Goldsmiths
UNIVERSITY OF LONDON

# Predicting Airbnb Prices with Machine Learning and R

**Student No.: 200532303**

Submitted in partial fulfilment of the requirements
for the degree of Master of Science (MSc) in Data Science
of the University of London

Goldsmiths, University of London

August 2021

I certify that this project and the research to which it refers, are
the result of my own work

## Abstract

This study compared a variety of different machine learning methods in an attempt to build a model to predict the price of Airbnb listings in Rio de Janeiro, Brazil. The study also investigated different ways of dealing with missing data, from dropping all such observations, to dropping features, to imputing missing values.

Random Forest, with all missing value observations removed, was used as a benchmark model, achieving a root-mean-square error (RMSE) of 130.59 and R-squared of 0.49. The different approaches to handling missing data failed to improve upon this benchmark model, and so the study moved on to testing different methods.

Support Vector Regression using a radial basis kernel, XGBoost, and a Neural Network model all managed to improve upon the benchmark. The SVR model reported an RMSE of 116.95 and R-squared of 0.60; XGBoost showed RMSE of 77.65 and R-squared of 0.82; while the neural network had an RMSE of 121.48 (R-squared not initially measured). These models were then evaluated on the test data. All three models showed clear signs of overfitting, with significantly worse performance on the test data compared to the training data (RMSE of 129.14, 121.21, 132.90 and R-squared of 0.50, 0.56, 0.47 respectively).

While the XGBoost model proved to be the best model for the dataset, the overall predictive power remained weak. It seems unlikely that such a model could be relied upon as an accurate pricing model.

# Contents

## 1. INTRODUCTION

This project examines Airbnb listings in Rio de Janeiro, Brazil. Airbnb is an online platform that makes it easy for homeowners to let their properties. According to the company,

> Airbnb was born in 2007 when two Hosts welcomed three guests to their San Francisco home. It has since grown to 4 million Hosts who have welcomed more than 900 million guest arrivals across over 220 countries and regions. (*Airbnb Announces Second Quarter 2021 Results*, 2021)

The original aim of the project was to attempt to predict the average review score based on the features of a listing. However, it soon became apparent that this would not be feasible (see section 2.1 for details). Therefore, the aim was revised to predict the price of a listing based on those same features.

Correct pricing of a listing is important. It is invariably one of the first things considered when searching for a property, and a listing with an extortionately high price will gain very few bookings. Conversely, under-pricing a listing will reduce profits for the host and could even drive down the market. Therefore, having a tool to predict the appropriate price for a listing would be extremely helpful, both for hosts and for guests.

### 1.1 Related Work

Wang and Nicolau (2017) investigated which factors have the most impact on pricing of an Airbnb listing. Their analysis used both ordinary least squares (OLS) and quantile regression (QR). They used data from 33 cities in 13 different countries, analysing the impact of 25 separate features. The features considered were:

- host is superhost
- number of listings
- host has profile picture
- host’s identity is verified

- distance to city centre
- accommodation type 1 (apartment, condominium or loft)
- accommodation type 2 (bed & breakfast or dorm)
- accommodation type 3 (bungalow, house, townhouse, villa, cabin or chalet)
- entire home
- private room
- shared room
- number of guests
- number of bathrooms
- number of bedrooms
- ‘real’ bed included
- internet
- breakfast
- free parking
- instant booking available
- smoking allowed
- cancellation policy
- guest’s picture required
- guest’s phone required
- reviews per year
- overall review score.

OLS analysis found 24 of the 25 features to be good predictors of price, while QR analysis found all 25 features to have a significant effect, though this was often dependent on the price range being predicted.

This study proves useful in identifying features to include in a price predictor model. However, one potential issue with the study, as acknowledged by the authors, is the minimal exploration of the impact of the city (and by extension, country) on the importance of features in relation to price: “this study does not provide insights on the differences of each price determinant’s impact on price in different cities" (Wang and Nicolau, 2017, p. 130). It may be the case that certain features are, in fact, unimportant in one city, yet significant in another.

Choudhary, Jain and Baijal, (no date) examined Airbnb data from San Francisco with the aim of using machine learning both to predict pricing and to predict availability of properties. For the pricing predictor, they began by fitting a linear regression model to the data. This revealed two distinct groups of pricing data; approximately 45% of the data had a prediction error of less than 30 USD, whereas the remaining 55% of the data had a greater prediction error. The authors subsequently categorised these as 'easy' and 'hard'. The remainder of the study focused only on predicting the 'easy' group of listing.

A random forest model trained on these so-called easy listings showed low training error but high test error, suggesting the model was overfitting to the training data. To combat this, the study first applied both upsampling and downsampling to the data in order to get a more balanced dataset in terms of prices of listings. They then repeated the random forest training, but this time using 10-fold cross-validation combined with a grid search through various hyperparameters including number of trees, maximum number of features considered when splitting a node, maximum depth of the tree, minimum number of samples required to split a node, minimum number of samples required to form a leaf, and whether to bootstrap or not.

The best performing model had a root-mean-square error of 28 USD and a mean percentage error of 0.04%, leading the authors to conclude that they would be able to predict prices for new listings within 29 USD. While this does appear promising, the clear limitation of this study is that only 45% of the original data, the data that was considered to be 'easy' to predict, was analysed. As a result, the model developed for the study would be unusable on over half of all Airbnb listings, giving it limited practical application.

Chakraborty *et al.*, (no date) used Airbnb data from New York to build and compare a series of price predicting models. The study began by building a linear regression model using just nine features:  neighbourhood_group, latitude, longitude, room_type, minimum_nights, number_of_reviews, reviews_per_month, calculated_host_listings_count and availability_365. When plotting the model, the residuals showed a clear pattern, while the normality assumptions were not satisfied (as revealed by the Normal Q-Q plot), and so this linear model was considered inappropriate.

The next step was to use various feature selection methods to try to build a better model. Best Subset Regression, Stepwise Regression with AIC, Stepwise Regression with BIC, and Lasso Regression were all applied to the dataset. Both stepwise models performed equally well. Given that the Stepwise Regression using BIC model involved fewer explanatory variables, and was therefore simpler, this was chosen as the most appropriate model.

On the training set, the BIC model produced a mean-square error (MSE) value of 1,651 and R-squared value of 0.45. On the test set, the MSE was 1,633. R-squared was not reported for the test set. R-squared represents the amount of variance in the dependant variable that can be explained by the model. In this case, the best performing model could only explain 45% of the price variance. Again, this may prevent the model from being of real practical use. One possible reason for the relatively poor performance is the small number of explanatory variables used, when compared with those identified by Wang and Nicolau (2017) as being significant.

Lewis (2019) used 17 explanatory variables to predict pricing from a dataset of London Airbnb listings. XGBoost and Neural Networks were employed as the models. Categorical features were encoded using one-hot encoding, while numerical data was standardised. The dataset was split 80/20 for training and testing. XGBoost was used as a benchmark model, reporting a mean-square error value of 0.1576 on the training set and 0.159 on the test set, and R-squared value of 0.7321 training and 0.7274 test.

A series of neural networks were then implemented, with the author experimenting with different depths, regularisation, dropout, optimisers, batch sizes, epochs, standardisations and different explanatory variables. The best performing neural network was found to be a four-layer network utilising L1 regularisation with Adam as the optimiser. However, this model still performed less well than the XGBoost model (test mean-square-error of 0.1689 and test R-squared of 0.7105).

The study concluded that XGBoost is the most appropriate model for predicting Airbnb pricing, and could potentially be improved even further with tuning of the hyperparameters. The study further considered that photographs of listings may be an important explanatory factor, suggesting this a potential area for further research.

This project follows similar methods to those discussed above, but focuses on Airbnb listings in Rio de Janeiro. Rio de Janeiro is one of the most popular tourist destinations in the world, with famous landmarks such as Christ the Redeemer and Copacabana Beach, and events including Carnival and Rock in Rio. As a result, the city has over 35,000 Airbnb listings (*Inside Airbnb: Rio de Janeiro. Adding data to the debate.*, no date). Indeed, Airbnb is important to Brazil as a whole. Pofahl (2017) cites a study by the Economic Research Institute Foundation, showing that Airbnb rentals and related tourism added 2.5 billion BRL to Brazil's GDP in 2016 (over 300 million GBP).

Furthermore, this study compares a larger number of machine learning methods in order to determine the best method for price prediction. The following section briefly outlines the methods employed.

## 1.2 Machine Learning Methods

This study applies the following methods to the dataset: Random Forest, Multiple Linear Regression, Support Vector Regression, XGBoost and Neural Networks.

Random Forest algorithms (Breiman, 2001) involve a series of decision trees. One of the main problems with decision tree algorithms is that they tend to overfit the training data, resulting in poor performance when used to make predictions on previously unseen data. One way to rectify this is by pruning the tree (reducing its depth), but it can be difficult to determine the appropriate pruning level in order to reduce overfitting without introducing underfitting. Random Forest algorithms solve this by training multiple decision trees on different subsets of the training data. The final model is a combination of each individual tree. This significantly reduces overfitting. Another advantage of Random Forest algorithms is that they tend to be quick to train and evaluate. For this reason, Random Forest was used in this study to create a benchmark model and to aid feature selection (see section 2).

Multiple Linear Regression is an extension of simple linear regression. Simple linear regression involves fitting a line to the data of the form $y = a + bx$, where $y$ is the dependent variable, $x$ is the explanatory variable, and $a$ and $b$ are parameters to be learned (the intercept and the slope). Multiple Linear Regression simply extends this equation, adding multiple explanatory variables, each with their own slope parameter. Linear regression (both single and multiple) involves some important assumptions about the dependent variable, namely that:

> the errors are normally distributed, the errors are confined to the response variable, and the variance is constant. The explanatory variables are assumed to be measured without error. (Crawley, 2012, p. 490)

Support Vector Regression is a variation on Support Vector Machines (SVM). SVMs use kernels to map data to a higher dimension in order to fit a hyperplane to the data. This hyperplane is linear in the higher dimension but non-linear in the original dimension. It is known as the maximum-margin hyperplane, and is fit so as to best separate the data points into their distinct classes. The data points that lie closest to the maximum-margin hyperplane are the support vectors. Support Vector Regression operates similarly, with a small number of support vectors used to define the model. As with SVMs, various kernels can be used to model nonlinear problems (Witten *et al.*, 2011).

XGBoost, or eXtreme Gradient Boosting (Chen and Guestrin, 2016), is currently one of the most widely used machine learning methods, both for regression and classification problems, due to its speed and performance. XGBoost is a type of gradient tree boosting algorithm. A decision tree is produced and evaluated. Another tree is then produced, this time optimised to correct the errors from the first tree. This pattern continues a pre-defined number of times, until there exists a number of 'weak' decision trees. The final model is a weighted combination of all of these weak trees (Saraswat, no date).

Finally, feed forward neural networks are deep learning algorithms modelled on the human brain. They consist of a series of layers, with each layer containing a number of nodes or neurons. The first layer is the input layer, which takes the data. The last layer is the output layer, which gives the results of the model. The layers in between are known as hidden layers, and these are the layers that create the mapping from input to output. This mapping relies on weights and biases. In a simple, fully connected feedforward network, every node in one layer is connected to every node in the next layer. Each connection has an associated weight, which is multiplied by the output of the previous layer before it is passed to the neuron in the next layer. That neuron takes this input, adds a bias term, and then passes the result through an activation function. The output of this activation function is then passed on to the next layer, with new weights and biases.

The difference between the network output and the expected output is the loss of the network. A method known as backpropagation is then used to work backwards through the network, updating the weights and biases in such a way as to minimise this loss. Neural networks can be extremely powerful, highly performant models. However, they are generally quite complex, involving a lot of hyperparameters, and may take a long time to train, depending on the network architecture and amount of training data involved.

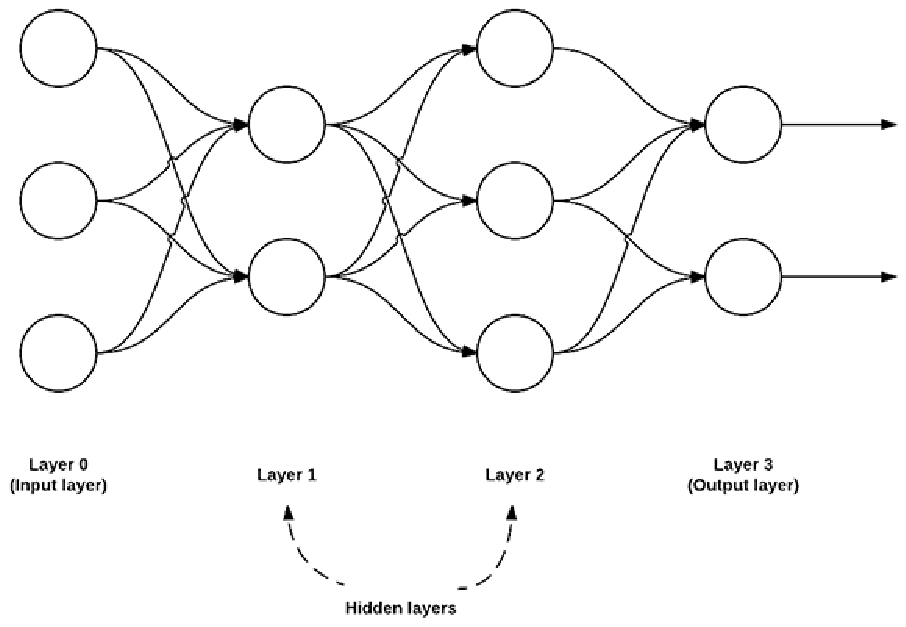


*Figure 1: Neural network example with two hidden layers (Rosebrock, 2016)*

## 2. METHOD

### 2.1 Data Preparation and Initial Modelling Exploration

The data for this project was obtained from http://insideairbnb.com (*Inside Airbnb. Adding data to the debate.*, no date). The data was obtained and cleaned as part of the first Coursework. As a brief summary:

- The CSV file was read into RStudio as a Dataframe object. Any features (columns) considered irrelevant were dropped.
- Since the target variable was intended to be review_scores_rating, any rows with missing values in this column were dropped. As mentioned, this project ultimately focused on

predicting price, so it may be that these observations could have been retained. However, as will be seen, review_scores_rating remained an important feature in predicting the price, and so it is likely these rows would have been dropped regardless.

- The column host_verifications, which contained multiple entries per cell, was one-hot encoded.
- The amenities columns, which also contained multiple entries per cell, was dropped due to the complexity that would have been involved in performing one-hot encoding.
- Character columns were converted to factor format and numeric columns converted to numeric format.
- Columns containing True or False values were converted to 1 or 0.
- Any rows with a missing price value were dropped, since this was considered to be an important feature for prediction (even more so with the revised aim of the project).
- Two columns not used in the final dataframe, name and description, were searched for extra information that could be used to fill missing values in the bathrooms, beds and bedrooms columns.
- Any observations with minimum nights greater than 14 days were dropped, on the basis that these types of stay are of a different character to typical holiday lets.
- Any observations with a maximum nights' value greater than 1,125 were changed to 1,125 on the basis that these were likely errors in the data, and that 1,125 appeared to be a standard cut-off for Airbnb.
- The final dataset contained 14,936 observations with 43 variables.

Two of the columns dropped initially were host_total_listings_count and availability_90. However, both Wang and Nicolau (2017) and Lewis (2019) found total listings to be an important predictor of price, while Lewis also found availability_90 to be an important factor. For this reason, these columns were added back in to this dataset (matching on the ID column). The ID column was then dropped, owing to the fact that it contained only unique values which would therefore have no predictive power.

During the intial data cleaning process, a number of observations with missing values were deliberately retained, so that the decision as to how to handle them could be made during the modelling process. In order to get a simple benchmark model, the first approach employed here was simply to drop any rows containing NA values. After this, the data was split into a training and test set (employing an 80% training to 20% test split). The 'createDataPartition' function from the caret package was used in order to quickly and easily create a balanced training/test split. Having balanced training and test sets is important when creating predictive models in order to avoid overfitting to the training data.

The randomForest package was used to create the benchmark model. 100 trees were used in order to give a good balance between speed of training and accuracy. As predicting pricing is a regression problem (it involves a continuous rather than discrete value), the appropriate performance metrics were considered to be root-mean-square-error (RMSE) and R-squared. Mean-square-error (MSE) could have been used, but RMSE was preferred due to the fact that it returns a value in the same units as the target variable, and so is simpler to interpret that MSE. R-squared tells us the amount of variance that is explained by the model.

The random forest model was created with review_scores_rating as the target variable. This returned an RMSE of 9.24, which was considered relatively high, given the maximum review score was 100, and especially when considering that almost 90% of the scores were between 90 and 100. Furthermore, the model produced an R-squared score of -0.04, signifying that the model performed worse than using a simple average of the review scores as a prediction.

As a result of this poor predictive power, it was decided at this point that a better approach would be to attempt to predict price rather than review score. The random forest modelling was repeated with price as the target variable, resulting in an RMSE of 130.59 and R-squared of 0.49. This was a considerably better performance compared to the review score predictor, and so the remainder of the project focused on improving this price predictor, both by trying different model types and by adjusting the hyperparameters of those models.

## 2.2 Multiple Linear Regression

The first alternative model type to be tried was a multiple linear regression model. In order to determine which features of the dataset to include in the linear regression model, R's step function was utilised. Due to the large number of features, step-forward was used (starting with no features and adding one at a time) rather than step-backward (starting with all features and removing one at a time) (Long and Teetor, 2019, p. 352). The best model used 27 of the features of the dataset, and had an adjusted R-squared value of 0.50. This model showed promise. However, as discussed in section 1.2, there are a number of assumptions that must be met in order for a linear regression model to be appropriate. In this case, the residuals showed a clear pattern, while the normal-QQ plot deviated significantly from the straight line (see Figure 2: Residual plot and QQ-plot for multiple linear regression model). While it may have been possible to address this by log-transforming the target variable, it was instead decided to move on to a different method.

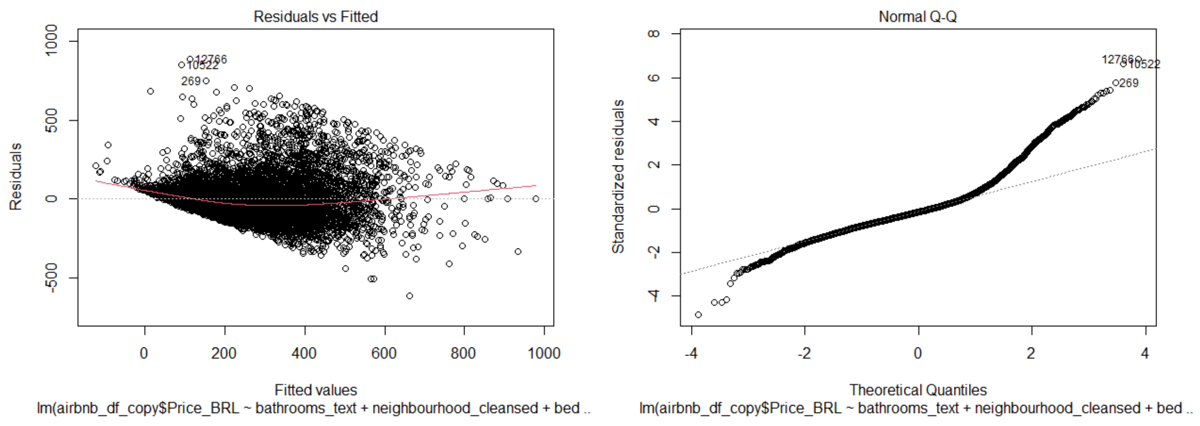


*Figure 2: Residual plot and QQ-plot for multiple linear regression model*

## 2.3 Random Forest

The next step was to return to the random forest model to see if this could be improved. As discussed, the original approach to NA values was to simply drop all such observations in order to produce a quick benchmark model. The downside to such an approach is that these observations may contain important information in other, non-empty columns, which is lost when they are dropped. An alternative approach is to drop the columns that contain the missing information, while retaining all rows. Of course, this can also result in the loss of vital information, especially if the dropped features are highly important explanatory variables, and so this does not guarantee an improved performance.

In the current dataset, two distinct groups of features with missing values could be observed. Four of the features (bedrooms, host_acceptance_rate, host_response_rate and host_response_time) contained over 1,000 missing values. The remaining features each contained fewer than 500. For the first attempt to improve the random forest model, rather than dropping all of the observations, only those observations found in the second group were dropped, while the columns in the first group were dropped. This left 14,456 rows and 39 columns for modelling, rather than the 9,532 rows and 43 columns used in the benchmark model.

Before dropping these columns, a random forest model was used to determine the overall importance of each feature to the model's predictive power. The model determined that bedrooms was the 5th most important variable, host_acceptance_rate the 10th most important variable, host_response_rate 14th most important and host_response_time 18th most important. Given the relatively high importance of these features, it was perhaps unsurprising that dropping these columns did not improve the model (RMSE = 130.17, R-squared = 0.40).

An alternative to dropping observations and columns is to impute missing values. While this can help to preserve useful information, the danger is in distorting the dataset. It is important not to impute values in the test set, otherwise the reported model performance will be artificial, and the model will not perform as well on new, unseen data.

Two approaches to imputation were examined in this study. Firstly, the reduced dataset (14,456 rows) was used, but rather than dropping the other four features, the missing values were imputed. To perform the imputation the MICE (Multivariate Imputation by Chained Equations) package was used (Buuren and Groothuis-Oudshoorn, 2011). MICE performs multiple imputations (here, 3 imputations were performed) using one of a variety of possible imputation methods (in this case, predictive mean matching).

After imputing the values, a density plot was created to ensure that the imputed values followed the overall pattern of the existing values (Figure 3: Density plot showing distribution of original values and imputed values). Three separate random forest models were then created using each of the three datasets created by the MICE package. All of the models gave approximately the same results, with the best model achieving an RMSE of 143.57 and R-squared of 0.42.

This was perhaps unsurprising, given the large percentage of missing values in these four columns. Imputation generally performs better when less than 5% of the values are imputed. With this in mind, the second approach took the complete dataset and dropped any observations with missing host_response_time values (the feature with the highest number of missing values). This left 10 columns with missing values, with 195 values missing on average per column. These values were then imputed, again using the MICE package. This approach showed an improvement over the first. However, the best model still underperformed when compared to the benchmark model (RMSE = 135.14, R-squared = 0.47). It was therefore concluded at this point that simply dropping all observations containing NA values remained the best approach. The remainder of the analysis continued with this dataset.

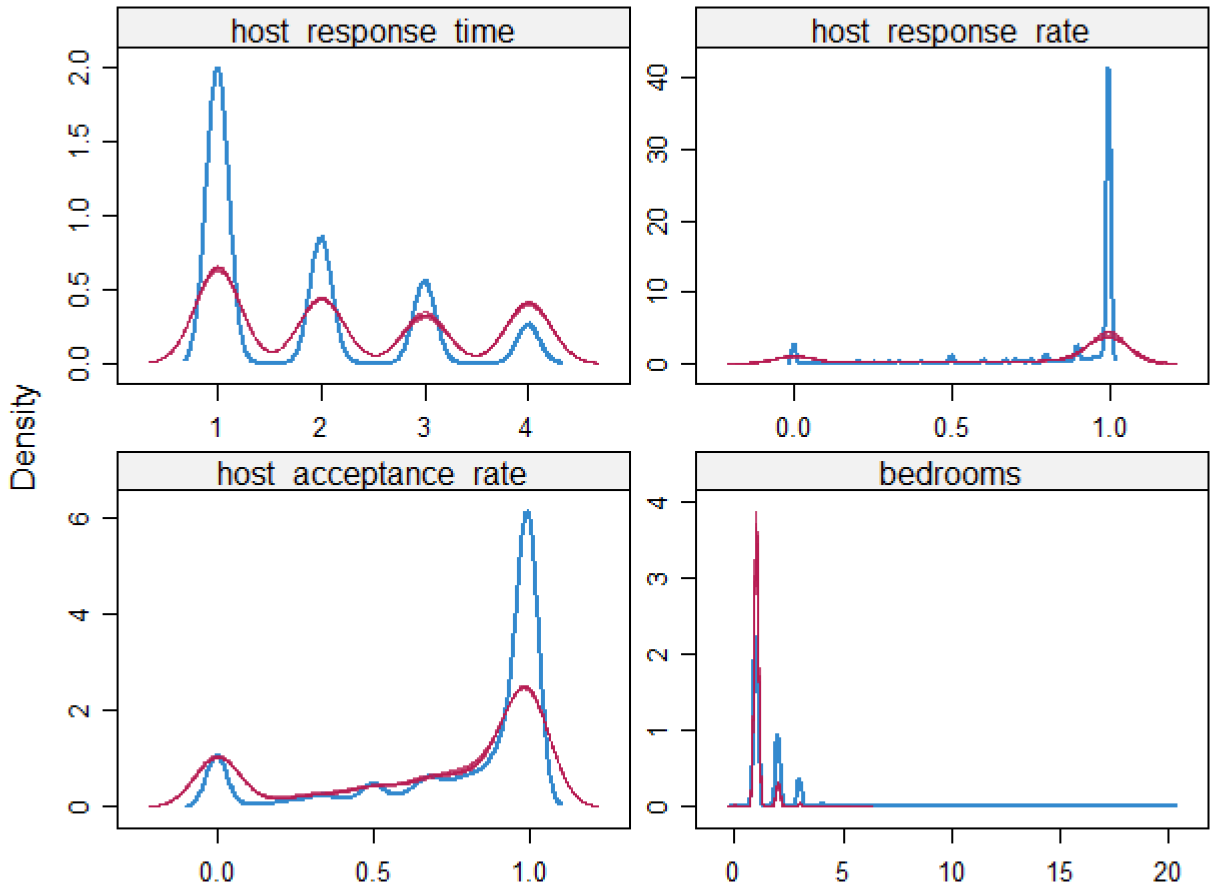


*Figure 3: Density plot showing distribution of original values and imputed values*

## 2.4 Support Vector Regression

The next modelling technique to be tested was Support Vector Regression (SVR). SVR requires numerical data, with different features sharing similar scales. First, the categorical columns host_response_time, neighbourhood_cleansed, and room_type were converted to numerical format using ordinal encoding (recasting the values as numeric). Before converting the bathrooms_text column, a new column was created to indicate whether the bathroom was shared or not (using 1s and 0s to represent true or false). This meant the remaining values in bathrooms_text could be easily converted into numerical format.

For scaling the data, Caret's preProcess function was used, with the method set to 'range' in order to scale the values between 0 and 1. Only the explanatory variables were scaled, with the target variable left in its original scale to aid interpretation and comparison of results.

The last step before building the SVR models was to check for multicollinearity in the data. Multicollinearity exists when two or more of the features in the model are highly correlated with one another. Not only can multicollinearity hamper interpretation of a model, as it is difficult to determine which feature is having what effect, but it means that the model may be more complicated than is necessary. The aim when building a machine learning model is to use the simplest model with the highest accuracy/explanatory power. Consequently, whenever it is possible to drop a feature from the model without having too detrimental an impact on the performance of the model, that feature should be removed.

To check for multicollinearity, a correlation plot was constructed, with larger circles indicating a higher correlation. The immediate observation from the plot was that there was an issue calculating the correlation for the 'None' feature. Upon inspection, it became clear that, as a result of dropping rows earlier in the analysis, this column now contained only 0 values. It was consequently removed from the dataset. Host_response_time and host_response_rate showed a high level of correlation. Given that the earlier investigation of the importance of features showed response rate to be more important to the random forest model, response time was dropped. Government_id showed a high correlation with host_identity_verified. The majority of the other verification methods also showed at least some correlation with host_identity_verfied. This was unsurprising, given that these features all represent different ways of the host verifying their identity. Considering the relative unimportance of these features shown by the earlier importance plot, these were also all dropped from the dataset. Two other significant correlations were observed between beds and accommodates, and between shared_bathroom and room_type. Again, taking into account relative importance, beds and shared_bathroom were dropped. Finally, all of the individual review scores showed correlation with the overall review score, which was also to be expected, considering that this feature is derived from the individual scores. As a result, the individual scores were also removed.

To perform the modelling, the e1071 package was used to build two simple SVRs, one with a polynomial kernel and the other with a radial basis function kernel. Both models performed relatively well, with an RMSE of 126.35 and R-squared of 0.53 for the polynomial model, and an RMSE of 116.95 and R-squared of 0.60 for the radial basis function model.

## 2.5 XGBoost

For the XGBoost modelling, an initial XGBoost model was created with default hyperparameters, using two-fold cross validation. This model showed an improvement over the random forest benchmark, with an RMSE of 125.50 and R-squared of 0.54. Given this improvement, and given the findings of Lewis (2019) (see section 1.1), it was considered appropriate to perform a hyperparameter grid search in an attempt to improve the model. This was done by creating a dataframe of multiple hyperparameter combinations using R’s expand.grid function, which was then passed to the tuneGrid argument of Caret’s train function. The hyperparameters considered were nrounds (number of trees: 100, 200 or 500), max_depth (maximum depth of the tree: 5, 10, 20 or 25), subsample (ratio of training data sampled prior to growing the tree, used to prevent overfitting: 0, 0.25, 0.5, 0.75 or 1) and colsample_bytree (ratio of columns used when constructing the tree: 0, 0.25, 0.5, 0.75 or 1). Additionally, five different learning rates (eta) were supplied. These values were chosen to be between 0.5 and 0.001 inclusive, and were selected on a log-scale in order to get an even distribution of values between the two end points. Gamma and min_child_weight were set to their default values of 0 and 1 respectively (*XGBoost Parameters — xgboost 1.5.0-dev documentation*, no date). Cross-fold validation with five folds was used in order to maximise the amount of training data, implemented using Caret’s trainControl function. The grid search found the optimal hyperparameters to be as follows:

- nrounds = 500
- max_depth = 5
- subsample = 0.75
- colsample_bytree = 0.75
- eta = 0.0594

Due to the time taken to perform the hyperparameter search, this section of code is commented out in the accompanying script. The model was instead reconstructed using these optimal hyperparameters. This required converting the training data to DMatrix format.

This model had an RMSE of just 77.65 and R-squared value of 0.82.

## 2.6 Neural Network

The optimal neural network architecture identified by Lewis (2019) was employed here in order to get an initial assessment of the neural network’s potential. This architecture consisted of an input layer with 128 nodes, three hidden layers with 256, 256 and 512 nodes respectively, and an output layer using a linear activation function. The input layer and hidden layers all used the ReLU activation function along with L1 regularisation with a regularisation penalty of 0.005. Adam was used as the optimiser.

This architecture produced a training RMSE of just 99.62. However, the final validation RMSE (after 100 epochs) was 148.66, indicating severe overfitting of the network to the training data. The best result occurred around epoch 49, where the training and validation RMSE were roughly equal at approximately 121.48.

To see if this could be improved upon, the tfruns function (Allaire, no date) was used to perform a grid search over various hyperparameters. This first involved creating a new R script in which the encoded and scaled dataset was loaded, split into training and test sets (using the same 80/20 split), and then reshaped into matrix format as required by Keras.

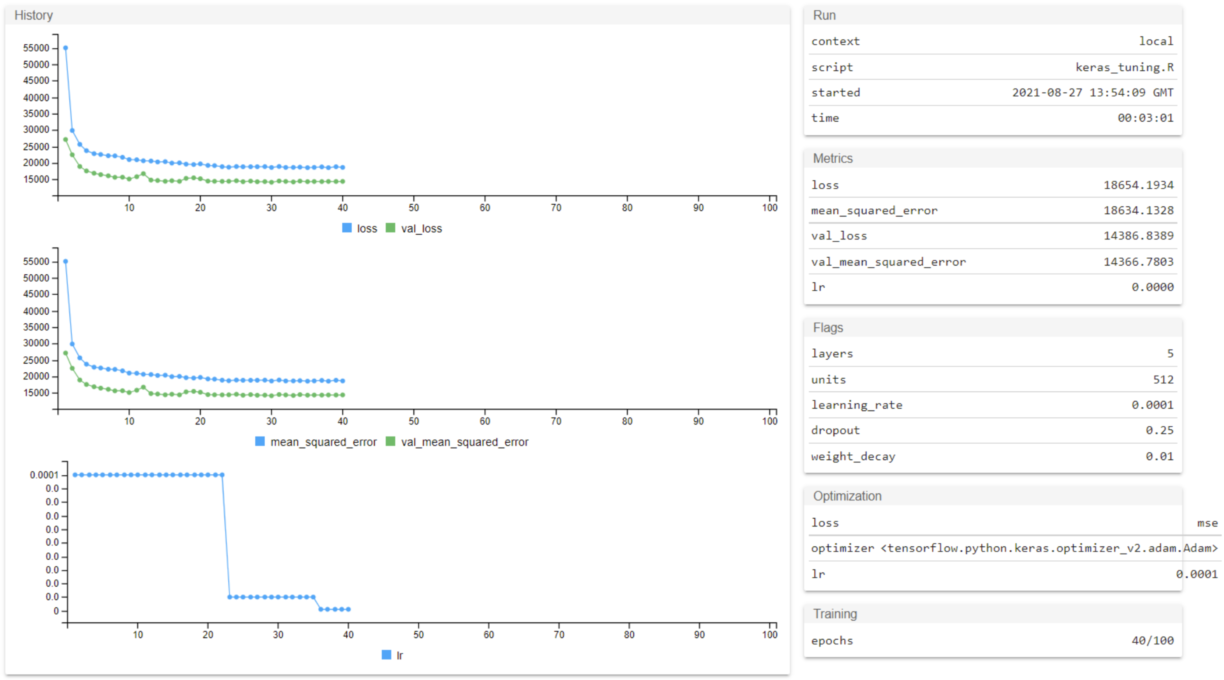
This same script contained a list of flags, indicating the different hyperparameters to be used (along with default values). Following the flags, the network architecture was defined. The architecture used for this project was developed from code produced by Bradley Boehmke, made available via GitHub (Boehmke, 2021). Similarly to the initial neural network, the network used for the hyperparameter search consisted of a series of fully connected layers using the ReLU activation function, followed by a final output layer using a linear activation function. Unlike the initial model, L2 regularisation was used in place of L1 regularisation, and a dropout layer was added in between each fully connected layer. The number of layers, number of nodes, optimiser learning rate, dropout rate and the L2 regularisation penalty were the hyperparameters being searched.

Finally, the fitting of the model was defined using 100 epochs with a validation split of 20% and two callbacks. The first callback was used to reduce the learning rate if the learning plateaued for five consecutive epochs. The second callback stopped model training early if the validation MSE failed to improve for 10 consecutive epochs. The resultant model was saved as a (TensorFlow) SavedModel.

The script (which is included as part of the project submission) was called using Keras's tuning_run function, which was supplied with the script source file and the list of flags to be searched. These were as follows:

- layers – 3, 4 or 5
- units (nodes) – 128, 256, 512
- learning_rate – 0.0001, 0.0008, 0.007, 0.059, 0.5 (as per the XGBoost model)
- droput – 0, 0.25, 0.5
- weight_decay (L2 regularisation penalty) – 0.1, 0.01, 0.001

This resulted in 405 separate models, each of which were trained and evaluated on the validation set. The best performing model (which stopped at epoch 40) had an MSE of 14,752, which equates to an RMSE of 121.46. Note - Keras does not report an R-squared value as standard. However, the R-squared value can be calculated based on predictions (see section 3). This network consisted of an input layer with 512 nodes, followed by a dropout layer using a dropout rate of 0.25. This was followed by four hidden layers of fully-connected, 512 node layers, plus dropout, before a final output later. The learning rate chosen was 0.0001, with a regularisation penalty of 0.01.



Run

| | |
|---|---|
| context | local |
| script | keras_tuning.R |
| started | 2021-08-27 13:54:09 GMT |
| time | 00:03:01 |

Metrics

| | |
|---|---|
| loss | 18654.1934 |
| mean_squared_error | 18634.1328 |
| val_loss | 14386.8389 |
| val_mean_squared_error | 14366.7803 |
| lr | 0.0000 |

Flags

| | |
|---|---|
| layers | 5 |
| units | 512 |
| learning_rate | 0.0001 |
| dropout | 0.25 |
| weight_decay | 0.01 |

Optimization

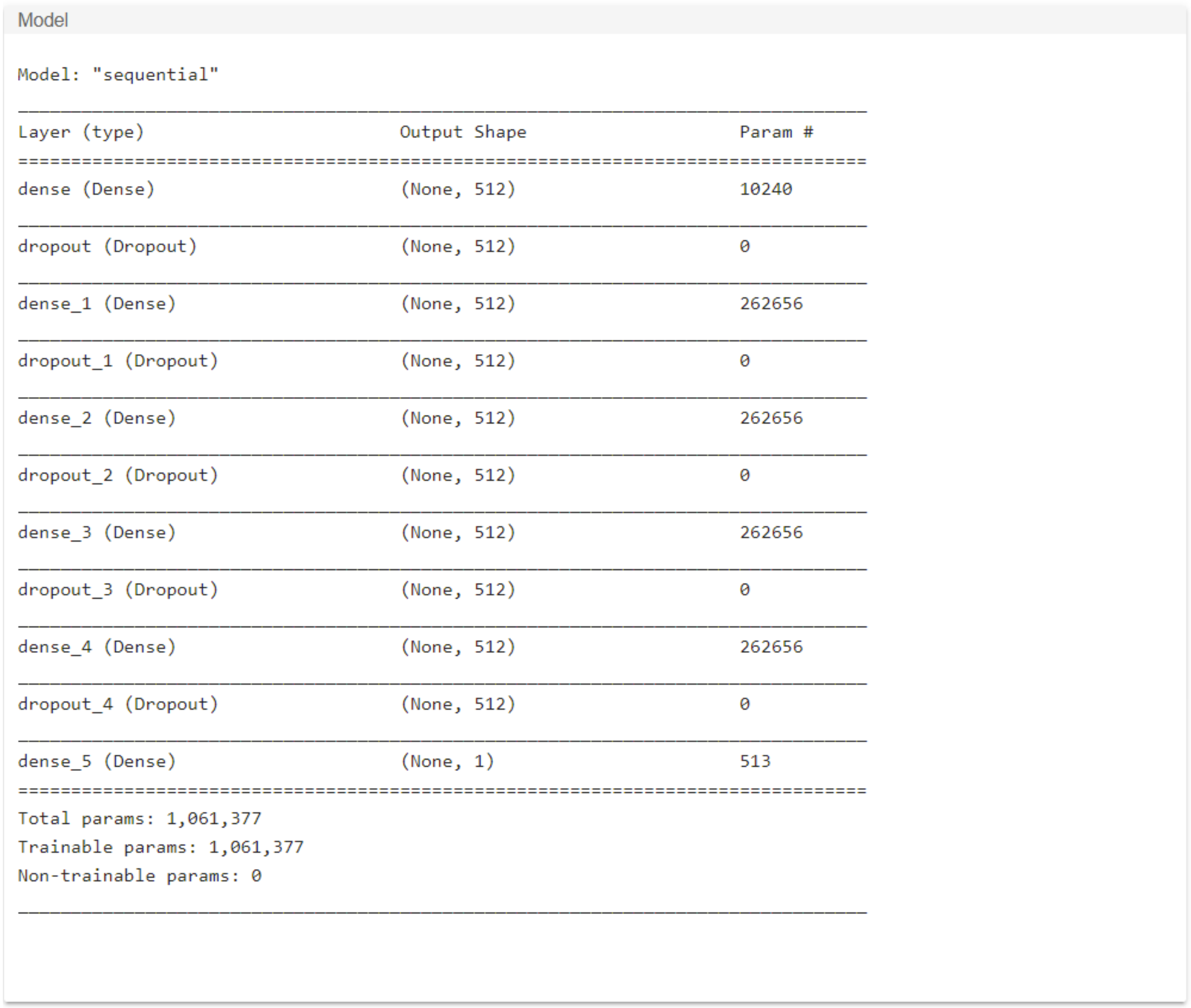
| | |
|---|---|
| loss | mse |
| optimizer | <tensorflow.python.keras.optimizer_v2.adam.Adam> |
| lr | 0.0001 |

Training

| | |
|---|---|
| epochs | 40/100 |

*Figure 4: Tuned neural network training and hyperparameters*

Similarly to the tuned XGBoost model, due to the time taken for training and validation of each model (over 16 hours on a laptop), the neural network hyperparameter search is commented out in the accompanying script. The model was instead reconstructed using the identified values.


Model

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               10240
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 513
=================================================================
Total params: 1,061,377
Trainable params: 1,061,377
Non-trainable params: 0
_________________________________________________________________
```

*Figure 5: Tuned neural network architecture.*

## 3. RESULTS

The results discussed in section 2 (summarised in Table 1: Validation results for each model) were derived from the training or validation sets (depending on model format). These results were used to select the best models.

| Model | Dataset Details | Hyperparameters | RMSE | $R^2$ |
|---|---|---|---|---|
| Random Forest | • NA rows dropped | • No. trees = 100 | 130.59 | 0.49 |
| Random Forest | • Four columns dropped<br>• Remaining NA rows dropped | • No. trees = 100 | 130.17 | 0.40 |
| Random Forest | • host_response_time encoded as numeric<br>• Four columns with NAs imputed<br>• Remaining NA rows dropped | • No. trees = 100 | 143.57 | 0.42 |
| Random Forest | • host_response_time NA rows dropped<br>• Remaining NA values imputed | • No. trees = 100 | 135.14 | 0.47 |
| SVR Polynomial | • NA rows dropped<br>• Categorical variables encoded<br>• Features scaled to be between 0 and 1<br>• Highly correlated features dropped | • C = 1<br>• Gamma = 0.05263158 | 126.35 | 0.53 |
| SVR Radial Basis Function | • NA rows dropped<br>• Categorical variables encoded<br>• Features scaled to be between 0 and 1<br>• Highly correlated features dropped | • C = 1<br>• Gamma = 0.05263158 | 116.95 | 0.60 |
| XGBoost Tree | • NA rows dropped<br>• Categorical variables encoded<br>• Features scaled to be between 0 and 1<br>• Highly correlated features dropped | • Eta = 0.3<br>• Max_depth = 3<br>• Colsample_bytree = 0.6<br>• Subsample = 1<br>• Nrounds = 100 | 125.50 | 0.54 |
| XGBoost Tree (tuned model) | • NA rows dropped<br>• Categorical variables encoded<br>• Features scaled to be between 0 and 1<br>• Highly correlated features dropped | • Eta = 0.0594<br>• Max_depth = 5<br>• Colsample_bytree = 0.5<br>• Subsample = 0.75<br>• Nrounds = 200 | 77.65 | 0.82 |
| Neural Network | • NA rows dropped<br>• Categorical variables encoded<br>• Features scaled to be between 0 and 1<br>• Highly correlated features dropped | • Hidden Layers = 3<br>• Nodes = 128, 256, 256, 512<br>• Regularisation = L1<br>• Regularisation penalty = 0.005<br>• Activation = ReLU<br>• Optimiser = Adam<br>• Learning rate = 0.001<br>• Epochs = 49 | 121.48 | N/A |
| Neural Network (tuned model) | • NA rows dropped<br>• Categorical variables encoded<br>• Features scaled to be between 0 and 1<br>• Highly correlated features dropped | • Hidden Layers = 4<br>• Nodes = 512<br>• Regularisation = L2<br>• Regularisation penalty = 0.01<br>• Dropout = 0.25<br>• Activation = ReLU<br>• Optimiser = Adam<br>• Learning rate = 0.0001<br>• Epochs = 40 | 121.46 | N/A |

*Table 1: Validation results for each model*

However, to properly evaluate model performance, the model must be tested on data it has not been exposed to during training. Given the results seen above, the radial basis function SVR model, the tuned XGBoost model, and the tuned neural network model were all chosen to be evaluated on test data.

| **Model** | **Test RMSE** | **Test $R^2$** |
|---|---|---|
| SVR Radial Basis Function | 129.14 | 0.50 |
| XGBoost Tree (tuned model) | 121.21 | 0.56 |
| Neural Network (tuned model) | 132.90 | 0.47 |

*Table 2: Test results for best models*

As Table 2 shows, all three models performed significantly less well during testing when compared to training. This is clear evidence of overfitting.

The XGBoost model demonstrated the best performance during testing. However, this is still a relatively poor performance, explaining only 56% of the variance in the price. Furthermore, an RMSE of 121.21 is quite high, given that the average price of a listing in the dataset is 263.03.

The underfitting of the model is very apparent when plotting the predicted values against the actual values (Figure 6: Graph of predictions vs. actual values using XGBoost model (Berhane, 2018). While the model's predictions below approximately 250 BRL are generally fairly accurate, above this the accuracy appears to drop off considerably. For example, in the top-left hand corner of the graph there is a data point with a predicted price of approximately 150 BRL, but with an actual price of 1,00 BRL. If the host used this model to price their listing, they would be seriously undervaluing the property.

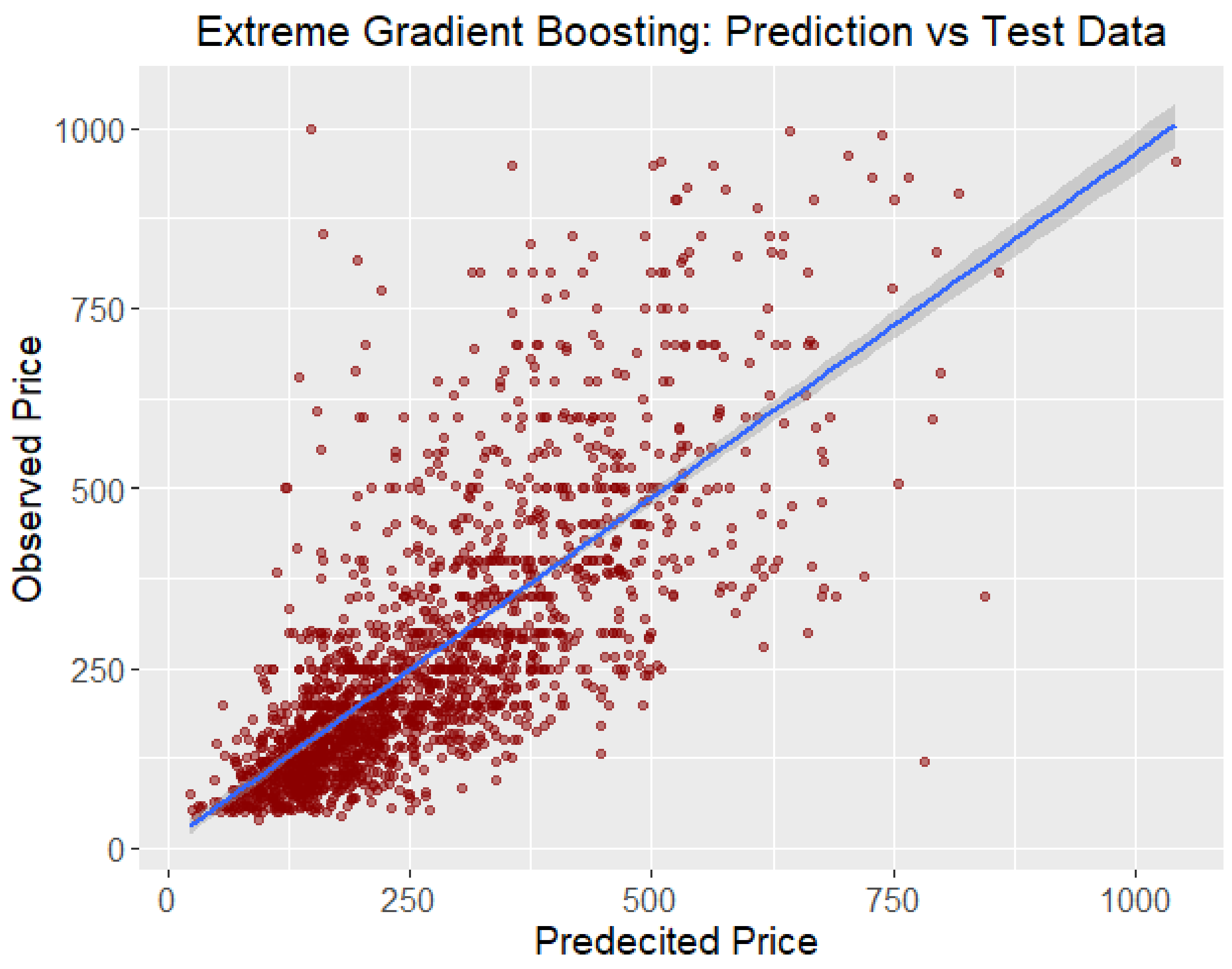


*Figure 6: Graph of predictions vs. actual values using XGBoost model*

## 4. DISCUSSION

This project conducted a comparison of different machine learning methods with the aim of producing a model that could accurately predict Airbnb pricing for listings in Rio de Janeiro. The study found XGBoost to be the best performing model for this purpose, backing up the findings of Lewis (2019).

However, the overall results from the study were somewhat disappointing, with the best performing model only able to explain 56% of the variance in pricing. This model could potentially be used a starting point for pricing Airbnb listings, but should not be relied upon alone.

There are a number of possible reasons for the comparatively poor results found in this study when compared with contemporary studies. One likely reason is the different features used in the modelling process. In particular, Lewis included a number of amenities as features which were not included in this study (see Coursework 1 for details). Another possible reason is that there may exist some fundamental differences in pricing models between different cities and countries, with some cities having more consistent, predictable pricing than others.

Both of these could be areas to investigate in future studies, particularly the inclusion of amenities in the dataset. Given the way that the amenities are included in the original dataset, this would likely require some natural language processing.

Another factor that could be taken into account in future work is timing of bookings. Certain events in Rio de Janeiro, such as Rio Carnival and Rock in Rio, attract visitors from all over the world, and often properties being listed on Airbnb solely for the duration of these events. It may be that an improved model could be developed by categorising these prices differently to the year-round pricing.

Future studies could also take different technical approaches, such as performing one-hot encoding rather than ordinal encoding, or performing a wider search of hyperparameters, especially when it comes to the architecture of the neural network.

Word Count: **5,967**

# REFERENCES

*Airbnb Announces Second Quarter 2021 Results* (2021). Available at: https://investors.airbnb.com/press-releases/news-details/2021/Airbnb-Announces-Second-Quarter-2021-Results/ (Accessed: 20 August 2021).

Allaire, J. J. (no date) *TensorFlow for R*. Available at: https://tensorflow.rstudio.com/tools/tfruns/overview/ (Accessed: 23 August 2021).

Berhane, F. (2018) 'Extreme Gradient Boosting with R', *DataScience+*, 5 March. Available at: https://datascienceplus.com/extreme-gradient-boosting-with-r/ (Accessed: 23 August 2021).

Boehmke, B. (2021) *Deep Learning with Keras and TensorFlow in R*. rstudio::conf 2020. Available at: https://github.com/rstudio-conf-2020/dl-keras-tf/blob/c0ead47e32a94566092bb9f4908f56e70e21a646/materials/99-extras/imdb-grid-search.R (Accessed: 22 August 2021).

Breiman, L. (2001) 'Random Forests', *Machine Learning*, 45(1), pp. 5–32. doi: 10.1023/A:1010933404324.

Buuren, S. and Groothuis-Oudshoorn, C. (2011) 'MICE: Multivariate Imputation by Chained Equations in R', *Journal of Statistical Software*, 45. doi: 10.18637/jss.v045.i03.

Chakraborty, S. *et al.* (no date) *RPubs - New York AirBnB - Regression Analysis, Visualization and Modelling*. Available at: https://rpubs.com/SayakChakraborty/NewYorkAirBnB_Modelling (Accessed: 23 August 2021).

Chen, T. and Guestrin, C. (2016) 'XGBoost: A Scalable Tree Boosting System', *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794. doi: 10.1145/2939672.2939785.

Choudhary, P., Jain, A. and Baijal, R. (no date) 'Unravelling Airbnb Predicting Price for New Listing', p. 10.

Crawley, M. J. (2012) *The R Book*. Hoboken, UNITED KINGDOM: John Wiley & Sons, Incorporated. Available at:

http://ebookcentral.proquest.com/lib/londonww/detail.action?docID=1120574 (Accessed: 23 August 2021).

*Inside Airbnb. Adding data to the debate.* (no date) *Inside Airbnb*. Available at: http://insideairbnb.com (Accessed: 21 August 2021).

*Inside Airbnb: Rio de Janeiro. Adding data to the debate.* (no date) *Inside Airbnb*. Available at: http://insideairbnb.com/rio-de-janeiro (Accessed: 20 August 2021).

Lewis, L. (2019) *Predicting Airbnb prices with machine learning and deep learning*, *Towards Data Science*. Available at: https://towardsdatascience.com/predicting-airbnb-prices-with-machine-learning-and-deep-learning-f46d44afb8a6?gi=5c4009d72f9f (Accessed: 20 August 2021).

Long, J. D. and Teetor, P. (2019) *R cookbook: proven recipes for data analysis, statistics, and graphics*. Second edition. Beijing Boston Farnham Sebastopol Tokyo: O'Reilly.

Pofahl, A. (2017) *How Airbnb added BRL 2,5 billion to Brazil's GDP*, *LABS English*. Available at: https://labsnews.com/en/articles/business/how-airbnb-added-brl-25-billion-to-brazils-gdp/ (Accessed: 23 August 2021).

Rosebrock, A. (2016) 'A simple neural network with Python and Keras', *PyImageSearch*, 26 September. Available at: https://www.pyimagesearch.com/2016/09/26/a-simple-neural-network-with-python-and-keras/ (Accessed: 23 August 2021).

Saraswat, M. (no date) *Beginners Tutorial on XGBoost and Parameter Tuning in R Tutorials & Notes | Machine Learning*, *HackerEarth*. Available at: https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/beginners-tutorial-on-xgboost-parameter-tuning-r/tutorial/ (Accessed: 22 August 2021).

Wang, D. and Nicolau, J. L. (2017) 'Price determinants of sharing economy based accommodation rental: A study of listings from 33 cities on Airbnb.com', *International Journal of Hospitality Management*, 62, pp. 120–131. doi: 10.1016/j.ijhm.2016.12.007.

Witten, I. H. *et al.* (2011) *Data Mining: Practical Machine Learning Tools and Techniques*. San Francisco, UNITED STATES: Elsevier Science & Technology. Available at:

http://ebookcentral.proquest.com/lib/londonww/detail.action?docID=634862 (Accessed: 22 August 2021).

*XGBoost Parameters — xgboost 1.5.0-dev documentation* (no date). Available at: https://xgboost.readthedocs.io/en/latest/parameter.html (Accessed: 22 August 2021).