Predicting Airbnb Prices

Logan O'Brien
lrobrien@wisc.edu

Sam Peterson
sjpeterson6@wisc.edu

Dylan Asoh
dasoh@wisc.edu

Abstract

Several studies have tried to predict the price of a rental property listing on Airbnb using various forecasting models; however, predicting a price that is accurate is difficult due to each rental property having distinct features that may influence its price. We collected 74,111 data points from Kaggle for listings in major cities (New York City, Los Angeles, San Francisco, Washington D.C., and Boston) to be used for predicting the price of rental properties. In this project we used various machine learning techniques to forecast an accurate price of an Airbnb rental property. Linear Regression was used as a baseline model and then its results were compared to Ridge Regression, Lasso Regression, Elastic Net Regression, XGBoost, LightGBM, Random Forests, Bagging, Support Vector Machines, and k-Nearest Neighbors. In order for all our features we selected to be used by these models, we experimented using ordinal encoding and one hot encoding on the categorical features we selected for predicting the price of an Airbnb rental property. With one hot encoding on the categorical features and LightGBM as our model, we were able to achieve an accuracy of 61.6%.

1. Introduction

The ways in which people find short term rental properties have changed dramatically in recent years with the development of apps that give people the ability to rent properties from their phone. While there are a wide variety of housing options in major cities across the United States, such as motels, hotels, camp sites, apartments, and houses, short term rentals through Airbnb are becoming ever more popular. Short term rentals have been on the rise since the market introduction of VRBOs in 1995. With the global industry valued at $169 billion, the success of short term rentals can in part be attributed to the 2008 real estate startup Airbnb [1]. The comfort and privacy is unmatched in most temporary housing options because renters feel "right at home". However, renters can also find high quality houses and apartments to rent through Airbnb for much cheaper than a stay at a high quality hotel.

In light of the ongoing COVID-19 pandemic that has swept the world there are growing inquiries to rent short term from other property owners. Due to these recent events, it has become pertinent to ensure an equitable renting experience for renters, listers, and Airbnb themselves. Through the use of machine learning, we want to ensure renters are being given a fair price for the properties they are interested in. Listers will be given a good estimate to price their property at, and Airbnb can advertise their platform as the best place for anyone to participate in an equitable renting process.

We are in part motivated to examine these short term rental properties due to their resilience throughout the pandemic. Short term rental properties like the ones listed on Airbnb reported higher average daily rates in the US of July of 2020 (five months into the pandemic) than a year prior. When compared to the hotel industry, the short-term rental industry remained stable in average daily rates with revenue per available room (RevPAR) only being down 4.5%, with hotels experiencing a 64.8% drop in RevPAR [5].

In this project, we focus on developing machine learning models to enable property owners and vacation goers alike to determine fair prices for their short term rental properties based on a variety of variables related to the rental property. To do this we conduct analysis with Linear Regression, L2 regularization (Ridge Regression), L1 regularization (Lasso Regression), and a combination of L1 and L2 regularization (Elastic Net Regression), Gradient Boosting (XGBoost, LightGBM), k-Nearest Neighbors, Random Forests, Linear Support Vector Machines and an ensemble of machine learning models all in an effort to predict the price of short term rental properties.

2. Related Work

The ability to accurately estimate housing and rental prices has been a topic of interest for decades. In recent years, machine learning models have made this process more accessible to the general public. In fact, there exist entire businesses who specialize in home and rental property pricing such as CoreLogic or PriceLabs.

Researchers continue to attempt to improve current models. Laura Lewis, Data Scientist at Amazon and Oxford

PhD, conducted a similar analysis to our own on Airbnb data in an effort to maximize a host's potential rental income using XGBoost and neural networks. Using vanilla XGBoost as a baseline model, Lewis found that complex neural networks usually performed worse than the baseline XGBoost or they were too computationally expensive to be practical [3].

Further analysis on price prediction for rental properties conducted by Fei, Yue at the University of California Los Angeles found similar results. Yue compared machine learning methods such as Ridge Regression, Random Forest, Support Vector Machine Regression, and XGBoost. XGBoost continued to be the best model in respects to RMSE, MAE, R2 [6].

3. Proposed Method

3.1. Problem Overview

To provide a baseline for our reasoning and selection of methods, it is important to understand the problem we aim to address. Our project will focus on predicting the price of an Airbnb listing, giving various features relating to the listing and listers. We are tasked with exploring the features of our dataset, preparing the data for analysis, performing various methods of regression, and evaluating the success at each interval.

We will start with simple Linear Regression and Gradient Boosting models following our data cleaning, to evaluate if our attempt to encode and select features was particularly successful compared to the raw data. From there, we will implement a diverse selection of regression methods and modeling strategies, tuning each as necessary using GridSearchCV or other methods.

3.2. Linear Regression

The baseline machine learning model that we used was Linear Regression. We decided to use Linear Regression as our baseline model because it is easy to interpret due to it's mathematical simplicity, and has been widely known to perform well for predicting continuous values such as the price of an Airbnb rental property. Mathematically, Linear Regression minimizes the residual sum of squares between the features and the target variable after fitting a straight line to the data. The high bias of Linear Regression makes for an easy to understand model, however a complex problem such as this may require a more complex model in turn.

3.2.1 Ridge Regression

Due to the fact that we used a large number of features, we thought it might be useful to add a regularization term to our Linear Regression model's loss function to prevent over fitting the data. Therefore, we used Ridge Regression to do this. Ridge Regression, also known as Linear Regression with L2 regularization, aims to reduce the coefficients of each feature close to zero, but not exactly zero by adding a regularization term to the loss function that is minimized during training. The regularization term is the summation of the weights of each feature. Furthermore, the strength of the regularization term can be adjusted by multiplying it by a value called alpha.

3.2.2 Lasso Regression

We also wanted to experiment reducing the coefficients of each feature to zero. To do this we used Lasso Regression. Lasso Regression, also known as Linear Regression with L1 regularization, aims to reduce the coefficients of each feature to zero by adding a regularization term to the loss function that is minimized during training. The regularization term is the absolute value of the summation of the weights of each feature. Furthermore, the strength of the regularization term can be adjusted by multiplying it by a value called alpha.

3.2.3 Elastic Net Regression

Furthermore, we thought it might also be useful to experiment doing adding a mixture of L1 regularization and L2 regularization in our Linear Regression model's loss function. To do this we used Elastic Net Regression. Elastic Net Regression, also known as Linear Regression with a mixture of L1 regularization and L2 regularization, adds the regularization terms used in L1 regularization and L2 regularization. The regularization terms are multiplied by a known as alpha that is used to control the strength of L1 and L2 regularization used in the model. Furthermore, the regularization terms are multiplied by another hyper parameter called the "l1_ratio" that adjusts the ratio of L1 and L2 regularization used in the model.

3.3. Gradient Boosting

3.3.1 XGBoost

XGBoost is an optimized gradient boosting library that was designed to be highly-efficient, flexible and portable. XGBoost algorithms have become one of the most widely used machine learning algorithms because of how well they are able to solve machine learning problems in a fast and highly accurate way. We used the XGBoost regression algorithm for our use case. The XGBoost model is an ensemble of decision trees where each decision tree, or "weak" learner, learns from the previous decision tree's residuals to fix errors of the previously trained trees. For example, the first tree is trained on the training set and the training set residual error is calculated from the predicted values. The next predictor uses the residual error of the previous trees as labels and predicts a new set of residuals for the next tree to learn from. Trees are added to the ensemble until there are no performance improvements that can be made to the model and a ”strong” learner is created. The lower bias of the decision tree algorithms would suggest that the model will make less assumptions about the target function, however with that comes a higher variance. To account for this variance, we must ensure our gradient boosting models are tuned properly, so the model remains consistent. This is an example of the Bias-Variance Trade-off, which discusses finding a balance between bias and variance, minimizing both to create a model that is neither overfitted nor underfitted.

3.3.2 LightGBM

LightGBM(Light Gradient Boosting Machine), similar to XGBoost, is a high-performance gradient boosting framework based on decision tree algorithm [2]. LightGBM implements decision trees in a slightly different method than XGBoost, splitting the trees leaf-wise rather than level-wise. The leaf-wise growth algorithm is surprisingly fast, and more effective at reducing loss through each split, often resulting in a more accurate model. While the leaf-wise split creates more complex decision trees and can result in more accurate models, it is important we toggle the max depth of the trees to avoid overfitting the model. The training speed of LightGBM is great for evaluation, as it allows you to tune hyperparameters and evaluate various strategies much more efficiently, even with our large dataset.

3.4. k-Nearest Neighbors

k-Nearest Neighbors (kNN) is an supervised machine learning algorithm that stores the training examples during the training step of the algorithm. When the kNN model makes prediction on a query point, the k nearest neighbors of a query point are located which leads to the prediction of the query point based on the most similar points surrounding it. In the context of regression, the k-Nearest Neighbors algorithm predicts the query point as the average of its k nearest neighbors. The k number of similar points to the query point are the number of points from 1 to k with the smallest Euclidean Distance to the query point. The Euclidean Distance between two points is the square root of the summation of the difference squared between each data points features.

3.5. Bagging

Bagging is a simple and powerful ensemble method that runs a machine learning algorithm on random subsets, bootstraps, of the complete dataset and aggregates each individual prediction to return a final prediction. We are using the BaggingRegressor in our analysis. BaggingRegressor creates a decision tree for each random subset drawn. The predictions from each decision tree are then combined, averaged, and usually form a more robust, accurate prediction than a single Linear Regression model alone. Typically, this process is effective because each decision tree is fitted to a slightly different training subset, in turn providing minor differences that can help reduce variance in the overall prediction.

3.6. Model Stacking

Stacking is a special case of ensemble methods that we experimented with to predict the price of Airbnb listings. Model stacking ensembles models that are called ”base learners” together that learn from the initial training set. Then the resulting models that are learned from the training set make predictions that serve as the input features to a ”meta learner.” Model stacking algorithms are prone to extensive overfitting so it is important to use cross validation with stacking to avoid overfitting. Luckily, the sklearn package implements cross validation into the stacking algorithm for us.

4. Experiments

4.1. Dataset

The primary dataset used in this project is obtained from Kaggle. It features 27 housing indicators that were used to predict the price of Airbnb listings in major US cities. Many of the fields focus on physical features of the listing (amenities, property type, city, etc.), while others are drawn from the user profile of the listing owner(number of reviews, profile picture, last review date, etc.). The dataset provides an initial 74,100 class of prices. The prices were initially preprocessed into a logarithmic format.

It is important to note that this data was generated specifically for a case competition and is not data directly from Airbnb. Quality data from rental properties is usually locked behind paywalls from websites like AirDNA. However, this data still serves as a good proof of concept.

4.1.1 Data Preparation

Starting with a very large dataset that included missing values, numerous categorical features, and several irrelevant features, we had decisions to make regarding how to approach the preparation of our data. We began with simple exploratory analysis to get a better understanding of the data from surface level, searching for correlations such as in Figure 1. The numerical data in our dataset included some missing values, which we handled by dropping or by inputting the median value for that feature.

Since our dataset included a large amount of columns, we needed to remove some meaningless columns to reduce the noise of our data before performing our regression anal-

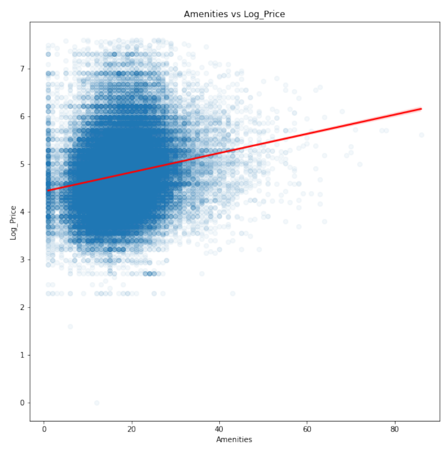

Figure 1. Exploratory Data Analysis- correlation between total number of amenities and the listing price.

ysis. Many of these columns were obvious noise, such as if the lister's profile contained a profile picture, but to ensure we were making informed decisions we examined the correlation matrix and feature importance of our data as well. We decided to approach the cleaning and selection of our categorical features in two ways, eventually leading to two datasets composed of ordinal data transformations or binary transformations via one-hot encoding. Through simple regression analysis, we determined that our dataset composed of numerical and one-hot encoded features was most effective for modeling moving forward.

4.2. Training Models

Following the processing of our data, we split each of our cleaned datasets into training and testing subsets. Using train_test_split, we decided on using 80% of our data for training and holding 20% for testing purposes, and used a random seed of 0.

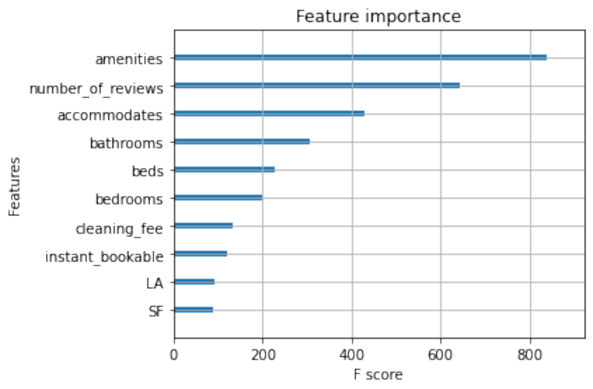

Figure 2. XGBoost Regression feature importance.

Majority of our analysis was done using the complete cleaned dataset, however we tried several different partitions of the data in modeling. For example, following the implementation of a XGBoost Regressor on our complete data set, we created a subset of our cleaned dataset based of the model's feature importance scores, as seen in Figure 2. Many of the models we ran included hyperparameters that we needed to tune to ensure we were optimizing the use of such model. We used GridSearchCV cross-validation to select the best performing model from a grid of hyperparameters.

4.3. Software/Hardware

The bulk of our project was performed on our personal laptops using Jupyter Notebooks. We implemented various python libraries, including NumPy, Pandas, Seaborn, and Scikit-Learn to handle cleaning, modeling, and visualizations. We also implemented a shared Google Colab for simple code sharing, as well as a shared Google Doc to record our thoughts, plans, and results throughout the process.

5. Results and Discussion

Method	Accuracy
Linear Regression	0.5506
XGBoost	0.6142
LightGBM	0.6164
AdaBoost	0.5020
Random Forests	0.6030
Ridge	0.5506
Lasso	0.5506
ElasticNet	0.3853
Bagging	0.5587
LinearSVR	0.5422
KNN	0.5525
Model Stacking	0.6104

Table 1. Best Testing Accuracy Achieved for Each Model

Table 1 shows a summary of the results we achieved for each method, when fitted with the same training data and tested with the same testing data subsets.

5.1. Linear Regression

Serving as a baseline machine learning model, we performed no tuning to attempt to create a more accurate result. We did, however, incorporate cross validation into our process to better estimate the model's ability to fit to new data. Our Linear Regression model achieved an accuracy of 55.06%. Surprisingly, there wasn't strong evidence of over fitting with this model, which we expected because of the large quantity of features in our dataset. We thought it might be useful to examine how the predicted values of this model compared to the true values of the data, as you can see in Figure 3.

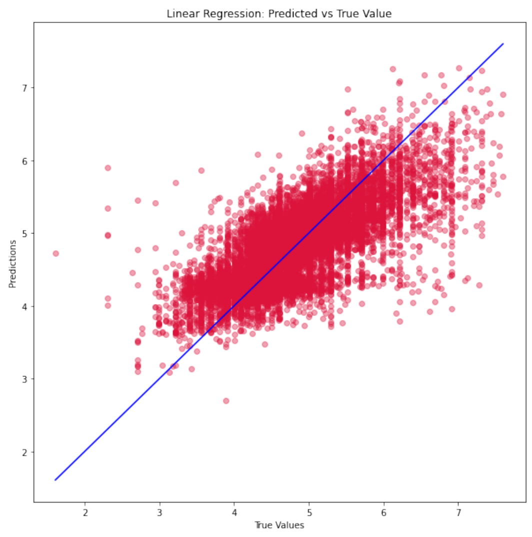

Figure 3. Linear Regression- Predicted vs True Values.

In an effective and accurate model, the points representing would be very close to the null model, or the blue linear trend line we see in Figure 3. There is a lot of noise, and some large residuals in this plot, however, indicating that this base Linear Regression model was a fairly poorly fitted model. Its important to note that at higher dimensions, Linear Regression may struggle to learn the more complex relationships between all features. A large number of mostly binary features may not be best suited for a simple Linear Regression model, but for the purpose of serving as a base, the results are expected.

5.1.1 Ridge Regression

Although Linear Regression we learned didn't suffer from much over fitting, we still wanted to see whether using some sort of regularization would benefit our Linear Regression model. We experimented using multiple values (0, 0.25, 0.5, 0.75, 1.0, 2.0, 3.0, 4.0, 5.0) for alpha with Grid Search Cross Validation. The best value for alpha was zero, indicating that when no regularization is used the model performs best. Therefore, the Ridge Regression model with alpha of zero achieved the same accuracy as Linear Regression.

5.1.2 Lasso Regression

Although we knew that Linear Regression wasn't performing well with any sort of regularization, we had time to experiment with adding L1 regularization. We experimented using multiple values (0, 0.25, 0.5, 0.75, 1.0, 2.0, 3.0, 4.0, 5.0) for alpha with Grid Search Cross Validation. Again, the best value for alpha was zero. Therefore, the Lasso Regression model with alpha of zero (no regularization) achieved the same accuracy as Linear Regression.

5.1.3 Elastic Net Regression

To ensure we tested the most common types of Linear Regression we also tested Elastic Net Regression despite knowing that adding regularization was helping Linear Regression perform better. We experimented using multiple values (0, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0) for alpha with Grid Search Cross Validation. Like we expected, the best value for alpha was zero and we can conclude no regularization is needed for Linear Regression to perform its best to predict the price of Airbnb listings.

5.2. XGBoost

To improve upon our Linear Regression model, we tried several attempts at gradient boosting, allowing us more flexibility in our model to tune hyperparameters. Starting with a simple XGBoost, we already obtained a better accuracy than the Linear Regression model, with 61.42%. From here, we used GridSearchCV to determine the best hyperparameter max_depth to input for XGBoost. Max_depth refers to the limit set on the number of nodes allowed in each decision tree. We determined that the most accurate model used max_depth = 3, as seen in Figure 4.

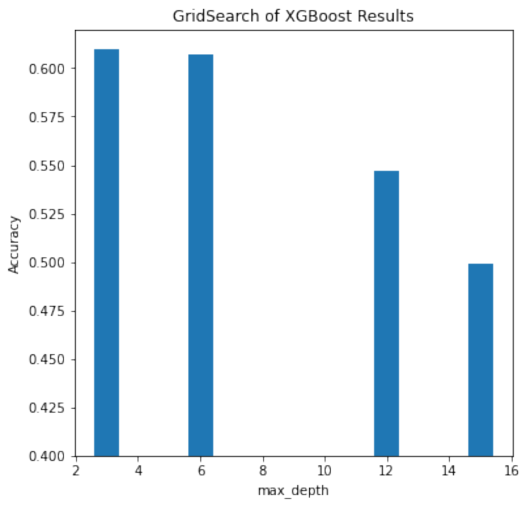

Figure 4. Grid Search result for XGBoost max_depth.

Through our Grid Search, however, we were unable to improve the performance of the model. It is likely we did not have a wide enough grid to search. We were limited by the performance of our machines and Jupyter notebooks, and hyperparameter tuning for XGBoost, along with several other models we worked with, can be very computationally expensive.

5.3. LightGBM

As discussed previously, LightGBM is very similar to XGBoost, merely implementing a different growth algorithm(leaf-wise rather than level-wise). Based on this, we felt we could develop a LightGBM model to slightly outperform our XGBoost model. Since LightGBM is much more efficient than XGBoost, it allowed us to experiment further with hyperparameter tuning, passing larger parameter grids to GridSearchCV and optimizing our model much better than we had previously.

max_depth	num_leaves	accuracy
23	75	0.614856
23	100	0.613824
23	125	0.611881
24	75	0.614853
24	100	0.613934
24	125	0.612074
25	75	0.614922
25	100	0.613695
25	125	0.612368

Table 2. Grid Search Results for LightGBM

We found that the optimal hyperparameters were a max_depth of 25 and num_leaves 75. Using these in our LightGBM model, paired with the other hyperparameters seen in Table 3, we eventually achieved what would end up being our highest performing model, with a test accuracy of 61.64%. We expected to find a better performing model using LightGBM, as it is particularly efficient and scalable for high-dimensional data. While this doesn't necessarily imply a higher accuracy, it provides us with a better opportunity to try larger parameter grids, which eventually led to a higher performing model.

max_depth	num_leaves	learning_rate	n_estimators
25	75	0.01	1000

Table 3. LightGBM Best Model Hyperparameters

5.4. k-Nearest Neighbors

To try to find the most optimal value for k, in our k Nearest Neighbors model, we tested multiple values for the hyper-parameter "n_neighbors" using Grid Search. The values we tested for k are 3, 5, 7, 9. The best value for k was 9, which means are model uses the 9 closest data points to each query point for making predictions. The model achieved a training accuracy of 64.7% and a testing accuracy of 55.3%.

5.5. Bagging

Our bagging model, implementing BaggingRegressor, performed fairly similarly to our simple Linear Regression model, scoring 55.87% on our test data. This was relatively expected, as bagging doesn't always offer an improvement, and can even decrease performance for a high-variance model that already performs fairly well, such as our original regression model. If it weren't for our computational limitations, we would have explored the use of bagging with a model of higher bias, such as our XGBoost model.

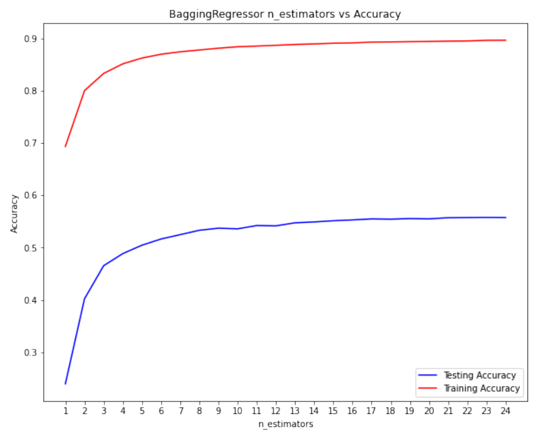

Figure 5. BaggingRegressor n_estimators vs model accuracy

As seen in Figure 5, we experimented with n_estimators from 0 to 25. The results, when plotted, seemed to follow a logarithmic regression. This makes sense when put in context, each preceding estimator learns a little bit more about a different, random subset of the data, creating less for the estimator to learn each time. We thought the visualization of this model was very useful in understanding the BaggingRegressor.

5.6. Model Stacking

Our stacking model we implemented performed very well and was our third best performing model. We implemented a stacking model with linear support vector machines, k-Nearest Neighbors, and XGBoost as our "base learners." While our "meta model" was Random Forests. The k-Nearest Neighbors model was tuned with a value of 9 for k, while the Random Forests model was tuned with a max depth of 10 and 100 estimators. The stacking model achieved a training accuracy of 66.2% and a test accuracy of 61.04%.

6. Conclusions

In our attempt to address the pricing problem that exists for short-term rental properties we examine the ability of machine learning models to predict rental prices. We evaluate a spread of methods on two datasets with either ordinal or one hot encoding, looking for which models produce the best performance compared to the baseline. These methods include Linear Regression, Ridge Regression, Lasso Regression, Elastic Net Regression, XGBoost, LightGBM, Random Forests, Bagging, Support Vector Machines, and K-Nearest Neighbors.

Table 1 shows that LightGBM had the highest overall performance, with an 61.64% accuracy. As others have found, XGBoost also performed very well at 61.42% accuracy, with Model Stacking close behind. LightGBM also has the benefit of being a very low computational cost. That being said, the potential of this project continues beyond our work so far.

In the future we believe it would be important to test our models on higher quality data, applying more complex tasks. With that, it would be very beneficial to the potential of our models if we could increase our computational capabilities. We were somewhat limited in the models we could create because of our machines and software choice. We also are inspired by the potential of adding sentimental analysis from reviews of listings and seasonality considerations in terms of their impact on modeling pricing.

7. Acknowledgements

We would like to thank Rudy Mizrahi, who is the owner of our initial dataset that was publicly available on Kaggle.[4] We would also like to thank Dr. Sebastian Raschka, whose lectures and notes were crucial to our understanding of the materials we put to use in this project.

8. Contributions

We collectively found the original dataset on Kaggle. We then as a group cleaned up the data for further analysis. Dylan Asoh coded a bulk of the initial models with further testing and tuning conducted by Logan O'Brien and Samuel Peterson. Logan led the development of visualizations. Regarding the report, Samuel drafted the Introduction, Related Work, and Conclusion. Logan and Dylan put together the proposed methods sections. Dylan wrote the abstract and covered the Results section. We all collectively worked together on the experiments section.

References

[1] Airbnb and Skift. How airbnb is building a vacation rental platform for the future. *Skift*, Jul 2021.

[2] E. Khandelwal. Which algorithm takes the crown: Light gbm vs xgboost? *Analytics Vidhya*, 2017.

[3] L. Lewis. Predicting airbnb prices with machine learning and deep learning. *Medium*, May 2019.

[4] R. Mizrahi. Airbnb listings in major us cities, 2018. Last updated 03 March 2018.

[5] K. Sprindyte. Covid-19 impact on hotels and short-term rentals. *AirDNA*, Mar 2021.

[6] F. Yue. California rental price prediction using machine learning algorithms. *UCLA Electronic Theses and Dissertations*, 2020.