Airbnb - Boston

In [1]:
```
# Import Neccessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
import math
import time


# Import library for VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.linear_model import ElasticNet
from sklearn import preprocessing
from sklearn.metrics import *


import folium
from folium import plugins
from folium.plugins import HeatMap
```

In [2]:
```
ur = "http://data.insideairbnb.com/united-states/ma/boston/2021-12-17/data/
url1 = "http://data.insideairbnb.com/united-states/ma/boston/2021-12-17/visu
```

We have imported pandas ,numpy for basic data analysis. Seaborn and matplotlib for data visualization. Folium for maps.

In [3]:
```
# Loading Dataset
boston = pd.read_csv(ur,compression='gzip',low_memory=False)
boston2 = pd.read_csv(url1)
```

Data Exploration , Visualization and Processing

In [4]:
```
### Created a Subset of data
dat = boston[['host_id','accommodates','bedrooms','beds','amenities']].copy
boston3 = pd.merge(boston2,dat)
boston3.tail()
```

Out[4]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitu
107384	53772114	Entire Modern Apartment in Downtown Boston 2 R...	434990435	Luke	NaN	South End	42.3448
107385	53772114	Entire Modern Apartment in Downtown Boston 2 R...	434990435	Luke	NaN	South End	42.3448
107386	53772114	Entire Modern Apartment in Downtown Boston 2 R...	434990435	Luke	NaN	South End	42.3448
107387	53772114	Entire Modern Apartment in Downtown Boston 2 R...	434990435	Luke	NaN	South End	42.3448
107388	53756737	The Arbor Retreat	42715907	Emma	NaN	Jamaica Plain	42.3048

5 rows × 22 columns

In [5]:
```python
### Calculating Summary Statistics
boston3.describe()
```

Out[5]:

	id	host_id	neighbourhood_group	latitude	longitude	pr
count	1.073890e+05	1.073890e+05	0.0	107389.000000	107389.000000	107389.00
mean	4.461479e+07	1.758024e+08	NaN	42.342998	-71.084340	147.84
std	1.192542e+07	1.221205e+08	NaN	0.019351	0.033584	116.54
min	3.781000e+03	4.804000e+03	NaN	42.235330	-71.172520	0.00
25%	4.208014e+07	1.074344e+08	NaN	42.335150	-71.101850	50.00
50%	4.961078e+07	1.074344e+08	NaN	42.347970	-71.070840	149.00
75%	5.227869e+07	2.978601e+08	NaN	42.355180	-71.061640	212.00
max	5.383997e+07	4.349904e+08	NaN	42.392790	-70.997810	10000.00

In [6]:
```python
### Precise Summary of the Dataframe
boston3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 107389 entries, 0 to 107388
Data columns (total 22 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   id                              107389 non-null  int64  
 1   name                            107389 non-null  object 
 2   host_id                         107389 non-null  int64  
 3   host_name                       61820 non-null   object 
 4   neighbourhood_group             0 non-null       float64
 5   neighbourhood                   107389 non-null  object 
 6   latitude                        107389 non-null  float64
 7   longitude                       107389 non-null  float64
 8   room_type                       107389 non-null  object 
 9   price                           107389 non-null  int64  
 10  minimum_nights                  107389 non-null  int64  
 11  number_of_reviews               107389 non-null  int64  
 12  last_review                     29052 non-null   object 
 13  reviews_per_month               29052 non-null   float64
 14  calculated_host_listings_count  107389 non-null  int64  
 15  availability_365                107389 non-null  int64  
 16  number_of_reviews_ltm           107389 non-null  int64  
 17  license                         23450 non-null   object 
 18  accommodates                    107389 non-null  int64  
 19  bedrooms                        88219 non-null   float64
 20  beds                            82280 non-null   float64
 21  amenities                       107389 non-null  object 
dtypes: float64(6), int64(9), object(7)
memory usage: 18.8+ MB
```

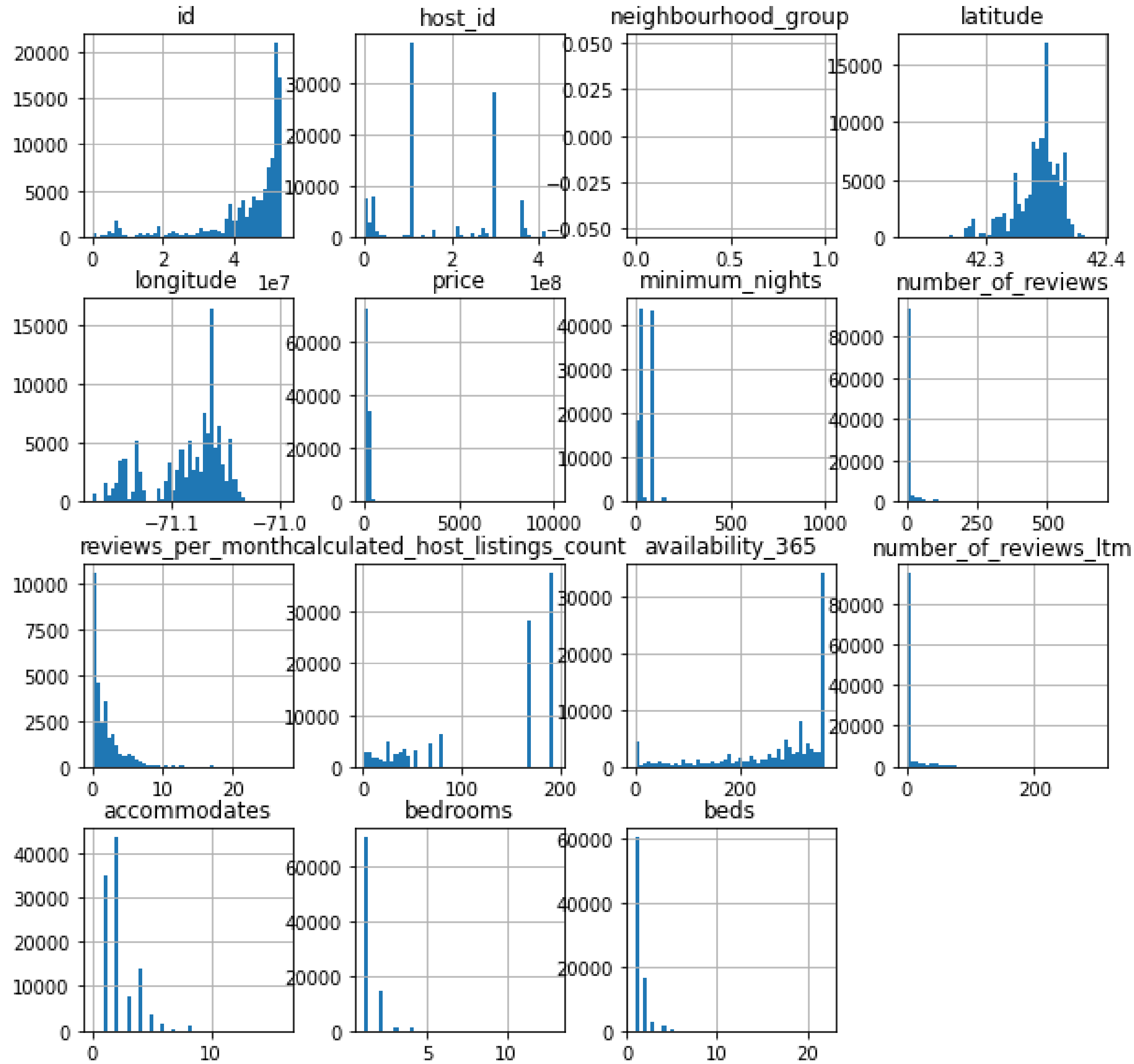
In [7]:
```
### Check for null values
boston3.isna().sum()
```

Out[7]:
```
id                                     0
name                                   0
host_id                                0
host_name                          45569
neighbourhood_group               107389
neighbourhood                          0
latitude                               0
longitude                              0
room_type                              0
price                                  0
minimum_nights                         0
number_of_reviews                      0
last_review                        78337
reviews_per_month                  78337
calculated_host_listings_count         0
availability_365                       0
number_of_reviews_ltm                  0
license                            83939
accommodates                           0
bedrooms                           19170
beds                               25109
amenities                              0
dtype: int64
```

In [8]:
```
### Percentage of null rows in each column
def missing(datas):
    print (round((datas.isnull().sum() * 100/ len(datas)),2).sort_values(as

missing(boston3)
```

```
neighbourhood_group              100.00
license                           78.16
last_review                       72.95
reviews_per_month                 72.95
host_name                         42.43
beds                              23.38
bedrooms                          17.85
id                                 0.00
accommodates                       0.00
number_of_reviews_ltm              0.00
availability_365                   0.00
calculated_host_listings_count     0.00
number_of_reviews                  0.00
name                               0.00
minimum_nights                     0.00
price                              0.00
room_type                          0.00
longitude                          0.00
latitude                           0.00
neighbourhood                      0.00
host_id                            0.00
amenities                          0.00
dtype: float64
```

In [9]:
```
boston3.replace({'f': 0, 't': 1},inplace = True)
boston3.hist(bins=50, figsize=(10,10))
plt.savefig('distribution.png', dpi=650, bbox_inches='tight')
plt.show()
```



We can drop the columns with less categories.Checking whether boolean and categorical features contain sufficient numbers of instances in each category to make them worth including.It can be seen that several columns only contain one category and can be dropped while preprocessing.

Data Preprocessing

In [10]:
```
# To protect the privacy of the hosts and reviewers, we drop them as well a
boston3.drop(['id','host_name','last_review','license',], axis=1, inplace=T
```

In [11]:
```
boston3.drop(['neighbourhood_group'], axis=1, inplace=True)
```

In [12]:
```
boston3.isnull().sum()
```

Out[12]:
```
name                                  0
host_id                               0
neighbourhood                         0
latitude                              0
longitude                             0
room_type                             0
price                                 0
minimum_nights                        0
number_of_reviews                     0
reviews_per_month                 78337
calculated_host_listings_count        0
availability_365                      0
number_of_reviews_ltm                 0
accommodates                          0
bedrooms                          19170
beds                              25109
amenities                             0
dtype: int64
```

In [13]:
```
#Percentage of null rows in each column
def missing(datas):
    print (round((datas.isnull().sum() * 100/ len(datas)),2).sort_values(as

missing(boston3)
```

```
reviews_per_month               72.95
beds                            23.38
bedrooms                        17.85
name                             0.00
accommodates                     0.00
number_of_reviews_ltm            0.00
availability_365                 0.00
calculated_host_listings_count   0.00
number_of_reviews                0.00
host_id                          0.00
minimum_nights                   0.00
price                            0.00
room_type                        0.00
longitude                        0.00
latitude                         0.00
neighbourhood                    0.00
amenities                        0.00
dtype: float64
```

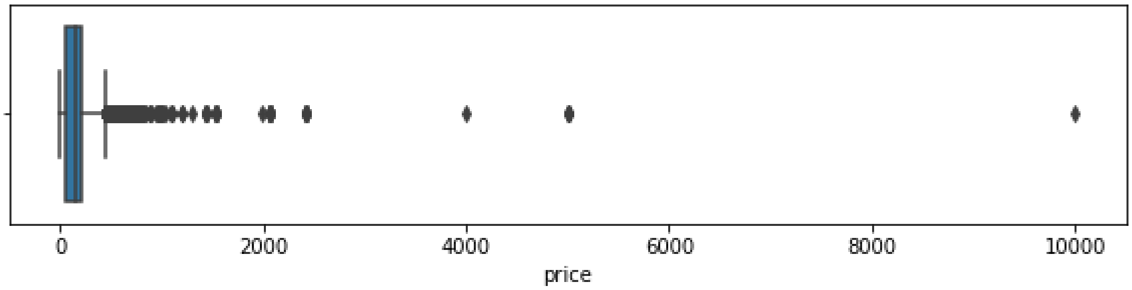
In [14]:
```
### Filling the null values with mean and median

boston3['reviews_per_month'].fillna(boston3['reviews_per_month'].mean(), in
boston3['beds'].fillna(boston3['beds'].median(), inplace=True)
boston3['bedrooms'].fillna(boston3['bedrooms'].median(), inplace=True)
```

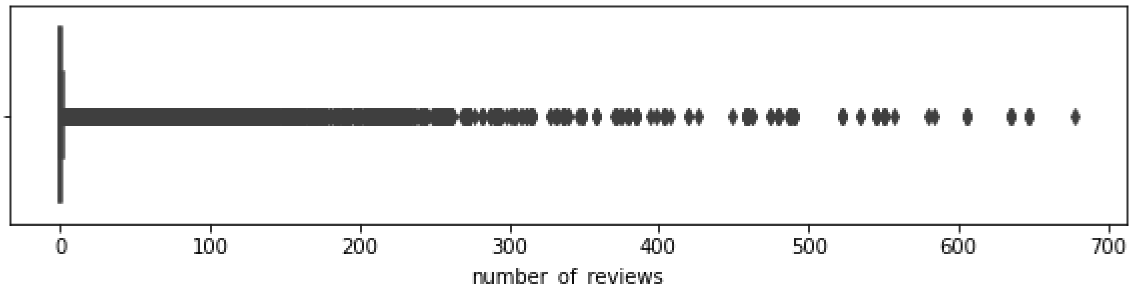
In [15]:
```
missing(boston3)
```

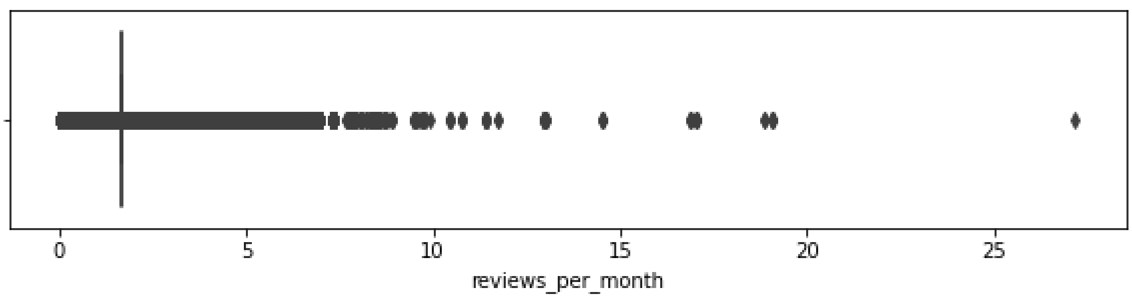
```
name                            0.0
reviews_per_month               0.0
beds                            0.0
bedrooms                        0.0
accommodates                    0.0
number_of_reviews_ltm           0.0
availability_365                0.0
calculated_host_listings_count  0.0
number_of_reviews               0.0
host_id                         0.0
minimum_nights                  0.0
price                           0.0
room_type                       0.0
longitude                       0.0
latitude                        0.0
neighbourhood                   0.0
amenities                       0.0
dtype: float64
```

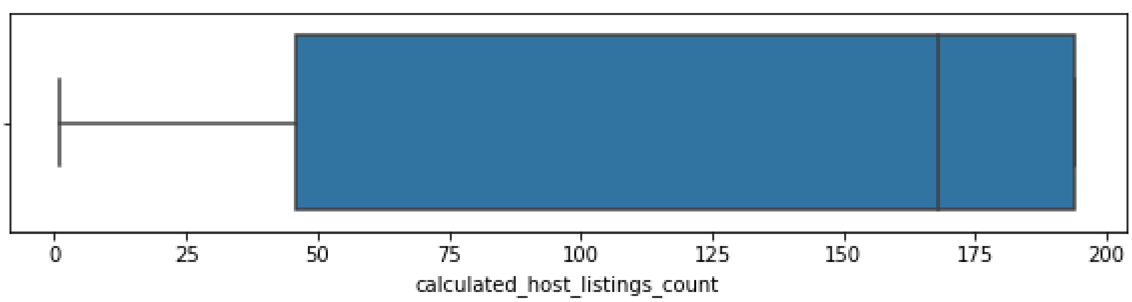
In [16]:
```
bosub = boston3[["price","number_of_reviews","reviews_per_month","calculate
```

In [17]:
```
### Plot to check the outliers
boston1=bosub.select_dtypes(exclude=['object'])

for column in boston1:
        plt.figure(figsize=(10,2))
        sb.boxplot(data=boston1, x=column)
```









In [18]:
```
#Since our target variable is the price, we have standardized the values su
# are in one standarad deviation
x_data = boston3.price
standard = preprocessing.scale(x_data)
print(standard)
```

```
[-0.19604426 -0.41914292 -0.41914292 ... -0.07591421 -0.07591421
 -0.30759359]
```

In [19]:
```
boston3.amenities[:1].values
```

Out[19]:
```
array(['["Cable TV", "Dishes and silverware", "Cooking basics", "Patio or
balcony", "Hair dryer", "Refrigerator", "Hot water", "Dedicated workspac
e", "Bed linens", "Wifi", "Essentials", "Washer", "Air conditioning", "TV
with standard cable", "Oven", "Stove", "Dryer", "Smoke alarm", "Extra pil
lows and blankets", "Hangers", "Carbon monoxide alarm", "Dishwasher", "He
ating", "Shampoo", "Free parking on premises", "Iron", "Kitchen", "Coffee
maker", "Microwave", "Long term stays allowed", "Free street parking"]'],
      dtype=object)
```

In [20]:
```
amenities_list = list(boston3.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_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)
amenities_set
```

Out[20]:
```
{'',
 '2-5 years old',
 '21\\ HDTV with Roku',
 '24-hour fitness center',
 '32\\ HDTV',
 '32\\ HDTV with',
 '32\\ HDTV with Amazon Prime Video',
 '32\\ HDTV with Netflix',
 '32\\ HDTV with standard cable',
 '36\\ HDTV with Apple TV',
 '37\\ HDTV with Amazon Prime Video',
 '40\\ HDTV with Amazon Prime Video',
 '40\\ HDTV with Apple TV',
 '40\\ HDTV with Roku',
 '40\\ HDTV with standard cable',
 '40\\ TV',
 '40\\ TV with Apple TV',
 '42\\ HDTV',
 '42\\ HDTV with Amazon Prime Video',
```

In [21]:
```
boston3.loc[boston3['amenities'].str.contains('Air conditioning|Central air
boston3.loc[boston3['amenities'].str.contains('Gym|24-hour fitness center|P
boston3.loc[boston3['amenities'].str.contains('Apple TV|Game console|Game c
boston3.loc[boston3['amenities'].str.contains('Avanti stainless steel elect
boston3.loc[boston3['amenities'].str.contains('Electric stove|Gas stove|Ike
boston3.loc[boston3['amenities'].str.contains('BBQ grill|Fire pit|Barbecue
boston3.loc[boston3['amenities'].str.contains('balcony|Patio or balcony|Pat
boston3.loc[boston3['amenities'].str.contains('Beach view|Beachfront|Lake a
boston3.loc[boston3['amenities'].str.contains('Bed linens|Bed sheets and pi
boston3.loc[boston3['amenities'].str.contains('Breakfast|Complimentary cont
boston3.loc[boston3['amenities'].str.contains('Coffee maker|Espresso machin
boston3.loc[boston3['amenities'].str.contains('Cooking basics'), 'cooking_b
boston3.loc[boston3['amenities'].str.contains('Dryer|Dryer \\u2013 In build
boston3.loc[boston3['amenities'].str.contains('Elevator'), 'elevator'] = 1
boston3.loc[boston3['amenities'].str.contains('Children\\u2019s books and t
boston3.loc[boston3['amenities'].str.contains('Free driveway parking on pre
boston3.loc[boston3['amenities'].str.contains('Private fenced garden or bac
boston3.loc[boston3['amenities'].str.contains('Host greets you'), 'host_gre
boston3.loc[boston3['amenities'].str.contains('Private hot tub|Shared outdo
boston3.loc[boston3['amenities'].str.contains('Internet|Pocket wifi|Wifi|Po
boston3.loc[boston3['amenities'].str.contains('Long term stays allowed'), '
boston3.loc[boston3['amenities'].str.contains('Private entrance'), 'private
boston3.loc[boston3['amenities'].str.contains('Pets|pet|Cat(s)|Dog(s)|Pets
boston3.loc[boston3['amenities'].str.contains('Safe|Security system|Securit
boston3.loc[boston3['amenities'].str.contains('Self check_in'), 'self_check
boston3.loc[boston3['amenities'].str.contains('Smoking allowed'), 'smoking_
boston3.loc[boston3['amenities'].str.contains('Step-free access|Wheelchair|
boston3.loc[boston3['amenities'].str.contains('Suitable for events'), 'even
boston3.loc[boston3['amenities'].str.contains('Aveeno body soap|Bathtub|Bee
boston3.loc[boston3['amenities'].str.contains('Sonos Bluetooth sound system
boston3.loc[boston3['amenities'].str.contains('Dedicated workspace|Dedicate
boston3.loc[boston3['amenities'].str.contains('Freezer|GE refrigerator| LG
boston3.loc[boston3['amenities'].str.contains('Onsite restaurant \\u2014 Co
```

```
<ipython-input-21-90ad71994ea5>:23: UserWarning: This pattern is interpre
ted as a regular expression, and has match groups. To actually get the gr
oups, use str.extract.
  boston3.loc[boston3['amenities'].str.contains('Pets|pet|Cat(s)|Dog(s)|P
ets allowed'), 'pets_allowed'] = 1
```

In [22]:
```
replacecols = boston3.iloc[:,0:].columns
boston3[replacecols] = boston3[replacecols].fillna(0)
nonessential_amenities = []
for col in boston3.iloc[:,17:].columns:
    if boston3[col].sum() < len(boston3)/10:
        nonessential_amenities.append(col)
boston3.drop(nonessential_amenities, axis=1, inplace=True)
boston3.drop('amenities', axis=1, inplace=True)
```

In [23]:
```
boston3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 107389 entries, 0 to 107388
Data columns (total 34 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   name                            107389 non-null  object 
 1   host_id                         107389 non-null  int64  
 2   neighbourhood                   107389 non-null  object 
 3   latitude                        107389 non-null  float64
 4   longitude                       107389 non-null  float64
 5   room_type                       107389 non-null  object 
 6   price                           107389 non-null  int64  
 7   minimum_nights                  107389 non-null  int64  
 8   number_of_reviews               107389 non-null  int64  
 9   reviews_per_month               107389 non-null  float64
 10  calculated_host_listings_count  107389 non-null  int64  
 11  availability_365                107389 non-null  int64  
 12  number_of_reviews_ltm           107389 non-null  int64  
 13  accommodates                    107389 non-null  int64  
 14  bedrooms                        107389 non-null  float64
 15  beds                            107389 non-null  float64
 16  air_conditioning                107389 non-null  float64
 17  gym                             107389 non-null  float64
 18  tv                              107389 non-null  float64
 19  bbq                             107389 non-null  float64
 20  nature_and_views                107389 non-null  float64
 21  bed_linen                       107389 non-null  float64
 22  coffee_machine                  107389 non-null  float64
 23  cooking_basics                  107389 non-null  float64
 24  white_goods                     107389 non-null  float64
 25  elevator                        107389 non-null  float64
 26  parking                         107389 non-null  float64
 27  hot_tub_sauna_or_pool           107389 non-null  float64
 28  internet                        107389 non-null  float64
 29  long_term_stays                 107389 non-null  float64
 30  private_entrance                107389 non-null  float64
 31  toiletries                      107389 non-null  float64
 32  workspace                       107389 non-null  float64
 33  refrigerator                    107389 non-null  float64
dtypes: float64(23), int64(8), object(3)
memory usage: 28.7+ MB
```

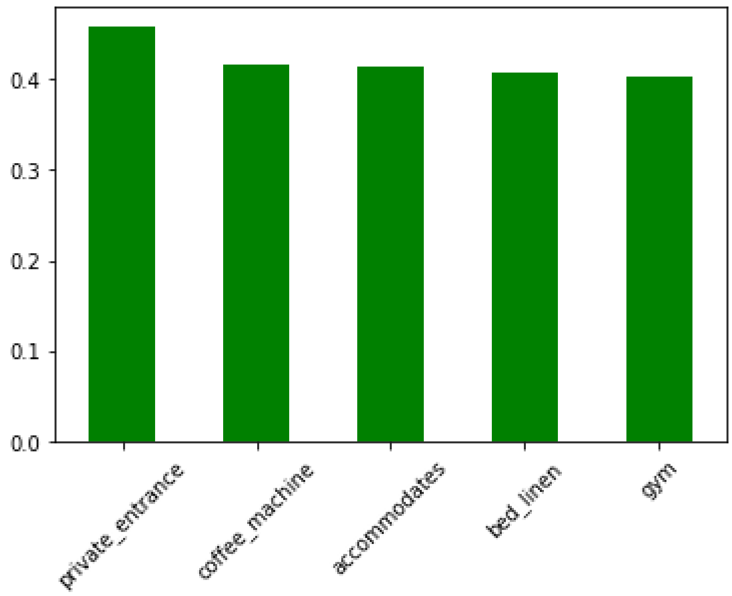
Data Visualization

In [24]:
```
boston3=boston3.drop(['nature_and_views'],axis=1)
```

In [25]:
```
price_corr = boston3.corr()['price'].sort_values(ascending = False).head(6)
price_corr
```

Out[25]:
```
private_entrance    0.457659
coffee_machine      0.415872
accommodates        0.413670
bed_linen           0.406062
gym                 0.403006
Name: price, dtype: float64
```

In [26]:
```
### Features that influence Price
price_corr.plot(kind = 'bar', color = 'g');
plt.xticks(rotation=45);
```



In [27]:
```
boston3.head()
```

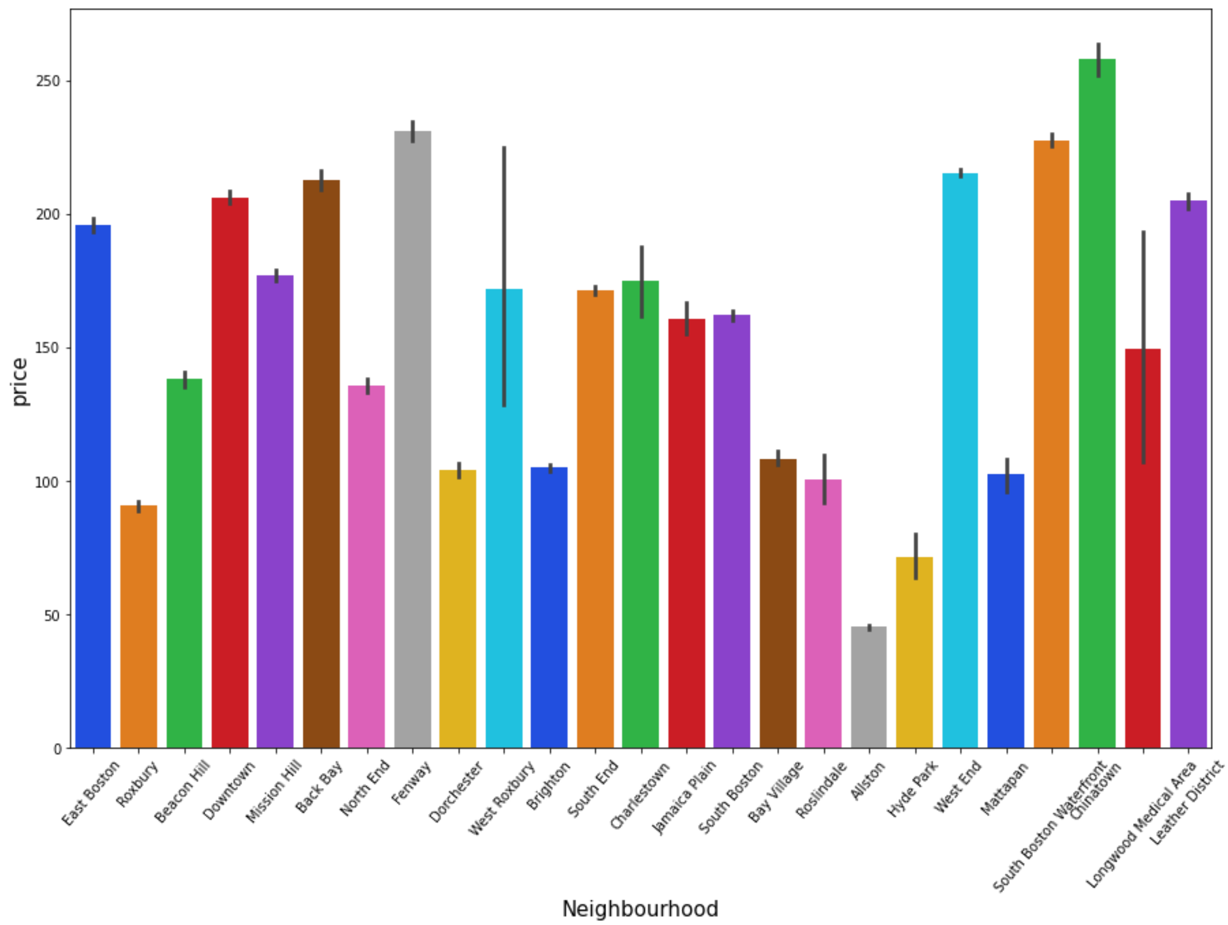
Out[27]:

	name	host_id	neighbourhood	latitude	longitude	room_type	price	minimum_nights
0	HARBORSIDE-Walk to subway	4804	East Boston	42.36413	-71.02991	Entire home/apt	125	32
1	** Private! Minutes to center!**	8229	Roxbury	42.32844	-71.09581	Entire home/apt	99	3
2	** Private! Minutes to center!**	8229	Roxbury	42.32844	-71.09581	Entire home/apt	99	3
3	** Private! Minutes to center!**	8229	Roxbury	42.32844	-71.09581	Entire home/apt	99	3
4	** Private! Minutes to center!**	8229	Roxbury	42.32844	-71.09581	Entire home/apt	99	3

5 rows × 33 columns

In [28]:
```
### Barplot to find the price of different neighbourhoods in Boston

plt.figure(figsize=(15,10))
plt.xticks(rotation=50)
listings = boston3.sort_values(by = 'price')
sb.barplot(x='neighbourhood', y= 'price',palette= "bright", data = boston3)
plt.xlabel(xlabel='Neighbourhood', fontsize=15)
plt.ylabel(ylabel='price', fontsize=15)
```

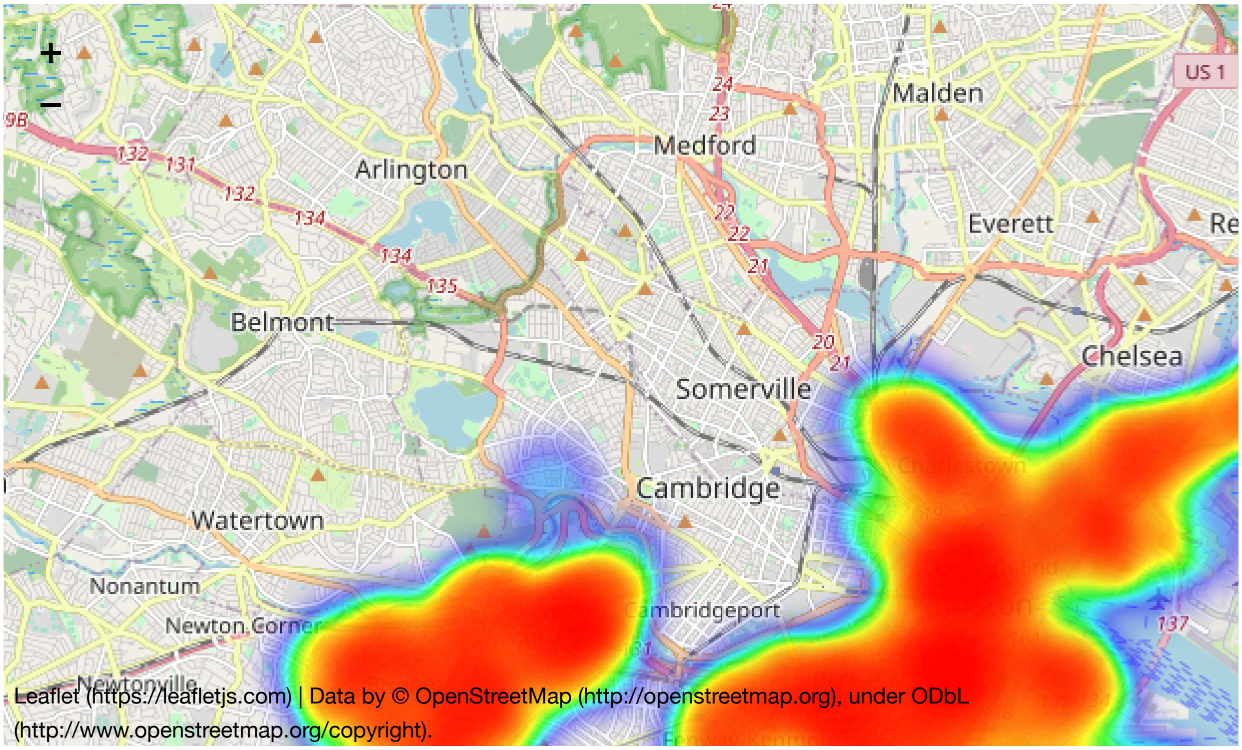
Out[28]: Text(0, 0.5, 'price')



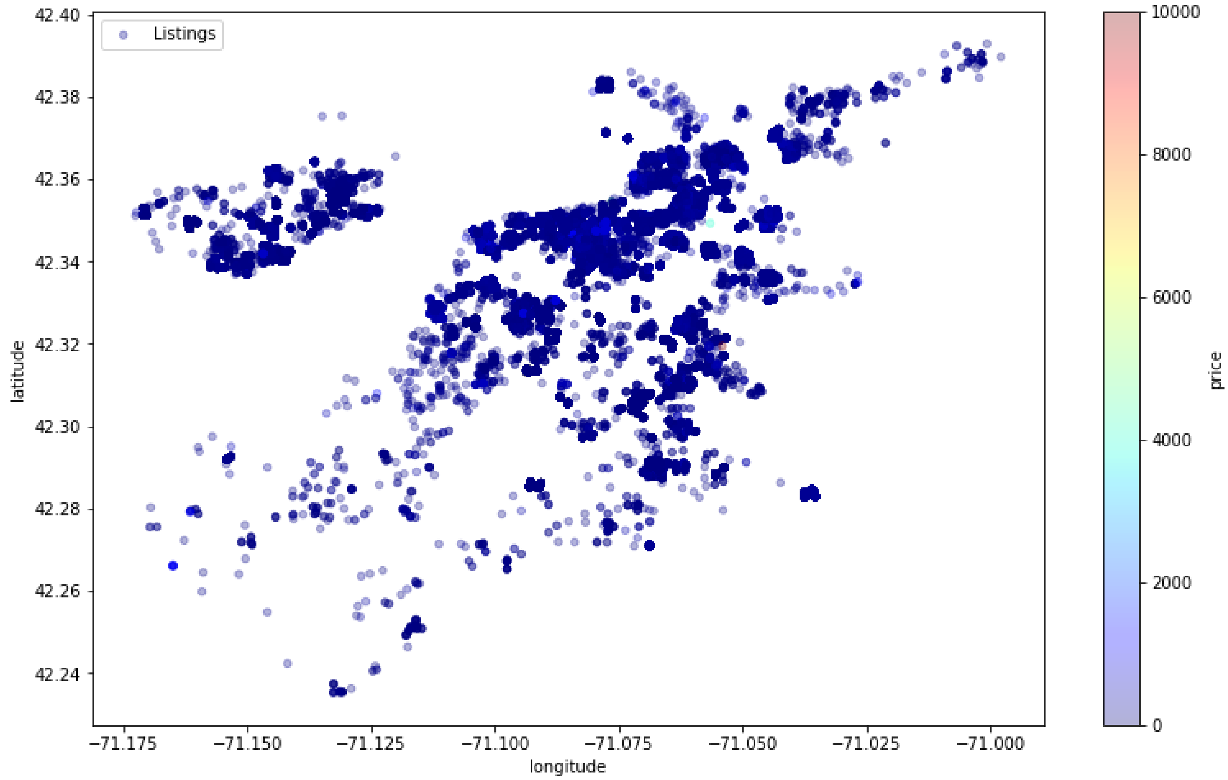
In [29]:

```
maps = folium.Map(location=[42.361145, -71.057083], zoom_start = 12)

heatmap_data = [[row['latitude'],row['longitude']] for index, row in
                boston3[['latitude', 'longitude']].iterrows()]

htm =  HeatMap(heatmap_data).add_to(maps)

maps
```

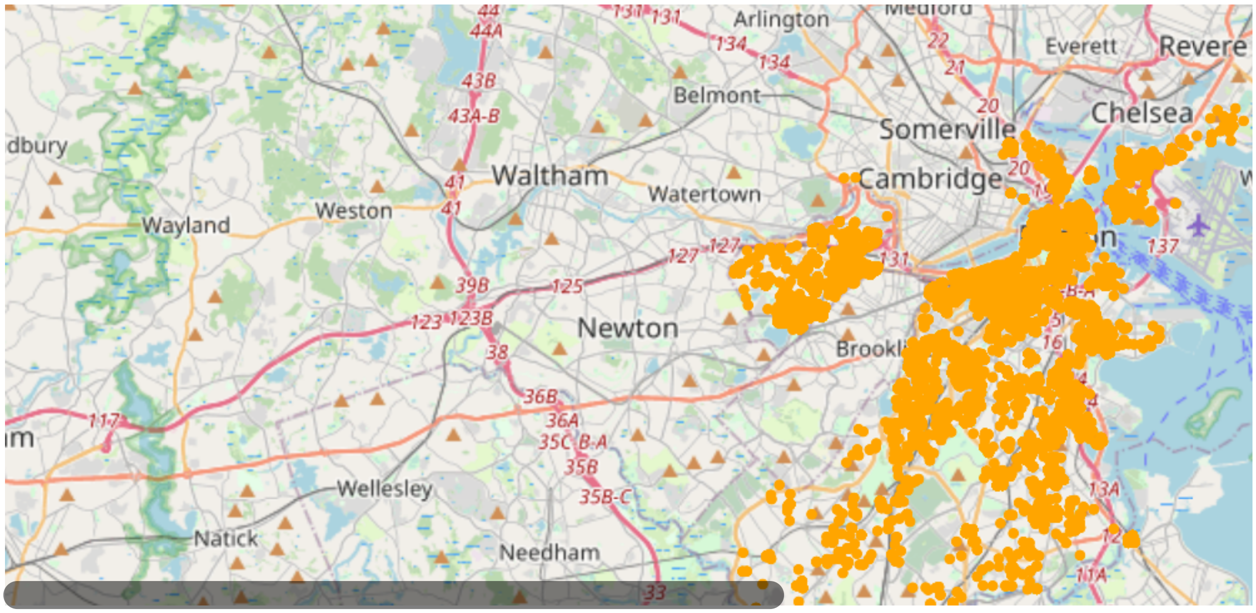
Out[29]:



In [30]:
```
from PIL import Image

plt.figure(figsize=(13,8))
gx = plt.gca()
boston3.plot(kind='scatter',x='longitude',y='latitude',label='Listings', c=
plt.legend()
plt.show()
```



In [31]:
```
### Plot Map for Distribution of location of Airbnb properties in Boston

import plotly.express as px

fig = px.scatter_mapbox(boston3, lat="latitude", lon="longitude", hover_nam
                        color_discrete_sequence=["orange"], zoom=10, height
fig.update_layout(mapbox_style="open-street-map")
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```



In [32]:
```
boston3.describe()
```

Out[32]:

	host_id	latitude	longitude	price	minimum_nights	number_of_revie
count	1.073890e+05	107389.000000	107389.000000	107389.00000	107389.000000	107389.000
mean	1.758024e+08	42.342998	-71.084340	147.84707	51.843345	11.181
std	1.221205e+08	0.019351	0.033584	116.54091	36.352791	38.862
min	4.804000e+03	42.235330	-71.172520	0.00000	1.000000	0.000
25%	1.074344e+08	42.335150	-71.101850	50.00000	30.000000	0.000
50%	1.074344e+08	42.347970	-71.070840	149.00000	32.000000	0.000
75%	2.978601e+08	42.355180	-71.061640	212.00000	91.000000	1.000
max	4.349904e+08	42.392790	-70.997810	10000.00000	1000.000000	678.000

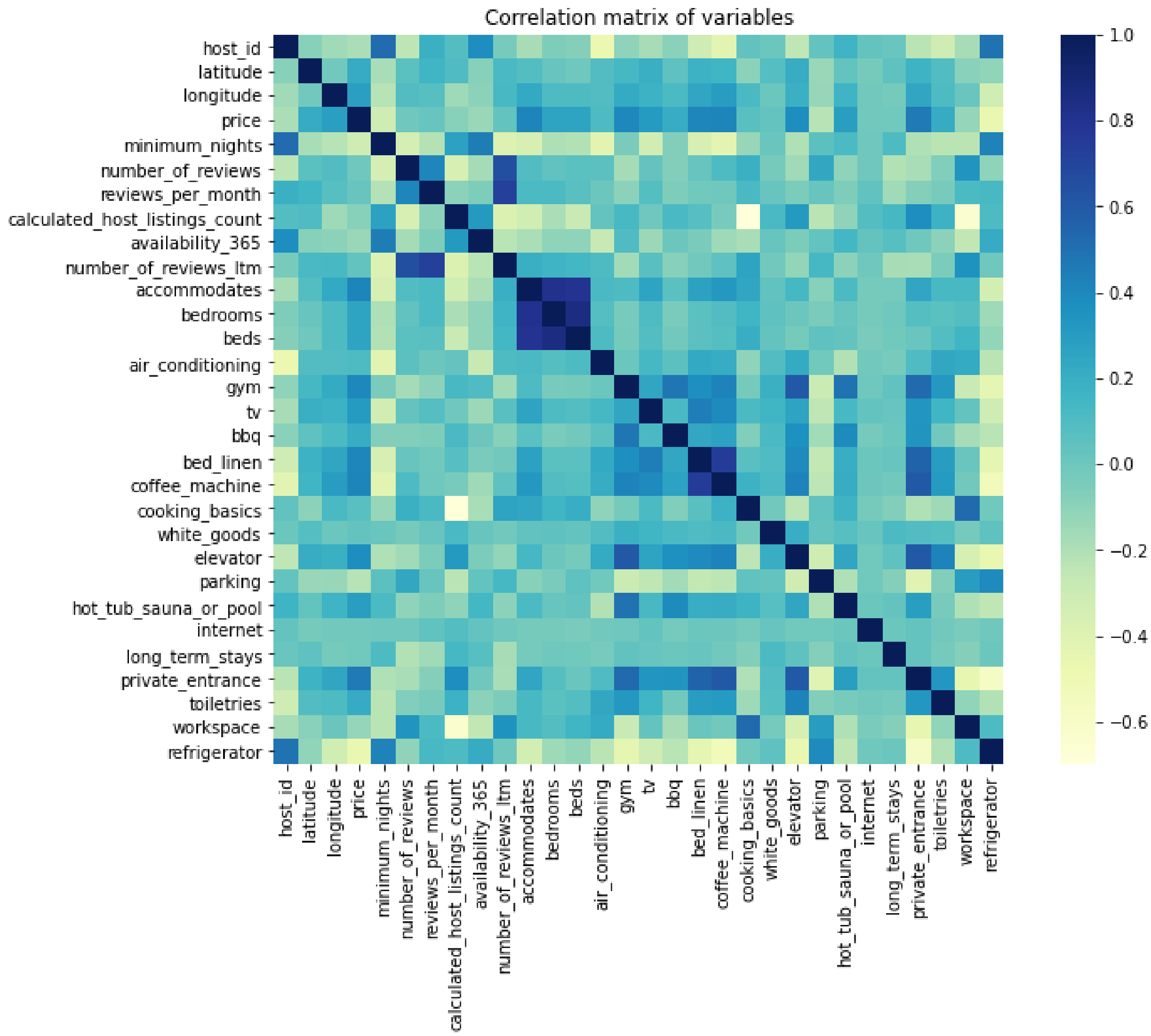
8 rows × 30 columns

In [33]:
```
boston3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 107389 entries, 0 to 107388
Data columns (total 33 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   name                            107389 non-null  object 
 1   host_id                         107389 non-null  int64  
 2   neighbourhood                   107389 non-null  object 
 3   latitude                        107389 non-null  float64
 4   longitude                       107389 non-null  float64
 5   room_type                       107389 non-null  object 
 6   price                           107389 non-null  int64  
 7   minimum_nights                  107389 non-null  int64  
 8   number_of_reviews               107389 non-null  int64  
 9   reviews_per_month               107389 non-null  float64
 10  calculated_host_listings_count  107389 non-null  int64  
 11  availability_365                107389 non-null  int64  
 12  number_of_reviews_ltm           107389 non-null  int64  
 13  accommodates                    107389 non-null  int64  
 14  bedrooms                        107389 non-null  float64
 15  beds                            107389 non-null  float64
 16  air_conditioning                107389 non-null  float64
 17  gym                             107389 non-null  float64
 18  tv                              107389 non-null  float64
 19  bbq                             107389 non-null  float64
 20  bed_linen                       107389 non-null  float64
 21  coffee_machine                  107389 non-null  float64
 22  cooking_basics                  107389 non-null  float64
 23  white_goods                     107389 non-null  float64
 24  elevator                        107389 non-null  float64
 25  parking                         107389 non-null  float64
 26  hot_tub_sauna_or_pool           107389 non-null  float64
 27  internet                        107389 non-null  float64
 28  long_term_stays                 107389 non-null  float64
 29  private_entrance                107389 non-null  float64
 30  toiletries                      107389 non-null  float64
 31  workspace                       107389 non-null  float64
 32  refrigerator                    107389 non-null  float64
dtypes: float64(22), int64(8), object(3)
memory usage: 27.9+ MB
```

In [34]:
```
### correlation verify each feature against the target feature

plt.figure(figsize=(12,8))
title = 'Correlation matrix of variables'
sb.heatmap(boston3.corr(), square=True, cmap='YlGnBu')
plt.title(title)
plt.ioff()
```

Out[34]: <matplotlib.pyplot._IoffContext at 0x7f893dd399a0>



In [35]:
```
boston3=boston3._get_numeric_data()
boston3=boston3.dropna(axis=0)
boston3=boston3.drop(['latitude','longitude'],axis=1)
boston3=boston3.drop(['number_of_reviews_ltm'],axis=1)
boston3.head()
```

Out[35]:

	host_id	price	minimum_nights	number_of_reviews	reviews_per_month	calculated_host_listings_c
0	4804	125	32	21	0.27	
1	8229	99	3	110	0.71	
2	8229	99	3	110	0.71	
3	8229	99	3	110	0.71	
4	8229	99	3	110	0.71	

5 rows × 27 columns

Multi-Collinearity

In [36]:
```
def calc_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.s

    return(vif)
```

In [37]:
```
x = boston3.iloc[:,:-1]
calc_vif(x)
```

Out[37]:

	variables	VIF
0	host_id	7.131918
1	price	4.364104
2	minimum_nights	7.098761
3	number_of_reviews	1.909629
4	reviews_per_month	6.208965
5	calculated_host_listings_count	17.979208
6	availability_365	11.736312
7	accommodates	17.591002
8	bedrooms	30.443464
9	beds	17.887075
10	air_conditioning	6.963641
11	gym	3.731983
12	tv	11.819420
13	bbq	1.743540
14	bed_linen	9.566801
15	coffee_machine	9.450196
16	cooking_basics	4.028319
17	white_goods	10.950189
18	elevator	3.603388
19	parking	2.176904
20	hot_tub_sauna_or_pool	2.105857
21	internet	135.087303
22	long_term_stays	119.966757
23	private_entrance	6.258084
24	toiletries	1.913979
25	workspace	4.101958

In [38]:
```
boston3=boston3.drop(['internet'],axis=1)
boston3=boston3.drop(['long_term_stays'],axis=1)
boston3=boston3.drop(['host_id'],axis=1)
```

In [39]:
```
boston3.iloc[:,6:20].corr()
```

Out[39]:

	accommodates	bedrooms	beds	air_conditioning	gym	tv
accommodates	1.000000	0.811684	0.791003	0.124014	0.109612	0.253955
bedrooms	0.811684	1.000000	0.852407	0.084183	-0.034519	0.109999
beds	0.791003	0.852407	1.000000	0.105523	-0.030464	0.086498
air_conditioning	0.124014	0.084183	0.105523	1.000000	0.006799	0.136304
gym	0.109612	-0.034519	-0.030464	0.006799	1.000000	0.247503
tv	0.253955	0.109999	0.086498	0.136304	0.247503	1.000000
bbq	0.062040	-0.041420	-0.014476	0.057425	0.472893	0.119941
bed_linen	0.274648	0.065092	0.055282	0.228350	0.365264	0.439707
coffee_machine	0.323299	0.093667	0.080691	0.209567	0.418689	0.394103
cooking_basics	0.247697	0.158002	0.203304	-0.103488	-0.031880	0.117889
white_goods	0.044392	0.050702	0.030545	0.092243	0.188861	0.156572
elevator	0.132249	0.005295	-0.031894	0.223253	0.600709	0.257625
parking	-0.086699	-0.045489	0.051400	-0.011805	-0.299307	-0.252522
hot_tub_sauna_or_pool	0.113075	0.019042	0.027351	-0.209335	0.493113	0.127826

In [40]:
```
boston3 = boston3.loc[boston3['price'] < 5000]
```

In [41]:
```
boston3['price'].describe()
```

Out[41]:
```
count    107382.000000
mean        147.484206
std         106.597849
min           0.000000
25%          50.000000
50%         149.000000
75%         212.000000
max        3999.000000
Name: price, dtype: float64
```

In [42]:
```
boston3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 107382 entries, 0 to 107388
Data columns (total 24 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   price                           107382 non-null  int64  
 1   minimum_nights                  107382 non-null  int64  
 2   number_of_reviews               107382 non-null  int64  
 3   reviews_per_month               107382 non-null  float64
 4   calculated_host_listings_count  107382 non-null  int64  
 5   availability_365                107382 non-null  int64  
 6   accommodates                    107382 non-null  int64  
 7   bedrooms                        107382 non-null  float64
 8   beds                            107382 non-null  float64
 9   air_conditioning                107382 non-null  float64
 10  gym                             107382 non-null  float64
 11  tv                              107382 non-null  float64
 12  bbq                             107382 non-null  float64
 13  bed_linen                       107382 non-null  float64
 14  coffee_machine                  107382 non-null  float64
 15  cooking_basics                  107382 non-null  float64
 16  white_goods                     107382 non-null  float64
 17  elevator                        107382 non-null  float64
 18  parking                         107382 non-null  float64
 19  hot_tub_sauna_or_pool           107382 non-null  float64
 20  private_entrance                107382 non-null  float64
 21  toiletries                      107382 non-null  float64
 22  workspace                       107382 non-null  float64
 23  refrigerator                    107382 non-null  float64
dtypes: float64(18), int64(6)
memory usage: 20.5 MB
```

In [43]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(boston3.drop(["price"],
```

In [44]:
```
from ALY6040_Group3 import normalize_data
xscaler,x_processed = normalize_data(x_train)
```

Decision Tree

In [45]:
```
from sklearn.tree import DecisionTreeRegressor
from ALY6040_Group3 import train_model,score,predict

model1 = decisionTree = train_model(x_processed,y_train,DecisionTreeRegress
start_time = time.time()
e = predict(model1,xscaler,x_test)
o = score(y_test,e)
print(type(model1).__name__,o)
print("Execution time: " + str((time.time() - start_time)) + ' ms')
a,b,c = o
xc=str((time.time() - start_time))

tab = pd.DataFrame({'Actual Values': np.array(y_test).flatten(), 'Decision
tab.set_index('Actual Values', inplace=True)
tab
```

```
DecisionTreeRegressor {'MAE': 11.325720752933316, 'R2': 0.847319073813082
5, 'RMSE': 41.55910163033422}
Execution time: 0.020350217819213867 ms
```

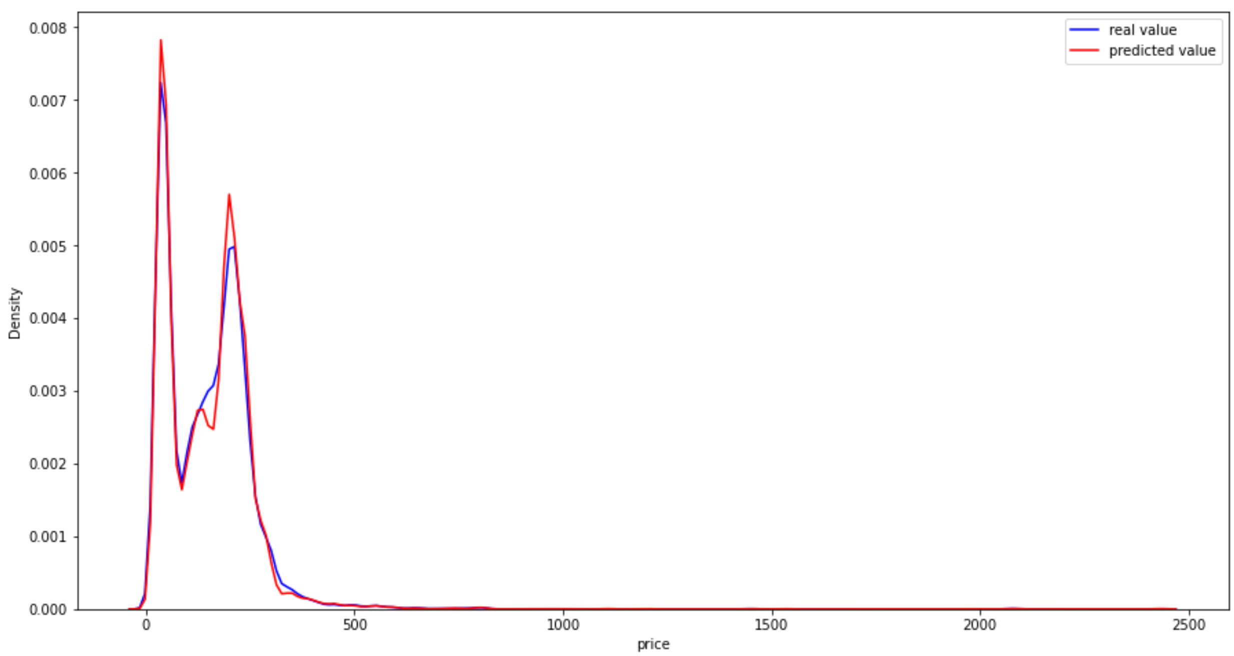
Out[45]:

	Decision Tree
Actual Values	
34	33.969697
203	201.531915
240	240.000000
128	128.000000
131	131.000000
341	341.000000
40	42.285714
247	247.000000
52	52.000000
34	35.338462
299	299.000000
196	196.000000
177	214.333333
120	120.000000
181	181.000000
205	179.800000
194	187.666667
132	132.000000
277	277.000000
60	60.000000

In [46]:
```
### Predicted Values vs Actual Values in Decision Tree

plt.figure(figsize=(15,8))
ax1=sb.distplot(y_test,hist=False,color='blue',label='real value')
ax2=sb.distplot(e,hist=False,color='red',label='predicted value')
plt.legend()
```

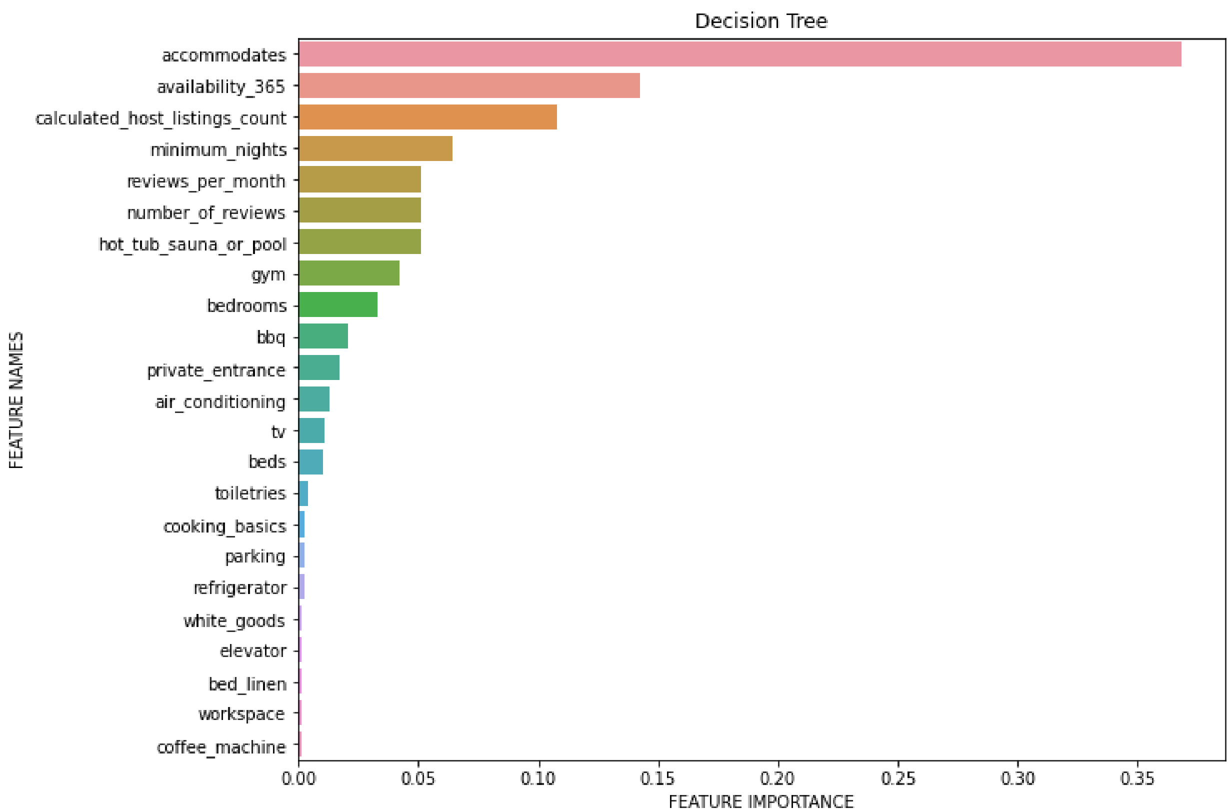
```
/Users/miravparekh/opt/anaconda3/lib/python3.8/site-packages/seaborn/dist
ributions.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future versi
on. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `kdeplot` (an axes-level function for ker
nel density plots).

/Users/miravparekh/opt/anaconda3/lib/python3.8/site-packages/seaborn/dist
ributions.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future versi
on. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `kdeplot` (an axes-level function for ker
nel density plots).
```

Out[46]: <matplotlib.legend.Legend at 0x7f8889bc4c70>



Feature Importance

In [47]:
```
### Feature importance plot of Decision Tree

def plot_feature_importance(importance,names,model_type):

    feature_importance = np.array(importance)
    feature_names = np.array(names)
    data={'feature_names':feature_names,'feature_importance':feature_import
    fi_df = pd.DataFrame(data)
    fi_df.sort_values(by=['feature_importance'], ascending=False,inplace=Tr
    plt.figure(figsize=(10,8))
    sb.barplot(x=fi_df['feature_importance'], y=fi_df['feature_names'])
    plt.title(model_type)
    plt.xlabel('FEATURE IMPORTANCE')
    plt.ylabel('FEATURE NAMES')

plot_feature_importance(model1.feature_importances_,x_train.columns,'Decisi
```



Cross Validation

In [48]:
```
from sklearn.model_selection import cross_val_score

lm3 = DecisionTreeRegressor()
scores = cross_val_score(lm3, x_train, y_train, scoring='r2', cv=7)
print('scores:',scores)
print('Mean Score',np.mean(scores))
```

```
scores: [0.87907627 0.83702184 0.82045976 0.86588401 0.89675677 0.7702325
4
 0.88322284]
Mean Score 0.8503791470722932
```

Random Forest Regressor

In [49]:
```
from sklearn.ensemble import RandomForestRegressor

model2 = RandomForestRegressor = train_model(x_processed,y_train,RandomFore
start_time = time.time()
u = predict(model2,xscaler,x_test)
r = score(y_test,u)
print(type(model2).__name__,r)
print("Execution time: " + str((time.time() - start_time)) + ' ms')

xc1=str((time.time() - start_time))

tab['Random Forest Regressor'] = np.array(u[:20])
tab
```

```
RandomForestRegressor {'MAE': 11.190295587176706, 'R2': 0.909354308780441
3, 'RMSE': 32.021917965836806}
Execution time: 0.6434817314147949 ms
```

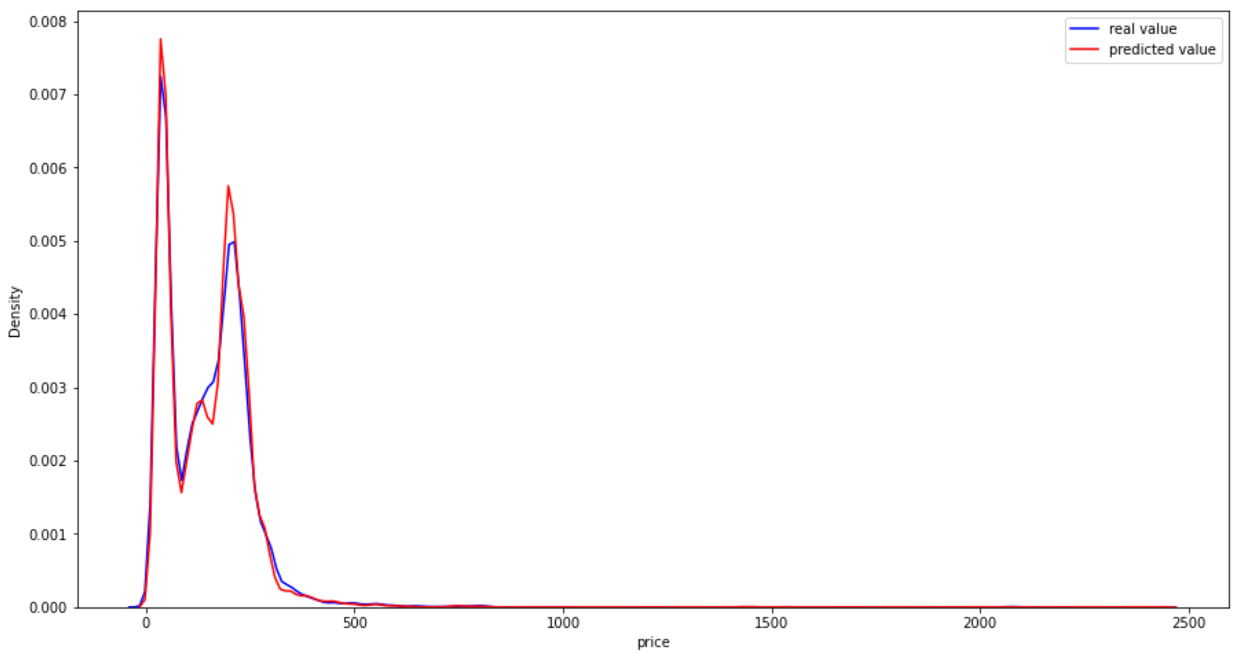
Out[49]:

	Decision Tree	**Random Forest Regressor**
Actual Values		
34	33.969697	33.925001
203	201.531915	201.523025
240	240.000000	240.000000
128	128.000000	128.000000
131	131.000000	131.180000
341	341.000000	341.000000
40	42.285714	42.209118
247	247.000000	247.000000
52	52.000000	52.000000
34	35.338462	35.395881
299	299.000000	299.000000
196	196.000000	196.000000
177	214.333333	214.460994
120	120.000000	120.000000
181	181.000000	181.050000
205	179.800000	176.761500
194	187.666667	186.478005
132	132.000000	132.000000
277	277.000000	277.000000
60	60.000000	59.980000

In [50]:
```
### Predicted Values vs Actual Values in Random Forest Regressor

plt.figure(figsize=(15,8))
ax1=sb.distplot(y_test,hist=False,color='blue',label='real value')
ax2=sb.distplot(u,hist=False,color='red',label='predicted value')
plt.legend()
```

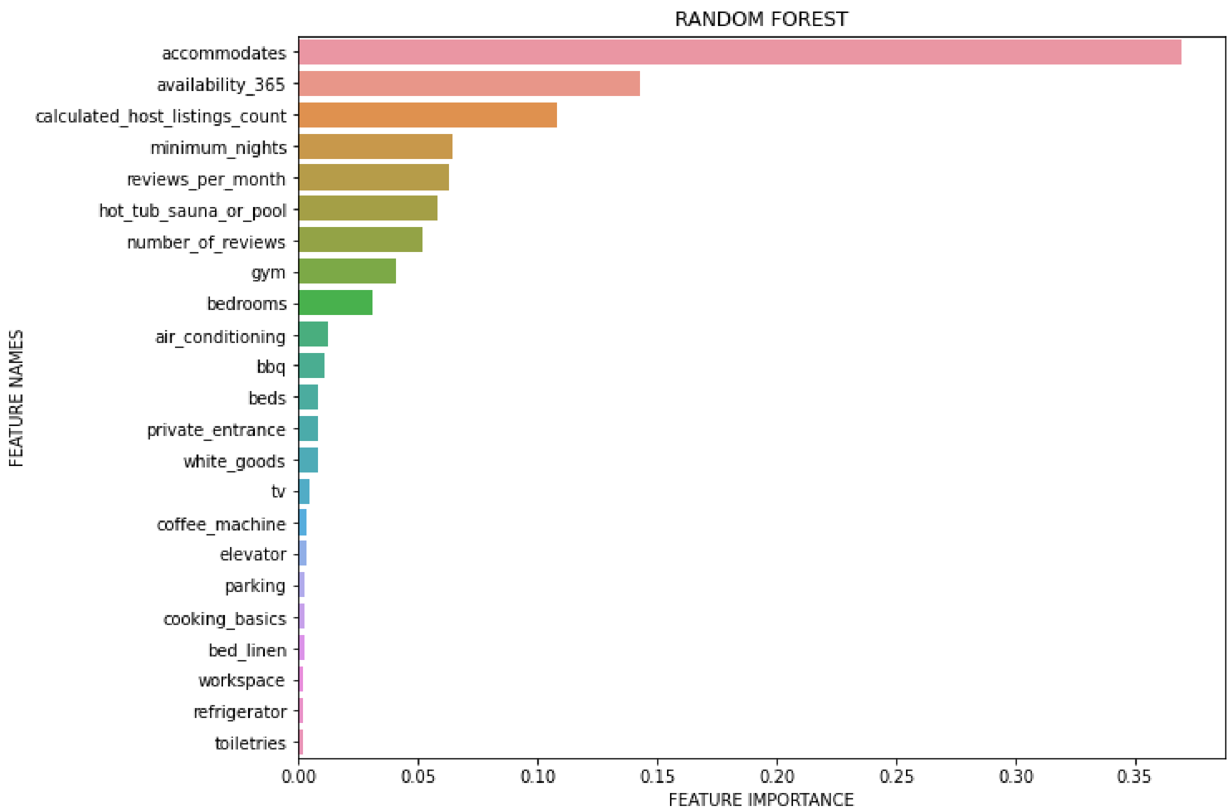
```
/Users/miravparekh/opt/anaconda3/lib/python3.8/site-packages/seaborn/dist
ributions.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future versi
on. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `kdeplot` (an axes-level function for ker
nel density plots).

/Users/miravparekh/opt/anaconda3/lib/python3.8/site-packages/seaborn/dist
ributions.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future versi
on. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `kdeplot` (an axes-level function for ker
nel density plots).
```

Out[50]: <matplotlib.legend.Legend at 0x7f893d41f2b0>



Feature Importance

In [51]:
```
def plot_feature_importance(importance,names,model_type):

    feature_importance = np.array(importance)
    feature_names = np.array(names)
    data={'feature_names':feature_names,'feature_importance':feature_import
    fi_df = pd.DataFrame(data)
    fi_df.sort_values(by=['feature_importance'], ascending=False,inplace=Tr
    plt.figure(figsize=(10,8))
    sb.barplot(x=fi_df['feature_importance'], y=fi_df['feature_names'])
    plt.title(model_type)
    plt.xlabel('FEATURE IMPORTANCE')
    plt.ylabel('FEATURE NAMES')

plot_feature_importance(model2.feature_importances_,x_train.columns,'RANDOM
```



Hyperparameter Tuning

In [52]:
```
from sklearn.ensemble import RandomForestRegressor
Rf = RandomForestRegressor(random_state=42)
Rf.fit(x_train, y_train)
```

Out[52]:
```
RandomForestRegressor(random_state=42)
```

In [53]:
```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 85, 90],
    'max_features': ['auto'],
    'min_samples_leaf': [1],
    'min_samples_split': [2, 4],
    'n_estimators': [780, 800, 820]
}
```

In [54]:
```
grid_search = GridSearchCV(estimator = Rf,
                           param_grid = param_grid,
                           cv = 2, n_jobs = -1, verbose = 2,
                           scoring = 'accuracy')
```

In [55]:
```
grid_search.fit(x_train, y_train)
grid_search.best_params_
```

```
Fitting 2 folds for each of 18 candidates, totalling 36 fits

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call las
t)
<ipython-input-55-a52e1defc50d> in <module>
----> 1 grid_search.fit(x_train, y_train)
      2 grid_search.best_params_

~/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py i
n inner_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64 
     65             # extra_args > 0

~/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_sear
ch.py in fit(self, X, y, groups, **fit_params)
    839                 return results
    840 
--> 841             self._run_search(evaluate_candidates)
    842 
    843             # multimetric is determined here because in the case
 of a callable

~/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_sear
ch.py in _run_search(self, evaluate_candidates)
   1286     def _run_search(self, evaluate_candidates):
   1287         """Search all candidates in param_grid"""
-> 1288         evaluate_candidates(ParameterGrid(self.param_grid))
   1289 
   1290 

~/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_sear
ch.py in evaluate_candidates(candidate_params, cv, more_results)
    793                               n_splits, n_candidates, n_candidate
s * n_splits))
    794 
--> 795                 out = parallel(delayed(_fit_and_score)(clone(base
_estimator),
    796                                                        X, y,
    797                                                        train=trai
n, test=test,

~/opt/anaconda3/lib/python3.8/site-packages/joblib/parallel.py in __call_
_(self, iterable)
   1052 
   1053             with self._backend.retrieval_context():
-> 1054                 self.retrieve()
   1055             # Make sure that we get a last message telling us we
 are done
   1056             elapsed_time = time.time() - self._start_time
```

```
~/opt/anaconda3/lib/python3.8/site-packages/joblib/parallel.py in retriev
e(self)
    931             try:
    932                 if getattr(self._backend, 'supports_timeout', Fal
se):
--> 933                     self._output.extend(job.get(timeout=self.time
out))
    934                 else:
    935                     self._output.extend(job.get())

~/opt/anaconda3/lib/python3.8/site-packages/joblib/_parallel_backends.py
 in wrap_future_result(future, timeout)
    540         AsyncResults.get from multiprocessing."""
    541         try:
--> 542             return future.result(timeout=timeout)
    543         except CfTimeoutError as e:
    544             raise TimeoutError from e

~/opt/anaconda3/lib/python3.8/concurrent/futures/_base.py in result(self,
timeout)
    432                 return self.__get_result()
    433 
--> 434             self._condition.wait(timeout)
    435 
    436             if self._state in [CANCELLED, CANCELLED_AND_NOTIFIED]
:

~/opt/anaconda3/lib/python3.8/threading.py in wait(self, timeout)
    300         try:    # restore state no matter what (e.g., KeyboardInt
errupt)
    301             if timeout is None:
--> 302                 waiter.acquire()
    303                 gotit = True
    304             else:

KeyboardInterrupt:
```

In [ ]:

```
from sklearn.ensemble import RandomForestRegressor

Rf = RandomForestRegressor(random_state=42, bootstrap= True,
 max_depth= 80,
 max_features= 'auto',
 min_samples_leaf= 1,
 min_samples_split= 2,
 n_estimators= 780)

Rf.fit(x_train, y_train)
```

In [ ]:

```
ss= Rf.predict(x_test)
ssg = score(y_test,ss)
print(type(Rf).__name__,ssg)
```

Cross Validation

In [ ]:
```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor

lm2 = RandomForestRegressor()
scores = cross_val_score(lm2, x_train, y_train, scoring='r2', cv=7)
print('scores:',scores)
print('Mean Score',np.mean(scores))
```

XGBoost

In [56]:
```
import xgboost as xgb

model3 = xgb = train_model(x_processed,y_train,xgb.XGBRegressor())
start_time = time.time()
h = predict(model3,xscaler,x_test)
j = score(y_test,h)
print(type(model3).__name__,j)
print("Execution time: " + str((time.time() - start_time)) + ' ms')

xc2=str((time.time() - start_time))

tab['XGBoost'] = np.array(h[:20])
tab
```

```
/Users/miravparekh/opt/anaconda3/lib/python3.8/site-packages/xgboost/comp
at.py:31: FutureWarning:

pandas.Int64Index is deprecated and will be removed from pandas in a futu
re version. Use pandas.Index with the appropriate dtype instead.


XGBRegressor {'MAE': 15.525294087041559, 'R2': 0.9120283150987354, 'RMS
E': 31.546066398182603}
Execution time: 0.055891990661621094 ms
```

Out[56]:

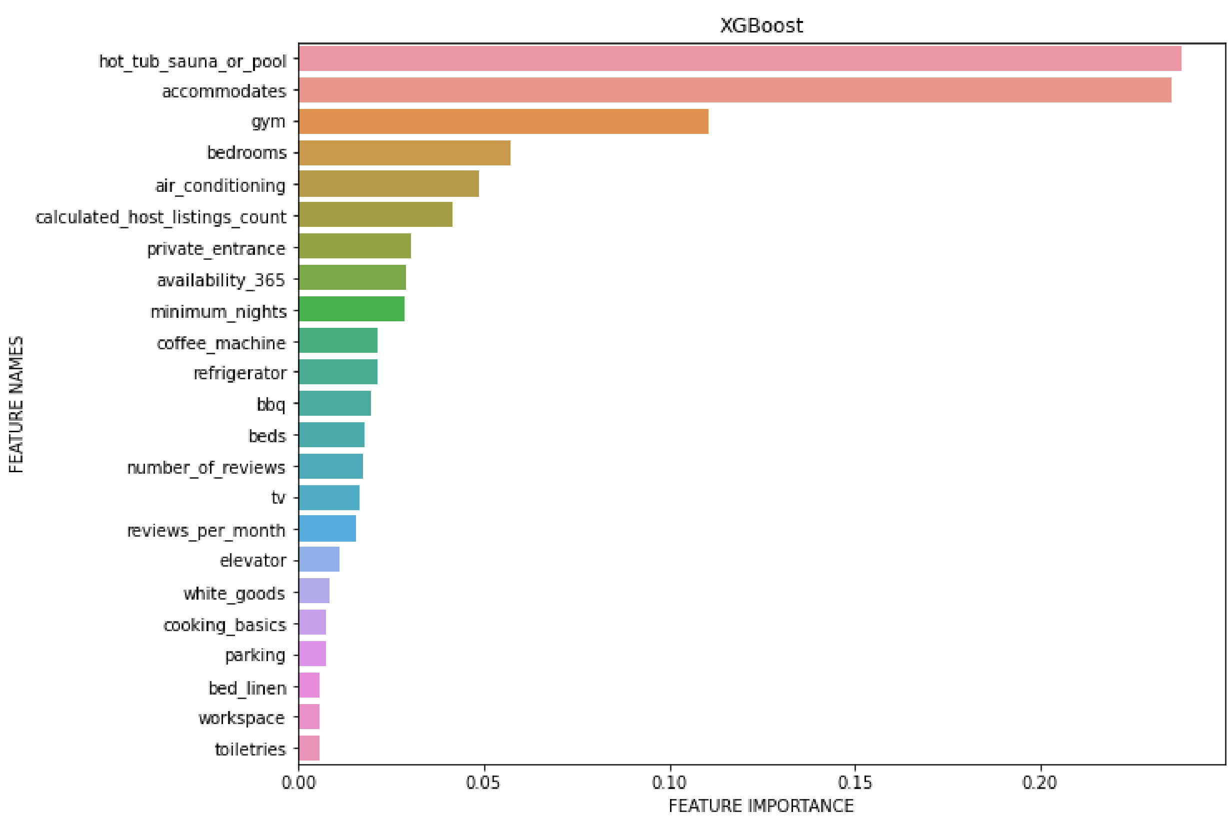
	Decision Tree	Random Forest Regressor	XGBoost
Actual Values			
34	33.969697	33.925001	35.097530
203	201.531915	201.523025	196.505692
240	240.000000	240.000000	230.773285
128	128.000000	128.000000	151.425720
131	131.000000	131.180000	131.134521
341	341.000000	341.000000	328.379211
40	42.285714	42.209118	45.962440
247	247.000000	247.000000	233.773773
52	52.000000	52.000000	48.915230
34	35.338462	35.395881	38.652424
299	299.000000	299.000000	310.659882
196	196.000000	196.000000	190.672577
177	214.333333	214.460994	214.501129
120	120.000000	120.000000	130.386276
181	181.000000	181.050000	185.921860
205	179.800000	176.761500	178.372894
194	187.666667	186.478005	193.043060
132	132.000000	132.000000	156.580856

	Decision Tree	Random Forest Regressor	XGBoost
Actual Values			
277	277.000000	277.000000	252.791519
60	60.000000	59.980000	67.213402

In [ ]:
```
plt.figure(figsize=(15,8))
ax1=sb.distplot(y_test,hist=False,color='blue',label='real value')
ax2=sb.distplot(u,hist=False,color='red',label='predicted value')
plt.legend()
```

Feature Importance

In [57]:
```
def plot_feature_importance(importance,names,model_type):

    feature_importance = np.array(importance)
    feature_names = np.array(names)
    data={'feature_names':feature_names,'feature_importance':feature_import
    fi_df = pd.DataFrame(data)
    fi_df.sort_values(by=['feature_importance'], ascending=False,inplace=Tr
    plt.figure(figsize=(10,8))
    sb.barplot(x=fi_df['feature_importance'], y=fi_df['feature_names'])
    plt.title(model_type)
    plt.xlabel('FEATURE IMPORTANCE')
    plt.ylabel('FEATURE NAMES')

plot_feature_importance(model3.feature_importances_,x_train.columns,'XGBoos
```



Hyperparameter Tuning

In [ ]:
```
import xgboost as xgb
xg = xgb.XGBRegressor(random_state=42)
xg.fit(x_train, y_train)
```

In [ ]:
```
from sklearn.model_selection import GridSearchCV
param_grid = {'learning_rate': [0.1, 0.05],
              'max_depth': [5, 7, 9],
              'n_estimators': [100, 500, 900]}
```

In [ ]:
```
gs = GridSearchCV(estimator = xg,
                           param_grid = param_grid,
                           cv = 2, n_jobs = -1, verbose = 2,
                           scoring = 'accuracy')
```

In [ ]:
```
gs.fit(x_train, y_train)
gs.best_params_
```

In [ ]:
```
from xgboost import XGBRegressor
xg = XGBRegressor(random_state = 42, learning_rate = 0.1, max_depth = 5, n_

xg.fit(x_train,y_train)
```

In [ ]:
```
xj= xg.predict(x_train)
dsx = score(y_train,xj)
print(type(xg).__name__,dsx)
```

Cross Validation

In [ ]:
```
from sklearn.model_selection import cross_val_score
import xgboost as xgb
lm1 = xgb.XGBRegressor()
scores = cross_val_score(lm1, x_train, y_train, scoring='r2', cv=7)
print('scores:',scores)
print('Mean Score',np.mean(scores))
```

Linear Regression

In [58]:
```
from sklearn.linear_model import LinearRegression

model4 = linearRegression = train_model(x_processed,y_train,LinearRegressio
start_time = time.time()
w = predict(model4,xscaler,x_test)
p = score(y_test,w)
print(type(model4).__name__,p)
print("Execution time: " + str((time.time() - start_time)) + ' ms')

xc3=str((time.time() - start_time))

tab['Linear Regression'] = np.array(w[:20])
tab
```

```
LinearRegression {'MAE': 41.337713933599446, 'R2': 0.4810204575904966, 'R
MSE': 76.6211873143184}
Execution time: 0.013142108917236328 ms
```

Out[58]:

Actual Values	Decision Tree	Random Forest Regressor	XGBoost	Linear Regression
34	33.969697	33.925001	35.097530	59.999143
203	201.531915	201.523025	196.505692	207.851565
240	240.000000	240.000000	230.773285	203.099159
128	128.000000	128.000000	151.425720	193.862797
131	131.000000	131.180000	131.134521	154.622943
341	341.000000	341.000000	328.379211	206.177947
40	42.285714	42.209118	45.962440	47.359935
247	247.000000	247.000000	233.773773	210.418266
52	52.000000	52.000000	48.915230	70.707581
34	35.338462	35.395881	38.652424	103.220043
299	299.000000	299.000000	310.659882	255.795042
196	196.000000	196.000000	190.672577	206.727563
177	214.333333	214.460994	214.501129	210.148438
120	120.000000	120.000000	130.386276	187.213011
181	181.000000	181.050000	185.921860	199.358890
205	179.800000	176.761500	178.372894	205.837079
194	187.666667	186.478005	193.043060	203.354411
132	132.000000	132.000000	156.580856	198.270749
277	277.000000	277.000000	252.791519	172.610189
60	60.000000	59.980000	67.213402	143.253494

Cross Validation

In [ ]:
```
from sklearn.model_selection import cross_val_score

lm = LinearRegression()
scores = cross_val_score(lm, x_train, y_train, scoring='r2', cv=7)
print('scores:',scores)
print('Mean Score',np.mean(scores))
```

In [ ]:
```
plt.figure(figsize=(15,8))
ax1=sb.distplot(y_test,hist=False,color='blue',label='real value')
ax2=sb.distplot(u,hist=False,color='red',label='predicted value')
plt.legend()
```

Feature Importance

In [ ]:
```
importance = model4.coef_
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
    plt.xlabel('FEATURE IMPORTANCE')
    plt.ylabel('FEATURE NAMES')
plot_feature_importance([x for x in range(len(importance))],importance,'Lin
```

Regularization

In [ ]:
```
# Creating a linear regression model with Elastic net
re = ElasticNet(alpha=1.0,l1_ratio=0.5)
re.fit(boston3.drop(["price"],axis=1),boston3.price)
y_pred=re.predict(x_test)

print("RMSE: %.3f" % mean_squared_error(y_test, y_pred))
print("R2: %.3f" % r2_score(y_test, y_pred))
```

In [ ]:
```
import statsmodels.formula.api as smf

#fit regression model
fit = smf.ols('price ~ host_id+minimum_nights+number_of_reviews+reviews_per

#view model summary
print(fit.summary())

from statsmodels.compat import lzip
import statsmodels.stats.api as sms

#perform Bresuch-Pagan test
names = ['Lagrange multiplier statistic', 'p-value',
        'f-value', 'f p-value']
test = sms.het_breuschpagan(fit.resid, fit.model.exog)

lzip(names, test)
```

In [ ]:
```
from tabulate import tabulate

tab1 = [
    ["Decision Tree", xc],
    ["Random Forest Regressor", xc1],
    ["XGBoost", xc2],
    ["Linear Regression", xc3]
]

head = ["Model", "Execution Time"]
print(tabulate(tab1, headers=head, tablefmt="grid"))
```

In [ ]:
```
from tabulate import tabulate

xz=o['RMSE']
xz1=r['RMSE']
xz2=dsx['RMSE']
xz3=p['RMSE']

xxz=o['MAE']
xxz1=r['MAE']
xxz2=dsx['MAE']
xxz3=p['MAE']

xxxz=o['R2']
xxxz1=r['R2']
xxxz2=dsx['R2']
xxxz3=p['R2']


tab2 = [
    ["Decision Tree",xxz,xxxz,xz],
    ["Random Forest Regressor",xxz1,xxxz1,xz1],
    ["XGBoost",xxz2,xxxz2,xz2],
    ["Linear Regression",xxz3,xxxz3,xz3]
]

head = ["Model","MAE","R2","RMSE"]
print(tabulate(tab2, headers=head, tablefmt="grid"))
```

In [ ]: