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.

1. **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 (see below). 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

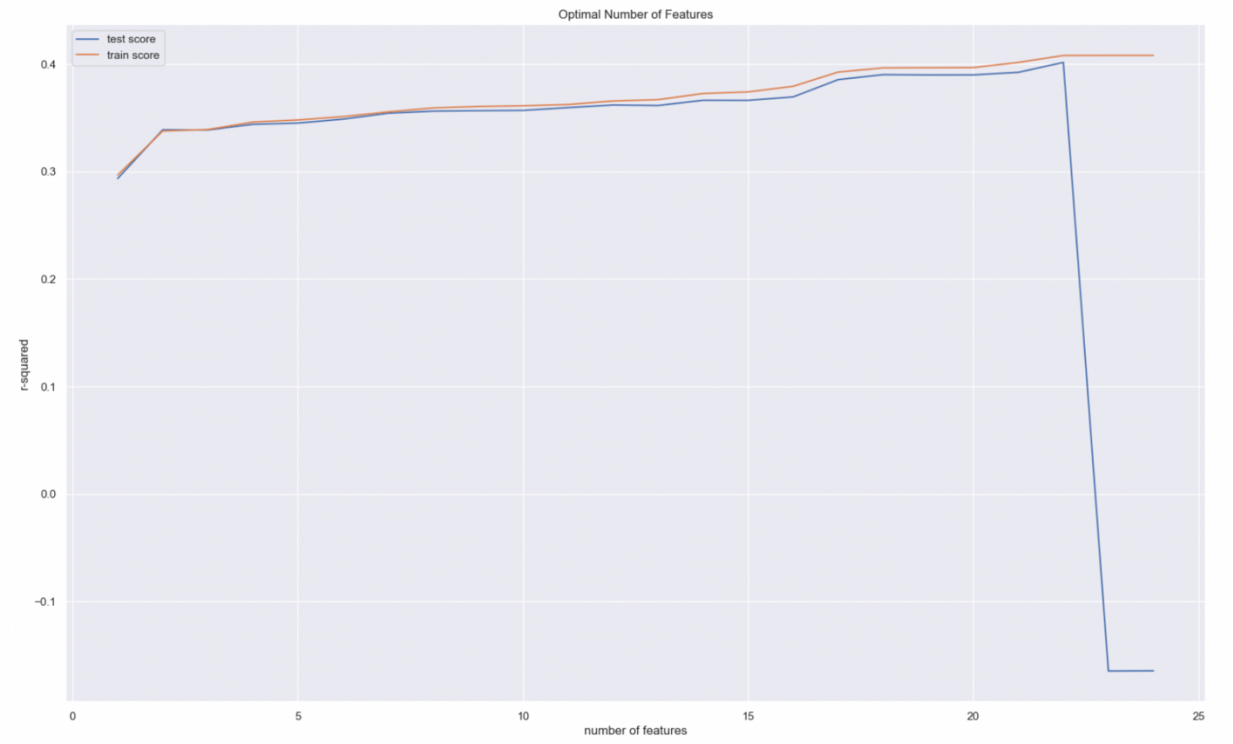
can make an inference on ‘room_type_cat’ (factor variable; 0 = Private room, 1 = Entire home/apt, 2 = Shared room, 3 = Hotel room) with a coefficient of 1.184, ‘bedrooms’ (number of bedrooms) with a coefficient of 1.16, and ‘instant_bookable’ (whether the listing is instantly bookable = 0 or 1) with a coefficient of 1.1.

	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

2. 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 below).



3. 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 7this 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 (see below).

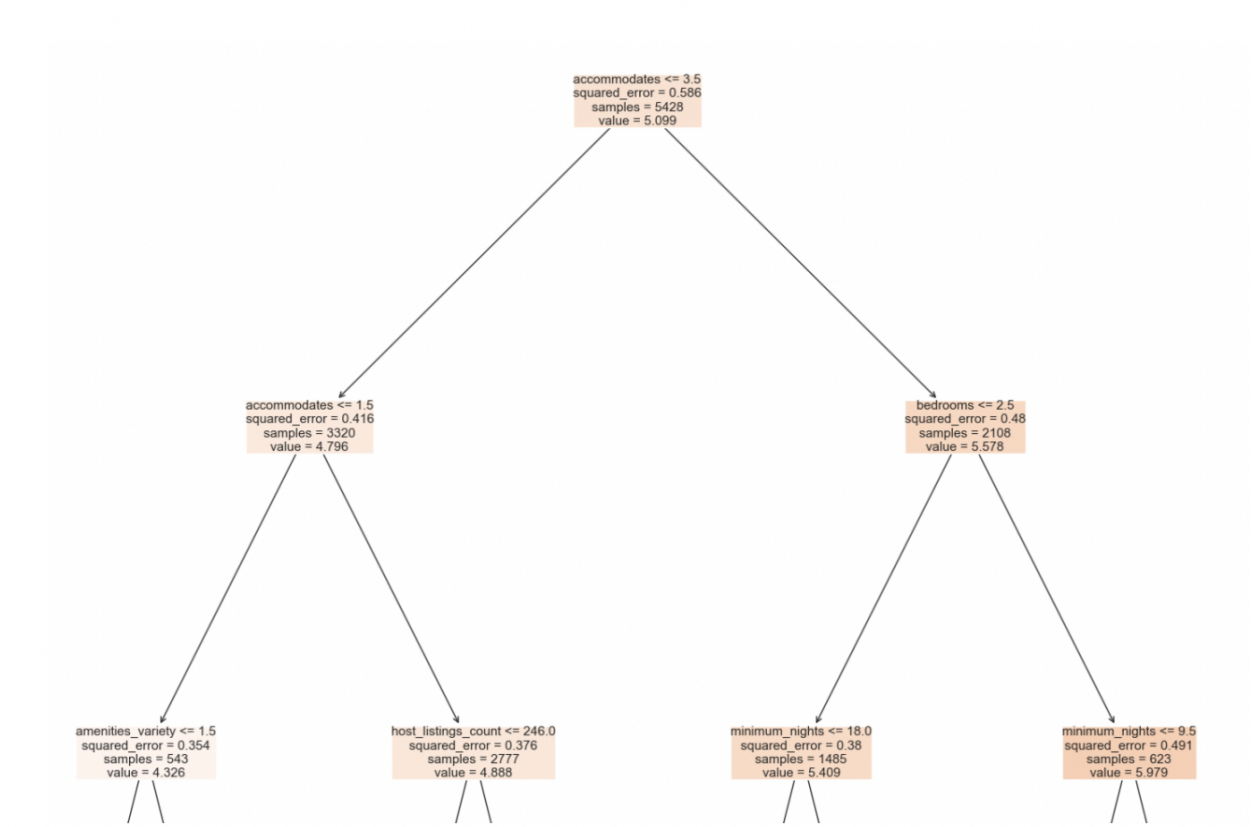
	Attribute	Importance
1	accommodates	0.317640
4	minimum_nights	0.083014
2	bedrooms	0.077273
22	room_type_cat	0.076212
14	host_acceptance_rate	0.059707
0	host_listings_count	0.055049
21	amenities_variety	0.046203
13	host_response_rate	0.035531
11	reviews_per_month	0.027760
19	has_availability	0.014722
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

4. **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.

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. (See next page for visualization of CART decision nodes)



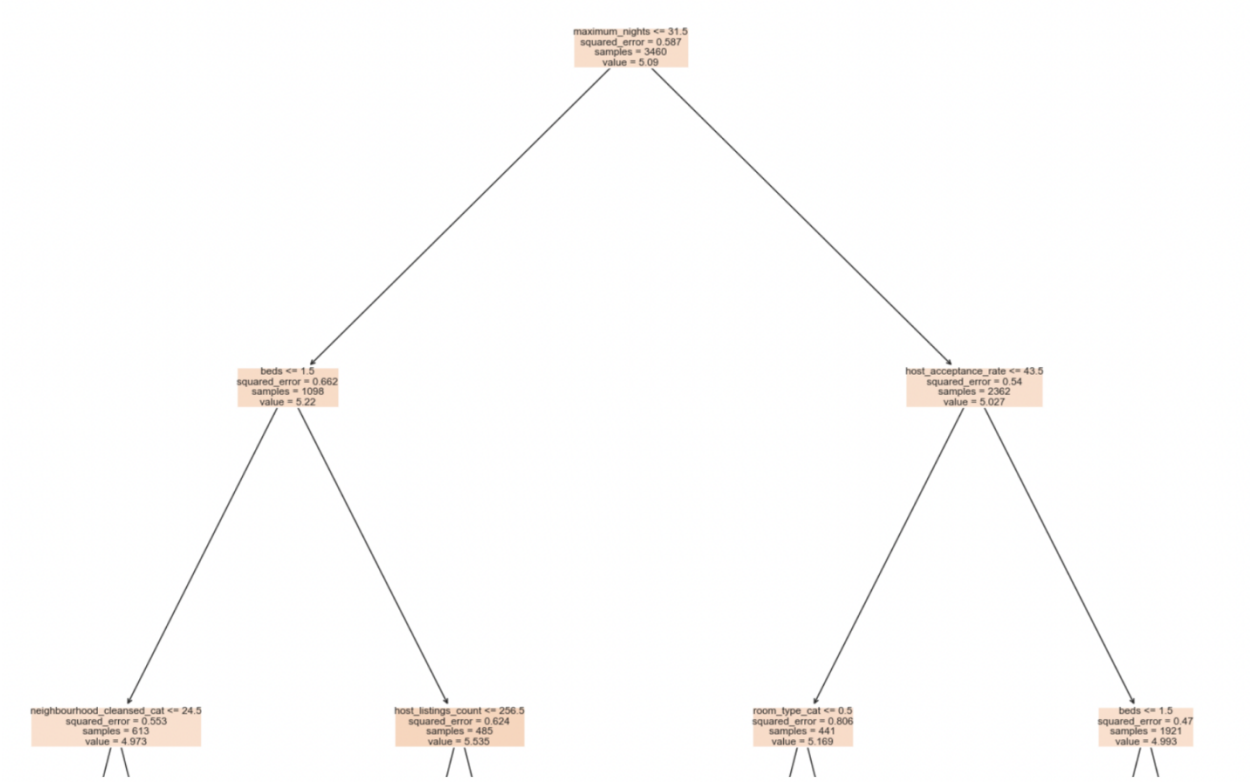
5. Random Forest Model:

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.

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. (See next page for visualization of the finalized RF decision tree)



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 (see below). 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 to consolidate our findings.