Capstone 1 – Final Report

Developing a Machine Learning Model to Predict Airbnb Listing Prices

Source Code can be found here in my Github repository

Author: Zach Palamara

Project Aim and Background

Founded in 2008, Airbnb is a growing and popular alternative to traditional lodging. It is essentially an open market where as a host, you can list a spare room, entire apartment or home for prospective travelers to book and use as a domicile during a trip. Currently there are no free resources for hosts to use when trying to price their Airbnb listing on the open market. Since the amount of listings for Airbnb rentals has exploded over the last several years, being able to competitively price your rental space is a must in order to maintain a high booking rate and rate of return for your property. My goal is to develop a tool for hosts to use to gain insight on how to appropriately price their Airbnb rental space in the market they reside in.

Target Audience

I developed this model using a dataset from Austin, TX. This report will be especially insightful for hosts in the Austin, TX area looking to competitively price their listings as well as existing hosts who may want to revisit their pricing strategy to optimize returns.

Sourcing The Data

I was able to acquire the data for my project from Inside Airbnb, which is an independant, non-commercial set of tools that allows you to explore how Airbnb is being used in various cities around the globe. They scrape publicly available data from Airbnb which includes listing descriptions, reviews, calendar data and much more. Inside Airbnb has pre-prepared .csv files as well as geo-spatial data in the form of geo-json files, which can be easily downloaded for free directly from their website. There is also another site, AirDNA, which scrapes higher quality data for a fee.

Data Cleaning and Preprocessing

I downloaded the Austin, TX airbnb dataset directly from Inside Airbnb, which includes updated information up to February 19th, 2020. This dataset consisted of 11,151 listings with 78 distinct features. I was easily able to load the data into a Pandas dataframe using Jupyter notebooks. I also dropped 31 initial features that seemed futile for the goal of the project.

Missing Data

After dropping my initial features, I wrote a simple “for” loop to print out the feature names and percentage of missing data for those features. Luckily it was evident that most of the features had either no missing data points or less than 25% of the data was missing. There were however a few features such as *monthly_price* and *square_feet* that were more than 90% missing, so I decided to drop those features. I also wrote a custom function that accepted a single feature name as an argument and returned the following: percent of data missing, the name and count of each unique value, and the number of unique values for that feature. I went through each feature and called this function to provide insight on how to intimately handle the cleaning process of each feature. Overall there were 6 features that had missing data points that I explicitly address. For the *beds*” (number of beds in the property) feature, rather than dropping any rows, I decided to fill in the missing data points with the median value because this is a rather important feature that will surely influence the pricing model. There were 4 categorical features that had Pandas default “NaN” values, which I decided to fill with an “unknown” string. Lastly, I filled in the missing data in the *security_deposit* feature with “0” since this was most likely the correct value.

Binary Features

Upon initial examination, I noticed that there were several columns that had either “t” or “f” strings to denote the binary condition for that particular feature. One of the first steps that I took was to convert these strings into numeric representations (‘t’ : 1 , ‘f ’: 0) so they could easily be used in machine learning models. I was able to accomplish this by leveraging the Pandas *df.replace()* function with a dictionary that correctly mapped these values to each other.

Date-Time Features

I converted all of the date features that were default strings into DateTime objects so that they could be easily manipulated in data analysis. Since I did this, I was able to manipulate two datetime features to create a new feature that represents the amount of days that the listing has been posted on Airbnb.

Categorical Features

There are a lot of categorical features within this dataset. I used “binning” techniques to cut down on the number of unique items within a lot of the features for easier implementation of machine learning models. One example of this was the *property_type* feature. There were about 31 unique descriptions that defined the type of property that the listing was. I cut down these unique features into the following 4 categories: House, Apartment/Condo, Other and Hotel. I was able to accomplish this by creating a dictionary that mapped the appropriate descriptions to these 4 new ones, and then I was able to use the pandas *str.replace()* method.

Numerical Features

I used some “binning” operations with a lot of the numerical features as well. For example, I decided to bin the *days_since_last_review* column into 5 distinct groups; ['0-6 months', '6-12 months', '1-2 years', '2-3 years', '4+ years']. This was also done in an effort to use a regression model later on.

Outliers

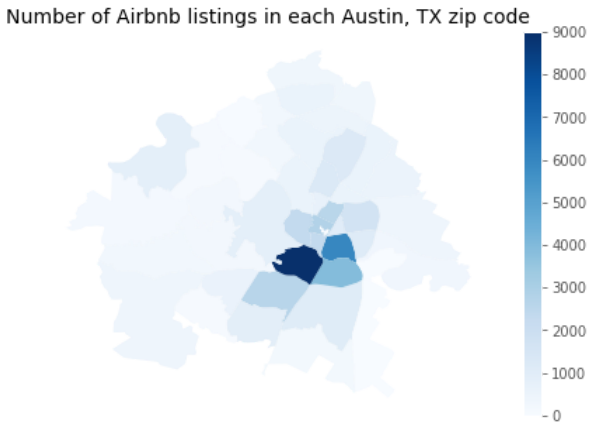
One of the most notable challenges in this dataset were the outliers within the *price* feature. The price per night feature ranged from $0 up to $3000+, and after viewing the histogram distribution, it became evident that most of the data resided in the $0-$250 range. I decided to only use the data within this range for further analysis since the outlying more expensive listings would probably not be a consistent representation of the patterns within the data. ONce the data was trimmed down, I decided to bin the remaining data into 5 distinct buckets using the Pandas *pd.cut()* method.

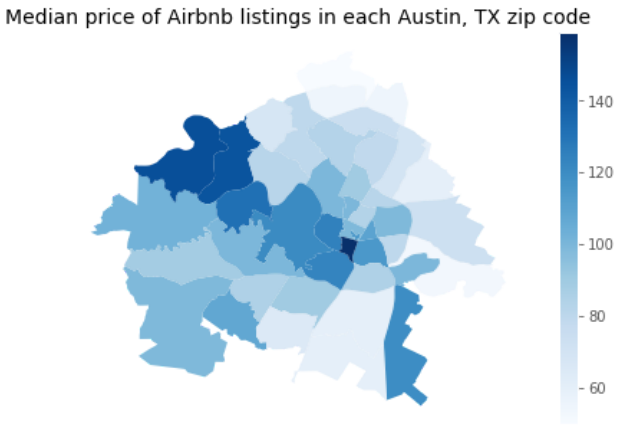
Data Story: Exploratory Data Analysis

After sourcing and wrangling the dataset, I came up with a series of questions to help further explore this data set and get a better understanding of the feature relationships.

Which areas/neighborhoods in Austin, TX are the most expensive and have the most listings?

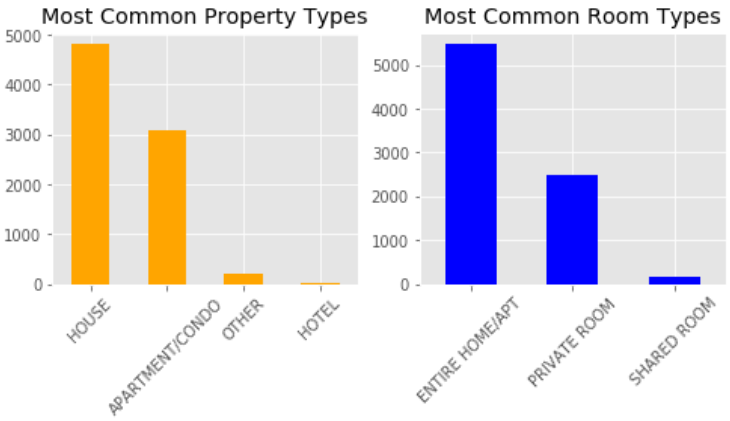
Answer: With some help from a similar project I found online, I was able to source some code and use the GeoPandas library to visually explore this question on an overlaid map of Austin, TX. I found that while the majority of the listings were located in downtown Austin, the median price point was more spread out. Downtown Austin was still one of the more expensive areas, however there were also some expensive areas to the west and northwest. I suspect that this is because Lake Travis is located in that direction, which is also a popular destination area for people traveling to the Austin area.





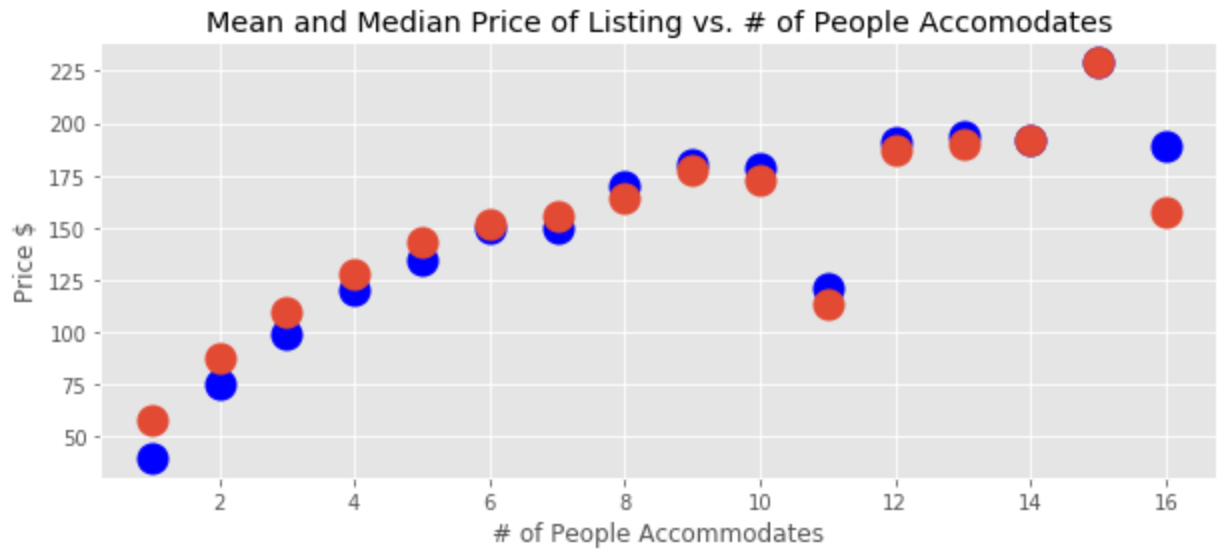
What are the most common listing types?

Answer: Houses are the most common property type for this data set, with the entire home/space being the most common room type.



How do Airbnb prices correlate with the number of people a listing accommodates?

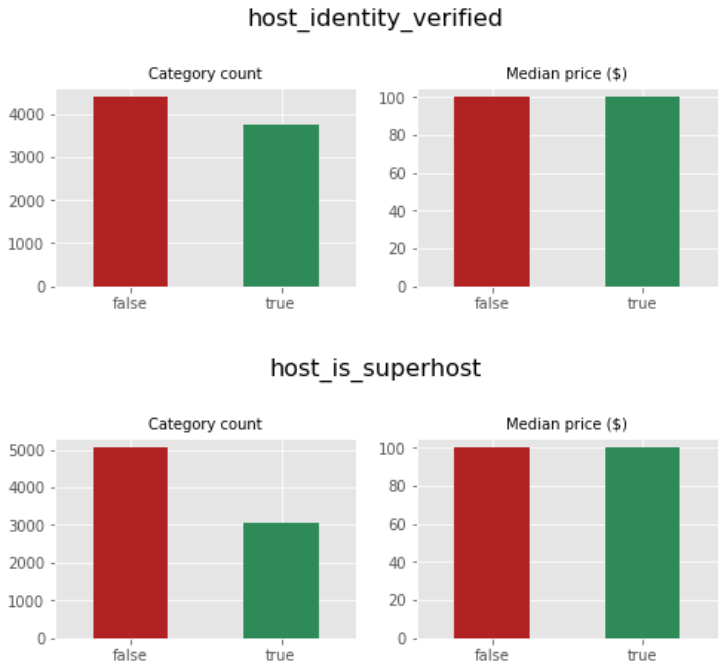
Answer: It appears that on average, price has a positive relationship with the # of people that an Airbnb listing accommodates, which is a logical pretense of this data set. Also, based on the data in this scatterplot, it seems that this relationship is logarithmic, meaning that the impact of increasing the listing price by accommodating more people subsequently diminishes. I found that the Pearson correlation coefficient for these two features was 0.45, which isn't strong, but still a positive correlation.



****Mean is in blue, Median is in red**

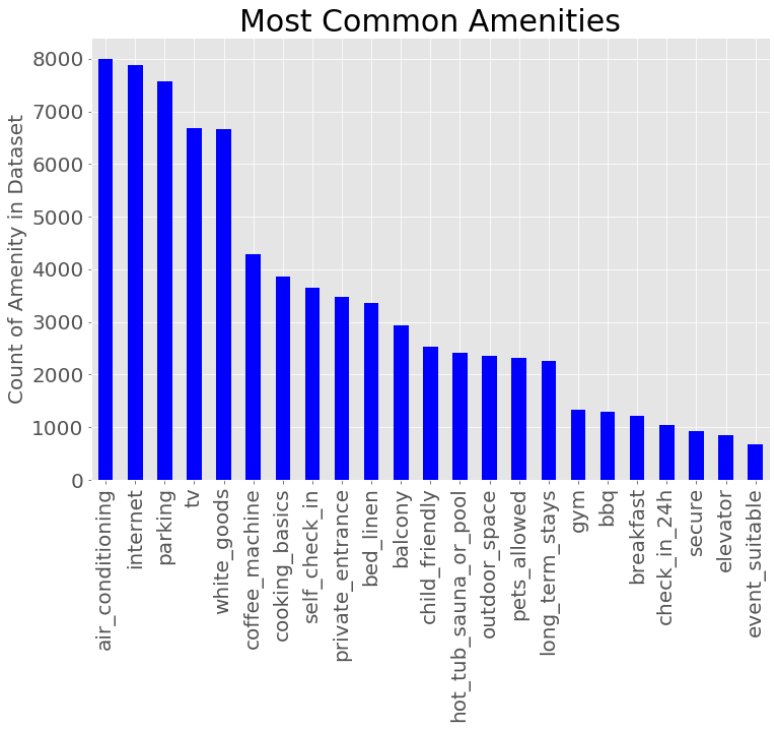
What is the value of being a superhost? Is it worth getting verified?

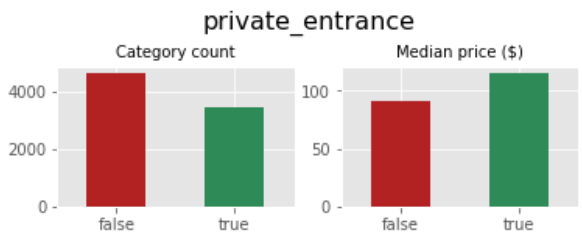
Answer: Neither being a superhost or verified seems to have any impact on median listing prices

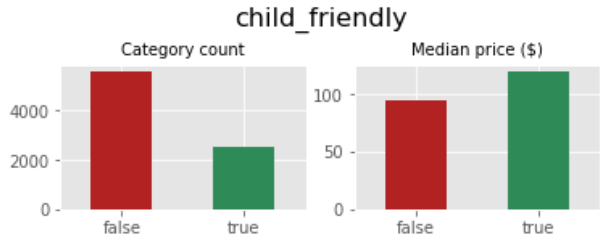


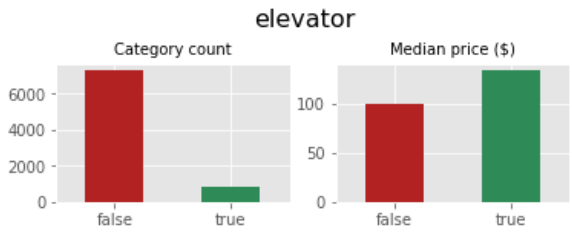
What are the most common amenities, and which amenities are likely to increase the listing price?

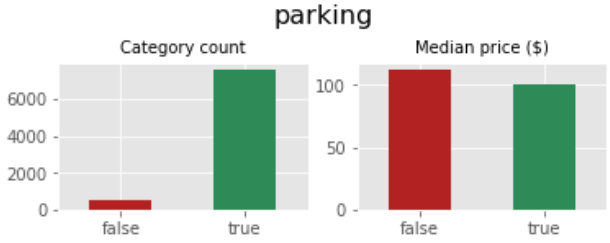
Answer: The Most common amenities include A/C, internet, parking, TV and white goods (linens etc.) However it appears that these aren't necessarily the most important amenities to increase listing price. Parking, for example is the 3rd most common amenity, however the data shows a negative impact on the median listing price when it is included in a listing. This could just be that since most of the more expensive listings are located downtown, they simply do not have parking spaces. Also, the majority of listings are not child friendly, however the listings that are child friendly have a significantly higher median price.







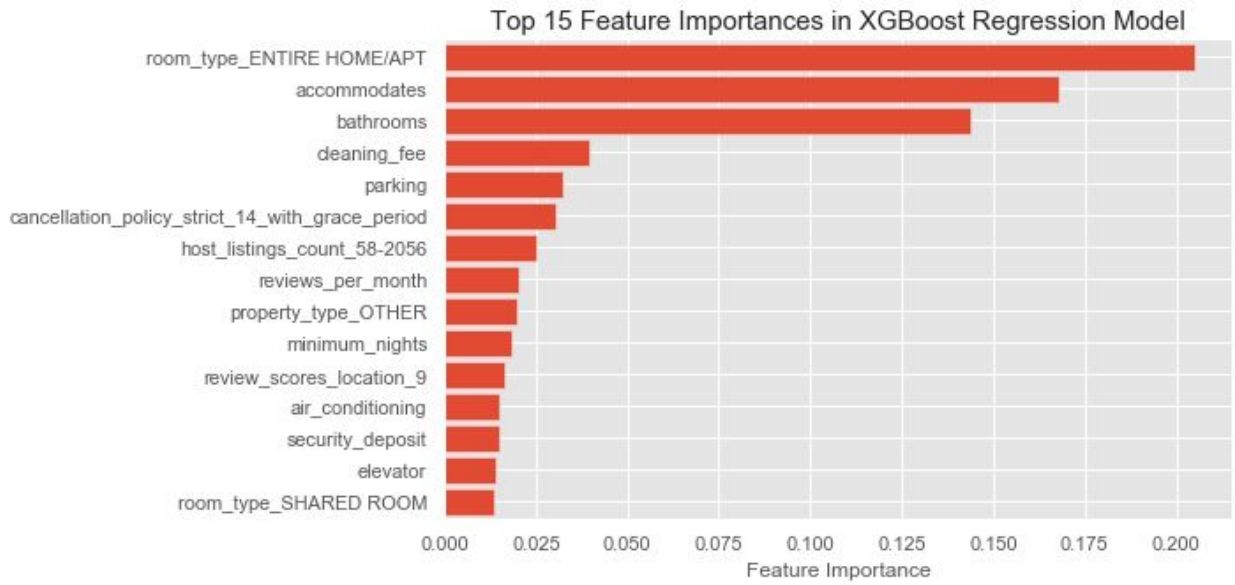




Statistical Analysis

Significant Features in the Data Set

After running a XGBoost regression model, I was able to determine which features in my dataset were most important in predicting the price of each listing.

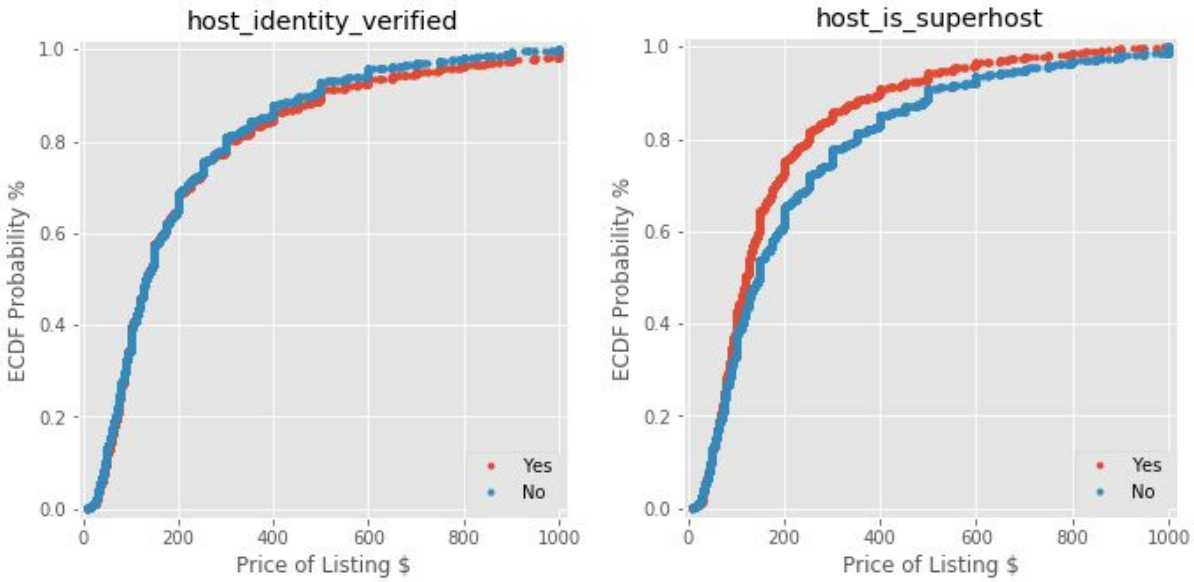


The top 3 features account for the majority of the influence on the model. The most important feature seems to be whether the listing is an entire house/apartment, which is not surprising considering that larger properties are more likely to accommodate more people compared to a shared room or something similar. Not surprisingly, the number of people that the listing accommodates is the 2nd most important feature followed by the number of bathrooms, which are features that have collinearity.

Subgroups of Data

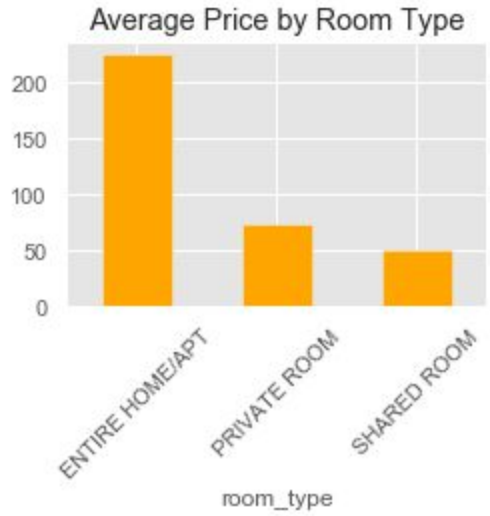
Superhosts

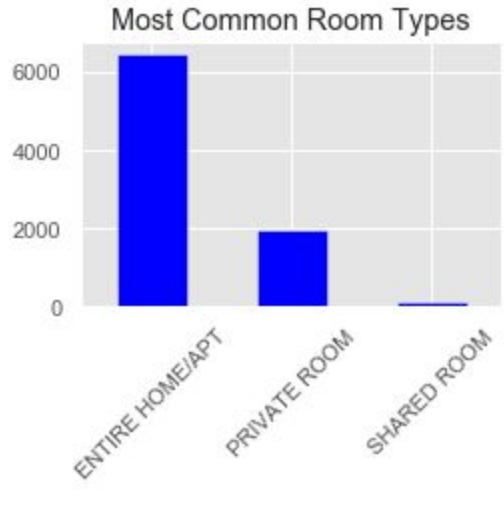
I wanted to explore whether or not being a superhost was valuable in terms of being able to charge a higher price for your listing. However, the ECDFs about the hosts indicate that prices are actually higher for non superhosts vs.superhosts.



Property/Room Type

It appears that there's a pretty big gap in average price between entire home/apartment listings vs. private rooms. Since this gap is so big, it might make the most sense to develop two independent models that are more representative of these two populations.

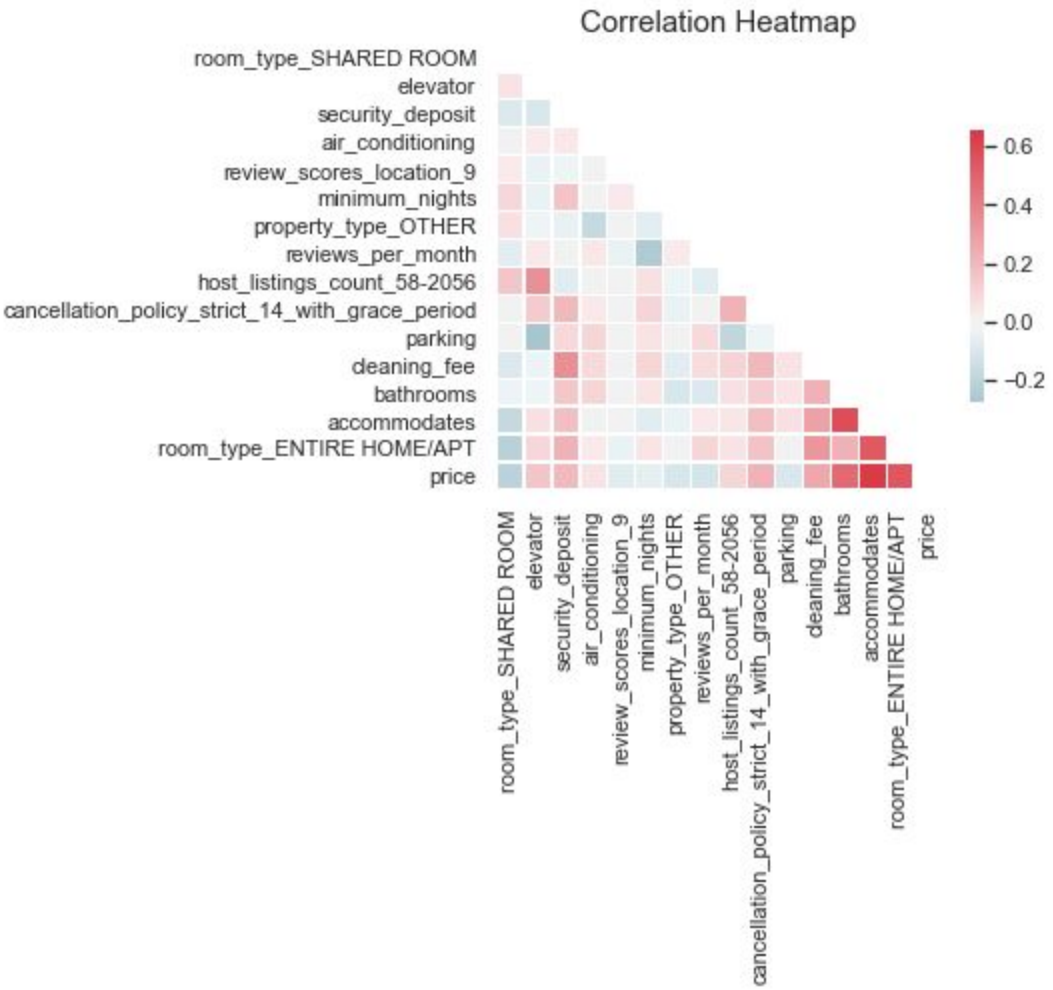

Average Price by Room Type
200
150
100
50
0
ENTIRE HOME/APT
PRIVATE ROOM
SHARED ROOM
room_type


Most Common Room Types
6000
4000
2000
0
ENTIRE HOME/APT
PRIVATE ROOM
SHARED ROOM


Price Distribution
Entire Home/Apt.
Private Room
1000
800
600
400
200
0
Frequency
3
4
5
6
7
Log Transformed Price

Feature Correlation and Feature/Target Correlations

I was able to generate a correlation heatmap for the top 15 most important features using the seaborn library.



Positively Correlated Features

This heat map reaffirms the feature importance figure (above) generated in the XGBoost model, which shows that price is most strongly positively correlated with the accommodations, bathrooms and entire home/apt. features.

Negatively Correlated Features

One interesting thing that this heat map suggests is that the location score rating is actually slightly negatively correlated with price. Since the majority of the listings in this data set have room types of entire homes/apt it does make sense. Parking and elevators are also negatively

correlated features, which makes sense since most of the listings that have elevators are probably located closer to downtown where parking is always more scarce.

Machine Learning

Revisiting the Project Goal

At the core, the fundamental goal of this project was to develop a model to accurately predict the nightly price of an Airbnb listing using features of the listing itself. In other words, this simply boils down to predicting the value of the dependent variable (price), by using those listing features, which serve as the dependent variables. Given this problem statement, I decided to use a linear regression model since its inherent properties would be able to accomplish this.

Model Preparation

One-Hot Encoding

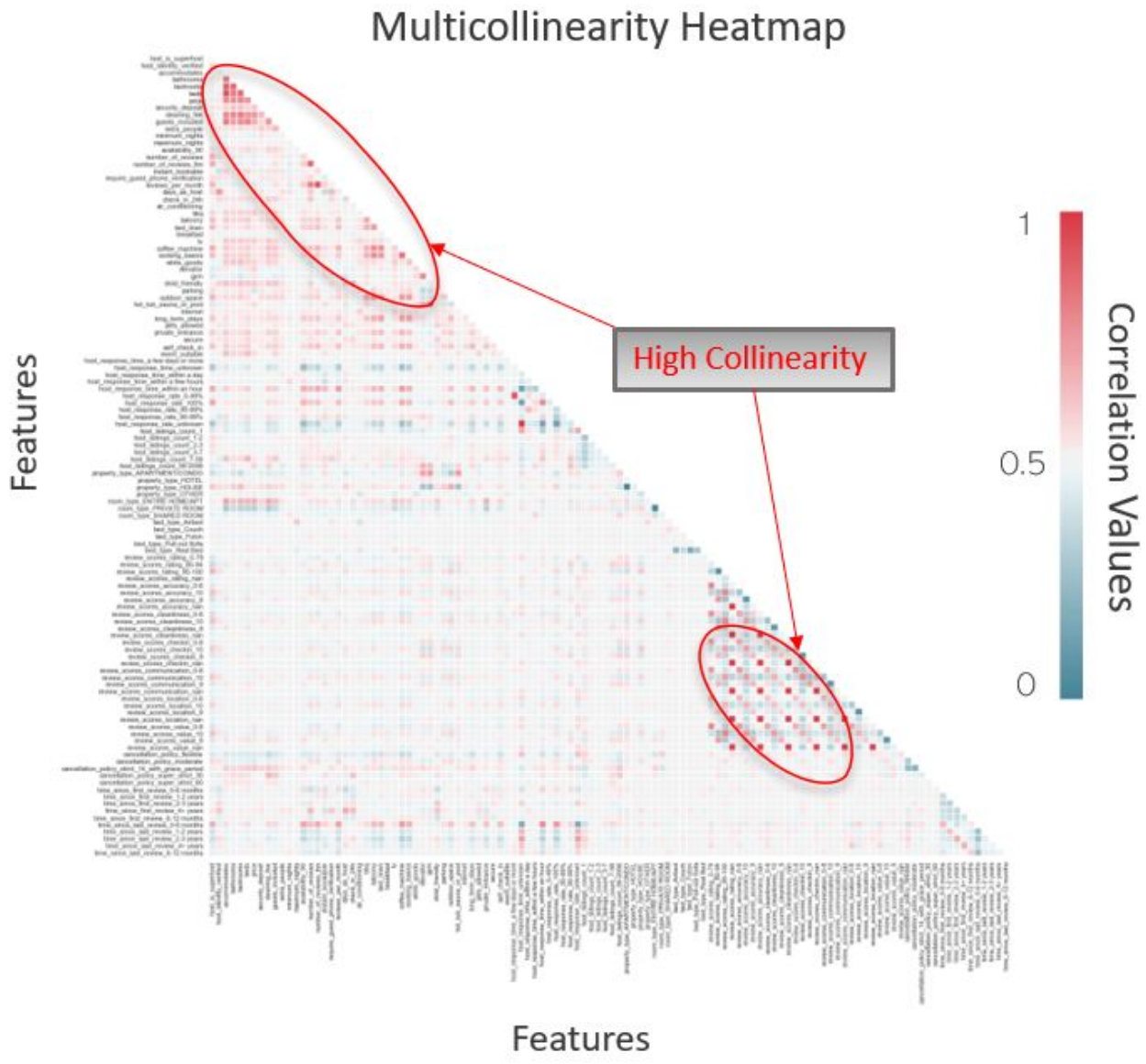
Before applying any regression models to the data, I had to massage the features of my data. The first step in this process was to use a common method called “One-Hot Encoding”, which allows categorical features to be expressed as integers. For example, instead of having a single feature such as “room_type” that contains string values such as “private_room” or “entire_home/apt”, One-Hot Encoding transforms each one of those unique values into separate columns containing either a “1” or a “0”. The result of implementing this technique can be seen below.

One-Hot Encoded Dataframe

id	host_is_superhost	host_identity_verified	accommodates	bathrooms	bedrooms	beds
2265	1.0	1.0	4	2.0	2.0	2.0
5245	1.0	1.0	2	1.0	1.0	2.0
5456	1.0	1.0	3	1.0	1.0	2.0
5769	1.0	1.0	2	1.0	1.0	1.0
6413	1.0	0.0	2	1.0	0.0	1.0

Multicollinearity

The next step I took was to assess the features for multicollinearity, which highlights features that have very strong linear relationships with each other. I was able to accomplish this by using a function from the Seaborn library. Using this function, I was able to generate a visual heat map that clearly showed areas of multicollinearity in the data.



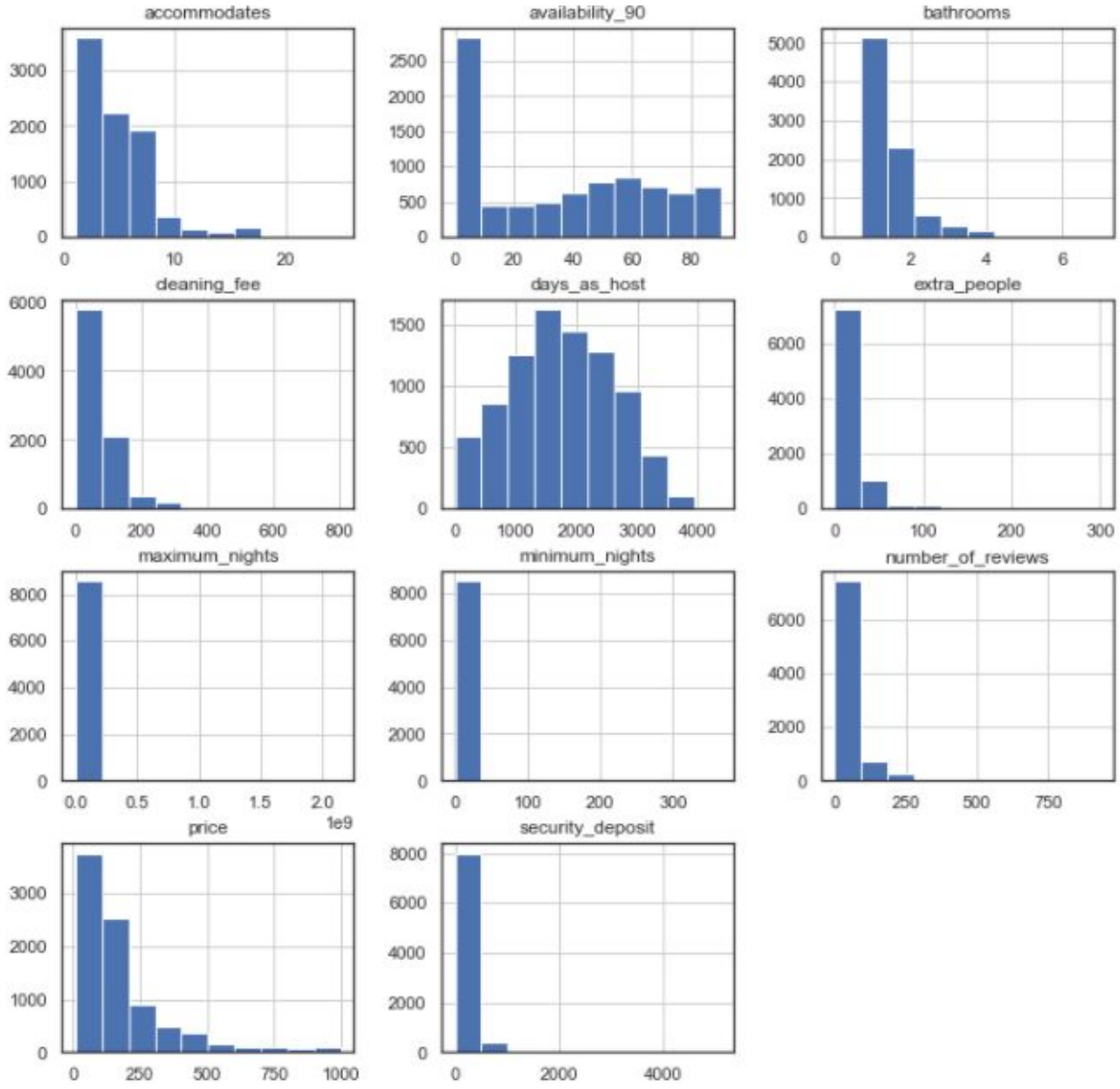
The dark red areas on the heat map consist of the beds, bedrooms and guests features. Since these features are all highly correlated and relatively similar in nature with the

number of people that a listing accommodates, I decided to drop them from the data set.

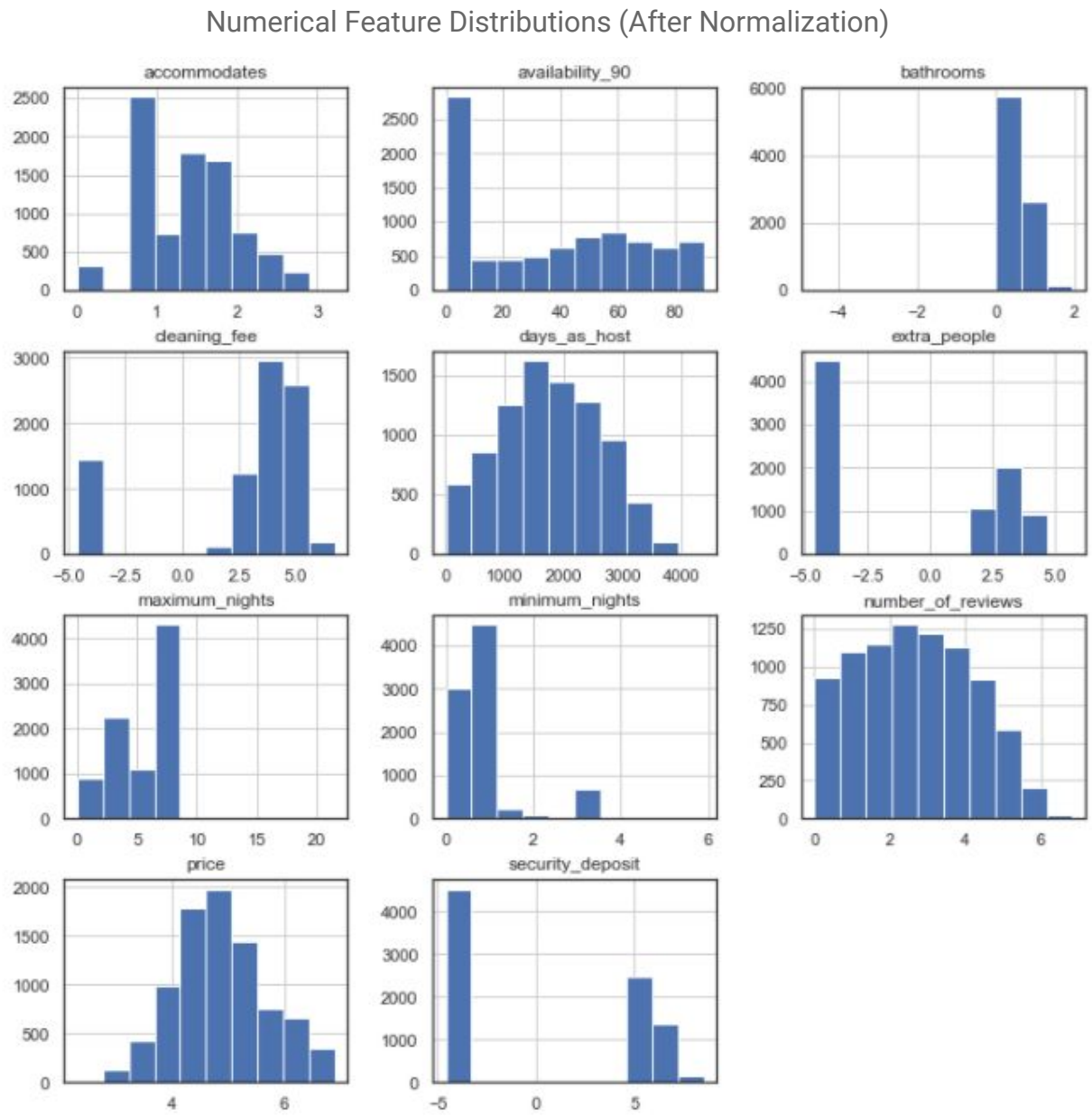
Normalization and Standardization

As a last step in the model preparation process, I decided to take a look at some of the numerical features to see if there were any features that needed to be normalized.

Numerical Feature Distributions (Before Normalization)



Since a lot of these features seem to be positively skewed, I decided to do a log transformation on the following features: *'price','accommodates', 'bathrooms', 'security_deposit', 'cleaning_fee', 'extra_people', 'minimum_nights', 'maximum_nights', 'number_of_reviews'*. Most of the resulting distributions appear much more normal and should provide better results in the modeling phase.



Lastly, I scaled these numerical features so they would have equal weights in the machine learning model. I accomplished this by using the StandardScaler in SKLearn.

Machine Learning

Model Implementation

After completing steps for model preparation, I began setting up regression models. My initial model was constructed using a gradient boosted regression classifier from the SKLearn ensemble. I also decided to test a gradient boosted regression model using the XGBoost framework. The results of these two models are shown below.

Model Performance

Metric	Model	
	XGBoost Regression	SKLearn Gradient Boosted Regression
Train MSE	19.61%	19.28%
Test MSE	21.11%	21.64%
Train R^2	0.7228	0.7274
Test R^2	0.6943	0.6867
Computational Time (sec)	0.9	11.8

Feature Importance

Feature Importance	Model	
	XGBoost Regression	SKLearn Gradient Boosted Regression
1	room_type_entire home/apt	accomodates
2	accomodates	property_type_other
3	bathrooms	security_deposit
4	cleaning_fee	bathrooms
5	air_conditioning	require_guest_verification
6	parking	extra_people
7	cancellation_policy_strict_14_ with_grace_period	reviews_per_month
8	host_listings_count_58-2056	child_friendly
9	security_deposit	cleaning_fee
10	property_type_other	white_goods

Interestingly, the feature importance is actually quite different between these two models, but the most important feature across appears to be the number of people a listing and the number of bathrooms a listing accomodates. There are however, some other important ancillary

features that could help boost the price of the listing, such as having parking, air conditioning, the entire home to yourself, being child friendly or having linens supplied in addition to some other features.

Conclusion

Considering the data presented above, it appears that both models had nearly the same results. The SKLearn model had slightly better training results, while the XGBoost model had better results on the test set. This indicates that the SKLearn model might have been slightly overfitting the data. However, the biggest standout metric was computational time. The XGBoost model was more than 13x faster in computational speed, which could have huge implications as the data set grows.