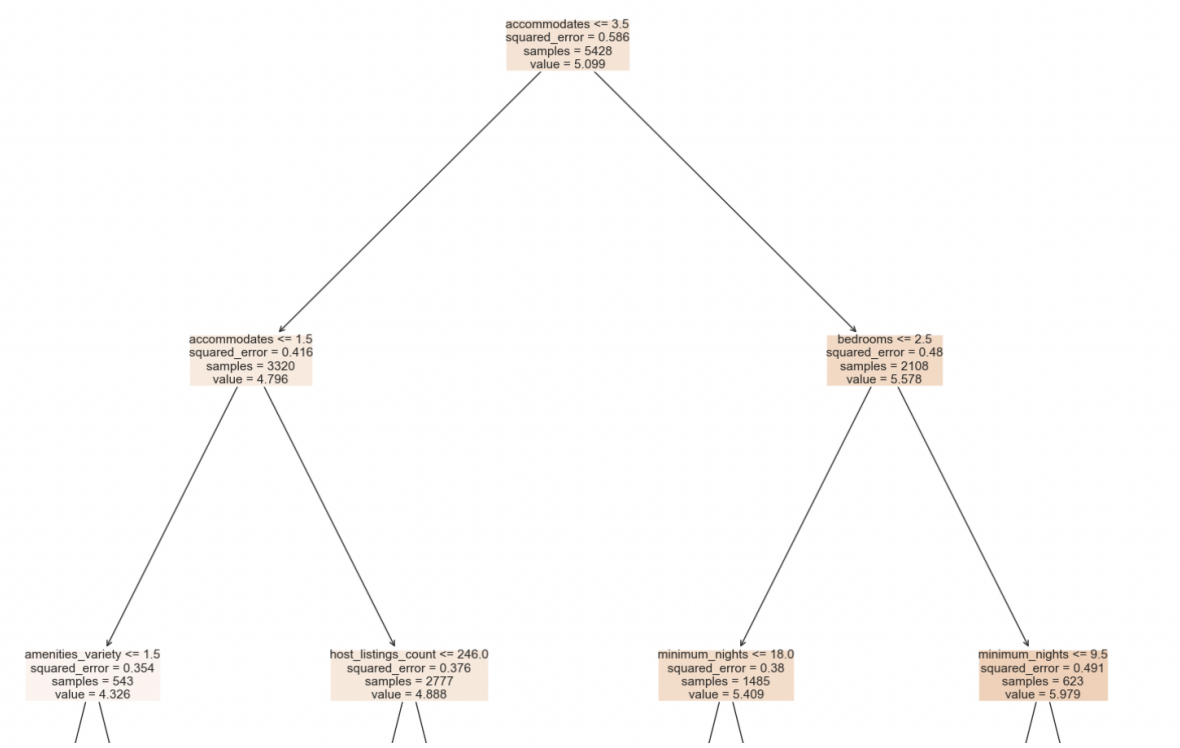
Cross Validation: Number of features selected vs. R-squared value in train/test set



Lasso Regression: Feature Importance Ranking

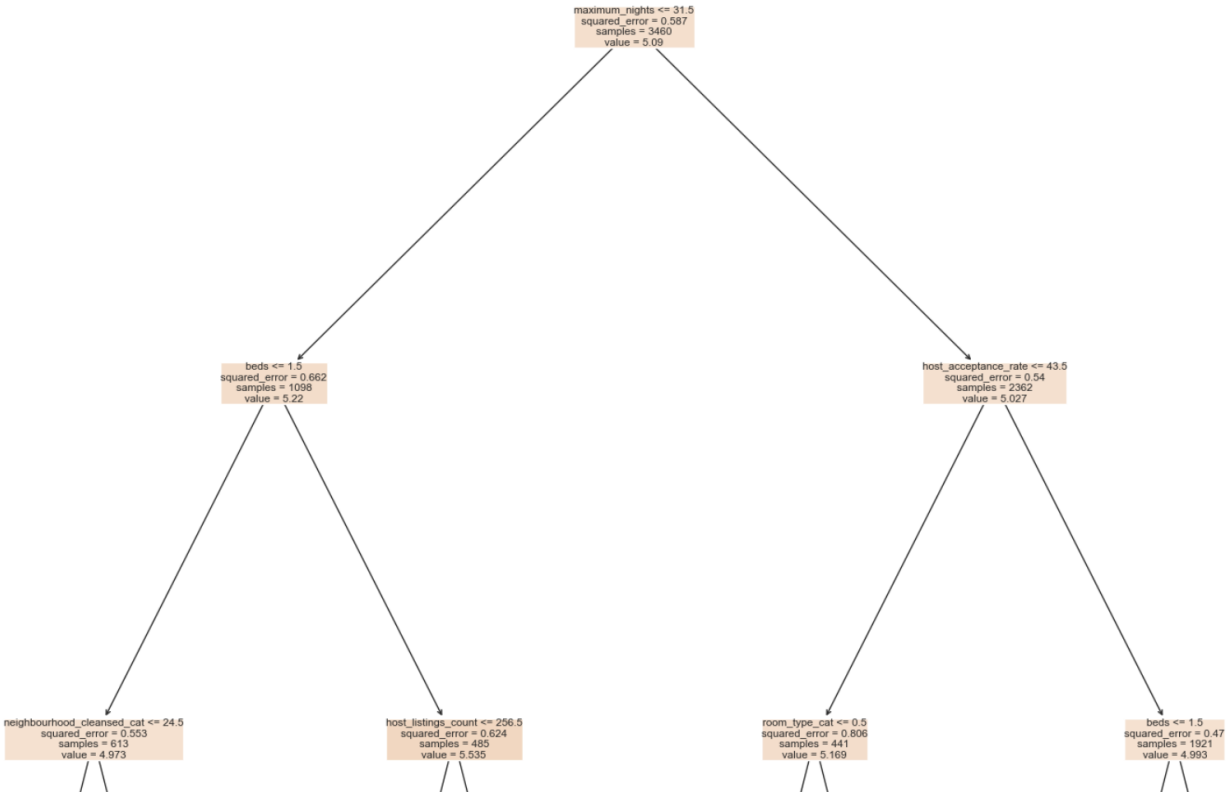
	Attribute	Importance
1	accommodates	0.317640
4	minimum_nights	0.083014
2	bedrooms	0.077273
22	room_type_cat	0.076212
14	host_acceptance_rate	0.059707
0	host_listings_count	0.055049
21	amenities_variety	0.046203
13	host_response_rate	0.035531
11	reviews_per_month	0.027760
19	has_availability	0.014722
23	neighbourhood_cleansed_cat	0.012917
20	instant_bookable	0.007937
18	host_identity_verified	0.007878
10	number_of_reviews_l30d	0.007816
15	host_is_superhost	0.001211
9	number_of_reviews_ltm	0.000000
8	number_of_reviews	0.000000
7	availability_365	0.000000
16	host_verifications	0.000000
17	host_has_profile_pic	0.000000
6	availability_30	0.000000
5	maximum_nights	0.000000
3	beds	0.000000
12	host_response_time	0.000000

CART Decision Tree: Visualization of decision nodes



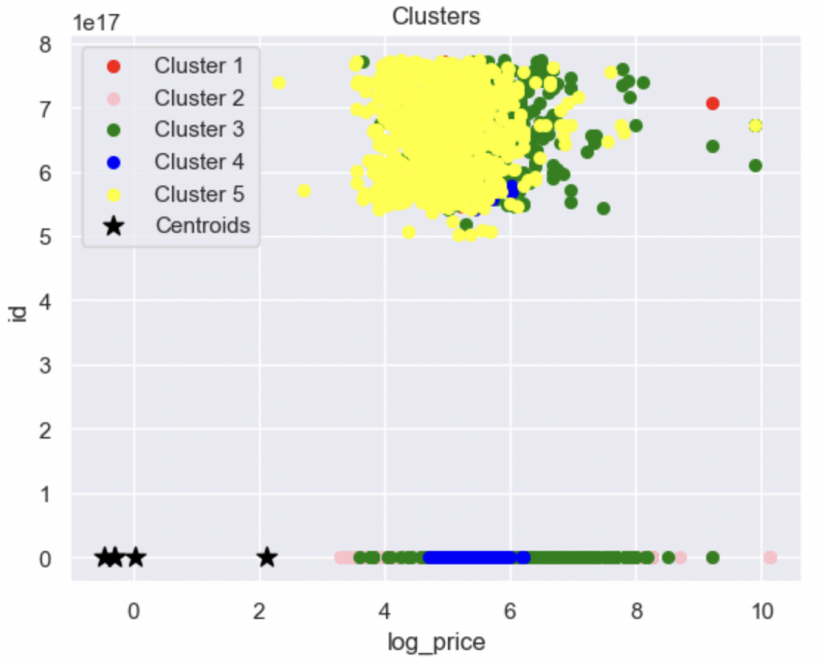
The tree produced by the CART technique splits on 'accommodates' as the first node. If ‘accommodate’ <= 3.5, we traverse to the left, and if > 3.5 we traverse to the right. At the second level, on the left we further split on 'accommodates', while on the right we use the node 'bedrooms' to split. As we keep traversing the tree, the end nodes will have the average predicted price for all the samples that satisfy the condition leading to that node. For example, the average predicted logged price is expected to be 5.578 for properties with ‘accommodate’ (number of guests allowed) > 3.5 and ‘bedrooms’ (number of bedrooms in the property) <= 2.5.

Random Forest: Visualization of the finalized tree



The tree through random forest bagging is split on 'maximum_nights' as the first node. If ‘maximum_nights’ is <= 31.5, we traverse to the left, and if > 31.5 we traverse to the right. At the second level, on the left we further split on 'beds', while on the right we use the node 'host_acceptance_rate' to split. As we keep traversing the tree, the end nodes will have the average predicted price for all the samples that satisfy the condition leading to that node. For example, the average predicted logged price will be 5.22 for properties with ‘maximum_nights’ > 31.5 and ‘beds’ <= 1.5.

K-means Clustering: Price Distribution of Hosts in 5 different clusters



We attempted K-means clustering in order to identify relationships and patterns between different features and group listings into different clusters based on their similarities in an unsupervised format. We performed k-means clustering on the top columns that were commonly identified under ‘feature selection’ through the 4 models including "id", "log_price", "accommodates", "bedrooms", "minimum_nights", "host_listings_count", "beds", and "amenities_variety".

We were hoping to identify patterns and relationships between different features that we hadn't considered. For example, discovering that listings with a high number of bedrooms and high host listings count tend to have higher prices. Or, discovering that listings with a high variety of amenities tend to have a high number of beds and accommodates.

We visualized the clusters and centroids generated by the k-means algorithm to see how the different clusters are distributed in the "log_price" and "id" space. Yet, we didn't find any discernible patterns in the plot and saw that the clusters overlapped with each other, as well as the centroid overlapped. We would need further analysis in this area and hence didn't include it as a part of our analysis in the main text.