Predicting Airbnb Listing Prices in Boston

Introduction

We use both listing and review datasets to predict Airbnb listing prices in Boston. Our knowledge of the real estate market and the existing literature informs[1] us that the price of a listing usually depends on three main factors: location, size and seasonality. Other factors, for example: cleanliness of property, average ratings of review and etc. might affect price as well, however, they are not as significant as the three main causes.

Thus we use variables given in the datasets, which might be closely correlated with location, listing size and seasonality, generate statistics of them, determine their statistical significance by running t-test, and regress on them to predict the price. At the same time, we drop variables, which either convey information that have been already described by other variables or they are not closely correlated with price.

In addition, we transform some non-numeric variables to numeric variables, for example, we transform text in datasets into continuous and dummy variables so that information conveyed by text can be used in regression. We also generate dummy variables based on seasonality of a year. Moreover, we use some third party data to better describe the location of the listings. There are some variables that affect the price as well, though in a less significant way. We include these variables as well in analysis. A more detailed description and descriptive statistics of the variables will be provided in the next section.

Descriptive Statistics

In this section, we will describe variables we use for analysis. We first divide the variables into two categories: variables we adopt directly from datasets and variables we create. In each category, we group variables by their similarity, and give each group a name. Afterwards we describe each variable, its role in analysis and list the hypotheses we would like to test. Finally, we would like to show descriptive statistics for these variables.

Variables Adopted Directly From Datasets

- **Host Information Variables**

host_is_superhost, host_has_profile_pic, host_identity_verified

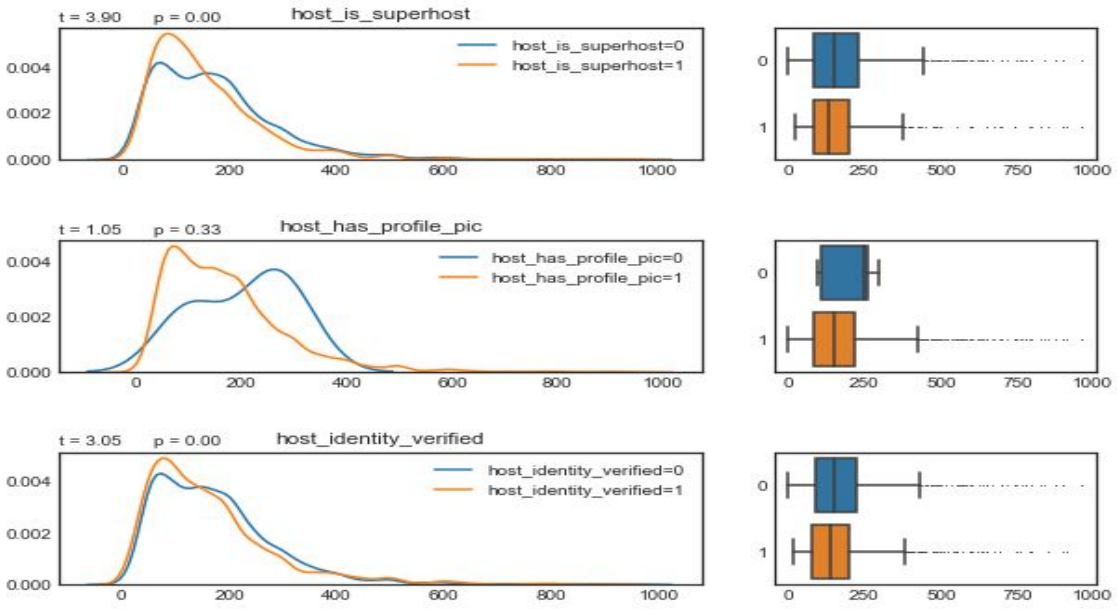
[1] https://aresjournals.org/doi/abs/10.5555/rees.6.3.l532111124144176?journalCode=rees

These variables show how trustworthiness and friendliness of the host. Host information related dummy variable indicating 1 means that host is more trustworthy.

The hypothesis we would like to test using these variables is:
If a host are trustworthy, then the price of his/her listing is higher.

In three subplots below, we show how price is distributed for different values of each dummy variable mentioned above.

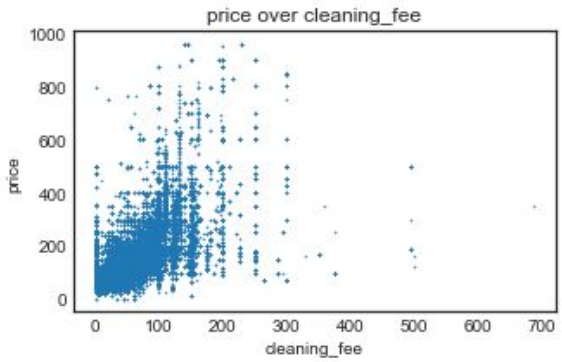
For host_is_superhost plot, it can be shown that position of being a superhost affects the price significantly; whereas, the effect of having_profile_pic is not statistically significant at 5% significance level. Finally, having an identity verified being positively correlated with price is shown in the third subplot.

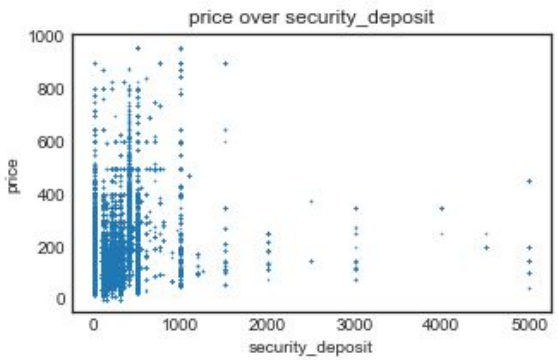


- **Policy Variables**

cleaning_fee, security_deposit

These variables will incur extra cost in price and thus bring up the price of listings. In addition, a larger amount of cleaning fee generally indicates a larger/more luxurious place being rented to tenant, and larger/more luxurious listings generally have higher price. The same rule applies to security_deposit as well. In fact, it is shown that both cleaning_fee and security_deposit have positive correlation with price of listing in the plots below, and our conjectures are confirmed.



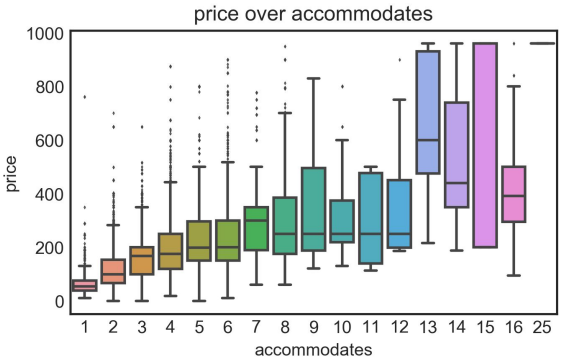


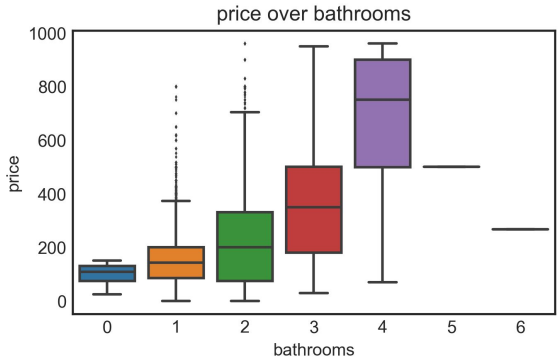
- **Size Related Variables**

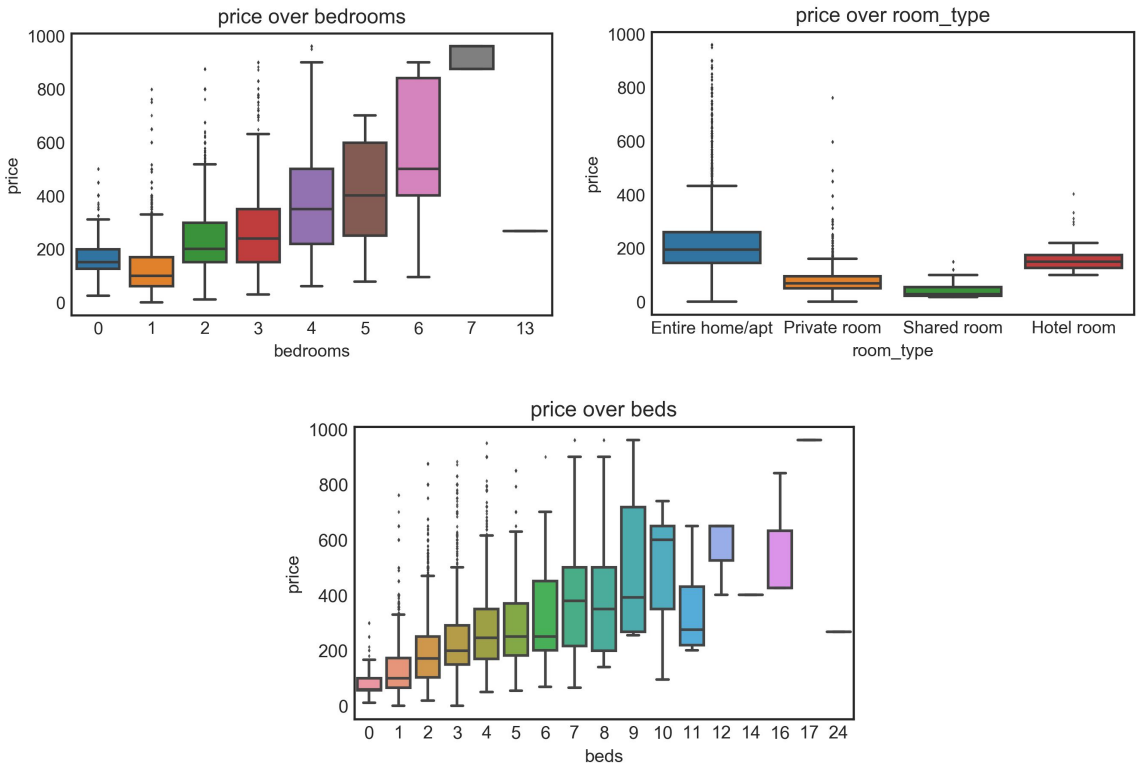
accommodates, bathrooms, bedrooms, beds, room_type

These variables indicate the area size of lodgings. In general, the more people a lodging can accommodate, more bathrooms or bedrooms a lodging has, or more beds a lodging possesses, the higher the price of listing. Airbnb rooms in our dataset are divided into 4 room types: entire home, private room, shared room and hotel room. Entire home type is usually larger than private room and shared room and thus incurs a higher price.

The box plots shown below for these variables support the assumption that the larger the size, the higher the price. Moreover, the positive bivariate relationships between several variables and the price show that these variables serve good indicators for size.



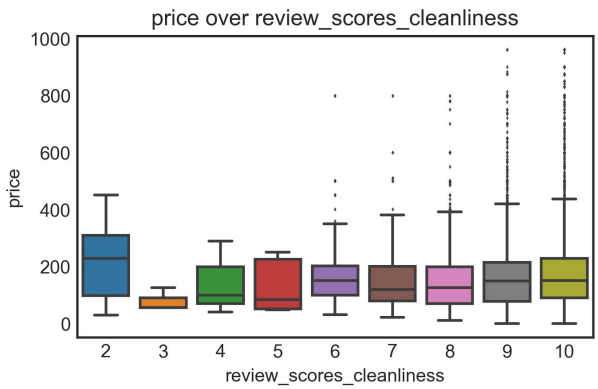


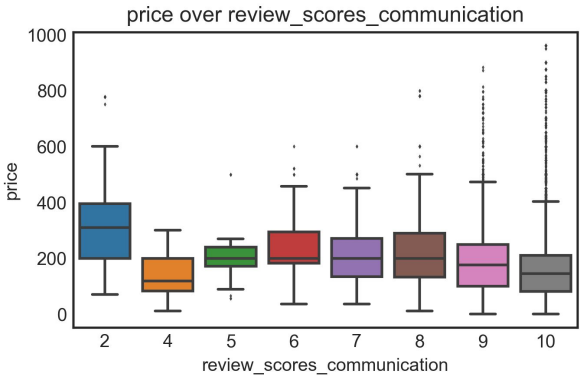


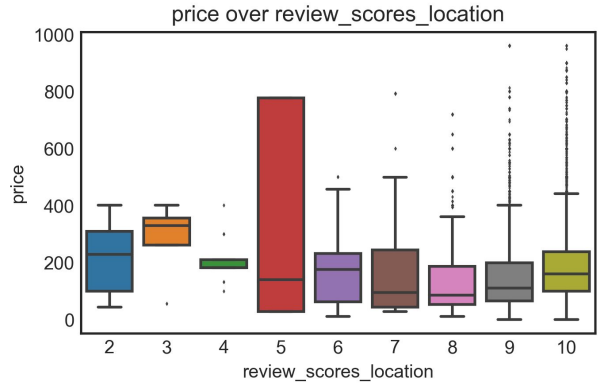
- **Review Variables**

review_scores_accuracy, review_scores_cleanliness, review_scores_checkin, review_scores_communication, review_scores_location, review_scores_value

These variables are scores that tenants give after their stay in the lodgings. They are numerical values (range 0-10) that show how tenants perceive a listing. The hypotheses we have are: the higher the review score, the more popular the listing is, and the higher the price it charges. It is easy to find that many trending lodgings on Airbnb receive high review scores across all review categories. As a result, potential customers would pay a higher price to have the experience in these lodgings. However, plots we generated below to describe the relationship between price and each review score do not confirm our theory. Price range is similar for different scores of one review category. One explanation for that is reviews do not cause prices to change significantly. People tend to leave high score reviews for good experience, however, good experience is not necessarily correlated with price of listing. In fact, people sometimes feel they are overcharged, if they have plain experience staying in a luxurious place and will leave a low score for high-price staying.







- **Statistics Summary Table For Selected Numeric Features**

Variable	Mean	Std	Min	25%	50%	75%	Max
price	192.57	214.04	0.00	85.00	150.00	225.00	5000.00
host_is_superhost	0.26	0.44	0.00	0.00	0.00	1.00	1.00
host_has_profile_pic	1.00	0.04	0.00	1.00	1.00	1.00	1.00
host_identity_verified	0.36	0.48	0.00	0.00	0.00	1.00	1.00
is_location_exact	0.80	0.40	0.00	1.00	1.00	1.00	1.00

accommodates	3.62	2.39	1.00	2.00	3.00	4.00	24.12
bathrooms	1.27	0.52	0.00	1.00	1.00	1.50	6.00
bedrooms	1.37	0.99	0.00	1.00	1.00	2.00	13.00
beds	1.92	1.48	0.00	1.00	1.00	2.00	24.00
review_scores_accuracy	9.52	0.94	2.00	9.00	10.00	10.00	10.00
review_scores_cleanliness	9.43	0.90	2.00	9.00	10.00	10.00	10.00
review_scores_checkin	9.68	0.83	2.00	9.83	10.00	10.00	10.00
review_scores_communication	9.64	0.87	2.00	9.56	10.00	10.00	10.00
review_scores_location	9.55	0.77	2.00	9.00	10.00	10.00	10.00
review_scores_value	9.22	0.96	2.00	9.00	9.22	10.00	10.00
cleaning_fee	69.94	48.98	0.00	30.56	61.67	100.00	495.00
security_deposit	198.42	317.37	0.00	0.00	100.00	300.00	5000.00

- **Statistics summary table for selected categorical features**

Room Type	**N**	**Min**	**Max**	**Mean**	**SD**
Entire home	3343	250.09	5000.00	0.00	242.30
Hotel room	25	178.64	403.00	100.00	81.45
Private room	1670	82.58	762.00	0.00	51.17
Shared room	55	42.45	150.00	17.33	29.68

Variables Created

- **Variable Created Based On Text:**

avg_len_review, review_dummy, desc_length, listing_desc, location, n_reviews

In this section, we used nltk package to mine text data columns in both listings and reviews

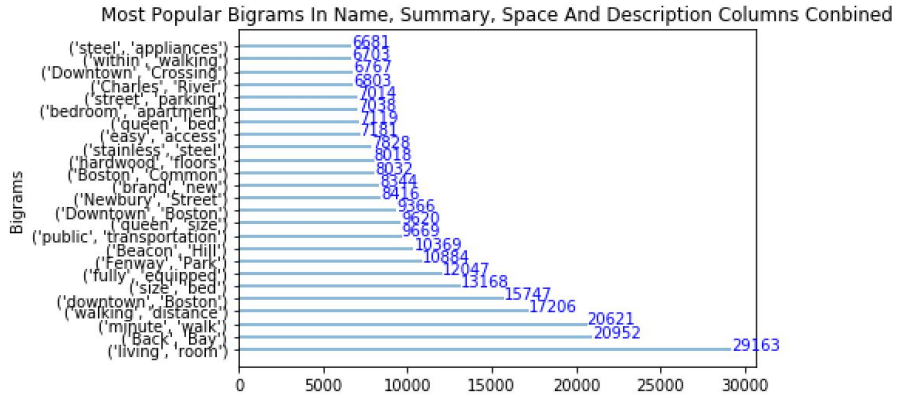
datasets, these columns are name, summary, space, description, neighborhood and transit in listings dataset, comments column in reviews dataset. We group the columns into three categories:

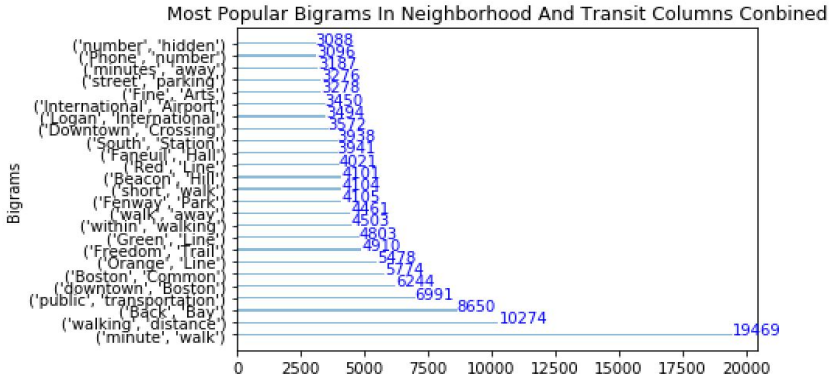
- group 1, description related text information, and it includes name, summary, space and description columns;
- group 2, neighborhood related text information, and it includes neighborhood and transit columns;
- group 3, review related text information, and it is composed of only the comments column, and we would like to transform the text-based variables into quantitative variables and to test hypotheses on the correlation among these variables with price.

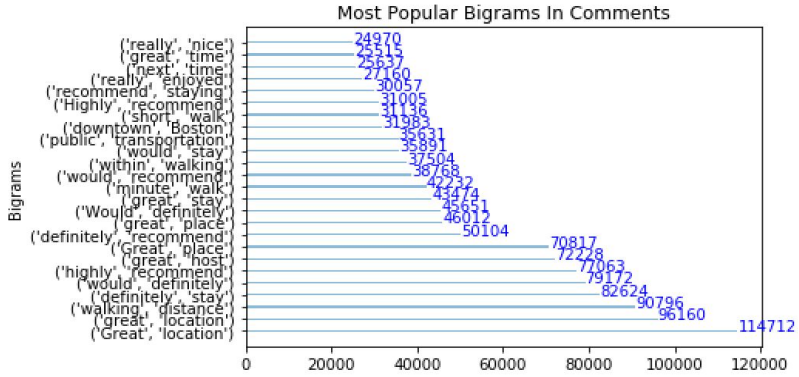
The transformation methods include:

- Calculate the length of description of each listing by counting tokens of the description; we assume that the longer the description, more attention to the owner of the property pays to the listing and the higher the credibility of the listing thus the higher the price of listing.
- Calculate the average length of reviews per listing by counting the average number of tokens for reviews per listing; the average length of reviews per listing also indicates the popularity of that listing, and the longer the average length of reviews a listing has, the higher the price of listing
- Count the average number of reviews per listing; the average number of reviews per listing indicates the popularity of that listing as well, and more reviews a listing has, the higher the price of listing
- Generate the 50 most popular words used in each group, and observe if there are any hot words that listings would like to use for description and neighborhood, and reviews would like to use for comments; Create dummy variables for description, neighborhood and review. If each group of text of each listing hits one of the most popular bigrams. Label 1 for corresponding dummy variable if it does, otherwise 0.

We generate a vocabulary set for all words used in these four columns. After that we remove all punctuation and filler words in these texts and obtain the most common words by applying nltk.FreqDist() method. The result is actually not informative. Unigrams out of contexts do not make much sense to indicate popularity of listings. Consequently we decide to use collocations instead. The most popular bigrams in description, location or review columns suggest that location is one important factor to determine the popularity of a listing. For example, downtown Boston, Boston Common and several location related phrases appear in all three plots.







- **Variable Created To Label Neighborhood Based on Income**

Incomelevel = 1*Low_income+2*Low_middle+3*Upper_middle+4*Upper_class

This is a score for each neighborhood based on its median household income. We used household income by zipcode data provided by the US Census [2]bureau. We categorize households as follows:

- Low_income: 0-$50,000

[2] We used ACS 5-year estimates

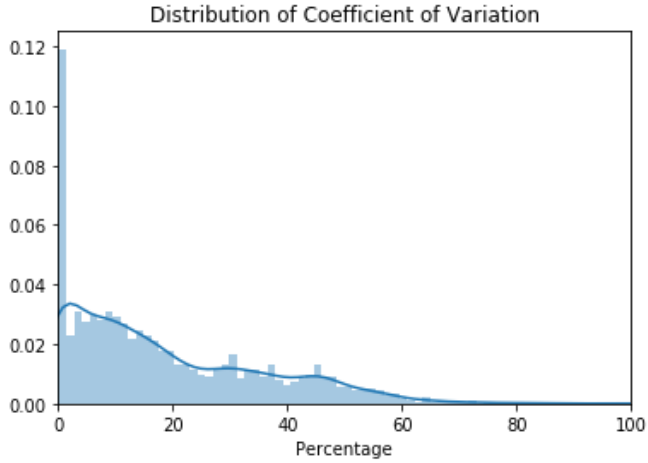
- Low_middle: $49,000- $99,999
- Upper_middle: $100,000- $149,999
- Upper_class: Above $150,000

As of 2017, the median household income for Boston is $66,758[3]. We obtain the percentage of households in each income level within each neighborhood, and calculate the weighted average income within each neighborhood while assigning 1 to low_income, 2 to low_middle, 3 to upper_middle and 4 to upper_class. We calculate the relative income level of each neighborhood out of a scale of 4. We would like to test whether the price of listing is higher if it is posted in richer neighborhood, or whether it is lower in a poorer neighborhood.

- **Variables Based On Time Effects**

is_Fri_Sat, is_holiday, season_autumn, season_winter, season_spring

Time series tools cannot be used in our analysis since the date set is not a perfect panel data. However, an exploratory analysis on time effects in calendar dataset shows time effects may have significant effects on price fluctuations. By computing the coefficient of variation of calendar dataset and plotting its distribution, we find lodgings with price fluctuating rate within 15% only account for about 50% of all listings. About 25% lodgings have price fluctuating rate greater than 30% over time.

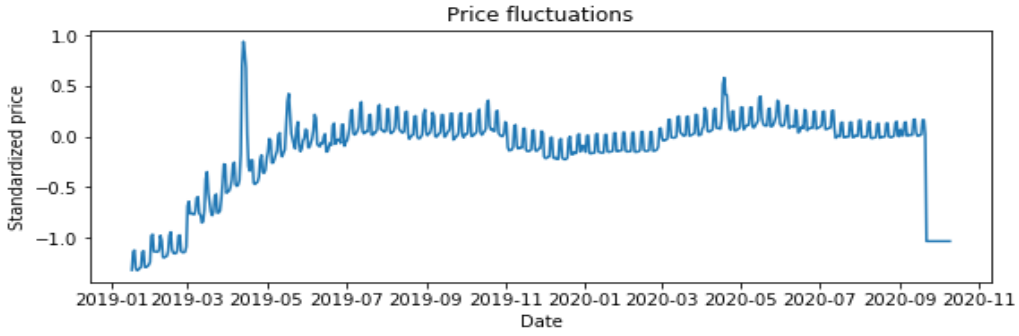


For further exploration, we visualize the price trend for the entire time period as well as a short time period to show price cycles in both long term and short term, and find weekends and seasons have effects on price fluctuation. As shown on the plot below, it indicates that price of listings rise sharply in April and it plummets in November.

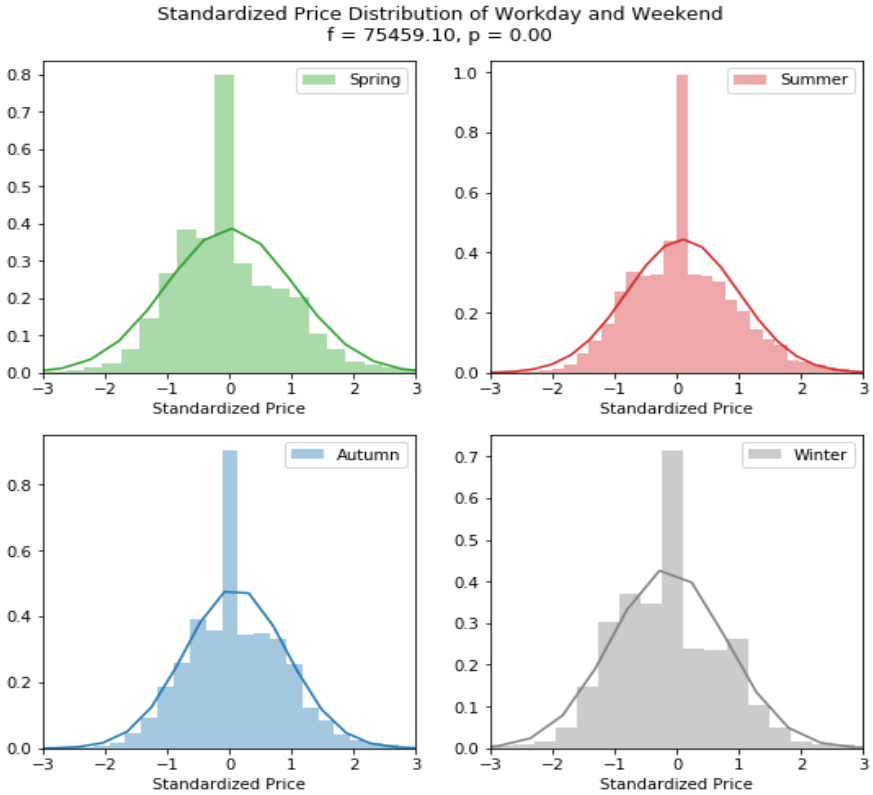
These results make sense since Boston is tourist friendly during spring and summer seasons, and thus the price of listings go up from the beginning of spring in April. During the winter season from November, the city is cold, windy, and full of snow. It is not an attractive tourist

[3] https://datausa.io/profile/geo/boston-ma/

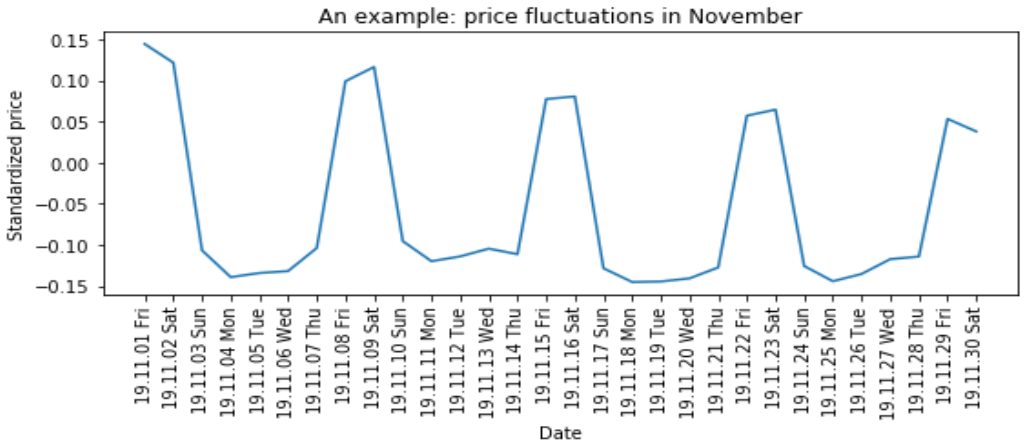
destination compared to other states such as Florida and California. As a result, the price of many many listings experience sharp decrease from November.



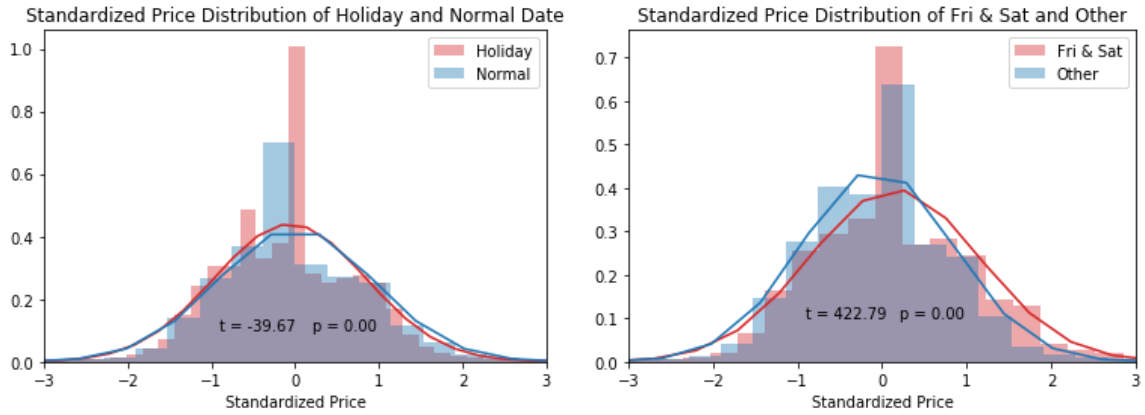
We plot standardized price for different seasons in the four subplots below, and use a smoothing line to obtain the distribution of the standardized price in different seasons. The curves in spring and summer flatten out to more extreme prices while the curves in winter tends to concentrate in the middle, which indicates that prices are less extreme, hence lower in winter.



The price over a short term period illustrates how it peaks during weekends and it goes down after Sunday. The pattern is repetitive over 4 weeks period in November presented below, and clearly shows how weekday affects price.



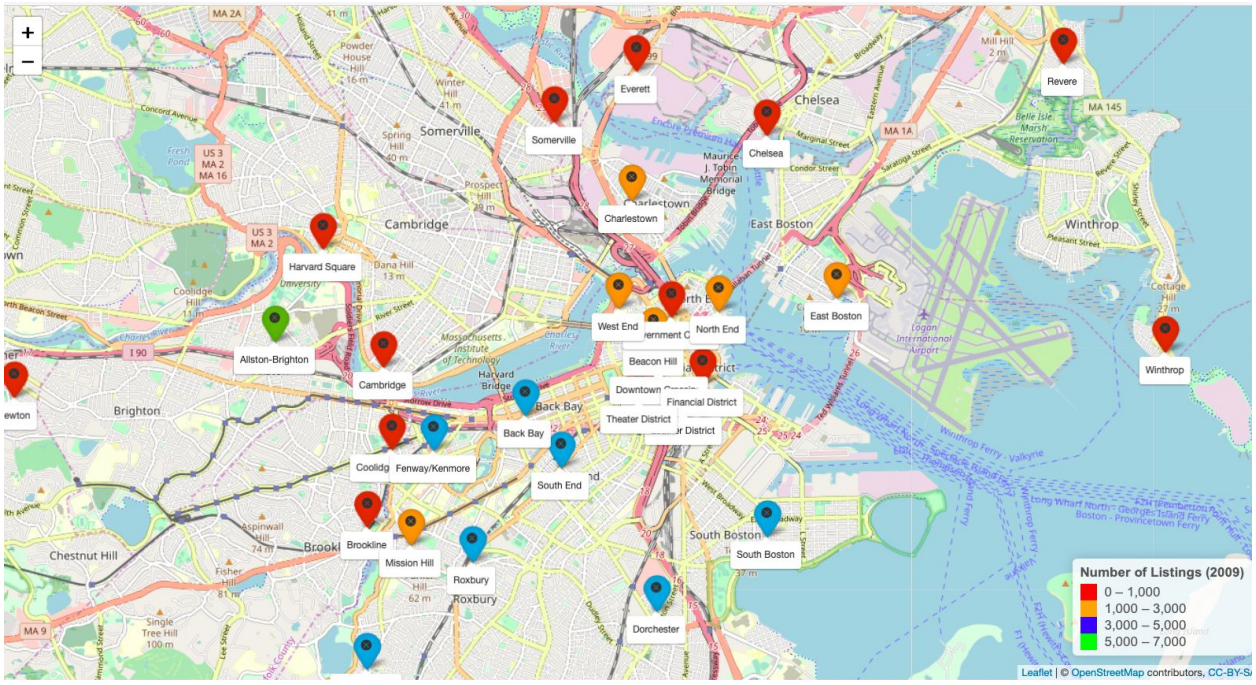
In addition, we plot two graphs below to illustrate how price differs between holidays and workdays and how it differs between weekends and workdays. Both results are statistically significant. However, in the first subplot, the price is lower during holidays than during normal days, whereas the second plot shows that the price is higher during weekends than during workdays. The second result makes sense, however, the first one is a little counter-intuitive. It might be the reason that holidays concentrate during the second half of the year, and in later fall and winter seasons, the price of listing is not as high as that in the first half of the year.



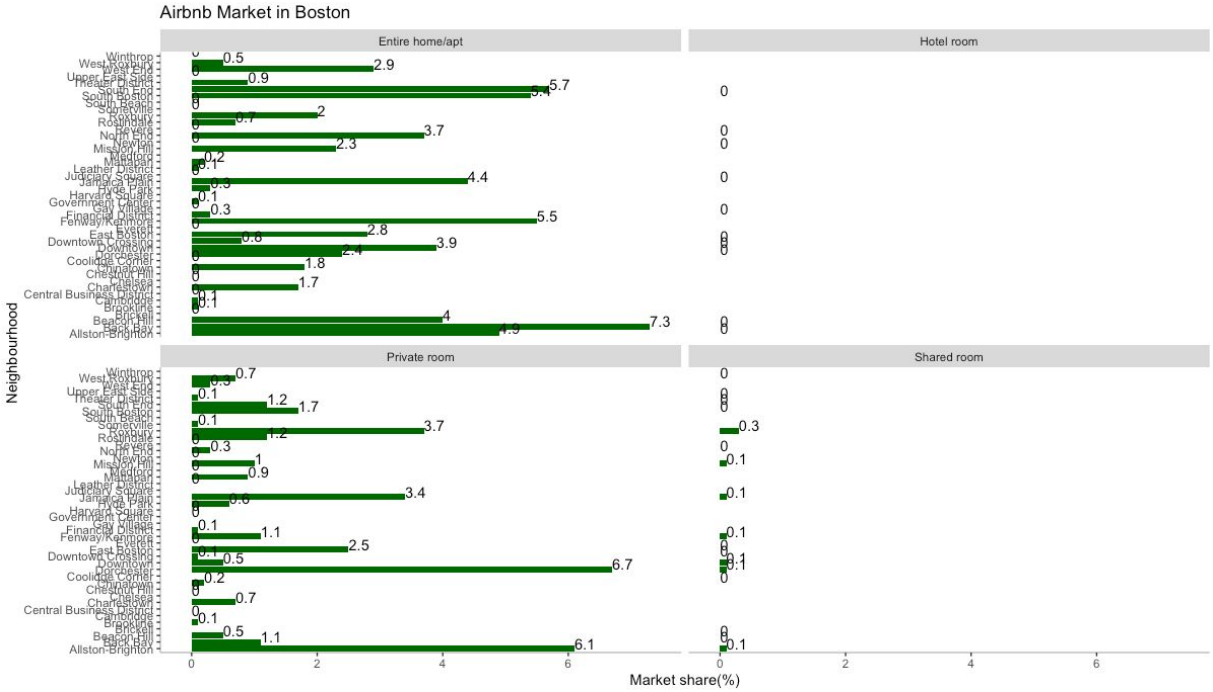
So far we have found holiday, weekend and season play roles to decide the price, and thus we use six dummy variables to capture the time feature of the listing.
Is_holiday: is_holiday = 1 if the date of listing is a Federal holiday, 0 otherwise;
Is_Fri_Sat: is_Fri_Sat = 1 if the day of listing is a Friday or a Saturday, 0 otherwise;
season_spring = 1 if the day in Mar, Apr or May, 0 otherwise;
season_autumn = 1 if the day in Sep, Oct or Nov, 0 otherwise;
season_winter = 1 if the day in Dec, Jan or Feb, 0 otherwise;

Neighborhood Analysis

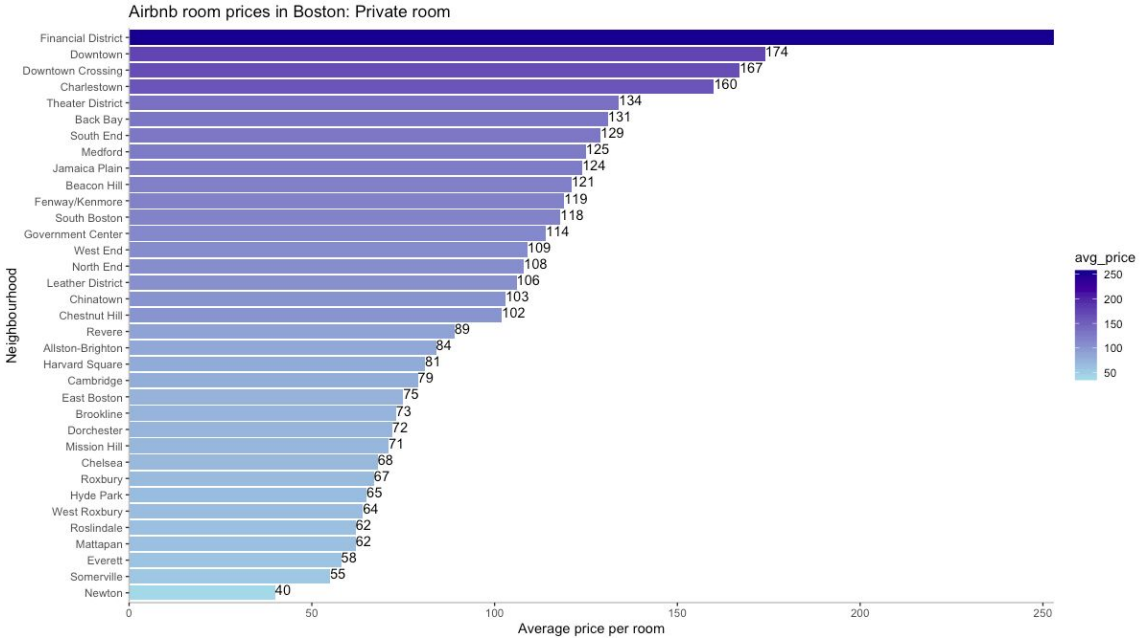
The most popular neighborhoods are located at the center of Boston and in other densely populated suburbs outside of the city's downtown area. In our analysis we find that Allston-Brighton is the most popular neighborhood with about 11% of all listings. Dochetser and Backbay are second and third with a market share of 9% and 8% respectively. Neighborhoods located away from the city center such as Witnthrop and Chelsea have the fewest listings.The leaflet map below shows the number of listings in each neighborhood.



We also find that the popularity of neighbood varies greatly by the room type. The graph below shows Dochester (6.7%) is the most popular neighborhood for private rooms while BackBay (6.7%) is the most popular neighborhood for Entire home/apt. Overall, the Airbnb market in Boston is concentrated in private rooms and entire apartments with very few listings in hotels and shared room across all neighborhoods.



The most expensive private rooms in Boston are located within a mile radius of Boston city center. The closer a listing is to the city's center, the higher the price. A room in the Financial district can cost as much as $253 per night. The bar plot below shows these prices in detail. The result supports our assumption the areas close to the city center have access to better amenities such as transportation, restaurants, recreational services. These neighborhoods are also where many large financial institutions offices are based, which tends to result in a tight real estate market. In the barplot note that neighborhoods located far away from the city center, such as Newton(10.9 mi), Everett (4.3mi), Somerville (3.9mi) have room prices below $60.



Regression and Analysis

Sampling Method

The file calendar.csv contains more than 20 million rows of price record. Limited by computing capability, we cannot use the whole data set to train our model. Thus, we design a sampling method to balance between model performance and computing time.

Based on our previous analysis, location variables contain many useful information that are helpful for predicting price. However, this feature is dropped in regression analysis, though some variables correlated with it are included. To avoid losing information, we use stratified sampling based on the neighborhood variable to ensure all neighborhoods are included in the sample.

We take 1000 lodgings from listings_details.csv as a sample, and then match these records with daily price records in calendar.csv. Finally our sample contains about 200 thousand rows of price records.

OLS Results Table[4]

We transform the price using natural log as price is initially right skewed. Log price results in a normally distributed shape and we proceed to run a regression[5] of log(price) on all the explanatory variables outlined in the summary statistics statistics table. The table below shows results for a few explanatory variable[6]s from the location, space area, and seasonality variables

[4] See the appendix for an html file with all detailed OLS regression results
[5] We split the data into a train and test data with ration of 75%: 25%.The results are for test data
[6] For detailed regression results, please see the appendix

Regression Model Results

	Dependent variable:
	log_price
cleaning_fee	0.0047
	(0.00004)
	t = 107.8295
	p = 0.0000***
host_is_superhost	0.0048
	(0.0033)
	t = 1.4821
	p = 0.1384
host_has_profile_pic	0.8493
	(0.1140)
	t = 7.4524
	p = 0.0000***
host_identity_verified	-0.0658
	(0.0027)
	t = -24.5477
	p = 0.0000***
accommodates	0.0640
	(0.0006)
	t = 112.9079
	p = 0.0000***
security_deposit	-0.00004
	(0.000005)
	t = -9.4493
	p = 0.0000***
review_scores_accuracy	0.0392
	(0.0016)
	t = 24.3186
	p = 0.0000***
review_scores_value	-0.0436
	(0.0017)
	t = -26.0260
	p = 0.0000***
n_reviews	-0.00002
	(0.000002)
	t = -7.9362
	p = 0.0000***
avg_len_review	-0.00002
	(0.00004)
	t = -0.5631
	p = 0.5734
review_dummy	-0.0786
	(0.0031)
	t = -24.9563
	p = 0.0000***
incomelevel	0.2244
	(0.0035)
	t = 64.9637
	p = 0.0000***
desc_length	-0.0004
	(0.00002)
	t = -18.5173
	p = 0.0000***
listing_desc	0.0687
	(0.0049)
	t = 14.0394
	p = 0.0000***
location	0.0279
	(0.0035)
	t = 8.0072
	p = 0.0000***
room_type_entire_home_apt	0.3348
	(0.0213)
	t = 15.7429
	p = 0.0000***
room_type_private_room	-0.2993
	(0.0216)
	t = -13.8611
	p = 0.0000***
room_type_shared_room	-0.7882
	(0.0297)
	t = -26.5079
	p = 0.0000***
is_Fri_Sat	0.0682
	(0.0024)
	t = 27.8685
	p = 0.0000***
is_holiday	-0.0462
	(0.0051)
	t = -9.1469
	p = 0.0000***
season_autumn	-0.0066
	(0.0034)
	t = -1.9052
	p = 0.0568*
season_winter	-0.3463
	(0.0032)
	t = -106.6548
	p = 0.0000***
season_spring	-0.0932
	(0.0032)
	t = -29.2157
	p = 0.0000***
Constant	3.3796
	(0.1165)
	t = 29.0074
	p = 0.0000***
Observations	158,946
R^2	0.5717
Adjusted R^2	0.5717
Residual Std. Error	0.4681 (df = 158922)
F Statistic	9,224.4860*** (df = 23; 158922)
Note:	*p<0.1; **p<0.05; ***p<0.01

OLS Findings

The results show that a $10 increase in the cleaning fee results to a 4.7% increase in the price of a listing. We also find that listings whose hosts have a profile picture are 84.9% higher than listings for hosts who do not have a profile picture ceteris paribus.

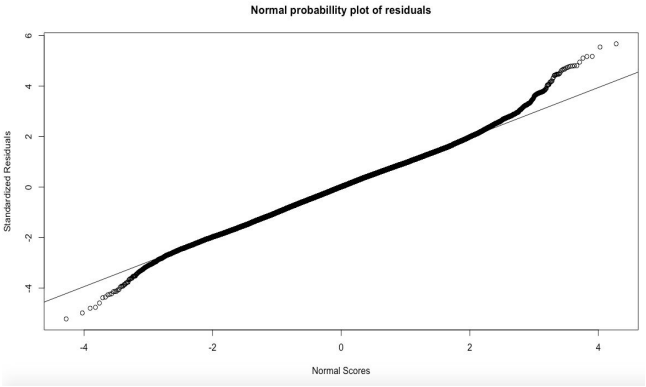
We also find that prices for private rooms are 29.9% lower than those of hotels rooms. The income level variable is also important as a one unit increase in the neighborhood income level results to a 22.4% percent increase in a listing's price. A one unit jump here represents movement from one income category to another, for example lower class to lower middle class or from upper-middle class to upper-class.

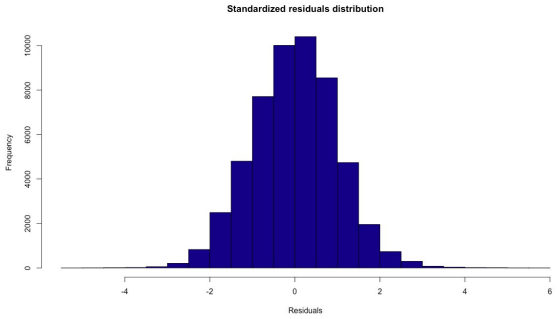
Perhaps some of the most interesting findings involve seasonality variables. The OLS estimates show that prices for listings during weekends are 6.8% higher than prices during weekdays. In addition, prices during winter are 34.6% lower than during the summer. In the spring, prices are

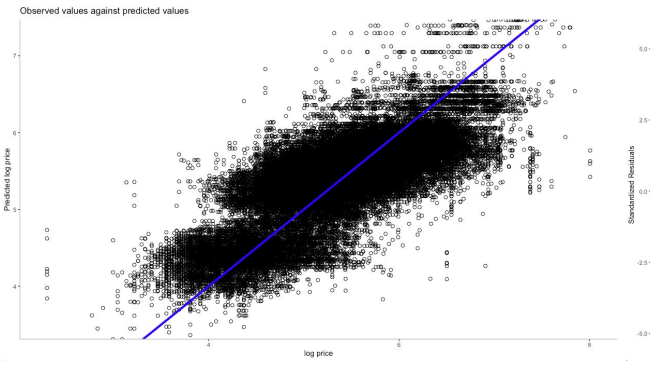
9% lower than during the summer. All the results reported here are statistically at the 5 percent level.

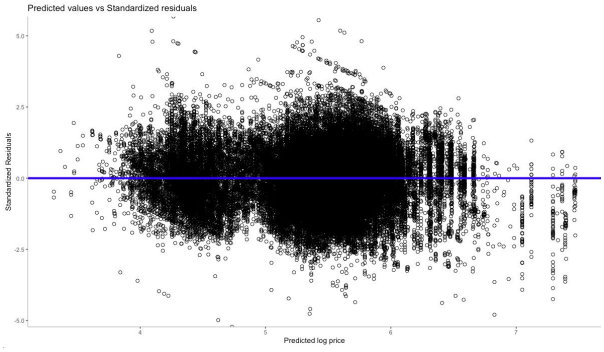
OLS Model Selection & Assumptions

Our model satisfies the inference assumptions of multiple linear regression. The plots below show linear relationship between price and the explanatory variables analyzed in our model. The relevant Normal Quantile plot (with almost linear residuals) and a histogram of the residuals below show that the residuals are normally distributed thus satisfying the assumption of normality. Furthermore, the Residuals vs Fitted plot shows that the residuals are randomly distributed and show no pattern. This suggests that the residuals in the final model satisfy the assumption of equal variance. Lastly, the Airbnb listings were scraped from the internet randomly hence we can assume this data represents normal Airbnb listings in Boston.









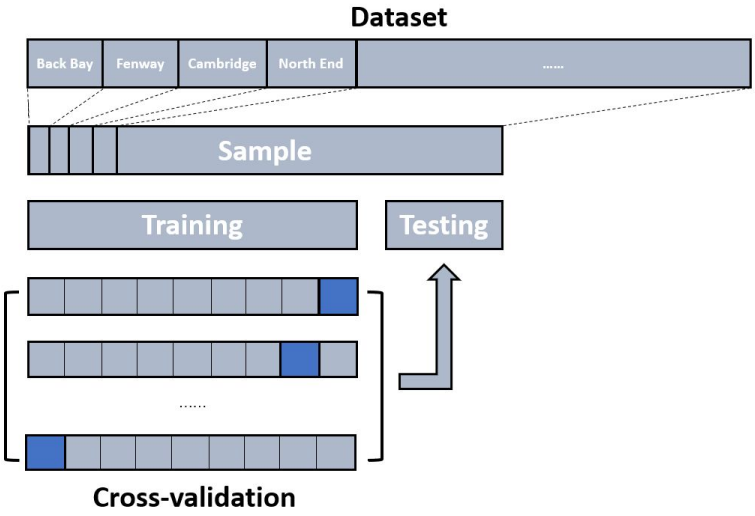
Machine Learning and Prediction

One disadvantage of OLS regression is that we cannot include all categorical variables. For example, we need to generate a lot of dummy variables if we want to capture all the features in our regression model. The regression will be too complicated. However, in machine learning

models, we can label the categorical variables and let the model to learn their distributions automatically. Hence, we can include many important features that were compelled to drop in regression model.

Sampling and Cross-validation

The sampling method we used in this part is the same as the one used in regression analysis. In this part, we will include the neighborhood variable that is dropped in previous analysis.



We use 10-fold cross-validation in our training process to avoid overfitting. The validation processes are shown above. Firstly, we split the dataset into 10 groups. For each unique group, we take the group as a test data set and the remaining groups as a training data set to train models, and adjust parameters based on evaluation metrics. After finishing cross-validation, we use testing set to evaluate model performances.

Lasso Regression

Lasso (least absolute shrinkage and selection operator; also Lasso or LASSO) is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model it produces.[7] All coefficients will be calculated by the formula below:

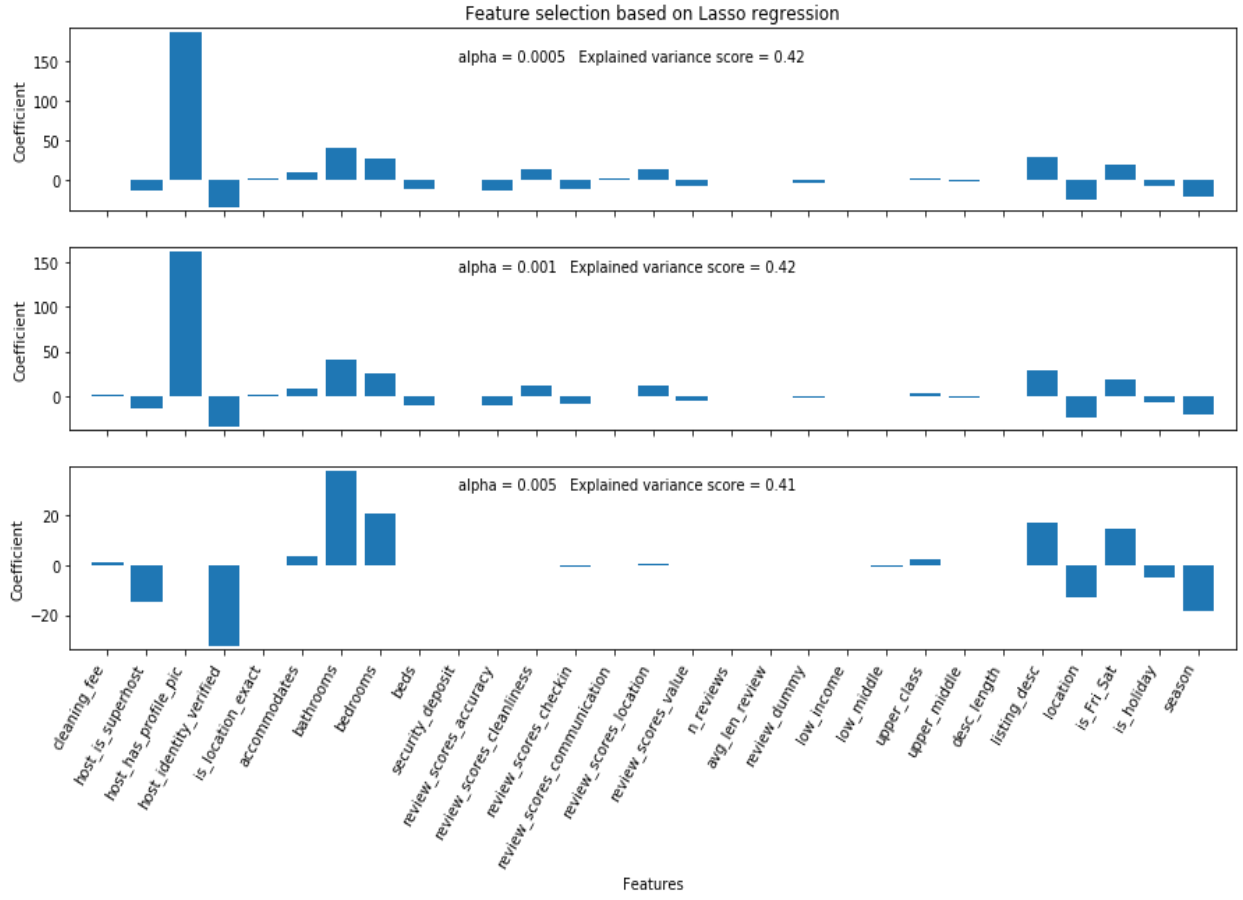
$$\hat{\beta}^{\text{lasso}} = \arg\min_{\beta} \sum_{n=1}^{N} \frac{1}{2}(y_n - \beta x_n)^2 + \alpha \sum_{i=1}^{p} |\beta_i|$$

[7] https://en.wikipedia.org/wiki/Lasso_(statistics)

The bar chart below shows model performances with different features selected by experimenting different parameter α. Difference of explained variance scores on testing set between different Lasso Regressions are very small, which indicates that by eliminating unimportant variables we will get a simpler model but with the same predicting accuracy.

The results of Lasso Regression also can be used in supporting the model selection in Regression And Analysis part. The selected features are highly consistent in these two models.

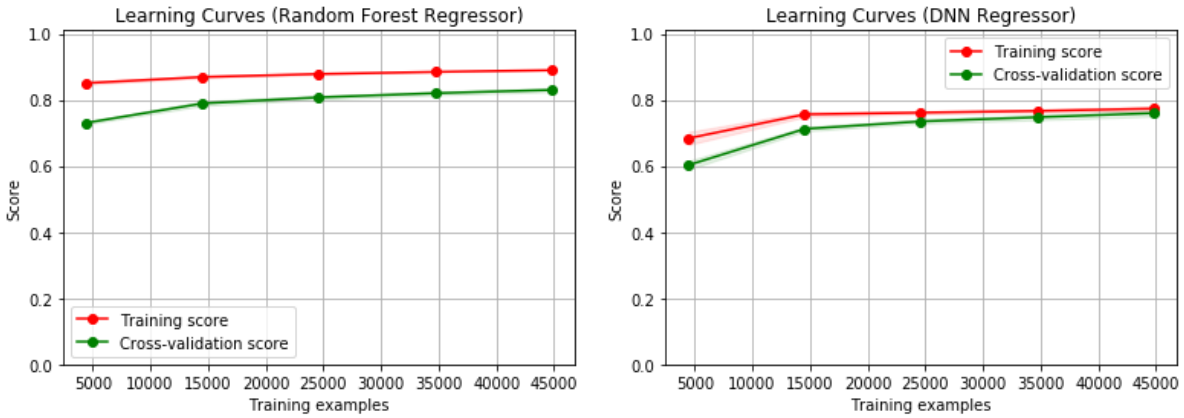
However, the explained variance score itself still seems unsatisfactory. We will experiment with other models to improve the predicting accuracy in the following sections.



Random Forest Regression & DNN Regression

The volume and number of features of Airbnb listing prices data set provide advantages for models like Random Forest and DNN, which are good at progressively extracting higher level features from the raw input, to systematically fit the distribution of prices. By using grid searching tools, we find the best parameters and hyperparameters for our models to achieve

the best scores[8]. In our results, scores of Random Forest Regression and DNN Regression are very close to each other. Score of DNN Regression seems a little bit lower, but still far better than Lasso Regression. With training sample increases, the best score of both model slowly goes up. Finally, DNN Regressor reaches an upper limit of about 0.78 and Random Fohu
rest Regressor reaches about 0.83.



By comparing training score and cross-validation score, we will see that differences between these two kinds of score are tiny, which means our models perform very well in generalization.

Machine Learning Findings

Feature selection in this part is different from the previous part:

1. In the previous analysis, we use income level to represent differences between neighborhoods, but in machine learning we can include them all.
2. There is no need to create a new variable to represent total income level. We can just put all information about income distributions in a certain area to our model, which helps model find more differences between locations.

These variables help us explain the price variations in detail. The findings in Regression and Analysis has clearly revealed relationships between most selected variables and price. Machine Learning models make further efforts to find the impact of location. And all results in previous analyses indicate that location information should be considered as one of the key factors in predicting price.

[8] Please refer to the appendix for further details.

Conclusion

Our analysis in this report shows that in Boston listings of private rooms and entire room apartments take a huge chunk out of all listings on Airbnb. More than 90% of all listings belong to these two categories.

Listings in neighborhoods that are closer to Boston city center have higher prices across all room types, with listings in financial district exceed more than $259 while listings in suburban areas such as Newton or Waltham are below $60.

Our analysis suggests that Friday and Saturday are usually the two most popular days of a week, while late spring and summer are the most popular seasons for Airbnb renting. The hosts can charge more during the 'hot' time. Besides, prices of listings slightly go down during holidays, and during that time of the year, hosts should take a break and relax instead of worrying about their listings on Airbnb.

Moreover, hosts should be cautious with their cleaning fee strategy, higher cleaning fees could disincentivize potential customer from renting. In addition, hosts should ensure that they appear trustworthy on Airbnb, i.e. they should have a profile picture on their accounts, have their account verified by Airbnb, and apply for superhost badge. Having a neat and trustworthy profile page not only means better business opportunities for the hosts, but also implies they can charge more for their trustworthiness. Tenants are generally willing to pay more for a listing that is posted by a trustworthy host than a murky posting that pops out of nowhere, even though the latter might be in the same neighborhood with similar amenities.

In addition, hosts should pay attention to what they write in listing description and neighborhood description. To feature collocations that indicate proximity to downtown Boston or to T-line stations will increase their chance to attract tenants, moreover, indicating closeness to downtown areas implies that hosts can charge slightly higher than market price. In fact, as shown in most popular bigrams in all comments, tenants are looking for location and transit convenience and are very willing to leave remarks on locations.

References

https://aresjournals.org/doi/abs/10.5555/rees.6.3.l532111124144176?journalCode=rees

https://datausa.io/profile/geo/boston-ma/

https://en.wikipedia.org/wiki/Lasso_(statistics)

https://www.census.gov/

Appendix

[Detailed regression results]
[Interactive map to see Airbnb listing density in Boston]

Code for Exploratory Analysis

```
Import numpy as np
import pandas as pd
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt
from matplotlib import gridspec
plt.style.use('seaborn-white')
sns.set_style("white")
other = ['price', 'id']
columns_01 = ['host_is_superhost', 'host_has_profile_pic',
              'host_identity_verified', 'is_location_exact']
columns_cat = ['room_type','accommodates', 'bathrooms', 'bedrooms', 'beds',
               'review_scores_accuracy', 'review_scores_cleanliness',
               'review_scores_checkin', 'review_scores_communication',
               'review_scores_location', 'review_scores_value']
columns_ctn = ['cleaning_fee', 'security_deposit', ]

def plot_01(col):
    plt.figure(figsize=(10, 10))
    plt.subplots_adjust(wspace = 0.15, hspace = 0.6)
    for i in range(len(columns_01)):
        gs = gridspec.GridSpec(len(columns_01), 2, width_ratios=[2, 1])
        plt.subplot(gs[i*2])
        ax = plt.gca()
        plt.title(columns_01[i])
        neg_class = df[col][df[columns_01[i]]==0].values
        pos_class = df[col][df[columns_01[i]]==1].values
        sns.kdeplot(neg_class, label='%s=0'%columns_01[i])
        sns.kdeplot(pos_class, label='%s=1'%columns_01[i])
        t, p = stats.ttest_ind(neg_class, pos_class, equal_var=False)
        ax.annotate('t = {:.2f} '.format(t), xy=(0, 1.05), xycoords=ax.transAxes)
        ax.annotate('p = {:.2f} '.format(p), xy=(0.15, 1.05),
xycoords=ax.transAxes)
        plt.subplot(gs[i*2+1])
        sns.boxplot(df[col].values, df[columns_01[i]].values,
                    orient='h', fliersize=0.25)
    plt.show()
```

```
•
•  df = pd.read_csv('listings_details.csv')
•  df = df[other+columns_01+columns_cat+columns_ctn].dropna()
•  df['price'] = df['price'].apply(lambda x: float(str(x).replace('$',
   '').replace(',', '')))
•  df['cleaning_fee'] = df['cleaning_fee']                    .apply(lambda x:
   float(str(x).replace('$', '').replace(',', '')))
•  df['security_deposit'] = df['security_deposit'] \
•                           .apply(lambda x: float(str(x).replace('$',
   '').replace(',', '')))
•  for col in columns_01: df[col] = df[col].apply(lambda x: {'t':1, 'f':0}[x])
•  df = df.groupby(['id','room_type']).mean().reset_index()
•  for col in columns_01: df[col] = df[col].apply(lambda x: int(round(x,0)))
•  df_desc = df.copy()[other+columns_01+columns_cat+columns_ctn]
•  df = df[(np.abs(stats.zscore(df['price'])) < 3)]
•
•  # descripe dummies
•  plot_01('price')
•
•  # descripe categorical var
•  for col in columns_cat:
•      if df[col].dtype is np.dtype('float64'): x = df[col].apply(int)
•      else: x = df[col]
•      plt.subplots(figsize=(5,3))
•      sns.boxplot(x, df['price'], fliersize=0.25)
•      plt.xlabel(col)
•      plt.ylabel('price')
•      plt.title('price over %s'%col)
•      plt.show()
•
•  # descripe continous var
•  for col in columns_ctn:
•      plt.subplots(figsize=(5,3))
•      plt.scatter(df[col], df['price'], s=0.25)
•      plt.xlabel(col)
•      plt.ylabel('price')
•      plt.title('price over %s'%col)
•      plt.show()
•
•  # summary table
•  pd.set_option('display.float_format',lambda x : '%.2f' % x)
•  df_desc[df.columns[1:]].describe().T
•
•  # room_type
•  df_desc[['price', 'room_type']].groupby('room_type')\
•                                 .agg(['count', 'mean', 'max', 'min', 'std'])
```

Code for Time Effects Analysis

```
import datetime
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats

df_calendar = pd.read_csv('calendar.csv')[['listing_id', 'date', 'price']].dropna()
df_calendar['price'] = df_calendar['price']                    .apply(lambda x:
float(x.replace('$', '').replace(',', '')))
df_calendar = df_calendar[(np.abs(stats.zscore(df_calendar['price'])) < 3)]
df_calendar.head()

df_calendar_mean = df_calendar[['listing_id', 'price']]
                   .groupby('listing_id', as_index=False).mean()
df_calendar_std = df_calendar[['listing_id', 'price']]
                  .groupby('listing_id', as_index=False).std()
coefficient_of_variation = df_calendar_std['price'] \
                           .values / df_calendar_mean['price'].values * 100

coefficient_of_variation_len = len(coefficient_of_variation)
for i in [15, 30, 50]:
    print('COV < %s%%: %.3f' % (i, (coefficient_of_variation < i).sum()
/ coefficient_of_variation_len))

# distplot cov
plt.xlim(0, 100)
sns.distplot(coefficient_of_variation, bins=100)
plt.title('Distribution of Coefficient of Variation')
plt.xlabel('Percentage')

df_calendar['standardize'] = (df_calendar['price']                              -
df_calendar.groupby('listing_id')['price'] \
                               .transform('mean'))                             /
df_calendar.groupby('listing_id')['price']
.transform('std')
df_calendar['standardize'] = df_calendar['standardize'].fillna(0)
df_calendar['date'] = pd.to_datetime(df_calendar['date'])
df_calendar['weekday'] = df_calendar['date'].dt.dayofweek
df_calendar['is_Fri_Sat'] = df_calendar['weekday'].apply(lambda x: x in [4, 5])
df_calendar['is_holiday'] = (df_calendar['date'].dt.month * 100 +
df_calendar['date'].dt.day)                                .apply(lambda x: x in [101,
121, 704, 902, 1128, 1225])
df_calendar['season'] = df_calendar['date'].dt.month         .apply(lambda x: {1:4,
2:4, 3:1, 4:1, 5:1, 6:2, 7:2, 8:2, 9:3, 10:3, 11:4, 12:4}[x])
df_drop_abnormal = df_calendar[(np.abs(stats.zscore(df_calendar['price'])) < 3)]
```

```
df_daily_price = df_drop_abnormal[['date', 'standardize']].groupby('date').mean()

# fluctuation over the whole time period
plt.figure(figsize=(9, 3))
plt.plot(df_daily_price)
plt.title('Price fluctuations')
plt.xlabel('Date')
plt.ylabel('Standardized price')
plt.show()

# fluctuation in Nov
plt.figure(figsize=(9, 3))
time_slice = df_daily_price['2019-11-01':'2019-11-30']
plt.plot(time_slice)
plt.xticks(time_slice.index, time_slice.index.strftime('%y.%m.%d %a'), rotation=90)
plt.title('An example: price fluctuations in November')
plt.xlabel('Date')
plt.ylabel('Standardized price')
plt.show()

# Standardized Price Distribution of Fri & Sat and Other
plt.xlim(-3, 3)
sns.distplot(df_calendar['standardize'][df_calendar['is_Fri_Sat']==True],
             fit_kws={"color":"tab:red"}, color='tab:red',
             bins=300, fit=stats.norm, kde=False, label='Fri & Sat')
sns.distplot(df_calendar['standardize'][df_calendar['is_Fri_Sat']==False],
             fit_kws={"color":"tab:blue"}, color='tab:blue',
             bins=300, fit=stats.norm, kde=False, label='Other')
plt.legend()
t, p = stats.ttest_ind(df_calendar['standardize'][df_calendar['is_Fri_Sat']==True],

df_calendar['standardize'][df_calendar['is_Fri_Sat']==False],
                       equal_var=False)
plt.annotate('t = {:.2f} '.format(t), xy=(-0.9, 0.1))
plt.annotate('p = {:.2f} '.format(p), xy=(0.2, 0.1))
plt.title('Standardized Price Distribution of Fri & Sat and Other')
plt.xlabel('Standardized Price')
plt.show()

# Standardized Price Distribution of Holiday and Normal Date
plt.xlim(-3, 3)
sns.distplot(df_calendar['standardize'][df_calendar['is_holiday']==True],
             fit_kws={"color":"tab:red"}, color='tab:red',
             bins=300, fit=stats.norm, kde=False, label='Holiday')
sns.distplot(df_calendar['standardize'][df_calendar['is_holiday']==False],
             fit_kws={"color":"tab:blue"}, color='tab:blue',
             bins=300, fit=stats.norm, kde=False, label='Normal')
plt.legend()
t, p = stats.ttest_ind(df_calendar['standardize'][df_calendar['is_holiday']==True],
```

```
df_calendar['standardize'][df_calendar['is_holiday']==False],
                        equal_var=False)
plt.annotate('t = {:.2f} '.format(t), xy=(-0.9, 0.1))
plt.annotate('p = {:.2f} '.format(p), xy=(0.2, 0.1))
plt.title('Standardized Price Distribution of Holiday and Normal Date')
plt.xlabel('Standardized Price')
plt.show()

# Standardized Price Distribution of Workday and Weekend
f, p = stats.f_oneway(*[df_calendar['standardize'][df_calendar['season']==i+1] for
i in range(4)])
plt.subplots(figsize=(9,9))
plt.suptitle('Standardized Price Distribution of Workday and Weekend\nf = {:.2f}, p
= {:.2f}'
              .format(f, p), y=0.94)
plt.subplot(221)
plt.xlim(-3, 3)
sns.distplot(df_calendar['standardize'][df_calendar['season']==1],
             fit_kws={"color":"tab:green"}, color='tab:green',
             bins=300, fit=stats.norm, kde=False, label='Spring')
plt.xlabel('Standardized Price')
plt.legend()
plt.subplot(222)
plt.xlim(-3, 3)
sns.distplot(df_calendar['standardize'][df_calendar['season']==2],
             fit_kws={"color":"tab:red"}, color='tab:red',
             bins=300, fit=stats.norm, kde=False, label='Summer')
plt.xlabel('Standardized Price')
plt.legend()
plt.subplot(223)
plt.xlim(-3, 3)
sns.distplot(df_calendar['standardize'][df_calendar['season']==3],
             fit_kws={"color":"tab:blue"}, color='tab:blue',
             bins=300, fit=stats.norm, kde=False, label='Autumn')
plt.xlabel('Standardized Price')
plt.legend()
plt.subplot(224)
plt.xlim(-3, 3)
sns.distplot(df_calendar['standardize'][df_calendar['season']==4],
             fit_kws={"color":"tab:grey"}, color='tab:grey',
             bins=300, fit=stats.norm, kde=False, label='Winter')
plt.xlabel('Standardized Price')
plt.legend()
plt.show()
```

Code For Text Analysis

```
import csv
import os
import math
import re
import string
import pandas as pd
import numpy as np
from pandas import DataFrame
import matplotlib.pyplot as plt
import nltk
from nltk.probability import FreqDist
from nltk.text import Text as text1
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer
from sklearn.linear_model import LinearRegression
from matplotlib.pyplot import figure

listing = pd.read_csv(r'/Users/xupech/Desktop/brandeis graduate school/2019 Data Competition/Data Sets/listings_details.csv', low_memory = False)

# get rid of all emojis in comments; collect results in a list with index.


price=listing['price']

""" name only, templist0 returns tokenized names, name_length returns length of each name."""
""" name only, templist1 returns tokenized names, desc_length returns length of each description."""




templist0={}
name_length = {}

templist1={}
desc_length=[]
for index, row in listing.iterrows():
    temp = str(row['name'])
    temp2 = str(row['description'])
    if len(str(temp))>0:
        temp.encode('ascii', 'ignore').decode('ascii')
        temp2.encode('ascii', 'ignore').decode('ascii')

        temp1 = nltk.word_tokenize(temp)
        temp3 = nltk.word_tokenize(temp2)
```

```
            length = len(temp1)
            length0 = len(temp3)
        else:
            temp1 = None
            temp3 = None
        templist0.update({index: temp1})
        name_length.update({index: length})

        templist1.update({index: temp3})
        desc_length.append(length0)

dummy_bigram=pd.DataFrame(desc_length, columns=['desc_length'])

""" combining name, summary, space, description together, templist returns
tokenized thing """
""" vocab is the vocaburary for above four columns, including punctuations/content
words"""
templist={}
vocab=[]
for index, row in listing.iterrows():
    temp = str(row['name'])+' '+str((row['summary']))+' '+str((row['space']))+'
'+str((row['description']))
    if len(str(temp))>0:
        temp.encode('ascii', 'ignore').decode('ascii')
        temp1 = nltk.word_tokenize(temp)
    else:
        temp1 = None
    templist.update({index: temp1})
    vocab+=temp1

""" return top 50 common non-content words used in the four columns combined """
STOPLIST = set(nltk.corpus.stopwords.words())
def is_content_word(word):
    return word.lower() not in STOPLIST and word[0].isalpha()


dist = nltk.FreqDist([w.lower() for w in vocab if is_content_word(w)])
freq2=dist.most_common(50)
# Oops, the words here are not informative. I will try bigrams instead.

""" bigrams, b_dict returns a dictionary of bigrams each row; b_vocab gives the
whole bigrams vocaburary """
b_dict={}
bivocab=[]
for index, row in templist.items():
    filtered_temp =[b for b in list(nltk.bigrams(row)) if is_content_word(b[0])
                    and is_content_word(b[1])]
    b_dict.update({index: filtered_temp})
    bivocab+=filtered_temp

```

```
dist1 = nltk.FreqDist([b for b in bivocab])
freq0 = dist1.most_common(50)


biig, biigfreq=zip(*freq0)

fig, ax = plt.subplots()
index = np.arange(len(biig))
bar_width = 0.25
opacity = 0.8

chart0 = ax.barh(index, biigfreq, bar_width, align = 'center', alpha=0.5)
for i, v in enumerate(biigfreq):
    ax.text(v+3, i, str(v), color='blue')

plt.yticks(index, biig)
plt.ylabel('Bigrams')
plt.title('Most Popular Bigrams In Name, Summary, Space And Description Columns Conbined')

plt.show()

fig.savefig('Most Popular Bigrams In Name, Summary, Space And Description Columns Conbined')

dist_desc=[]
listing_desc=[]
for key, value in b_dict.items():
    for bigram in value:
        n=0
        if bigram in biig:
            n+=1

        else:
            continue
    if n == 0:
        listing_desc.append(0)
        dist_desc.append(0)
    else:
        listing_desc.append(1)
        dist_desc.append(n)


dummy_bigram['listing_desc']=listing_desc


""" neighborhood and transit combined, replicate the process to have each line tokenized,
    create a vocaburary, get the non-content words distribution and bigrams distribution. """
```

```
templist2 = {}
tranvo = []
tb_dict={}
tbvocab=[]
for index, row in listing.iterrows():
    temp = str(row['neighborhood_overview'])+' '+str((row['transit']))
    if len(str(temp))>0 and temp != 'nan nan':
        temp.encode('ascii', 'ignore').decode('ascii')
        temp1 = nltk.word_tokenize(temp)
        templist2.update({index: temp1})
        tranvo+=temp1
    else:
        templist2.update({index: None})


""" return top 50 common non-content words in neighborhood and transit combined."""
dist2 = nltk.FreqDist([w.lower() for w in tranvo if is_content_word(w)])
freqtran=dist2.most_common(50)


for index, value in templist2.items():
    if value != None:
        filtered_temp =[b for b in list(nltk.bigrams(value)) if
(is_content_word(b[0])
                        and is_content_word(b[1]))]
        tb_dict.update({index: filtered_temp})
        tbvocab+=filtered_temp
    else:
        tb_dict.update({index: None})

distbit = nltk.FreqDist([b for b in tbvocab])
freqbit = distbit.most_common(50)


biit, biitfreq=zip(*freqbit)

fig, ax = plt.subplots()
index = np.arange(len(biit))
bar_width = 0.15
opacity = 0.8

chart0 = ax.barh(index, biitfreq, bar_width, align = 'center', alpha=0.5)
for i, v in enumerate(biitfreq):
    ax.text(v+3, i, str(v), color='blue')

plt.yticks(index, biit)
plt.ylabel('Bigrams')
plt.title('Most Popular Bigrams In Neighborhood And Transit Columns Conbined')

```

```
plt.show()
fig.savefig('Most Popular Bigrams In Neighborhood And Transit Columns Conbined')

location=[]
dist_location=[]
for key, value in tb_dict.items():
    if value != None:
        for bigram in value:
            n=0
            if bigram in biit:
                n+=1

            else:
                continue
        if n == 0:
            location.append(0)
            dist_location.append(0)
        else:
            location.append(1)
            dist_location.append(n)
    else:
        location.append(0)
        dist_location.append(0)

dummy_bigram['location']=location


review = pd.read_csv(r'/Users/xupech/Desktop/brandeis graduate school/2019 Data Competition/Data Sets/reviews_details.csv', low_memory = False)
review_token={}
review_length={}
review_ave_length=[]
review_number ={}
reviewv = []
review_bigram ={}
rbivo =[]

""" generate review_token as a dict with listing_id as keys and all tokens in the listing as values. """
""" generate review_length to record length of each review in a list, list is stored as values in a dict. """
""" generate review_number as a dict to record number of reviews per listing."""
""" generate a reviewv for all vocaburary in reviews. """


for index, row in review.iterrows():
    tempin = int(row['listing_id'])
    temp = str(row['comments'])
    if len(str(temp))>0:
        temp.encode('ascii', 'ignore').decode('ascii')
```

```
            temp1 = nltk.word_tokenize(temp)
            templen = [len(temp1)]
        else:
            temp1 = None
            templen = 0

        if tempin not in review_token.keys():
            review_token.update({tempin: temp1})

        else:
            review_token[tempin]+=temp1

        if tempin not in review_length.keys():
            review_length.update({tempin: templen})
        else:
            review_length[tempin]+=templen

        reviewv+=temp1


    for index, value in review_length.items():
        temp = len(value)
        review_number.update({index: temp})
        review_ave_length.append(int(np.mean(value)))


    # most frequent words.
    distrev = nltk.FreqDist([w.lower() for w in reviewv if is_content_word(w)])
    freqrev=distrev.most_common(50)

    for index, row in review_token.items():
        filtered_temp =[b for b in list(nltk.bigrams(row)) if is_content_word(b[0])
                        and is_content_word(b[1])]

        review_bigram.update({index: filtered_temp})
        rbivo+=filtered_temp

    distbrt = nltk.FreqDist([b for b in rbivo])
    freqbrt = distbrt.most_common(50)


    brt, brtfreq=zip(*freqbrt)

    fig, ax = plt.subplots()
    index = np.arange(len(brt))
    bar_width = 0.15
    opacity = 0.8

    chart0 = ax.barh(index, brtfreq, bar_width, align = 'center', alpha=0.5)
    for i, v in enumerate(brtfreq):
```

```
    ax.text(v+3, i, str(v), color='blue')

plt.yticks(index, brt)
plt.ylabel('Bigrams')
plt.title('Most Popular Bigrams In Comments')

plt.show()
fig.savefig('Most Popular Bigrams In Comments')


review_dummy=[]

dist_review=[]
for key, value in review_bigram.items():
    for bigram in value:
        n=0
        if bigram in brt:
            n+=1

        else:
            continue
    if n == 0:
        review_dummy.append(0)
        dist_review.append(0)
    else:
        review_dummy.append(1)
        dist_review.append(n)

reviewdf = pd.DataFrame(list(review_number.items()), columns=['listing
index','number of reviews'])
reviewdf['average length of reviews per listing']=review_ave_length
reviewdf['review_dummy']=review_dummy




reviewdf.to_csv('review_text_prcessed.csv', index=False)

dummy_bigram.to_csv('listing_dummy and location_dummy.csv', index=False)


del dummy_bigram['std index']
```

Code for Feature Engineering and Sampling

```
Import re
import numpy as np
import pandas as pd

pd.set_option('display.max_columns', 100)
ori_feature = [
    'id', 'neighbourhood','cleaning_fee', 'host_is_superhost',
'host_has_profile_pic',
    'host_identity_verified', 'is_location_exact', 'accommodates', 'bathrooms',
    'bedrooms', 'beds', 'room_type', 'security_deposit', 'zipcode'
    'review_scores_accuracy', 'review_scores_cleanliness', 'review_scores_checkin',
    'review_scores_communication', 'review_scores_location', 'review_scores_value'
]

def re_sub(content):
    if str(content) == 'nan': return 0
    content = re.sub(r'^[A-Z].*? ', '', content)
    content = re.sub(r' .*?$', '', content)
    content = re.sub(r'-.*?$', '', content)
    return int(content)

listing_details = pd.read_csv('listings_details.csv')[ori_feature]
review_text_prcessed = pd.read_csv('review_text_prcessed.csv')
listing_location_dummy = pd.read_csv('listing_location_dummy.csv')
income_by_zipcode = pd.read_csv('Income_by_zipcode.csv')
df_calendar = pd.read_csv('calendar.csv')[['listing_id', 'date', 'price']].dropna()

review_text_prcessed.columns = ['listing index', 'n_reviews', 'avg_len_review',
'review_dummy']

listing_details['cleaning_fee'] = listing_details['cleaning_fee'].fillna('0') \
                                  .apply(lambda x: float(x.replace('$',
'').replace(',', '')))
listing_details['security_deposit'] =
listing_details['security_deposit'].fillna('0') \
                                       .apply(lambda x: float(str(x).replace('$',
'')
                                                                .replace(',',
'')))
listing_details['host_is_superhost'] = listing_details['host_is_superhost']\
                                        .apply(lambda x: 1 if x=='t' else 0)
listing_details['host_has_profile_pic'] = listing_details['host_has_profile_pic']\
                                           .apply(lambda x: 1 if x=='t' else 0)
listing_details['host_identity_verified'] =
listing_details['host_identity_verified']\
                                            .apply(lambda x: 1 if x=='t' else 0)
```

```
listing_details['is_location_exact'] = listing_details['is_location_exact']\
                                        .apply(lambda x: 1 if x=='t' else 0)
listing_details['zipcode'] = listing_details['zipcode'].apply(re_sub)
listing_details = listing_details.fillna(0)

df_calendar['price'] = df_calendar['price']\
                        .apply(lambda x: float(x.replace('$', '').replace(',',
''')))
df_calendar['date'] = pd.to_datetime(df_calendar['date'])
df_calendar['weekday'] = df_calendar['date'].dt.dayofweek
df_calendar['is_Fri_Sat'] = df_calendar['weekday'].apply(lambda x: x in [4, 5])
df_calendar['is_holiday'] = (df_calendar['date']\
                             .dt.month * 100 + df_calendar['date'].dt.day)\
                             .apply(lambda x: x in [101, 121, 704, 902, 1128,
1225])
df_calendar['season'] = df_calendar['date'].dt.month\
                    .apply(lambda x: {1:4, 2:4, 3:1, 4:1,
                                      5:1, 6:2, 7:2, 8:2,
                                      9:3, 10:3, 11:4, 12:4}[x])
df_calendar = df_calendar[['listing_id', 'price', 'is_Fri_Sat', 'is_holiday',
'season']]
df_calendar['is_Fri_Sat'] = df_calendar['is_Fri_Sat']*1
df_calendar['is_holiday'] = df_calendar['is_holiday']*1

listing_details['neighbourhood'][listing_details['neighbourhood']==0] = '0'
count_by_neighbourhood = listing_details[['neighbourhood', 'id']]
                         .groupby('neighbourhood').agg(['count'])
index_by_neighbourhood = list(count_by_neighbourhood.index)
neighbourhood_dict = {}
for i in range(len(index_by_neighbourhood)):
    if count_by_neighbourhood.values[i] < 100:
        neighbourhood_dict[index_by_neighbourhood[i]] = '0'
    else:
        neighbourhood_dict[index_by_neighbourhood[i]] = index_by_neighbourhood[i]

listing_details['neighbourhood'] = listing_details['neighbourhood']\
                                   .apply(lambda x: neighbourhood_dict[x])

listing_details_unique = listing_details.groupby(['id',
'neighbourhood','room_type'])
                                        .mean().reset_index()
neighbourhood_cat = listing_details_unique['neighbourhood'].drop_duplicates()
sample_list = list(map(lambda x:
listing_details_unique[listing_details['neighbourhood']==x]
                       .sample(frac=0.15, random_state=0), neighbourhood_cat))
sample_concat = pd.concat(sample_list)
income_table = income_by_zipcode.pivot_table('income_dist', 'zip_code',
'income_category')
                                .reset_index()
merge_table = pd.concat([
```

```
    sample_concat.merge(review_text_prcessed, left_on='id',
                        right_on='listing index', how='left') \
                .merge(income_table, left_on='zipcode', right_on='zip_code',
how='left'),
    listing_location_dummy],  axis=1)
#merge_table_clean = pd.get_dummies(merge_table.dropna()).drop(['listing index',
'zip_code'], axis=1)
merge_table_clean = merge_table.dropna().drop(['listing index', 'zip_code'],
axis=1)

final_dataset = merge_table_clean.merge(df_calendar, left_on='id',
                                        right_on='listing_id', how='left')
final_dataset = final_dataset.dropna()
final_dataset = final_dataset.dropna().drop(['listing_id', 'zipcode'], axis=1)
final_dataset = final_dataset.drop(['id'], axis=1)
dict_neighbourhood = {n:i+1 for i, n in enumerate(final_dataset['neighbourhood']\
                                                  .drop_duplicates())}
final_dataset['neighbourhood'] = final_dataset['neighbourhood'].apply(lambda x:
dict_neighbourhood[x])
dict_apt = {n:i+1 for i, n in
enumerate(final_dataset['room_type'].drop_duplicates())}
final_dataset['room_type'] = final_dataset['room_type'].apply(lambda x:
dict_apt[x])
final_dataset.to_csv('sample_ml.csv', index=False)
final_dataset['reg_columns'].to_csv('sample_reg.csv', index=False)
final_dataset.to_csv('large_sample_ml.csv', index=False)
```

Code for Machine Learning and Predicting

```
Import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import explained_variance_score
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV

df = pd.read_csv('large_sample_ml.csv')
df = df[(np.abs(stats.zscore(df['price'])) < 2)]

df_columns = [x for x in df.columns if x != 'price']
y = df.price.values.reshape(-1, 1)
X = df[df_columns].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0)
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

df_lasso = pd.get_dummies(df[df.columns[2:]], drop_first=True)
df_columns = [x for x in df_lasso.columns if x != 'price']
y = df_lasso.price.values.reshape(-1, 1)
X = df_lasso[df_columns].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0)

plt.subplots_adjust(hspace = 0)
plt.subplots(figsize=(15, 7.5))
lasso = Lasso(alpha=0.0005, normalize=True)
lasso.fit(X_train, y_train)
vs = round(explained_variance_score(y_test, lasso.predict(X_test)), 2)
plt.subplot(311)
plt.bar(range(len(df_columns)), lasso.coef_)
plt.xticks(range(len(df_columns)), ['']*len(df_columns))
plt.margins(0.02)
plt.title('Feature selection based on Lasso regression')
plt.ylabel('Coefficient')
plt.annotate('alpha = 0.0005   Explained variance score = %s'%vs, xy=(9, 150))
```

```
•
•   lasso = Lasso(alpha=0.001, normalize=True)
•   lasso.fit(X_train, y_train)
•   vs = round(explained_variance_score(y_test, lasso.predict(X_test)), 2)
•   plt.subplot(312)
•   plt.bar(range(len(df_columns)), lasso.coef_)
•   plt.xticks(range(len(df_columns)), ['']*len(df_columns))
•   plt.margins(0.02)
•   plt.ylabel('Coefficient')
•   plt.annotate('alpha = 0.001   Explained variance score = %s'%vs, xy=(9, 140))
•
•   lasso = Lasso(alpha=0.005, normalize=True)
•   lasso.fit(X_train, y_train)
•   vs = round(explained_variance_score(y_test, lasso.predict(X_test)), 2)
•   plt.subplot(313)
•   plt.bar(range(len(df_columns)), lasso.coef_)
•   plt.xticks(range(len(df_columns)), df_columns, rotation=60, ha='right')
•   plt.margins(0.02)
•   plt.ylabel('Coefficient')
•   plt.xlabel('Features')
•   plt.annotate('alpha = 0.005   Explained variance score = %s'%vs, xy=(9, 30))
•   plt.show()
•
•   from sklearn.model_selection import learning_curve
•   from sklearn.model_selection import ShuffleSplit
•
•   def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
•                           n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
•       scaler = StandardScaler()
•       scaler.fit(X)
•       X = scaler.transform(X)
•       plt.figure()
•       plt.title(title)
•       if ylim is not None:
•           plt.ylim(*ylim)
•       plt.xlabel("Training examples")
•       plt.ylabel("Score")
•       train_sizes, train_scores, test_scores = learning_curve(
•           estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
•       train_scores_mean = np.mean(train_scores, axis=1)
•       train_scores_std = np.std(train_scores, axis=1)
•       test_scores_mean = np.mean(test_scores, axis=1)
•       test_scores_std = np.std(test_scores, axis=1)
•       plt.grid()
•       plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
•                        train_scores_mean + train_scores_std, alpha=0.1,
•                        color="r")
•       plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
•                        test_scores_mean + test_scores_std, alpha=0.1, color="g")
•       plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
```

```
                label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
                label="Cross-validation score")

    plt.legend(loc="best")
    return plt

clf = MLPRegressor(solver='adam', alpha=1e-5, hidden_layer_sizes=(30, 30, 30))
clf.fit(X_train, y_train)
explained_variance_score(y_test, clf.predict(X_test))

clf = MLPRegressor(solver='adam', alpha=1e-5, hidden_layer_sizes=(30, 30, 30))
title = r"Learning Curves (DNN Regressor)"
cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=0)
plot_learning_curve(clf, title, X, y, (0.0, 1.01), cv=cv, n_jobs=4)

from sklearn.ensemble import RandomForestRegressor

gbr = RandomForestRegressor(max_features='auto', min_samples_leaf=5
                            , min_samples_split=2, n_estimators=80)
gbr.fit(X_train, y_train)
explained_variance_score(y_test, gbr.predict(X_test))

gbr = RandomForestRegressor(max_features='auto', min_samples_leaf=5
                            , min_samples_split=2, n_estimators=80)
title = r"Learning Curves (DNN Regressor)"
cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=0)
plot_learning_curve(gbr, title, X, y, (0.0, 1.01), cv=cv, n_jobs=4)

param_grid = {
    'n_estimators': np.arange(60, 100, 5),
    'max_features': ['log2', 'auto'],
    'min_samples_split': np.arange(2, 20, 3),
    'min_samples_leaf': np.arange(2, 20, 3)
}
kflod = StratifiedKFold(n_splits=5, shuffle=True, random_state=7)
grid_search = GridSearchCV(gbr, param_grid, n_jobs=-1, cv=kflod)
grid_search.fit(X_train, y_train)

grid_search.best_params_
```