Capstone Project

Exploration and Modelling Documentation

Shujie Deng 301171263

Centennial College

BA 723001 Business Analytics Capstone

Bilal Hasanzadah

David Parent

August 12, 2022

Table of Contents

Executive Summary

Users of the Airbnb platform have the opportunity to compete with traditional accommodations by renting out extra bedrooms or entire homes to tourists. However, as Airbnb allows the user to set the price of their listing, the challenge is on the host to find the most appropriate value. This project explores data from New York City listings to understand better how listing features might be applied as predictors for the price of a new listing. Information collected from the Inside Airbnb page is analyzed statistically and with machine learning techniques for the study. The project begins with data cleansing, removal of null values, replacement of null values, and number extraction from a string. Then I began to investigate the dataset and divided it into 60% train data and 40% valid data. Finally, this project aimed to determine if a decision tree, a random forest, a Gradient boosting model, or an XGBoost model was the best effective tool for forecasting the price of an Airbnb rental. Those models are compared based on the RMSE, and R squared. The results demonstrate that the XGBoost is the more stable model with the lowest RMSE.

Introduction

1.0. Executive

Airbnb is a United States-based online community marketplace and hospitality agency. Local hosts can use Airbnb as a platform to accommodate guests for short-term stays and engage in tourism-related activities. Various search options, such as types of lodging, dates, price, and locations, will be available to guests. They can also narrow their search to certain types of homes, vacation homes, and bed and breakfasts. The hosts provide information such as the rental fee, location, number of guests, restrictions, home types, and other facilities. Airbnb's most recent data shows it has more than six million listings in more than 100,000 cities and towns and

more than 220 countries. New York City is the third-largest Airbnb market in the world and one of the oldest.

2.0. Problem Statement

Every time a reservation is made, Airbnb receives commissions from both hosts and guests. Airbnb charges the guest 6% - 12% of the booking fee for each reservation. Additionally, Airbnb charges the host 3% of each successful transaction. Due to Airbnb's revenue model, it is essential to grow the number of bookings to ensure the company's profitability. One way would be for hosts to offer lodgings and experiences at reasonable prices and to provide prospective guests with efficient recommendations of popular listings. Airbnb can increase its revenue by helping hosts quote fair pricing and understanding market trends and guests' expectations.

Several variables might affect the pricing of any listing, making it difficult for hosts to come up with a reasonable price. An understanding of guest willingness to book in light of various features of hosts and houses has been gathered from historical data collected from Airbnb listings. Hosts can gain a competitive advantage by focusing on aspects like location, room type, and history of pleasant experiences to set pricing at mutually desirable levels.

3.0. Objectives

This project's primary purpose is to develop and analyze many regression models that recommend the most reasonable pricing for a new Airbnb host's listing, considering the listing's features. The objective is to enable hosts to comprehend which attributes are most crucial for price assignment. To be more specific, here are some of the study topics hoping to answer: (1) What are the most common Airbnb room types available for rent in New York? (2) What is the most expensive location in New York? (3) What are the most important features of a New York

listing? (4) Can the selected features be used to create a data-driven model that can predict the price of a new listing?

4.0. Assumptions and Limitations

There may be factors influencing the listing price that were not included in the dataset used to train the model. The price may change if something unexpected happens, like a pandemic of Covid-19. The time horizon of the given dataset should be taken into account to increase prediction accuracy.

Data Sources

5.0. Data Set Introduction

The data retrieved from Inside Airbnb http://insideairbnb.com/get-the-data/, data including 71 features collect from June 05, 2021 to June 05, 2022. This project will focus on 16 variables following: id, host_is_superhost, host_identity_verified, neighbourhood_group_cleansed, latitude, longitude, room_type, bathrooms_text, bedrooms, beds, amenities, price, availability_365, number_of_reviews, review_scores_rating, instant_bookable.

6.0. Data Dictionary

Variables	Description
id	Airbnb's unique identifier for the listing. It will not be used in the prediction, only used for indexing and referencing purposes
host_is_superhost	Boolean [t= true, f = false]
host_identity_verified	Boolean [t= true, f = false]

neighbourhood_group_cleansed	All listing homes are grouped into the following five areas: Manhattan, Brooklyn, Queens, Staten Island, and the Bronx
latitude	location
longitude	location
room_type	All homes are grouped into the following four room types: Entire home/apt, Private room, Hotel room, and Shared room
bedrooms	The number of bedrooms
beds	The number of bed(s)
price	The price of a home
availability_365	The calendar determines the availability of the listing x days in the future.
number_of_reviews	The number of reviews the listing has
review_scores_rating	The rating of home, out of 5.00
instant_bookable	Boolean [t= true, f = false]. Whether the guest can automatically book the listing without the host requiring to accept their booking request.
bathroom_qty	The number of bathrooms
bathroom_type	All bathrooms are grouped into the following five types: bath, baths, shared, private, and half_bath

Data Exploration

7.0. Data Exploration Techniques

```
import numpy as np
import pandas as pd
import math
import pylab
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from scipy.stats import norm
from sklearn import linear_model
```

8.0. Data Cleansing

- Reading dataset

```
df = pd.read_excel("nycairbnb_2022.xlsx")
df.shape
```

```
(37410, 17)
```

```
df.head()
```

	id	host_is_superhost	host_identity_verified	neighbourhood_cleansed	neighbourhood_group_cleansed	latitude	longitude	room_type	bathrooms_text	bedrooms	beds
0	2595	f	t	Midtown	Manhattan	40.75356	-73.98559	Entire home/apt	1 bath	NaN	1.0
1	5121	f	t	Bedford-Stuyvesant	Brooklyn	40.68535	-73.95512	Private room	NaN	1.0	1.0
2	5136	f	t	Sunset Park	Brooklyn	40.66265	-73.99454	Entire home/apt	1.5 baths	2.0	2.0
3	5178	f	f	Midtown	Manhattan	40.76457	-73.98317	Private room	1 bath	1.0	1.0
4	5203	f	t	Upper West Side	Manhattan	40.80380	-73.96751	Private room	1 shared bath	1.0	1.0

- Columns in the dataset

```
df.columns
```

```
Index(['id', 'host_is_superhost', 'host_identity_verified',
       'neighbourhood_cleansed', 'neighbourhood_group_cleansed', 'latitude',
       'longitude', 'room_type', 'bathrooms_text', 'bedrooms', 'beds',
       'amenities', 'price', 'availability_365', 'number_of_reviews',
       'review_scores_rating', 'instant_bookable'],
      dtype='object')
```

- Splitting bathrooms_text into number and text

```
[43] df['bathroom_qty'] = df["bathrooms_text"].str.split(" ", expand=True)[0]
     df["bathroom_type"] = df["bathrooms_text"].str.split(" ", expand=True)[1]
```

- Feature type

```
variable_dic = {'nominal_var':['room_type', 'bathroom_type', 'neighbourhood_group_cleansed'],
                'binary_var':['host_is_superhost', 'host_identity_verified', 'instant_bookable'],
                'numerical_var':['id', 'latitude', 'longitude', 'bedrooms', 'beds', 'availability_365', 'number_of_reviews', 'review_scores_rating', 'bathroom_qty']}
```

8.1. Describing numerical variables

```
df.describe()
```

	id	latitude	longitude	bedrooms	beds	price	availability_365	number_of_reviews	review_scores_rating	bathroom_qty
count	3.741000e+04	37410.000000	37410.000000	33756.000000	36509.000000	37410.000000	37410.000000	37410.000000	29461.000000	37265.000000
mean	6.100385e+16	40.729308	-73.946171	1.354219	1.634720	190.775221	119.704892	26.347875	4.615201	1.158621
std	1.806040e+17	0.058041	0.056701	0.734038	1.148007	342.491748	132.758373	55.060905	0.759865	0.455790
min	2.595000e+03	40.504560	-74.269520	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.470053e+07	40.688260	-73.983518	1.000000	1.000000	75.000000	0.000000	1.000000	4.590000	1.000000
50%	3.484341e+07	40.724885	-73.953794	1.000000	1.000000	125.000000	60.000000	5.000000	4.830000	1.000000
75%	5.018400e+07	40.763380	-73.926340	1.000000	2.000000	203.000000	254.000000	25.000000	5.000000	1.000000
max	6.412414e+17	40.928340	-73.693210	15.000000	42.000000	12900.000000	365.000000	1419.000000	5.000000	15.500000

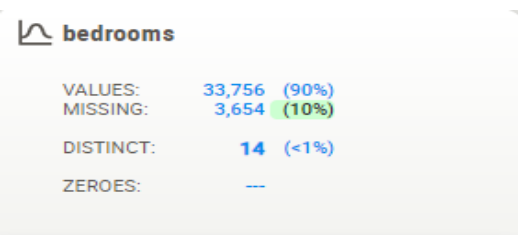
8.2. Missing value imputation

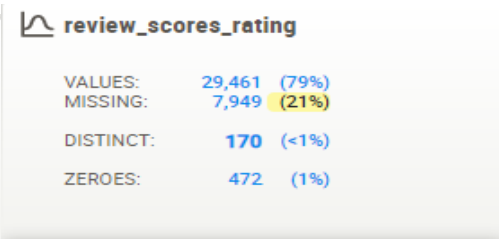
Some variables had missing values, but none had more than 50% of their missing data. Thus, they were replaced using an imputation method. In imputation, missing values are replaced with the mean, median, or mode of the variable's non-missing values. The attributes "bedrooms" and "reviews score rating" had 10% and 21% of missing values respectively; other attributes are less than 1% of missing values. The mean was used to fill out missing values for the variable "reviews score rating," the mode was used to replace the missing values of "bathroom quantity"

and "bathroom type", and the median of different types of room was used to replace the missing value of "bedrooms" and "bed".

```
[11] df.isnull().sum()
```

```
id                                0
host_is_superhost                66
host_identity_verified           66
neighbourhood_group_cleansed      0
latitude                          0
longitude                         0
room_type                         0
bedrooms                       3654
beds                            901
price                             0
availability_365                  0
number_of_reviews                 0
review_scores_rating           7949
instant_bookable                  0
bathroom_qty                    145
bathroom_type                   119
dtype: int64
```






```
#fill with mean
mean_df = df
mean_df['review_scores_rating']=mean_df['review_scores_rating'].fillna(mean_df['review_scores_rating'].mean())
mean_df.isnull().sum()
```

```
# relationship between room type, bedrooms and beds
df.groupby(by = 'room_type').mean().round()[['bedrooms','beds','bathroom_qty']]
```

room_type	bedrooms	beds	bathroom_qty
Entire home/apt	2.0	2.0	1.0
Hotel room	1.0	1.0	1.0
Private room	1.0	1.0	1.0
Shared room	1.0	2.0	1.0

```
bedroom_dic = {'Entire home/apt': 2, 'Hotel room': 1, 'Private room': 1, 'Shared room':1}
bed_dic = {'Entire home/apt': 2, 'Hotel room': 2, 'Private room': 1, 'Shared room':2}
```

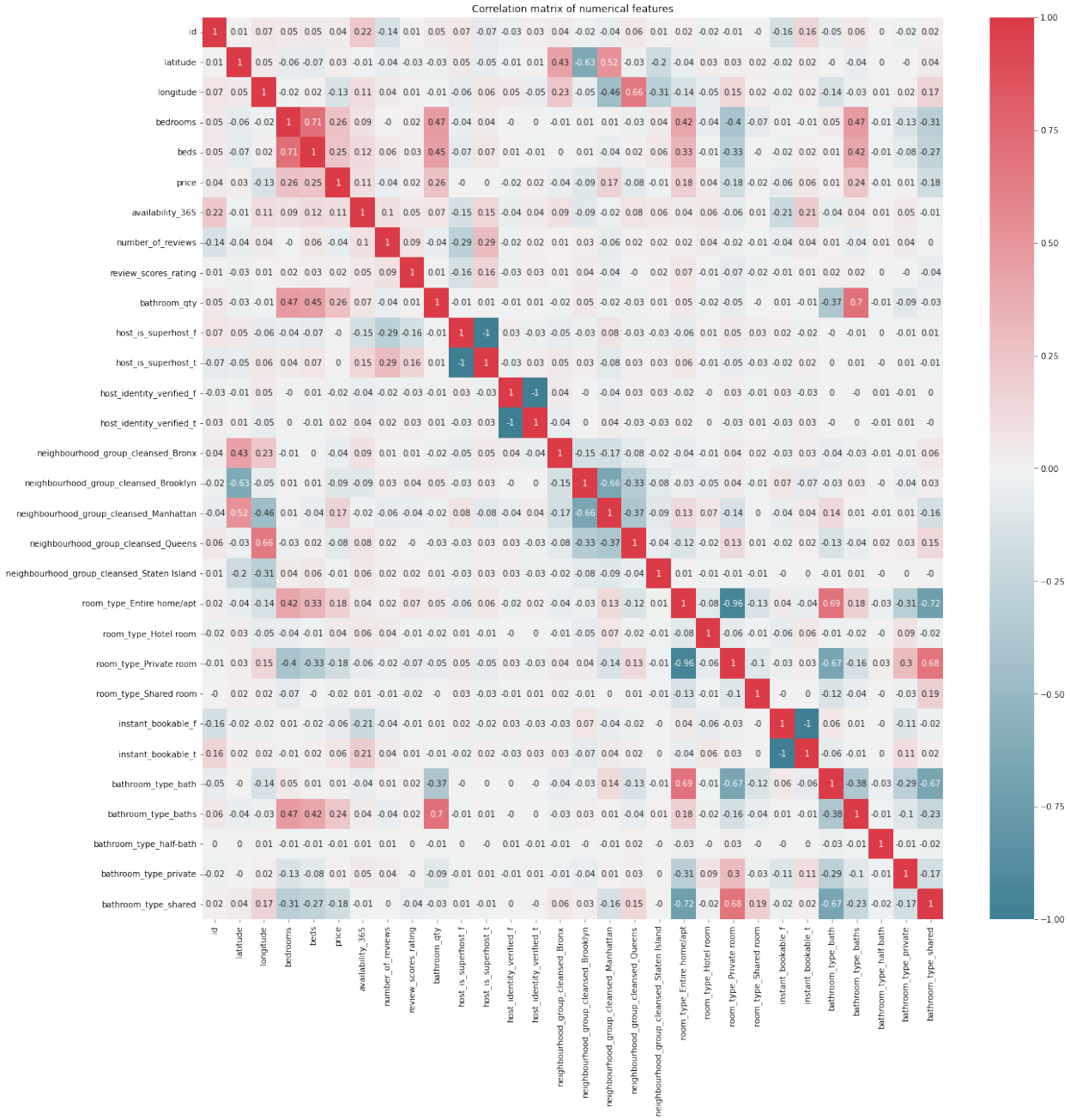
```
df.bedrooms[df.bedrooms.isnull()] = df[df.bedrooms.isnull()].room_type.apply(lambda x: bedroom_dic[x])
df.beds[df.beds.isnull()] = df[df.beds.isnull()].room_type.apply(lambda x: bed_dic[x])
```

```
#fill with mode
new_df = df
new_df['bathroom_qty'].fillna(new_df['bathroom_qty'].mode()[0], inplace=True)
new_df['bathroom_type'].fillna(new_df['bathroom_type'].mode()[0], inplace=True)
new_df.isnull().sum()
```

```
id                              0
host_is_superhost               0
host_identity_verified          0
neighbourhood_group_cleansed    0
latitude                        0
longitude                       0
room_type                       0
bedrooms                        0
beds                            0
price                           0
availability_365                0
number_of_reviews               0
review_scores_rating            0
instant_bookable                0
bathroom_qty                    0
bathroom_type                   0
dtype: int64
```

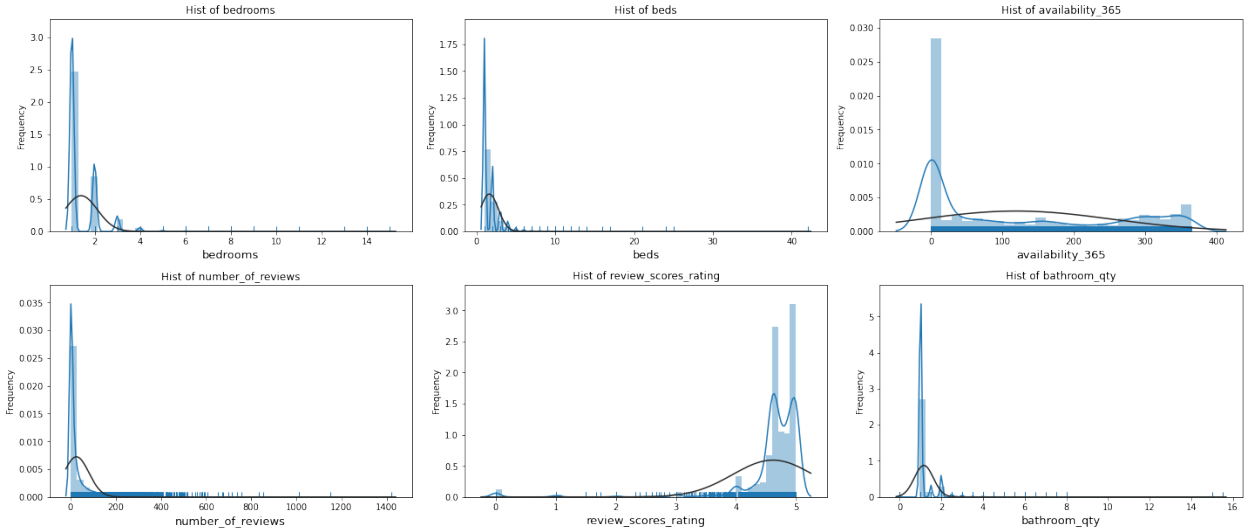
9.0. Correlation analysis

```
corr=new_df.corr().round(2)
plt.figure(figsize=(20,20))
sns.heatmap(corr, annot=True, cmap=cmap, center=0)
plt.title('Correlation matrix of numerical features')
```

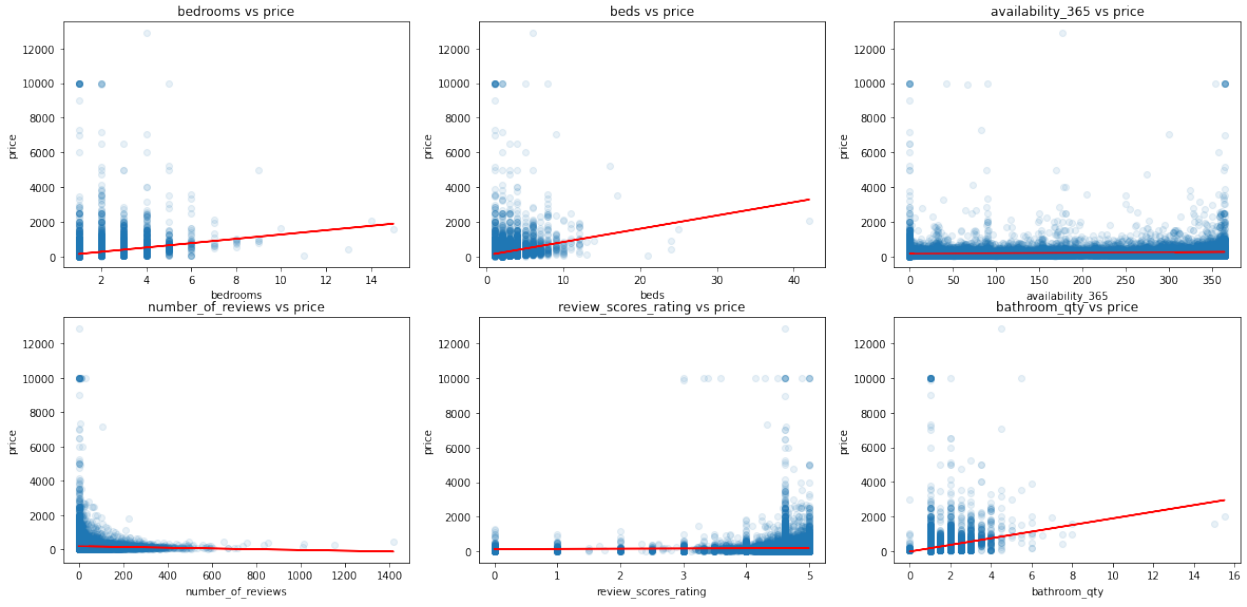


10.0. Variable distribution

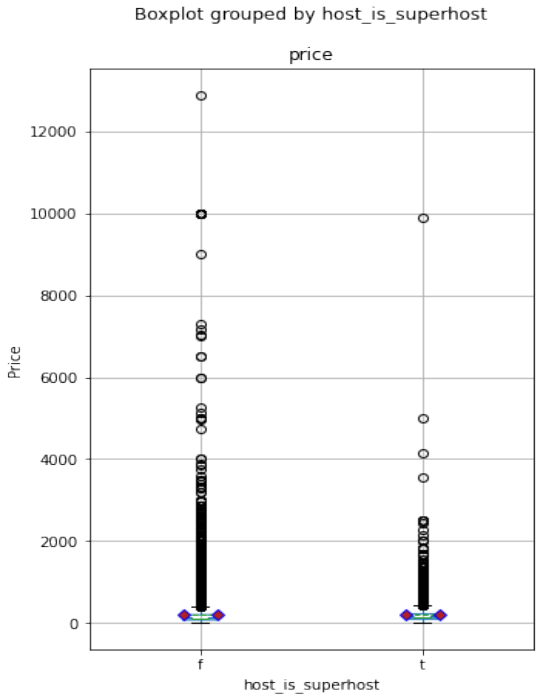
```
fig = plt.figure(figsize=(20, 25))
for i in numerical_var:
    plt.subplot(6,3,numerical_var.index(i)+1)
    sns.distplot(new_df[i], rug=True, fit=norm)
    plt.xlabel(i, fontsize=13)
    plt.ylabel('Frequency')
    plt.title(f'Hist of {i}')
    plt.tight_layout()
```



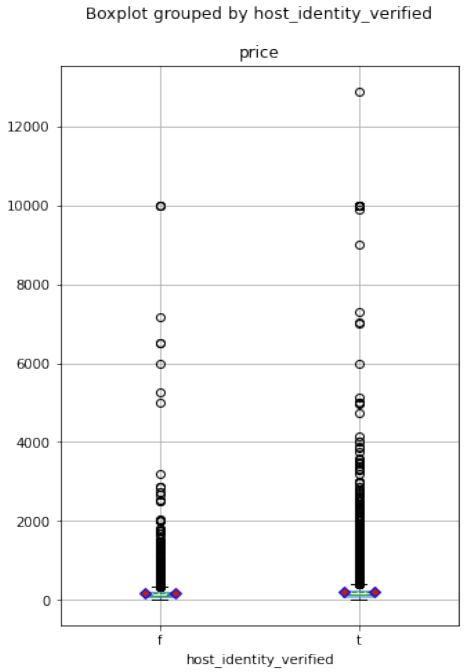

```
fig = plt.figure(figsize=(20, 30))
for i in numerical_var:
    fig.add_subplot(6,3,numerical_var.index(i)+1)
    x = new_df[i].to_numpy().reshape(-1, 1)
    y = new_df['price']
    plt.scatter(x, y
                , alpha = 0.1)
    reg = linear_model.LinearRegression()
    reg.fit(x, y)
    y_pre = reg.predict(x)
    plt.plot(x, y_pre , c = 'r')
    plt.title(f'{i} vs price')
    plt.xlabel(i)
    plt.ylabel('price')
```



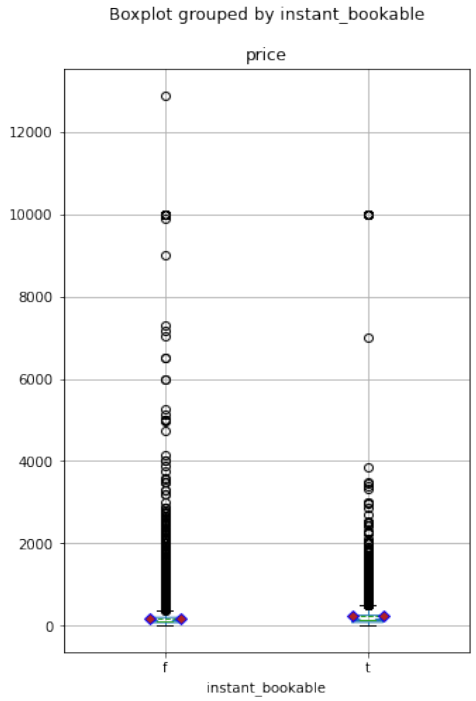
```
box_host_is_superhost = new_df.boxplot(figsize=(5,8)
                                       , column = "price"
                                       , by = "host_is_superhost"
                                       , notch=True
                                       , showmeans = True
                                       , meanprops = dict(marker='D'
                                                          , markeredgecolor='blue'
                                                          , markerfacecolor='firebrick')
                                       ,meanline=True)
box_host_is_superhost.set_ylabel("Price")
plt.show()
```




```
box_host_identity_verified = new_df.boxplot(figsize=(5,8)
                                       , column = "price"
                                       , by = "host_identity_verified"
                                       , notch=True
                                       , showmeans = True
                                       , meanprops = dict(marker='D'
                                                          , markeredgecolor='blue'
                                                          , markerfacecolor='firebrick')
                                       ,meanline=True)
box_host_is_superhost.set_ylabel("Price")
plt.show()
```

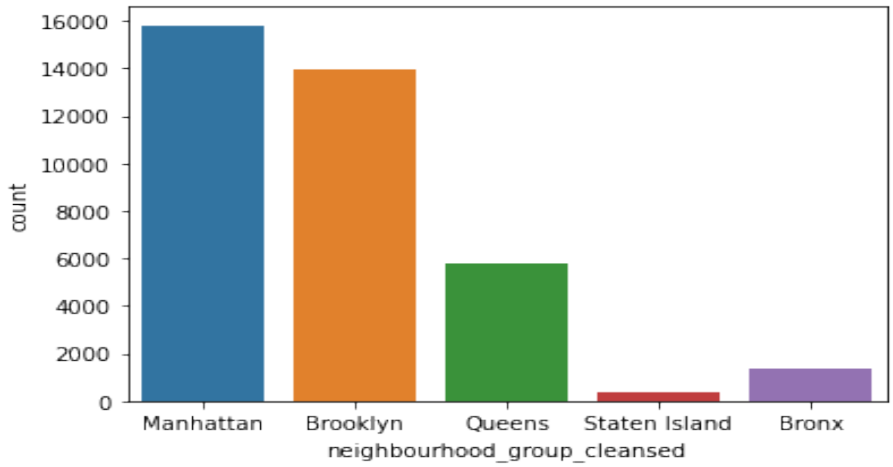


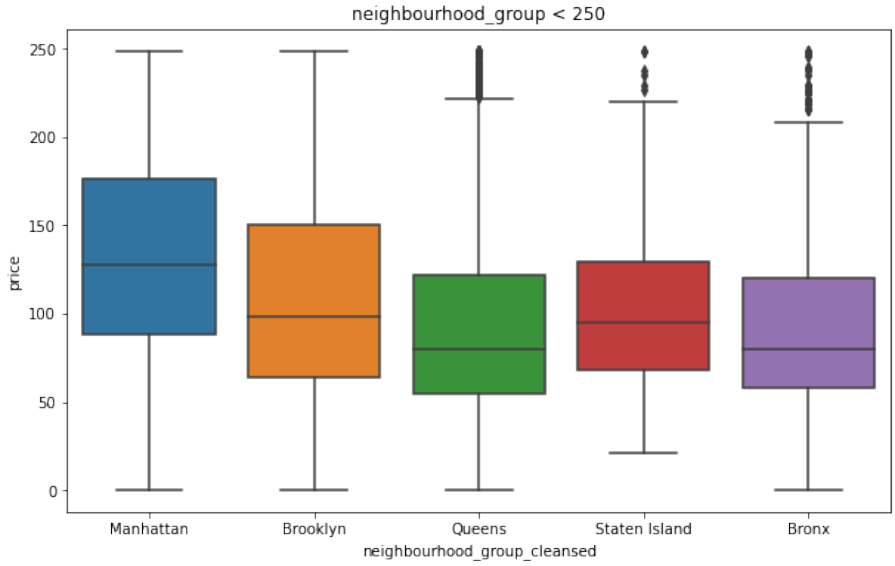

```
box_instant_bookable = new_df.boxplot(figsize=(5,8)
                                      , column = "price"
                                      , by = "instant_bookable"
                                      , notch=True
                                      , showmeans = True
                                      , meanprops = dict(marker='D'
                                                         , markeredgecolor='blue'
                                                         , markerfacecolor='firebrick')
                                      ,meanline=True)
box_host_is_superhost.set_ylabel("Price")
plt.show()
```




```
sns.countplot(df["neighbourhood_group_cleansed"])
plt.show()
```

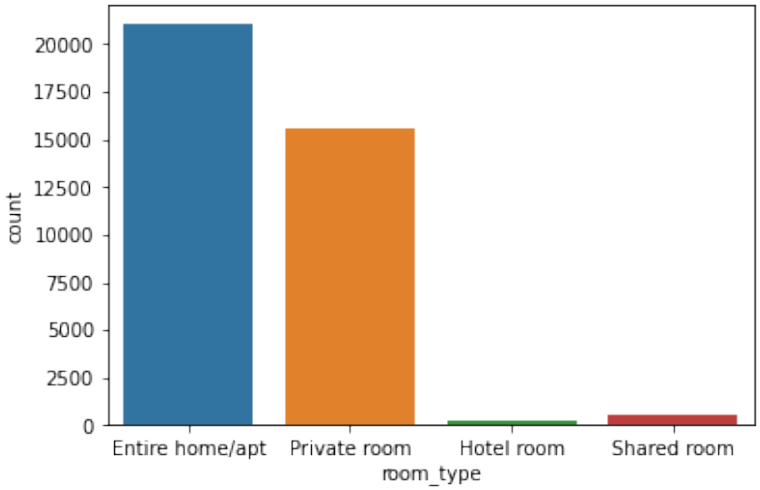
```
df1 = df[df.price < 250]
plt.figure(figsize=(10,6))

sns.boxplot(x = 'neighbourhood_group_cleansed',
            y = 'price',
            data=df1
           )

plt.title("neighbourhood_group < 250")

plt.show()
```





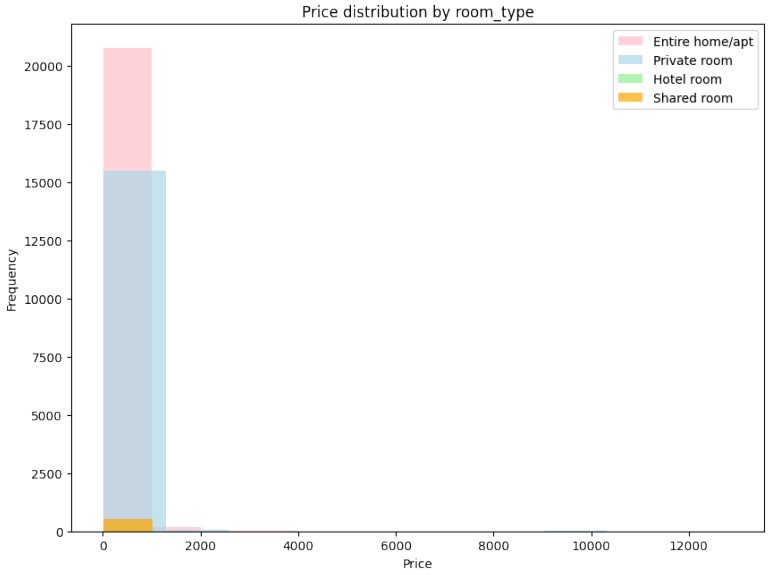
It can be seen that the average listing price in Manhattan is more than any other district in the city, and that it was also the most favored location.

```
sns.countplot(df["room_type"])
plt.show()
```



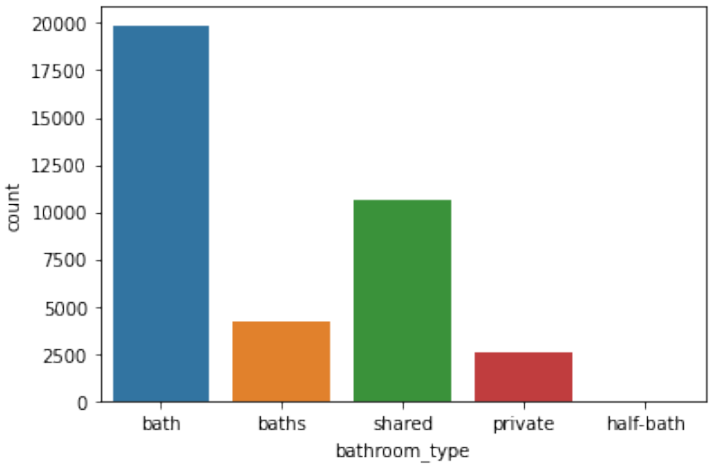
```
room_lis = list(new_df.room_type.unique())
```

```
figure = plt.figure(figsize = (10,7.5) ,dpi = 100)
color_list = ['pink', 'lightblue', 'lightgreen','orange']
n = 0
for i in room_lis:
    a = new_df[new_df.room_type == i].price
    plt.hist(a
             , color = color_list[n]
             , alpha = 0.7
             , label = i)
    n += 1
plt.legend()
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.title('Price distribution by room_type')
plt.show()
```

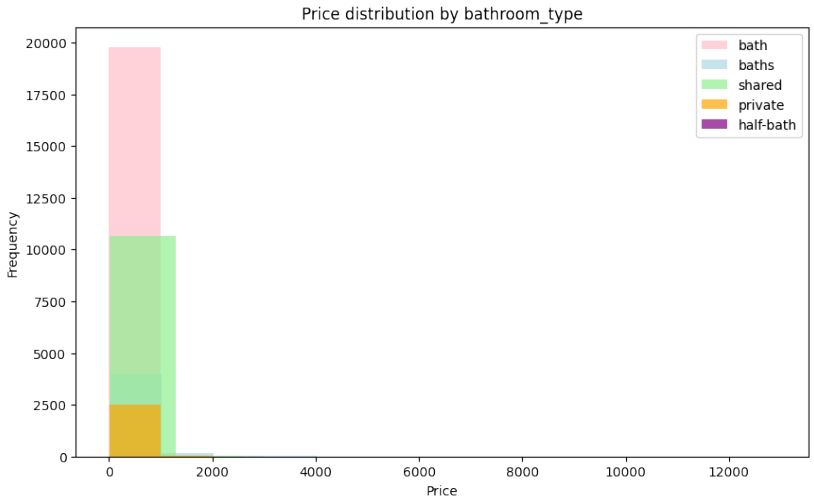


It can be seen that the number of shared rooms and hotel rooms is low, whereas entire home and apartments top the list.

```
sns.countplot(df["bathroom_type"])
plt.show()
```




```
type_lis = list(new_df.bathroom_type.unique())
figure = plt.figure(figsize = (10,6) ,dpi = 100)
color_list = ['pink', 'lightblue', 'lightgreen','orange', 'purple']
n = 0
for i in type_lis:
    a = new_df[new_df.bathroom_type == i].price
    plt.hist(a
             , color = color_list[n]
             , alpha = 0.7
             , label = i)
    n += 1
plt.legend()
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.title('Price distribution by bathroom_type')
plt.show()
```



11.0. Summary

- Raw dataset includes 17 columns and 37410 rows. Out of there 17 columns, 5 are float64, 4 int64, and 8 object
- 7 features have missing value. Out of there 7 features, 5 are less than 1%. The attributes "bedrooms" and "reviews score rating" had 10% and 21% of missing values respectively
- No strong correlation has been found between the variables
- Target variable “price” has extremely value and is heavily skewed
- Availability_365 contains a number of properties that are available only for a couple of days

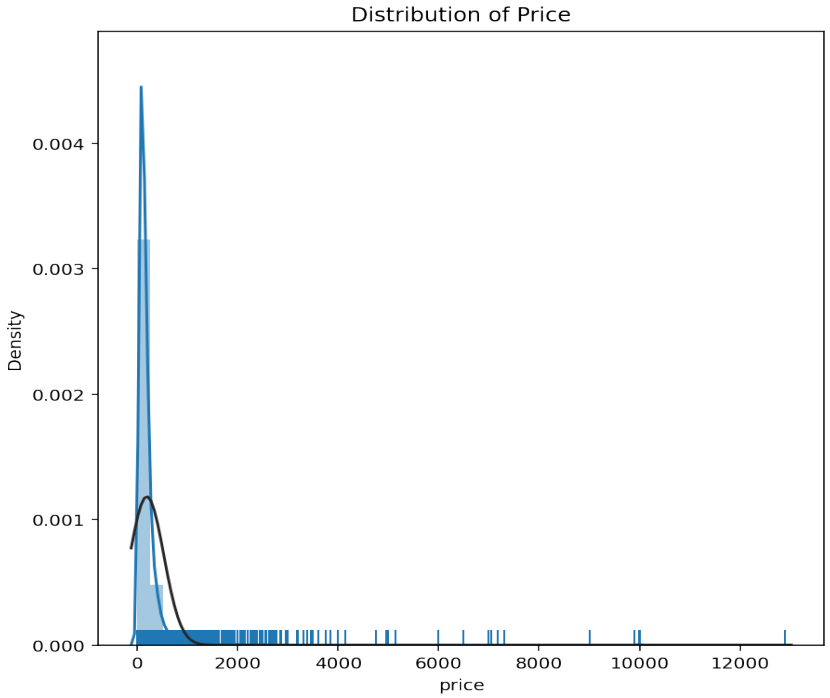
12.0. Data Preparation Needs

The pricing distribution is significantly skewed to the right. This makes sense, given that the majority of Airbnb listings are personal rentals. In addition, Airbnb aggressively caters to tourists who are looking for inexpensive accommodations for short stays. There are also postings with high prices; this connects logically with hosts listing a high-value property, such as an entire house. To mitigate the right-skewed distribution, I log the target variable and replace the results in the original column.

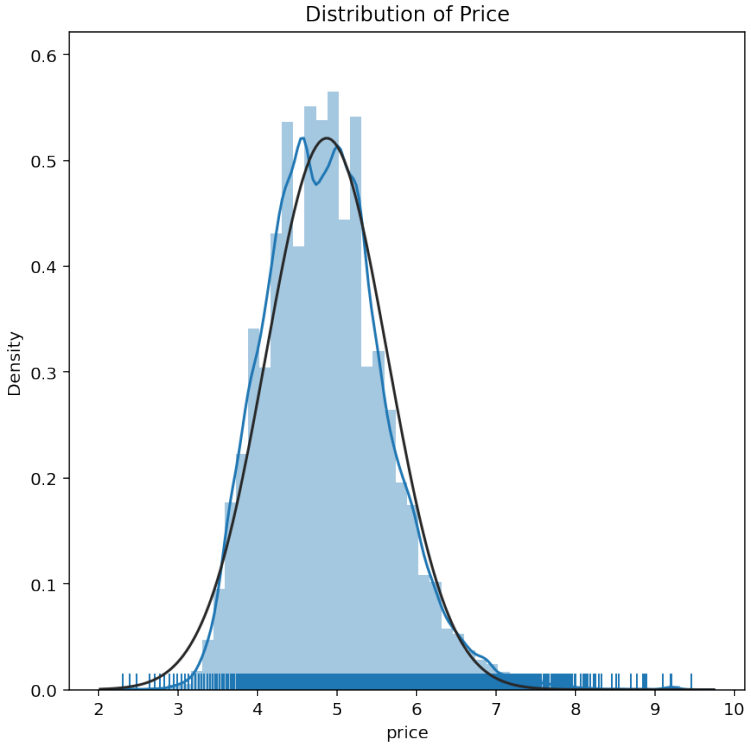
```
new_df['price'].skew()
```

```
18.02980536741831
```

```
fig = plt.figure(figsize = (7,7),dpi = 144)
sns.distplot(new_df.price
             , rug = True,
             fit=norm)
plt.title('Distribution of Price')
plt.show()
```




```
new_df = new_df[new_df.price!=0]
new_df.loc[:,"price"] = np.log(new_df.price)
fig2 = plt.figure(figsize=(7, 7), dpi=144)
sns.distplot(new_df.price, rug=True, fit=norm)
plt.title('Distribution of Price')
plt.show()
```



12.1. Log transformation for predictors

As data with a normal distribution are more conducive to model building, the log transformation is used to transform skewed data into data with a normal distribution.

```
model_df = new_df.drop(columns=['latitude', 'longitude','id' ])
```

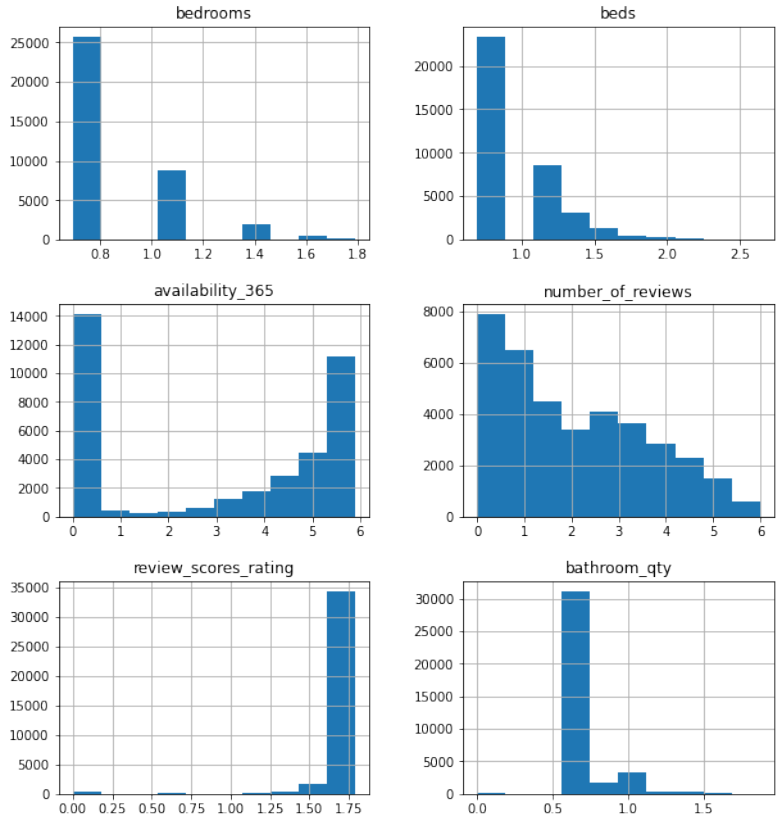
```
model_df.skew()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Dropping of nuisance columns in DataFrame reductio
  """Entry point for launching an IPython kernel.
bedrooms                1.919757
beds                    2.673700
price                   0.616263
availability_365        0.620978
number_of_reviews       3.323000
review_scores_rating   -5.045446
bathroom_qty            3.104692
dtype: float64
```

```
log_var = model_df.drop(columns=['price'])
```

```
for c in [c for c in log_var.columns if np.issubdtype(log_var[c].dtype , np.number)]:
    log_var[c] = np.log1p(log_var[c])
```

```
log_var.hist(figsize=(10,11))
```



Model Exploration

13.0. Modeling Approach/Introduction

The primary functions of regression models are relationship discovery and prediction, with the latter often resulting from the former. After determining the property's relevant predictor factors through Exploratory Analysis, I employ the DecisionTreeRegressor, RandomForestRegressor, GradientBoostingRegressor, and XGBRegressor to improve price prediction and identify the most significant variables. By attempting to predict the price, I can verify the most crucial factors influencing that price.

14.0. Decision Tree

The Decision Tree is one of the most often employed, practical ways to supervise learning. It can handle both Regression and Classification problems, with the latter having greater practical relevance. There are three sorts of nodes in this tree-structured classifier. Root Node is the starting node that represents the full sample and can be subdivided into other nodes. The Interior Nodes describe the data collection characteristics, whereas the Branch Nodes represent the decision rules. The Leaf Nodes ultimately depict the outcome. This approach is beneficial for addressing decision-related issues (K, 2020).

```
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.model_selection import train_test_split
!pip install dmba
from dmba import plotDecisionTree, classificationSummary, regressionSummary
import graphviz
from sklearn.tree import export_graphviz
```

```
X = pd.get_dummies(log_var)
y = model_df['price']

train_X, valid_X, train_y, valid_y = train_test_split(X,y, test_size=0.4, random_state=1)
```

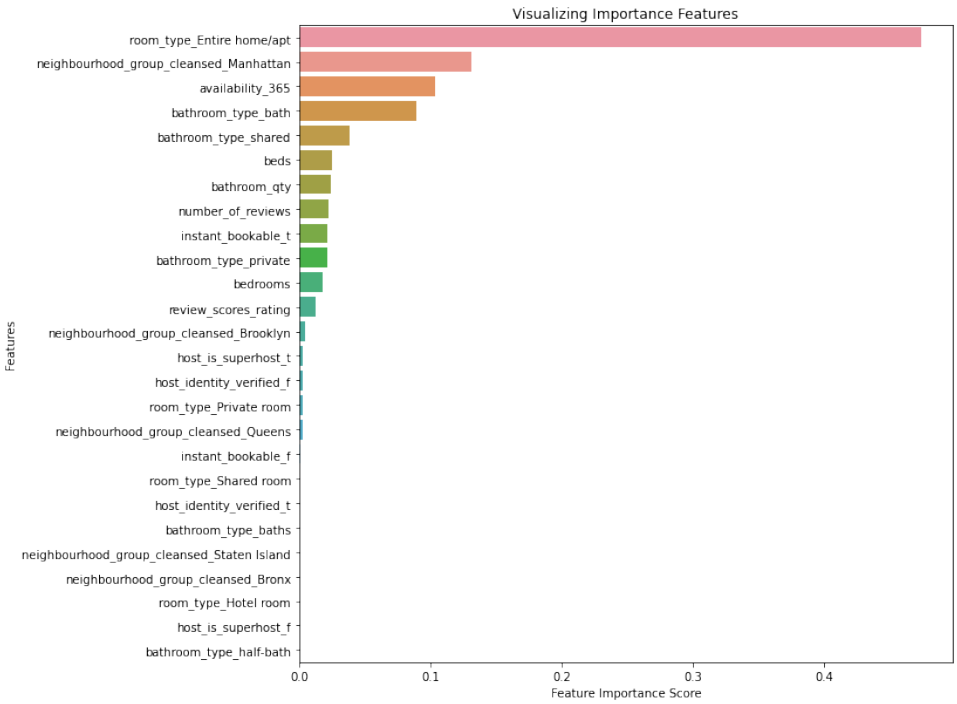
```
Parameters = {'max_depth': np.arange(1,15)
              , 'min_samples_split' : [2,3,4,5,6,7]
              , 'min_samples_leaf' : np.arange(1,15)}
dec_search = RandomizedSearchCV(DecisionTreeRegressor()
                                , Parameters
                                , scoring='neg_mean_squared_error'
                                , cv = 10
                                , random_state = 1
                                , n_jobs = 6
                                , n_iter = 100)
dec_search.fit(train_X, train_y)
dec_search.best_params_
```

```
{'max_depth': 10, 'min_samples_leaf': 14, 'min_samples_split': 4}
```

```
Dec_tree = dec_search.best_estimator_
```

```
a = pd.DataFrame(Dec_tree.feature_importances_
                 , index = train_X.columns
                 , columns=['Importance'])
a = a[a.Importance > 0].sort_values(by = 'Importance')
a
```

```
fig = plt.figure(figsize=(10, 10))
feature_imp=pd.Series(Dec_tree.feature_importances_,index=train_X.columns).sort_values(ascending=False)
sns.barplot(x = feature_imp, y=feature_imp.index)
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title('Visualizing Importance Features')
```




```
regressionSummary(valid_y, Dec_tree.predict(valid_X) )
```

```
Regression statistics

                    Mean Error (ME) : 0.0049
         Root Mean Squared Error (RMSE) : 0.5052
              Mean Absolute Error (MAE) : 0.3675
            Mean Percentage Error (MPE) : -0.8664
Mean Absolute Percentage Error (MAPE) : 7.5430
```

15.0. Random Forest

Random Forest is an ensemble method that can do both Regression and Classification tasks by utilizing many decision trees and the Bootstrap Aggression technique. Rather than relying just on individual decision trees, this technique combines numerous decision trees into a single prediction. Importance offers a score indicating how useful or valuable each feature was in developing the boosted decision trees within the model. The more a variable's relative relevance, the more it is utilized in decision trees to make crucial selections. Consequently, feature importance might be used to analyze the data in order to comprehend the most vital elements that characterize the forecasts.

```
from sklearn.ensemble import RandomForestRegressor
```

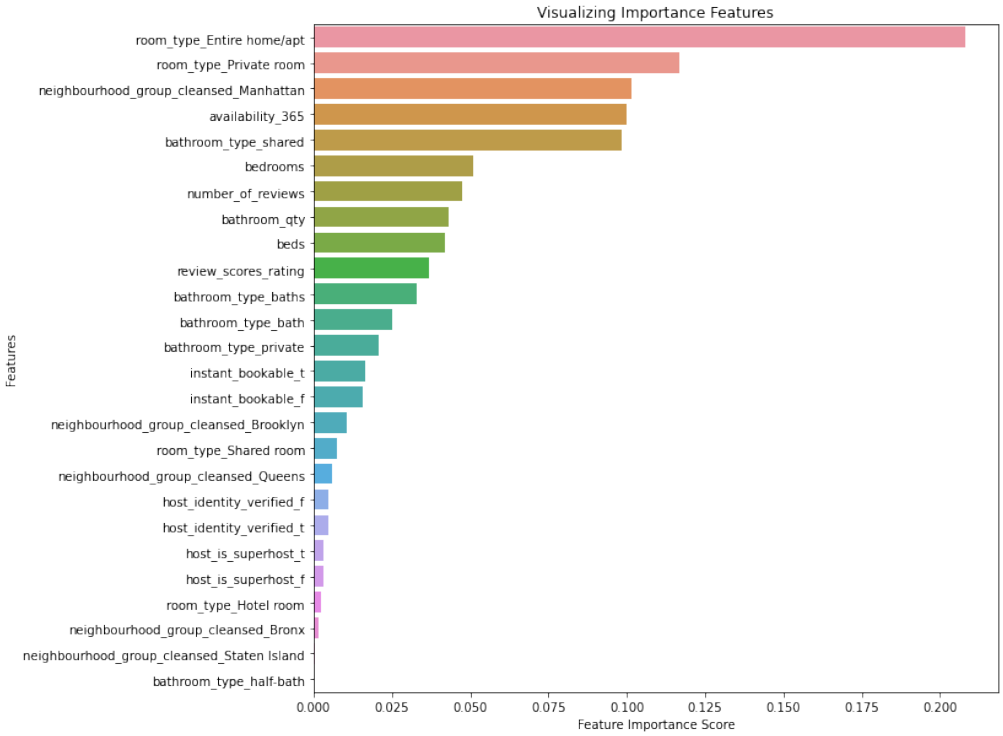
```
Parameters = {'n_estimators': [2000]
              , 'min_samples_split' : [2,3,4,5,6,7]
              , 'min_samples_leaf' : np.arange(2,10)
              , 'max_features': np.arange(1, train_X.shape[1])
              }
RF_search = RandomizedSearchCV( RandomForestRegressor()
                        , Parameters
                        , cv = 10
                        , random_state = 20
                        , n_jobs = -1)
RF_search.fit(train_X, train_y)
RF_search.best_params_
```

```
/usr/local/lib/python3.7/dist-packages/joblib/externals/loky/process_executor.py:705: UserWarning: A worker stopped while some jobs were give
  "timeout or by a memory leak.", UserWarning
{'max_features': 12,
 'min_samples_leaf': 6,
 'min_samples_split': 2,
 'n_estimators': 2000}
```

```
Rand_forest = RF_search.best_estimator_
```

```
a = pd.DataFrame(Rand_forest.feature_importances_
                 , index = train_X.columns
                 , columns=['Importance'])
a = a[a.Importance > 0].sort_values(by = 'Importance',ascending=False)
a
```

```
fig2 = plt.figure(figsize=(10, 10))
feature_imp=pd.Series(Rand_forest.feature_importances_,index=train_X.columns).sort_values(ascending=False)
sns.barplot(x = feature_imp, y=feature_imp.index)
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title('Visualizing Importance Features')
```




```
Rand_forest_result = regressionSummary(valid_y, Rand_forest.predict(valid_X) )
```

```
Regression statistics

                    Mean Error (ME) : 0.0040
     Root Mean Squared Error (RMSE) : 0.4860
          Mean Absolute Error (MAE) : 0.3527
        Mean Percentage Error (MPE) : -0.8759
Mean Absolute Percentage Error (MAPE) : 7.2491
```

16.0. Gradient Boost

Gradient boosting is a technique for continually adding decision trees so that each consecutive decision tree corrects the errors of the prior decision tree. The outcomes are more robust to parameter settings during training than to Random Forest. With the precise parameter settings, Gradient boosting can produce superior test results (Masui, 2020).

```
from sklearn.ensemble import GradientBoostingRegressor
```

```
Parameters = {'n_estimators': [2000]
              , 'learning_rate' : np.arange(0.01,0.105,0.01)
              , 'subsample' : np.arange(0.6,1.01,0.1)
              , 'max_depth': [2,3,4,5,6]}
```

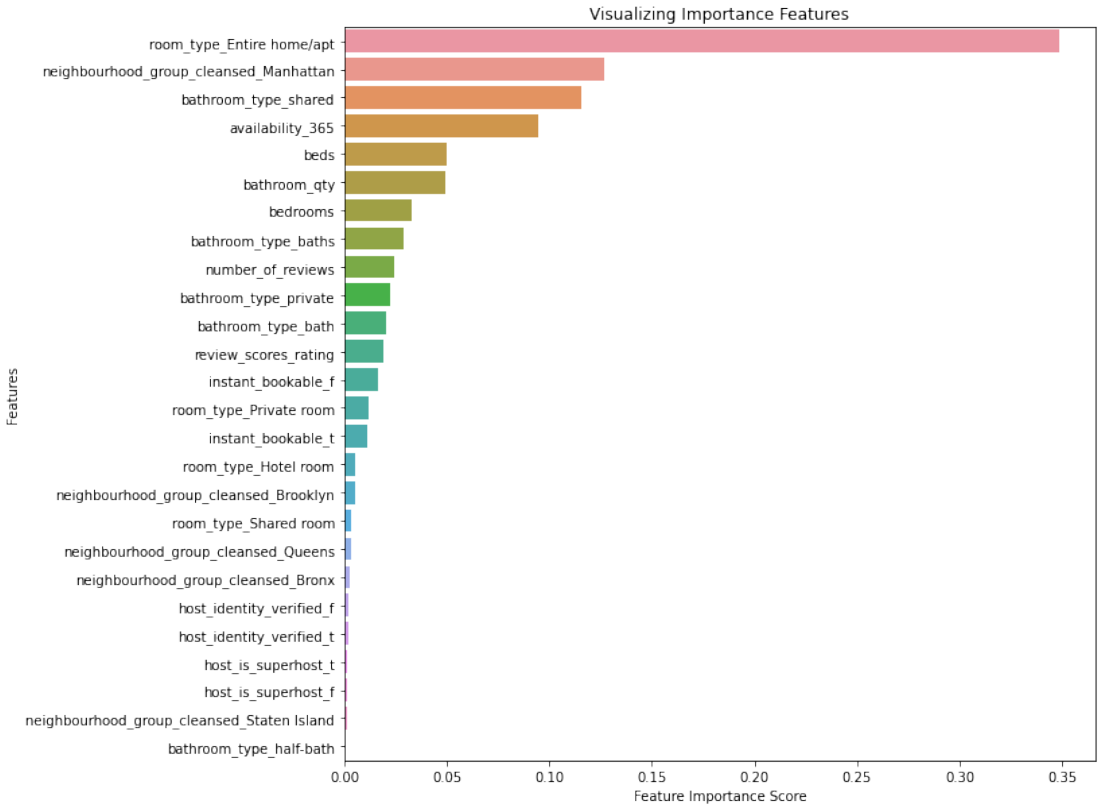
```
GB_search = RandomizedSearchCV(GradientBoostingRegressor()
                        , Parameters
                        , cv = 10
                        , random_state = 20
                        , n_jobs = -1
                        , n_iter = 20)
GB_search.fit(train_X, train_y)
GB_search.best_params_
```

```
{'learning_rate': 0.01,
 'max_depth': 4,
 'n_estimators': 2000,
 'subsample': 0.8999999999999999}
```

```
GB = GB_search.best_estimator_
```

```
a = pd.DataFrame(GB.feature_importances_
                 , index = train_X.columns
                 , columns=['Importance'])
a = a[a.Importance > 0].sort_values(by = 'Importance')
a
```

```
fig4 = plt.figure(figsize=(10, 10))
feature_imp=pd.Series(GB.feature_importances_,index=train_X.columns).sort_values(ascending=False)
sns.barplot(x = feature_imp, y=feature_imp.index)
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title('Visualizing Importance Features')
```




```
GB_result = regressionSummary(valid_y, GB.predict(valid_X) )
```

```
Regression statistics

                    Mean Error (ME) : 0.0031
     Root Mean Squared Error (RMSE) : 0.4835
          Mean Absolute Error (MAE) : 0.3535
        Mean Percentage Error (MPE) : -0.8735
Mean Absolute Percentage Error (MAPE) : 7.2750
```

17.0. XGBoost

XGBoost is an abbreviation for 'Extreme Gradient Boosting,' which employs a gradient descent framework to improve the learning ability of weak learner. It integrates machine learning algorithms within the Gradient Boosting framework and enhances the fundamental GBM framework by means of system optimization and algorithm improvement. XGBoost provides regular terms to the cost function to regulate the model's complexity. In terms of Bias-Variability Trade-off, the realization decreases model variance and prevents overfitting. XGBoost is a precise and versatile method for making predictions. Using the sparse perception approach and

automatically learning its splitting direction, it can also deal with samples with missing values.

However, optimization is difficult due to the high number of turning parameters.

```
from xgboost import XGBRegressor
```

```
%%time
xgb_model = XGBRegressor(objective='reg:squarederror', reg_lambda=0)

parameters = {
    'learning_rate': [0.01,0.05,0.1],
    'gamma' : [0.1,0.2],
    'n_estimators' : [0,500],
    'max_depth' : [5,6],
    'subsample' : [0.5,0.6,0.8,0.9,1.0],
}


xgb_model = RandomizedSearchCV(xgb_model
                               , parameters
                               , n_iter = 100
                               , cv = 10
                               , random_state=1
                               , n_jobs=6
                               , scoring ='neg_mean_squared_error')

xgb_model.fit(train_X, train_y)
xgb_model.best_params_
```

```
xgb_model.best_score_
```

```
-0.23327552064553356
```

```
rmes_boost = np.sqrt(-xgb_model.best_score_ )
rmes_boost
```

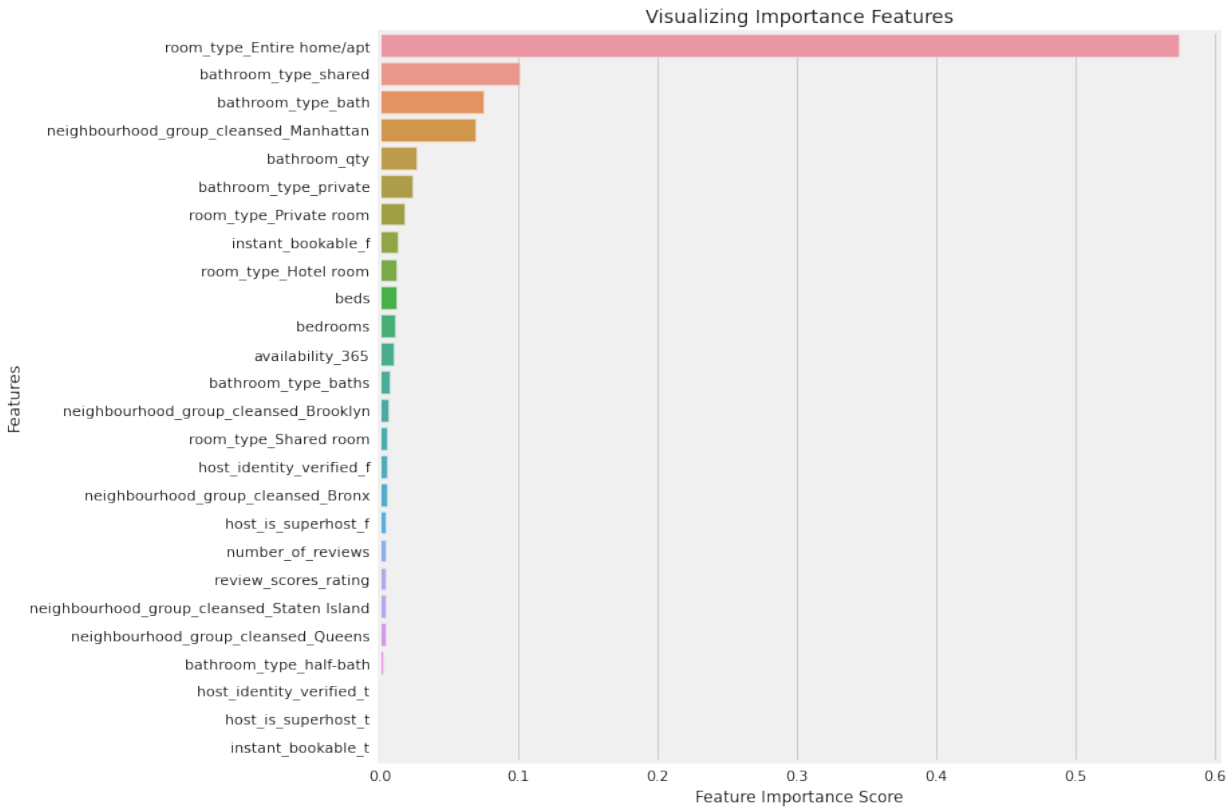
```
0.48298604601534145
```

```
xgb_model = XGBRegressor(subsample=xgb_model.best_params_['subsample'],
                         n_estimators=xgb_model.best_params_['n_estimators'],
                         max_depth =xgb_model.best_params_['max_depth'],
                         learning_rate =xgb_model.best_params_['learning_rate'],
                         gamma = xgb_model.best_params_['gamma'] )
xgb_model.fit(X,y)
```

```
a = pd.DataFrame(xgb_model.feature_importances_
                 , index = train_X.columns
                 , columns=['Importance'])
a = a[a.Importance > 0].sort_values(by = 'Importance')
a
```

	Importance
bathroom_type_half-bath	0.002254
neighbourhood_group_cleansed_Queens	0.004327
neighbourhood_group_cleansed_Staten Island	0.004353
review_scores_rating	0.004577
number_of_reviews	0.004629
host_is_superhost_f	0.004631
neighbourhood_group_cleansed_Bronx	0.004939
host_identity_verified_f	0.005231
room_type_Shared room	0.005431
neighbourhood_group_cleansed_Brooklyn	0.006384
bathroom_type_baths	0.006838
availability_365	0.010172
bedrooms	0.010986
beds	0.012145
room_type_Hotel room	0.012354
instant_bookable_f	0.013432
room_type_Private room	0.018290
bathroom_type_private	0.023845
bathroom_qty	0.026209
neighbourhood_group_cleansed_Manhattan	0.069180
bathroom_type_bath	0.074364
bathroom_type_shared	0.101105
room_type_Entire home/apt	0.574325

```
fig3 = plt.figure(figsize=(10, 10))
feature_imp=pd.Series(xgb_model.feature_importances_,index=train_X.columns).sort_values(ascending=False)
sns.barplot(x = feature_imp, y=feature_imp.index)
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title('Visualizing Importance Features')
```




```
XGB_result = regressionSummary(valid_y, xgb_model.predict(valid_X) )
```

```
Regression statistics

                    Mean Error (ME) : 0.0014
     Root Mean Squared Error (RMSE) : 0.4546
          Mean Absolute Error (MAE) : 0.3331
        Mean Percentage Error (MPE) : -0.8519
Mean Absolute Percentage Error (MAPE) : 6.8750
```

Model Recommendation

18.0. Model Selection

	RMSE
Dec_tree	0.5062
Rand_forest	0.4860
GB	0.4835
Xgb_model	0.4546

```
Model_performance = pd.DataFrame({ 'R2' : r2_list}
                                 , index = model_name_list)
Model_performance
```

	R2
Dec_tree	0.161362
Rand_forest	0.163738
GB	0.194981
xgb_model	0.190081

When R-squared is compared, it is clear that Gradient boosting has a significantly higher R-squared than decision trees and random forests, and is slightly 0.4% higher than XGBoost. However, when comparing RMSE, XGBoost has the lowest error for prediction. As suggested by the 'XGBoost' model, numerous sorts of features, such as the size of the home or apartment and the number of bathrooms, have a significant impact on the listing price. With the 'XGBoost' model, the listing price of accommodations may be accurately predicted by outlining these important features and assisting property owners in setting their desired price.

19.0. Model Assumptions and Limitations

XGBoost is a tree-based. It makes no difference whether there is a single decision tree, a random forest with 100 trees, or a 1000-tree XGBoost model. Due to the process through which tree-based models partition the input space of any given problem, these algorithms are unable to predict target values beyond the boundaries of the training data (Brownlee, 2016).

20.0. Model Sensitivity to Key Drivers

Once the boosted trees have been built using gradient boosting, retrieving significance ratings for each attribute is a simple. For a single decision tree, importance is determined by the degree to which each attribute split point enhances the performance measure, weighted by the number of observations the node is responsible for. The performance metric may be the purity - Gini index

used to determine the split points or another error function with more particular parameters (Abu-Rmileh, 2019).

Validation and Governance

21.0. Variable Level Monitoring

```
df.describe()
```

	id	latitude	longitude	bedrooms	beds	price	availability_365	number_of_reviews	review_scores_rating	bathroom_qty
count	3.741000e+04	37410.000000	37410.000000	33756.000000	36509.000000	37410.000000	37410.000000	37410.000000	29461.000000	37265.000000
mean	6.100385e+16	40.729308	-73.946171	1.354219	1.634720	190.775221	119.704892	26.347875	4.615201	1.158621
std	1.806040e+17	0.058041	0.056701	0.734038	1.148007	342.491748	132.758373	55.060905	0.759865	0.455790
min	2.595000e+03	40.504560	-74.269520	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.470053e+07	40.688260	-73.983518	1.000000	1.000000	75.000000	0.000000	1.000000	4.590000	1.000000
50%	3.484341e+07	40.724885	-73.953794	1.000000	1.000000	125.000000	60.000000	5.000000	4.830000	1.000000
75%	5.018400e+07	40.763380	-73.926340	1.000000	2.000000	203.000000	254.000000	25.000000	5.000000	1.000000
max	6.412414e+17	40.928340	-73.693210	15.000000	42.000000	12900.000000	365.000000	1419.000000	5.000000	15.500000

In the numerical data, we can see that the target variable's price has exceptionally high values. For instance, the price of such listings is $12,900. It will influence the model, despite the fact that it is reasonable, because some houses have numerous rooms or particular amenities. Prediction requires log transformation processing. There is a minimum value of 0 in the independent variable availability_365, indicating that the listing does not accept bookings for the year, and a couple small values, indicating that only just few days in a year are available for reservation. The optimal number of days to accept reservations per year is 200. These outliers must be treated with cap and floor to make the model more accurate.

```
variable_dic = {'nominal_var':['room_type', 'bathroom_type', 'neighbourhood_group_cleansed'],
                'binary_var':['host_is_superhost', 'host_identity_verified', 'instant_bookable'],
                'numerical_var':['id', 'latitude', 'longitude', 'bedrooms', 'beds', 'availability_365', 'number_of_reviews', 'review_scores_rating', 'bathroom_qty']}
```

In the dataset,3 variables are categorical features that needed to be transformed into numerical using dummy encoding. To fit the model, n-1 dummy variables will be created for each category. The primary step in encoding categorical data is category reduction. The number of created

dummy variables from categorical features should be as few as possible to avoid the overfitting problem and reduce the computation time caused by too many features.

In addition, 3 binary variables need to proceed with encoding, “0” and “1” are used to encode category “t” and “f” respectively.

22.0. Model Health & Stability

```
from sklearn.model_selection import cross_val_predict as cvp
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import r2_score as r2
```

```
model_name_list = ['Dec_tree', 'Rand_forest', 'GB', 'xgb_model']
r2_list = []
for i in model_name_list:
    if i == 'Dec_tree':
        pred = cvp(eval(i),train_X,train_y, cv = 10)
        r2_list.append(r2(np.exp(train_y), np.exp(pred)))
    else:
        pred = cvp(eval(i),train_X,train_y, cv = 10)
        r2_list.append(r2(np.exp(train_y), np.exp(pred)))
```

```
Model_performance = pd.DataFrame({ 'R2' : r2_list}
                                 , index = model_name_list)
Model_performance
```

	R2
Dec_tree	0.161362
Rand_forest	0.163738
GB	0.194981
xgb_model	0.190081

The coefficient of determination (R-squared) represents the proportion of variance in the response variable y that the independent variable X can account for in a linear regression model. The linear regression model explains more variance as the R-squared increases. When comparing R-squared, it is evident that Gradient boosting has a substantially larger R-squared than decision trees and random forests and is slightly 0.4% greater than XGBoost.

23.0. Initial Model Fit Statistics

	RMSE
Dec_tree	0.5062

Rand_forest	0.4860
GB	0.4835
Xgb_model	0.4546

RMSE is a statistic that provides the square root of the average squared difference between predicted and actual values in a dataset. The smaller the RMSE, the more accurately a model fits a data set. From the table, it shows xgb_model is the smallest one.

24.0. Risk Tiering

Since all of the Airbnb data for New York is up-to-date, this result may be used as a reference with limited risk. A data package must be updated on Airbnb if there are significant changes to the set of variables.

25.0. Conclusion

The exploratory data analysis is able to provide answers to a number of the research objectives posed in the executive summary of the New York Airbnb data. I analyze information regarding the most popular room type, the costliest neighborhood, and the average property price. Moreover, with the use of visualization, the study determines if the super host or instant bookable increased the price. The positive association between a listing's price and its attributes was finally found through the use of statistics. The XGBoost model earned a higher fitness score for the data than the other models based on the results of the modeling. This model's most significant parameters for predicting price were: ‘room type of entire home or apartment’, ‘bathroom type’, ‘neighborhood in Manhattan’, and ‘number of bathrooms.

26.0. Recommended Next Steps

- Using more detailed data that includes the actual average nightly cost

- As well as projecting base prices, data on seasonality might be used to build a sequence model for calculating daily rates
- By excluding features that would not be known at the time, such as available days, and reviews rating, the model can be tailored more specifically to new listings in order to help hosts determine rates for new properties.

References

Abu-Rmileh, A. (2019, Feb 08). *The Multiple faces of 'Feature importance' in XGBoost*. Retrieved from Towards Data Science: https://towardsdatascience.com/be-careful-when-interpreting-your-features-importance-in-xgboost-6e16132588e7

Brownlee, J. (2016, August 17). *A Gentle Introduction to XGBoost for Applied Machine Learning*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

K, G. M. (2020, Jul 14). *Machine Learning Basics: Decision Tree Regression*. Retrieved from Towards Data Science: https://towardsdatascience.com/machine-learning-basics-decision-tree-regression-1d73ea003fda

Masui, T. (2020, Jan). *All You Need to Know about Gradient Boosting Algorithm − Part 1. Regression*. Retrieved from Towards Data Science: https://towardsdatascience.com/all-you-need-to-know-about-gradient-boosting-algorithm-part-1-regression-2520a34a502