Price Predictions for AirBnB Listings in Buenos Aires:

Technical Report

Introduction

Data from AirBnB listings in Buenos Aires was taken from insideairbnb.com for listings in September 22, 2023. It contains data for all listings on AirBnB's website as of the specified date including many different features and amenities listed and also lists prices in the local currency, this in case the Argentine Peso. A dictionary of all the features and explanations of each within the original data[1].According to the company's request, the data was filtered for only showing listings that list a total number of guests accommodated between 2-6 because the company is looking to set prices of their new apartments that aren't on the market and want to compare them to small and mid-size listings. The code for the entire process can be viewed on github[2].

Data Decisions

The data pre-filtering for guests accommodated between 2-6 was n=29,346. Post filtering it was n=27,340 observations. The dataset itself was dirty and required significant cleaning with several decisions being made as to how.

The table below shows the number of null observations in descending order:

```
# Checking sum of all null values, sorted by descending order
df.isna().sum().sort_values(ascending=False).head(30)
```

neighbourhood_group_cleansed	27340
bathrooms	27340
calendar_updated	27340
license	26932
host_about	12298
neighbourhood	12195
neighborhood_overview	12195
host_neighbourhood	9052
host_location	6300
review_scores_checkin	5022
review_scores_cleanliness	5022
review_scores_accuracy	5021
review_scores_location	5021
review_scores_value	5021
review_scores_communication	5020
first_review	4969
last_review	4969
reviews_per_month	4969
review_scores_rating	4967

The bathrooms column which indicates how many bathrooms the listing has available is completely null in this dataset. That would otherwise be an important data point for analysis but given that it is completely full of null values, the column will be dropped along with all other columns that are fully or mostly null. The other columns that have many null values include columns with information/data about the host of the listing and not information about the accommodation itself. This includes the neighbourhood the host has reported they reside in, and information about the host. What may be important, however, is the location of the host (`host_location`) so that was be kept. There are other factors within the data concerning the host that are more impactful for analysis, such as whether the host is a superhost, therefore these columns will be dropped as well. Null values seen within `host_is_superhost` can be seen as the host not being a superhost. Given as it is a desirable status symbol on AirBnB to be a superhost, if the value is null it can be assumed that it is 0. Null values in that column will be imputed as 0 with a flag variable created. Also during this phase the columns containing the word 'neighbourhood' were renamed to 'neighborhood' for consistency.

The dataset was filtered further to account for only entire homes/apartments (classified together within the data) and shared rooms. This meant excluding hotels (n=56) from the analysis.

Mean values were imputed to columns that were null/missing from the data when number of reviews was equal to 0 with their respective means per number of guests accommodated (meaning imputing the mean for a listing that accommodates 4 people with the mean review scores for existing reviews amongst other listings that accommodates 4 people). For all values that were imputed flag variables were created. This was performed for the following columns:

---

[1] https://docs.google.com/spreadsheets/d/1iWCNJcSutYqpULSQHlNyGInUvHg2BoUGoNRIGa6Szc4/edit#gid=1322284596

[2] https://github.com/nxfern/DA3_Assignment_2/blob/main/Assignment_2_NF.ipynb

- ‘host_acceptance_rate’
- ‘host_response_rate’
- ‘bedrooms’
- ‘beds’
- All columns that began with ‘review_scores’

Listings that had a number of reviews all time equal to 0 were classified as a new listing for this review and given the flag ‘f_new_listings’.

The bathrooms_text column was restructured to be a numeric column. A dummy variable was added prior to restructuring to indicate whether or not the listing contains a ‘shared bath’ with the dummy variable ‘d_shared_bath’.

The amenities column contained a text dump of all the amenities listed within the observation. This column was parsed out via text searches in pandas and several dummy variables were created as a result:

- ‘d_luxury’ – Denoted as a 1 if the listing contained one of the following amenities:
  - Pools
  - Pool Tables
  - Hot Tubs
  - Saunas
  - Dishwashers
  - Grills
  - Rooftops
  - Premium Cable
  - Gyms
- ‘d_patio’ – Denotes whether the listing has a patio
- ‘d_toaster – Whether the listing has a toaster
- ‘d_microwave’ – Whether the listing has a microwave
- ‘d_bidet’ – Whether the listing has a bidet
- ‘d_skyline’ – Whether the listing has a view of the city skyline
- ‘d_pets’ -Whether the listing indicates that pets are permissible

Additional dummy variables were created for comparison, listed in the image below:

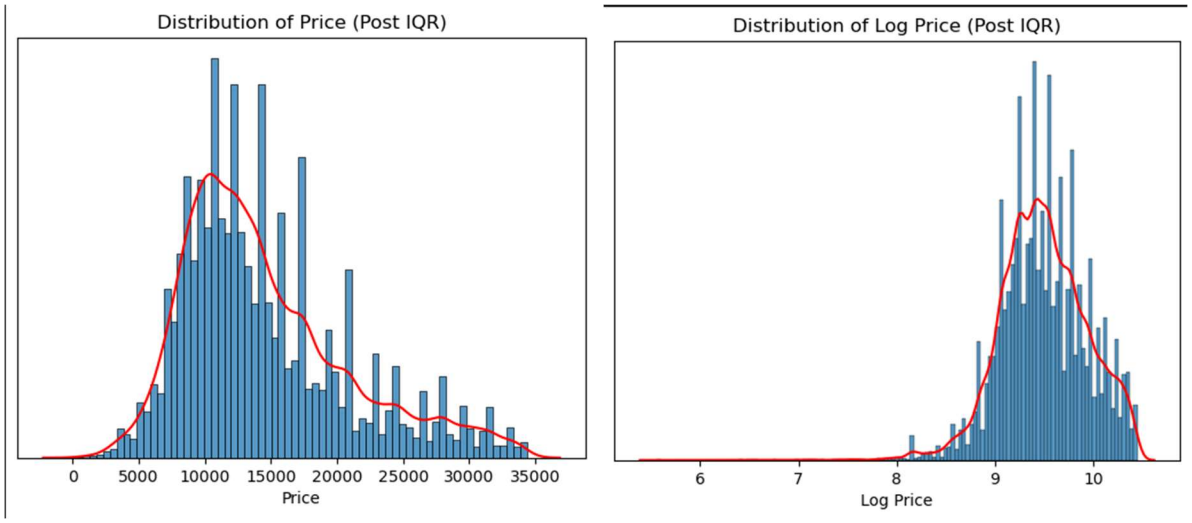
```
# Dummy variable for private rooms
df['d_private_room'] = (df.room_type == 'Private room').astype(int)

# Dummy variable for high review scores
df['d_high_review_score'] = (df.review_scores_value >= 4.8).astype(int) # Review of data hs 4.8 marker has returning 45.6% of observations (n=26970)

# Dummy variable for more than 3 nights minimum stay required
df['d_minstay_over_3'] = (df.minimum_nights > 3).astype(int)

# Dummy variable for more than 1 review in last 30 days
df['d_mult_reviews_l30d'] = (df.number_of_reviews_l30d > 1).astype(int)

# Dummy variable for high amount of reviews (over 30)
df['d_high_review_cnt'] = (df.number_of_reviews > 30).astype(int)

# Dummy variable for a listing in high demand
df['d_high_demand'] = (df.availability_30 <= 5).astype(int)

# Dummy variable for whether the host is local
df['d_local_host'] = df.host_location.str.contains('buenos aires', case=False).astype(int)

# Dummy variable for whether the host has multiple listings
df['d_host_many_listings'] = (df.host_total_listings_count >= 5).astype(int)

# Dummy variable for host having multiple listings in Buenos Aires
df['d_host_many_local_listings'] = (df.host_total_listings_count >= 5).astype(int)

# Dummy variable for if the neighborhood is in Palermo
df['d_palermo'] = (df.neighborhood_cleansed == 'Palermo').astype(int)

# Dummy variable for if the neighborhood is in Recoleta
df['d_recoleta'] = (df.neighborhood_cleansed == 'Recoleta').astype(int)

# Dummy variable for if the neighborhood is in Puerto Madero, a very expensive neighborhood
df['d_puerto_madero'] = (df.neighborhood_cleansed == 'Puerto Madero').astype(int)

# Dummy variable to say an apartment is small, defined by listings that accommodate 3 or fewer people
df['d_small_size'] = (df.accommodates <= 3).astype(int)
```

After data cleaning and feature engineering descriptive statistics were run on price. It showed a mean price of 23,371.014 with an extremely large standard deviation of 320,394.898. The decision to perform IQR on price to exclude extreme values was made by determining that any value greater or less than 1.5 times the value of either the 25% or 75% quartiles can be considered as extreme values to be dropped. Descriptive statistics were run post-IQR and provided a much more reasonable standard deviation of 6,333.464 with a total of n=2155 observations dropped. The comparison of descriptive tables can be seen below:

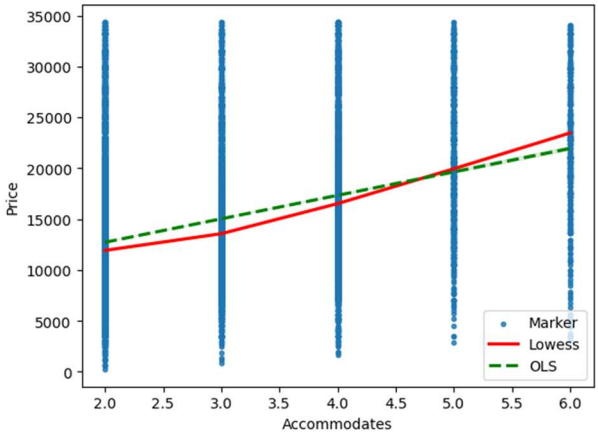
```
count      26970.000
mean       23371.014
std       320394.898
min          260.000
25%        10301.000
50%        13753.000
75%        19951.000
max     35002562.000
Name: price, dtype: float64
```

```
count    24815.000
mean     14546.011
std       6333.464
min        260.000
25%      10064.000
50%      13046.000
75%      17501.000
max      34352.000
Name: price, dtype: float64
```

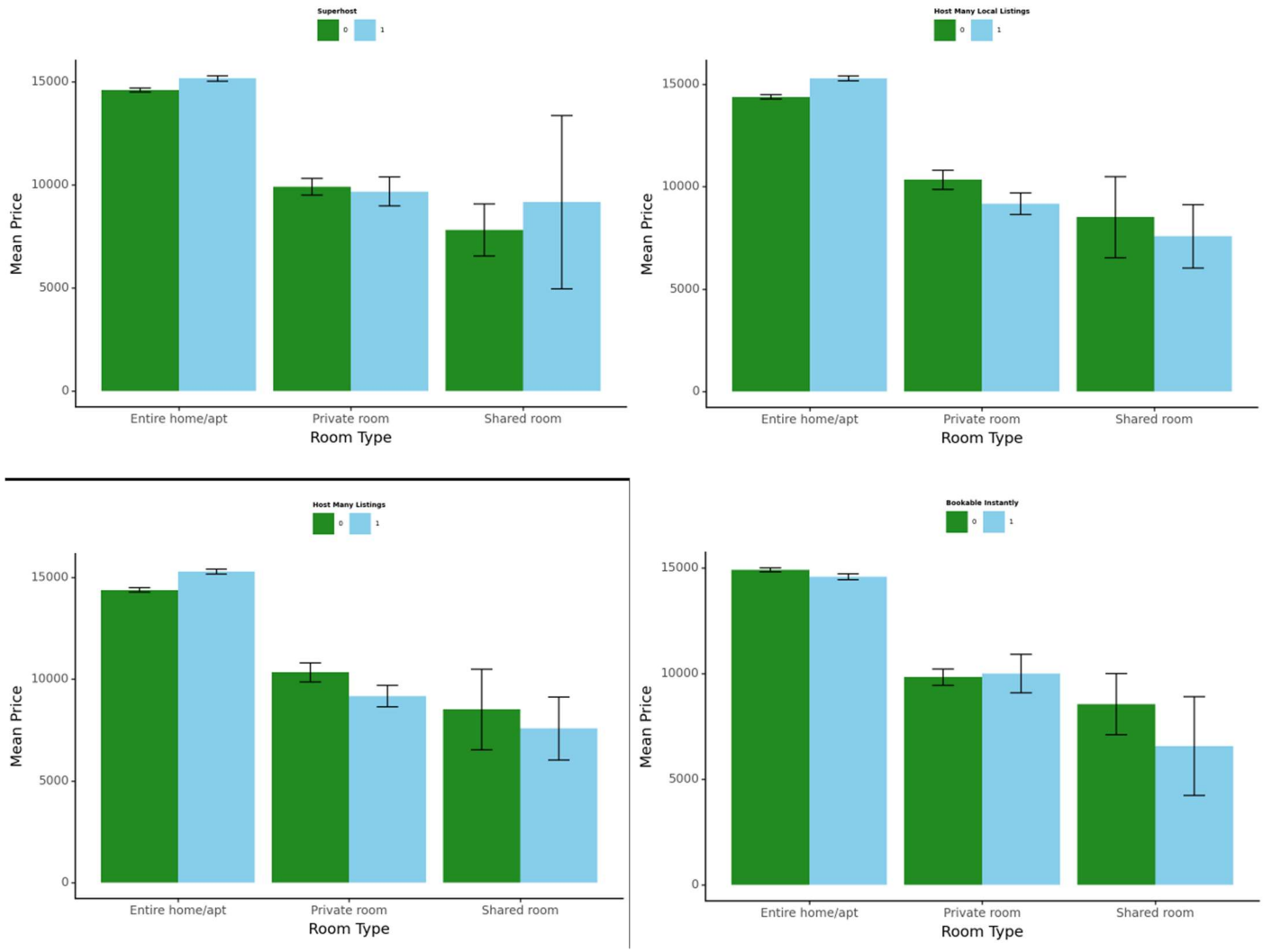
A comparison of the distributions of price post IQR was also made to determine whether to transform price into it's log values for better predictions however it was determined that the log transformation did not make an improvement and that the level distribution was close enough to normal, therefore no transformation was made. Graph comparisons are below:



Another plot was created to check for any potential non-linearity in the data however it did not show any particular non-linearity compared to a basic OLS plot model when examining price according to number of guests a listing accommodates.



Several plots were created to check interactions between certain explanatory variables. Examples are shown in the figures below:



For each of the figures above an interaction was plotted out between a categorial variable, in this case 'room_type' and a binary/dummy variable to check to see whether or not an interaction exists. With all of the figures shown above, confidence interval in values for the shared rooms is very likely due to the low amount of observations in the data and not due to any particular strong interaction term between any of the dummy variables. That being said, a noteworthy interaction can be seen when checking if the host is a superhost and whether the listing is instantly bookable, meaning that no contact with the host is needed. It can be inferred that a host being a superhost can give a user a level of confidence that the host is more trustworthy than others while a listing being instantly bookable makes it much easier and more reliable to book a stay which could be more appealing to a particular user looking for convenience and/or a streamlined experience.

Lastly, all variable naming conventions followed the following metrics:

- Categorical variables (such as room_type) were given the 'c_' label at the start of the variable name
- Numerical variables (such as accommodates) were given the 'n_' label at the start of the variable name
- All dummy variables that were created were given the 'd_' label at the start of the variable name
- All flag variables created were given the 'f_' label at the start of the variable name

Models

Four models were created to compare them to each other and provide the best predictions. A work set and a holdout set were created for training the models at an 80/20% split along with setting up cross validation with 5 folds. All models used 'accommodates' as the main explanatory variable. RMSE values were taken from both performances on the work set and the holdout set for comparison. The work set contained an about of n=19,853 observations and the holdout set n=4,692 for comparison. The basis of the comparison point was to create two OLS based models and two Machine Learning models to see how their predictions compared against each other in an attempt to make the best model for predicting price on live data.

Simple Linear OLS

Hand-made thought-out OLS model being made for comparison sake. The model uses `n_accommodates` as the main explanatory variable various categorical and binary/dummy variables being used as well to build the model. The reasoning behind the selections on the price are as follows:

- `c_room_type`, `n_bathrooms`, `n_bedrooms`, and `n_beds` were included because the number of each can sway a person's decision to select one listing over another
- `n_has_availability`, `n_availability_30`, `n_instant_bookable`, `d_high_demand`, and `d_minstay_over_3` were chosen because how soon a listing is available for, for how long a person must rent it for, and how in demand said listing is can affect how often a listing is selected
- `n_host_has_profile_pic`, `n_host_identity_verified`, `n_host_is_superhost`, `d_local_host`, `n_host_response_rate`, and `d_host_many_listings` are all variables to account for iformation about the host potentially influencing the price. If a host is viewed as undesirable in any way it could be likely to negatively affect how a listing performs on the market. This is also meant to account for a particular host that has many listings compared to another that has relatively few
- `c_neighborhood_cleansed`, `d_palermo`, and `d_recoleta` are to account for the neighborhoods. Palerma and Recoleta are two of the fancier and most popular neighborhoods so dummy variables were incuded in the model to account for those
- `n_review_scores_rating` and `d_high_review_score` are to account for review scores influencing price
- `n_number_of_reviews` and `d_high_review_cnt` are to account for the number of reviews of a listing, set a dummy variable for any listing that is considered to have a high amount of reviews
- `d_luxury`, `d_microwave`, `d_patio`, `d_pets`, `d_private_room`, `d_shared_bath`, and `d_toaster` are all dummy variables to account for certain features of a listing, whether it be certain amenities or whether pets are permissable
- `f_new_listings` was included to account whether the listing in question was identified as a new listing or not

For the interaction terms used, they are explained by certain factors potentially having effects on both `room_type` and `accommodates`, for instance the listing of a room type being likely dependent on whether the host is classified as a superhost, whether pets are allowed, whether there are shared baths, and so forth. Same for the number of accommodations in the listing.

LASSO

A LASSO model was created using lambda turning. All the parameters from the hand-made OLS model prior along with the following additional explanatory variables and all dummy variables were used with the LASSO algorithm to allow for LASSO to determine which features were important or not:

- n_host_total_listings_count – count of total listings a host has
- n_calculated_host_listings_count – a recalculated count of total listings a host has
- n_minimum_nights – the number of minimum nights a user must book the particular listing for
- n_maximum_nights – the maximum number of nights a user can book for
- n_maximum_minimum_nights – the largest minimum_night value from the calendar looking 365 days ahead
- n_host_acceptance_rate – the rate at which a host accepts a booking
- c_host_response_time – how long the user takes to response, listed as a categorical value because of descriptions such as “Under an hour” or “Over a day”
- n_reviews_per_month – the amount of reviews a listing has on average in a month
- c_host_location – the reported location of the host (Buenos Aires, New York, etc.)

The LASSO model, being fed these variables, was fairly slow compared to the runtimes of other models, taking upwards of 8 minutes to complete the cross validations and provide the best model. The best estimator from LASSO ended with 370 coefficients and a lambda of .25.

Random Forest (RF)

A Random Forest model was created using all the values from the LASSO model without the dummy variables, interaction terms, and the flag variable for new listings. The tuning parameters selected for the Random Forest model was determined by the following code:

```
# Finding the recommended number of variables to set for the Random Forest using math library
print('The theoretical recommended number of variables: {:.2f}.'.format(math.sqrt(len(X.design_info.column_names))))
```

```
The theoretical recommended number of variables: 24.64.
```

From there, the features set for the random forest model were +/- ~5 from that value and were the following: 20, 23, 27, 30 with the sample leafs set to 5, 10, and 15.

The model took approximately 2 minutes to complete returning the following results:

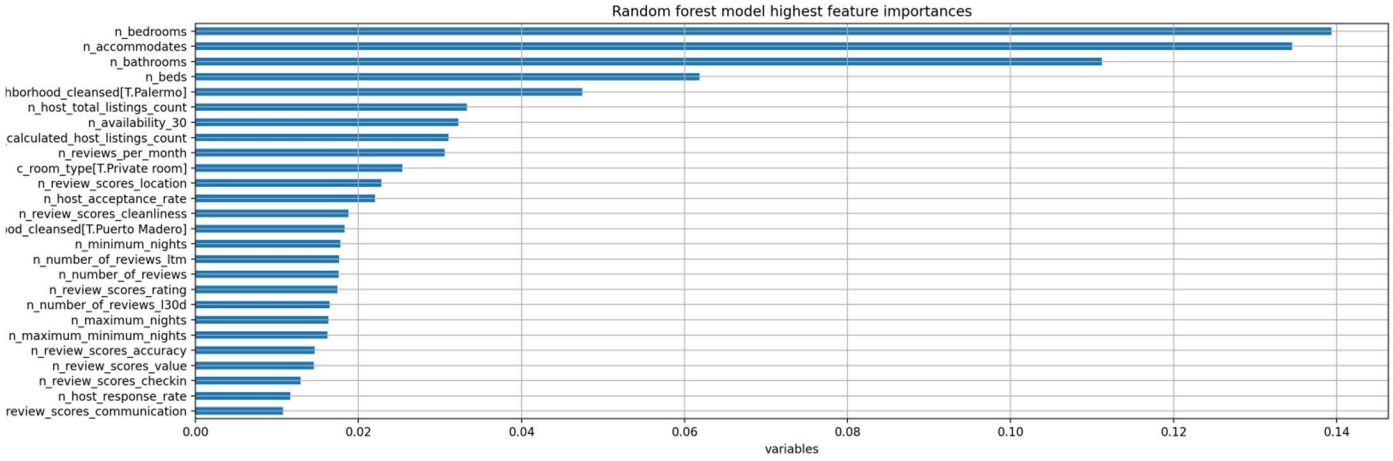
min node size	5	10	15
max features			
20	5567.430	5698.000	5789.310
23	5481.240	5640.100	5705.380
27	5379.230	5489.170	5588.330
30	5306.910	5426.820	5500.070

```
# Printing best RMSE score
rf_rmse = abs(rf_model.best_score_)
print(rf_rmse)

# Displaying best estimator
rf_model.best_estimator_
```

```
5306.905986563872
```

```
RandomForestRegressor(max_features=30, min_samples_leaf=5,
                      random_state=20240210)
```

The best RMSE from the RF model had a maximum feature count of 30 parameters and a minimum amount of samples in each node being 5. Within this model it was instructive to take a look at what predictors the algorithm identified were most important, excluding parameters that had 1% or less significance compared to other parameters:



From the graph we can see that bedrooms, accommodates, and bathrooms are graded out as the most important features. After that the number of beds is important, with also a listing being in the Palermo neighborhood. Beyond that most of the remaining features are similar in importance.

Gradient Boosting Machine (GBM)

A GBM model was created using the exact same variables from the RF model. The GBM model used was a naïve tuning parameter to cutdown on the required computational power. The model took approximately 30 seconds to complete, making it the fastest of the models outside of the hand-made OLS model. The maximum features selected for the model was set to 30 to mirror the Random Forest model with a maximum depth set to 10 nodes and a minimum sample split of each node set to 10 observations. The results of the best estimator of the GBM model is shown below:

```
GradientBoostingRegressor(max_depth=10, max_features=30, min_samples_split=20,
                          n_estimators=300)
```

Results and Conclusion

The RMSEs from all models from both the work set and holdout set are shown in the table below:

	model	Workset RMSE	Holdout RMSE
0	OLS	5230.963	5054.841
1	LASSO	5227.052	4820.723
2	RF	5306.906	4236.691
3	GBM	4743.295	1867.580

For each model outside of the hand-made OLS model, best estimator was used to generate the RMSE values on both the work set and holdout set, the latter of which was used for making a prediction on the holdout set.

From these results the GBM model has by far the lowest RMSE on both sets. The hand-made OLS model performs the worst, which implies that there is a fair amount of variance that is not being accounted for properly in the model. The LASSO model performs better on the work set than the Random Forest model but performs worse on the holdout set while also taking longer to process than any of the other models chosen.

The holdout set can be construed as the best attempt to simulate live data, and the model that gives the best prediction RMSE can be the best predictive model. The GBM model performs the best on the work set and by far the best on the holdout set. It appears that gradient boosting for this dataset is far and away the superior model for running predictions with live data. The RMSE values being as high as they are fine given the mean price value being 14,546.01. Additionally, the GBM model takes little time to reproduce without tradeoff.

Using the GBM model you can then view the predicted prices and determine prices for new listings depending on the neighborhood the listing is in and whether the listing is small or not. See the following table below:

c_neighborhood_cleansed	d_small_size	RMSE	Avg Pred Price	rmse_norm
Belgrano	0	1643.550	17929.570	0.090
	1	1978.320	13355.760	0.150
Palermo	0	2062.820	19685.090	0.100
	1	2099.870	14753.640	0.140
Puerto Madero	0	1063.090	27514.460	0.040
	1	1372.610	23506.110	0.060
Recoleta	0	1848.760	18994.600	0.100
	1	2013.460	13773.660	0.150
Retiro	0	1638.360	17419.170	0.090
	1	1551.580	13447.450	0.120
San Nicolas	0	1654.520	16338.580	0.100
	1	1680.760	11784.750	0.140

In the table above a 1 indicates that the apartment is small for each respective neighborhood with the mean RMSE value for each, the average predicted price by the GBM model, and rmse_norm being the normalized RMSE values. Based on viewing the rmse_norm values you can see what a competitive price may be for a new listing with the lower the normalized RMSE score the better the model does in predicting the price.