DATA SCIENCE FOR BIZ - Airbnb Pricing Prediction

Team-11: Claire Park, Kidae Hong, Prerna Mishra, Rahul Singh

Abstract

Airbnb is a commonly used platform across the world for hosting and booking short-term rentals. In this project, we are trying to estimate the “right price” for Airbnb listings in New York City using machine learning techniques such as Linear Regression, K-Nearest Neighbors, Random Forest, etc. The “right price” here can be defined as the price at which the host maximizes their profit without compromising on the popularity of the listing.

The Random Forest model returned the best results with R^2 values of 0.533 and Mean Absolute Error of 36.955. The following table lists the various R^2 values that we got for the different predictive models that we tried:

Prediction Model	**R^2**	**MAE**
Default KNN (K-nearest Neighbors Regression)	39.195444	0.505803
Best KNN (Using optimal K for KNN Regression)	39.195444	0.517412
Default SVR (Support Vector Regression)	40.976521	0.374812
Best SVR	40.964648	0.375094
Decision Tree	40.979619	0.343876
Random Forest	36.955923	0.533239
Linear Regression	42.825844	0.461389
Ridge Regression	42.504782	0.457371
Elastic Net Regression	42.537543	0.431806

Introduction

Airbnb prices are usually determined by the hosts empirically. It is equally challenging for a new host or an experienced host to set the prices reasonably without losing popularity or leaving money on the table. On the other side, renters can compare the price across other similar listings and determine whether the current price is justified and if it is a good deal for them to book the rooms.

Business Understanding

One challenge that Airbnb hosts face is determining the optimal price for their apartment. Renters have an option to choose from a wide array of selection and can choose the apartment to their liking based on various criteria such as price, number of bedrooms and bathrooms, room type, location, etc.

Let's consider we have an apartment that we'd like to list on Airbnb. As hosts, if we try to charge a price above the market price, renters will select more affordable alternatives and we won't make any money. On the other hand, if we set our price too low, we'll miss out on potential revenue and leave money on the table. This is a double-edged sword. Ideally, we'd like to price our apartment in such a way that we consistently attract renters for our apartment.

So, how can we find the "sweet spot" to maximize our revenue? An easy fix approach that we can do is as follows:

1. Find listings that are similar to ours: no. of beds, baths, room type etc.
2. Calculate the average price in the locality and for apartments similar to ours
3. Use this calculated average price to set a comparable price (competitive pricing)

- But is this the ideal approach?

Let's start by answering the question we posted to you above. No, this is not the ideal approach! This is a manual task and can be really time consuming. But the market is dynamic, so the prices need to be updated regularly and determining the average price is tedious each time. Moreover, this approach is prone to errors in data and outliers and significantly offset the average price leading to either underpricing or overpricing, either of which we don't want to happen. The biggest flaw in this approach is what if the prices set already are **not** the best prices for our reference apartments? In this case, we will end up pricing our apartment incorrectly. Most importantly, we're data scientists! So, instead of manually setting up the price, we're going to build a model that can identify the key features and help us find the optimal price for our apartment listing.

- How (precisely) will a data mining solution address the business problem?

Airbnb provides monthly updates on all the listings. There are large datasets being collected from the Airbnb listings with rich features. Once we have the model set up, we can import the latest data and set the optimal price for a listing. This is less time consuming and is less prone to error as well. In our model, outliers are removed and any noise in the data has been removed. The most

important aspect to note here is that we are only looking at the top-25% of the listings, for which we believe the hosts have figured out the right price for their apartments. Here's our approach:

We created a new parameter called "**popularity**". Our approach is to measure popularity using review ratings * monthly number of reviews and select top 25% and make a model to predict the price of the most popular listings. Our interpretation is that the popular listings have figured out the price, so they are a decent proxy for the "right" price. Thus, predicting price for these listings is a reasonable surrogate problem for predicting the right price.

Literature Review and Related Work

When it comes to predicting prices for AirBnb listings, there has been some prior related work done. In 2019, Luo et al. [1] in their project *Predicting Airbnb Listing Price Across Different Cities*, estimated the Airbnb listing price in three cities – New York City (NYC), Paris and Berlin with various machine learning approaches. They achieved r-squared values of 0.769 in train and 0.741 in test on the NYC dataset, values of 0.762 in train and 0.716 in test on the Paris dataset, and values of 0.816 in train and 0.773 in test on the combined dataset using neural networks.

In 2017, Wang et al. [2] used Airbnb datasets from 33 cities and a sample of 180,533 listings using ordinary least squares and quantile regression analysis to identify 25 price determinants. Teubner et al. [3] used linear regression to carry out a similar analysis on reputation-related features and their effect on pricing. In 2019, Kalehbasti et al.[4] used machine learning and sentiment analysis to predict Airbnb prices using the NYC dataset. They achieved an r-squared value of 0.69 on their test dataset. Other analyses and projects related to predicting AirBnb prices have been carried out as well and available for reference in the open internet. While we have studied and referenced them, we have tried to achieve the best results using our "popularity" approach.

Data Understanding

For this analysis, we use Airbnb monthly listings data on the Inside Airbnb website. More specifically, we are going to use the detailed data for detailed listings data for New York city, which is compiled on Feb 4th, 2021. (Data Link)

This dataset has 37,012 listings data with 74 attributes including id, listing_url, name, neighborhood, host_id, latitude and longitude, property_type, room_type, bathrooms, bedrooms, price, number of reviews, review_score, reviews_per_month and etc.

Data Preparation

Data Cleaning

Although there are 74 attributes for each data instance, for the purpose of this project, we selected **13 attributes** for each listing based on complexity of our dataset and our assumptions about which aspect might be related to the popularity of an Airbnb listing.

Here are descriptions of selected attributes for the data mining process:

1. Neighborhood_group_cleansed: One of the 5 boroughs of NY that each listing is located in.
2. Host_is_superhost: Whether of not the host is the 'Super host' on Airbnb (T/F)
3. Room_type: One of "Entire home/apt", "Private room", "Shared room" or "Hotel room".
4. Accommodates: The number of guests a listing can accommodate
5. Bathrooms_text: The number of bathrooms a listing offers.
6. Bedrooms: The number of bedrooms a listing offers.
7. Beds: The number of beds a listing offers.
8. Price: The price of the listing offers.
9. Availability_365: The number of days for which a particular host is available in a year.
10. Minimum_nights: The minimum stay for a visit, as posted by the host.
11. Maximum_nights: The maximum stay for a visit, as posted by the host.
12. Review_scores_rating: The average review score of each listing.
13. Reviews_per_month: The number of reviews that a listing has received per month.

When we imported the original data and looked into it, we could see that we have more than 37,000 listings in the data. Also, there are 73 columns or features, some of which are not necessary. We can also see that the data contains Null values.

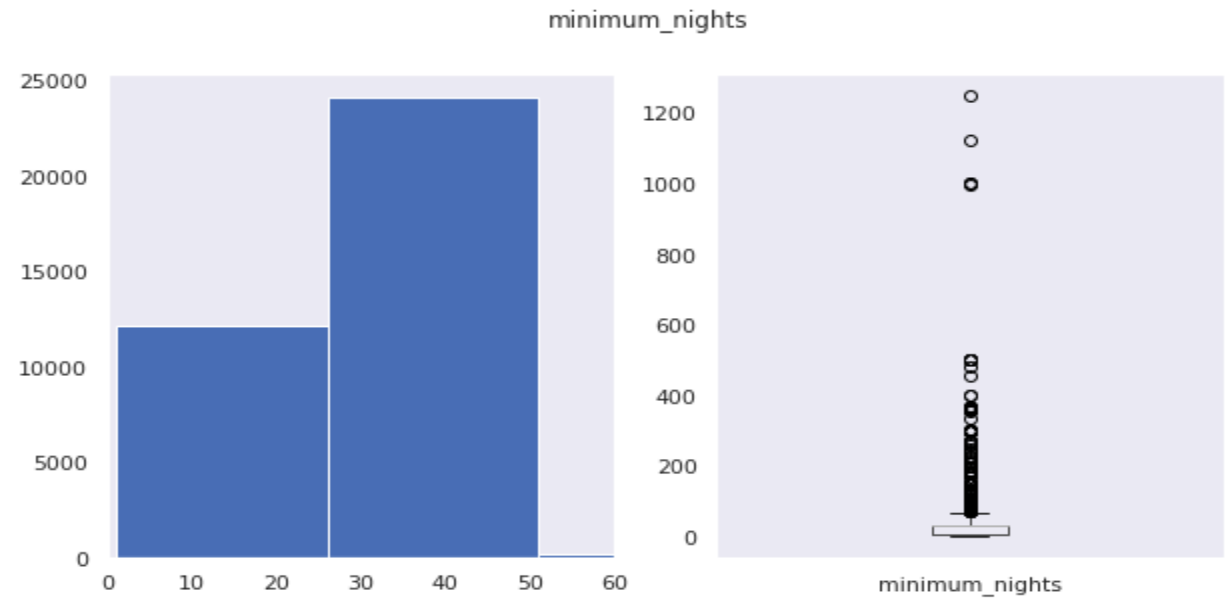
The logical next step is to clean the data and keep only those features which are relevant, and handle the missing data (null values). When we checked the null values in each column, we got the following result:

```
neighbourhood_cleansed              0
neighbourhood_group_cleansed        0
latitude                            0
longitude                           0
host_is_superhost                  18
room_type                           0
accommodates                        0
bathrooms_text                    102
bedrooms                         3608
beds                              490
price                               0
availability_365                    0
minimum_nights                      0
maximum_nights                      0
review_scores_rating            10235
reviews_per_month                9523
dtype: int64
```

We can see that the columns host_is_superhost, bathrooms_text, bedrooms, beds, review_scores_rating, and reviews_per_month have null values. The review_scores_rating has the maximum number of missing values, which is almost a third of the total data points. We're replacing the features beds and bedrooms with the mean of the columns and all other column values with zero to impute the missing values.

Once we were done taking care of the missing values, let's look at the outliers. First we looked at the minimum_nights variable.

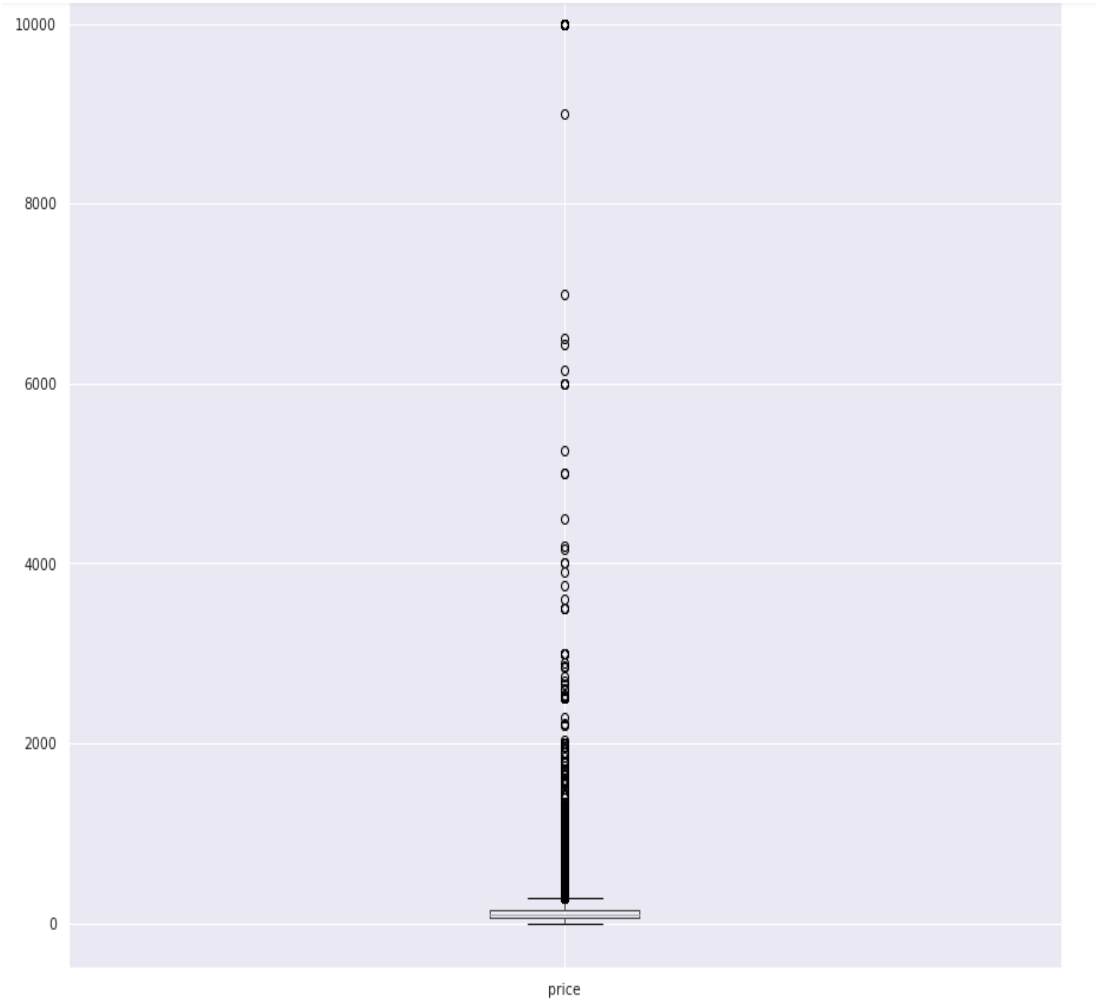
```
minimum_nights max value  1250
minimum_nights min value  1
```



Clearly, a minimum 1250 nights seems absurd given that AirBnb is a short-term rentals hosting platform. For further analysis, we will limit the maximum value of minimum nights at 50 nights, which is more than 98th percentile of the data.

The next, and probably the most important column that we checked for outliers was the Price column. The box plot shown below shows that there are many outliers in price. We decided to drop these outliers. We decided to keep 99.5% percentile of the data and drop the data in the long tail. Hence, we limited the maximum value of minimum nights at $1200, which is the 99.5th percentile of the data.

Once we did that, the skewness in the data has dropped significantly from 21.81 to 3.51. This meant that the tail (outliers) in the data had been trimmed and the data is more uniformly distributed.



Feature Engineering

The next step of our project was to do feature engineering for two purposes:

1. Preparing the data to be compatible with the machine learning algorithm requirements
2. Improving the performance of machine learning models

To begin, we manipulated the data type of 'host_is_superhost' for the easier analysis. We converted the data type from object to int and then converted it to a Binary variable: 0 for False and 1 for True.

```
neighbourhood_cleansed          object
neighbourhood_group_cleansed    object
latitude                       float64
longitude                      float64
host_is_superhost               object
room_type                       object
accommodates                     int64
bathrooms_text                  object
bedrooms                       float64
beds                           float64
price                          float64
availability_365                 int64
minimum_nights                   int64
maximum_nights                   int64
review_scores_rating           float64
reviews_per_month              float64
dtype: object
```

```
neighbourhood_cleansed          object
neighbourhood_group_cleansed    object
latitude                       float64
longitude                      float64
host_is_superhost                int64
room_type                       object
accommodates                     int64
bathrooms_text                  object
bedrooms                       float64
beds                           float64
price                          float64
availability_365                 int64
minimum_nights                   int64
maximum_nights                   int64
review_scores_rating           float64
reviews_per_month              float64
popularity                     float64
dtype: object
```

Next, we created a new parameter called **"popularity"**. This was probably the most important step of our project. Our approach is to measure popularity using review ratings * monthly number of reviews and select the top 25%, and then make a model to predict the price of the most popular listings. Our interpretation is that the popular listings have figured out the price, so they are a decent proxy for the "right" price. Thus, predicting price for these listings is a reasonable surrogate problem for predicting the right price. When we selected the **Top 25%** listings based on the popularity score, we ended up with 6585 listings.

After shortlisting the listings to carry out further analysis, we checked the categorical variables viz. 'neighbourhood_cleansed','neighbourhood_group_cleansed','room_type', 'bathrooms_text'.

The bathrooms_text seems to have too many unique values for the analysis.

	neighbourhood_cleansed	neighbourhood_group_cleansed	room_type	bathrooms_text
count	6585	6585	6585	6585
unique	199	5	4	23
top	Bedford-Stuyvesant	Brooklyn	Entire home/apt	1 bath
freq	638	2690	3548	3243

When we looked into the data in more detail, we were able to find why it is. The data was more of a descriptive form of the number and type of bathroom of each listing, and included both private and shared bathrooms.

```
array(['1 bath', '1 private bath', '1 shared bath', '3 baths',
       '1.5 shared baths', '2 shared baths', '0 shared baths', '2 baths',
       '1.5 baths', 'Half-bath', '2.5 baths', '4.5 baths', 0, '5.5 baths',
       '3 shared baths', 'Shared half-bath', '4 shared baths',
       '3.5 baths', '4 baths', '0 baths', '2.5 shared baths',
       'Private half-bath', '5 baths'], dtype=object)
```

We know that private bathrooms are preferred over private bathrooms and this is an important factor in decision making. So we made a new column for the number of bathrooms and whether it is shared or not. We filled the missing values with zero. This was the final step of our feature engineering

Finally formed a dataframe called ***'new_pop'*** to carry out data exploration and machine learning.

```
top25_pop.dtypes
```

```
neighbourhood_cleansed          object
neighbourhood_group_cleansed    object
latitude                       float64
longitude                      float64
host_is_superhost                int64
room_type                       object
accommodates                     int64
bathrooms_text                  object
bedrooms                       float64
beds                           float64
price                          float64
availability_365                 int64
minimum_nights                   int64
maximum_nights                   int64
review_scores_rating           float64
reviews_per_month              float64
popularity                     float64
bathrooms                      float64
bathrooms_share                  int64
dtype: object
```

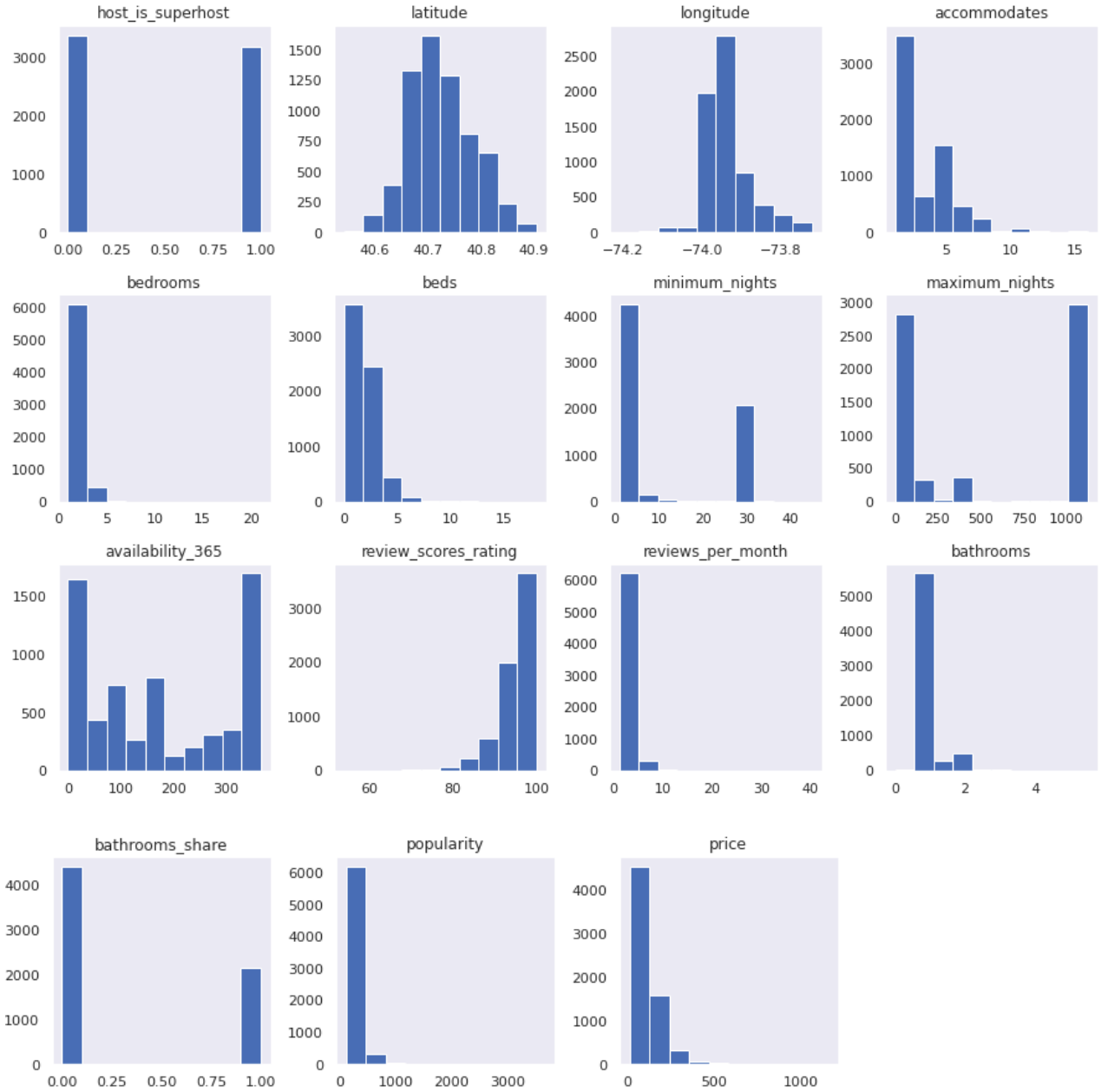
Data Exploration

At this point, we decided to explore our data to understand and visualize the data, find any correlation between variables and to uncover different insights.

Here are some of the visualizations you can check on our analysis jupyter notebook.

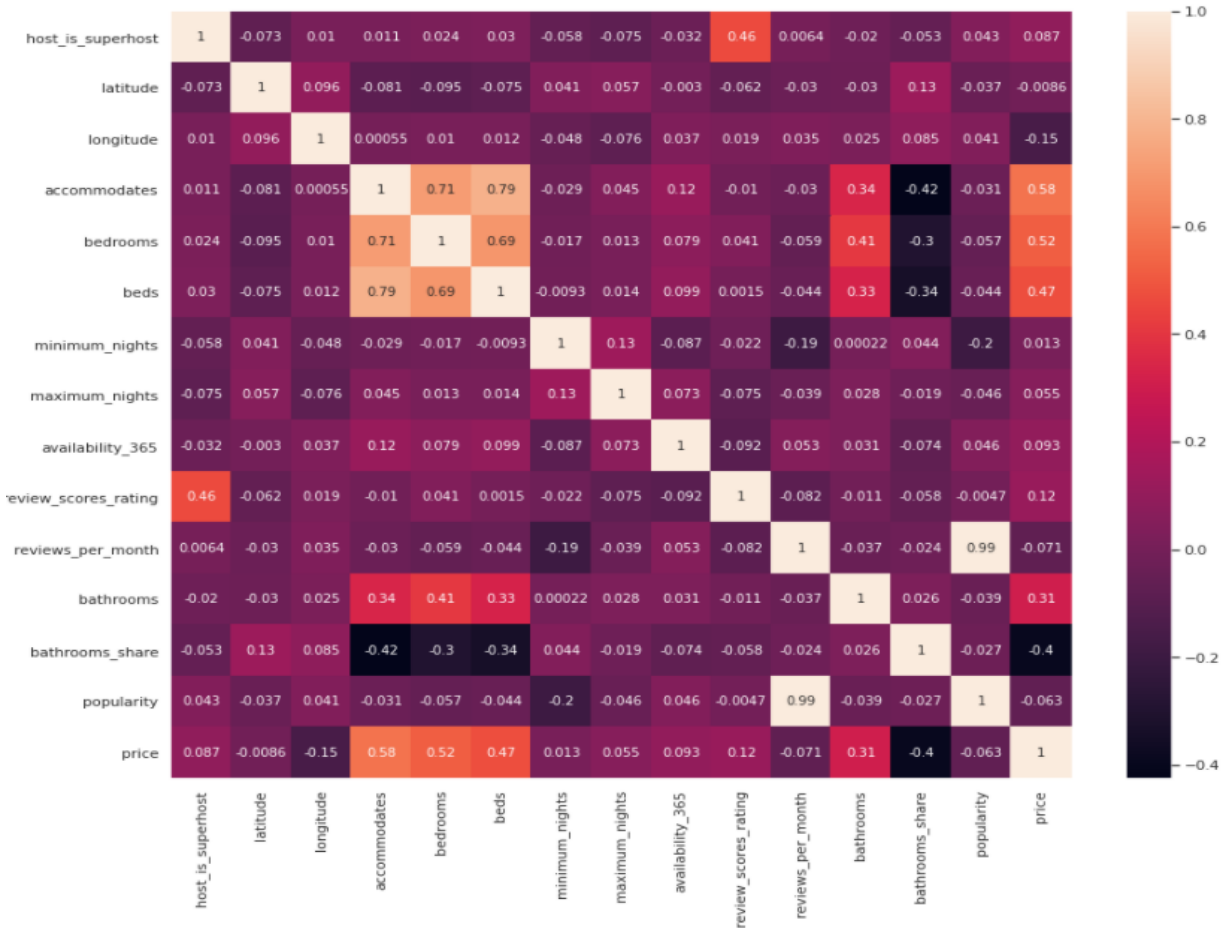
- Distribution of each column:

We visualized all the features in the data set to see the distribution:



- Correlation graphics between each column and the price

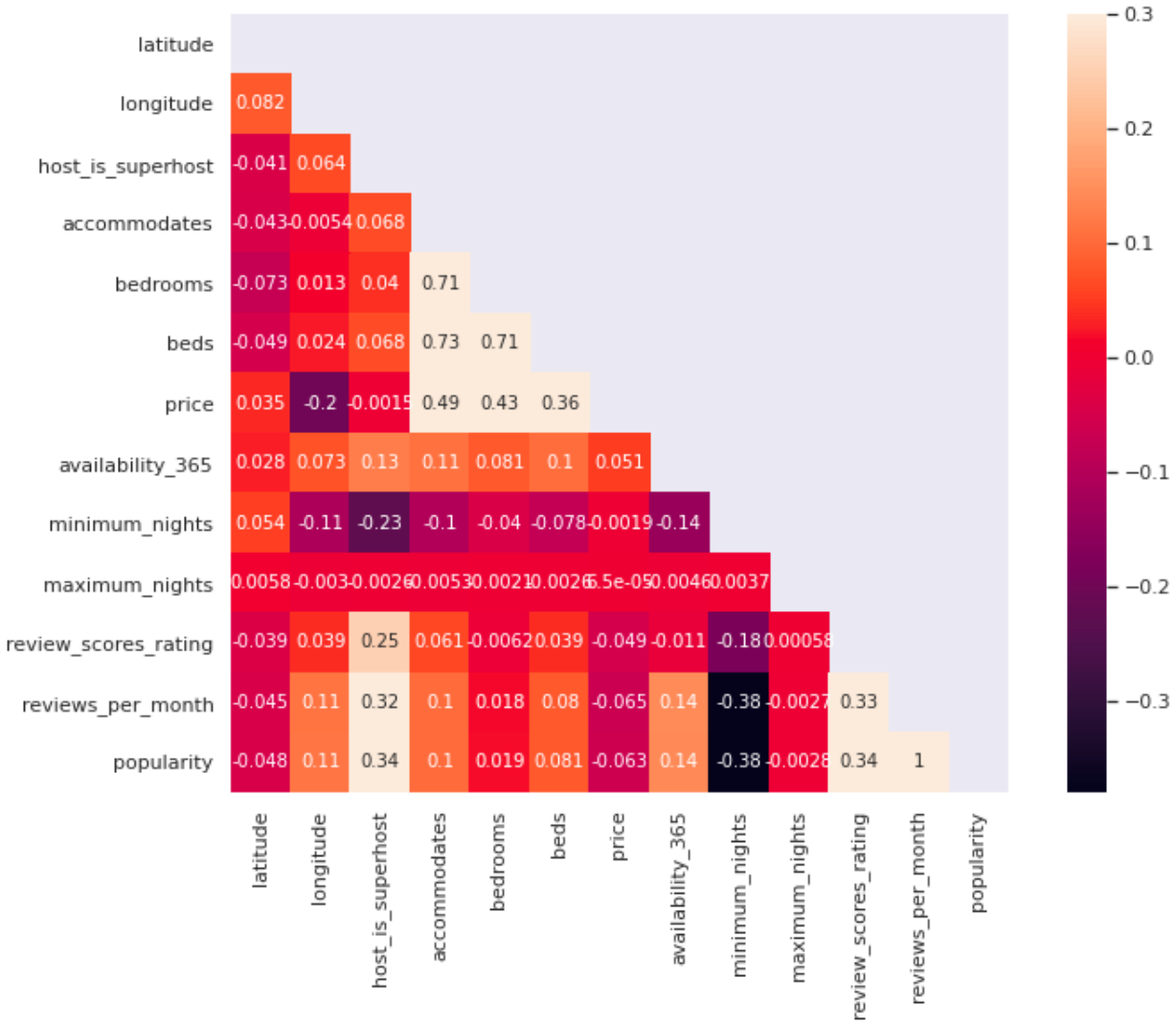
  We plotted a Heatmap to visualize the correlation between the columns



The heatmap effectively shows in a visual way that the attribute most correlated with price is “accomodates” and “bedrooms” in a positive correlation and “bedroom share” in a negative correlation. We can see that the feature 'accomodates' is strongly related with the features 'beds' and 'bedrooms'. This is totally intuitive. The feature 'popularity' is positively and strongly correlated with 'host_is_superhost', 'longitude', 'availability', and 'review_scores_rating', and 'reviews_per_month'. This indicates the normal human nature, where renters do some research about the listings before finalizing it. An apartment that has "super host" status and receives a higher number of reviews is more popular. The popularity, however, has a strong negative correlation with the minimum_nights feature. This shows that the apartments that enforce a higher number of minimum nights are less popular among the renters. This makes sense because only those renters, who meet this criteria will go for such apartments. This decreases the range of renters who find such apartments attractive and hence the popularity is low.

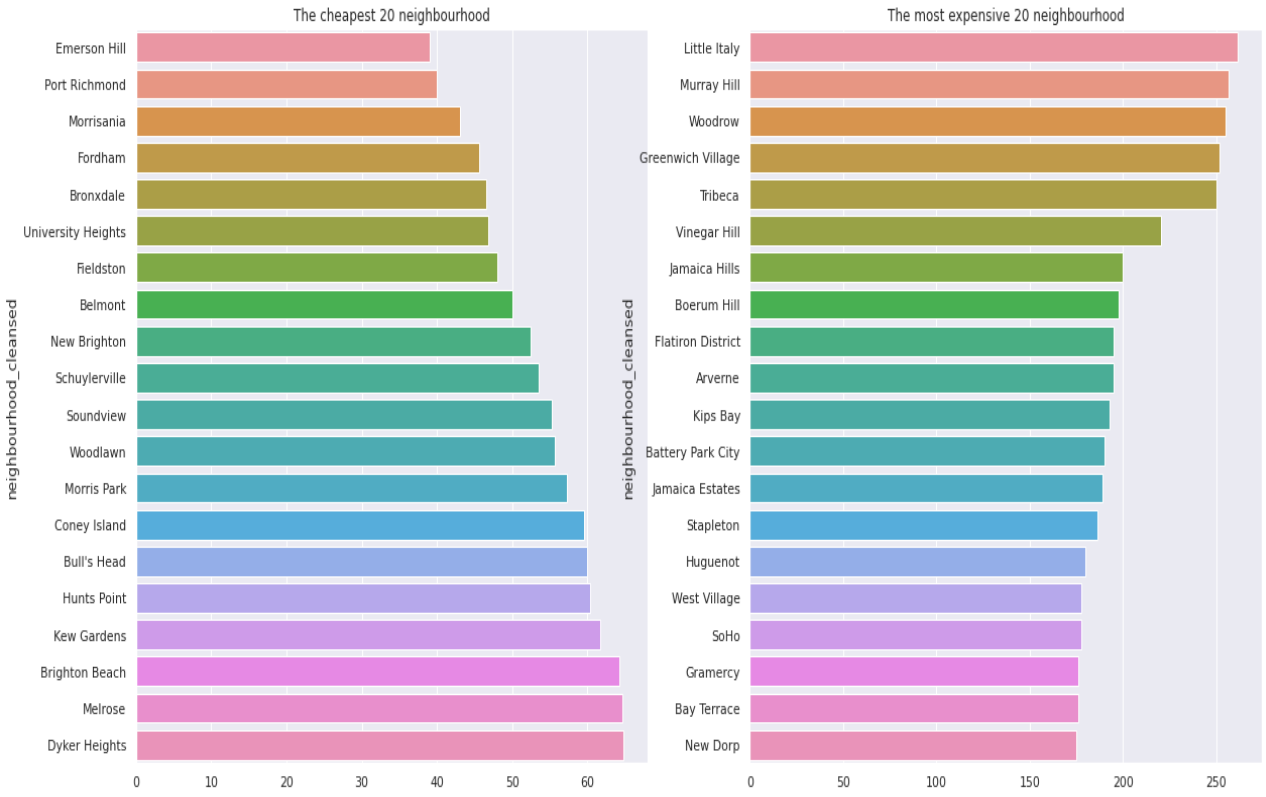
- Correlation map between numerical variables

  This we did to uncover relationship between the numerical variables



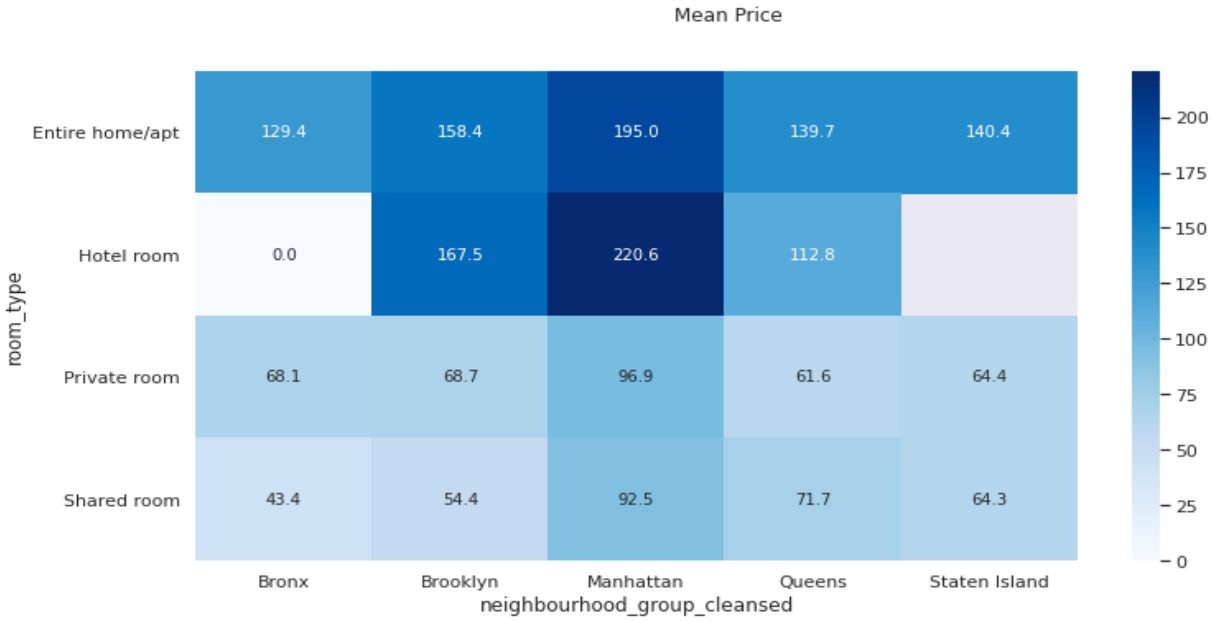
- Most expensive and the cheapest neighbourhoods

  This distribution showed which neighbourhoods are least expensive and which are the most expensive. The comparison of prices and respective neighbourhoods is shown below.

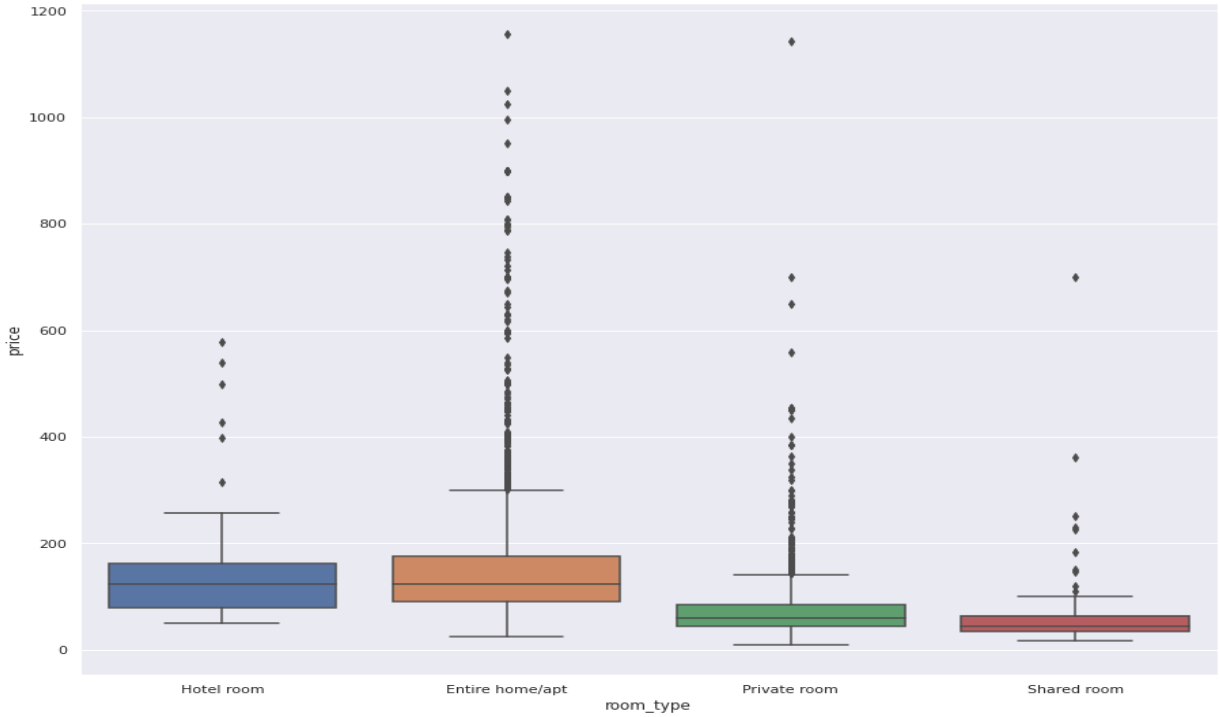


- Variance of mean price with respect to the boroughs and room type

  Clearly, Manhattan is the priciest borough to rent an AirBnb. Among all the room types, the Hotels are the costliest.

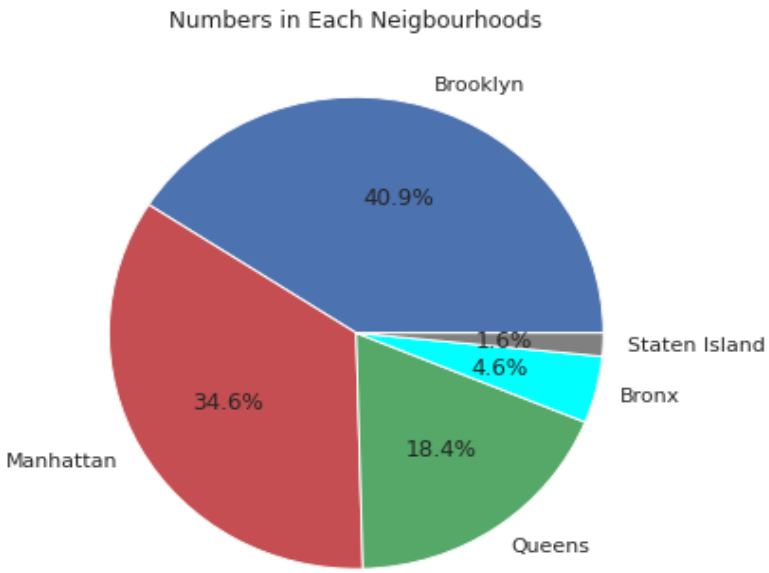


- Outliers in Price per Room Type



We can see that the Entirehome/apt and Private Rooms have many outliers. These could either be luxury apartments or may just be overpriced for no reason!

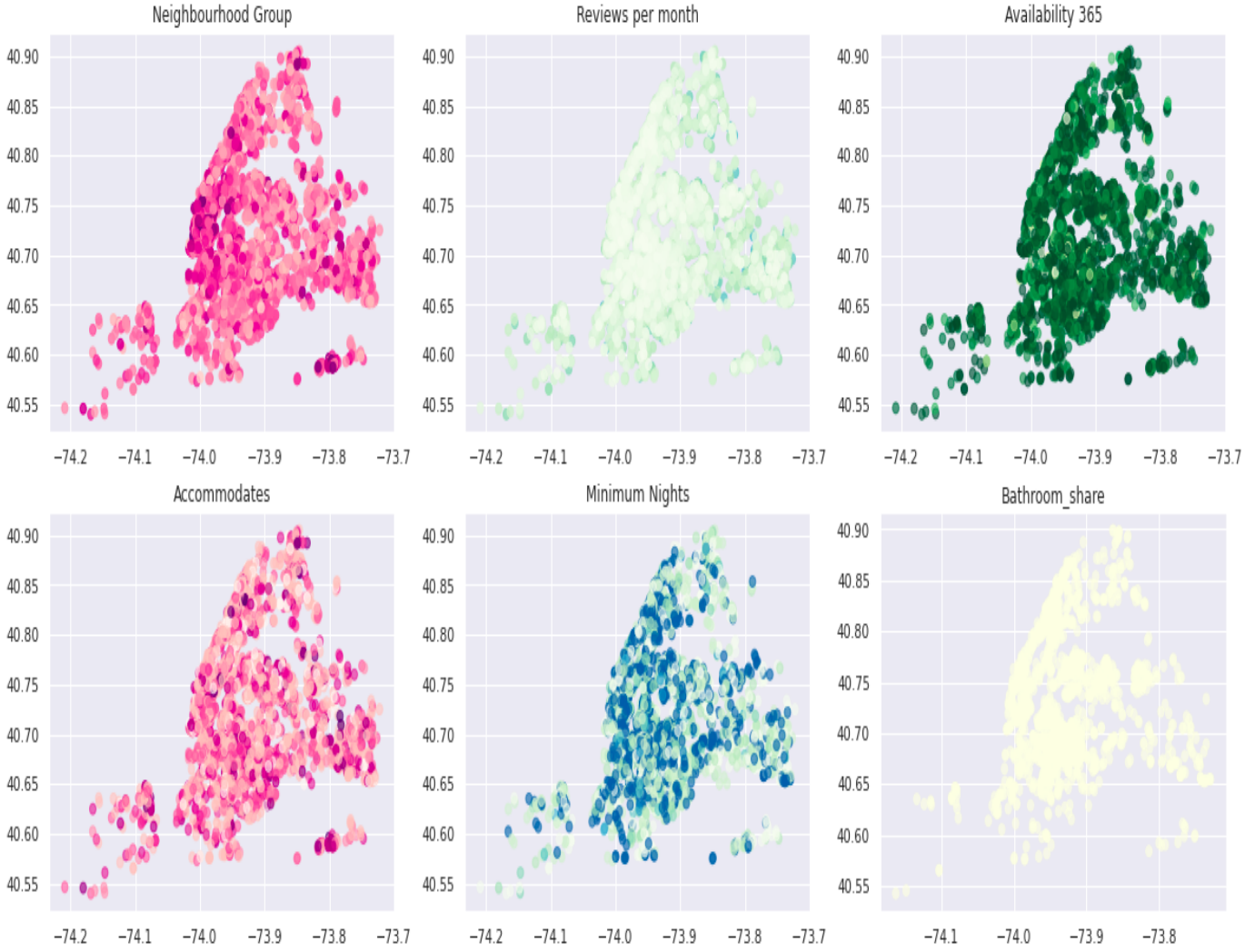
- Number of listing in each neighborhood



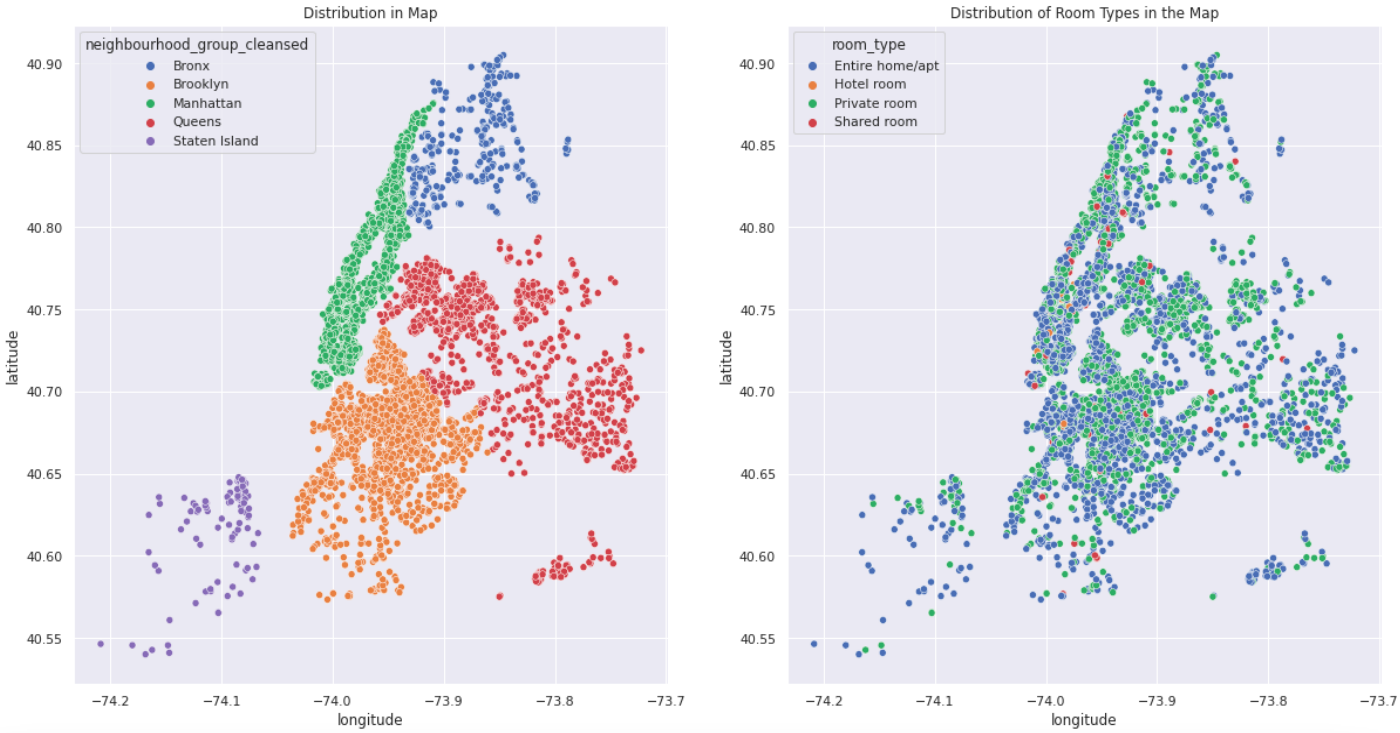
The average price per borough shows Manhattan to be the most expensive, as expected. On the contrary, AirBnbs in the Bronx are the least expensive of all boroughs on an average.

```
neighbourhood_group_cleansed
Bronx             84.516556
Brooklyn         112.084758
Manhattan        135.456302
Queens            91.921617
Staten Island    102.990385
Name: price, dtype: float64
```

- Variance of various features based on the location in NYC



- Distribution of room types across boroughs



- Cluster map of the listings

The map shows a concentration of listings in all the neighbourhoods Manhattan, Brooklyn, Queens, Bronx and Staten Island.

Modeling

For our project, we used nine different algorithms to create a pricing prediction model. The prediction models used were:

1. Default KNN (K-nearest Neighbors Regression)
2. Best KNN
3. Default SVR (Support Vector Regression)
4. Best SVR
5. Decision Tree
6. Random Forest
7. Linear Regression
8. Ridge Regression
9. Elastic Net Regression

Our purpose is to estimate the price based on the given dataset. Since price is a continuous variable we need to do a regression task. Therefore, we will use the Regression models of the traditional models such as KNN, SVM.

We are not after accuracy, because we are doing a regression task, not a classification task. Our main metric will be mean squared error and we will try to minimize it with some methods such as hyperparameter tuning.

Almost in every model we have sections such as:

Before Hyperparameter Tuning (we are trying model with default)

- Train model with the default parameters
- See the error and r2 score
- Compare with other models and results

After Hyperparameter Tuning

- Determine the parameter grid
- Do a gridsearch to get best model parameters.
- Compare with the default model and other models.

We have used **Grid search** for hyperparameter tuning. It's a tuning technique that attempts to compute the optimum values of hyperparameters. It is an exhaustive search that is performed on the specific parameter values of a model. The model is also known as an estimator. **Grid search** exercise can save us time, effort and resources.

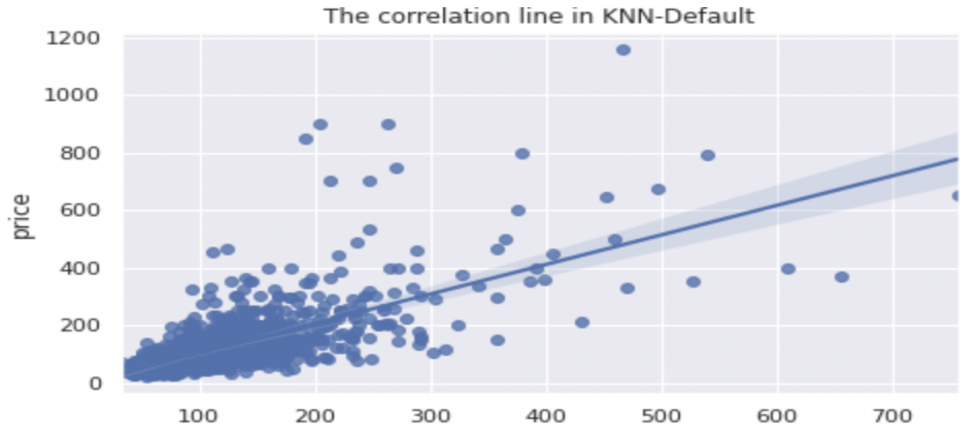
To evaluate the accuracy of the models, we will use the following *Evaluation Metrics:*

- **MSE** (mean squared error): $1N\sum i(ytruei-ypredi)2$
- **RMSE** (root mean squared error): $\sqrt{1N\sum i(ytruei-ypredi)2}$
- **MAE** (mean absolute error): $1N\sum i|ytruei-ypredi|$
- **R-squared**

Since we are measuring the errors, we want to **minimize** these metrics. For each prediction model, we trained the model by fitting our data to the model. Then, we selected some actual values and saw what the corresponding predicted values were for each model. Using the difference between the actual values and predicted values, we were able to evaluate the models on the following parameters: mean squared error, mean absolute error, root mean absolute error, as well as the $\mathbf{R^2}$ (coefficient of determination) regression score function.

1. Default KNN (K-nearest Neighbors Regression)

K-nearest neighbours can be used for *regression*. But there is a better model which is KNeighborsRegressor which is used for regression tasks. Therefore, we have used KNeighborsRegressor for our model.



	Mean Absolute Error	R2 Score
KNN - Default	39.195444	0.505803

2. KNN - Best

Using hyperparamaters such as number of nearest neighbours, or distance metric, we will apply hyperparamater tuning to see if we can decrease the error. Let's test it by:

- Number of Neighbours(1, ... ,10)
- LDistance Function (1: Euclidean, 2: Manhattan)

Cross Validations:

- And since we split the data by 80 training, 20 test data. We will do 100 / 20 = 5 cross validation splits.

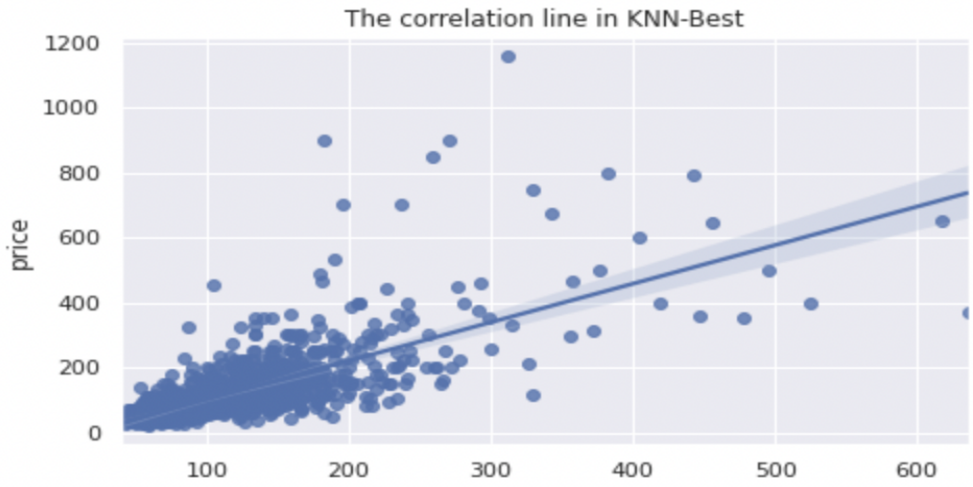
We have used GridSearchCV which searches the model with all possible combinations using cross validation. It calculates all the scores and finds the model which performed best.

```
from sklearn.model_selection import GridSearchCV
param_grid = {'p': [1, 2],
              'n_neighbors' : [ 5, 10, 15]
             }

grid_knn = GridSearchCV(KNeighborsRegressor(n_jobs=-1), param_grid, refit = True, verbose = 10, n_jobs=-1, cv=5,scoring="neg_mean_squared_error")

grid_knn.fit(X, y)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    0.8s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    1.2s
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:    1.8s remaining:    1.1s
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:    2.2s remaining:    0.7s
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:    2.4s remaining:    0.3s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:    2.4s finished
GridSearchCV(cv=5, estimator=KNeighborsRegressor(n_jobs=-1), n_jobs=-1,
             param_grid={'n_neighbors': [5, 10, 15], 'p': [1, 2]},
             scoring='neg_mean_squared_error', verbose=10)
```

```
print(f"Best parameters are {grid_knn.best_params_}")
print("Best score is {}".format(grid_knn.best_score_ * -1))
print("Best model is {}".format(grid_knn.best_estimator_))
#print("The score for hyperparameter tuning are {}".format(gr
```

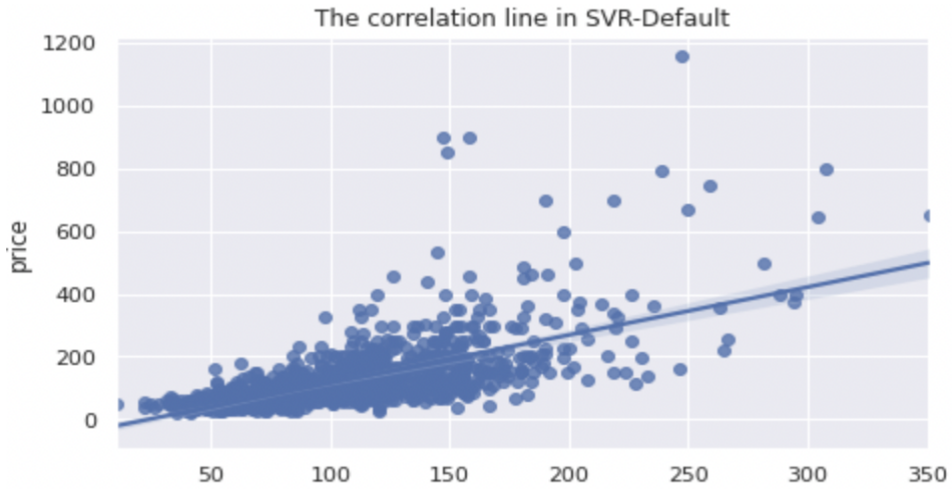
```
Best parameters are {'n_neighbors': 15, 'p': 1}
Best score is 3583.9288645614033
Best model is KNeighborsRegressor(n_jobs=-1, n_neighbors=15, p=1)
```



	Mean Absolute Error	R2 Score
KNN - Default	39.195444	0.505803
MSE KNN - Best	39.195444	0.517412

3. Default SVR (Support Vector Regression)

LinearSVR is a support vector machine for regression problems. Therefore we have used first default parameters and we will apply hyperparameter tuning. In most regression cases LinearSVR performs faster and more accurate results.



	Mean Absolute Error	R2 Score
KNN - Default	39.195444	0.505803
MSE KNN - Best	39.195444	0.517412
SVR - Default	40.976521	0.374812

4. Best SVR

Applying hyperparameter tuning and cross-validation points:

The parameters for the Linear SVR:

- **loss:** Specifies the loss function (Either L1 Loss or l2 loss)
- **C:** Regularization parameter. ('C': [0.1, 1, 10, 100, 1000])
- **dual:** dual or primal optimization problem. (Either True or False)

What we have done:

- Use grid search to fit all those parameters
- In each iteration use 5 cross validation points

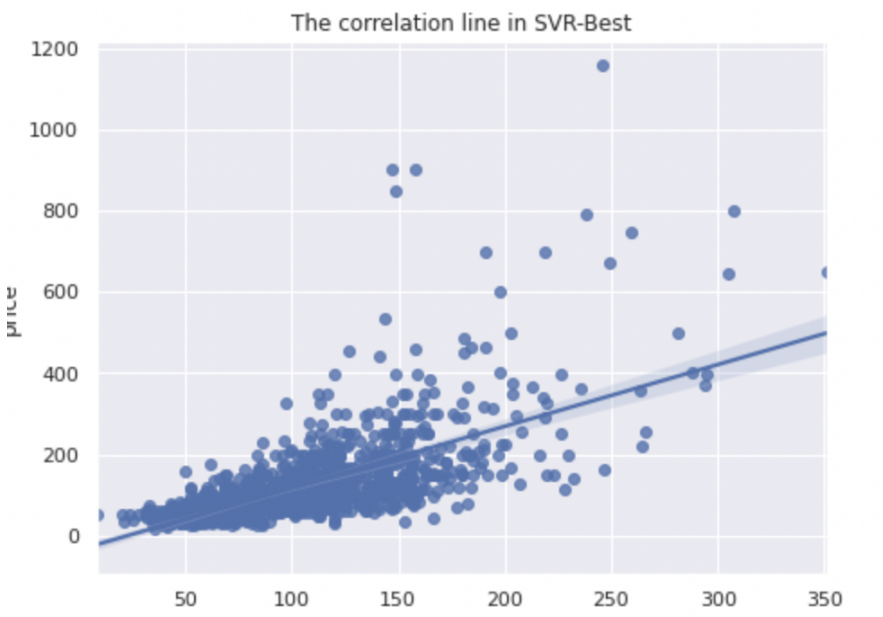
```
            'loss': ['epsilon_insensitive', 'squared_epsilon_insensitive'],
            'dual': [True, False],
            'tol': [0.0001, 0.00001]}

grid = GridSearchCV(LinearSVR(), param_grid, refit = True, verbose = 10, n_jobs=-1, cv=5,scoring="neg_mean_squared_error")

grid.fit(X, y)
```

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Batch computation too fast (0.0261s.) Setting batch_size=2.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done  19 tasks      | elapsed:    0.6s
[Parallel(n_jobs=-1)]: Done  35 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Batch computation too fast (0.1938s.) Setting batch_size=4.
[Parallel(n_jobs=-1)]: Done  52 tasks      | elapsed:    0.8s
[Parallel(n_jobs=-1)]: Done  78 tasks      | elapsed:    1.6s
[Parallel(n_jobs=-1)]: Done 112 tasks      | elapsed:    3.6s
[Parallel(n_jobs=-1)]: Batch computation too slow (2.0885s.) Setting batch_size=1.
[Parallel(n_jobs=-1)]: Done 161 tasks      | elapsed:    5.5s
[Parallel(n_jobs=-1)]: Done 200 out of 200 | elapsed:    6.4s finished
```

```
GridSearchCV(cv=5, estimator=LinearSVR(), n_jobs=-1,
             param_grid={'C': [0.1, 1, 10, 100, 1000], 'dual': [True, False],
                         'loss': ['epsilon_insensitive',
                                  'squared_epsilon_insensitive'],
                         'tol': [0.0001, 1e-05]},
             scoring='neg_mean_squared_error', verbose=10)
```

```
print(f"Best parameters are {grid.best_params_}")
print("Best score is {}".format(grid.best_score_ * -1))
print("Best model is {}".format(grid.best_estimator_))
print("scores {}".format(grid.cv_results_['mean_test_score']))
```

```
Best parameters are {'C': 1, 'dual': True, 'loss': 'squared_epsilon_insensitive', 'tol': 0.0001}
Best score is 4128.128695651688
Best model is LinearSVR(C=1, loss='squared_epsilon_insensitive')
scores [-5381.04877098 -5381.15771699 -4153.41080878 -4153.72379159
            nan            nan -4153.7027139  -4153.69249262
 -4547.40189678 -4556.20172801 -4128.12869565 -4129.22547368
            nan            nan -4140.83177408 -4140.93365203
 -4457.30997813 -4455.12044904 -4131.0141759  -4196.8597271
            nan            nan -4138.58827881 -4138.65957139
 -4432.66580845 -4456.46343227 -6168.66565591 -6053.2818124
            nan            nan -4138.39508705 -4138.46870034
 -4593.51027341 -4483.79704621 -7735.45336169 -6749.77819658
```



	Mean Absolute Error	R2 Score
KNN - Default	39.195444	0.505803
MSE KNN - Best	39.195444	0.517412
SVR - Default	40.976521	0.374812
SVR - Best	40.964648	0.375094

5. Decision Tree

Decision Tree can be used for classification and regression. Here, we have used Decision Tree for regression. We have used Decision Tree with hyperparameter tuning.

Parameters that is used for Decision Tree Regressor:

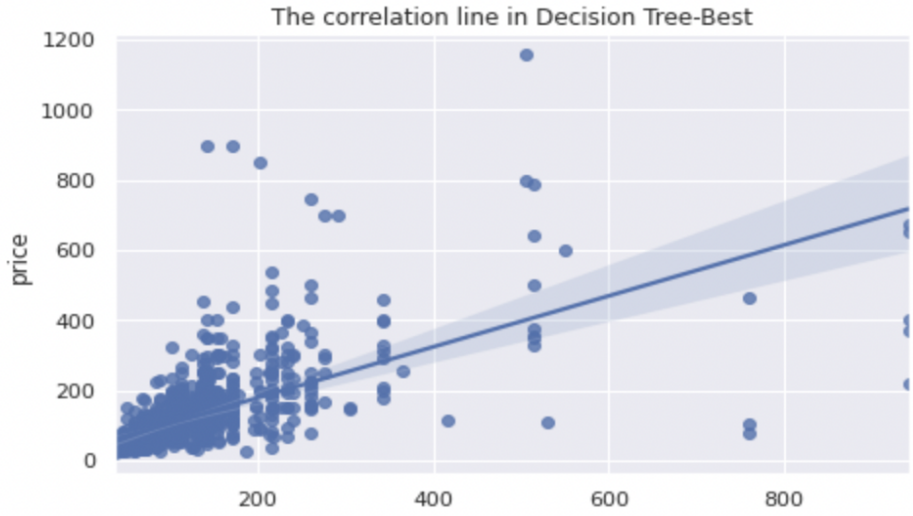
1. max_depth: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
2. min_samples_leaf:The minimum number of samples required to be at a leaf node.
3. min_samples_split:The minimum number of samples required to split an internal node.

What we have done:

1. Use grid search to fit all those parameters
2. In each iteration use 5 cross validation points

```
parameters = {
    'max_depth': [1, 2, 3, 4, 5, 6, 7, 8],
    'min_samples_leaf': [1, 2, 3, 4, 5],
    'min_samples_split': [2, 3, 4, 5],
}
tree_grid = GridSearchCV(tree_model, parameters, refit = True, verbose = 1, n_jobs=-1, cv=5, scoring="neg_mean_squared_error")
tree_grid.fit(X, y)
```

```
Fitting 5 folds for each of 160 candidates, totalling 800 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 800 out of 800 | elapsed:    3.5s finished
GridSearchCV(cv=5, estimator=DecisionTreeRegressor(), n_jobs=-1,
             param_grid={'max_depth': [1, 2, 3, 4, 5, 6, 7, 8],
                         'min_samples_leaf': [1, 2, 3, 4, 5],
                         'min_samples_split': [2, 3, 4, 5]},
             scoring='neg_mean_squared_error', verbose=1)
```



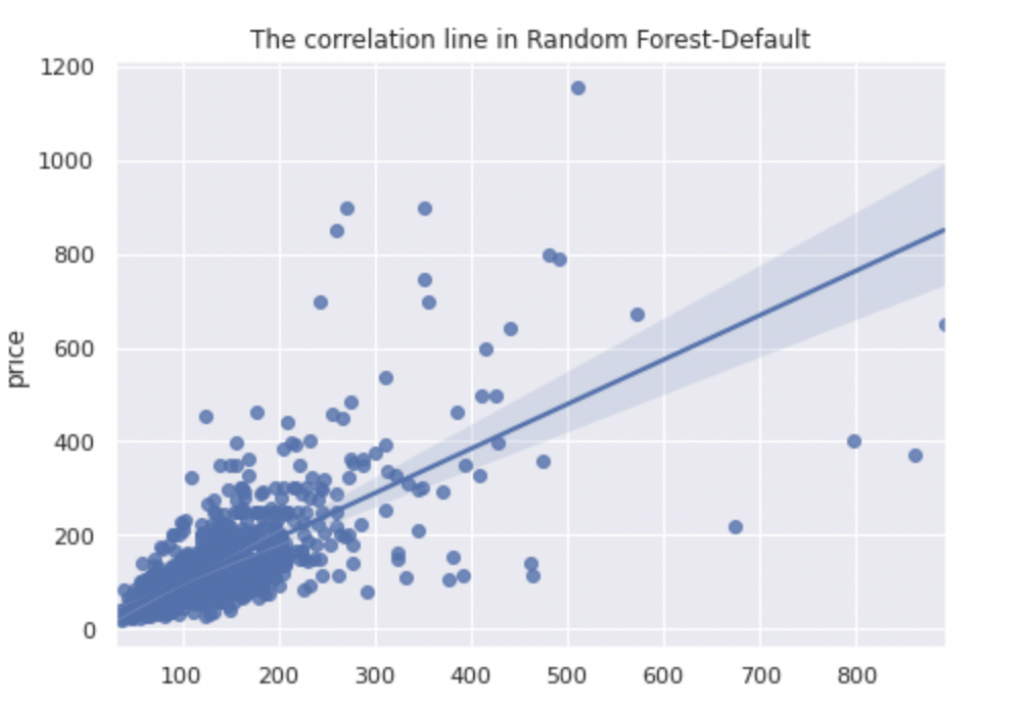
	Mean Absolute Error	R2 Score
KNN - Default	39.195444	0.505803
MSE KNN - Best	39.195444	0.517412
SVR - Default	40.976521	0.374812
SVR - Best	40.964648	0.375094
Decision Tree - Best	40.979619	0.343876

6. Random Forest

Random Forest is supervised machine learning algorithms which can be used for regression.

Hyperparameter tuning takes too much time, approximately around half an hour, to finish for a random forest model. Therefore, we didn't apply Hyperparameter tuning.

Also default random forest gives the best solution among the models that we tried so far.

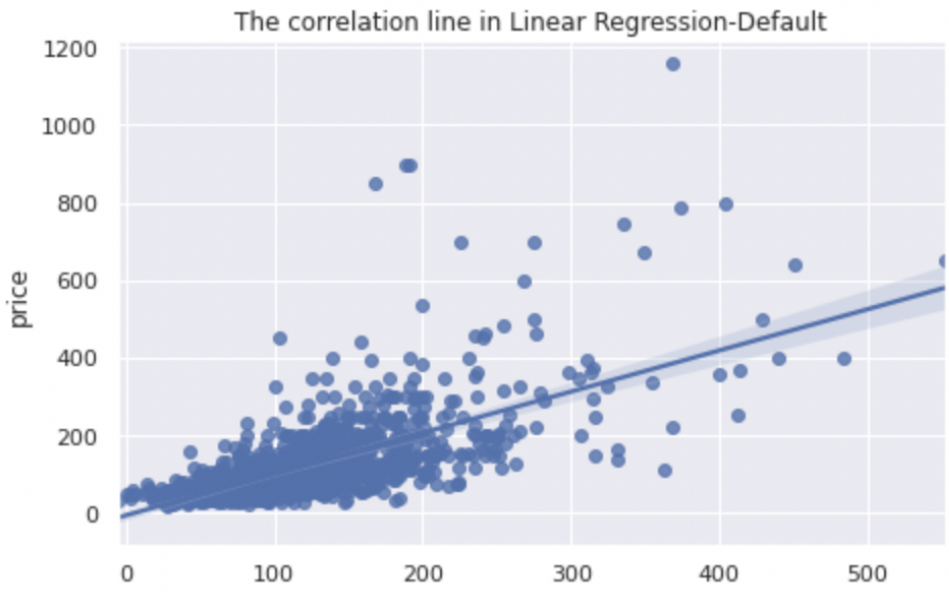


	Mean Absolute Error	R2 Score
KNN - Default	39.195444	0.505803
MSE KNN - Best	39.195444	0.517412
SVR - Default	40.976521	0.374812
SVR - Best	40.964648	0.375094
Decision Tree - Best	40.979619	0.343876
Random Forest - Default	36.955923	0.533239

The points are more spreaded in the Random Forest. Since the Random Forest runs lots of Decision Tree and gets the best tree as the classifier, it is very natural to have a better performance than the Decision Tree. The correlation line performed better than the rest. Since the points are more spreaded and in correct places the mean absolute error is less than the rest. Random Forest performed best for now. But let's see other models if we can improve.

7. Linear Regression

Performed basic Linear Regression with default values.



	Mean Absolute Error	R2 Score
KNN - Default	39.195444	0.505803
MSE KNN - Best	39.195444	0.517412
SVR - Default	40.976521	0.374812
SVR - Best	40.964648	0.375094
Decision Tree - Best	40.979619	0.343876
Random Forest - Default	36.955923	0.533239
Linear Regression - Default	42.825844	0.461389

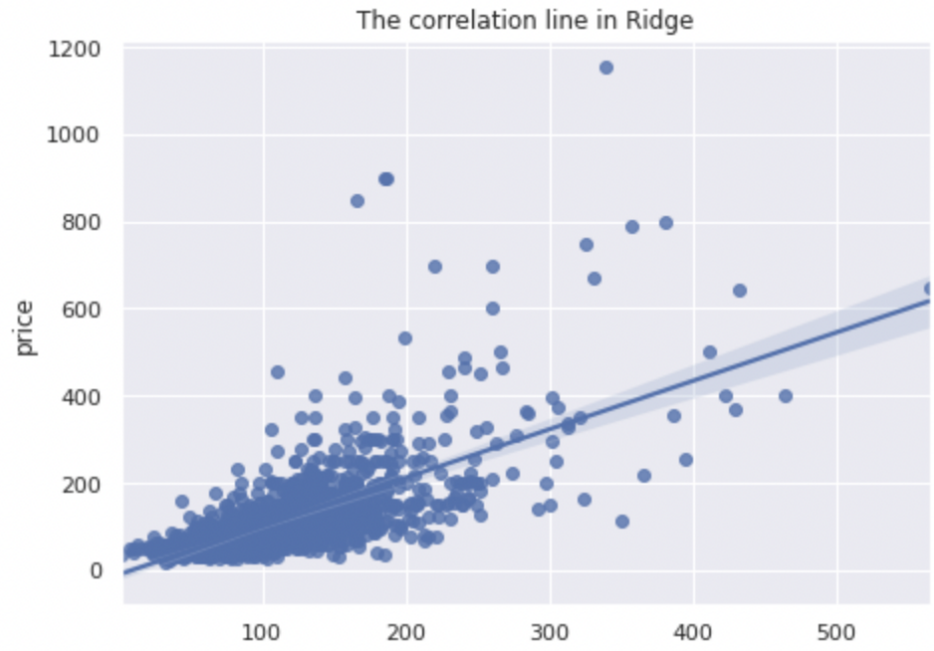
8. Ridge Regression

We use different techniques for validation of our linear regression model called Ridge.

Since we cannot apply hyperparameter tuning into Linear Regression directly, we will use Ridge in order to apply Hyperparameter tuning.

```
grid_mse = GridSearchCV(estimator=ridge, param_grid=param_grid, scoring='neg_mean_squared_error', verbose=1, n_jobs=-1)
grid_result_mse = grid_mse.fit(X_train, y_train)

grid_r2 = GridSearchCV(estimator=ridge, param_grid=param_grid, scoring='r2', verbose=1, n_jobs=-1)
grid_result_r2 = grid_r2.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  52 tasks      | elapsed:    1.0s
[Parallel(n_jobs=-1)]: Done  90 out of  90 | elapsed:    1.1s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done  90 out of  90 | elapsed:    0.2s finished
```

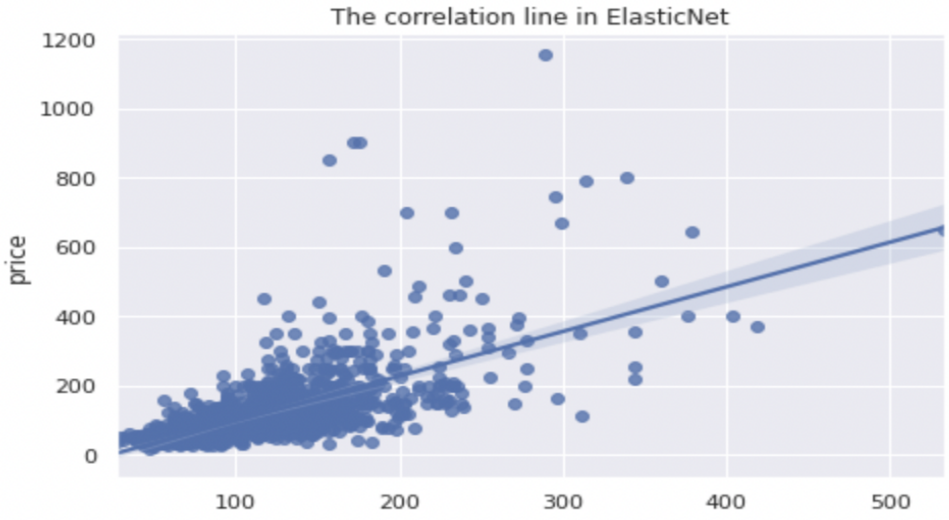


	Mean Absolute Error	R2 Score
KNN - Default	39.195444	0.505803
MSE KNN - Best	39.195444	0.517412
SVR - Default	40.976521	0.374812
SVR - Best	40.964648	0.375094
Decision Tree - Best	40.979619	0.343876
Random Forest - Default	36.955923	0.533239
Linear Regression - Default	42.825844	0.461389
Ridge	42.504782	0.457379

After using Grid Search CV, we found that alpha=500 is the best parameter for Ridge. We can see the effect of Hyperparameter Tuning compared to default Ridge. However, we do not have a significant difference between Ridge and Linear Regression model.

9. Elastic Net Regression

Elastic net is a popular type of regularized linear regression that combines two popular penalties, specifically the L1 and L2 penalty functions.



	Mean Absolute Error	R2 Score
KNN - Default	39.195444	0.505803
MSE KNN - Best	39.195444	0.517412
SVR - Default	40.976521	0.374812
SVR - Best	40.964648	0.375094
Decision Tree - Best	40.979619	0.343876
Random Forest - Default	36.955923	0.533239
Linear Regression - Default	42.825844	0.461389
Ridge	42.504782	0.457379
ElasticNet	42.537543	0.431806

Finally, we can compare the predicted value against the actual value for each model.

Random Forest model gives us the best score in R2 as well as MAE. However, as we described in Random Forest section, running time of the Random Forest model is slightly more than other models. It can be seen that **KNN performed better mostly** (we can see it also from MAE scores). **Hyperparameter tuning for SVR does perform better than the default model.** But there is not much difference. The reason might be that there is not much difference between models in terms of parameters and the difference is not enough to get a huge difference.

Actual Values	KNN-Default	KNN-Best	SVR-Default	SVR-Best	Decision Tree-Best	Random Forest-Default	Linear Regression-Default	Ridge	ElasticNet
671.0	497.0	341.933333	249.705009	249.424391	942.000000	571.88	348.727705	331.411299	249.705009
75.0	235.8	176.000000	148.259815	148.078439	141.001845	167.34	224.702543	216.182662	148.259815
171.0	113.2	122.466667	105.669845	105.636067	113.211806	145.89	117.556201	119.309728	105.669845
138.0	163.8	144.133333	129.537308	129.612125	141.001845	148.71	160.815965	157.928822	129.537308
165.0	137.6	129.733333	112.677784	112.921113	131.333333	139.23	126.511513	124.642636	112.677784
116.0	111.4	105.133333	96.508572	96.327815	88.287554	108.05	102.563604	105.746648	96.508572
95.0	176.0	127.066667	113.555388	113.528912	141.001845	130.52	131.474172	128.189416	113.555388
140.0	191.8	153.266667	173.807741	174.198495	125.222222	153.32	193.461042	195.159963	173.807741
95.0	175.2	163.800000	130.051845	130.207804	141.001845	129.66	156.746870	145.041558	130.051845
75.0	173.2	73.933333	92.135386	92.208860	97.835616	91.56	113.635426	108.301909	92.135386
125.0	178.2	152.600000	96.263117	96.362015	141.001845	151.79	100.369729	103.346774	96.263117
47.0	61.2	53.733333	44.100024	44.212910	59.789744	57.18	39.373511	44.209313	44.100024
255.0	260.2	203.333333	207.708318	207.828090	137.587189	202.51	257.707438	247.658612	207.708318
110.0	189.6	169.733333	104.404177	104.693983	141.001845	151.30	115.144674	115.584487	104.404177
35.0	75.0	56.200000	21.923745	21.644375	50.959596	81.05	-5.489623	1.568742	21.923745
200.0	144.4	127.933333	101.685384	102.101657	141.001845	153.84	108.709753	110.885200	101.685384
329.0	110.0	133.533333	146.867268	147.062994	170.743590	169.61	168.328002	163.974677	146.867268
80.0	76.6	92.733333	102.424679	102.144490	113.211806	93.18	111.985917	110.814197	102.424679
69.0	88.0	118.600000	91.887163	92.303845	103.750000	115.63	109.443071	105.707471	91.887163
60.0	48.0	53.400000	57.062551	56.615105	53.885714	57.12	71.820166	71.359561	57.062551

Evaluation

To evaluate the different models used, we will look at both the Mean Absolute Error and the R2 Score. We will interpret a lower Mean AbsoluteError to be good and a higher R2 score closest to 1.0 to be good.

With these two criteria, we can determine the Random Forest model to be the best model as it has the lowest Mean Absolute Error of 36.95 and highest R2 score of 0.53 from all of the models. For this model the accuracy is 69.18%.

Therefore, we will use the Random Forest model to be the algorithm that a new host should use in order to predict the price of a unit per night to maximize revenue. Any price lower than that may lead to missed revenue and anything higher than that will lead to an overpriced unit that will not be rented as often and will therefore lose out on potential revenue.

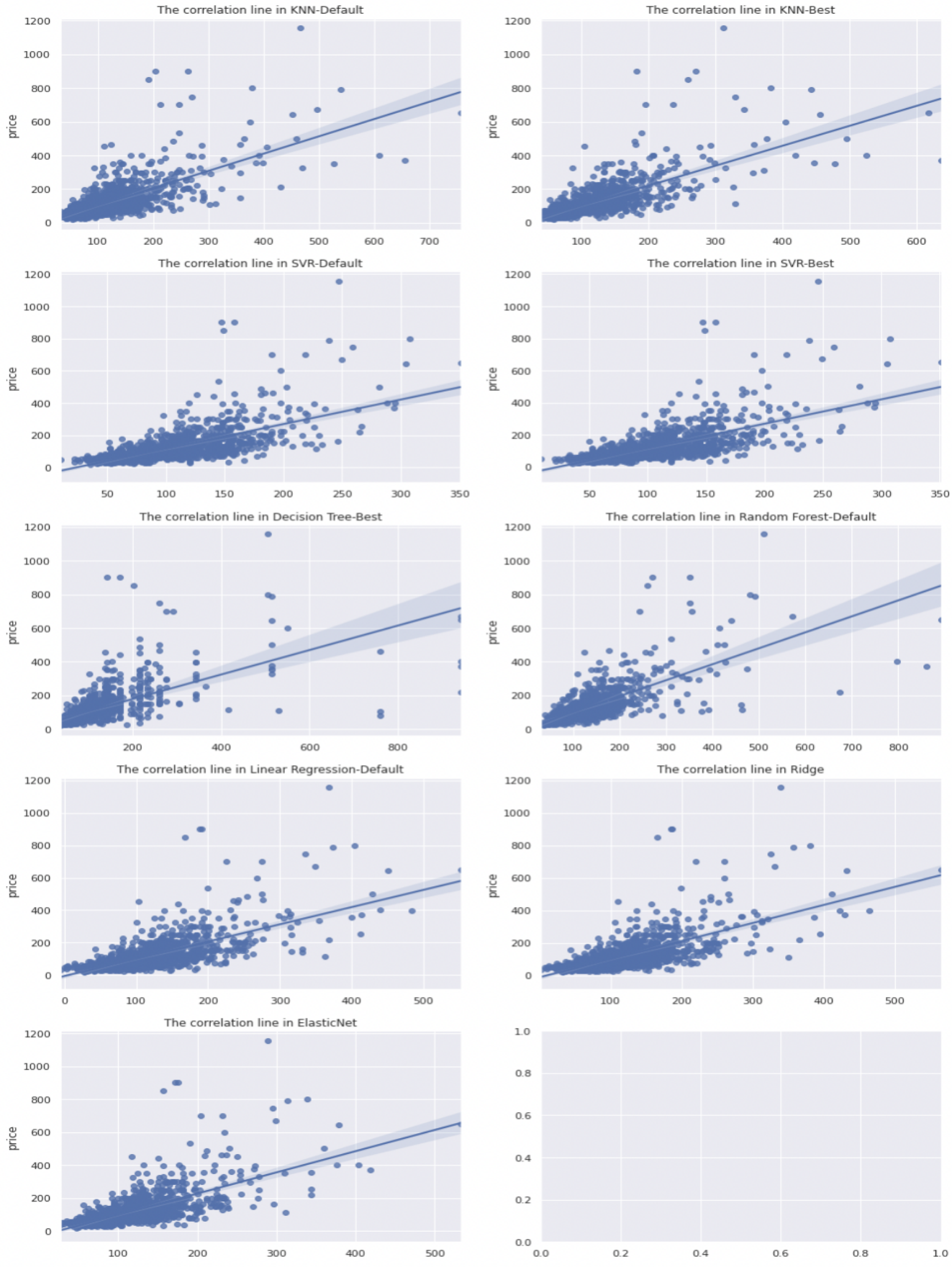
The Random Forest model gives us the best score in R2 as well as MSE. However, as we described in the Random Forest section, the running time of the Random Forest model is slightly more than other models followed by Hyperparameter tuned KNN. MSE KNN Best and Random Forest and two of them performed best among the other models.

If we did not have time restriction, KNN or Neural Network can be used. Both give us noticeable performance and accuracy. If we had more data other models like Neural Networks could have performed better than the other models.

Deployment

- The result of the data mining will be deployed by giving new hosts a recommended price for their units using the algorithm we created. This will allow new hosts to optimally set the price of the unit, which will maximize revenue for the host and satisfy guests. This will also provide the most revenue for the platform, AirBnB as well.
- A simple website could be made where the new host inputs the values for the various attributes of the new unit and a new recommended price is given to the user.
- A potential issue with this algorithm would be that if everyone uses this tool, the market could become more competitive. All the prices would be set at a similar prices compared to other comparable units, so the other attributes will become more important to the user.
- Furthermore, we did not look at the facilities in the vicinity of the unit or the unit's proximity to a point-of-interest such as a tourist destination or a popular commercial location. These attributes could be added with further research.
- The business problem we focused on for this project does not seem to have ethical considerations as it is a commercial topic.

The correlation line between actual and predicted values :


The correlation line in KNN-Default
The correlation line in KNN-Best
The correlation line in SVR-Default
The correlation line in SVR-Best
The correlation line in Decision Tree-Best
The correlation line in Random Forest-Default
The correlation line in Linear Regression-Default
The correlation line in Ridge
The correlation line in ElasticNet
price

References

1. *Predicting Airbnb Listing Price Across Different Cities*, Yuanhang Luo, Xuanyu Zhou, Yulian Zhou, December 14, 2019
2. *Reasonable Price Recommendation on AirBnb using Multi-Scale Clustering*, Yang Li, Quan Pan, Tao Yang, and Lantian Guo, CCC, pages 7038–7041. IEEE, 2016.
3. *Price Determinants of Sharing Economy based Accommodation Rental: A Study of Listings from 33 cities on AirBnb*, Dan Wang and Juan L Nicolau, International Journal of Hospitality Management, pg. 120–131, 2017.
4. *Price determinants on Airbnb: How Reputation Pays off in the Sharing Economy*, Timm Teubner, Florian Hawlitschek, and David Dann. Journal of Self-Governance & Management Economics, 5(4), 2017.
5. *Airbnb Price Prediction using Machine Learning and Sentiment Analysis*, Pouya Rezazadeh Kalehbasti, Liubov Nikolenko, and Hoormazd Rezaei, 2019.
6. Airbnb Open NYC Data
7. Inside AirBnb Website