ALY6040-80514 Group 3

Group Members: - Karolina Grodzinska, Mirav Parekh, Rhythm Desai, Vaidehi Chauhan

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/listi
ur1 = "http://data.insideairbnb.com/united-states/ma/boston/2021-12-17/visualisa
```

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(ur1)
```

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

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	
107384	53772114	Entire Modern Apartment in Downtown Boston 2 R...	434990435	Luke	NaN	South End	42.
107385	53772114	Entire Modern Apartment in Downtown Boston 2 R...	434990435	Luke	NaN	South End	42.
107386	53772114	Entire Modern Apartment in Downtown Boston 2 R...	434990435	Luke	NaN	South End	42.
107387	53772114	Entire Modern Apartment in Downtown Boston 2 R...	434990435	Luke	NaN	South End	42.
107388	53756737	The Arbor Retreat	42715907	Emma	NaN	Jamaica Plain	42.

5 rows × 22 columns

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

Out[5]:

	id	host_id	neighbourhood_group	latitude	longitude	
count	1.073890e+05	1.073890e+05	0.0	107389.000000	107389.000000	10738
mean	4.461479e+07	1.758024e+08	NaN	42.342998	-71.084340	14
std	1.192542e+07	1.221205e+08	NaN	0.019351	0.033584	11
min	3.781000e+03	4.804000e+03	NaN	42.235330	-71.172520	
25%	4.208014e+07	1.074344e+08	NaN	42.335150	-71.101850	5
50%	4.961078e+07	1.074344e+08	NaN	42.347970	-71.070840	14
75%	5.227869e+07	2.978601e+08	NaN	42.355180	-71.061640	21
max	5.383997e+07	4.349904e+08	NaN	42.392790	-70.997810	1000

In [6]:
```
### 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
```

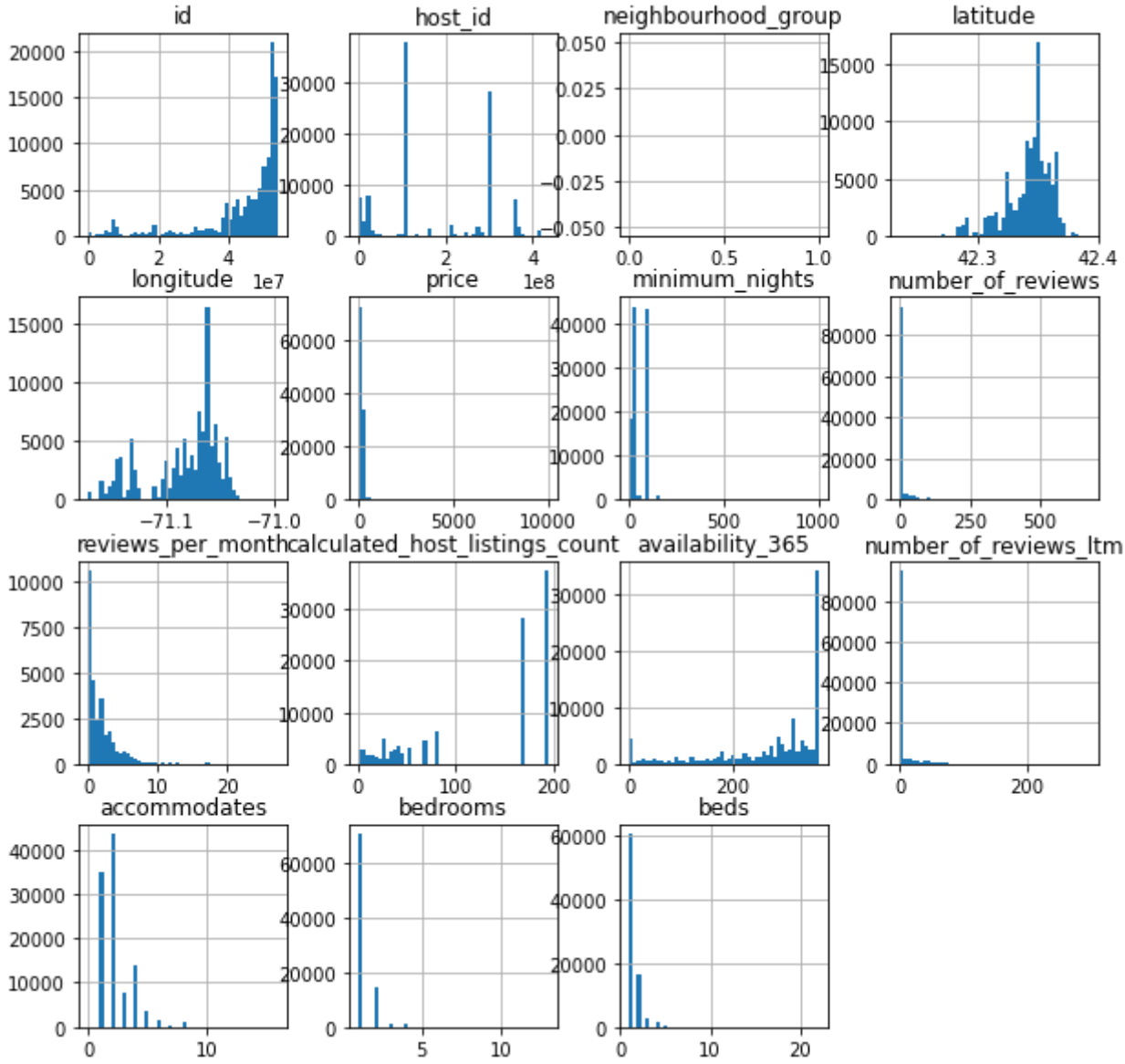
In [8]:
```
### Percentage of null rows in each column
def missing(datas):
```

```
    print (round((datas.isnull().sum() * 100/ len(datas)),2).sort_values(ascendi

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 as the
boston3.drop(['id','host_name','last_review','license',], axis=1, inplace=True)
```

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(ascendi

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

In [15]:

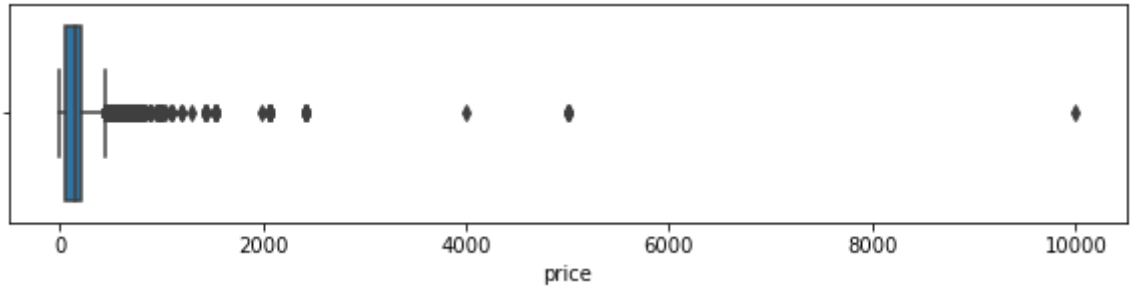
```
missing(boston3)
```

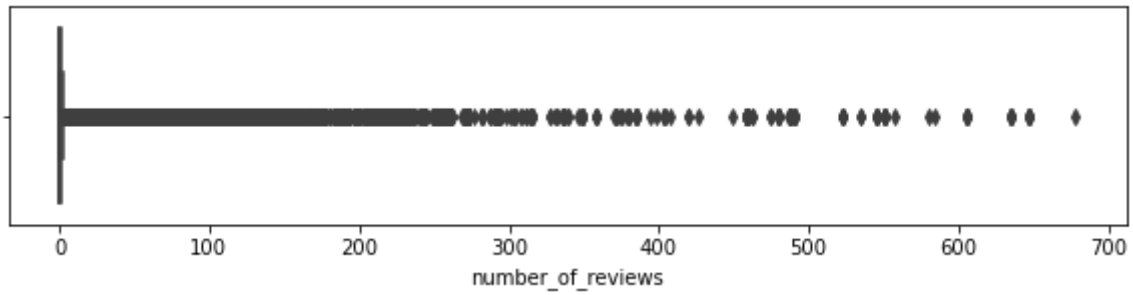
```
name                            0.0
reviews_per_month               0.0
beds                            0.0
bedrooms                        0.0
accommodates                    0.0
number_of_reviews_ltm           0.0
availability_365                0.0
calculated_host_listings_count  0.0
number_of_reviews               0.0
host_id                         0.0
minimum_nights                  0.0
price                           0.0
room_type                       0.0
```

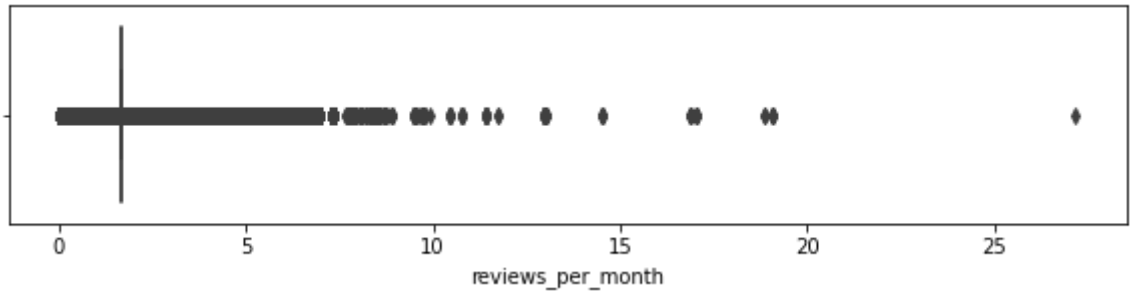
```
longitude                       0.0
latitude                        0.0
neighbourhood                   0.0
amenities                       0.0
dtype: float64
```

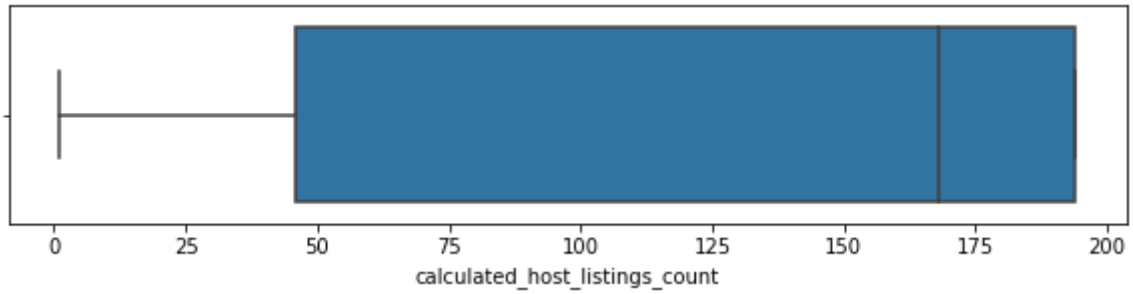
In [16]:
```
bosub = boston3[["price","number_of_reviews","reviews_per_month","calculated_hos
```

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 such th
# 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 balcon
y", "Hair dryer", "Refrigerator", "Hot water", "Dedicated workspace", "Bed linen
s", "Wifi", "Essentials", "Washer", "Air conditioning", "TV with standard cabl
e", "Oven", "Stove", "Dryer", "Smoke alarm", "Extra pillows and blankets", "Hang
ers", "Carbon monoxide alarm", "Dishwasher", "Heating", "Shampoo", "Free parking
on premises", "Iron", "Kitchen", "Coffee maker", "Microwave", "Long term stays a
llowed", "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',
 '42\\ HDTV with Roku',
 '42\\ HDTV with standard cable',
 '43\\ HDTV with Amazon Prime Video',
 '43\\ HDTV with Roku',
 '45\\ HDTV with premium cable',
 '46\\ HDTV with Amazon Prime Video',
 '48\\ HDTV',
 '48\\ HDTV with',
 '5-10 years old',
 '50\\ HDTV with',
 '50\\ HDTV with Chromecast',
 '50\\ HDTV with Roku',
 '50\\ HDTV with premium cable',
 '50\\ HDTV with standard cable',
```

```
'52\\ HDTV',
'52\\ HDTV with standard cable',
'52\\ TV with standard cable',
'54\\ HDTV with Netflix',
'54\\ HDTV with standard cable',
'55\\ HDTV',
'55\\ HDTV with Amazon Prime Video',
'55\\ HDTV with Apple TV',
'55\\ HDTV with HBO Max',
'55\\ HDTV with Netflix',
'55\\ HDTV with Roku',
'55\\ TV with premium cable',
'55\\ TV with standard cable',
'57\\ HDTV with Roku',
'60\\ HDTV with Amazon Prime Video',
'65\\ HDTV with Amazon Prime Video',
'65\\ HDTV with Chromecast',
'65\\ HDTV with Netflix',
'65\\ HDTV with standard cable',
'70\\ HDTV with Amazon Prime Video',
'75\\ HDTV with Amazon Prime Video',
'75\\ HDTV with Roku',
'75\\ TV',
':) body soap',
':) conditioner',
'Aesop body soap',
'Air conditioning',
'Alaffia',
'Amana refrigerator',
'Amazon  conditioner',
'Amazon Prime Video',
'Amazon body soap',
'Apple TV',
'Audio-Technica Record Player  sound system with Bluetooth and aux',
'Avanti stainless steel electric stove',
'Avanti stainless steel oven',
'Aveeno body soap',
'BBQ grill',
'Baby bath',
'Baby monitor',
'Baby safety gates',
'Babysitter recommendations',
'Backyard',
'Baking sheet',
'Barbecue utensils',
'Bath & Body Works body soap',
'Bath & Body Works conditioner',
'Bath & Body Works shampoo',
'Bathtub',
'Beach essentials',
'Beachfront',
'Bed linens',
'Bed sheets and pillows',
'Beekman Hotel conditioner',
'Beekman body soap',
'Beekman conditioner',
'Beekman shampoo',
'Bidet',
'Bikes',
'Bluetooth sound system',
'Bluetooth speaker',
'Bluetooth speaker Bluetooth sound system',
'Board games',
'Boat slip',
'Body soap',
```

```
 'Bosch refrigerator',
 'Bosch stainless steel oven',
 'Bose sound system with Bluetooth and aux',
 'Both bar soaps and body wash are available for you. body soap',
 'Bread maker',
 'Breakfast',
 'Breakfast buffet available \\u2014 $19 per person per day',
 'Building staff',
 'Cable TV',
 'Cambridge Soundworks / Doss (two small units) sound system with Bluetooth and
aux',
 'Carbon monoxide alarm',
 'Ceiling fan',
 'Central AC  conditioner',
 'Central air conditioning',
 'Central heating',
 'Changing table',
 'Children\\u2019s books and toys',
 'Children\\u2019s books and toys for ages 0-2 years old',
 'Children\\u2019s books and toys for ages 0-2 years old and 2-5 years old',
 'Children\\u2019s books and toys for ages 2-5 years old',
 'Children\\u2019s books and toys for ages 2-5 years old and 5-10 years old',
 'Children\\u2019s books and toys for ages 5-10 years old and 10+ years old',
 'Children\\u2019s dinnerware',
 'Chromecast',
 'Cleaning before checkout',
 'Cleaning products',
 'Clothing storage',
 'Clothing storage: closet',
 'Clothing storage: closet and dresser',
 'Clothing storage: closet and walk-in closet',
 'Clothing storage: closet and wardrobe',
 'Clothing storage: dresser',
 'Clothing storage: dresser and closet',
 'Clothing storage: dresser and walk-in closet',
 'Clothing storage: dresser and wardrobe',
 'Clothing storage: walk-in closet',
 'Clothing storage: walk-in closet and closet',
 'Clothing storage: walk-in closet and dresser',
 'Clothing storage: wardrobe',
 'Clothing storage: wardrobe and closet',
 'Clothing storage: wardrobe and dresser',
 'Coffee maker',
 'Complimentary continental breakfast',
 'Concierge',
 'Conditioner',
 'Connect your device via Bluetooth sound system',
 'Cooking basics',
 'Crib',
 'DOVE Usually body soap',
 'Dedicated workspace',
 'Dedicated workspace: desk',
 'Dedicated workspace: desk and office chair',
 'Dedicated workspace: desk and table',
 'Dedicated workspace: monitor',
 'Dedicated workspace: monitor and desk',
 'Dedicated workspace: office chair',
 'Dedicated workspace: office chair and desk',
 'Dedicated workspace: office chair and table',
 'Dedicated workspace: table',
 'Dedicated workspace: table and desk',
 'Dedicated workspace: table and office chair',
 'Dial Soap Bars body soap',
 'Dining table',
 'Dishes and silverware',
```

```
'Dishwasher',
'Dove',
'Dove body soap',
'Dove conditioner',
'Dove shampoo',
'Dryer',
'Dryer \\u2013 In building',
'Dryer \\u2013\\u00a0In unit',
'Drying rack for clothing',
'EV charger',
'Ecco sound system',
'Eco bar soap body soap',
'Electric stove',
'Elevator',
'Essentials',
'Ethernet connection',
'Extra pillows and blankets',
'Fast wifi \\u2013 100 Mbps',
'Fast wifi \\u2013 102 Mbps',
'Fast wifi \\u2013 103 Mbps',
'Fast wifi \\u2013 121 Mbps',
'Fast wifi \\u2013 157 Mbps',
'Fast wifi \\u2013 191 Mbps',
'Fast wifi \\u2013 220 Mbps',
'Fast wifi \\u2013 276 Mbps',
'Fast wifi \\u2013 302 Mbps',
'Fast wifi \\u2013 395 Mbps',
'Fast wifi \\u2013 409 Mbps',
'Fast wifi \\u2013 445 Mbps',
'Fast wifi \\u2013 51 Mbps',
'Fast wifi \\u2013 54 Mbps',
'Fast wifi \\u2013 55 Mbps',
'Fast wifi \\u2013 58 Mbps',
'Fast wifi \\u2013 59 Mbps',
'Fast wifi \\u2013 63 Mbps',
'Fast wifi \\u2013 68 Mbps',
'Fast wifi \\u2013 78 Mbps',
'Fast wifi \\u2013 86 Mbps',
'Fast wifi \\u2013 93 Mbps',
'Fast wifi \\u2013 95 Mbps',
'Fast wifi \\u2013 96 Mbps',
'Fast wifi \\u2013 97 Mbps',
'Fast wifi \\u2013 98 Mbps',
'Fenced garden or backyard',
'Fire extinguisher',
'Fire pit',
'Fireplace guards',
'First aid kit',
'Free carport on premises \\u2013 1 space',
'Free driveway parking on premises',
'Free driveway parking on premises \\u2013 1 space',
'Free driveway parking on premises \\u2013 2 spaces',
'Free dryer',
'Free dryer \\u2013 In building',
'Free dryer \\u2013 In unit',
'Free parking on premises',
'Free parking on premises \\u2013 1 space',
'Free residential garage on premises',
'Free residential garage on premises \\u2013 1 space',
'Free street parking',
'Free washer',
'Free washer \\u2013 In building',
'Free washer \\u2013 In unit',
'Free wifi',
'Freezer',
```

```
'Freshscent body soap',
'Freshscent shampoo',
'GE 20.2 CU FT refrigerator',
'GE Profile refrigerator',
'GE Profile stainless steel gas stove',
'GE Profile stainless steel oven',
'GE gas stove',
'GE oven',
'GE refrigerator',
'Gaggenau stainless steel gas stove',
'Game console',
'Game console: Nintendo Wii',
'Game console: PS2',
'Game console: PS4',
'Game console: PS5',
'Game room',
'Gas oven',
'Gas stove',
'Gel body soap',
'Gel shampoo',
'Generic conditioner',
'Gilchrist And Saomes body soap',
'Gilchrist and Soames body soap',
'Gilchrist and Somes body soap',
'Google Home Bluetooth sound system',
'Gym',
'HBO Max',
'HDTV',
'HDTV with Amazon Prime Video',
'HDTV with Apple TV',
'HDTV with Chromecast',
'HDTV with HBO Max',
'HDTV with Netflix',
'HDTV with Roku',
'HDTV with premium cable',
'HDTV with standard cable',
'Hair dryer',
'Hangers',
'Heating',
'High chair',
'Honest body soap',
'Host greets you',
'Hot tub',
'Hot water',
'Hot water kettle',
'House bikes',
'Housekeeping',
'Ikea gas stove',
'Ikea oven',
'Indoor fireplace',
'Indoor pool',
'Induction stove',
'Infuse - Made from white tea and coconut conditioner',
'Infuse made from white tea and coconut body soap',
'Infuse- made from white tea And coconut shampoo',
'Irish Spring Shower gel/Dial liquid hand soap/Tulip body soap body soap',
'Iron',
'Ivory body soap',
'KEF sound system with Bluetooth and aux',
'Kayak',
'Kenmore electric stove',
'Kenmore oven',
'Kenmore stainless steel electric stove',
'Kenmore stainless steel oven',
'Keurig coffee machine',
```

```
'Keypad',
"Kiehl's body soap",
"Kiehl's conditioner",
'Kirkland body soap',
'Kirkland conditioner',
'Kirkland shampoo',
'Kitchen',
'KitchenAid refrigerator',
'KitchenAid stainless steel gas stove',
'KitchenAid stainless steel oven',
'Kitchenaid refrigerator',
'Kitchenaid stainless steel gas stove',
'Kitchenaid stainless steel oven',
'Kitchenette',
'LG New as of 3/21 stainless steel oven',
'LG refrigerator',
'Lake access',
'Laundromat nearby',
'Laundry services',
'Limited housekeeping \\u2014',
'Limited housekeeping \\u2014 on request',
'Liquid body soap',
'Liquid conditioner',
'Liquid shampoo',
'Lock on bedroom door',
'Lockbox',
'Long term stays allowed',
'Loreal conditioner',
'Luggage dropoff allowed',
'MIlk',
'Maxxam conditioner',
'Maxxam shampoo',
'Microwave',
'Milk conditioner',
'Mini fridge',
'Mosquito net',
'Nespresso machine',
'Netflix',
'Neutrogena or Kirkland body soap',
'Nexus or Kirkland shampoo',
'No stove but has a hot plate electric stove',
'Not sure  body soap',
'Olay Body Wash with Retinol',
'Olay body soap',
'Onsite restaurant \\u2014 Cosmica',
'Onsite restaurant \\u2014 canteenM-open 24/7 Open 24/7',
'Organic  body soap',
'Organic 365% and EVERYBODY shampoo',
'Organic conditioner',
'Organic shampoo',
'Oster Toaster Oven oven',
'Outdoor dining area',
'Outdoor furniture',
'Outdoor shower',
'Outlet covers',
'Oven',
'PANTENE Usually conditioner',
'PANTENE Usually shampoo',
'Pack \\u2019n play/Travel crib',
'Paid dryer',
'Paid dryer \\u2013 In building',
'Paid dryer \\u2013 In unit',
'Paid parking garage off premises',
'Paid parking garage on premises',
'Paid parking garage on premises \\u2013 1 space',
```

```
'Paid parking lot off premises',
'Paid parking lot on premises',
'Paid parking lot on premises \\u2013 1 space',
'Paid parking lot on premises \\u2013 2 spaces',
'Paid parking lot on premises \\u2013 3 spaces',
'Paid parking lot on premises \\u2013 4 spaces',
'Paid parking lot on premises \\u2013 6 spaces',
'Paid parking off premises',
'Paid parking on premises',
'Paid parking on premises \\u2013 1 space',
'Paid street parking off premises',
'Paid valet parking on premises',
'Paid washer',
'Paid washer \\u2013 In building',
'Paid washer \\u2013 In unit',
'Panasonic sound system',
'Pantene  conditioner',
'Pantene conditioner',
'Pantene shampoo',
'Patio or balcony',
'Pets allowed',
'Piano',
'Ping pong table',
'Pocket wifi',
'Polk Bluetooth sound system',
'Pool',
'Pool table',
'Portable air conditioning',
'Portable fans',
'Portable heater',
'Pour-over coffee',
'Private entrance',
'Private fenced garden or backyard',
'Private garden or backyard',
'Private gym in building',
'Private hot tub',
'Private outdoor lap pool',
'Private patio or balcony',
'Private pool',
'Private sauna',
'Purchased in 2020. refrigerator',
'Radiant heating',
'Radio and CD player sound system',
'Record player',
'Refrigerator',
'Rice maker',
'Rikoko conditioner',
'Roku',
'Room-darkening shades',
'SONOS Bluetooth sound system',
'Safe',
'Samsung  gas stove',
'Samsung  stainless steel gas stove',
'Samsung  stainless steel oven',
'Samsung french door refrigerator',
'Samsung oven',
'Samsung refrigerator',
'Samsung stainless steel gas stove',
'Samsung stainless steel oven',
'Security cameras on property',
'Self-parking \\u2014 $42/stay',
'Shampoo',
'Shared fenced garden or backyard',
'Shared garden or backyard',
'Shared gym in building',
```

```
'Shared gym nearby',
'Shared hot tub',
'Shared outdoor heated saltwater pool',
'Shared outdoor pool',
'Shared patio or balcony',
'Shared pool',
'Shared refrigerator',
'Shower gel',
'Single level home',
'Ski-in/Ski-out',
'Slippers',
'Small fridge in room. Larger fridge in kitchenette  refrigerator',
'Smart lock',
'Smoke alarm',
'SoapBox Sea Minerals & Blue Iris body soap',
'SoapBox Tea Tree Clean & Purify conditioner',
'SoapBox Tea Tree Clean & Purify shampoo',
'Softsoap body soap',
'Sono sound system',
'Sonos  sound system',
'Sonos Bluetooth sound system',
'Sonos and Amazon Tap Bluetooth sound system',
'Sonos sound system',
'Sonos sound system with Bluetooth and aux',
'Sonos wifi sound system',
'Sound system',
'Sound system with Bluetooth and aux',
'Sound system with aux',
'Stainless steel electric stove',
'Stainless steel gas stove',
'Stainless steel oven',
'Stainless steel refrigerator',
'Stainless steel stove',
'Stove',
'Suave body soap',
'TRESemm\\u00e9 conditioner',
'TRESemm\\u00e9 shampoo',
'TV',
'TV with Amazon Prime Video',
'TV with Apple TV',
'TV with Chromecast',
'TV with HBO Max',
'TV with Netflix',
'TV with Roku',
'TV with standard cable',
'Table corner guards',
'Teenage Engineering Bluetooth sound system',
'There are two induction burners with pots and pans. induction stove',
'There is a 24\\ Blomberg fridge/freezer in the unit. refrigerator',
'Thermador  stainless steel electric stove',
'Toaster',
'Toiletries',
'Toshiba with Boss Speakers sound system with Bluetooth and aux',
'Trash compactor',
'Tresemme  conditioner',
'Tresemme shampoo',
'Varies conditioner',
'Various body soap',
'Various conditioner',
'WHIRLPOOL refrigerator',
'WHIRPOOL refrigerator',
'WHRILPOOL refrigerator',
'Washer',
'Washer \\u2013\\u00a0In building',
'Washer \\u2013\\u00a0In unit',
```

```
 'Waterfront',
 'Whirlpool  refrigerator',
 'Whirlpool refrigerator',
 'Whirlpool stainless steel oven',
 'Whole Foods body soap',
 'Wifi',
 'William Roam toiletries',
 'Window AC unit',
 'Window guards',
 'Wine Mini Fridge (For Beverages Only) refrigerator',
 'Wine glasses',
 'Your choice 100% Organic or Skin Sensitive no scent - MILK Brand body soap',
 'and 10+ years old',
 'and 5-10 years old',
 'and Axe Body Wash body soap',
 'and closet',
 'and desk',
 'and dresser',
 'and goodies for sharing. Our baristas will make you a perfect coffee all day
(and night) long',
 'and in the evenings they transform into expert mixologists!',
 'and monitor',
 'and office chair',
 'and table',
 'and walk-in closet',
 'bose sound system with Bluetooth and aux',
 'canteenM is our casual and cosy kitchen',
 'closet',
 'desk',
 'dove  conditioner',
 'dove body soap',
 'dresser',
 'google assistant sound system',
 'healthy snacks',
 'hypoallergenic body soap',
 'linens',
 'mini fridge refrigerator',
 'minifridge refrigerator',
 'monitor',
 'offering a choice of homemade meals',
 'office chair',
 'premium cable',
 'something nice! :) body soap',
 'standard cable',
 'table',
 'toiletries',
 'trader joes conditioner',
 'under counter refrigerator',
 'vegan and cruelty-free body soap',
 'vegan and cruelty-free conditioner',
 'vegan and cruelty-free shampoo',
 'walk-in closet',
 'wardrobe',
 'whirlpool  oven'}
```

In [21]:

```
boston3.loc[boston3['amenities'].str.contains('Air conditioning|Central air cond
boston3.loc[boston3['amenities'].str.contains('Gym|24-hour fitness center|Privat
boston3.loc[boston3['amenities'].str.contains('Apple TV|Game console|Game consol
boston3.loc[boston3['amenities'].str.contains('Avanti stainless steel electric s
boston3.loc[boston3['amenities'].str.contains('Electric stove|Gas stove|Ikea gas
boston3.loc[boston3['amenities'].str.contains('BBQ grill|Fire pit|Barbecue utens
boston3.loc[boston3['amenities'].str.contains('balcony|Patio or balcony|Patio'),
boston3.loc[boston3['amenities'].str.contains('Beach view|Beachfront|Lake access
boston3.loc[boston3['amenities'].str.contains('Bed linens|Bed sheets and pillows
```

```
boston3.loc[boston3['amenities'].str.contains('Breakfast|Complimentary continent
boston3.loc[boston3['amenities'].str.contains('Coffee maker|Espresso machine|Cof
boston3.loc[boston3['amenities'].str.contains('Cooking basics'), 'cooking_basics
boston3.loc[boston3['amenities'].str.contains('Dryer|Dryer \\u2013 In building|D
boston3.loc[boston3['amenities'].str.contains('Elevator'), 'elevator'] = 1
boston3.loc[boston3['amenities'].str.contains('Children\\u2019s books and toys|C
boston3.loc[boston3['amenities'].str.contains('Free driveway parking on premises
boston3.loc[boston3['amenities'].str.contains('Private fenced garden or backyard
boston3.loc[boston3['amenities'].str.contains('Host greets you'), 'host_greeting
boston3.loc[boston3['amenities'].str.contains('Private hot tub|Shared outdoor he
boston3.loc[boston3['amenities'].str.contains('Internet|Pocket wifi|Wifi|Pocket
boston3.loc[boston3['amenities'].str.contains('Long term stays allowed'), 'long_
boston3.loc[boston3['amenities'].str.contains('Private entrance'), 'private_entr
boston3.loc[boston3['amenities'].str.contains('Pets|pet|Cat(s)|Dog(s)|Pets allow
boston3.loc[boston3['amenities'].str.contains('Safe|Security system|Security cam
boston3.loc[boston3['amenities'].str.contains('Self check_in'), 'self_check_in']
boston3.loc[boston3['amenities'].str.contains('Smoking allowed'), 'smoking_allow
boston3.loc[boston3['amenities'].str.contains('Step-free access|Wheelchair|Acces
boston3.loc[boston3['amenities'].str.contains('Suitable for events'), 'event_sui
boston3.loc[boston3['amenities'].str.contains('Aveeno body soap|Bathtub|Beekman
boston3.loc[boston3['amenities'].str.contains('Sonos Bluetooth sound system|Sono
boston3.loc[boston3['amenities'].str.contains('Dedicated workspace|Dedicated wor
boston3.loc[boston3['amenities'].str.contains('Freezer|GE refrigerator| LG refri
boston3.loc[boston3['amenities'].str.contains('Onsite restaurant \\u2014 Cosmica
```

```
<ipython-input-21-90ad71994ea5>:23: UserWarning: This pattern has match groups.
To actually get the groups, use str.extract.
  boston3.loc[boston3['amenities'].str.contains('Pets|pet|Cat(s)|Dog(s)|Pets all
owed'), '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
```

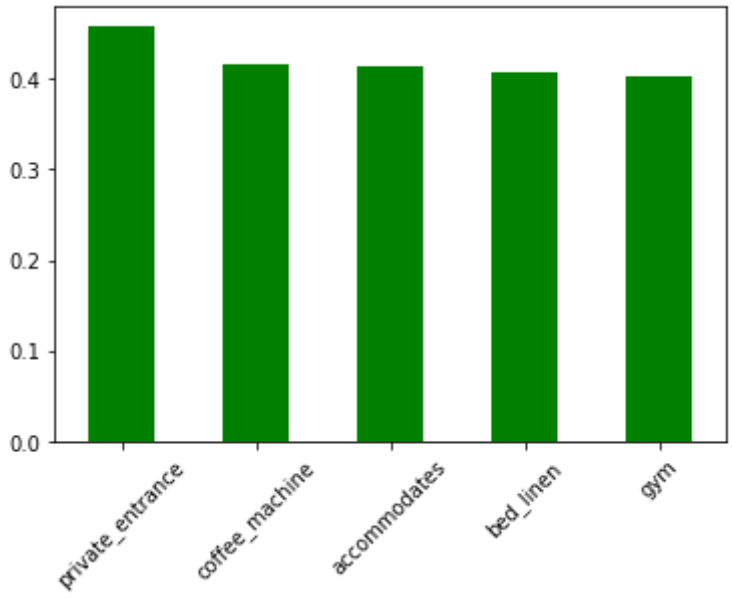
Data Visualization

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

In [25]:
```
price_corr = boston3.corr()['price'].sort_values(ascending = False).head(6)[1:]
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()
```

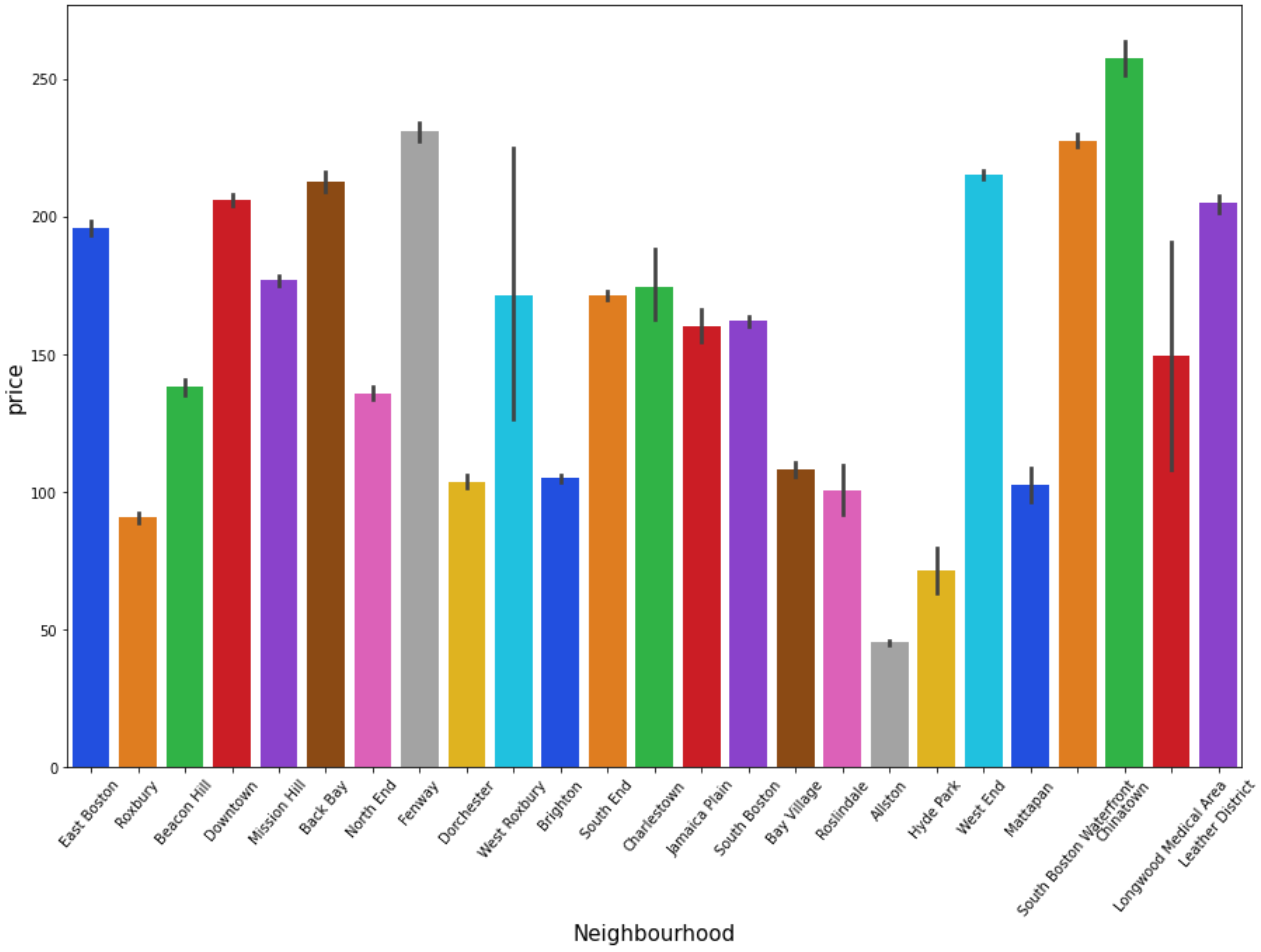
Out[27]:

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

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)
```

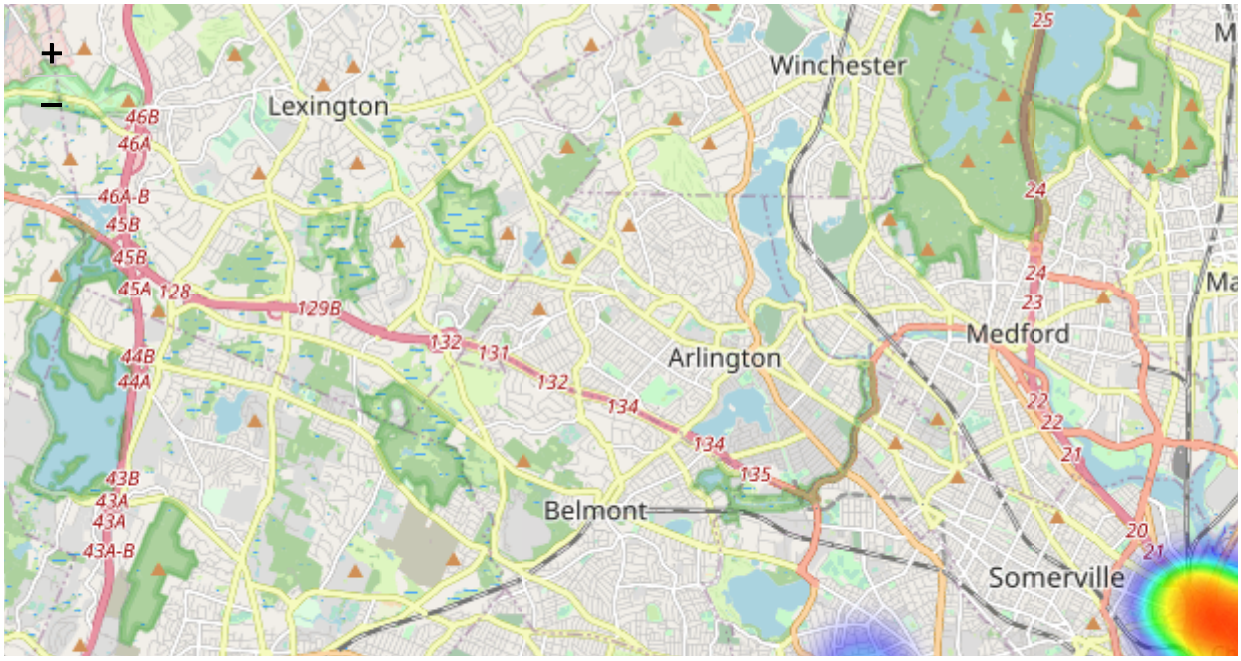
Out[28]:
```
Text(0, 0.5, 'price')
```



In [29]:

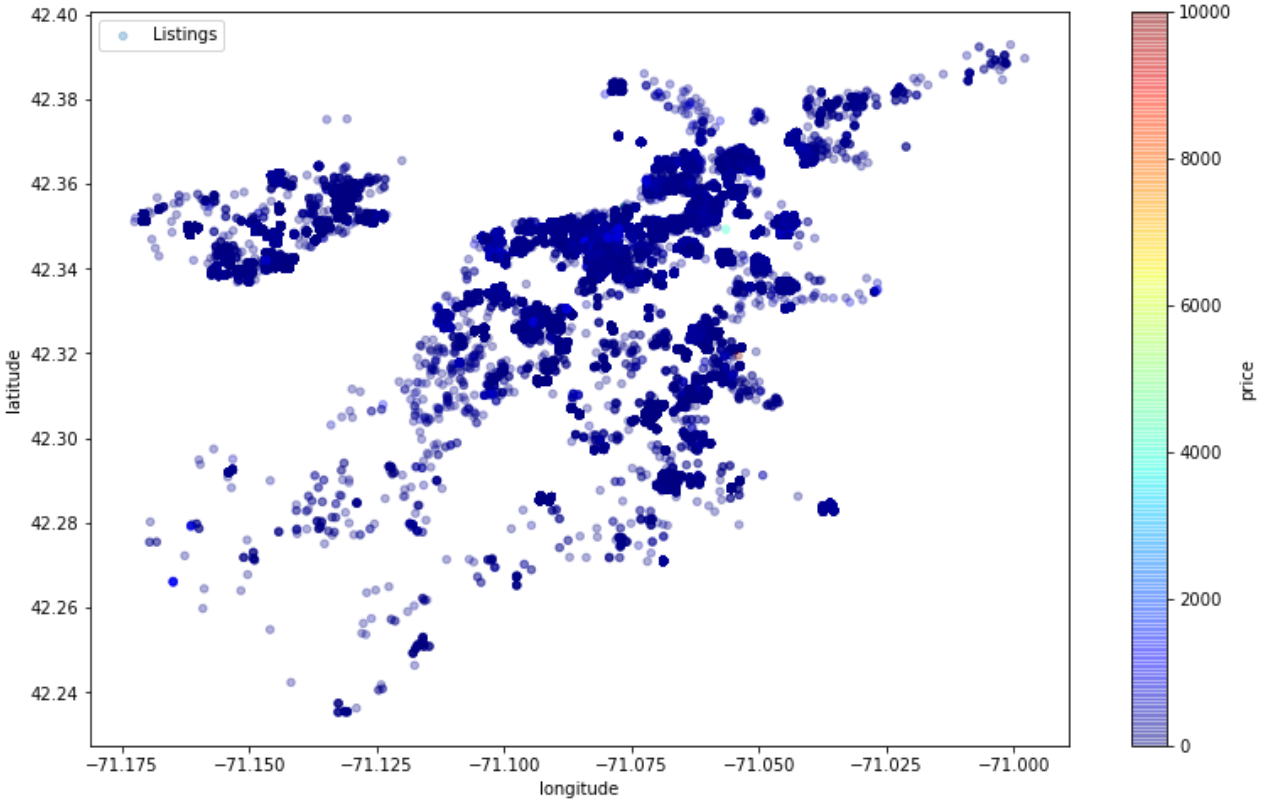
```
maps = folium.Map(location=[42.361145, -71.057083], zoom_start = 12)

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

htm =  HeatMap(heatmap_data).add_to(maps)

maps
```

Out[29]:



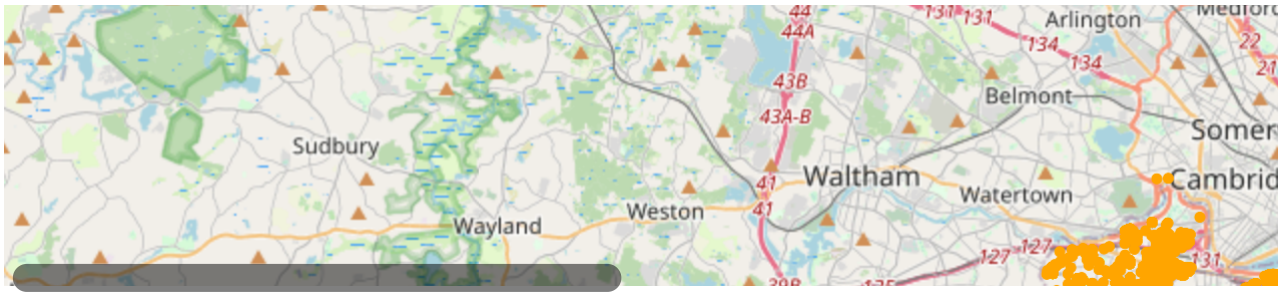
In [30]:

```
from PIL import Image

plt.figure(figsize=(13,8))
gx = plt.gca()
boston3.plot(kind='scatter',x='longitude',y='latitude',label='Listings', c='pric
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_name="na
                        color_discrete_sequence=["orange"], zoom=10, height=300)
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_
count	1.073890e+05	107389.000000	107389.000000	107389.00000	107389.000000	107389
mean	1.758024e+08	42.342998	-71.084340	147.84707	51.843345	1
std	1.221205e+08	0.019351	0.033584	116.54091	36.352791	38
min	4.804000e+03	42.235330	-71.172520	0.00000	1.000000	0
25%	1.074344e+08	42.335150	-71.101850	50.00000	30.000000	0
50%	1.074344e+08	42.347970	-71.070840	149.00000	32.000000	0
75%	2.978601e+08	42.355180	-71.061640	212.00000	91.000000	1
max	4.349904e+08	42.392790	-70.997810	10000.00000	1000.000000	678

8 rows × 30 columns

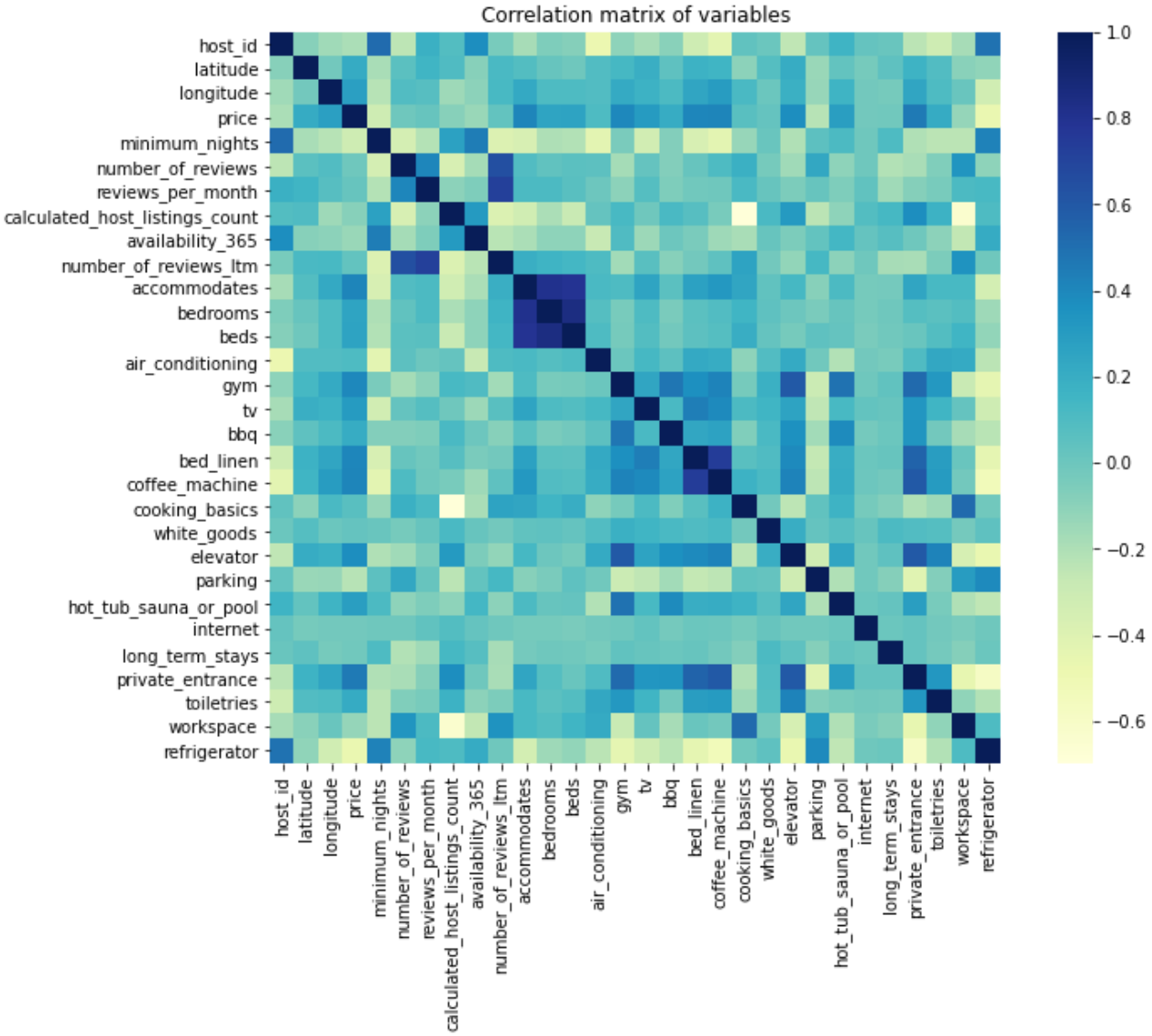
In [33]:
```
boston3.info()
```

```
<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: 31.9+ 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()
```



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_listi
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.shape[

    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

	variables	VIF
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)
```

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

Out[39]:

	availability_365	accommodates	bedrooms	beds	air_conditioning	gy
availability_365	1.000000	-0.186957	-0.104487	-0.106226	-0.273973	0.1041
accommodates	-0.186957	1.000000	0.811684	0.791003	0.124014	0.1096
bedrooms	-0.104487	0.811684	1.000000	0.852407	0.084183	-0.0345
beds	-0.106226	0.791003	0.852407	1.000000	0.105523	-0.0304
air_conditioning	-0.273973	0.124014	0.084183	0.105523	1.000000	0.0067
gym	0.104145	0.109612	-0.034519	-0.030464	0.006799	1.0000
tv	-0.145792	0.253955	0.109999	0.086498	0.136304	0.2475
bbq	0.005453	0.062040	-0.041420	-0.014476	0.057425	0.4728
bed_linen	-0.042461	0.274648	0.065092	0.055282	0.228350	0.3652
coffee_machine	-0.149284	0.323299	0.093667	0.080691	0.209567	0.4186
cooking_basics	-0.183022	0.247697	0.158002	0.203304	-0.103488	-0.0318
white_goods	0.024872	0.044392	0.050702	0.030545	0.092243	0.1888
elevator	-0.049019	0.132249	0.005295	-0.031894	0.223253	0.6007
parking	0.033534	-0.086699	-0.045489	0.051400	-0.011805	-0.2993

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 25 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   host_id                         107382 non-null  int64  
 1   price                           107382 non-null  int64  
 2   minimum_nights                  107382 non-null  int64  
 3   number_of_reviews               107382 non-null  int64  
 4   reviews_per_month               107382 non-null  float64
 5   calculated_host_listings_count  107382 non-null  int64  
 6   availability_365                107382 non-null  int64  
 7   accommodates                    107382 non-null  int64  
 8   bedrooms                        107382 non-null  float64
 9   beds                            107382 non-null  float64
 10  air_conditioning                107382 non-null  float64
 11  gym                             107382 non-null  float64
 12  tv                              107382 non-null  float64
 13  bbq                             107382 non-null  float64
 14  bed_linen                       107382 non-null  float64
 15  coffee_machine                  107382 non-null  float64
 16  cooking_basics                  107382 non-null  float64
 17  white_goods                     107382 non-null  float64
 18  elevator                        107382 non-null  float64
 19  parking                         107382 non-null  float64
 20  hot_tub_sauna_or_pool           107382 non-null  float64
 21  private_entrance                107382 non-null  float64
 22  toiletries                      107382 non-null  float64
 23  workspace                       107382 non-null  float64
 24  refrigerator                    107382 non-null  float64
dtypes: float64(18), int64(7)
memory usage: 21.3 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"],axis=
```

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,DecisionTreeRegressor(ra
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 Tree'
tab.set_index('Actual Values', inplace=True)
tab
```

```
DecisionTreeRegressor {'MAE': 10.810399215347308, 'R2': 0.8635165281256281, 'RMS
E': 39.29287339266401}
Execution time: 0.0217893123626709 ms
```

Out[45]:

Actual Values	Decision Tree
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

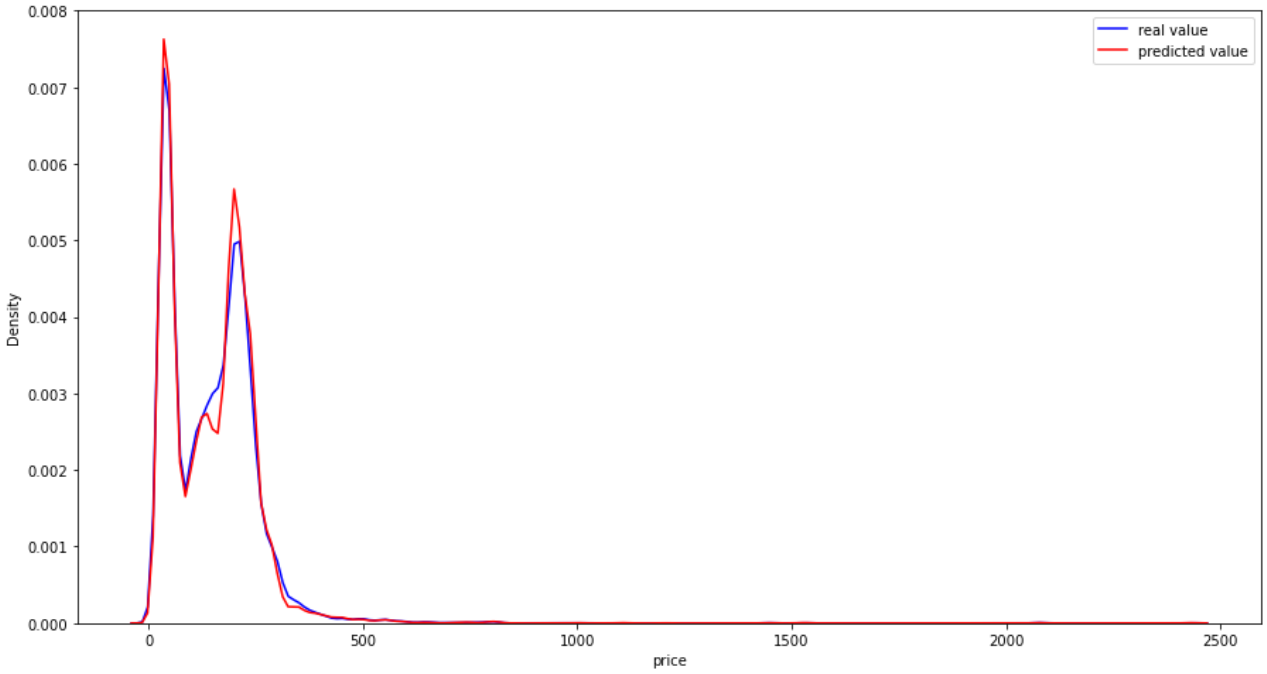
In [46]:

```
### 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/distributio
ns.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Ple
ase adapt your code to use either `displot` (a figure-level function with simila
r flexibility) or `kdeplot` (an axes-level function for kernel density plots).

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

`distplot` is a deprecated function and will be removed in a future version. Ple
ase adapt your code to use either `displot` (a figure-level function with simila
r flexibility) or `kdeplot` (an axes-level function for kernel density plots).
```

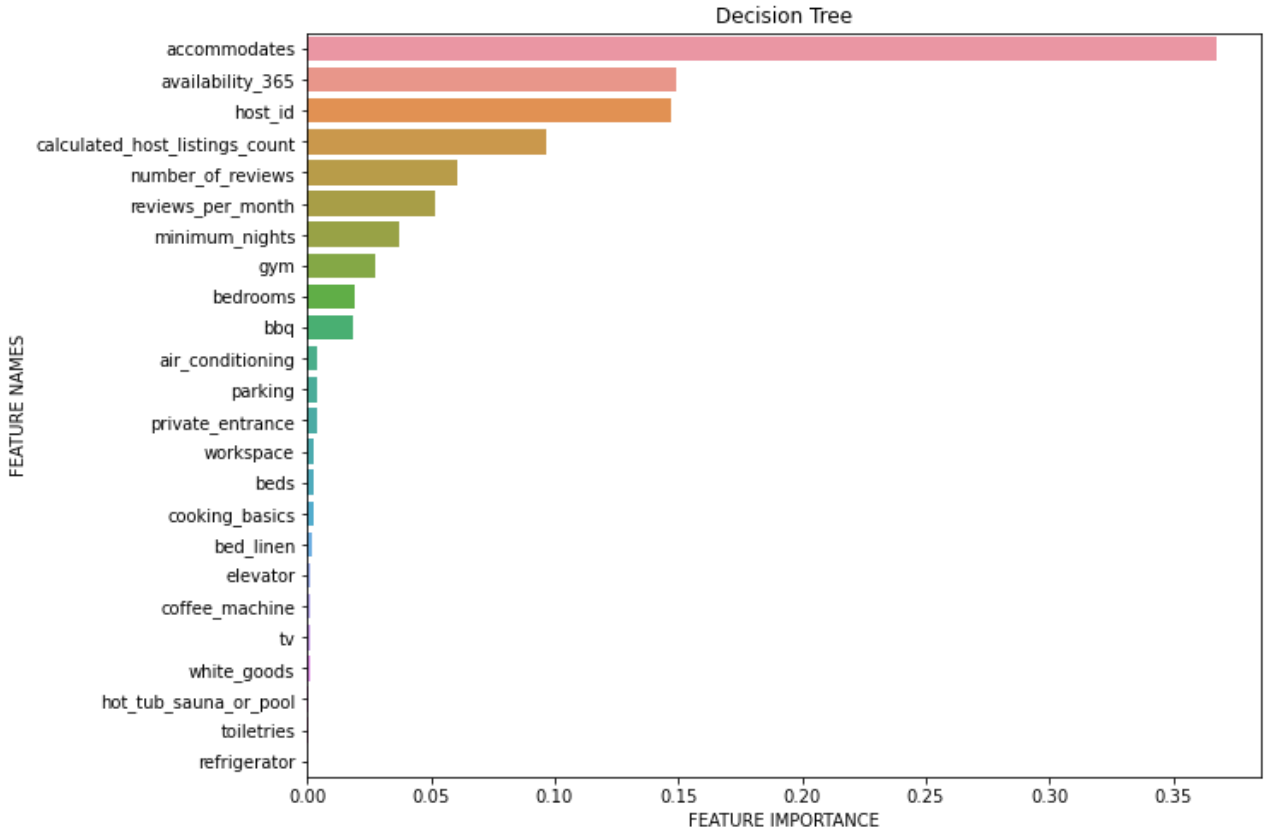
Out[46]: <matplotlib.legend.Legend at 0x7f817d0e0a30>



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_importance}
    fi_df = pd.DataFrame(data)
    fi_df.sort_values(by=['feature_importance'], ascending=False,inplace=True)
    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,'Decision Tr
```



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: [0.86733081 0.87912521 0.87162044 0.79435536 0.84334319 0.72238655
 0.83341221]
Mean Score 0.8302248230311889
```

Random Forest Regressor

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

model2 = RandomForestRegressor = train_model(x_processed,y_train,RandomForestReg
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': 10.593758473669945, 'R2': 0.925983382025452, 'RMS
```

```
E': 28.935992063394828}
Execution time: 0.750648021697998 ms
```

Out[49]:

Actual Values	Decision Tree	Random Forest Regressor
34	33.969697	33.973765
203	201.531915	201.576361
240	240.000000	240.000000
128	128.000000	128.000000
131	131.000000	131.000000
341	341.000000	341.000000
40	42.285714	42.255147
247	247.000000	247.000000
52	52.000000	52.000000
34	35.338462	35.338416
299	299.000000	299.300000
196	196.000000	196.000000
177	214.333333	213.871968
120	120.000000	120.000000
181	181.000000	181.150000
205	179.800000	180.685886
194	187.666667	186.255269
132	132.000000	132.000000
277	277.000000	277.000000
60	60.000000	60.000000

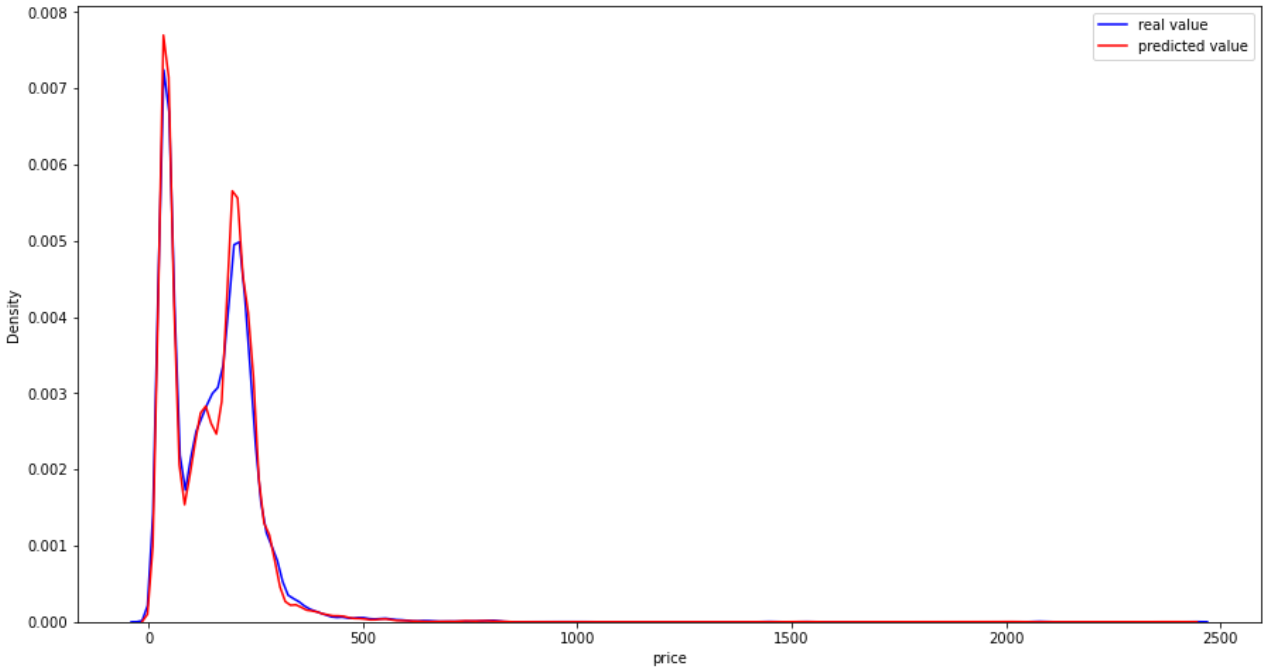
In [50]:

```
### 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/distributio
ns.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Ple
ase adapt your code to use either `displot` (a figure-level function with simila
r flexibility) or `kdeplot` (an axes-level function for kernel density plots).

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

`distplot` is a deprecated function and will be removed in a future version. Ple
ase adapt your code to use either `displot` (a figure-level function with simila
```

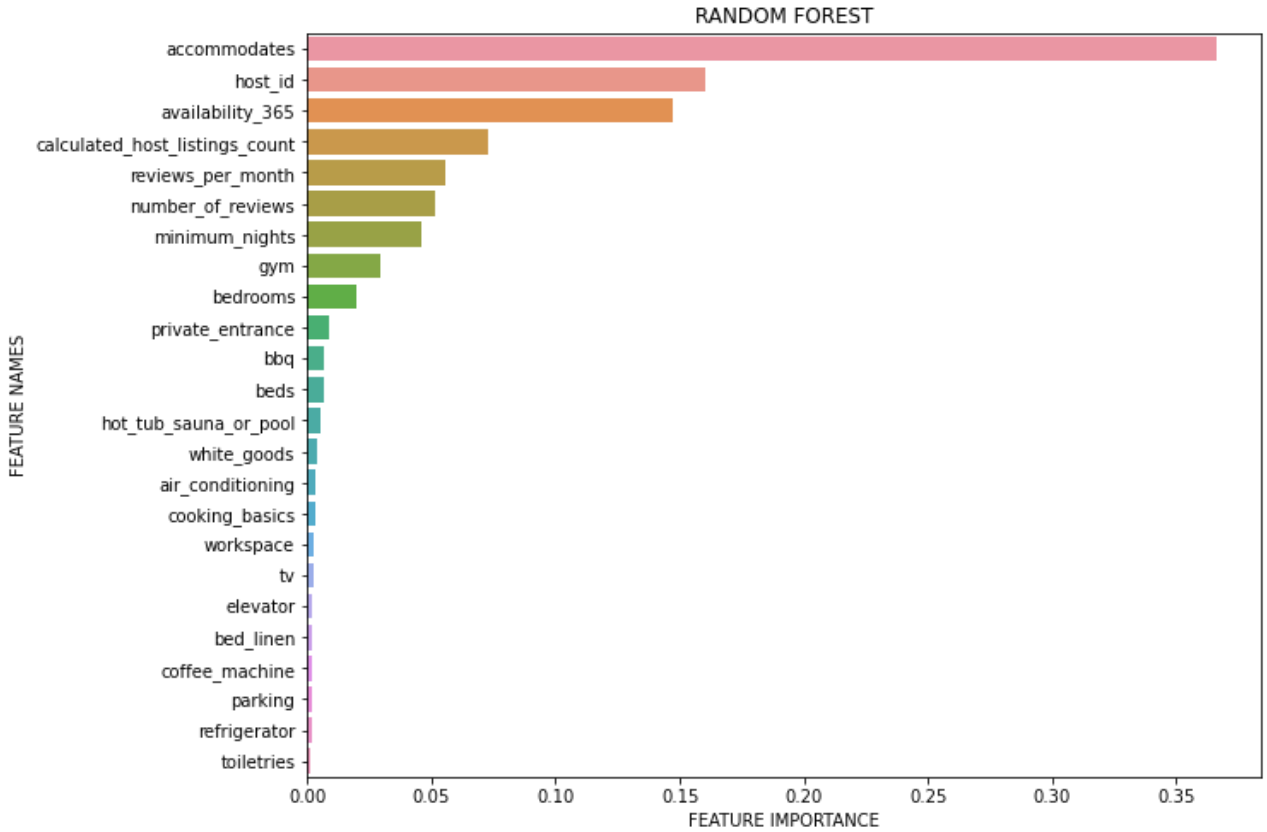
```
r flexibility) or `kdeplot` (an axes-level function for kernel density plots).
```

Out[50]: <matplotlib.legend.Legend at 0x7f80d895eeb0>



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_importance}
    fi_df = pd.DataFrame(data)
    fi_df.sort_values(by=['feature_importance'], ascending=False,inplace=True)
    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 FORE
```



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
```

```
/Users/miravparekh/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selec
tion/_search.py:918: UserWarning:

One or more of the test scores are non-finite: [nan nan nan nan nan nan nan nan
nan nan nan nan nan nan nan nan nan nan]
```

Out[55]:
```
{'bootstrap': True,
 'max_depth': 80,
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 780}
```

In [56]:
```
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)
```

Out[56]:
```
RandomForestRegressor(max_depth=80, n_estimators=780, random_state=42)
```

In [57]:
```
ss= Rf.predict(x_test)
ssg = score(y_test,ss)
print(type(Rf).__name__,ssg)
```

```
RandomForestRegressor {'MAE': 10.57967267585481, 'R2': 0.926752322489496, 'RMS
E': 28.78529519626836}
```

Cross Validation

In [58]:
```
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))
```

```
scores: [0.91434573 0.93405282 0.91353045 0.93526381 0.90861115 0.83945689
 0.92384953]
Mean Score 0.9098729108510354
```

XGBoost

In [59]:
```
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
```

```
XGBRegressor {'MAE': 15.126933598781102, 'R2': 0.9155190681137386, 'RMSE': 30.91
3850760056885}
Execution time: 0.0595247745513916 ms
```

Out[59]:

Actual Values	Decision Tree	Random Forest Regressor	XGBoost
34	33.969697	33.973765	35.467617
203	201.531915	201.576361	197.058044
240	240.000000	240.000000	226.534836
128	128.000000	128.000000	148.631653
131	131.000000	131.000000	135.591751
341	341.000000	341.000000	334.073975
40	42.285714	42.255147	44.872803
247	247.000000	247.000000	224.099289
52	52.000000	52.000000	53.537823
34	35.338462	35.338416	37.944340
299	299.000000	299.300000	318.489502
196	196.000000	196.000000	192.045746
177	214.333333	213.871968	214.736115
120	120.000000	120.000000	128.055267
181	181.000000	181.150000	184.120438
205	179.800000	180.685886	185.990601
194	187.666667	186.255269	197.347580
132	132.000000	132.000000	142.325180
277	277.000000	277.000000	257.914093
60	60.000000	60.000000	60.759827

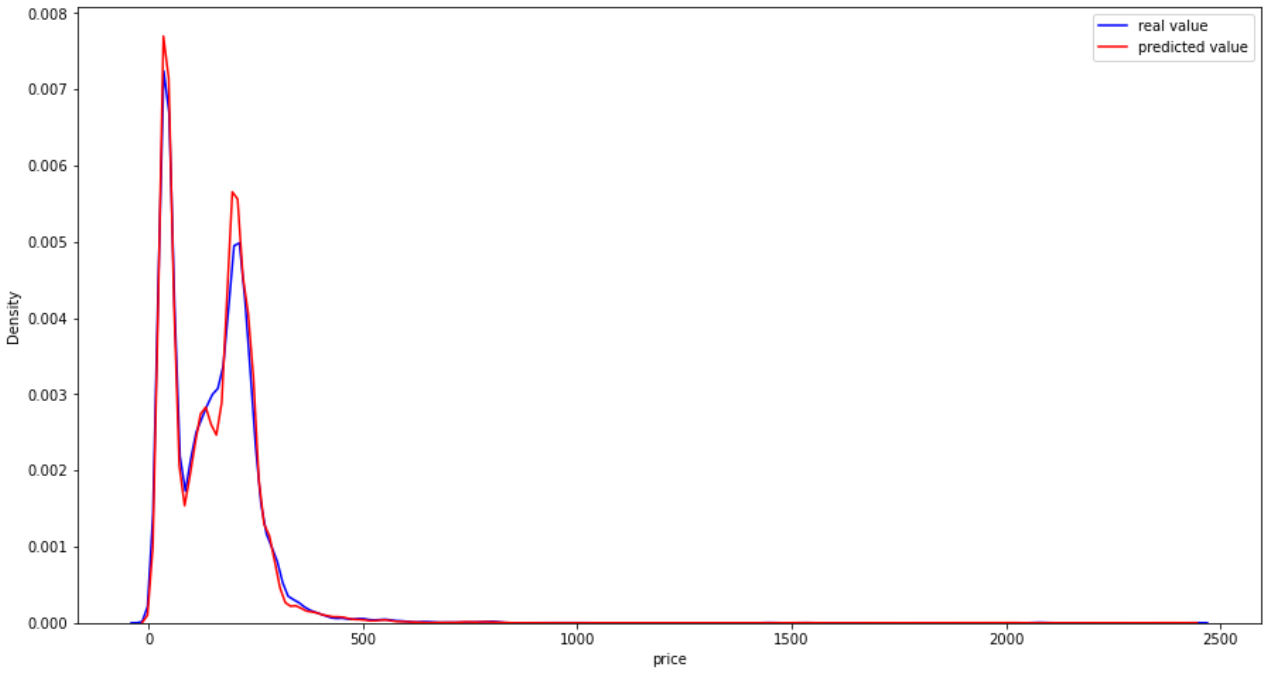
In [60]:

```
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/distributio
ns.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Ple
ase adapt your code to use either `displot` (a figure-level function with simila
r flexibility) or `kdeplot` (an axes-level function for kernel density plots).
```

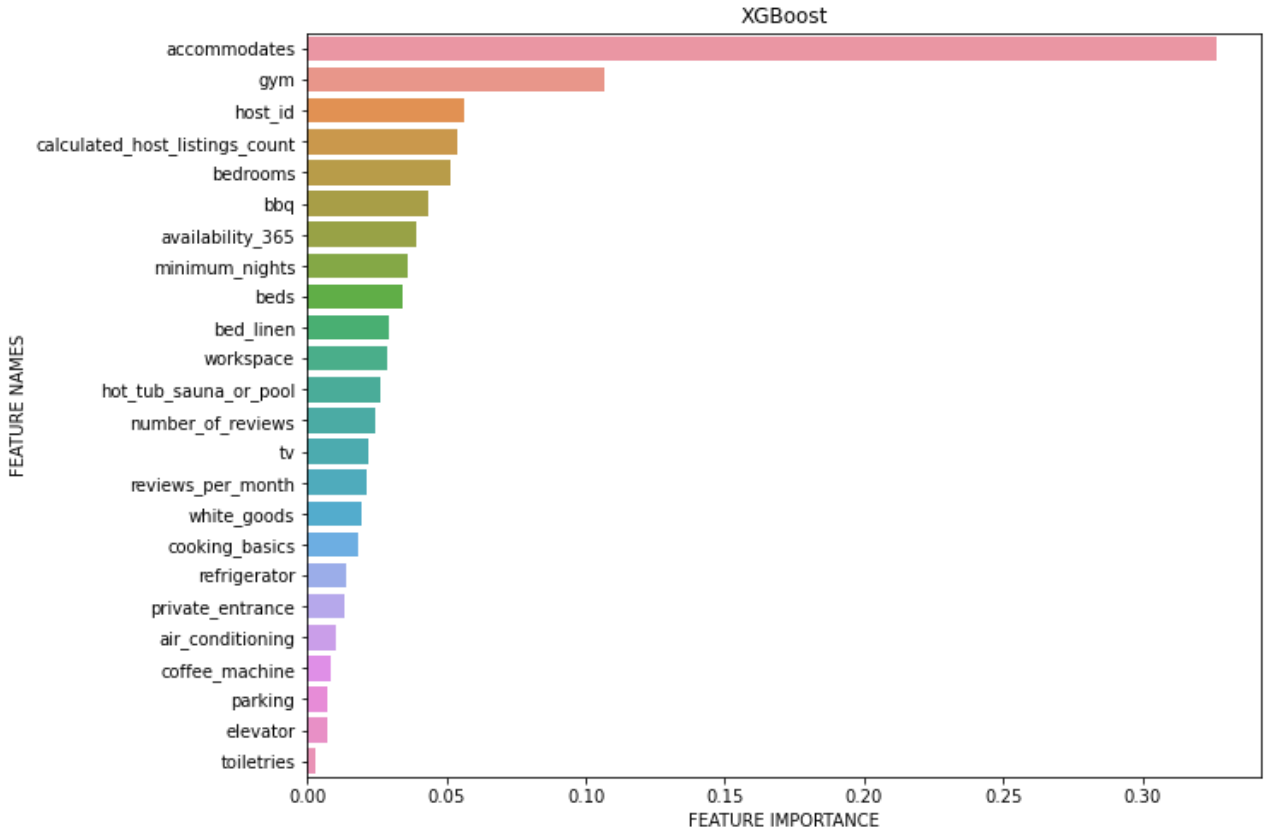
```
/Users/miravparekh/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributio
ns.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Ple
ase adapt your code to use either `displot` (a figure-level function with simila
r flexibility) or `kdeplot` (an axes-level function for kernel density plots).
```

Out[60]: <matplotlib.legend.Legend at 0x7f81501dc670>



Feature Importance

In [61]:
```
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_importance}
    fi_df = pd.DataFrame(data)
    fi_df.sort_values(by=['feature_importance'], ascending=False,inplace=True)
    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,'XGBoost')
```



Hyperparameter Tuning

In [62]:
```
import xgboost as xgb
xg = xgb.XGBRegressor(random_state=42)
xg.fit(x_train, y_train)
```

Out[62]:
```
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=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=8, num_parallel_tree=1, random_state=42,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

In [63]:
```
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 [64]:
```
gs = GridSearchCV(estimator = xg,
                            param_grid = param_grid,
                            cv = 2, n_jobs = -1, verbose = 2,
                            scoring = 'accuracy')
```

In [65]:
```
gs.fit(x_train, y_train)
gs.best_params_
```

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

```
/Users/miravparekh/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selec
tion/_search.py:918: UserWarning:

One or more of the test scores are non-finite: [nan nan nan nan nan nan nan nan
nan nan nan nan nan nan nan nan nan nan nan nan]
```

Out[65]:
```
{'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 100}
```

In [66]:
```
from xgboost import XGBRegressor
xg = XGBRegressor(random_state = 42, learning_rate = 0.1, max_depth = 5, n_estim

xg.fit(x_train,y_train)
```

Out[66]:
```
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=0.1, max_delta_step=0, max_depth=5,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=8, num_parallel_tree=1, random_state=42,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

In [67]:
```
xj= xg.predict(x_train)
dsx = score(y_train,xj)
print(type(xg).__name__,dsx)
```

```
XGBRegressor {'MAE': 22.300058190926986, 'R2': 0.8558736323563431, 'RMSE': 40.50
715142320512}
```

Cross Validation

In [68]:
```
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))
```

```
scores: [0.91779301 0.90600384 0.90688503 0.92982838 0.92023318 0.84506543
 0.90491184]
Mean Score 0.9043886721051678
```

Linear Regression

In [69]:
```
from sklearn.linear_model import LinearRegression

model4 = linearRegression = train_model(x_processed,y_train,LinearRegression())
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': 41.38232020752309, 'R2': 0.48145774616298775, 'RMSE': 7
6.58890027046596}
Execution time: 0.05413365364074707 ms
```

Out[69]:

Actual Values	Decision Tree	Random Forest Regressor	XGBoost	Linear Regression
34	33.969697	33.973765	35.467617	61.381180
203	201.531915	201.576361	197.058044	208.086258
240	240.000000	240.000000	226.534836	203.392866
128	128.000000	128.000000	148.631653	193.283456
131	131.000000	131.000000	135.591751	153.006542
341	341.000000	341.000000	334.073975	206.762669
40	42.285714	42.255147	44.872803	48.021129
247	247.000000	247.000000	224.099289	208.063210
52	52.000000	52.000000	53.537823	71.150062
34	35.338462	35.338416	37.944340	103.245866
299	299.000000	299.300000	318.489502	253.415464
196	196.000000	196.000000	192.045746	206.856012
177	214.333333	213.871968	214.736115	210.600238
120	120.000000	120.000000	128.055267	185.310023
181	181.000000	181.150000	184.120438	198.661646
205	179.800000	180.685886	185.990601	205.565172
194	187.666667	186.255269	197.347580	202.862957
132	132.000000	132.000000	142.325180	198.123181
277	277.000000	277.000000	257.914093	173.444394
60	60.000000	60.000000	60.759827	142.124072

Cross Validation

In [70]:

```
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))
```

```
scores: [0.45170104 0.47914104 0.48783615 0.48341618 0.45492606 0.45007964
 0.53101433]
Mean Score 0.4768734896931458
```

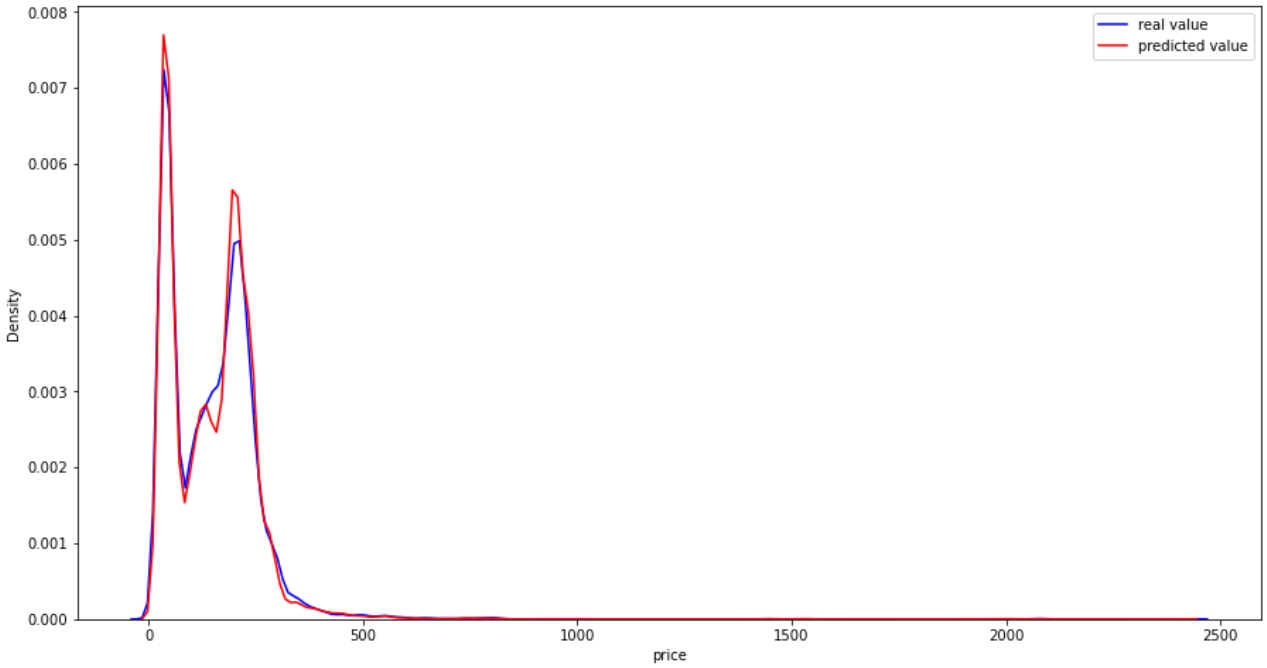
In [71]:

```
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/distributio
ns.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Ple
ase adapt your code to use either `displot` (a figure-level function with simila
r flexibility) or `kdeplot` (an axes-level function for kernel density plots).

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

`distplot` is a deprecated function and will be removed in a future version. Ple
ase adapt your code to use either `displot` (a figure-level function with simila
r flexibility) or `kdeplot` (an axes-level function for kernel density plots).
```

Out[71]: <matplotlib.legend.Legend at 0x7f81514f8ca0>



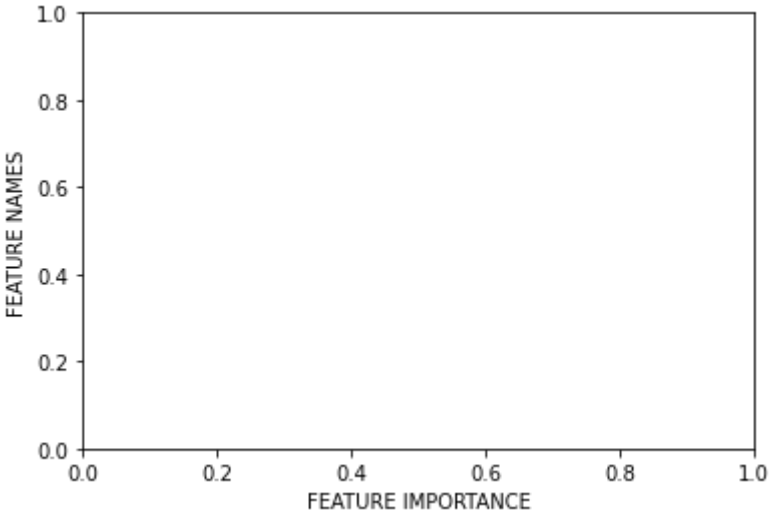
Feature Importance

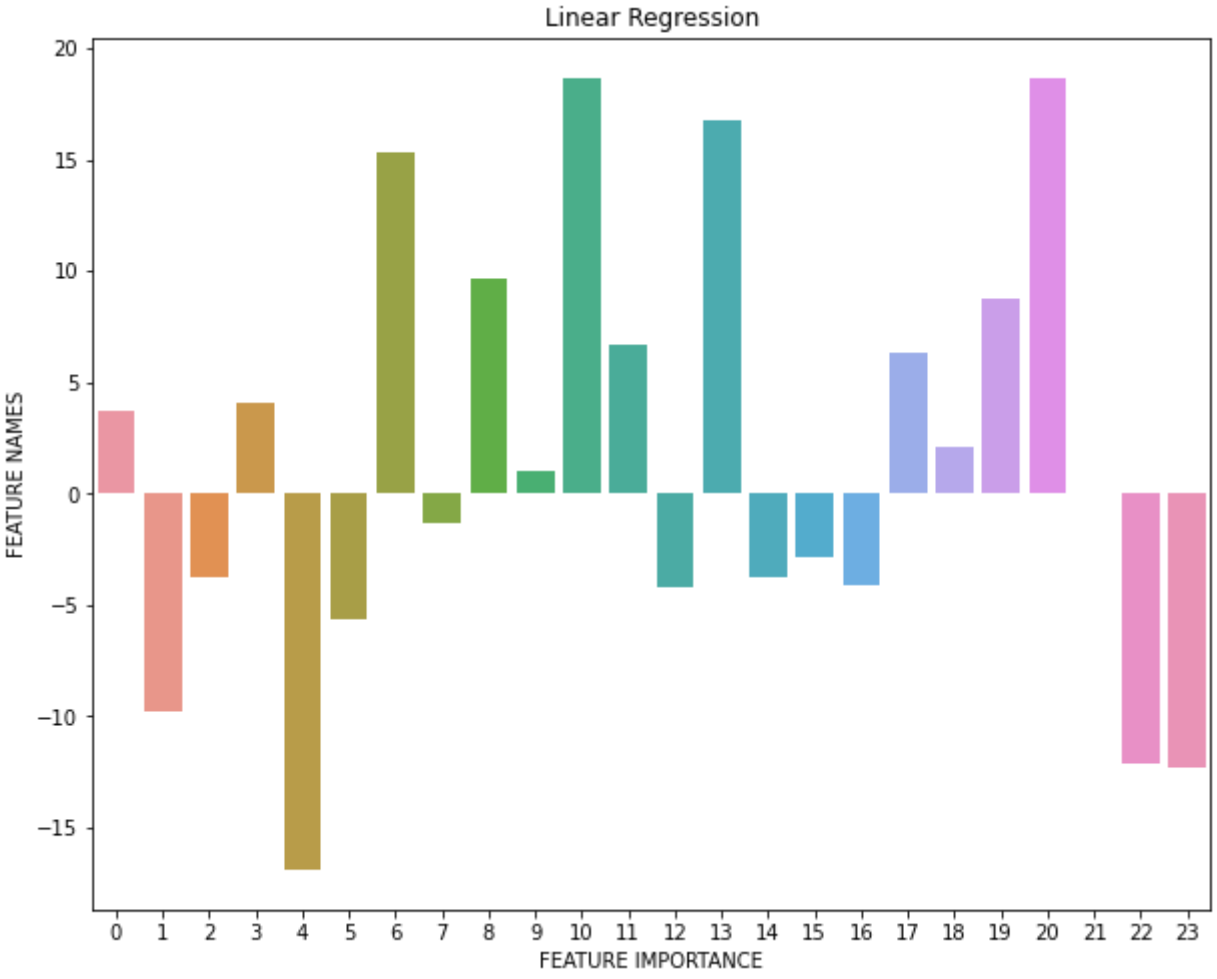
In [72]:

```
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,'Linear R
```

```
Feature: 0, Score: 3.73971
Feature: 1, Score: -9.77464
Feature: 2, Score: -3.71530
Feature: 3, Score: 4.05382
Feature: 4, Score: -16.89746
Feature: 5, Score: -5.67384
Feature: 6, Score: 15.32625
Feature: 7, Score: -1.34488
Feature: 8, Score: 9.61857
Feature: 9, Score: 1.00833
Feature: 10, Score: 18.67895
```

```
Feature: 11, Score: 6.69534
Feature: 12, Score: -4.22704
Feature: 13, Score: 16.72055
Feature: 14, Score: -3.79758
Feature: 15, Score: -2.83531
Feature: 16, Score: -4.14429
Feature: 17, Score: 6.32307
Feature: 18, Score: 2.07148
Feature: 19, Score: 8.78921
Feature: 20, Score: 18.64928
Feature: 21, Score: 0.00886
Feature: 22, Score: -12.10492
Feature: 23, Score: -12.30166
```





Regularization

In [74]:
```
# 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))
```

```
RMSE: 6527.931
R2: 0.423
```

In [75]:
```
import statsmodels.formula.api as smf

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

#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)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.478
Model:                            OLS   Adj. R-squared:                  0.478
Method:                 Least Squares   F-statistic:                     4092.
Date:                Tue, 10 May 2022   Prob (F-statistic):               0.00
Time:                        01:54:22   Log-Likelihood:            -6.1886e+05
No. Observations:              107382   AIC:                         1.238e+06
Df Residuals:                  107357   BIC:                         1.238e+06
Df Model:                          24
Covariance Type:            nonrobust
================================================================================
==================
                                   coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
------------------
Intercept                        109.4723      1.710     64.017      0.000      1
06.121     112.824
host_id                       3.011e-08   3.03e-09      9.928      0.000    2.
42e-08    3.61e-08
minimum_nights                   -0.2707      0.010    -27.068      0.000
-0.290      -0.251
number_of_reviews                -0.0973      0.008    -11.943      0.000
-0.113      -0.081
reviews_per_month                 4.2909      0.310     13.823      0.000
3.682       4.899
calculated_host_listings_count   -0.2302      0.007    -32.531      0.000
-0.244      -0.216
availability_365                 -0.0539      0.003    -19.742      0.000
```

```
-0.059      -0.049
accommodates                    10.9075      0.361     30.205      0.000
10.200      11.615
bedrooms                        -3.4943      0.989     -3.534      0.000
-5.432      -1.556
beds                            13.2494      0.668     19.839      0.000
11.940      14.558
air_conditioning                 2.3467      0.719      3.265      0.001
0.938       3.756
gym                             39.4521      0.799     49.351      0.000
37.885      41.019
tv                              20.1714      0.863     23.377      0.000
18.480      21.863
bbq                            -12.1013      0.885    -13.672      0.000     -
13.836     -10.367
bed_linen                       36.8910      0.863     42.741      0.000
35.199      38.583
coffee_machine                  -7.6541      0.905     -8.460      0.000
-9.427      -5.881
cooking_basics                  -5.8459      0.792     -7.377      0.000
-7.399      -4.293
white_goods                    -12.9413      0.824    -15.712      0.000     -
14.556     -11.327
elevator                        13.2787      0.762     17.434      0.000
11.786      14.772
parking                          4.6657      0.624      7.473      0.000
3.442       5.889
hot_tub_sauna_or_pool           25.6992      0.914     28.108      0.000
23.907      27.491
private_entrance                35.1307      0.895     39.261      0.000
33.377      36.884
toiletries                       0.1239      0.732      0.169      0.866
-1.311       1.558
workspace                      -26.7723      0.826    -32.408      0.000     -
28.391     -25.153
refrigerator                   -27.0922      0.881    -30.764      0.000     -
28.818     -25.366
==============================================================================
Omnibus:                   163999.837   Durbin-Watson:                   0.331
Prob(Omnibus):                  0.000   Jarque-Bera (JB):        235962444.844
Skew:                           9.178   Prob(JB):                         0.00
Kurtosis:                     231.913   Cond. No.                     1.64e+09
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.64e+09. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Out[75]:
```
[('Lagrange multiplier statistic', 1631.033120389776),
 ('p-value', 0.0),
 ('f-value', 68.99181313752081),
 ('f p-value', 0.0)]
```

In [76]:
```
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"))
```

```
+-------------------------+------------------+
| Model                   |   Execution Time |
+=========================+==================+
| Decision Tree           |        0.021867  |
+-------------------------+------------------+
| Random Forest Regressor |        0.751734  |
+-------------------------+------------------+
| XGBoost                 |        0.0596209 |
+-------------------------+------------------+
| Linear Regression       |        0.0545328 |
+-------------------------+------------------+
```

In [77]:

```
from tabulate import tabulate

xz=o['R2']
xz1=r['R2']
xz2=j['R2']
xz3=p['R2']

tab2 = [
    ["Decision Tree", xz],
    ["Random Forest Regressor", xz1],
    ["XGBoost", xz2],
    ["Linear Regression", xz3]
]

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

```
+-------------------------+----------+
| Model                   |       R2 |
+=========================+==========+
| Decision Tree           | 0.863517 |
+-------------------------+----------+
| Random Forest Regressor | 0.925983 |
+-------------------------+----------+
| XGBoost                 | 0.915519 |
+-------------------------+----------+
| Linear Regression       | 0.481458 |
+-------------------------+----------+
```

In [ ]: