Price Prediction for Airbnb Listings in San Francisco

Introduction

a) **Problem Statement:**
Airbnb is an online marketplace for owners to list their properties so that guests can rent to stay for lodging and tourism activities. Even though Airbnb provides general guidance currently there is no free service that will help owners to set the prices.

b) **Goal:**
Our goal is to develop a machine learning model that will help owners to set the prices for their properties using the pricing data for other listings in San Francisco.

Dataset

a) **Listings:**
The detailed information for each listing in San Francisco is obtained from Inside Airbnb website in csv format. The data is scraped from the Airbnb website on 03-02-2021. The dataset contains 6883 rows with 73 attributes and loaded as df. The definition of each attributes that is used in analysis can be found in Data Wrangling section.

b) **Calendar:**
Calendar data contains the pricing information at the date of scraping for each listing for the next year. The dataset contains 2512295 rows with 7 attributes and loaded as calendar. The definition of each attributes that is used in analysis can be found in Data Wrangling section.

Data Wrangling

a) **Listings**
Attributes:
Definitions of the attributes that are used in the model are listed below.

host_id: Id of the host
host_since: registration date of the host
host_response_time: Response time of the host
host_response_rate: Response rate of the host
host_acceptance_rate: Acceptance rate by the host
host_is_superhost: Is host a super host
host_listings_count: Number of listings that the host owns
host_total_listings_count: Number of listings that the host owns

host_verifications: Methods of verifications that host has done
host_has_profile_pic: Host has profile picture or not
host_identity_verified: Host has verified identity or not
neighbourhood: neighborhood information
neighbourhood_cleansed: neighborhood information
neighbourhood_group_cleansed: neighborhood information
latitude: latitude of the listing
longitude: longitude of the listing
property_type: sub-groups of room_type column
room_type: entire home, private room, shared room or hotel
accommodates: how many guests can a listing accommodate
bathrooms_text: number of bathrooms
bedrooms: number of bedrooms
beds: number of beds
amenities: amenities in the listing
price: price of the listing at the time of scraping
minimum_nights: minimum nights a listing available
maximum_nights: maximum nights a listing available
has_availability: listing is available or not at the time of the scraping
availability_30: availability in next 30 days
availability_60: availability in next 60 days
availability_90: availability in next 90 days
availability_365: availability in next 365 days
number_of_reviews
number_of_reviews_ltm: number of reviews lifetime
number_of_reviews_l30d: number of reviews last 30 days
first_review: date of first review
last_review: date of last review
review_scores_rating
review_scores_accuracy
review_scores_cleanliness
review_scores_checkin
review_scores_communication
review_scores_location
review_scores_value
instant_bookable: instant bookable or not
calculated_host_listings_count: Number of listings that the host owns
calculated_host_listings_count_entire_homes: Number of entire homes that the host owns
calculated_host_listings_count_private_rooms: Number of private rooms that the host owns
calculated_host_listings_count_shared_rooms: Number of shared rooms that the host owns
reviews_per_month

Dropping the attributes that will not be used in analysis:

Several columns are dropped from the data frame since they contained irrelevant information or they had too many missing values (all or >50%).

Investigation of individual attributes:

***General procedure for individual attributes, details can be found in the jupyter notebook.

1. host_listings_count, host_total_listings attributes are dropped since they contain wrong information.
2. minimum_nights and maximum_nights attributes are kept and other similar attributes are dropped due to the high similarity.

3. host_since is converted to datetime and created a new attribute as host_days_active.
4. host_response_time contains several text entries and some missing data. Missing data is filled as “unknown”.
5. host_response_rate & host_acceptance rate is converted to numeric type, binned into 4 categories and missing data is filled as “unknown”.
6. host_is_superhost: The t, f entries are converted to 1 and 0s.
7. host_verifications has various verification methods. To simplify the data, entries that contain government_id is encoded as 2, other methods as 1 and if there is no verification it is encoded as 0.
8. host_has_profile_pic, host_identity_verified: The t, f entries are converted to 1 and 0s.
9. neighbourhood, neighbourhood_cleansed: The neighbourhood column contains cities and it is too broad. neighbourhood_cleansed column has more granular neighbourhood information. neighbourhood column will be dropped and neighbourhood_cleansed column is renamed as neighbourhood.
10. latitude, longitude columns have been assigned to different data frame.
11. room_type, property_type: Entries in property_type column are sub-groups of entries in room_type column. This column will be dropped. There are 4 unique entries in room_type. room_type column will be renamed as property_type.
12. bathroom_text: The bathrooms_text column contains strings such as '1 bath', '1 private bath', '1 shared bath'. Letters are removed and numbers are converted to numeric type.
13. accommodates,bedrooms, beds: The missing values are filled using median.
14. price: $ sign is removed and converted to numeric type.
15. amenities: Each listing has amenities entered as string. New attributes are created based on the important amenities. 1 or 0 entered into new attributes depending on the presence of the attribute in the amenities column.
16. has_availability, has_avalilability_X: The has_availability column is dropped. Has_availablity_60 column is kept and others are dropped due to high correlation between them.
17. number_of_reviews,number_of_reviews_ltm,number_of_reviews_l30d: Only number_of_reviews column is kept and others are dropped due to high correlation.
18. first_review,last_review: Both columns are converted to datetime format and two new attributes are created as since_first_review, since_last_review by subtracting from 03-02-2021. Both of these columns are binned and first_review, last_review columns are dropped.
19. reviews: The missing values are filled with median and binned into four categories.

b) Calendar

Attributes:

listing_id: id of the listing
date: date of the pricing
available: listing available or not
price: price of the listing
adjusted_price: price of the listing
minimum_nights: minimum nights a listing available
maximum_nights: maximum nights a listing available

Investigation of individual attributes:

1. date: Converted to datetime format.
2. available: The t,f are converted to 1,0s.
3. price: $ sign is removed and column is converted to numeric type.
4. adjusted_price: $ sign is removed and column is converted to numeric type.

Exploratory Data Analysis and Initial Findings:

a) Price vs property type:

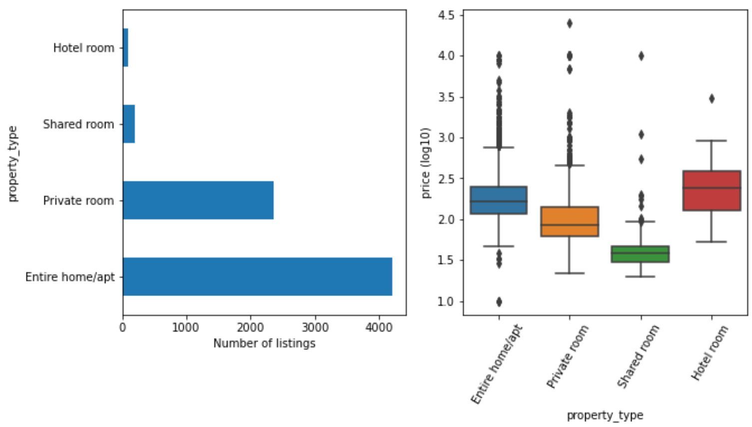

Figure 1: Number of listings vs property type (left), Price (log10) vs property type

Majority of the listings consist entire home/apartments. Hotel rooms have the hi ghest median price per night followed by entire home/apartments.

b) Price vs Accommodates:

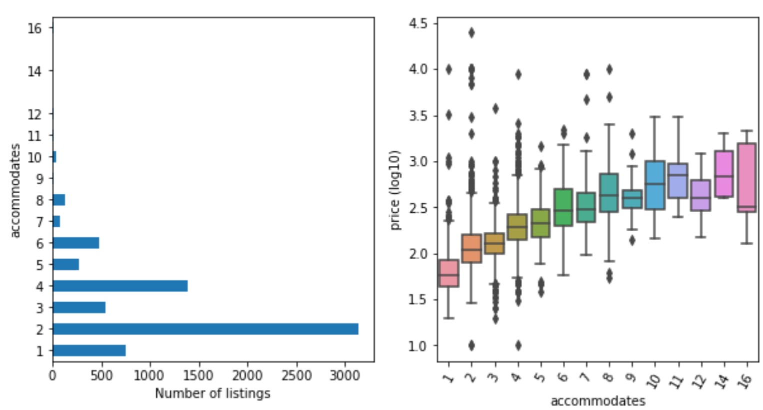

Figure 2: Number of listings vs accommodates (left), Price (log10) vs accommodates

The price of listings increases as the number of people it can accommodate. Majority of the listings can accommodate 3 or less guests.

c) Price vs Neighborhood:

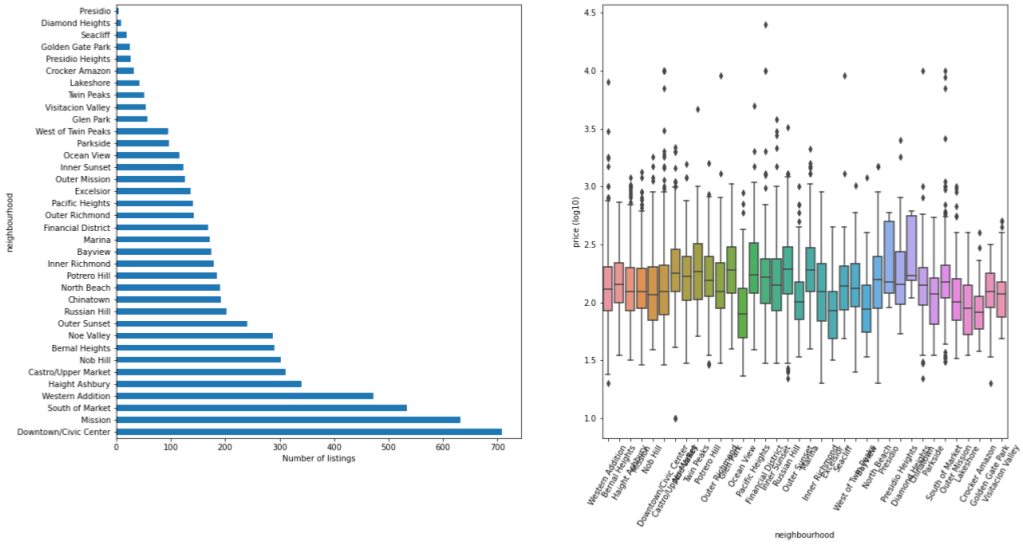

Figure 3: Number of listings vs neighbourhood (left), Price (log10) vs neighbourhood

Prices are similar among the neighborhoods in the San Francisco. Downtown/Civic Center has the most of the listings.

d) Price vs Date:

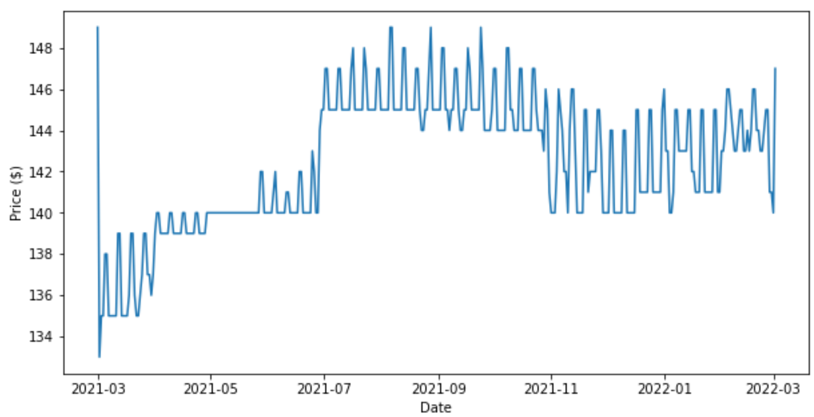

Figure 4: The median of listed prices between 03-02-2021 and 03-02-2021.

The listing prices have increased in summer months and show spikes that correspond to weekends.

Model Selection and Optimization

Three models will be compared for the price prediction.

1. Linear Regression
2. Random Forest Regressor
3. XGBoost Regressor

a) Preprocessing of the Data

Since XGBoost and Random Forest are both tree-based models, standardization or normalization of the data is not expected to make a difference on model performance. However, it would be impactful on linear regression. Thus, data will be normalized using MinMaxScaler.

	Training Data		Test Data	
Model	R2	Cross-Validation	R2	Cross-Validation
Linear Regression	0.11	0.11,0.14,0.11, 0.12,0.11	0.02	0.09,0.07,0.08, -0.04,0.09
XGBoost Regressor	0.99	0.99,0.99,0.99,0.99,0.99	- 0.02	-0.08,0.17,0.54,-0.01,0.52
Random Forest Regressor	0.92	0.91,0.93,0.91,0.92,0.91	0.11	0.37,0.23,0.62,0.07,0.55

Table 1: R-squared scores for 3 different models.

Initial results reveal that XGBoost Regressor has overfit to the training data and performs poorly on the test data. Linear Regression model is slightly better and Random Forest Regressor provided the best results. I will focus on XGBoost Regressor and plot the actual vs predicted prices. In addition, I will create a new variable as %error that shows the error percentage between the actual and predicted prices.

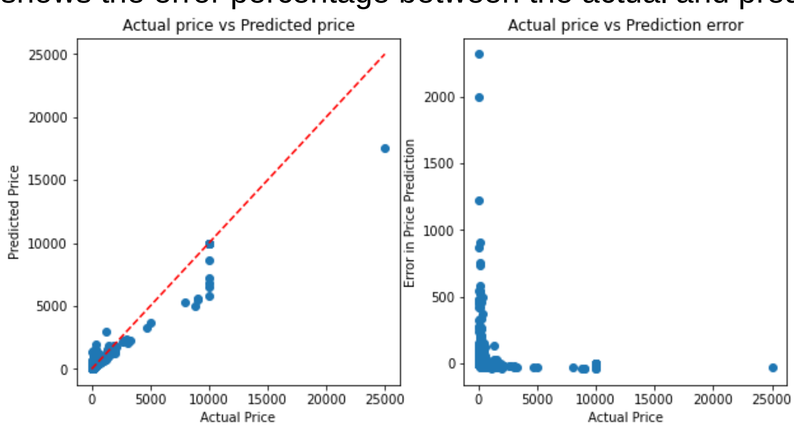

Figure 5: Actual price vs predicted price (left), Error in Price Prediction vs Actual Price (right) for the **training data**.

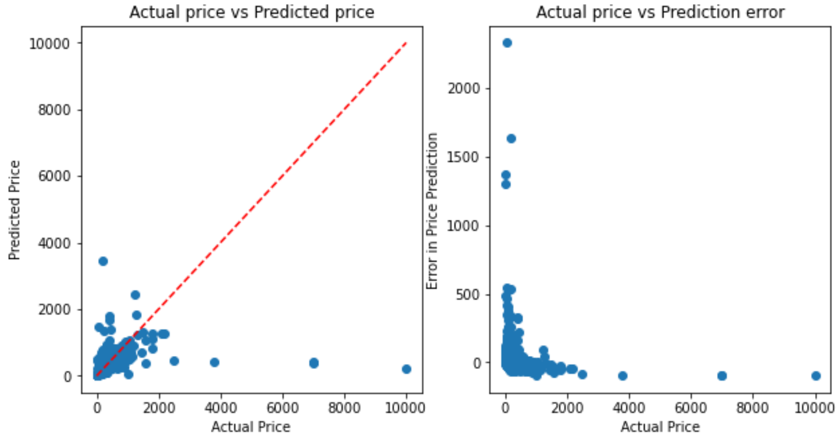

Figure 6: Actual price vs predicted price (left), Error in Price Prediction vs Actual Price (right) for the **test data**.

The error in price prediction is very high at lower prices for both training and test data. This indicates the presence of outliers in the dataset. I created a new column "price_per_accommodate" to normalize the listings price based on how many guests it can accommodate. Here is the summary of the column:

```
mean       77.494464
std       289.329700
min         2.500000
25%        37.250000
50%        50.000000
75%        75.000000
max     12500.000000
```

This is interesting since minimum price per guest is $2.5 which is extremely low in San Francisco. The outliers (lower 2.5% and upper 2.5%) are filtered out and models are trained with the filtered dataset.

	Training Data		Test Data	
Model	R2	Cross-Validation	R2	Cross-Validation
Linear Regression	0.53	0.54, 0.54, 0.54, 0.52, 0.55	0.53	0.52, 0.52, 0.47, 0.56, 0.48
XGBoost Regressor	0.94	0.96, 9.96, 0.95, 0.96, 0.96	0.66	0.58, 0.65, 0.59, 0.65, 0.62
Random Forest Regressor	0.95	0.94,0.94,0.95,0.94,0.94	0.65	0.59,0.60,0.59,0.66,0.61

Table 2: R-squared scores for 3 different models following removal of outliers.

Removal of the outliers improved the model performance. Hyperparameter tuning is performed for XGBoost Regressor using GridSearchCV. R2 for training data is 0.85 and test data is 0.67. Let's plot the Actual price vs predicted price and Error in Price Prediction vs Actual Price again.

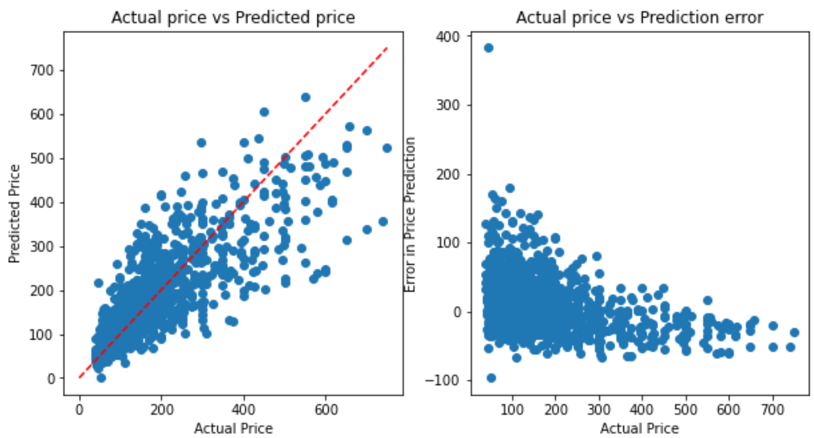

Figure 7: Actual price vs predicted price (left), Error in Price Prediction vs Actual Price (right) for the **test data** following hyperparameter tuning.

Lower prices tend to be predicted higher and higher prices are tend to be predicted lower than their true values. Is it possible to improve model performance by binning listings into different price ranges and optimize the model for each price range? Let's look into error distribution for several price ranges: 0-199, 200-499, 500-maximum

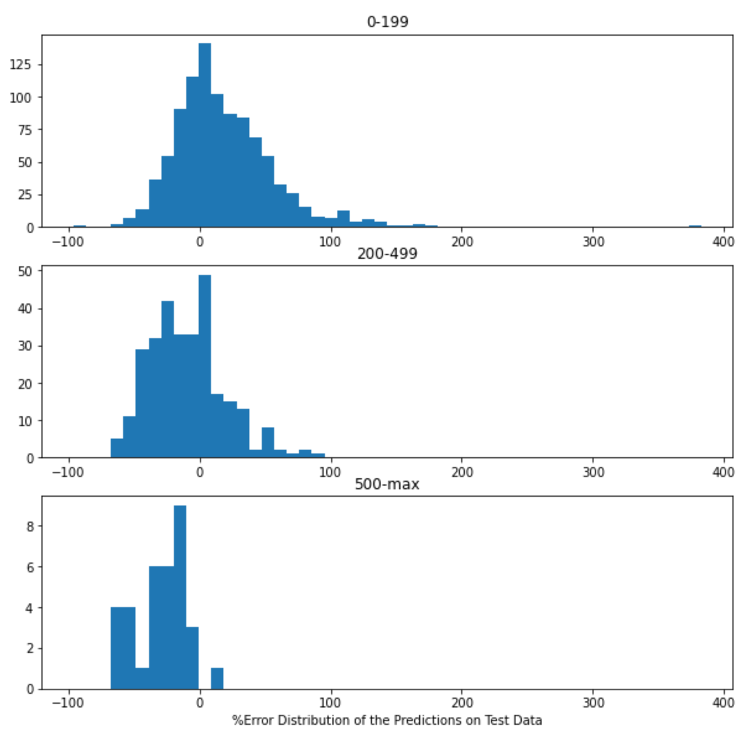

Figure 8: %Error distribution on price prediction with respect to price range ($0-199, $200-499, $500-max)

The error distribution tends to shift from positive to negative when the price is around $200. Dataset is divided into two; price of listings more than $200 and less than $200. Model is trained for each dataset. R-square obtained for the model trained on data frame that contains prices lower than 200 USD is very similar to the model trained on the whole data frame. Whereas R-square for higher prices are significantly lower. This could be for two reasons:

- The default parameters of the model is not suitable for this data frame so hyperparameter tuning can improve it.
- Size of the data frame for higher prices is small (~25% of the other data frame) so we may not have enough data points.

Following the hyperparameter tuning R2 for the test data has improved but still quite low. The poor performance of the model is likely to be result of small size of the dataset.

	Training Data		Test Data	
Dataset	R2	Cross-Validation	R2	Cross-Validation
Price>200	0.78	0.81,0.80,0.81,0.80,0.80	0.29	0.31,0.37,0.34,0.38,0.41
Price<200	0.88	0.89,0.89,0.89,0.89,0.89	0.65	0.66,0.63,0.64,0.64,0.63
Combined	0.85	0.86,0.87,0.87,0.87,0.87	0.67	0.63,0.65,0.65,0.67,0.65

Table 3: R-squared scores for XGBoost Regressor for 3 different models following hyperparameter tuning.

Summary

I have tested several models for predicting the price of the AirBnb listings. XGBoost Regressor yielded the best performance. Hyperparameter tuning of the model slightly improved performance. I have observed that the model over predicts the price for low price range and under predicts the price for the upper price range. Thus, I divided the data frame based on price range. Following the hyperparameter tuning, model performance for upper price range was significantly lower than the whole dataset. This could be due to limited amount of data for the upper price range.