ABSTRACT

This project analyzes Airbnb listings in the city of New York to better understand how different attributes such as bedrooms, location, house type amongst others can be used to accurately predict the price of listing that is optimal in terms of the host's profitability yet affordable to their guests.

This model is intended to be helpful to the internal pricing tools that Airbnb provides to its hosts.

Objective of the PROJECT is to find:

- Estimate listing price based on provided amenities
- How review scores effect price of listing
- how cancellation policy effects price of listing

dataset

collected from InsideAirBnB website for NewYork City(NYC) from jan-mar 2020
http://insideairbnb.com/get-the-data.html (http://insideairbnb.com/get-the-data.html)

Data Dictionary

- id - listing identifier that can be used to create a join with other files
- last_scraped - data scrapped date
- name -name of the listing
- host_id -unique id given to host
- host_since - joining date of host can be used to calculate host experience based on duration since the first listing
- host_is_superhost - categorical t or f - describing highly rated and relaible hosts (https://www.airbnb.co.uk/superhost (https://www.airbnb.co.uk/superhost))
- host_identity_verified - categorical t or f - another credibility metric
- host_response_rate - response rate is the percentage of new enquiries and reservation requests you responded to (by either accepting/pre-approving or declining) within 24 hours in the past 30 days
- host_listings_count - total listing host have
- neighbourhood_group- Burough of NYC
- neighbourhood_cleansed -neighbourhoods in a burough zipcode
- latitude - we will use it later to visualise the data on the map
- longitude - we will use it later to visualise the data on the map
- property_type -description of property ex:appartment,privatehome
- room_type - type of room ex:shared room
- accommodates - discrete value describing property number people can accomodate
- bathrooms - another discrete value describing property
- bedrooms - another discrete value describing property
- beds - another discrete value describing property
- bed_type - categorical value describing property type of bed ex: realbed or couch
- amenities - wifi tv dryer so on...

- price - price per night for number of included guests
- security_deposit - another continous value assiociated with the cost
- cleaning_fee - additional cost at the top of rent
- guests_included - number of guest that can be allowed on the price
- extra_people - cost of additional person per night
- minimum_nights - another discrete value that is cost related.Listing with high value of minimum nights are likely sublettings
- maximun nights -property availability
- availablitiy 365-availability of the listing from scrapped date to next 365 days
- first_review - first review date
- last_review - last review date
- number_of_reviews - total number of reviews in entire listing history
- review_scores_accuracy - discrete value - numbers between 2 and 10
- review_scores_value - discrete value - numbers between 2 and 10
- review_scores_rating - this value is calculated as weighted sum of other scores
- reviews_per_month - given reviews in a month
- instant_bookable - categorical value - t or false
- cancellation_policy - ordinal value with 5 categories that can be ordered from lowest to highest level of flexibility

In [1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
import pyforest
import statistics
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns
import random

from mpl_toolkits.mplot3d import Axes3D
import plotly.graph_objs as go

# import statistical libraries
from scipy.stats import norm,skew, boxcox_normmax
```

In [3]:
```
abm1 =  pd.read_csv('C:/Users/tharu/Desktop/Capstone Proj files/Tharun/a
abm1.head(3)
print('abm1.shape',abm1.shape)
print('abm1.size',abm1.size)
```

Out[3]:

	id	listing_url	scrape_id	last_scraped	name	summary
0	2595	https://www.airbnb.com/rooms/2595	20200212052319	2020-02-12	Skylit Midtown Castle	Beautiful, spacious skylit studio in the heart...
1	3831	https://www.airbnb.com/rooms/3831	20200212052319	2020-02-13	Cozy Entire Floor of Brownstone	Urban retreat: enjoy 500 s.f. floor in 1899 br...
2	5099	https://www.airbnb.com/rooms/5099	20200212052319	2020-02-12	Large Cozy 1 BR Apartment In Midtown East	My large 1 bedroom apartment has a true New Yo...

3 rows × 106 columns

```
abm1.shape (153254, 106)
abm1.size 16244924
```

Steps Followed

I. Data Processing

II. EDA

III. Feature Engineering

IV. Feature Selection

V. Model Building

I. Data Processing

We will do these following steps in Data Processing part:

- 1. Data Cleaning for special characters,spaces,nan and Checking Data types
- 2. Extracting new features from existing features
- 3. Imputing Missing Values
- 4. Check Numerical, Categorical Values

Initial Dropping of Unnecessary columns

In [4]:
```
# Dropping columns that are irrelevant to our analysis

# Created New Variable abm2 for dataset after dropping columns

abm = abm1.drop(columns = ['id','name',
                'summary','access','interaction',
                'listing_url','scrape_id','last_scraped',
                'space','description','experiences_offered',
                'neighborhood_overview','notes','transit',
                'house_rules','thumbnail_url','medium_url',
                'picture_url','xl_picture_url','host_url',
                'host_name','host_location','host_about',
                'host_acceptance_rate','host_thumbnail_url','host_picture
                'host_neighbourhood','host_verifications','host_has_profi
                'market','city','smart_location','country_code','is_locat
                'square_feet','minimum_minimum_nights','maximum_minimum_n
                'minimum_maximum_nights','maximum_maximum_nights','minimu
                'maximum_nights_avg_ntm','calendar_updated','zipcode',
                'neighbourhood','state',
                'street','host_listings_count',#'neighbourhood',
                'country','availability_30','availability_60','availabili
                'calendar_last_scraped','weekly_price','monthly_price',
                'review_scores_cleanliness','review_scores_checkin','revi
                'review_scores_location','review_scores_value','license',
                'jurisdiction_names','reviews_per_month','number_of_revie
                'is_business_travel_ready','require_guest_profile_picture
                'calculated_host_listings_count','calculated_host_listing
                'calculated_host_listings_count_private_rooms',
                'calculated_host_listings_count_shared_rooms','has_availa
```

In [5]:
```
abm1.shape # new dataset after removing features
```

Out[5]: (153254, 33)

In [6]:
```
abm1 = abm1.drop_duplicates()
print('abm1.shape after dropping duplicate rows: ',abm1.shape)
print('abm1.size:  ',abm1.size)
print('DataTypes wise size: \n', abm1.dtypes.value_counts())
abm1.head(2)
```

```
abm1.shape after dropping duplicate rows:  (121676, 33)
abm1.size:   4015308
DataTypes wise size:
 object     19
float64     8
int64       6
dtype: int64
```

Out[6]:

	host_since	host_response_time	host_response_rate	host_is_superhost	host_total_listings_c
0	2008-09-09	within a day	50%	f	
1	2008-12-07	within an hour	100%	f	

2 rows × 33 columns

Note1: We are left with 38 features after dropping initial columns which are repetative counts, irrelevant to anaysis, long text data, URL's and 1,22,818 rows after dropping duplicates

1. Data Cleaning

In [7]:
```
abm1.replace(('11249\n11249'),11249,inplace=True)
abm1.replace((' '),np.nan,inplace=True)
abm1.host_response_rate = abm1.host_response_rate.str[:-1].astype('float
```

In [8]:
```
def clean_data(df):

    for i in ['price','cleaning_fee','security_deposit', 'extra_people']
        df[i]=df[i].str.replace('$','').str.replace(',', '').astype(floa

    df.replace('', np.nan, inplace=True)

    return df.head(2)
clean_data(abm1)
```

Out[8]:

	host_since	host_response_time	host_response_rate	host_is_superhost	host_total_listings_c
0	2008-09-09	within a day	50.0	f	
1	2008-12-07	within an hour	100.0	f	

2 rows × 33 columns

In [9]:
```
# Converting Price our TARGET VAR into FLOAT
abm1['price']=abm1['price'].astype(float)
```

In [10]:
```
# Replacing columns with f/t with 0/1
abm1.replace({'f': 0, 't': 1}, inplace=True)

#host_super host,instantbookable,identityverified,has_availabilty,require
```

In [11]:
```
# converting Host_since dtype to datetime and creating new column host_da
from datetime import datetime

abm1.host_since = pd.to_datetime(abm1.host_since)
abm1.first_review = pd.to_datetime(abm1.first_review)
abm1.last_review = pd.to_datetime(abm1.last_review)

# Calculating the number of years and days
abm1['host_days_active_years'] = (datetime(2020, 4, 1) - abm1.host_since)
abm1['host_listing_since'] = (abm1.last_review - abm1.first_review).astyp
#abm1['host_days_active_days'] = (datetime(2020, 4, 1) - abm1.host_since)
# Printing mean and median
#print("Mean of host_years:", round(abm1['host_days_active_years'].mean(),
#print("Median of host_years:", abm1['host_days_active_years'].median())
#print("Mode of host_years:", abm1['host_days_active_years'].mode())

#print('-------------')

#print("Mean of host_days:", round(abm1['host_days_active_days'].mean(),6
#print("Median of host_days:", abm1['host_days_active_days'].median())
#print("Mode of host_days:", abm1['host_days_active_days'].mode())


# print('\nValueCounts:\n',df['host_days_active_days'].value_counts(norma


```

In [12]:

```
# splitting amenities feature

amenities_list = list(abm1.amenities)
amenities_list_string = " ".join(amenities_list)
amenities_list_string = amenities_list_string.replace('{', '')
amenities_list_string = amenities_list_string.replace('}', ',')
amenities_list_string = amenities_list_string.replace('"', '')
amenities_set = [x.strip() for x in amenities_list_string.split(',')]
amenities_set = set(amenities_set)
print('\n Number of amenities present in total:',len(amenities_set))

abm1.loc[abm1['amenities'].str.contains('Air conditioning|Central air co
abm1.loc[abm1['amenities'].str.contains('Amazon Echo|Apple TV|Game conso
abm1.loc[abm1['amenities'].str.contains('BBQ grill|Fire pit|Propane barb
abm1.loc[abm1['amenities'].str.contains('Balcony|Patio'), 'balcony'] = 1
abm1.loc[abm1['amenities'].str.contains('Beach view|Beachfront|Lake acce
abm1.loc[abm1['amenities'].str.contains('Bed linens'), 'bed_linen'] = 1
abm1.loc[abm1['amenities'].str.contains('Breakfast'), 'breakfast'] = 1
abm1.loc[abm1['amenities'].str.contains('TV'), 'tv'] = 1
abm1.loc[abm1['amenities'].str.contains('Coffee maker|Espresso machine'),
abm1.loc[abm1['amenities'].str.contains('Cooking basics'), 'cooking_basi
abm1.loc[abm1['amenities'].str.contains('Dishwasher|Dryer|Washer'), 'whi
abm1.loc[abm1['amenities'].str.contains('Elevator'), 'elevator'] = 1
abm1.loc[abm1['amenities'].str.contains('Exercise equipment|Gym|gym'), 'g
abm1.loc[abm1['amenities'].str.contains('Family/kid friendly|Children|ch
abm1.loc[abm1['amenities'].str.contains('parking'), 'parking'] = 1
abm1.loc[abm1['amenities'].str.contains('Garden|Outdoor|Sun loungers|Ter
abm1.loc[abm1['amenities'].str.contains('Host greets you'), 'host_greeti
abm1.loc[abm1['amenities'].str.contains('Hot tub|Jetted tub|hot tub|Saun
abm1.loc[abm1['amenities'].str.contains('Internet|Pocket wifi|Wifi'), 'i
abm1.loc[abm1['amenities'].str.contains('Long term stays allowed'), 'lon
abm1.loc[abm1['amenities'].str.contains('Pets|pet|Cat(s)|Dog(s)'), 'pets_
abm1.loc[abm1['amenities'].str.contains('Private entrance'), 'private_en
abm1.loc[abm1['amenities'].str.contains('Safe|Security system'), 'secure
abm1.loc[abm1['amenities'].str.contains('Self check-in'), 'self_check_in
abm1.loc[abm1['amenities'].str.contains('Smoking allowed'), 'smoking_all
abm1.loc[abm1['amenities'].str.contains('Step-free access|Wheelchair|Acc
abm1.loc[abm1['amenities'].str.contains('Suitable for events'), 'event_s
abm1.loc[abm1['amenities'].str.contains('24-hour check-in'), 'check_in_2
```

```
 Number of amenities present in total: 151
```

In [13]:
```
print('Amenities Column Names:\n',abm1.columns[35:],'\n')
print(' Number of Amenities columns after categorizing under same names:
```

```
Amenities Column Names:
 Index(['air_conditioning', 'high_end_electronics', 'bbq', 'balcony',
       'nature_and_views', 'bed_linen', 'breakfast', 'tv', 'coffee_machin
e',
       'cooking_basics', 'white_goods', 'elevator', 'gym', 'child_friendl
y',
       'parking', 'outdoor_space', 'host_greeting', 'hot_tub_sauna_or_poo
l',
       'internet', 'long_term_stays', 'pets_allowed', 'private_entrance',
       'secure', 'self_check_in', 'smoking_allowed', 'accessible',
       'event_suitable', 'check_in_24h'],
      dtype='object')

 Number of Amenities columns after categorizing under same names: (28,)
```

Note 2: Above are the Columns after categorizing amenieties with similar names we are left with 28 features from 148 columns in amenities set

In [14]:
```
frequent_amenities = []
infrequent_amenities=[]
for col in abm1.iloc[:,35:].columns:
    if abm1[col].sum() > len(abm1)/5:
        frequent_amenities.append(col)
    else:
        infrequent_amenities.append(col)
print('Common_amenities: \n',frequent_amenities)
print('-----------------------')
print('Special_amenities: \n',infrequent_amenities)
print('frequent_amenities',len(frequent_amenities))
print('infrequent_amenities',len(infrequent_amenities))
```

```
Common_amenities:
 ['air_conditioning', 'bed_linen', 'tv', 'coffee_machine', 'cooking_basic
s', 'white_goods', 'elevator', 'child_friendly', 'parking', 'internet', 'lo
ng_term_stays', 'private_entrance', 'self_check_in']
-----------------------
Special_amenities:
 ['high_end_electronics', 'bbq', 'balcony', 'nature_and_views', 'breakfas
t', 'gym', 'outdoor_space', 'host_greeting', 'hot_tub_sauna_or_pool', 'pets
_allowed', 'secure', 'smoking_allowed', 'accessible', 'event_suitable', 'ch
eck_in_24h']
frequent_amenities 13
infrequent_amenities 15
```

In [15]:
```
# Decreasig the value_counts in cancellation policy
abm1.cancellation_policy.replace({
    'super_strict_30': 'strict',
    'super_strict_60': 'strict',
    'strict_14_with_grace_period': 'strict'}, inplace=True)
abm1.cancellation_policy.value_counts()
```

Out[15]:
```
strict      61231
flexible    31258
moderate    29187
Name: cancellation_policy, dtype: int64
```

In [16]:
```
# Decreasig the value_counts in property_type
abm1['property_type'].value_counts()

abm1['property_type'].value_counts()/abm1['property_type'].value_counts(

#With 10 categories we account for 98% of the listings

(abm1['property_type'].value_counts()/abm1['property_type'].value_counts
```

Out[16]:
```
Apartment                92006
House                    11184
Townhouse                 4756
Condominium               4323
Loft                      3552
Serviced apartment        1269
Guest suite               1234
Boutique hotel            1175
Hotel                      802
Guesthouse                 202
Other                      196
Resort                     192
Hostel                     160
Bed and breakfast          142
Villa                       93
Bungalow                    91
Tiny house                  63
Camper/RV                   47
Aparthotel                  40
Cottage                     22
Boat                        20
Earth house                 18
Tent                        14
Casa particular (Cuba)       8
Cave                         8
Houseboat                    7
Barn                         7
Yurt                         6
Cabin                        6
Bus                          4
Castle                       4
Lighthouse                   4
Dome house                   3
Dorm                         3
Timeshare                    3
Island                       3
Train                        3
Treehouse                    2
In-law                       2
Farm stay                    2
Name: property_type, dtype: int64
```

Out[16]:
```
Apartment                75.615569
House                     9.191624
Townhouse                 3.908741
Condominium               3.552878
Loft                      2.919228
Serviced apartment        1.042934
```

```
Guest suite                0.014169
Boutique hotel             0.965679
Hotel                      0.659128
Guesthouse                 0.166015
Other                      0.161084
Resort                     0.157796
Hostel                     0.131497
Bed and breakfast          0.116703
Villa                      0.076432
Bungalow                   0.074789
Tiny house                 0.051777
Camper/RV                  0.038627
Aparthotel                 0.032874
Cottage                    0.018081
Boat                       0.016437
Earth house                0.014793
Tent                       0.011506
Casa particular (Cuba)     0.006575
Cave                       0.006575
Houseboat                  0.005753
Barn                       0.005753
Yurt                       0.004931
Cabin                      0.004931
Bus                        0.003287
Castle                     0.003287
Lighthouse                 0.003287
Dome house                 0.002466
Dorm                       0.002466
Timeshare                  0.002466
Island                     0.002466
Train                      0.002466
Treehouse                  0.001644
In-law                     0.001644
Farm stay                  0.001644
Name: property_type, dtype: float64
```

Out[16]: 95.1880403695059

In [17]:
```
Mod_prop_type=abm1['property_type'].value_counts()[5:len(abm1['property_t

def change_prop_type(label):
    if label in Mod_prop_type:
        label='Other'
    return label
```

In [18]:
```
abm1.loc[:,'property_type'] = abm1.loc[:,'property_type'].apply(change_pr
```

In [19]:
```
1 abm1['property_type'].value_counts()
```

Out[19]:
```
Apartment      92006
House          11184
Other           5855
Townhouse       4756
Condominium     4323
Loft            3552
Name: property_type, dtype: int64
```

Note3: We are going to create new column that sums up the total number of ameniteis present by each host

In [20]:
```
abm1['special_amenities']=abm1[['high_end_electronics','bbq','balcony','
 'outdoor_space',
 'host_greeting',
 'hot_tub_sauna_or_pool',
 'pets_allowed',
 'secure',
 'smoking_allowed',
#  'PH_Accessible',
 'event_suitable',
 'check_in_24h',
#  'Private bathroom',
#  'Baby protection'
        ]].sum(axis=1)
abm1['special_amenities'].isnull().sum()
abm1.columns
abm1['special_amenities'].astype(float)
#abm1['special_amenities']=abm1['special_amenities'].mask(abm1['special_
abm1['special_amenities']
```

Out[20]:
```
0
```

Out[20]:
```
Index(['host_since', 'host_response_time', 'host_response_rate',
       'host_is_superhost', 'host_total_listings_count',
       'host_identity_verified', 'neighbourhood_cleansed',
       'neighbourhood_group_cleansed', 'latitude', 'longitude',
       'property_type', 'room_type', 'accommodates', 'bathrooms', 'bedroom
s',
       'beds', 'bed_type', 'amenities', 'price', 'security_deposit',
       'cleaning_fee', 'guests_included', 'extra_people', 'minimum_nights',
       'maximum_nights', 'availability_365', 'number_of_reviews_ltm',
       'first_review', 'last_review', 'review_scores_rating',
       'review_scores_accuracy', 'instant_bookable', 'cancellation_policy',
       'host_days_active_years', 'host_listing_since', 'air_conditioning',
       'high_end_electronics', 'bbq', 'balcony', 'nature_and_views',
       'bed_linen', 'breakfast', 'tv', 'coffee_machine', 'cooking_basics',
       'white_goods', 'elevator', 'gym', 'child_friendly', 'parking',
       'outdoor_space', 'host_greeting', 'hot_tub_sauna_or_pool', 'interne
t',
       'long_term_stays', 'pets_allowed', 'private_entrance', 'secure',
       'self_check_in', 'smoking_allowed', 'accessible', 'event_suitable',
       'check_in_24h', 'special_amenities'],
      dtype='object')
```

Out[20]:
```
0         0.0
1         2.0
2         1.0
3         1.0
4         0.0
          ...
153249    0.0
153250    0.0
153251    1.0
153252    2.0
153253    0.0
Name: special_amenities, Length: 121676, dtype: float64
```

Out[20]:
```
0         0.0
```

```
1         2.0
2         1.0
3         1.0
4         0.0
          ...
153249    0.0
153250    0.0
153251    1.0
153252    2.0
153253    0.0
Name: special_amenities, Length: 121676, dtype: float64
```

In [21]:
```
1 abm1.isnull().sum()
```

Out[21]:
```
host_since                    194
host_response_time          27021
host_response_rate          27021
host_is_superhost             194
host_total_listings_count     194
                              ...
smoking_allowed            116315
accessible                 115994
event_suitable             117900
check_in_24h               111356
special_amenities               0
Length: 64, dtype: int64
```

In [22]:
```
## code for merging amenities into special features and if one amenity is
abm1['common_amenities']=abm1[['bed_linen',
 'tv',
 'coffee_machine',
 'cooking_basics',
 'white_goods',
 'elevator',
 'child_friendly',
 'parking',
 'internet',
 'long_term_stays',
 'private_entrance',
 'self_check_in',
#  'Toiletries',
#  'Safety'
                                ]].sum(axis=1)
abm1['common_amenities'].isnull().sum()
abm1.columns
abm1['common_amenities'].astype(float)
# abm1['common_amenities']=abm1['common_amenities'].mask(abm1['common_ame
abm1['common_amenities']
```

Out[22]:
```
0
```

Out[22]:
```
Index(['host_since', 'host_response_time', 'host_response_rate',
       'host_is_superhost', 'host_total_listings_count',
       'host_identity_verified', 'neighbourhood_cleansed',
       'neighbourhood_group_cleansed', 'latitude', 'longitude',
       'property_type', 'room_type', 'accommodates', 'bathrooms', 'bedroom
s',
       'beds', 'bed_type', 'amenities', 'price', 'security_deposit',
       'cleaning_fee', 'guests_included', 'extra_people', 'minimum_nights',
       'maximum_nights', 'availability_365', 'number_of_reviews_ltm',
       'first_review', 'last_review', 'review_scores_rating',
       'review_scores_accuracy', 'instant_bookable', 'cancellation_policy',
       'host_days_active_years', 'host_listing_since', 'air_conditioning',
       'high_end_electronics', 'bbq', 'balcony', 'nature_and_views',
       'bed_linen', 'breakfast', 'tv', 'coffee_machine', 'cooking_basics',
       'white_goods', 'elevator', 'gym', 'child_friendly', 'parking',
       'outdoor_space', 'host_greeting', 'hot_tub_sauna_or_pool', 'interne
t',
       'long_term_stays', 'pets_allowed', 'private_entrance', 'secure',
       'self_check_in', 'smoking_allowed', 'accessible', 'event_suitable',
       'check_in_24h', 'special_amenities', 'common_amenities'],
      dtype='object')
```

Out[22]:
```
0         9.0
1         8.0
2         5.0
3         1.0
4         5.0
          ...
153249    1.0
153250    3.0
153251    1.0
153252    5.0
```

```
153253    2.0
Name: common_amenities, Length: 121676, dtype: float64
```

Out[22]:
```
0         9.0
1         8.0
2         5.0
3         1.0
4         5.0
         ...
153249    1.0
153250    3.0
153251    1.0
153252    5.0
153253    2.0
Name: common_amenities, Length: 121676, dtype: float64
```

In [23]:
```
1  abm1.columns[35:]
```

Out[23]:
```
Index(['air_conditioning', 'high_end_electronics', 'bbq', 'balcony',
       'nature_and_views', 'bed_linen', 'breakfast', 'tv', 'coffee_machin
e',
       'cooking_basics', 'white_goods', 'elevator', 'gym', 'child_friendl
y',
       'parking', 'outdoor_space', 'host_greeting', 'hot_tub_sauna_or_poo
l',
       'internet', 'long_term_stays', 'pets_allowed', 'private_entrance',
       'secure', 'self_check_in', 'smoking_allowed', 'accessible',
       'event_suitable', 'check_in_24h', 'special_amenities',
       'common_amenities'],
      dtype='object')
```

In [24]:
```
1  #dropping the actual columns
2  abm1.drop(['air_conditioning', 'high_end_electronics', 'bbq', 'balcony',
3         'nature_and_views', 'bed_linen', 'breakfast', 'tv', 'coffee_machi
4         'cooking_basics', 'white_goods', 'elevator', 'gym', 'child_friend
5         'parking', 'outdoor_space', 'host_greeting', 'hot_tub_sauna_or_poo
6         'internet', 'long_term_stays', 'pets_allowed', 'private_entrance'
7         'secure', 'self_check_in', 'smoking_allowed', 'accessible',
8         'event_suitable', 'check_in_24h','first_review',
9          'last_review','host_since','amenities'],axis=1,inplace = True)
```

In [25]:
```
1  new_col = pd.DataFrame(columns=['avg_price_property_type'])
2  new_col['avg_price_property_type'] = abm1.groupby(['neighbourhood_cleanse
```

new name is given to the dataframe "abm2" after adding new cols

In [26]:
```
abm2 = abm1.merge(new_col,left_on=['neighbourhood_cleansed','property_ty
print(abm2.shape)
print(abm2.size)
abm2.head(2)
```

```
(121676, 34)
4136984
```

Out[26]:

	host_response_time	host_response_rate	host_is_superhost	host_total_listings_count	host_
0	within a day	50.0	0.0	6.0	
1	within an hour	100.0	0.0	1.0	

2 rows × 34 columns

new name is given to the dataframe "abm3" after adding new cols

In [27]:
```
new_col1 = pd.DataFrame(columns=['avg_review_score'])
new_col1['avg_review_score'] = abm2.groupby(['neighbourhood_cleansed','pr
```

In [28]:
```
abm3 = abm2.merge(new_col1,left_on=['neighbourhood_cleansed','property_ty
print(abm3.shape)
print(abm3.size)
abm3.head(2)
```

```
(121676, 35)
4258660
```

Out[28]:

	host_response_time	host_response_rate	host_is_superhost	host_total_listings_count	host_
0	within a day	50.0	0.0	6.0	
1	within an hour	100.0	0.0	1.0	

2 rows × 35 columns

In [29]:
```
# neighbourhood_group_cleansed is renamed as Borough
#abm3['Borough'] = abm3['neighbourhood_group_cleansed']

# guests_included is renamed as Num_of_guests_incl_forprice
#abm3['Num_of_guests_incl_forprice'] = abm3['guests_included']

# extra_people is renamed as price_per_extra_people
#abm3['price_per_extra_people'] = abm3['extra_people']

abm3=abm3.rename(columns={"neighbourhood_group_cleansed": "Borough", "gue
                          ,'extra_people':'price_per_extra_people'})
```

In [30]:
```
1 abm3.columns
```

Out[30]:
```
Index(['host_response_time', 'host_response_rate', 'host_is_superhost',
       'host_total_listings_count', 'host_identity_verified',
       'neighbourhood_cleansed', 'Borough', 'latitude', 'longitude',
       'property_type', 'room_type', 'accommodates', 'bathrooms', 'bedroom
s',
       'beds', 'bed_type', 'price', 'security_deposit', 'cleaning_fee',
       'Num_of_guests_incl_forprice', 'price_per_extra_people',
       'minimum_nights', 'maximum_nights', 'availability_365',
       'number_of_reviews_ltm', 'review_scores_rating',
       'review_scores_accuracy', 'instant_bookable', 'cancellation_policy',
       'host_days_active_years', 'host_listing_since', 'special_amenities',
       'common_amenities', 'avg_price_property_type', 'avg_review_score'],
      dtype='object')
```

IMPUTING MISSING NAN VALUES

In [31]:
```
1 abm3.isnull().sum()
```

Out[31]:
```
host_response_time             27021
host_response_rate             27021
host_is_superhost                194
host_total_listings_count        194
host_identity_verified           194
neighbourhood_cleansed             0
Borough                            0
latitude                           0
longitude                          0
property_type                      0
room_type                          0
accommodates                       0
bathrooms                         97
bedrooms                         189
beds                             948
bed_type                           0
price                              0
security_deposit               36087
cleaning_fee                   20150
Num_of_guests_incl_forprice        0
price_per_extra_people             0
minimum_nights                     0
maximum_nights                     0
availability_365                   0
number_of_reviews_ltm              0
review_scores_rating           22788
review_scores_accuracy         22826
instant_bookable                   0
cancellation_policy                0
host_days_active_years           194
host_listing_since             21023
special_amenities                  0
common_amenities                   0
avg_price_property_type            0
avg_review_score                  58
dtype: int64
```

In [32]:
```
abm3.bedrooms.value_counts()
```

Out[32]:
```
1.0     85599
2.0     17219
0.0     10979
3.0      5689
4.0      1403
5.0       365
6.0       116
7.0        46
8.0        38
10.0       12
9.0        11
14.0        3
21.0        3
13.0        2
11.0        1
12.0        1
Name: bedrooms, dtype: int64
```

In [33]:
```
from sklearn.impute import KNNImputer

imputer = KNNImputer(missing_values=np.nan,n_neighbors=2, weights="unifor

abm3['host_total_listings_count'] = imputer.fit_transform(abm3[['host_to
#abm1['host_days_active_years'] = imputer.fit_transform(abm1[['host_days_
#abm1['host_days_active_days'] = imputer.fit_transform(abm1[['host_days_c
```

In [34]:
```
#Group by neighborhood_cleansed and property_type fill in missing value b

# abm3["security_deposit"] = abm3.groupby(['neighbourhood_cleansed','prop
#     lambda x: x.fillna(x.median()))

# abm3["cleaning_fee"] = abm3.groupby(['neighbourhood_cleansed','property
#     lambda x: x.fillna(x.median()))

abm3["beds"] = abm3.groupby(['neighbourhood_cleansed','property_type'])['
    lambda x: x.fillna(x.mode()))

abm3["bathrooms"] = abm3.groupby(['neighbourhood_cleansed','property_type
    lambda x: x.fillna(x.mode()[0]))

abm3["bedrooms"] = abm3.groupby(['neighbourhood_cleansed','property_type
    lambda x: x.fillna(x.mode()))

abm3["review_scores_rating"] = abm3.groupby(['neighbourhood_cleansed','pr
    lambda x: x.fillna(x.mean()))

# abm3["review_scores_accuracy"] = abm3.groupby(['neighbourhood_cleansed
#     lambda x: x.fillna(x.mean()))

```

In [35]:
```
#Group by neighborhood_cleansed and property_type fill in missing value b
abm3["host_listing_since"] = abm3.groupby(['neighbourhood_cleansed'])["ho
    lambda x: x.fillna(x.mode()))


abm3["host_is_superhost"] = abm3.groupby(['neighbourhood_cleansed'])["hos
    lambda x: x.fillna(x.mode()))


abm3["host_identity_verified"] = abm3.groupby(['neighbourhood_cleansed']
    lambda x: x.fillna(x.mode()))
```

In [36]:
```
features_nan_remove=['host_response_time']
for i in features_nan_remove:
    abm3[i]=abm3[i].astype('str').str.replace("nan", "unknown").astype(st
    print('{}:{}'.format(i,abm2[i].isna().sum()))
```

```
host_response_time:27021
```

In [37]:
```
features_nan_remove=['review_scores_rating','host_response_rate','review_
for i in features_nan_remove:
    abm3[i]=abm3[i].astype('str').str.replace("nan", "100000000").astype
    print('{}:{}'.format(i,abm3[i].isna().sum()))
```

```
review_scores_rating:0
host_response_rate:0
review_scores_accuracy:0
cleaning_fee:0
bedrooms:0
security_deposit:0
host_identity_verified:0
beds:0
host_days_active_years:0
review_scores_rating:0
host_is_superhost:0
host_listing_since:0
avg_review_score:0
```

In [38]:

```
features_nan_remove=['review_scores_rating','host_response_rate','review_
for i in features_nan_remove:
    abm3[i]=abm3[i].replace(100000000, abm3[i].median())
    print('{}:{}'.format(i,abm3[i].isna().sum()))
```

```
review_scores_rating:0
host_response_rate:0
review_scores_accuracy:0
cleaning_fee:0
bedrooms:0
security_deposit:0
host_identity_verified:0
beds:0
host_days_active_years:0
review_scores_rating:0
host_is_superhost:0
host_listing_since:0
avg_review_score:0
```

In [39]:
```
1 abm3.isnull().sum()
```

Out[39]:
```
host_response_time             0
host_response_rate             0
host_is_superhost              0
host_total_listings_count      0
host_identity_verified         0
neighbourhood_cleansed         0
Borough                        0
latitude                       0
longitude                      0
property_type                  0
room_type                      0
accommodates                   0
bathrooms                      0
bedrooms                       0
beds                           0
bed_type                       0
price                          0
security_deposit               0
cleaning_fee                   0
Num_of_guests_incl_forprice    0
price_per_extra_people         0
minimum_nights                 0
maximum_nights                 0
availability_365               0
number_of_reviews_ltm          0
review_scores_rating           0
review_scores_accuracy         0
instant_bookable               0
cancellation_policy            0
host_days_active_years         0
host_listing_since             0
special_amenities              0
common_amenities               0
avg_price_property_type        0
avg_review_score               0
dtype: int64
```

In [40]:
```
1 abm3.dtypes
```

Out[40]:
```
host_response_time              object
host_response_rate             float64
host_is_superhost              float64
host_total_listings_count      float64
host_identity_verified         float64
neighbourhood_cleansed          object
Borough                         object
latitude                       float64
longitude                      float64
property_type                   object
room_type                       object
accommodates                     int64
bathrooms                      float64
bedrooms                       float64
beds                           float64
bed_type                        object
price                          float64
security_deposit               float64
cleaning_fee                   float64
Num_of_guests_incl_forprice      int64
price_per_extra_people         float64
minimum_nights                   int64
maximum_nights                   int64
availability_365                 int64
number_of_reviews_ltm            int64
review_scores_rating           float64
review_scores_accuracy         float64
instant_bookable                 int64
cancellation_policy             object
host_days_active_years         float64
host_listing_since             float64
special_amenities              float64
common_amenities               float64
avg_price_property_type        float64
avg_review_score               float64
dtype: object
```

In [41]:
```
# seperating categorical and numerical dtypes
categorical_types=abm3.select_dtypes(include=['object']).columns
print('categorical_types: \n',categorical_types)

print('-------------')

numerical_types=abm3._get_numeric_data().columns
print('numerical_types: \n',numerical_types)
```

```
categorical_types:
 Index(['host_response_time', 'neighbourhood_cleansed', 'Borough',
       'property_type', 'room_type', 'bed_type', 'cancellation_policy'],
      dtype='object')
-------------
numerical_types:
 Index(['host_response_rate', 'host_is_superhost', 'host_total_listings_cou
nt',
       'host_identity_verified', 'latitude', 'longitude', 'accommodates',
       'bathrooms', 'bedrooms', 'beds', 'price', 'security_deposit',
       'cleaning_fee', 'Num_of_guests_incl_forprice', 'price_per_extra_peop
le',
       'minimum_nights', 'maximum_nights', 'availability_365',
       'number_of_reviews_ltm', 'review_scores_rating',
       'review_scores_accuracy', 'instant_bookable', 'host_days_active_year
s',
       'host_listing_since', 'special_amenities', 'common_amenities',
       'avg_price_property_type', 'avg_review_score'],
      dtype='object')
```

II. EDA

- Exploring the Data.
- Getting Business Insights from the data
- Cardinality of Categorical features(PLOTS)

In [42]:
```
#numerical varibles further breaking down
# CONTINOUS AND DISCRETE VARIABLES

Discrete_features=[i for i in numerical_types if len(abm3[i].unique())<2
print(Discrete_features)
print(len(Discrete_features))
```

```
['host_is_superhost', 'host_identity_verified', 'accommodates', 'bathroom
s', 'bedrooms', 'Num_of_guests_incl_forprice', 'review_scores_accuracy', 'i
nstant_bookable', 'host_days_active_years', 'host_listing_since', 'special_
amenities', 'common_amenities']
12
```

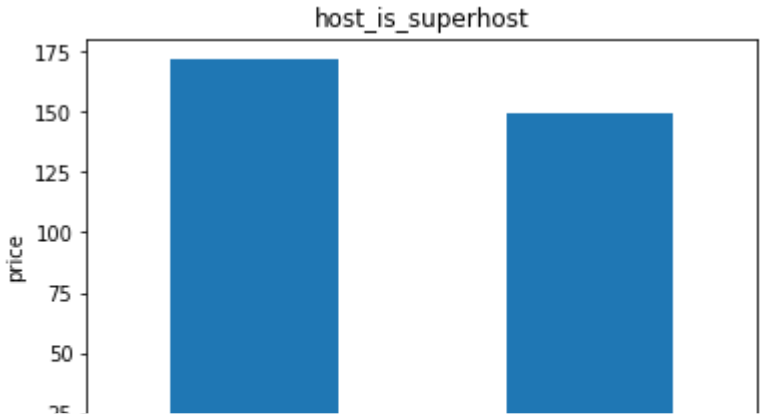
In [43]:
```
#relationship between discrete var and price
for i in Discrete_features:
    abm3.groupby(i)['price'].mean().plot.bar()
    plt.xlabel(i)
    plt.ylabel('price')
    plt.title(i)
    plt.show()
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x224e5ab3648>

Out[43]: Text(0.5, 0, 'host_is_superhost')

Out[43]: Text(0, 0.5, 'price')

Out[43]: Text(0.5, 1.0, 'host_is_superhost')



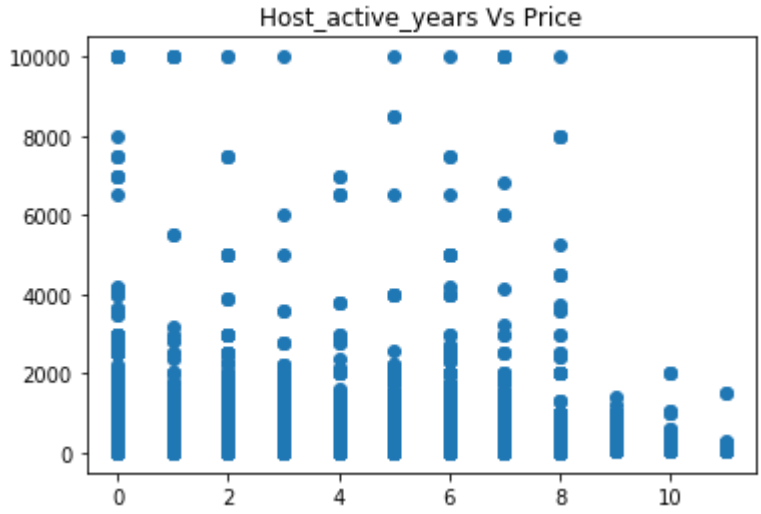
Points to be taken from above:

- from the above we can see the relationship being a superhost or not doesnot affect price
- accomodates has a linear increase in price with increase in accomadates
- special features also does not affect price much until special amenities count is greater than 7
- common amenities present properties have same price regardless of neighbourhood.
- review scores does not effect price of the property. So, we can drop review_score_accuracy.

In [44]:
```
#lets compare the difference between years and price
plt.scatter(abm3.host_days_active_years,abm3.price)
plt.title('Host_active_years Vs Price')
plt.show()
```

Out[44]: <matplotlib.collections.PathCollection at 0x224a3342a88>

Out[44]: Text(0.5, 1.0, 'Host_active_years Vs Price')



Note6:We can see that aas the number of years a host is registered and active the price is decreasing

In [45]:
```
1 abm3.dtypes
```

Out[45]:
```
host_response_time              object
host_response_rate             float64
host_is_superhost              float64
host_total_listings_count      float64
host_identity_verified         float64
neighbourhood_cleansed          object
Borough                         object
latitude                       float64
longitude                      float64
property_type                   object
room_type                       object
accommodates                     int64
bathrooms                      float64
bedrooms                       float64
beds                           float64
bed_type                        object
price                          float64
security_deposit               float64
cleaning_fee                   float64
Num_of_guests_incl_forprice      int64
price_per_extra_people         float64
minimum_nights                   int64
maximum_nights                   int64
availability_365                 int64
number_of_reviews_ltm            int64
review_scores_rating           float64
review_scores_accuracy         float64
instant_bookable                 int64
cancellation_policy             object
host_days_active_years         float64
host_listing_since             float64
special_amenities              float64
common_amenities               float64
avg_price_property_type        float64
avg_review_score               float64
dtype: object
```

In [46]:
```
## Continuous var
Continous_features=[i for i in numerical_types if i not in Discrete_featu
Continous_features[1:] #since Host_id is not required
```

Out[46]:
```
['host_total_listings_count',
 'latitude',
 'longitude',
 'beds',
 'price',
 'security_deposit',
 'cleaning_fee',
 'price_per_extra_people',
 'minimum_nights',
 'maximum_nights',
 'availability_365',
 'number_of_reviews_ltm',
 'review_scores_rating',
 'avg_price_property_type',
 'avg_review_score']
```

In [47]:
```
#ploting continous var
for i in Continous_features[1:]:
    data=abm3.copy()
    if 0 in data[i].unique():
        pass
    elif i in ['zipcode','latitude','longitude']:
        pass
    else:
#         data[i]=np.log(data[i])
        data[i].hist(bins=25)
        plt.xlabel(i)
        plt.ylabel('count')
        plt.title(i)
        plt.show()
```

Out[47]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x224a40304c8>
```

Out[47]:
```
Text(0.5, 0, 'minimum_nights')
```

Out[47]:
```
Text(0, 0.5, 'count')
```

Out[47]:
```
Text(0.5, 1.0, 'minimum_nights')
```

Note: From the above plots we observe the data is either right positively or negatively skewed, and also not normally distributed.

- Avg age of listings group by borough

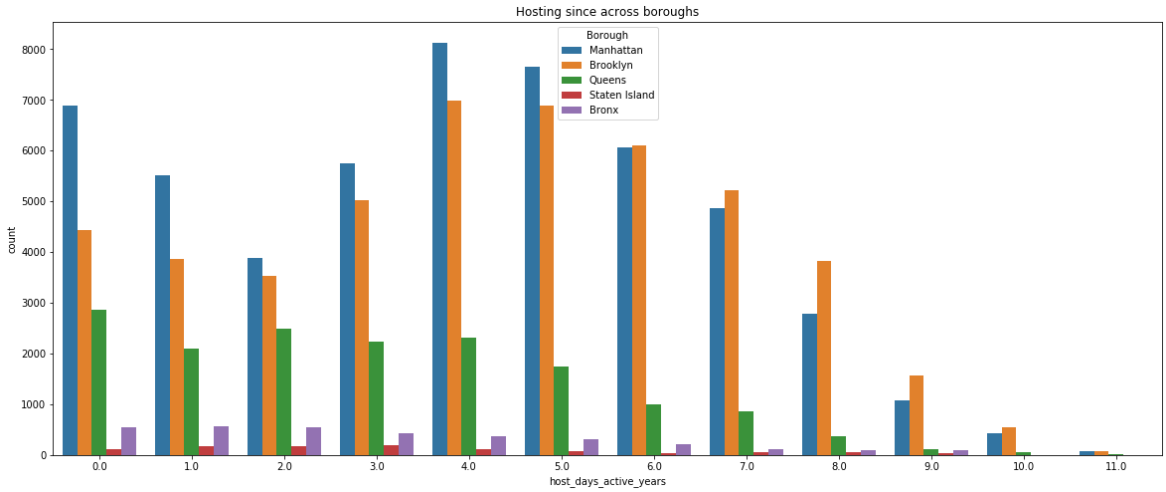
In [48]:
```
print(abm3.groupby(['Borough'])['host_days_active_years'].mean().sort_va
plt.figure(figsize=(20,8))
sns.countplot(x ='host_days_active_years',hue = "Borough",data = abm3)
plt.title("Hosting since across boroughs")
plt.show()
```

```
Borough
Bronx            2.939247
Queens           3.003464
Staten Island    3.266603
Manhattan        3.931208
Brooklyn         4.423512
Name: host_days_active_years, dtype: float64
```

Out[48]: `<Figure size 1440x576 with 0 Axes>`

Out[48]: `<matplotlib.axes._subplots.AxesSubplot at 0x224a235bac8>`

Out[48]: `Text(0.5, 1.0, 'Hosting since across boroughs')`



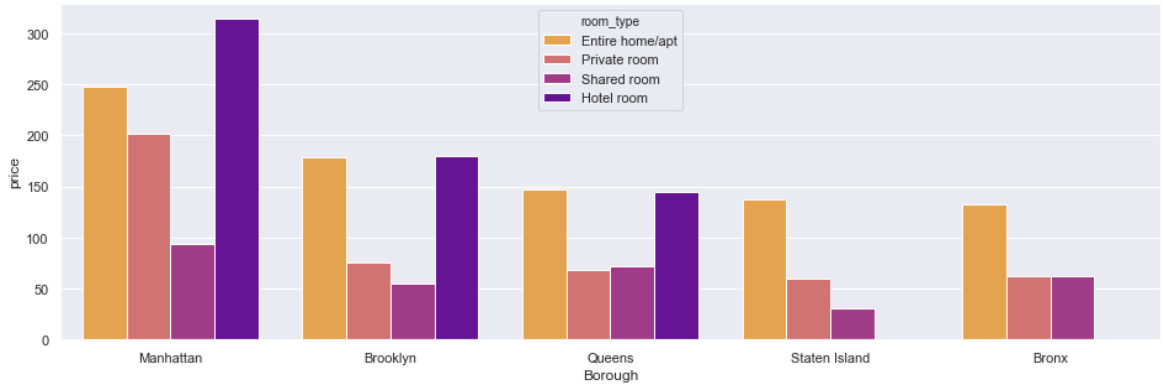
Brooklyn and Manhattan have the oldest listings by nearly 4 years.

-price distribution of various room types across neighbourhood groups

- It is clearly seen that hotel rooms are much costlier than entire house in manhattan and brooklyn region
- overall entire apt/home price is higher than private room
- Also it is seen that staten island and bronx do not have any hotel rooms

In [49]:
```
print(abm3.groupby(['room_type','Borough'])['price'].mean().sort_values(
sns.set(rc={'figure.figsize': (16, 5)})
ax = sns.barplot(x = 'Borough', y = 'price', hue = 'room_type', data =abr
                 palette ='plasma_r', ci = False)
```

```
room_type        Borough
Shared room      Staten Island     30.700000
                 Brooklyn          54.973684
Private room     Staten Island     59.732932
Shared room      Bronx             62.074830
Private room     Bronx             62.679856
                 Queens            67.929726
Shared room      Queens            72.369287
Private room     Brooklyn          75.474640
Shared room      Manhattan         93.873016
Entire home/apt  Bronx            132.799833
                 Staten Island    137.423729
Hotel room       Queens           144.518182
Entire home/apt  Queens           147.564728
                 Brooklyn         179.130284
Hotel room       Brooklyn         180.461538
Private room     Manhattan        201.839160
Entire home/apt  Manhattan        247.603358
Hotel room       Manhattan        314.123469
Name: price, dtype: float64
```



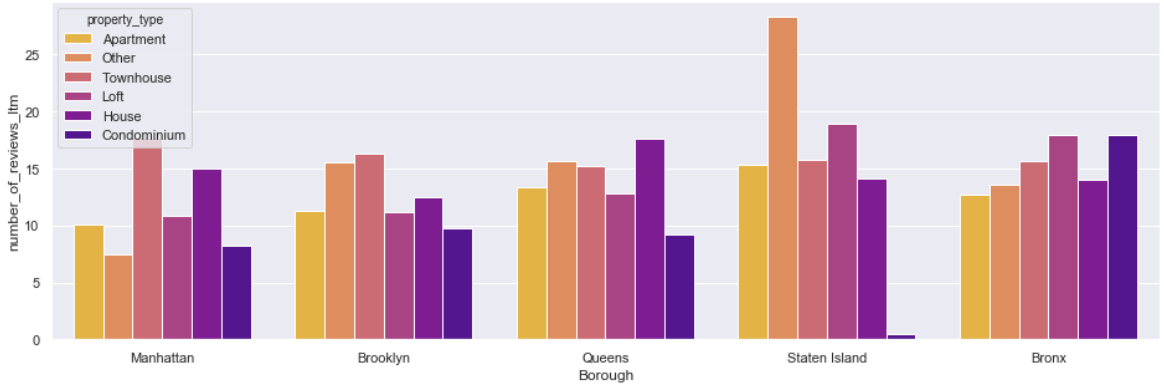
Highly reviewed of every feature based on Number of reviews(last twelve months).

- In entire city we have more number of Appartment listings
- Except for brooklyn every Borough has more Appartments than other property types.

In [50]:

```
print(abm3.groupby(['property_type'])['number_of_reviews_ltm'].mean())
sns.set(rc={'figure.figsize': (16, 5)})
ax = sns.barplot(x = 'Borough', y = 'number_of_reviews_ltm', hue = 'prope
                 palette ='plasma_r', ci = False)
```

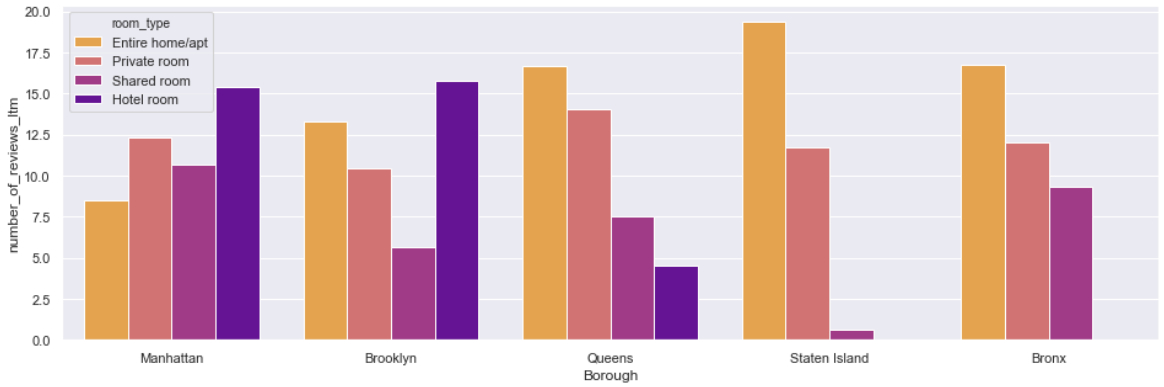
```
property_type
Apartment      10.940830
Condominium     9.136711
House          14.954220
Loft           11.279279
Other          11.029889
Townhouse      16.293734
Name: number_of_reviews_ltm, dtype: float64
```



In [51]:

```
print(abm3.groupby(['room_type'])['number_of_reviews_ltm'].mean())
sns.set(rc={'figure.figsize': (16, 5)})
ax = sns.barplot(x = 'Borough', y = 'number_of_reviews_ltm', hue = 'room_
                 palette ='plasma_r', ci = False)
```

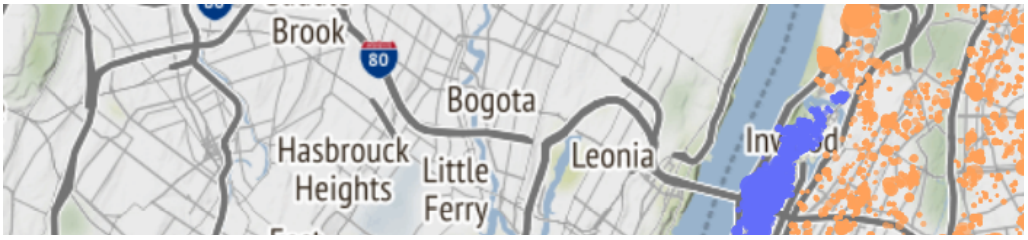
```
room_type
Entire home/apt    11.277961
Hotel room         14.425064
Private room       11.822188
Shared room         8.165271
Name: number_of_reviews_ltm, dtype: float64
```



In [52]:

```
from mpl_toolkits.mplot3d import Axes3D
from plotly.subplots import make_subplots
import plotly.graph_objs as go
from mpl_toolkits.mplot3d import Axes3D
from plotly import tools

fig = px.scatter_mapbox(abm3,
                        hover_data = ['price','minimum_nights','room_type
                        hover_name = 'neighbourhood_cleansed',
                        lat="latitude",
                        lon="longitude",
                        color="Borough",
                        size="price",
                        size_max=30,
                        opacity = .70,
                        zoom=10,
                       )
fig.layout.mapbox.style = 'stamen-terrain'
fig.update_layout(title_text = 'Airbnb by Borough in NYC<br>(Click legend
```

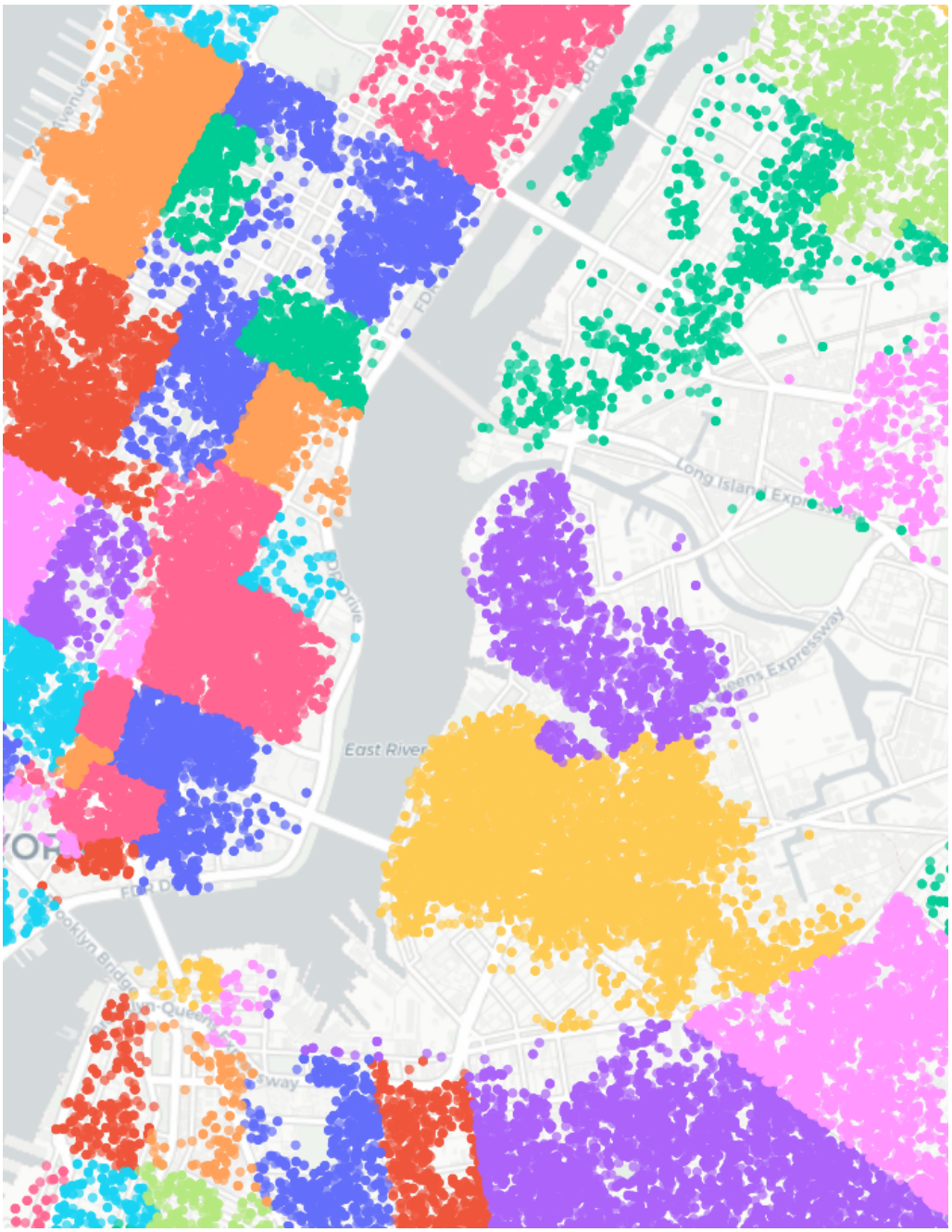
Airbnb by Borough in NYC
(Click legend to toggle borough)


Brook
80
Bogota
Hasbrouck
Heights
Little
Ferry
Leonia

In [53]:

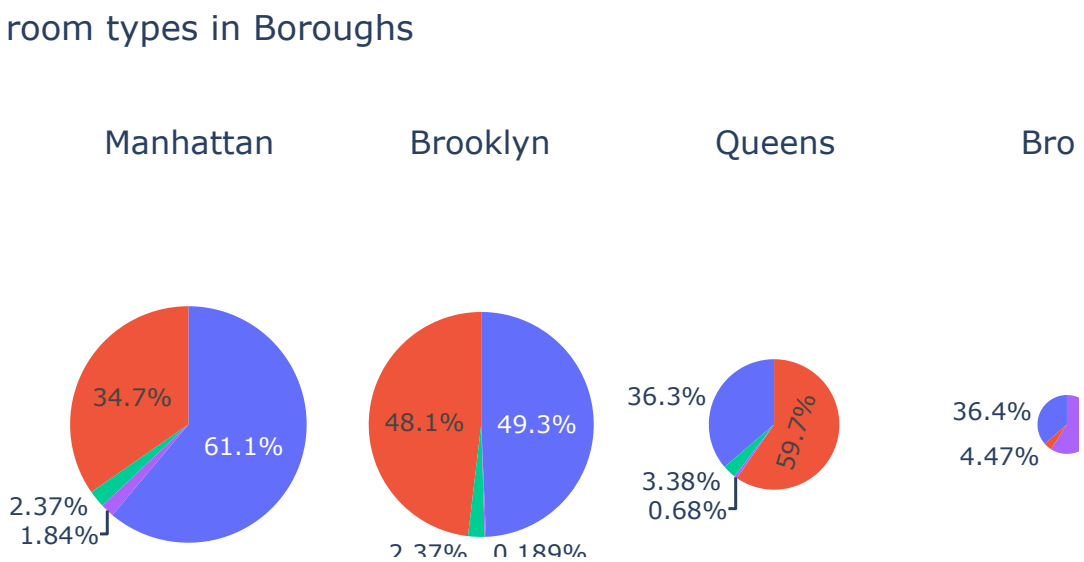
```
fig = px.scatter_mapbox(abm3,
                        hover_data=['price','property_type','room_type',
                        lat="latitude",
                        lon="longitude",
                        color="neighbourhood_cleansed",
                        size_max=30,
                        opacity = .70,
                        zoom=12,
                       )
fig.layout.mapbox.style = 'carto-positron'
fig.update_layout(title_text = 'NYC Airbnb by Neighbourhood<br>(Click leg
```

NYC Airbnb by Neighbourhood
(Click legend to toggle neighbourhood)



In [54]:

```
temp_bk = abm3[abm3.Borough == 'Brooklyn']
temp_qn = abm3[abm3.Borough == 'Queens']
temp_mn = abm3[abm3.Borough == 'Manhattan']
temp_bx = abm3[abm3.Borough == 'Bronx']
temp_si = abm3[abm3.Borough == 'Staten Island']

labels = abm3.room_type.value_counts().index.to_list()

fig = make_subplots(1, 5, specs=[[{'type':'domain'}, {'type':'domain'},
                    subplot_titles=['Manhattan', 'Brooklyn', 'Queens','Br
fig1= fig.add_trace(go.Pie(labels=labels, values=temp_mn.room_type.value_
                     name="Manhattan"),1,1)
fig2= fig.add_trace(go.Pie(labels=labels, values=temp_bk.room_type.value_
                     name="Brooklyn"),1,2)
fig3= fig.add_trace(go.Pie(labels=labels, values=temp_qn.room_type.value_
                     name="Queens"),1,3)
fig4= fig.add_trace(go.Pie(labels=labels, values=temp_bx.room_type.value_
                     name="Bronx"),1,4)
fig5= fig.add_trace(go.Pie(labels=labels, values=temp_si.room_type.value_
                     name="Staten Island"),1,5)

fig.update_layout(title_text='room types in Boroughs')
```

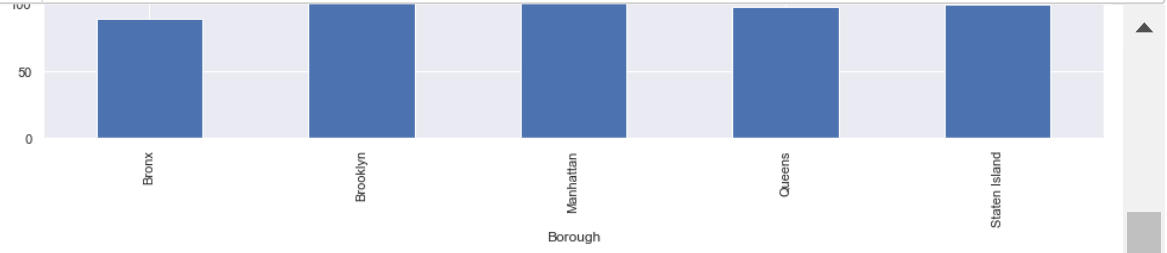


- Cardinality of categorical features

In [55]:
```
for i in categorical_types:
        print('feature-{} & number of categories-{}'.format(i,len(abm3[i
```

```
feature-host_response_time & number of categories-5
feature-neighbourhood_cleansed & number of categories-225
feature-Borough & number of categories-5
feature-property_type & number of categories-6
feature-room_type & number of categories-4
feature-bed_type & number of categories-5
feature-cancellation_policy & number of categories-3
```

In [56]:
```
#relationship b/w categorical and dependent var
for i in categorical_types:
    if len(data[i].unique())>40:
        pass
    elif len(data[i].unique())==1:
        pass
    else:
        data.groupby(i)['price'].mean().plot.bar()
        plt.xlabel(i)
        plt.ylabel('price')
        plt.title(i)
        plt.show()
```



Out[56]: <matplotlib.axes._subplots.AxesSubplot at 0x224a6ee5bc8>

Out[56]: Text(0.5, 0, 'property_type')

Out[56]: Text(0, 0.5, 'price')

Out[56]: Text(0.5, 1.0, 'property_type')

- from the above we can observe relationship between categorical feature and price feature and also we can reduce the labels

Statistics Analysis

Objective: To perform Statistical Analysis on the data set by implementing various stats modules

(on New York AirBnb data) such as Hypothesis Testing, Tests of Mean (Kruskal Wallis Test, ANOVA - one way and two way), Tests of Proportion (z test and chi-squared test) and Tests of Variance (F-test, Levene test), after checking for the three assumptions of (i) Normality of target variable (ii) Randomness of Sampling (iii) Equal variance across categories. The level of significance is assumed to be 5 percent (i.e. alpha = 0.05) If assumptions are satisfied, parametric tests(assumes already distribution is present(ANOVA) can be performed, else non-parametric tests(do not rely on any distributions(CHI-SQUARE) have to be performed. The results of the tests performed will enable us to find the associativity and dependability of different features on one-another.

In [499]:
```
cont = pd.crosstab(abm3.Borough,abm3.room_type)
cont
```

Out[499]:

room_type	Entire home/apt	Hotel room	Private room	Shared room
Borough				
Bronx	1199	0	1946	147
Brooklyn	23702	91	23127	1140
Manhattan	32455	980	18422	1260
Queens	5863	110	9648	547
Staten Island	531	0	498	10

Testing the assumptions:

- Randomness of Data
- Normality Test
- Variance Test

The target variable being the price.

In [500]:
```
plt.figure(figsize=(10,7))
sns.distplot(abm2.price,color='r')
plt.xlabel("Price")
plt.title("Distribution of Price of among property types")
plt.show()
```

Out[500]: <Figure size 720x504 with 0 Axes>

Out[500]: <matplotlib.axes._subplots.AxesSubplot at 0x224b5a24148>

Out[500]: Text(0.5, 0, 'Price')

Out[500]: Text(0.5, 1.0, 'Distribution of Price of among property types')

Shapiro Test (for checking Normality)

- H0 (Null Hypothesis) : Distribution is normal
- H1 (Alternate Hypothesis): Distribution is not normal

In [501]:
```
import scipy.stats as st
st.shapiro(abm2.price)
```

Out[501]: (0.1589747667312622, 0.0)

Levene Test (for testing of variance)

H0 (null hypothesis): variance(private_room) = variance(shared_room) = variance(entire_home)=variance(hotel_room)

H1 (alternate hypothesis): variance(private_room) != variance(shared_room) != variance(entire_home)!=variance(hotel_room)

In [502]:
```
pvt = abm3[abm3['room_type'] == 'Private room']
share = abm3[abm3['room_type'] == 'Shared room']
apt = abm3[abm3['room_type'] == 'Entire home/apt']
hotel=abm3[abm3['room_type'] == 'Hotel room']
```

In [503]:
```
st.levene(pvt.price, share.price, apt.price,hotel.price)
```

Out[503]: LeveneResult(statistic=70.80752126514611, pvalue=9.55959985524595e-46)

Here P-value is less than 0.5 and therefore we can reject null hypothesis and can say that price varies in different room types

price vs neighbourhood

H0 (null hypothesis): mean_price(Brooklyn) = mean_price(Manhattan) = ... = mean_price(Bronx)

H1 (null hypothesis): mean_price(Brooklyn) != mean_price(Manhattan) != ..... != mean_price(Bronx)

In [504]:
```
import  scipy.stats as stats
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm
```

In [505]:
```
## one way anova
mod = ols('price ~ Borough', data =abm3).fit()
aov_table = sm.stats.anova_lm(mod, typ=1)
print(aov_table)
```

```
                df        sum_sq       mean_sq           F  PR(>F)
Borough        4.0  3.891123e+08  9.727807e+07  547.682778     0.0
Residual  121671.0  2.161090e+10  1.776175e+05         NaN     NaN
```

Here pvalue obtained is greater than 0.5, so we fail to reject null hypothesis

Room Type vs Neighbourhood Group

Since both the variables Room Type and Neighbourhood Group are categorical having more than two categories, we can peform Chi-squared test.

Chi Squared Test

- H0 (null hypothesis): There is no association between Room Type and Neighbourhood Group.
- H1 (alternate hypothesis): There is an association between Room Type and Neighbourhood Group.

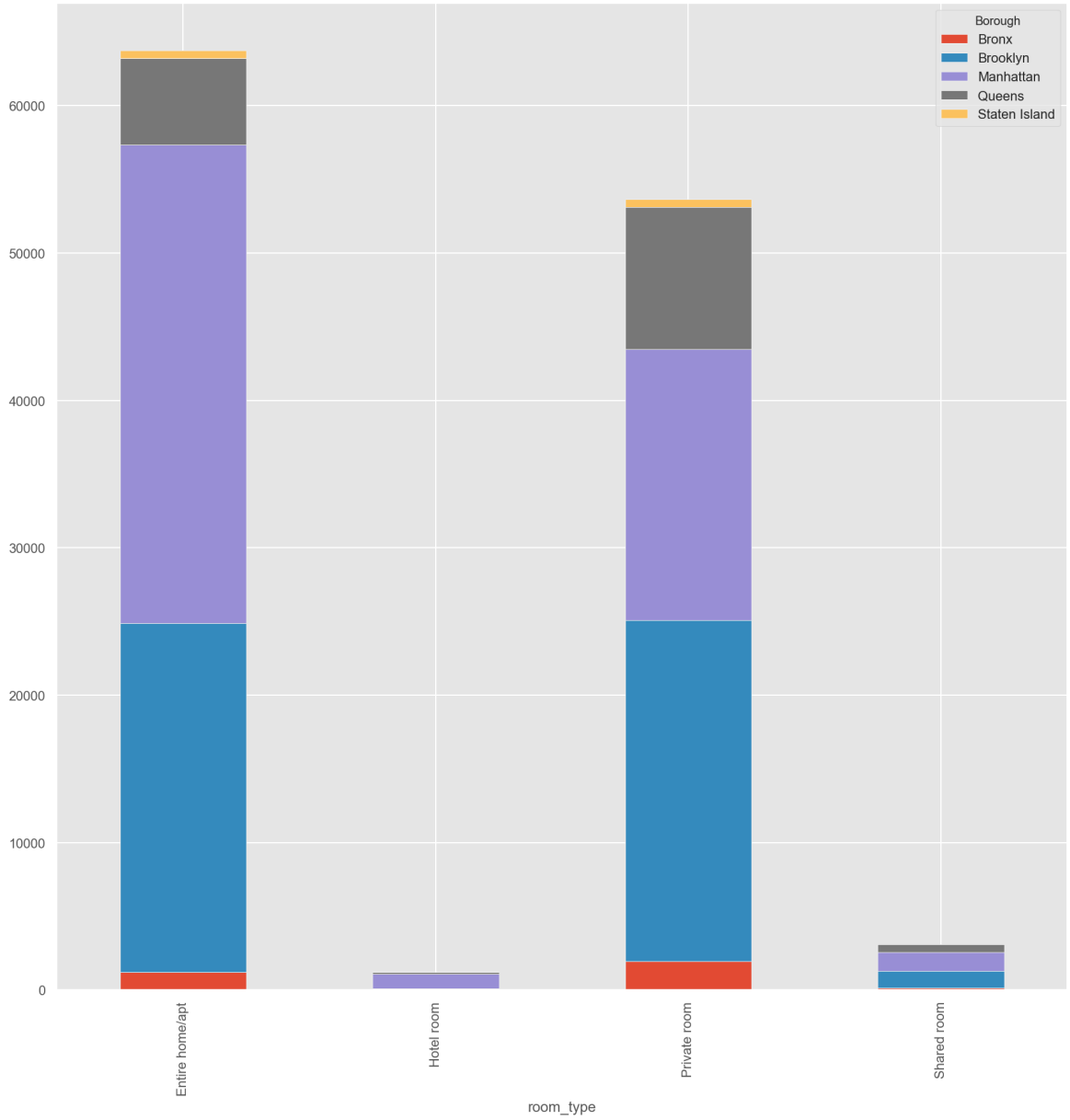
In [507]:
```
tab = pd.crosstab(abm3['room_type'],abm3['Borough'])
```

In [508]:
```
st.chi2_contingency(tab)
```

Out[508]:
```
(5016.088163783007,
 0.0,
 12,
 array([[1.72478550e+03, 2.51801917e+04, 2.78297179e+04, 8.47093922e+03,
         5.44365775e+02],
        [3.19524968e+01, 4.66475394e+02, 5.15559165e+02, 1.56928301e+02,
         1.00846428e+01],
        [1.45128186e+03, 2.11873045e+04, 2.34166886e+04, 7.12768079e+03,
         4.58044306e+02],
        [8.39801440e+01, 1.22602847e+03, 1.35503442e+03, 4.12451691e+02,
         2.65052763e+01]]))
```

In [509]:
```
1 ct = pd.crosstab(abm3['room_type'],abm3['Borough'])
2 ct.plot.bar(stacked=True)
3 plt.show()
```

Out[509]: <matplotlib.axes._subplots.AxesSubplot at 0x224b5b65188>



price vs review scores rating

In [510]:
```
mod = ols('price ~ review_scores_rating', data =abm3).fit()
aov_table = sm.stats.anova_lm(mod, typ=1)
print(aov_table)
```

```
                           df        sum_sq       mean_sq           F  \
review_scores_rating       1.0  3.946650e+07  3.946650e+07  218.666955
Residual              121674.0  2.196055e+10  1.804868e+05         NaN

                            PR(>F)
review_scores_rating  1.950579e-49
Residual                     NaN
```

here Pvalue obtained is less than 0.5, so we can reject null hypothesis and can say that reviews have an effect on price

price vs cancellation policy

In [511]:
```
mod = ols('price ~ cancellation_policy', data =abm3).fit()
aov_table = sm.stats.anova_lm(mod, typ=1)
print(aov_table)
```

```
                           df        sum_sq       mean_sq         F  \
cancellation_policy       2.0  2.601505e+07  1.300753e+07  72.02442
Residual             121673.0  2.197400e+10  1.805988e+05       NaN

                            PR(>F)
cancellation_policy  5.478900e-32
Residual                     NaN
```

here Pvalue obtained is less than 0.5, so we can reject null hypothesis and can say that cancellation policy have an effect on price

price vs availability_365

In [512]:
```
mod = ols('price ~ availability_365', data =abm3).fit()
aov_table = sm.stats.anova_lm(mod, typ=1)
print(aov_table)
```

```
                         df        sum_sq       mean_sq           F  \
availability_365       1.0  1.139076e+08  1.139076e+08  633.259721
Residual          121674.0  2.188611e+10  1.798750e+05         NaN

                          PR(>F)
availability_365  2.226360e-139
Residual                   NaN
```

chi2 on response rate and property type, borough etc

In [513]:

```
1 tab = pd.crosstab(abm3['host_response_rate'],abm3['property_type'])
```

In [514]:
```
1 st.chi2_contingency(tab)
```

Out[514]:
```
(1796.5916070258804,
 2.9040772258016473e-196,
 340,
 array([[1.02534712e+03, 4.81770275e+01, 1.24638417e+02, 1.04834906e+02,
         5.30025313e+01],
        [7.56155692e-01, 3.55287814e-02, 9.19162366e-02, 7.73118774e-02,
         3.90874125e-02],
        [1.51231138e+00, 7.10575627e-02, 1.83832473e-01, 1.54623755e-01,
         7.81748249e-02],
        [3.02462277e+01, 1.42115125e+00, 3.67664946e+00, 3.09247510e+00,
         1.56349650e+00],
        [6.04924554e+00, 2.84230251e-01, 7.35329893e-01, 6.18495020e-01,
         3.12699300e-01],
        [9.07386831e+00, 4.26345376e-01, 1.10299484e+00, 9.27742529e-01,
         4.69048950e-01],
        [7.56155692e+00, 3.55287814e-01, 9.19162366e-01, 7.73118774e-01,
         3.90874125e-01],
        [2.11723594e+01, 9.94805878e-01, 2.57365462e+00, 2.16473257e+00,
         1.09444755e+00],
        [3.02462277e+00, 1.42115125e-01, 3.67664946e-01, 3.09247510e-01,
         1.56349650e-01],
        [8.01525034e+01, 3.76605082e+00, 9.74312108e+00, 8.19505901e+00,
         4.14326572e+00],
        [7.56155692e-01, 3.55287814e-02, 9.19162366e-02, 7.73118774e-02,
         3.90874125e-02],
        [4.53693415e+00, 2.13172688e-01, 5.51497419e-01, 4.63871265e-01,
         2.34524475e-01],
        [3.78077846e+00, 1.77643907e-01, 4.59581183e-01, 3.86559387e-01,
         1.95437062e-01],
        [9.83002400e+01, 4.61874158e+00, 1.19491108e+01, 1.00505441e+01,
         5.08136362e+00],
        [7.56155692e-01, 3.55287814e-02, 9.19162366e-02, 7.73118774e-02,
         3.90874125e-02],
        [4.53693415e+00, 2.13172688e-01, 5.51497419e-01, 4.63871265e-01,
         2.34524475e-01],
        [2.26846708e+01, 1.06586344e+00, 2.75748710e+00, 2.31935632e+00,
         1.17262237e+00],
        [7.18347908e+01, 3.37523423e+00, 8.73204247e+00, 7.34462836e+00,
         3.71330418e+00],
        [1.51231138e+00, 7.10575627e-02, 1.83832473e-01, 1.54623755e-01,
         7.81748249e-02],
        [7.56155692e-01, 3.55287814e-02, 9.19162366e-02, 7.73118774e-02,
         3.90874125e-02],
        [2.44238289e+02, 1.14757964e+01, 2.96889444e+01, 2.49717364e+01,
         1.26252342e+01],
        [3.02462277e+00, 1.42115125e-01, 3.67664946e-01, 3.09247510e-01,
         1.56349650e-01],
        [6.80540123e+00, 3.19759032e-01, 8.27246129e-01, 6.95806897e-01,
         3.51786712e-01],
        [1.51231138e+00, 7.10575627e-02, 1.83832473e-01, 1.54623755e-01,
         7.81748249e-02],
        [1.89038923e+01, 8.88219534e-01, 2.29790591e+00, 1.93279694e+00,
         9.77185312e-01],
        [3.78077846e+00, 1.77643907e-01, 4.59581183e-01, 3.86559387e-01,
```

```
 1.95437062e-01],
[2.04162037e+02, 9.59277097e+00, 2.48173839e+01, 2.08742069e+01,
 1.05536014e+01],
[7.56155692e-01, 3.55287814e-02, 9.19162366e-02, 7.73118774e-02,
 3.90874125e-02],
[3.78077846e+00, 1.77643907e-01, 4.59581183e-01, 3.86559387e-01,
 1.95437062e-01],
[4.00762517e+01, 1.88302541e+00, 4.87156054e+00, 4.09752950e+00,
 2.07163286e+00],
[2.19285151e+01, 1.03033466e+00, 2.66557086e+00, 2.24204445e+00,
 1.13353496e+00],
[7.56155692e+01, 3.55287814e+00, 9.19162366e+00, 7.73118774e+00,
 3.90874125e+00],
[1.20984911e+01, 5.68460502e-01, 1.47065979e+00, 1.23699004e+00,
 6.25398600e-01],
[1.51231138e+00, 7.10575627e-02, 1.83832473e-01, 1.54623755e-01,
 7.81748249e-02],
[1.51231138e+00, 7.10575627e-02, 1.83832473e-01, 1.54623755e-01,
 7.81748249e-02],
[1.08659573e+03, 5.10548588e+01, 1.32083632e+02, 1.11097168e+02,
 5.61686117e+01],
[2.26846708e+00, 1.06586344e-01, 2.75748710e-01, 2.31935632e-01,
 1.17262237e-01],
[6.80540123e+00, 3.19759032e-01, 8.27246129e-01, 6.95806897e-01,
 3.51786712e-01],
[5.74678326e+01, 2.70018738e+00, 6.98563398e+00, 5.87570269e+00,
 2.97064335e+00],
[2.87339163e+01, 1.35009369e+00, 3.49281699e+00, 2.93785134e+00,
 1.48532167e+00],
[1.28546468e+01, 6.03989283e-01, 1.56257602e+00, 1.31430192e+00,
 6.64486012e-01],
[4.15885631e+01, 1.95408297e+00, 5.05539301e+00, 4.25215326e+00,
 2.14980769e+00],
[7.93963477e+01, 3.73052204e+00, 9.65120484e+00, 8.11774713e+00,
 4.10417831e+00],
[7.56155692e+00, 3.55287814e-01, 9.19162366e-01, 7.73118774e-01,
 3.90874125e-01],
[1.43669582e+01, 6.75046846e-01, 1.74640849e+00, 1.46892567e+00,
 7.42660837e-01],
[6.82808590e+02, 3.20824896e+01, 8.30003616e+01, 6.98126253e+01,
 3.52959335e+01],
[9.83002400e+00, 4.61874158e-01, 1.19491108e+00, 1.00505441e+00,
 5.08136362e-01],
[7.56155692e+00, 3.55287814e-01, 9.19162366e-01, 7.73118774e-01,
 3.90874125e-01],
[8.84702160e+01, 4.15686742e+00, 1.07541997e+01, 9.04548966e+00,
 4.57322726e+00],
[3.70516289e+01, 1.74091029e+00, 4.50389559e+00, 3.78828199e+00,
 1.91528321e+00],
[3.02462277e+01, 1.42115125e+00, 3.67664946e+00, 3.09247510e+00,
 1.56349650e+00],
[1.36108025e+01, 6.39518064e-01, 1.65449226e+00, 1.39161379e+00,
 7.03573425e-01],
[7.91695010e+02, 3.71986341e+01, 9.62362997e+01, 8.09455357e+01,
 4.09245209e+01],
[9.83002400e+00, 4.61874158e-01, 1.19491108e+00, 1.00505441e+00,
 5.08136362e-01],
```

```
[3.78077846e+01, 1.77643907e+00, 4.59581183e+00, 3.86559387e+00,
 1.95437062e+00],
[6.30633847e+02, 2.96310036e+01, 7.66581413e+01, 6.44781058e+01,
 3.25989020e+01],
[1.59548851e+02, 7.49657287e+00, 1.93943259e+01, 1.63128061e+01,
 8.24744403e+00],
[2.87339163e+01, 1.35009369e+00, 3.49281699e+00, 2.93785134e+00,
 1.48532167e+00],
[5.74678326e+01, 2.70018738e+00, 6.98563398e+00, 5.87570269e+00,
 2.97064335e+00],
[2.41969821e+01, 1.13692100e+00, 2.94131957e+00, 2.47398008e+00,
 1.25079720e+00],
[6.04924554e+02, 2.84230251e+01, 7.35329893e+01, 6.18495020e+01,
 3.12699300e+01],
[8.77140603e+01, 4.12133864e+00, 1.06622834e+01, 8.96817778e+00,
 4.53413985e+00],
[4.23447188e+01, 1.98961176e+00, 5.14730925e+00, 4.32946514e+00,
 2.18889510e+00],
[1.50474983e+02, 7.07022749e+00, 1.82913311e+01, 1.53850636e+01,
 7.77839508e+00],
[8.39332818e+01, 3.94369473e+00, 1.02027023e+01, 8.58161840e+00,
 4.33870278e+00],
[1.76789201e+03, 8.30662908e+01, 2.14900161e+02, 1.80755169e+02,
 9.13863704e+01],
[7.25909464e+01, 3.41076301e+00, 8.82395871e+00, 7.42194023e+00,
 3.75239160e+00],
[1.75428121e+02, 8.24267727e+00, 2.13245669e+01, 1.79363556e+01,
 9.06827969e+00],
[4.92257356e+02, 2.31292367e+01, 5.98374700e+01, 5.03300322e+01,
 2.54459055e+01],
[9.60317729e+01, 4.51215523e+00, 1.16733620e+01, 9.81860844e+00,
 4.96410138e+00],
[1.22497222e+02, 5.75566258e+00, 1.48904303e+01, 1.25245241e+01,
 6.33216082e+00],
[3.93957116e+02, 1.85104951e+01, 4.78883592e+01, 4.02794881e+01,
 2.03645419e+01],
[6.72978566e+01, 3.16206154e+00, 8.18054505e+00, 6.88075709e+00,
 3.47877971e+00],
[5.04355847e+02, 2.36976972e+01, 6.13081298e+01, 5.15670223e+01,
 2.60713041e+01],
[4.40838769e+02, 2.07132795e+01, 5.35871659e+01, 4.50728246e+01,
 2.27879615e+01],
[3.90251953e+03, 1.83364041e+02, 4.74379697e+02, 3.99006599e+02,
 2.01730136e+02],
[2.71459893e+02, 1.27548325e+01, 3.29979289e+01, 2.77549640e+01,
 1.40323811e+01],
[5.03599691e+02, 2.36621684e+01, 6.12162136e+01, 5.14897104e+01,
 2.60322167e+01],
[3.25903103e+02, 1.53129048e+01, 3.96158980e+01, 3.33214192e+01,
 1.68466748e+01],
[4.77134242e+02, 2.24186610e+01, 5.79991453e+01, 4.87837947e+01,
 2.46641573e+01],
[5.14185871e+02, 2.41595713e+01, 6.25030409e+01, 5.25720767e+01,
 2.65794405e+01],
[5.34602074e+02, 2.51188484e+01, 6.49847792e+01, 5.46594974e+01,
 2.76348006e+01],
[7.31202554e+02, 3.43563316e+01, 8.88830008e+01, 7.47605855e+01,
```

```
        3.77975279e+01],
       [1.00493091e+03, 4.72177504e+01, 1.22156678e+02, 1.02747485e+02,
        5.19471712e+01],
       [1.11835427e+03, 5.25470676e+01, 1.35944114e+02, 1.14344267e+02,
        5.78102830e+01],
       [7.15383777e+04, 3.36130695e+03, 8.69601131e+03, 7.31432210e+03,
        3.69798192e+03]]))
```

In [515]:
```
1 tab = pd.crosstab(abm3['host_response_rate'],abm3['Borough'])
```

In [516]:
```
1 st.chi2_contingency(tab)
```

Out[516]:
```
(3547.0451698390043,
 0.0,
 340,
 array([[3.66872021e+01, 5.35597488e+02, 5.91954469e+02, 1.80181860e+02,
         1.15789802e+01],
        [2.70554588e-02, 3.94983399e-01, 4.36544594e-01, 1.32877478e-01,
         8.53907098e-03],
        [5.41109175e-02, 7.89966797e-01, 8.73089188e-01, 2.65754956e-01,
         1.70781420e-02],
        [1.08221835e+00, 1.57993359e+01, 1.74617838e+01, 5.31509912e+00,
         3.41562839e-01],
        [2.16443670e-01, 3.15986719e+00, 3.49235675e+00, 1.06301982e+00,
         6.83125678e-02],
        [3.24665505e-01, 4.73980078e+00, 5.23853513e+00, 1.59452973e+00,
         1.02468852e-01],
        [2.70554588e-01, 3.94983399e+00, 4.36544594e+00, 1.32877478e+00,
         8.53907098e-02],
        [7.57552845e-01, 1.10595352e+01, 1.22232486e+01, 3.72056938e+00,
         2.39093987e-01],
```

Correlation Analysis

In [518]:
```
1 abm3.shape
```

Out[518]:
```
(121676, 35)
```

In [519]:
```
1 numerical=[i for i in abm3.columns if abm3[i].dtypes!='O']
2 len(numerical)
```

Out[519]:
```
28
```

In [524]:

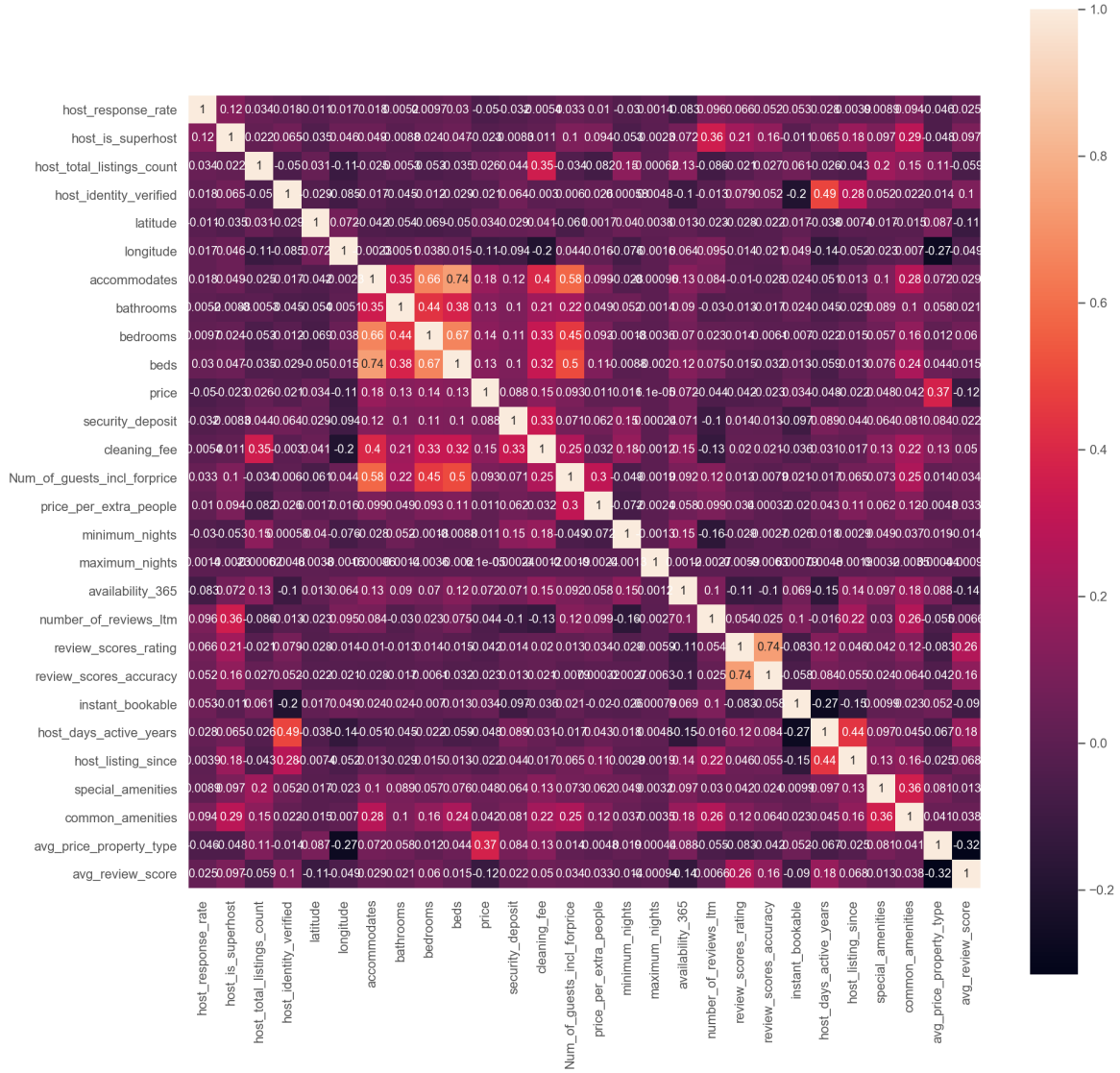
```
cor = abm3.corr()

#Correlation with output variable
cor_target = abs(cor["price"])

# Selecting highly correlated features
relevant_features = cor_target[cor_target>0.0]
relevant_features
```

Out[524]:

```
host_response_rate             0.049588
host_is_superhost              0.022927
host_total_listings_count      0.025971
host_identity_verified         0.021394
latitude                       0.033511
longitude                      0.105299
accommodates                   0.180508
bathrooms                      0.127249
bedrooms                       0.143716
beds                           0.131202
price                          1.000000
security_deposit               0.088196
cleaning_fee                   0.149144
Num_of_guests_incl_forprice    0.093140
price_per_extra_people         0.011131
minimum_nights                 0.010934
maximum_nights                 0.000061
availability_365               0.071956
number_of_reviews_ltm          0.044174
review_scores_rating           0.042355
review_scores_accuracy         0.023357
instant_bookable               0.034177
host_days_active_years         0.047673
host_listing_since             0.021741
special_amenities              0.048209
common_amenities               0.042290
avg_price_property_type        0.374809
avg_review_score               0.118327
Name: price, dtype: float64
```

In [526]:
```
top_corr_features = cor.index[abs(cor["price"])>0.0]
# plt.figure(figsize=(10,15))
sns.heatmap(abm3[top_corr_features].corr(), annot = True,
                cbar = True,square=True)
```

Out[526]: <matplotlib.axes._subplots.AxesSubplot at 0x224c2f9a688>



Since we can see except avg_price_property_type is the only feature with above 0.3 correlation we are dropping none features and moving with feature Engineering and Selection

In [528]:
```
# condition check for Multicollinearity  between 2 newly extracted featu
print(abm3[["avg_review_score","avg_price_property_type"]].corr())
```

```
                          avg_review_score  avg_price_property_type
avg_review_score                   1.0000                  -0.3157
avg_price_property_type           -0.3157                   1.0000
```

III. Feature Engineering

- Reducing Labels in the Object features
- Handling Outliers
- Transforming data

In [57]:
```
# Checking objects in Categorical variable.
print('host_response_time: \n',abm3['host_response_time'].value_counts()/
print('================================================')
print('cancellation_policy: \n',abm3.cancellation_policy.value_counts()/
```

```
host_response_time:
 within an hour        50.151221
unknown               22.207338
within a few hours    15.263487
within a day          10.160590
a few days or more     2.217364
Name: host_response_time, dtype: float64
================================================
cancellation_policy:
 strict      50.322989
flexible    25.689536
moderate    23.987475
Name: cancellation_policy, dtype: float64
```

In [58]:
```
# Reducig labdels in Host_response_time.
abm3.replace({'within an hour':'Hour','within a few hours':'One Day','wi

# Decreasig the value_counts in property_type

#With 10 categories we account for 95% of the listings
(abm3['property_type'].value_counts()/abm3['property_type'].value_counts
```

Out[58]: 97.0807718859923

In [59]:
```
Mod_prop_type=abm3['property_type'].value_counts()[5:len(abm1['property_t

def change_prop_type(label):
    if label in Mod_prop_type:
        label='Other'
    return label
# Mod_prop_type
abm3.loc[:,'property_type'] = abm3.loc[:,'property_type'].apply(change_pr
```

In [60]:
```
abm3.property_type.value_counts()
```

Out[60]:
```
Apartment      92006
House          11184
Other           9407
Townhouse       4756
Condominium     4323
Name: property_type, dtype: int64
```

In [61]:
```
numerical_types
```

Out[61]:
```
Index(['host_response_rate', 'host_is_superhost', 'host_total_listings_coun
t',
       'host_identity_verified', 'latitude', 'longitude', 'accommodates',
       'bathrooms', 'bedrooms', 'beds', 'price', 'security_deposit',
       'cleaning_fee', 'Num_of_guests_incl_forprice', 'price_per_extra_peop
le',
       'minimum_nights', 'maximum_nights', 'availability_365',
       'number_of_reviews_ltm', 'review_scores_rating',
       'review_scores_accuracy', 'instant_bookable', 'host_days_active_year
s',
       'host_listing_since', 'special_amenities', 'common_amenities',
       'avg_price_property_type', 'avg_review_score'],
      dtype='object')
```

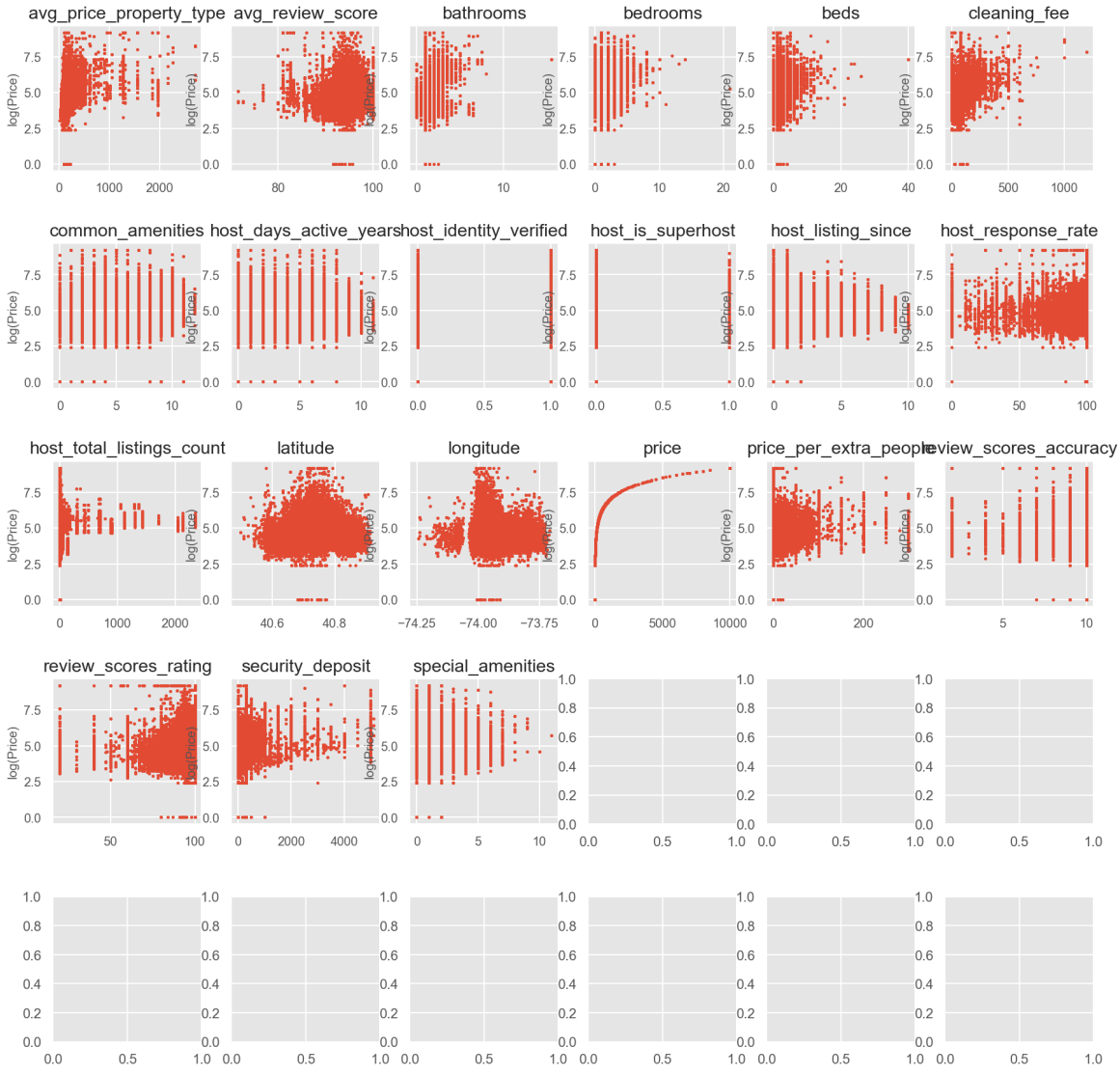
Handling outliers

In [286]:
```
cap_df = abm3.copy()
```

In [287]:

```
def features_plot(feat,df):
    plt.rcParams['figure.figsize']=(15,15)
    plt.style.use(style='ggplot')
    xxx,sub=plt.subplots(5,6)
    xxx.subplots_adjust(hspace=0.5)
    sub=sub.flatten()
    for i in range(len(feat)):
        sub[i].scatter(x=df[feat[i]], y=np.log1p(abm3["price"]),s=4)
        sub[i].set_title('{}'.format(feat[i],fontsize=10))
        sub[i].set_ylabel('log(Price)',fontsize=10)
        sub[i].tick_params(labelsize=10)
    plt.show()


Numerical_cols=[i for i in cap_df.columns if cap_df[i].dtypes=='float64']
len(Numerical_cols)
Numerical_cols = list(Numerical_cols)
# Numerical_cols.remove('price')
features_plot(sorted(Numerical_cols),cap_df)
```

Out[287]: 21



In [288]:

```
def cap_data(df):
    for col in df.columns:
#         print("capping the ",col)
        if (((df[col].dtype)=='float64') | ((df[col].dtype)=='int64')):
            percentiles = df[col].quantile([0.25,0.75]).values
            df[col][df[col] <= percentiles[0]] = percentiles[0]
            df[col][df[col] >= percentiles[1]] = percentiles[1]
            print(percentiles)
        else:
            df[col]=df[col]
    return df

# abm3 = cap_data(abm2)
```

In [289]:

```
# Capping outliers at lower and upper viscor
cap_df = cap_data(cap_df)
```

```
[100. 100.]
[0. 0.]
[1. 3.]
[0. 1.]
[40.6890075 40.76288  ]
[-73.98338 -73.93119]
[2. 4.]
[1. 1.]
[1. 1.]
[1. 2.]
[ 69. 179.]
[  0. 300.]
[30. 90.]
[1. 2.]
[ 0. 25.]
[1. 5.]
[  29. 1125.]
[  5. 282.]
[ 0. 16.]
[92.07142857 99.        ]
[10. 10.]
[0. 1.]
[2. 6.]
[0. 2.]
[0. 2.]
[3. 7.]
[101.87192049 191.80208333]
[93.21575342 95.04970812]
```

In [290]:

```
def features_plot(feat,df):
    plt.rcParams['figure.figsize']=(15,15)
    plt.style.use(style='ggplot')
    xxx,sub=plt.subplots(5,6)
    xxx.subplots_adjust(hspace=0.5)
    sub=sub.flatten()
    for i in range(len(feat)):
        sub[i].scatter(x=df[feat[i]], y=np.log1p(df["price"]),s=4)
        sub[i].set_title('{}'.format(feat[i],fontsize=10))
        sub[i].set_ylabel('log(Price)',fontsize=10)
        sub[i].tick_params(labelsize=10)
    plt.show()


Numerical_cols=[i for i in cap_df.columns if cap_df[i].dtypes=='float64']
len(Numerical_cols)
Numerical_cols = list(Numerical_cols)
# Numerical_cols.remove('price')
features_plot(sorted(Numerical_cols),cap_df)
```

Out[290]: 21

Feature transformation

In [291]:
```
# num_columns = ['host_is_superhost', 'host_total_listings_count',
#        'host_identity_verified', 'accommodates',
#        'bathrooms', 'bedrooms', 'beds', 'price']
# num_columns1 = ['price', 'availability_365','number_of_reviews_ltm', 'r
#                'instant_bookable', 'host_days_active_years','host_list

# num_columns2 = ['special_amenities',
#                'common_amenities', 'avg_price_property_type', 'avg_rev
#        'cleaning_fee', 'Num_of_guests_incl_forprice', 'price_per_extra_
```

In [292]:
```
# sns.pairplot(data=cap_df[num_columns],height=3)
# plt.show()
```

In [293]:
```
# sns.pairplot(data=cap_df[num_columns1],height=3)
# plt.show()
```

In [294]:
```
# sns.pairplot(data=cap_df[num_columns2],height=3)
# plt.show()
```

In [295]:
```
final_df = cap_df[numerical_types].drop(['latitude','longitude'],axis=1)
```

In [296]:

```
# Check the skew of all numerical features
skewed_feats = final_df.apply(lambda x: skew(x.dropna())).sort_values(as
print("\nSkew in numerical features: \n")
skewness = pd.DataFrame({'Skew' :skewed_feats})
skewness.head(10)
```

```
Skew in numerical features:
```

Out[296]:

	Skew
Num_of_guests_incl_forprice	0.827106
accommodates	0.599673
host_total_listings_count	0.583877
beds	0.566903
number_of_reviews_ltm	0.485560
instant_bookable	0.383710
host_identity_verified	0.324994
price_per_extra_people	0.324438
minimum_nights	0.276248
special_amenities	0.264433

In [297]:

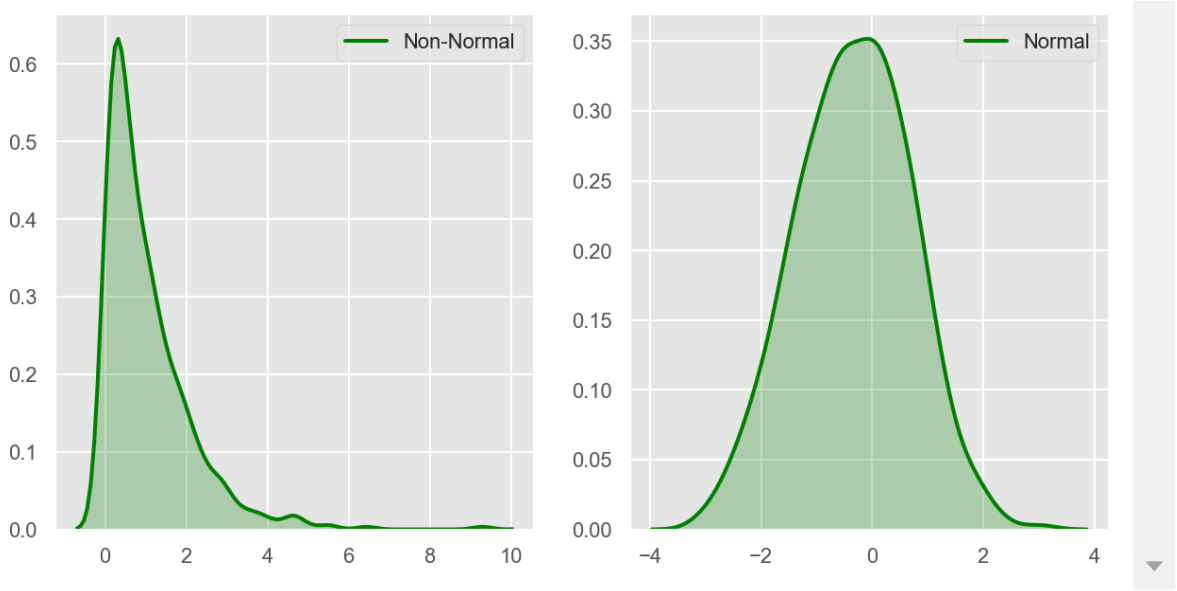
```
# import modules
import numpy as np
from scipy import stats

# plotting modules
import seaborn as sns
import matplotlib.pyplot as plt

# generate non-normal data (exponential)
final_df = np.random.exponential(size = 1000)

# transform training data & save lambda value
fitted_data, fitted_lambda = stats.boxcox(final_df)

# creating axes to draw plots
fig, ax = plt.subplots(1, 2)

# plotting the original data(non-normal) and
# fitted data (normal)
sns.distplot(final_df, hist = False, kde = True,
    kde_kws = {'shade': True, 'linewidth': 2},
    label = "Non-Normal", color ="green", ax = ax[0])

sns.distplot(fitted_data, hist = False, kde = True,
    kde_kws = {'shade': True, 'linewidth': 2},
    label = "Normal", color ="green", ax = ax[1])

# adding legends to the subplots
plt.legend(loc = "upper right")

# rescaling the subplots
fig.set_figheight(5)
fig.set_figwidth(10)

print(f"Lambda value used for Transformation: {fitted_lambda}")

```

Out[297]: <matplotlib.axes._subplots.AxesSubplot at 0x224befe7fc8>

Out[297]: <matplotlib.axes._subplots.AxesSubplot at 0x224b5d67408>

Out[297]: <matplotlib.legend.Legend at 0x224c11e5cc8>

```
Lambda value used for Transformation: 0.26426135584532734
```



In [298]:
```
# Check the skew of all numerical features
skewed_feats = pd.DataFrame(fitted_data).apply(lambda x: skew(x.dropna())
print("\nSkew in numerical features: \n")
skewness = pd.DataFrame({'Skew' :skewed_feats})
skewness.head(10)
```

```
Skew in numerical features:
```

Out[298]:

	Skew
0	-0.039392

Encoding Categorical Vars

In [299]:
```
categorical_types
```

Out[299]:
```
Index(['host_response_time', 'neighbourhood_cleansed', 'Borough',
       'property_type', 'room_type', 'bed_type', 'cancellation_policy'],
      dtype='object')
```

In [300]:
```
1 cap_df
```

Out[300]:

	host_response_time	host_response_rate	host_is_superhost	host_total_listings_count
0	Days	100.0	0.0	3.0
1	Hour	100.0	0.0	1.0
2	unknown	100.0	0.0	1.0
3	unknown	100.0	0.0	1.0
4	One Day	100.0	0.0	1.0
...	...	...	...	...
121671	Hour	100.0	0.0	3.0
121672	Hour	100.0	0.0	3.0
121673	unknown	100.0	0.0	1.0
121674	unknown	100.0	0.0	2.0
121675	a few days or more	100.0	0.0	1.0

121676 rows × 35 columns

In [301]:
```
1 # # # Bin into 5 categories
2 cap_df['host_response_rate'].value_counts(bins=5,sort=False)
```

Out[301]:
```
(99.899, 99.94]          0
(99.94, 99.98]           0
(99.98, 100.02]     121676
(100.02, 100.06]         0
(100.06, 100.1]          0
Name: host_response_rate, dtype: int64
```

In [302]:
```
1  # Bin into five categories
2  cap_df['host_response_rate_bins'] = pd.cut(cap_df.host_response_rate, bi
3
4  # Converting to string
5  cap_df['host_response_rate_bins'] = cap_df['host_response_rate_bins'].as
6
7  # Replace nulls with 'unknown'
8  #cap_df['host_response_rate_bins'].replace('nan', 'unknown', inplace=True
9
10 # Category counts
11 cap_df['host_response_rate_bins'].value_counts()
```

Out[302]:
```
100%    121676
Name: host_response_rate_bins, dtype: int64
```

In [303]:
```
dfe = pd.DataFrame(cap_df,columns = ['property_type','room_type','cancel
dfe.head()
```

Out[303]:

	property_type	room_type	cancellation_policy	host_response_rate_bins	bed_type
0	Apartment	Entire home/apt	strict	100%	Real Bed
1	Other	Entire home/apt	moderate	100%	Real Bed
2	Apartment	Entire home/apt	moderate	100%	Real Bed
3	Apartment	Private room	strict	100%	Futon
4	Apartment	Private room	strict	100%	Real Bed

In [254]:
```
cat_cols = dfe.select_dtypes(['object']).columns
```

In [304]:
```
for col in cat_cols:
    freqs = dfe[col].value_counts()
    k = freqs.index[freqs>20][:6]
    for cat in k:
        name = col+'_'+cat
        dfe[name]=(dfe[col]==cat).astype(int)
    del dfe[col]
    print(col)

print(dfe.dtypes)
```

```
property_type
room_type
cancellation_policy
host_response_rate_bins
bed_type
property_type_Apartment         int32
property_type_House             int32
property_type_Other             int32
property_type_Townhouse         int32
property_type_Condominium       int32
room_type_Entire home/apt       int32
room_type_Private room          int32
room_type_Shared room           int32
room_type_Hotel room            int32
cancellation_policy_strict      int32
cancellation_policy_flexible    int32
cancellation_policy_moderate    int32
host_response_rate_bins_100%    int32
bed_type_Real Bed               int32
bed_type_Futon                  int32
bed_type_Pull-out Sofa          int32
bed_type_Airbed                 int32
bed_type_Couch                  int32
dtype: object
```

In [305]:
```
abmen = pd.concat((dfe,cap_df),axis=1)
abmen.head(5)
```

Out[305]:

ɔerty_type_Townhouse	property_type_Condominium	room_type_Entire home/apt	room_type_Private room	room_t
0	0	1	0	
0	0	1	0	
0	0	1	0	
0	0	0	1	
0	0	0	1	

In [541]:
```
final_df = pd.DataFrame()
final_df = abmen.copy()
```

In [542]:
```
final_df.drop(['host_response_time', 'host_response_rate','host_response_
       'Borough', 'property_type', 'room_type', 'bed_type',
       'cancellation_policy'],axis=1,inplace=True)
```

In [543]:
```
final_df.columns
```

Out[543]:
```
Index(['property_type_Apartment', 'property_type_House', 'property_type_Oth
er',
       'property_type_Townhouse', 'property_type_Condominium',
       'room_type_Entire home/apt', 'room_type_Private room',
       'room_type_Shared room', 'room_type_Hotel room',
       'cancellation_policy_strict', 'cancellation_policy_flexible',
       'cancellation_policy_moderate', 'host_response_rate_bins_100%',
       'bed_type_Real Bed', 'bed_type_Futon', 'bed_type_Pull-out Sofa',
       'bed_type_Airbed', 'bed_type_Couch', 'host_is_superhost',
       'host_total_listings_count', 'host_identity_verified', 'latitude',
       'longitude', 'accommodates', 'bathrooms', 'bedrooms', 'beds', 'pric
e',
       'security_deposit', 'cleaning_fee', 'Num_of_guests_incl_forprice',
       'price_per_extra_people', 'minimum_nights', 'maximum_nights',
       'availability_365', 'number_of_reviews_ltm', 'review_scores_rating',
       'review_scores_accuracy', 'instant_bookable', 'host_days_active_year
s',
       'host_listing_since', 'special_amenities', 'common_amenities',
       'avg_price_property_type', 'avg_review_score'],
      dtype='object')
```

In [544]:
```
# seperating categorical and numerical dtypes
categorical_cols=final_df.select_dtypes(include=['object']).columns
print('categorical_types: \n',categorical_cols)

print('-------------')

numerical_cols=final_df._get_numeric_data().columns
print('numerical_types: \n',numerical_cols)
```

```
categorical_types:
 Index([], dtype='object')
-------------
numerical_types:
 Index(['property_type_Apartment', 'property_type_House', 'property_type_Ot
her',
       'property_type_Townhouse', 'property_type_Condominium',
       'room_type_Entire home/apt', 'room_type_Private room',
       'room_type_Shared room', 'room_type_Hotel room',
       'cancellation_policy_strict', 'cancellation_policy_flexible',
       'cancellation_policy_moderate', 'host_response_rate_bins_100%',
       'bed_type_Real Bed', 'bed_type_Futon', 'bed_type_Pull-out Sofa',
       'bed_type_Airbed', 'bed_type_Couch', 'host_is_superhost',
       'host_total_listings_count', 'host_identity_verified', 'latitude',
       'longitude', 'accommodates', 'bathrooms', 'bedrooms', 'beds', 'pric
e',
       'security_deposit', 'cleaning_fee', 'Num_of_guests_incl_forprice',
       'price_per_extra_people', 'minimum_nights', 'maximum_nights',
       'availability_365', 'number_of_reviews_ltm', 'review_scores_rating',
       'review_scores_accuracy', 'instant_bookable', 'host_days_active_year
s',
       'host_listing_since', 'special_amenities', 'common_amenities',
       'avg_price_property_type', 'avg_review_score'],
      dtype='object')
```

In [545]:
```
final_df.drop(['longitude','latitude'],axis=1,inplace=True)
```

In [ ]:
```

```

IV. Feature Selection

RFE

In [546]:
```
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import RFE
from sklearn.linear_model import RidgeCV, LassoCV, Ridge, Lasso
```

In [547]:
```
# Getting Data Ready
X = final_df.drop('price', axis=1)
y= final_df['price']
```

In [548]:
```
model = LinearRegression()
```

In [549]:
```
#Initializing RFE model
rfe = RFE(model, 40)
```

In [550]:
```
#Transforming data using RFE
X_rfe = rfe.fit_transform(X,y)
#Fitting the data to model
model.fit(X_rfe,y)
print(rfe.support_)
print(rfe.ranking_)
```

Out[550]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)

[ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True False
  True  True  True  True  True  True  True  True  True  True False  True
  True  True  True  True  True  True]
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 3 1 1
 1 1 1 1 1]
```

In [551]:
```
X.columns
```

Out[551]:
```
Index(['property_type_Apartment', 'property_type_House', 'property_type_Oth
er',
       'property_type_Townhouse', 'property_type_Condominium',
       'room_type_Entire home/apt', 'room_type_Private room',
       'room_type_Shared room', 'room_type_Hotel room',
       'cancellation_policy_strict', 'cancellation_policy_flexible',
       'cancellation_policy_moderate', 'host_response_rate_bins_100%',
       'bed_type_Real Bed', 'bed_type_Futon', 'bed_type_Pull-out Sofa',
       'bed_type_Airbed', 'bed_type_Couch', 'host_is_superhost',
       'host_total_listings_count', 'host_identity_verified', 'accommodate
s',
       'bathrooms', 'bedrooms', 'beds', 'security_deposit', 'cleaning_fee',
       'Num_of_guests_incl_forprice', 'price_per_extra_people',
       'minimum_nights', 'maximum_nights', 'availability_365',
       'number_of_reviews_ltm', 'review_scores_rating',
       'review_scores_accuracy', 'instant_bookable', 'host_days_active_year
s',
       'host_listing_since', 'special_amenities', 'common_amenities',
       'avg_price_property_type', 'avg_review_score'],
      dtype='object')
```

In [552]:
```
X.drop(['bedrooms', 'review_scores_accuracy'],axis=1,inplace=True)
```

In [553]:

```
#no of features
nof_list=np.arange(1,47)
high_score=0
#Variable to store the optimum features
nof=0
score_list =[]
for n in range(len(nof_list)):
    X_train, X_test, y_train, y_test = train_test_split(X,y, test_size =
    model = LinearRegression()
    rfe = RFE(model,nof_list[n])
    X_train_rfe = rfe.fit_transform(X_train,y_train)
    X_test_rfe = rfe.transform(X_test)
    model.fit(X_train_rfe,y_train)
    score = model.score(X_test_rfe,y_test)
    score_list.append(score)
    if(score>high_score):
        high_score = score
        nof = nof_list[n]
print("Optimum number of features: %d" %nof)
print("Score with %d features: %f" % (nof, high_score))
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=
```

```
False)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

Out[553]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)

Optimum number of features: 35
Score with 35 features: 0.642588
```

V. Model Building

In [554]:
```
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import RFE
from sklearn.linear_model import RidgeCV, LassoCV, Ridge, Lasso
```

In [555]:
```
## Raw linear regression model
X = final_df.drop('price', axis=1)
y= final_df['price']
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(X, y)

print(f'Coefficients: {lin_reg.coef_}')
print(f'Intercept: {lin_reg.intercept_}')
print(f'R^2 score: {lin_reg.score(X, y)}')
```

Out[555]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)

Coefficients: [ 1.66634075e-01 -1.35321530e+00  4.33469412e+00 -5.34274847e
+00
  2.19463558e+00  2.18225219e+01 -1.21685138e+01 -2.31322463e+01
  1.34782382e+01 -4.55588317e-01  5.02522878e-01 -4.69345615e-02
 -6.28830321e-13  3.41223451e+00 -3.43381547e+00 -2.87236412e-01
  1.45722007e+00 -1.14840269e+00  0.00000000e+00 -1.19998636e+00
 -9.54009798e-02  8.12466719e+00  8.88178420e-16  2.66453526e-15
  3.81697058e+00  1.49435553e-03  2.63588858e-01  3.43330718e+00
 -9.29797109e-02 -2.04087440e+00  1.29592883e-04  1.80106146e-02
 -3.32176245e-01  6.45602101e-01  0.00000000e+00  2.93625027e-01
  2.91233568e-01 -1.37856908e-01  1.68509553e+00  8.55414612e-01
  4.15381735e-01 -5.34732033e-01]
Intercept: -7.368685365013178
R^2 score: 0.6416348229921242
```

In [556]:

```
from sklearn.model_selection import train_test_split
X_train, X_test , y_train, y_test = train_test_split(X,y, test_size = 0.
print(X_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(85173, 42)
(36503, 42)
(36503,)
```

In [557]:

```
lin_reg = LinearRegression()
model = lin_reg.fit(X_train,y_train)
print(f'R^2 score for train: {lin_reg.score(X_train, y_train)}')
print(f'R^2 score for test: {lin_reg.score(X_test, y_test)}')
```

```
R^2 score for train: 0.6417653737540834
R^2 score for test: 0.6412206218294143
```

In [558]:
```
## Raw OLS Model
X = final_df.drop(['price'],axis=1)
y = final_df.price
X_constant = sm.add_constant(X)
lin_reg = sm.OLS(y,X_constant).fit()
lin_reg.summary()
```

Out[558]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.642
Model:	OLS	**Adj. R-squared:**	0.642
Method:	Least Squares	**F-statistic:**	6600.
Date:	Mon, 31 Aug 2020	**Prob (F-statistic):**	0.00
Time:	10:24:31	**Log-Likelihood:**	-5.7318e+05
No. Observations:	121676	**AIC:**	1.146e+06
Df Residuals:	121642	**BIC:**	1.147e+06
Df Model:	33		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
property_type_Apartment	0.1525	0.159	0.961	0.337	-0.159	0.464
property_type_House	-1.3674	0.251	-5.444	0.000	-1.860	-0.875
property_type_Other	4.3205	0.273	15.817	0.000	3.785	4.856
property_type_Townhouse	-5.3569	0.337	-15.873	0.000	-6.018	-4.695
property_type_Condominium	2.1805	0.355	6.134	0.000	1.484	2.877
room_type_Entire home/apt	21.8048	0.290	75.199	0.000	21.236	22.373
room_type_Private room	-12.1862	0.271	-44.918	0.000	-12.718	-11.654
room_type_Shared room	-23.1500	0.431	-53.688	0.000	-23.995	-22.305
room_type_Hotel room	13.4605	0.654	20.575	0.000	12.178	14.743
cancellation_policy_strict	-0.4792	0.114	-4.212	0.000	-0.702	-0.256
cancellation_policy_flexible	0.4789	0.133	3.596	0.000	0.218	0.740
cancellation_policy_moderate	-0.0706	0.130	-0.544	0.586	-0.325	0.184
host_response_rate_bins_100%	-0.0709	0.101	-0.704	0.481	-0.268	0.126
bed_type_Real Bed	3.3981	0.653	5.203	0.000	2.118	4.678
bed_type_Futon	-3.4480	1.072	-3.216	0.001	-5.549	-1.346
bed_type_Pull-out Sofa	-0.3014	1.123	-0.268	0.788	-2.502	1.899
bed_type_Airbed	1.4430	1.368	1.055	0.291	-1.238	4.124
bed_type_Couch	-1.1626	1.894	-0.614	0.539	-4.876	2.551
host_is_superhost	2.196e-15	8.52e-16	2.578	0.010	5.27e-16	3.87e-15
host_total_listings_count	-1.2000	0.099	-12.072	0.000	-1.395	-1.005

host_identity_verified	-0.0954	0.183	-0.521	0.603	-0.455	0.264
accommodates	8.1247	0.139	58.259	0.000	7.851	8.398
bathrooms	-0.0709	0.101	-0.704	0.481	-0.268	0.126
bedrooms	-0.0709	0.101	-0.704	0.481	-0.268	0.126
beds	3.8170	0.235	16.260	0.000	3.357	4.277
security_deposit	0.0015	0.001	2.358	0.018	0.000	0.003
cleaning_fee	0.2636	0.004	63.994	0.000	0.256	0.272
Num_of_guests_incl_forprice	3.4333	0.218	15.743	0.000	3.006	3.861
price_per_extra_people	-0.0930	0.009	-10.796	0.000	-0.110	-0.076
minimum_nights	-2.0409	0.057	-35.775	0.000	-2.153	-1.929
maximum_nights	0.0001	0.000	0.854	0.393	-0.000	0.000
availability_365	0.0180	0.001	23.275	0.000	0.016	0.020
number_of_reviews_ltm	-0.3322	0.014	-23.488	0.000	-0.360	-0.304
review_scores_rating	0.6456	0.030	21.234	0.000	0.586	0.705
review_scores_accuracy	-0.7086	1.007	-0.704	0.481	-2.682	1.264
instant_bookable	0.2936	0.167	1.761	0.078	-0.033	0.620
host_days_active_years	0.2912	0.060	4.886	0.000	0.174	0.408
host_listing_since	-0.1379	0.110	-1.252	0.210	-0.354	0.078
special_amenities	1.6851	0.101	16.616	0.000	1.486	1.884
common_amenities	0.8554	0.057	15.079	0.000	0.744	0.967
avg_price_property_type	0.4154	0.002	182.797	0.000	0.411	0.420
avg_review_score	-0.5347	0.113	-4.750	0.000	-0.755	-0.314

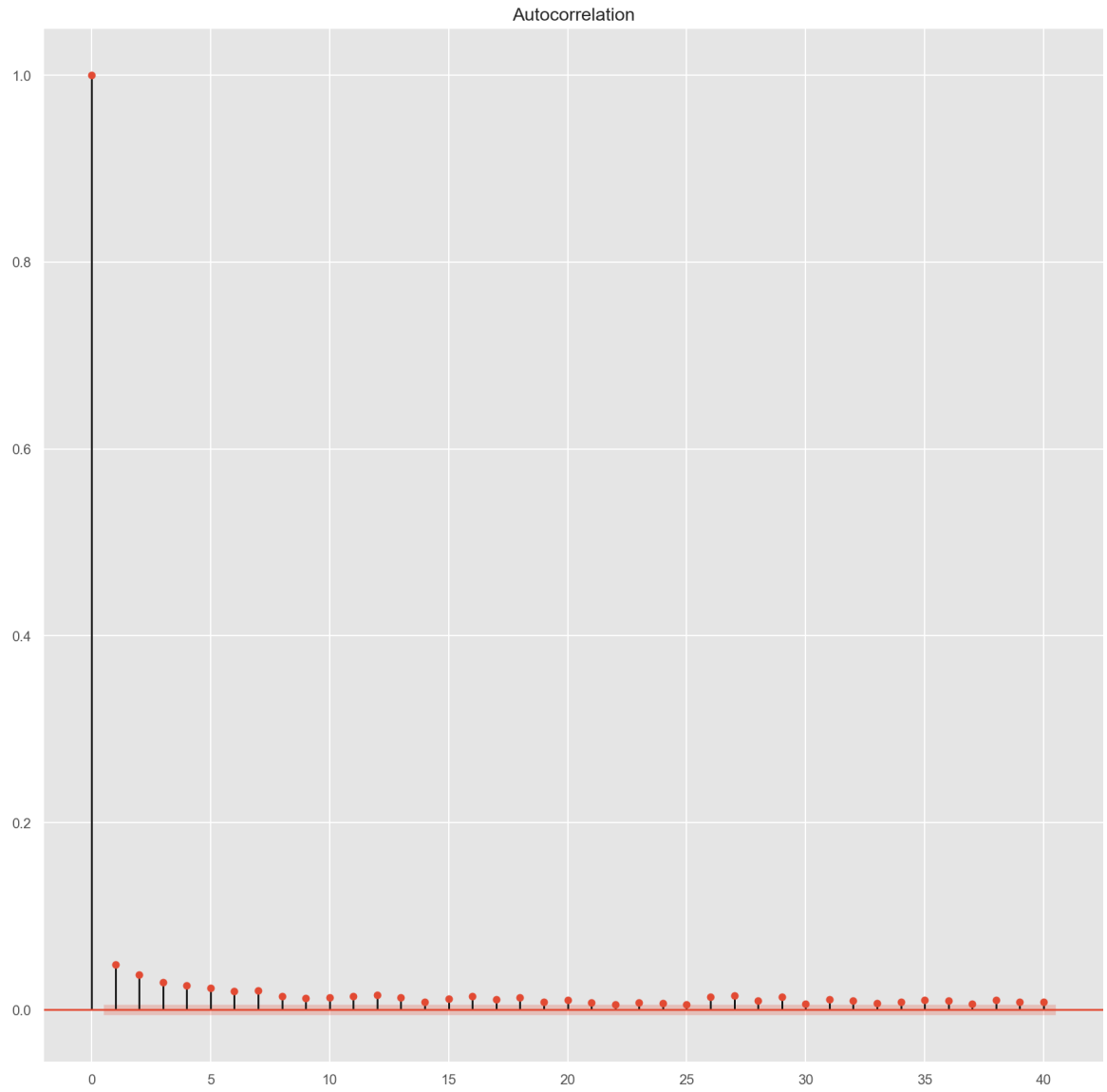
Omnibus:	1937.724	**Durbin-Watson:**	1.903
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	3397.666
Skew:	0.118	**Prob(JB):**	0.00
Kurtosis:	3.784	**Cond. No.**	3.40e+16

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 7.26e-23. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [559]:
```
1 ##### Assumption 1- No autocorrelation
```

In [560]:

```
import statsmodels.tsa.api as smt

acf = smt.graphics.plot_acf(lin_reg.resid, lags=40 , alpha=0.05)
acf.show()
```



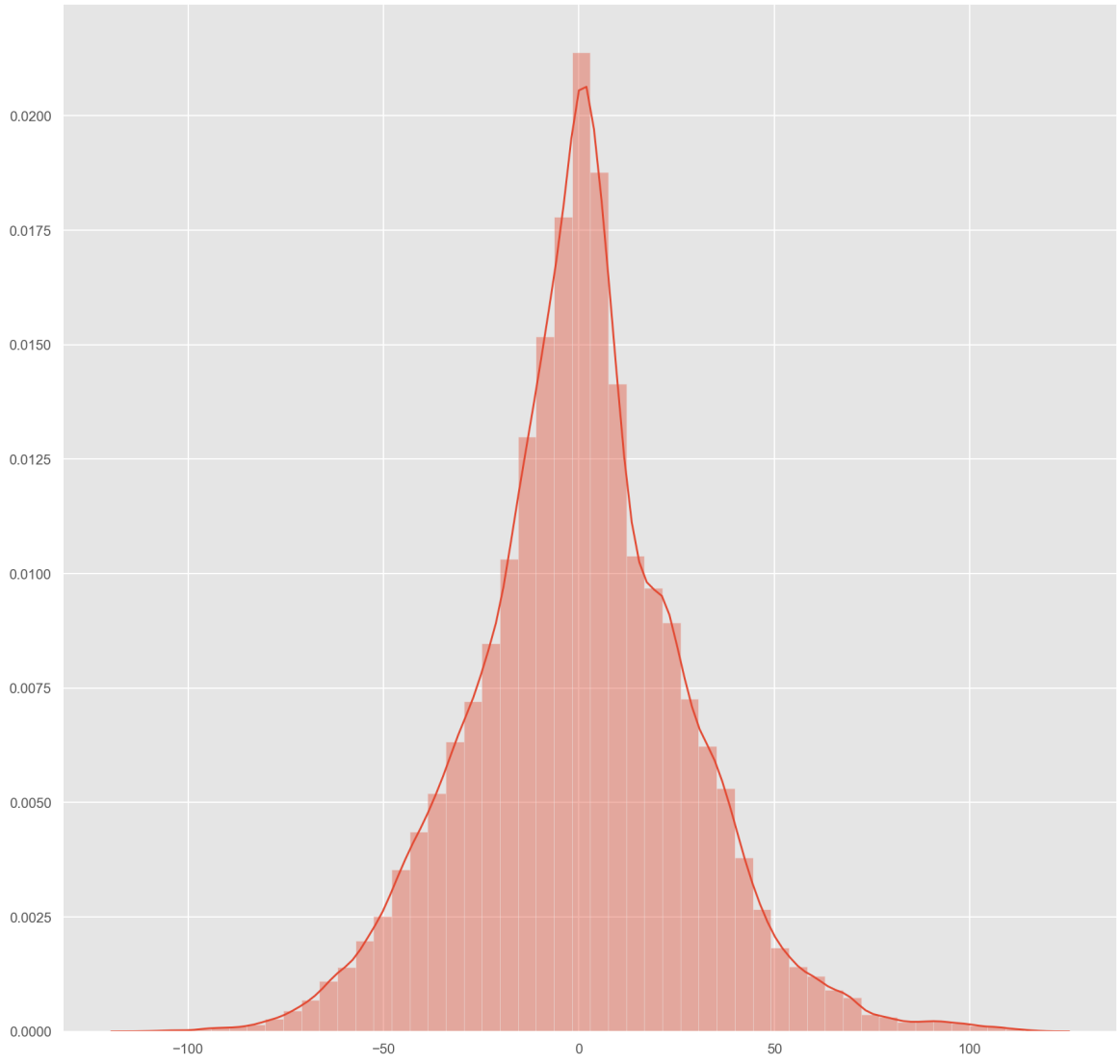
In [561]:

```
##### Assumption 2- Normality of Residuals
from scipy import stats
print(stats.jarque_bera(lin_reg.resid))
```

```
(3397.6663813313294, 0.0)
```

In [562]:
```
1 import seaborn as sns
2
3 sns.distplot(lin_reg.resid)
```

Out[562]: <matplotlib.axes._subplots.AxesSubplot at 0x224cd594208>



Asssumption 3 - Linearity of residuals

Here we have 2 options. Either we can plot the observed values Vs predicted values and plot the Residual Vs predicted values and see the linearity of residuals. OR We can go for rainbow test. Let's look both of them one by one.

Rainbow test

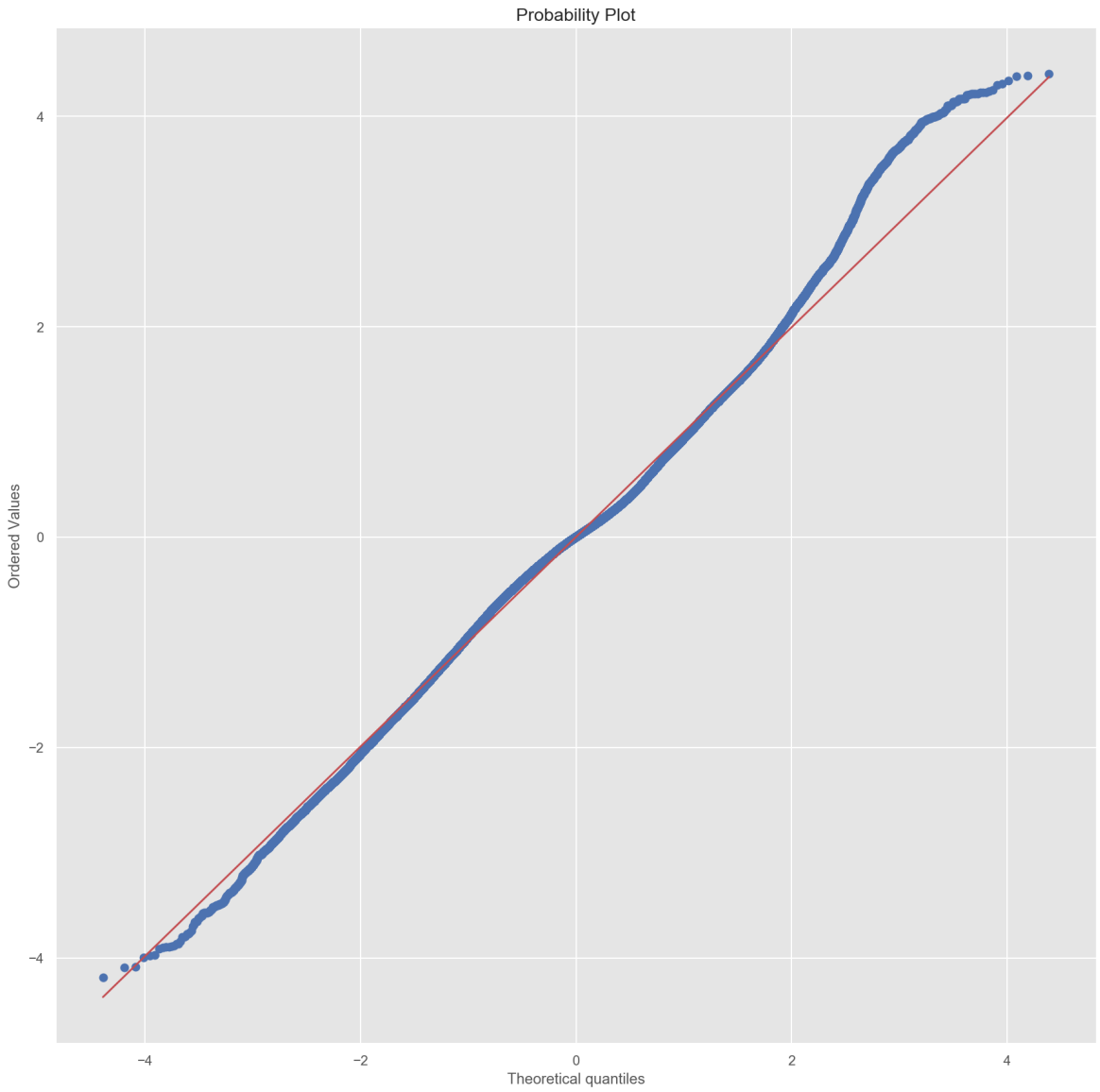
It is done to check the linearity of the residuals for a linear regression model. Linearity of residuals is preferred.

In [563]:
```
1 import statsmodels.api as sm
2 sm.stats.diagnostic.linear_rainbow(res=lin_reg, frac=0.5)
```

Out[563]:
```
(0.9870714499932058, 0.9457089018282858)
```

In [564]:
```
1 import scipy.stats as stats
2 import pylab
3 from statsmodels.graphics.gofplots import ProbPlot
4 st_residual = lin_reg.get_influence().resid_studentized_internal
5 stats.probplot(st_residual, dist="norm", plot = pylab)
6 plt.show()
```

Out[564]:
```
((array([-4.38888568, -4.1919612 , -4.08491805, ...,  4.08491805,
          4.1919612 ,  4.38888568]),
  array([-4.18707297, -4.09363663, -4.08426224, ...,  4.37757429,
          4.38147697,  4.40230132])),
 (0.9962365393285997, -5.5619335275836e-07, 0.9961951265369765))
```



In [565]:
```
lin_reg.resid.mean()
```

Out[565]:
```
7.088173588845645e-13
```

very close to 0 so linearity is present.

In [567]:
```
##### Assumption 4 -  Homoscedasticity_test(using goldfeld test) OR (Beus

# goldfeld test
from statsmodels.compat import lzip
import numpy as np
from statsmodels.compat import lzip
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.stats.api as sms

model = lin_reg
fitted_vals = model.predict()
resids = model.resid
resids_standardized = model.get_influence().resid_studentized_internal

name = ['F statistic', 'p-value']
test = sms.het_goldfeldquandt(model.resid, model.model.exog)
lzip(name, test)
```

Out[567]:
```
[('F statistic', 0.9644340487788085), ('p-value', 0.9999959918299126)]
```

In [566]:
```
##### Assumption 4 -  Homoscedasticity_test(using goldfeld test) OR (Beus

##### breuschpagan Test
import statsmodels.api as sm
from statsmodels.compat import lzip
name = ['Lagrange multiplier statistic', 'p-value',
        'f-value', 'f p-value']
test = sms.het_breuschpagan(lin_reg.resid, lin_reg.model.exog)
lzip(name, test)
```

Out[566]:
```
[('Lagrange multiplier statistic', 2995.7199915015685),
 ('p-value', 0.0),
 ('f-value', 93.04483445319447),
 ('f p-value', 0.0)]
```

so from above p- value we know that the data is heteroscedastic

Assumption 5- NO MULTI COLLINEARITY

In [333]:
```
X.shape
```

Out[333]:
```
(121676, 42)
```

In [334]:

```
##### Assumption 5- NO  MULTI COLLINEARITY
vif = [variance_inflation_factor(X_constant.values, i) for i in range(X_
pd.DataFrame({'vif': vif[0:]}, index=X.columns).T
```

Out[334]:

	property_type_Apartment	property_type_House	property_type_Other	property_type_Townho
vif	inf	inf	inf	

1 rows × 42 columns

#So, multicollinearity exists. Note : This vif column has be built with the help of X_constant and not the X_values. Because we built our model by adding Constant.

In [335]:
```
1 dict(pd.DataFrame({'vif': vif[0:]}, index=X.columns).T)
```

Out[335]:
```
{'property_type_Apartment': vif    inf
 Name: property_type_Apartment, dtype: float64,
 'property_type_House': vif    inf
 Name: property_type_House, dtype: float64,
 'property_type_Other': vif    inf
 Name: property_type_Other, dtype: float64,
 'property_type_Townhouse': vif    inf
 Name: property_type_Townhouse, dtype: float64,
 'property_type_Condominium': vif    inf
 Name: property_type_Condominium, dtype: float64,
 'room_type_Entire home/apt': vif    inf
 Name: room_type_Entire home/apt, dtype: float64,
 'room_type_Private room': vif    inf
 Name: room_type_Private room, dtype: float64,
 'room_type_Shared room': vif    inf
 Name: room_type_Shared room, dtype: float64,
 'room_type_Hotel room': vif    inf
 Name: room_type_Hotel room, dtype: float64,
 'cancellation_policy_strict': vif    inf
 Name: cancellation_policy_strict, dtype: float64,
 'cancellation_policy_flexible': vif    inf
 Name: cancellation_policy_flexible, dtype: float64,
 'cancellation_policy_moderate': vif    inf
 Name: cancellation_policy_moderate, dtype: float64,
 'host_response_rate_bins_100%': vif    0.0
 Name: host_response_rate_bins_100%, dtype: float64,
 'bed_type_Real Bed': vif    inf
 Name: bed_type_Real Bed, dtype: float64,
 'bed_type_Futon': vif    inf
 Name: bed_type_Futon, dtype: float64,
 'bed_type_Pull-out Sofa': vif    inf
 Name: bed_type_Pull-out Sofa, dtype: float64,
 'bed_type_Airbed': vif    inf
 Name: bed_type_Airbed, dtype: float64,
 'bed_type_Couch': vif    inf
 Name: bed_type_Couch, dtype: float64,
 'host_is_superhost': vif   NaN
 Name: host_is_superhost, dtype: float64,
 'host_total_listings_count': vif    1.272068
 Name: host_total_listings_count, dtype: float64,
 'host_identity_verified': vif    1.37643
 Name: host_identity_verified, dtype: float64,
 'accommodates': vif    2.635971
 Name: accommodates, dtype: float64,
 'bathrooms': vif    0.0
 Name: bathrooms, dtype: float64,
 'bedrooms': vif    0.0
 Name: bedrooms, dtype: float64,
 'beds': vif    2.145237
 Name: beds, dtype: float64,
 'security_deposit': vif    1.168179
 Name: security_deposit, dtype: float64,
 'cleaning_fee': vif    1.656475
 Name: cleaning_fee, dtype: float64,
```

```
'Num_of_guests_incl_forprice': vif    1.708242
Name: Num_of_guests_incl_forprice, dtype: float64,
'price_per_extra_people': vif    1.514801
Name: price_per_extra_people, dtype: float64,
'minimum_nights': vif    1.311486
Name: minimum_nights, dtype: float64,
'maximum_nights': vif    1.075634
Name: maximum_nights, dtype: float64,
'availability_365': vif    1.288483
Name: availability_365, dtype: float64,
'number_of_reviews_ltm': vif    1.497424
Name: number_of_reviews_ltm, dtype: float64,
'review_scores_rating': vif    1.163741
Name: review_scores_rating, dtype: float64,
'review_scores_accuracy': vif    0.0
Name: review_scores_accuracy, dtype: float64,
'instant_bookable': vif    1.12778
Name: instant_bookable, dtype: float64,
'host_days_active_years': vif    1.591821
Name: host_days_active_years, dtype: float64,
'host_listing_since': vif    1.355329
Name: host_listing_since, dtype: float64,
'special_amenities': vif    1.155556
Name: special_amenities, dtype: float64,
'common_amenities': vif    1.476945
Name: common_amenities, dtype: float64,
'avg_price_property_type': vif    1.258324
Name: avg_price_property_type, dtype: float64,
'avg_review_score': vif    1.156665
Name: avg_review_score, dtype: float64}
```

In [336]:

```
1 final_df.columns
```

Out[336]:

```
Index(['property_type_Apartment', 'property_type_House', 'property_type_Oth
er',
       'property_type_Townhouse', 'property_type_Condominium',
       'room_type_Entire home/apt', 'room_type_Private room',
       'room_type_Shared room', 'room_type_Hotel room',
       'cancellation_policy_strict', 'cancellation_policy_flexible',
       'cancellation_policy_moderate', 'host_response_rate_bins_100%',
       'bed_type_Real Bed', 'bed_type_Futon', 'bed_type_Pull-out Sofa',
       'bed_type_Airbed', 'bed_type_Couch', 'host_is_superhost',
       'host_total_listings_count', 'host_identity_verified', 'accommodate
s',
       'bathrooms', 'bedrooms', 'beds', 'price', 'security_deposit',
       'cleaning_fee', 'Num_of_guests_incl_forprice', 'price_per_extra_peop
le',
       'minimum_nights', 'maximum_nights', 'availability_365',
       'number_of_reviews_ltm', 'review_scores_rating',
       'review_scores_accuracy', 'instant_bookable', 'host_days_active_year
s',
       'host_listing_since', 'special_amenities', 'common_amenities',
       'avg_price_property_type', 'avg_review_score'],
      dtype='object')
```

In [361]:

```
##  removed like correlated variables
X = final_df[['property_type_Apartment', 'property_type_House', 'property
       'property_type_Townhouse', 'property_type_Condominium',
       'room_type_Entire home/apt', 'room_type_Private room',
       'room_type_Shared room', 'room_type_Hotel room',
       'cancellation_policy_strict', 'cancellation_policy_flexible', 'hos
       'bed_type_Real Bed', 'bed_type_Pull-out Sofa',
       'bed_type_Airbed', 'bed_type_Couch', 'host_is_superhost',
       'host_total_listings_count', 'host_identity_verified', 'accommoda
       'bathrooms', 'bedrooms', 'beds', 'security_deposit',
       'cleaning_fee', 'Num_of_guests_incl_forprice', 'price_per_extra_pe
       'minimum_nights', 'maximum_nights', 'availability_365',
       'number_of_reviews_ltm', 'review_scores_rating',
       'review_scores_accuracy', 'instant_bookable', 'host_days_active_ye
       'host_listing_since', 'special_amenities', 'common_amenities',
       'avg_price_property_type', 'avg_review_score']]
y = final_df['price']
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(X, y)

print(f'Coefficients: {lin_reg.coef_}')
print(f'Intercept: {lin_reg.intercept_}')
print(f'R^2 score: {lin_reg.score(X, y)}')
```

Out[361]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)


Coefficients: [ 1.66634075e-01 -1.35321530e+00  4.33469412e+00 -5.34274847e
+00
  2.19463558e+00  2.18225219e+01 -1.21685138e+01 -2.31322463e+01
  1.34782382e+01 -4.08653756e-01  5.49457440e-01 -1.26210153e-12
  6.84604998e+00  3.14657906e+00  4.89103554e+00  2.28541279e+00
  1.85451654e-12 -1.19998636e+00 -9.54009798e-02  8.12466719e+00
  2.66453526e-15 -7.10542736e-15  3.81697058e+00  1.49435553e-03
  2.63588858e-01  3.43330718e+00 -9.29797109e-02 -2.04087440e+00
  1.29592883e-04  1.80106146e-02 -3.32176245e-01  6.45602101e-01
 -4.44089210e-16  2.93625027e-01  2.91233568e-01 -1.37856908e-01
  1.68509553e+00  8.55414612e-01  4.15381735e-01 -5.34732033e-01]
Intercept: -10.849435401427215
R^2 score: 0.6416348229921242
```

In [362]:

```
import warnings
warnings.filterwarnings('ignore')
import statsmodels.api as sm

X_constant = sm.add_constant(X)
lin_reg = sm.OLS(y,X_constant).fit()
lin_reg.summary()
```

Out[362]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.642
Model:	OLS	**Adj. R-squared:**	0.642
Method:	Least Squares	**F-statistic:**	6600.
Date:	Sun, 30 Aug 2020	**Prob (F-statistic):**	0.00
Time:	22:34:01	**Log-Likelihood:**	-5.7318e+05
No. Observations:	121676	**AIC:**	1.146e+06
Df Residuals:	121642	**BIC:**	1.147e+06
Df Model:	33		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
property_type_Apartment	0.1457	0.159	0.918	0.359	-0.165	0.457
property_type_House	-1.3742	0.251	-5.471	0.000	-1.866	-0.882
property_type_Other	4.3137	0.273	15.792	0.000	3.778	4.849
property_type_Townhouse	-5.3637	0.337	-15.893	0.000	-6.025	-4.702
property_type_Condominium	2.1737	0.356	6.114	0.000	1.477	2.870
room_type_Entire home/apt	21.7963	0.290	75.176	0.000	21.228	22.365
room_type_Private room	-12.1947	0.271	-44.953	0.000	-12.726	-11.663
room_type_Shared room	-23.1585	0.431	-53.694	0.000	-24.004	-22.313
room_type_Hotel room	13.4520	0.654	20.562	0.000	12.170	14.734
cancellation_policy_strict	-0.4087	0.196	-2.083	0.037	-0.793	-0.024
cancellation_policy_flexible	0.5495	0.229	2.399	0.016	0.101	0.998
host_response_rate_bins_100%	-0.1049	0.101	-1.033	0.301	-0.304	0.094
bed_type_Real Bed	6.8460	1.114	6.145	0.000	4.663	9.030
bed_type_Pull-out Sofa	3.1466	1.624	1.938	0.053	-0.036	6.329
bed_type_Airbed	4.8910	1.912	2.558	0.011	1.144	8.638
bed_type_Couch	2.2854	2.552	0.896	0.371	-2.716	7.287
host_is_superhost	2.096e-13	3.33e-13	0.629	0.530	-4.44e-13	8.63e-13
host_total_listings_count	-1.2000	0.099	-12.072	0.000	-1.395	-1.005
host_identity_verified	-0.0954	0.183	-0.521	0.603	-0.455	0.264
accommodates	8.1247	0.139	58.259	0.000	7.851	8.398

bathrooms	-0.1049	0.101	-1.033	0.301	-0.304	0.094
bedrooms	-0.1049	0.101	-1.033	0.301	-0.304	0.094
beds	3.8170	0.235	16.260	0.000	3.357	4.277
security_deposit	0.0015	0.001	2.358	0.018	0.000	0.003
cleaning_fee	0.2636	0.004	63.994	0.000	0.256	0.272
Num_of_guests_incl_forprice	3.4333	0.218	15.743	0.000	3.006	3.861
price_per_extra_people	-0.0930	0.009	-10.796	0.000	-0.110	-0.076
minimum_nights	-2.0409	0.057	-35.775	0.000	-2.153	-1.929
maximum_nights	0.0001	0.000	0.854	0.393	-0.000	0.000
availability_365	0.0180	0.001	23.275	0.000	0.016	0.020
number_of_reviews_ltm	-0.3322	0.014	-23.488	0.000	-0.360	-0.304
review_scores_rating	0.6456	0.030	21.234	0.000	0.586	0.705
review_scores_accuracy	-1.0488	1.015	-1.033	0.301	-3.038	0.941
instant_bookable	0.2936	0.167	1.761	0.078	-0.033	0.620
host_days_active_years	0.2912	0.060	4.886	0.000	0.174	0.408
host_listing_since	-0.1379	0.110	-1.252	0.210	-0.354	0.078
special_amenities	1.6851	0.101	16.616	0.000	1.486	1.884
common_amenities	0.8554	0.057	15.079	0.000	0.744	0.967
avg_price_property_type	0.4154	0.002	182.797	0.000	0.411	0.420
avg_review_score	-0.5347	0.113	-4.750	0.000	-0.755	-0.314

Omnibus:	1937.724	**Durbin-Watson:**	1.903
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	3397.666
Skew:	0.118	**Prob(JB):**	0.00
Kurtosis:	3.784	**Cond. No.**	4.89e+16

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.51e-23. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [363]:

```
# remove 4 more parameters from the input
from statsmodels.stats.outliers_influence import variance_inflation_facto

vif = [variance_inflation_factor(X_constant.values, i) for i in range(X_
pd.DataFrame({'vif': vif[0:]}, index=X.columns).T
```

Out[363]:

	property_type_Apartment	property_type_House	property_type_Other	property_type_Townho
vif	inf	inf	inf	

1 rows × 40 columns

In [370]:

```
##  removed like correlated variables
X = final_df[['room_type_Entire home/apt', 'room_type_Private room',
       'room_type_Shared room', 'room_type_Hotel room',
       'cancellation_policy_strict', 'cancellation_policy_flexible', 'hos
       'bed_type_Real Bed', 'bed_type_Pull-out Sofa',
       'bed_type_Airbed', 'bed_type_Couch',
       'host_total_listings_count',  'accommodates',
       'bathrooms', 'bedrooms', 'beds',  'security_deposit',
       'cleaning_fee', 'Num_of_guests_incl_forprice', 'price_per_extra_pe
       'minimum_nights', 'maximum_nights', 'availability_365',
       'number_of_reviews_ltm', 'review_scores_rating',
       'review_scores_accuracy', 'instant_bookable', 'host_days_active_ye
       'host_listing_since', 'special_amenities', 'common_amenities',
       'avg_price_property_type', 'avg_review_score']]
y = final_df['price']
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(X, y)

print(f'Coefficients: {lin_reg.coef_}')
print(f'Intercept: {lin_reg.intercept_}')
print(f'R^2 score: {lin_reg.score(X, y)}')
```

Out[370]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)

Coefficients: [-6.05413312e+10 -6.05413313e+10 -6.05413313e+10 -6.05413312e
+10
 -4.44709794e-01  5.62847411e-01 -2.64129639e-02  6.71720574e+00
  3.18127758e+00  4.74089052e+00  2.26816292e+00 -1.25680028e+00
  8.08544106e+00  1.05209351e-02  2.90679932e-03  3.66906284e+00
  1.62088584e-03  2.63452723e-01  3.34112339e+00 -9.40152876e-02
 -2.08189350e+00  6.88803275e-05  1.84519570e-02 -3.47454267e-01
  6.35332622e-01 -5.96046448e-08  3.79916683e-01  2.78312288e-01
 -2.12873491e-01  1.70734767e+00  8.83113451e-01  4.21026634e-01
 -4.14658075e-01]
Intercept: 60541331236.79829
R^2 score: 0.6403415305465902
```

In [372]:

```
import warnings
warnings.filterwarnings('ignore')
import statsmodels.api as sm

X_constant = sm.add_constant(X)
lin_reg = sm.OLS(y,X_constant).fit()
lin_reg.summary()
```

Out[372]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.640
Model:	OLS	**Adj. R-squared:**	0.640
Method:	Least Squares	**F-statistic:**	7735.
Date:	Sun, 30 Aug 2020	**Prob (F-statistic):**	0.00
Time:	22:39:55	**Log-Likelihood:**	-5.7340e+05
No. Observations:	121676	**AIC:**	1.147e+06
Df Residuals:	121647	**BIC:**	1.147e+06
Df Model:	28		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
room_type_Entire home/apt	20.9409	0.282	74.345	0.000	20.389	21.493
room_type_Private room	-13.3676	0.262	-50.928	0.000	-13.882	-12.853
room_type_Shared room	-24.1107	0.427	-56.478	0.000	-24.947	-23.274
room_type_Hotel room	16.3418	0.626	26.103	0.000	15.115	17.569
cancellation_policy_strict	-0.4426	0.196	-2.256	0.024	-0.827	-0.058
cancellation_policy_flexible	0.5574	0.229	2.433	0.015	0.108	1.006
host_response_rate_bins_100%	-0.1956	0.099	-1.974	0.048	-0.390	-0.001
bed_type_Real Bed	6.7989	1.116	6.093	0.000	4.612	8.986
bed_type_Pull-out Sofa	3.2866	1.626	2.021	0.043	0.099	6.475
bed_type_Airbed	4.8637	1.915	2.540	0.011	1.110	8.617
bed_type_Couch	2.4383	2.556	0.954	0.340	-2.572	7.449
host_total_listings_count	-1.2602	0.099	-12.781	0.000	-1.453	-1.067
accommodates	8.0812	0.140	57.913	0.000	7.808	8.355
bathrooms	-0.1956	0.099	-1.974	0.048	-0.390	-0.001
bedrooms	-0.1956	0.099	-1.974	0.048	-0.390	-0.001
beds	3.6703	0.235	15.628	0.000	3.210	4.131
security_deposit	0.0016	0.001	2.531	0.011	0.000	0.003
cleaning_fee	0.2635	0.004	63.936	0.000	0.255	0.272
Num_of_guests_incl_forprice	3.3486	0.218	15.334	0.000	2.921	3.777
price_per_extra_people	-0.0942	0.009	-10.926	0.000	-0.111	-0.077

minimum_nights	-2.0816	0.057	-36.494	0.000	-2.193	-1.970
maximum_nights	7.059e-05	0.000	0.466	0.641	-0.000	0.000
availability_365	0.0185	0.001	23.928	0.000	0.017	0.020
number_of_reviews_ltm	-0.3477	0.014	-24.574	0.000	-0.375	-0.320
review_scores_rating	0.6353	0.030	20.882	0.000	0.576	0.695
review_scores_accuracy	-1.9556	0.991	-1.974	0.048	-3.898	-0.014
instant_bookable	0.3777	0.167	2.267	0.023	0.051	0.704
host_days_active_years	0.2770	0.054	5.105	0.000	0.171	0.383
host_listing_since	-0.2132	0.110	-1.946	0.052	-0.428	0.002
special_amenities	1.7065	0.101	16.826	0.000	1.508	1.905
common_amenities	0.8837	0.057	15.578	0.000	0.773	0.995
avg_price_property_type	0.4210	0.002	189.370	0.000	0.417	0.425
avg_review_score	-0.4143	0.110	-3.761	0.000	-0.630	-0.198

Omnibus:	2020.453	**Durbin-Watson:**	1.899
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	3525.922
Skew:	0.129	**Prob(JB):**	0.00
Kurtosis:	3.793	**Cond. No.**	1.29e+16

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 5.04e-22. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [373]:

```
# remove 4 more parameters from the input
from statsmodels.stats.outliers_influence import variance_inflation_facto

vif = [variance_inflation_factor(X_constant.values, i) for i in range(X_
pd.DataFrame({'vif': vif[0:]}, index=X.columns).T
```

Out[373]:

	room_type_Entire home/apt	room_type_Private room	room_type_Shared room	room_type_Hotel room	cancellation_po
vif	inf	inf	inf	inf	

1 rows × 33 columns

In [379]:

```
##  removed like correlated variables
X = final_df[['room_type_Entire home/apt',
       'room_type_Shared room',
       'cancellation_policy_strict', 'cancellation_policy_flexible', 'hos
       'bed_type_Real Bed', 'bed_type_Pull-out Sofa',
       'bed_type_Airbed', 'bed_type_Couch',
       'host_total_listings_count',  'accommodates',
       'bathrooms', 'bedrooms', 'beds',  'security_deposit',
       'cleaning_fee', 'Num_of_guests_incl_forprice', 'price_per_extra_pe
       'minimum_nights', 'maximum_nights', 'availability_365',
       'number_of_reviews_ltm', 'review_scores_rating',
       'review_scores_accuracy', 'instant_bookable', 'host_days_active_ye
       'host_listing_since', 'special_amenities', 'common_amenities',
       'avg_price_property_type', 'avg_review_score']]
y = final_df['price']
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(X, y)

print(f'Coefficients: {lin_reg.coef_}')
print(f'Intercept: {lin_reg.intercept_}')
print(f'R^2 score: {lin_reg.score(X, y)}')
```

Out[379]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)

Coefficients: [ 3.35190034e+01 -1.15956656e+01 -7.05196113e-01  9.42494858e
-01
 -1.45661261e-13  6.74194401e+00  3.30540299e+00  4.59615528e+00
  2.25253047e+00 -9.72529670e-01  8.20531026e+00 -1.06581410e-14
 -2.08721929e-14  3.68144535e+00  9.97247172e-04  2.56578829e-01
  4.15032092e+00 -1.12643646e-01 -2.22959803e+00 -6.10451986e-05
  2.03370794e-02 -3.69081996e-01  6.09746691e-01 -1.11022302e-15
  6.18873091e-01  2.06476090e-01 -2.53596625e-01  1.85684328e+00
  9.16073574e-01  4.33605207e-01 -5.79026827e-01]
Intercept: -17.397465131805276
R^2 score: 0.6365364821796806
```

In [380]:
```
import warnings
warnings.filterwarnings('ignore')
import statsmodels.api as sm

X_constant = sm.add_constant(X)
lin_reg = sm.OLS(y,X_constant).fit()
lin_reg.summary()
```

Out[380]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.637
Model:	OLS	**Adj. R-squared:**	0.636
Method:	Least Squares	**F-statistic:**	7890.
Date:	Sun, 30 Aug 2020	**Prob (F-statistic):**	0.00
Time:	22:44:39	**Log-Likelihood:**	-5.7404e+05
No. Observations:	121676	**AIC:**	1.148e+06
Df Residuals:	121648	**BIC:**	1.148e+06
Df Model:	27		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
room_type_Entire home/apt	33.5190	0.220	152.146	0.000	33.087	33.951
room_type_Shared room	-11.5957	0.510	-22.745	0.000	-12.595	-10.596
cancellation_policy_strict	-0.7052	0.197	-3.578	0.000	-1.091	-0.319
cancellation_policy_flexible	0.9425	0.230	4.097	0.000	0.492	1.393
host_response_rate_bins_100%	-0.1689	0.100	-1.692	0.091	-0.365	0.027
bed_type_Real Bed	6.7419	1.122	6.010	0.000	4.543	8.941
bed_type_Pull-out Sofa	3.3054	1.635	2.022	0.043	0.101	6.510
bed_type_Airbed	4.5962	1.925	2.387	0.017	0.823	8.370
bed_type_Couch	2.2525	2.570	0.876	0.381	-2.784	7.290
host_total_listings_count	-0.9725	0.099	-9.845	0.000	-1.166	-0.779
accommodates	8.2053	0.140	58.512	0.000	7.930	8.480
bathrooms	-0.1689	0.100	-1.692	0.091	-0.365	0.027
bedrooms	-0.1689	0.100	-1.692	0.091	-0.365	0.027
beds	3.6814	0.236	15.594	0.000	3.219	4.144
security_deposit	0.0010	0.001	1.564	0.118	-0.000	0.002
cleaning_fee	0.2566	0.004	62.005	0.000	0.248	0.265
Num_of_guests_incl_forprice	4.1503	0.218	19.005	0.000	3.722	4.578
price_per_extra_people	-0.1126	0.009	-13.018	0.000	-0.130	-0.096
minimum_nights	-2.2296	0.057	-38.986	0.000	-2.342	-2.118
maximum_nights	-6.105e-05	0.000	-0.401	0.688	-0.000	0.000

availability_365	0.0203	0.001	26.286	0.000	0.019	0.022
number_of_reviews_ltm	-0.3691	0.014	-25.975	0.000	-0.397	-0.341
review_scores_rating	0.6097	0.031	19.943	0.000	0.550	0.670
review_scores_accuracy	-1.6891	0.998	-1.692	0.091	-3.646	0.268
instant_bookable	0.6189	0.167	3.698	0.000	0.291	0.947
host_days_active_years	0.2065	0.055	3.788	0.000	0.100	0.313
host_listing_since	-0.2536	0.110	-2.303	0.021	-0.469	-0.038
special_amenities	1.8568	0.102	18.228	0.000	1.657	2.057
common_amenities	0.9161	0.057	16.066	0.000	0.804	1.028
avg_price_property_type	0.4336	0.002	196.456	0.000	0.429	0.438
avg_review_score	-0.5790	0.111	-5.233	0.000	-0.796	-0.362

Omnibus:	2014.047	**Durbin-Watson:**	1.893
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	3300.626
Skew:	0.154	**Prob(JB):**	0.00
Kurtosis:	3.746	**Cond. No.**	1.29e+16

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 5.04e-22. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [381]:

```
# remove 4 more parameters from the input
from statsmodels.stats.outliers_influence import variance_inflation_facto

vif = [variance_inflation_factor(X_constant.values, i) for i in range(X_
pd.DataFrame({'vif': vif[0:]}, index=X.columns).T
```

Out[381]:

...res_accuracy	instant_bookable	host_days_active_years	host_listing_since	special_amenities	co
0.0	1.120396	1.31275	1.337671	1.149674	

In [382]:

```
##  removed like correlated variables
X = final_df[['room_type_Entire home/apt',
       'room_type_Shared room',
       'cancellation_policy_strict', 'cancellation_policy_flexible', 'hos
       'bed_type_Real Bed', 'bed_type_Pull-out Sofa',
       'bed_type_Airbed', 'bed_type_Couch',
       'host_total_listings_count',  'accommodates',
       'bathrooms', 'bedrooms', 'beds',  'security_deposit',
       'cleaning_fee', 'Num_of_guests_incl_forprice', 'price_per_extra_pe
       'minimum_nights', 'availability_365',
       'number_of_reviews_ltm', 'review_scores_rating',
       'review_scores_accuracy', 'instant_bookable', 'host_days_active_ye
       'host_listing_since', 'special_amenities', 'common_amenities',
       'avg_price_property_type', 'avg_review_score']]
y = final_df['price']
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(X, y)

print(f'Coefficients: {lin_reg.coef_}')
print(f'Intercept: {lin_reg.intercept_}')
print(f'R^2 score: {lin_reg.score(X, y)}')
```

Out[382]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)

Coefficients: [ 3.35171479e+01 -1.15985883e+01 -7.07589214e-01  9.41048608e
-01
  1.79289916e-12  6.73776614e+00  3.30433825e+00  4.59761307e+00
  2.25386169e+00 -9.75075973e-01  8.20413651e+00  1.11022302e-14
  8.88178420e-16  3.68256373e+00  9.90581696e-04  2.56521173e-01
  4.15045692e+00 -1.12466225e-01 -2.22838196e+00  2.03073863e-02
 -3.68610953e-01  6.10335801e-01 -8.88178420e-16  6.18391632e-01
  2.06472672e-01 -2.51487210e-01  1.85775775e+00  9.16125631e-01
  4.33529505e-01 -5.79290163e-01]
Intercept: -17.444091583888493
R^2 score: 0.6365360019231117
```

In [383]:

```
import warnings
warnings.filterwarnings('ignore')
import statsmodels.api as sm

X_constant = sm.add_constant(X)
lin_reg = sm.OLS(y,X_constant).fit()
lin_reg.summary()
```

Out[383]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.637
Model:	OLS	**Adj. R-squared:**	0.636
Method:	Least Squares	**F-statistic:**	8194.
Date:	Sun, 30 Aug 2020	**Prob (F-statistic):**	0.00
Time:	22:46:27	**Log-Likelihood:**	-5.7404e+05
No. Observations:	121676	**AIC:**	1.148e+06
Df Residuals:	121649	**BIC:**	1.148e+06
Df Model:	26		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
room_type_Entire home/apt	33.5171	0.220	152.172	0.000	33.085	33.949
room_type_Shared room	-11.5986	0.510	-22.753	0.000	-12.598	-10.599
cancellation_policy_strict	-0.7076	0.197	-3.592	0.000	-1.094	-0.321
cancellation_policy_flexible	0.9410	0.230	4.092	0.000	0.490	1.392
host_response_rate_bins_100%	-0.1694	0.100	-1.697	0.090	-0.365	0.026
bed_type_Real Bed	6.7378	1.122	6.007	0.000	4.539	8.936
bed_type_Pull-out Sofa	3.3043	1.635	2.021	0.043	0.100	6.509
bed_type_Airbed	4.5976	1.925	2.388	0.017	0.824	8.371
bed_type_Couch	2.2539	2.570	0.877	0.380	-2.783	7.291
host_total_listings_count	-0.9751	0.099	-9.891	0.000	-1.168	-0.782
accommodates	8.2041	0.140	58.517	0.000	7.929	8.479
bathrooms	-0.1694	0.100	-1.697	0.090	-0.365	0.026
bedrooms	-0.1694	0.100	-1.697	0.090	-0.365	0.026
beds	3.6826	0.236	15.599	0.000	3.220	4.145
security_deposit	0.0010	0.001	1.554	0.120	-0.000	0.002
cleaning_fee	0.2565	0.004	62.029	0.000	0.248	0.265
Num_of_guests_incl_forprice	4.1505	0.218	19.006	0.000	3.722	4.578
price_per_extra_people	-0.1125	0.009	-13.014	0.000	-0.129	-0.096
minimum_nights	-2.2284	0.057	-39.019	0.000	-2.340	-2.116
availability_365	0.0203	0.001	26.368	0.000	0.019	0.022

number_of_reviews_ltm	-0.3686	0.014	-26.031	0.000	-0.396	-0.341
review_scores_rating	0.6103	0.031	19.986	0.000	0.550	0.670
review_scores_accuracy	-1.6936	0.998	-1.697	0.090	-3.650	0.263
instant_bookable	0.6184	0.167	3.695	0.000	0.290	0.946
host_days_active_years	0.2065	0.055	3.788	0.000	0.100	0.313
host_listing_since	-0.2515	0.110	-2.286	0.022	-0.467	-0.036
special_amenities	1.8578	0.102	18.242	0.000	1.658	2.057
common_amenities	0.9161	0.057	16.067	0.000	0.804	1.028
avg_price_property_type	0.4335	0.002	197.145	0.000	0.429	0.438
avg_review_score	-0.5793	0.111	-5.236	0.000	-0.796	-0.362

Omnibus:	2015.677	**Durbin-Watson:**	1.893
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	3302.774
Skew:	0.154	**Prob(JB):**	0.00
Kurtosis:	3.746	**Cond. No.**	1.38e+16

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 6.63e-23. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [384]:

```
# remove 4 more parameters from the input
from statsmodels.stats.outliers_influence import variance_inflation_facto

vif = [variance_inflation_factor(X_constant.values, i) for i in range(X_
pd.DataFrame({'vif': vif[0:]}, index=X.columns).T
```

Out[384]:

›res_accuracy	instant_bookable	host_days_active_years	host_listing_since	special_amenities	cc
0.0	1.120338	1.31275	1.334618	1.149098	

In [385]:
```
1 dict(pd.DataFrame({'vif': vif[0:]}, index=X.columns).T)
```

Out[385]:
```
{'room_type_Entire home/apt': vif    2.007273
 Name: room_type_Entire home/apt, dtype: float64,
 'room_type_Shared room': vif    1.071598
 Name: room_type_Shared room, dtype: float64,
 'cancellation_policy_strict': vif    1.609325
 Name: cancellation_policy_strict, dtype: float64,
 'cancellation_policy_flexible': vif    1.675005
 Name: cancellation_policy_flexible, dtype: float64,
 'host_response_rate_bins_100%': vif    0.0
 Name: host_response_rate_bins_100%, dtype: float64,
 'bed_type_Real Bed': vif    2.608384
 Name: bed_type_Real Bed, dtype: float64,
 'bed_type_Pull-out Sofa': vif    1.861815
 Name: bed_type_Pull-out Sofa, dtype: float64,
 'bed_type_Airbed': vif    1.502182
 Name: bed_type_Airbed, dtype: float64,
 'bed_type_Couch': vif    1.241129
 Name: bed_type_Couch, dtype: float64,
 'host_total_listings_count': vif    1.233663
 Name: host_total_listings_count, dtype: float64,
 'accommodates': vif    2.626958
 Name: accommodates, dtype: float64,
 'bathrooms': vif    0.0
 Name: bathrooms, dtype: float64,
 'bedrooms': vif    0.0
 Name: bedrooms, dtype: float64,
 'beds': vif    2.139234
 Name: beds, dtype: float64,
 'security_deposit': vif    1.16528
 Name: security_deposit, dtype: float64,
 'cleaning_fee': vif    1.646465
 Name: cleaning_fee, dtype: float64,
 'Num_of_guests_incl_forprice': vif    1.688803
 Name: Num_of_guests_incl_forprice, dtype: float64,
 'price_per_extra_people': vif    1.503874
 Name: price_per_extra_people, dtype: float64,
 'minimum_nights': vif    1.295996
 Name: minimum_nights, dtype: float64,
 'availability_365': vif    1.25846
 Name: availability_365, dtype: float64,
 'number_of_reviews_ltm': vif    1.480233
 Name: number_of_reviews_ltm, dtype: float64,
 'review_scores_rating': vif    1.157686
 Name: review_scores_rating, dtype: float64,
 'review_scores_accuracy': vif    0.0
 Name: review_scores_accuracy, dtype: float64,
 'instant_bookable': vif    1.120338
 Name: instant_bookable, dtype: float64,
 'host_days_active_years': vif    1.31275
 Name: host_days_active_years, dtype: float64,
 'host_listing_since': vif    1.334618
 Name: host_listing_since, dtype: float64,
 'special_amenities': vif    1.149098
 Name: special_amenities, dtype: float64,
```

```
'common_amenities': vif    1.471335
Name: common_amenities, dtype: float64,
'avg_price_property_type': vif    1.161955
Name: avg_price_property_type, dtype: float64,
'avg_review_score': vif    1.101564
Name: avg_review_score, dtype: float64}
```

Linear Regression

In [471]:

```
##  removed like correlated variables
X = final_df[['room_type_Entire home/apt',
       'room_type_Shared room',
       'cancellation_policy_strict', 'cancellation_policy_flexible', 'hos
       'bed_type_Real Bed', 'bed_type_Pull-out Sofa',
       'bed_type_Airbed', 'bed_type_Couch',
       'host_total_listings_count',  'accommodates',
       'bathrooms', 'bedrooms', 'beds',  'security_deposit',
       'cleaning_fee', 'Num_of_guests_incl_forprice', 'price_per_extra_pe
       'minimum_nights', 'availability_365',
       'number_of_reviews_ltm', 'review_scores_rating',
       'review_scores_accuracy', 'instant_bookable', 'host_days_active_ye
       'host_listing_since', 'special_amenities', 'common_amenities',
       'avg_price_property_type', 'avg_review_score']]
y = final_df['price']
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(X, y)

print(f'Coefficients: {lin_reg.coef_}')
print(f'Intercept: {lin_reg.intercept_}')
print(f'R^2 score: {lin_reg.score(X, y)}')
```

Out[471]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)

Coefficients: [ 3.35171479e+01 -1.15985883e+01 -7.07589214e-01  9.41048608e
-01
  1.79289916e-12  6.73776614e+00  3.30433825e+00  4.59761307e+00
  2.25386169e+00 -9.75075973e-01  8.20413651e+00  1.11022302e-14
  8.88178420e-16  3.68256373e+00  9.90581696e-04  2.56521173e-01
  4.15045692e+00 -1.12466225e-01 -2.22838196e+00  2.03073863e-02
 -3.68610953e-01  6.10335801e-01 -8.88178420e-16  6.18391632e-01
  2.06472672e-01 -2.51487210e-01  1.85775775e+00  9.16125631e-01
  4.33529505e-01 -5.79290163e-01]
Intercept: -17.444091583888493
R^2 score: 0.6365360019231117
```

Finally let's check for overfit and underfit condition

In [472]:
```
from sklearn.model_selection import train_test_split
X_train, X_test , y_train, y_test = train_test_split(X,y, test_size = 0.
print(X_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(85173, 30)
(36503, 30)
(36503,)
```

In [473]:
```
lin_reg = LinearRegression()
model = lin_reg.fit(X_train,y_train)
print(f'R^2 score for train: {lin_reg.score(X_train, y_train)}')
print(f'R^2 score for test: {lin_reg.score(X_test, y_test)}')
```

```
R^2 score for train: 0.6369799616222565
R^2 score for test: 0.635412119637978
```

Apply StandardScaler train and test

In [474]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaler = scaler.fit_transform(X_train)
X_test_scaler = scaler.fit_transform(X_test)
```

In [ ]:
```
# modeling
# libraries
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import Ridge, Lasso, ElasticNet, RidgeCV, Lass
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import KFold, cross_val_score, train_test_s
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor

```

Linear reg

In [475]:

```
# Getting Data Ready
##  removed like correlated variables
X = final_df[['room_type_Entire home/apt',
       'room_type_Shared room',
       'cancellation_policy_strict', 'cancellation_policy_flexible', 'hos
       'bed_type_Real Bed', 'bed_type_Pull-out Sofa',
       'bed_type_Airbed', 'bed_type_Couch',
       'host_total_listings_count',  'accommodates',
       'bathrooms', 'bedrooms', 'beds',  'security_deposit',
       'cleaning_fee', 'Num_of_guests_incl_forprice', 'price_per_extra_pe
       'minimum_nights', 'availability_365',
       'number_of_reviews_ltm', 'review_scores_rating',
       'review_scores_accuracy', 'instant_bookable', 'host_days_active_ye
       'host_listing_since', 'special_amenities', 'common_amenities',
       'avg_price_property_type', 'avg_review_score']]
y = final_df['price']
```

In [476]:

```
from sklearn.model_selection import train_test_split
train_x, test_x , train_y, test_y = train_test_split(X,y, test_size = 0.
print(train_x.shape)
print(test_x.shape)
print(test_y.shape)
```

```
(85173, 30)
(36503, 30)
(36503,)
```

In [477]:

```
model = LinearRegression()

# fit the model with the training data
model.fit(train_x,train_y)

# coefficeints of the trained model
print('\nCoefficient of model :', model.coef_)

# intercept of the model
print('\nIntercept of model',model.intercept_)

# predict the target on the test dataset
predict_train = model.predict(train_x)

# Root Mean Squared Error on training dataset
rmse_train = mean_squared_error(train_y,predict_train)**(0.5)
print('\nRMSE on train dataset : ', rmse_train)

# predict the target on the testing dataset
predict_test = model.predict(test_x)

# Root Mean Squared Error on testing dataset
rmse_test = mean_squared_error(test_y,predict_test)**(0.5)
print('\nRMSE on test dataset : ', rmse_test)
```

Out[477]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)


Coefficient of model : [ 3.34268165e+01 -1.15606951e+01 -6.37437868e-01  7.
66511357e-01
 -1.40565337e-12  7.72859112e+00  4.31581045e+00  5.25984408e+00
  2.35642894e+00 -9.95199014e-01  8.14868303e+00  8.88178420e-16
  1.64313008e-14  3.63705589e+00  5.70691657e-04  2.56699175e-01
  4.36941663e+00 -1.15721416e-01 -2.21782150e+00  2.01175127e-02
 -3.78250297e-01  6.15684685e-01  7.77156117e-16  6.27757829e-01
  1.97299849e-01 -1.20831478e-01  1.77191557e+00  9.62279711e-01
  4.32416788e-01 -5.46506538e-01]

Intercept of model -21.9251934075521

RMSE on train dataset :  27.063829839471378

RMSE on test dataset :  27.122013206993465
```

In [478]:
```
# Feature selection
## Raw OLS Model
X = final_df.drop(['price'],axis=1)
y = final_df.price
feature_select = LassoCV(precompute=True)
feature_select.fit(X, y)
print("Best alpha using built-in LassoCV: %f" % feature_select.alpha_)
print("Best score using built-in LassoCV: %f" %feature_select.score(X,y)
coef = pd.Series(feature_select.coef_, index = X.columns)
print("Lasso picked " + str(sum(coef != 0)) + " variables and eliminated
imp_coef = coef.sort_values()
```

Out[478]:
```
LassoCV(alphas=None, copy_X=True, cv=None, eps=0.001, fit_intercept=True,
        max_iter=1000, n_alphas=100, n_jobs=None, normalize=False,
        positive=False, precompute=True, random_state=None, selection='cycl
ic',
        tol=0.0001, verbose=False)

Best alpha using built-in LassoCV: 2.301280
Best score using built-in LassoCV: 0.613272
Lasso picked 11 variables and eliminated the other 2 variables
```

In [479]:
```
imp_coef=imp_coef[coef!=0]
imp_coef
```

Out[479]:
```
minimum_nights               -0.547182
number_of_reviews_ltm        -0.090485
maximum_nights                0.000125
security_deposit              0.000806
availability_365              0.012112
cleaning_fee                  0.376398
review_scores_rating          0.398364
avg_price_property_type       0.448685
common_amenities              0.776025
accommodates                  9.876608
room_type_Entire home/apt    21.990275
dtype: float64
```

In [480]:
```
X_new = final_df[['minimum_nights','number_of_reviews_ltm','maximum_nigh
                  'review_scores_rating',
                  'avg_price_property_type','common_amenities','accommod
y_new= final_df['price']
```

In [481]:
```
from sklearn.model_selection import train_test_split
trainx, testx , trainy, testy = train_test_split(X_new,y_new, test_size =
print(trainx.shape)
print(testx.shape)
print(testy.shape)
```

```
(85173, 11)
(36503, 11)
(36503,)
```

In [482]:
```
#linear reg after feature selectio
# fit the model with the training data
model.fit(trainx,trainy)

# coefficeints of the trained model
print('\nCoefficient of model :', model.coef_)

# intercept of the model
print('\nIntercept of model',model.intercept_)

# predict the target on the test dataset
predict_train = model.predict(trainx)

# Root Mean Squared Error on training dataset
rmse_train = mean_squared_error(trainy,predict_train)**(0.5)
print('\nRMSE on train dataset : ', rmse_train)

# predict the target on the testing dataset
predict_test = model.predict(testx)

# Root Mean Squared Error on testing dataset
rmse_test = mean_squared_error(testy,predict_test)**(0.5)
print('\nRMSE on test dataset : ', rmse_test)

print(f'R^2 score for train: {model.score(trainx, trainy)}')
print(f'R^2 score for test: {model.score(testx, testy)}')
```

Out[482]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)


Coefficient of model : [-2.30879632e+00 -3.81413625e-01 -3.05348271e-04  1.
35980048e-03
  1.73106195e-02  2.66709566e-01  6.19660172e-01  4.29845930e-01
  1.27226360e+00  9.86366851e+00  3.50093708e+01]

Intercept of model -62.847245096374266

RMSE on train dataset :  27.262963919520352

RMSE on test dataset :  27.31931774145764
R^2 score for train: 0.6316181474691046
R^2 score for test: 0.6300882896047266
```

In [483]:
```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
```

In [454]:
```
# Getting Data Ready
##  removed like correlated variables
X = final_df[['room_type_Entire home/apt',
       'room_type_Shared room',
       'cancellation_policy_strict', 'cancellation_policy_flexible', 'hos
       'bed_type_Real Bed', 'bed_type_Pull-out Sofa',
       'bed_type_Airbed', 'bed_type_Couch',
       'host_total_listings_count',  'accommodates',
       'bathrooms', 'bedrooms', 'beds',  'security_deposit',
       'cleaning_fee', 'Num_of_guests_incl_forprice', 'price_per_extra_pe
       'minimum_nights', 'availability_365',
       'number_of_reviews_ltm', 'review_scores_rating',
       'review_scores_accuracy', 'instant_bookable', 'host_days_active_ye
       'host_listing_since', 'special_amenities', 'common_amenities',
       'avg_price_property_type', 'avg_review_score']]
y = final_df['price']
```

In [485]:
```
# Ridge basic model

train_x = X_train
train_y = y_train
test_x = X_test
test_y = y_test

rr = Ridge(alpha=0.01)
rr.fit(train_x, train_y)
pred_train_rr= rr.predict(train_x)
print('train_rmse: ',np.sqrt(mean_squared_error(train_y,pred_train_rr)))
print('train_r2 score: ',r2_score(train_y, pred_train_rr))
print('-------------------------')
pred_test_rr= rr.predict(test_x)
print('test_rmse: ',np.sqrt(mean_squared_error(test_y,pred_test_rr)))
print('test_r2 score: ',r2_score(test_y, pred_test_rr))
```

Out[485]:
```
Ridge(alpha=0.01, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)

train_rmse:  27.063829839500183
train_r2 score:  0.6369799616214836
-------------------------
test_rmse:  27.122013030869358
test_r2 score:  0.6354121243730774
```

In [486]:
```
# feature selected

rr = Ridge(alpha=0.01)
rr.fit(trainx, trainy)
pred_train_rr= rr.predict(trainx)
print('ytrain_rmse: ',np.sqrt(mean_squared_error(trainy,pred_train_rr)))
print('ytrain_r2 score: ',r2_score(trainy, pred_train_rr))
print('-------------------------')
pred_test_rr= rr.predict(testx)
print('test_rmse: ',np.sqrt(mean_squared_error(testy,pred_test_rr)))
print('test_r2 score: ',r2_score(testy, pred_test_rr))
```

Out[486]:
```
Ridge(alpha=0.01, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)

ytrain_rmse:  27.262963919522388
ytrain_r2 score:  0.6316181474690495
-------------------------
test_rmse:  27.31931774864737
test_r2 score:  0.6300882894100244
```

lasso Reg

In [489]:
```
# lasso
model_lasso = Lasso(alpha=0.01)
model_lasso.fit(train_x, train_y)
pred_train_lasso= model_lasso.predict(train_x)
print('train_rmse: ',np.sqrt(mean_squared_error(train_y,pred_train_lasso)
print('train_r2 score: ',r2_score(train_y, pred_train_lasso))
print('-------------------------')
pred_test_lasso= model_lasso.predict(test_x)
print('test_rmse: ',np.sqrt(mean_squared_error(test_y,pred_test_lasso)))
print('test_r2 score: ',r2_score(test_y, pred_test_lasso))
```

Out[489]:
```
Lasso(alpha=0.01, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)

train_rmse:  27.06526953410705
train_r2 score:  0.6369413379754045
-------------------------
test_rmse:  27.121916828518213
test_r2 score:  0.6354147107702295
```

In [490]:
```
# Feature Selected
model_lasso = Lasso(alpha=0.01)
model_lasso.fit(trainx, trainy)
pred_train_lasso= model_lasso.predict(trainx)
print('train_rmse: ',np.sqrt(mean_squared_error(trainy,pred_train_lasso)))
print('train_r2 score: ',r2_score(trainy, pred_train_lasso))
print('-------------------------')
pred_test_lasso= model_lasso.predict(testx)
print('test_rmse: ',np.sqrt(mean_squared_error(testy,pred_test_lasso)))
print('test_r2 score: ',r2_score(testy, pred_test_lasso))
```

Out[490]:
```
Lasso(alpha=0.01, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)

train_rmse:  27.26297648274075
train_r2 score:  0.6316178079562413
-------------------------
test_rmse:  27.319375560168037
test_r2 score:  0.6300867238379024
```

Elastic Net

In [491]:
```
#Elastic Net
model_enet = ElasticNet(alpha = 0.01)
model_enet.fit(trainx, trainy)
pred_train_enet= model_enet.predict(trainx)
print('train_rmse: ',np.sqrt(mean_squared_error(trainy,pred_train_enet)))
print('train_r2 score: ',r2_score(trainy, pred_train_enet))

pred_test_enet= model_enet.predict(testx)
print('test_rmse: ',np.sqrt(mean_squared_error(testy,pred_test_enet)))
print('test_r2 score: ',r2_score(testy, pred_test_enet))
```

Out[491]:
```
ElasticNet(alpha=0.01, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=Fals
e,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=Fa
lse)

train_rmse:  27.26657574369064
train_r2 score:  0.6315205338260843
test_rmse:  27.323278460597063
test_r2 score:  0.6299810231926983
```

Boosting techniques

In [492]:
```
from sklearn.datasets import make_regression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn.ensemble import GradientBoostingRegressor
# define dataset
X, y = make_regression(n_samples=1000, n_features=20, n_informative=15, r
# define the model
model = GradientBoostingRegressor()
# define the evaluation procedure
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate the model
n_scores = cross_val_score(model, X, y, scoring='neg_mean_absolute_error
# report performance
print('MAE: %.3f (%.3f)' % (np.mean(n_scores), np.std(n_scores)))
```

```
MAE: -62.457 (3.238)
```

1. Gradient Boosting

In [494]:
```
# gradient boosting ensemble for making predictions for regression
from sklearn.datasets import make_regression
from sklearn.ensemble import GradientBoostingRegressor
# define dataset
X, y = make_regression(n_samples=1000, n_features=20, n_informative=15, r
# define the model
model = GradientBoostingRegressor()
# fit the model on the whole dataset
model.fit(trainx, trainy)
# make a single prediction

yhat = model.predict(testx)
# summarize prediction

print('test_rmse: ',np.sqrt(mean_squared_error(testy,yhat)))
print('test_r2 score: ',r2_score(testy, yhat))
```

Out[494]:
```
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_ms
e',
                          init=None, learning_rate=0.1, loss='ls', max_dept
h=3,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=Non
e,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=100,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=Fa
lse)

test_rmse:  25.98189694722493
test_r2 score:  0.6654199102825641
```

In [495]:

```
# VIF selected
from sklearn.datasets import make_regression
from sklearn.ensemble import GradientBoostingRegressor
# define dataset
X, y = make_regression(n_samples=1000, n_features=20, n_informative=15, r
# define the model
model = GradientBoostingRegressor()
# fit the model on the whole dataset
model.fit(train_x, train_y)
# make a single prediction

yhat = model.predict(test_x)
# summarize prediction

print('test_rmse: ',np.sqrt(mean_squared_error(test_y,yhat)))
print('test_r2 score: ',r2_score(test_y, yhat))
```

Out[495]:

```
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_ms
e',
                          init=None, learning_rate=0.1, loss='ls', max_dept
h=3,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=Non
e,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=100,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=Fa
lse)

test_rmse:  25.492786939440514
test_r2 score:  0.6778983021183561
```

2. XG Boost

In [497]:
```
import xgboost
from xgboost import plot_importance
xgb_model = xgboost.XGBRegressor(
                 max_depth=3,
                 n_estimators=100,
                 seed=42)
xgb_model.fit(train_x,train_y)
yhat = xgb_model.predict(test_x)
# summarize prediction

print('test_rmse: ',np.sqrt(mean_squared_error(test_y,yhat)))
print('test_r2 score: ',r2_score(test_y, yhat))
```

Out[497]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=3,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=0, num_parallel_tree=1,
             objective='reg:squarederror', random_state=42, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, seed=42, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)

test_rmse:  24.90947071759183
test_r2 score:  0.6924700760810385
```

In [498]:
```
import xgboost
from xgboost import plot_importance
xgb_model = xgboost.XGBRegressor(
                 max_depth=3,
                 n_estimators=100,
                 seed=42)
xgb_model.fit(trainx,trainy)
yhat = xgb_model.predict(testx)
# summarize prediction

print('test_rmse: ',np.sqrt(mean_squared_error(testy,yhat)))
print('test_r2 score: ',r2_score(testy, yhat))
```

Out[498]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=3,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=0, num_parallel_tree=1,
             objective='reg:squarederror', random_state=42, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, seed=42, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)

test_rmse:  25.593941826859893
test_r2 score:  0.6753370440057858
```

In [ ]:
```
1
```