Predict NYC Airbnb rental price

As of August 2019, this data set contains almost 50 thousand airbnb listings in NYC. The purpose of this task is to predict the price of NYC Airbnb rentals based on the data provided and any external dataset(s) with relevant information.

Columns -

idlisting ID

Name- name of the listing

host_idhost ID

Host_name - name of the host

Neighbourhood group - location

Neighbourhood - area

Latitude - latitude coordinates

Longitude - longitude coordinates

room_typelisting space type

Price - price in dollars

Minimum_nights - amount of nights minimum

Number_of_reviews - number of reviews

Last_review - latest review

Reviews_per_month - number of reviews per month

Calculated_host_listings_count - amount of listing per host

Availability_365 - number of days when listing is available for booking

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import  r2_score
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
```

In [2]:

```
df=pd.read_csv('AB_NYC_2019.csv')
```

In [3]:

```
df
```

Out[3]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhooc
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensingtor
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtowr
2	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlerr
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hil
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlerr
...	...	...	...	...	...	..
48890	36484665	Charming one bedroom - newly renovated rowhouse	8232441	Sabrina	Brooklyn	Bedford-Stuyvesan
48891	36485057	Affordable room in Bushwick/East Williamsburg	6570630	Marisol	Brooklyn	Bushwicl
48892	36485431	Sunny Studio at Historical Neighborhood	23492952	Ilgar & Aysel	Manhattan	Harlerr
48893	36485609	43rd St. Time Square-cozy single bed	30985759	Taz	Manhattan	Hell's Kitcher
48894	36487245	Trendy duplex in the very heart of Hell's Kitchen	68119814	Christophe	Manhattan	Hell's Kitcher

48895 rows × 16 columns

In [4]:

```
df.insert(0, 'ID', range(1, 1 + len(df)))
df.set_index("ID",inplace=True)
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 48895 entries, 1 to 48895
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   id                              48895 non-null  int64  
 1   name                            48879 non-null  object 
 2   host_id                         48895 non-null  int64  
 3   host_name                       48874 non-null  object 
 4   neighbourhood_group             48895 non-null  object 
 5   neighbourhood                   48895 non-null  object 
 6   latitude                        48895 non-null  float64
 7   longitude                       48895 non-null  float64
 8   room_type                       48895 non-null  object 
 9   price                           48895 non-null  int64  
 10  minimum_nights                  48895 non-null  int64  
 11  number_of_reviews               48895 non-null  int64  
 12  last_review                     38843 non-null  object 
 13  reviews_per_month               38843 non-null  float64
 14  calculated_host_listings_count  48895 non-null  int64  
 15  availability_365                48895 non-null  int64  
dtypes: float64(3), int64(7), object(6)
memory usage: 6.3+ MB
```

In [6]:

```
df.isnull().sum()
```

Out[6]:

```
id                                    0
name                                 16
host_id                               0
host_name                            21
neighbourhood_group                   0
neighbourhood                         0
latitude                              0
longitude                             0
room_type                             0
price                                 0
minimum_nights                        0
number_of_reviews                     0
last_review                       10052
reviews_per_month                 10052
calculated_host_listings_count        0
availability_365                      0
dtype: int64
```

EDA

In [7]:

```
# remove name,neighbourhood_group becasue there are in categorical data
df.drop(["name"],axis=1,inplace=True)
df.drop(["neighbourhood_group"],axis=1,inplace=True)
```

In [8]:

```
df.drop(["latitude"],axis=1,inplace=True)
df.drop(["longitude"],axis=1,inplace=True)
df.drop(["last_review"],axis=1,inplace=True)
df.drop(["reviews_per_month"],axis=1,inplace=True)
```

In [9]:

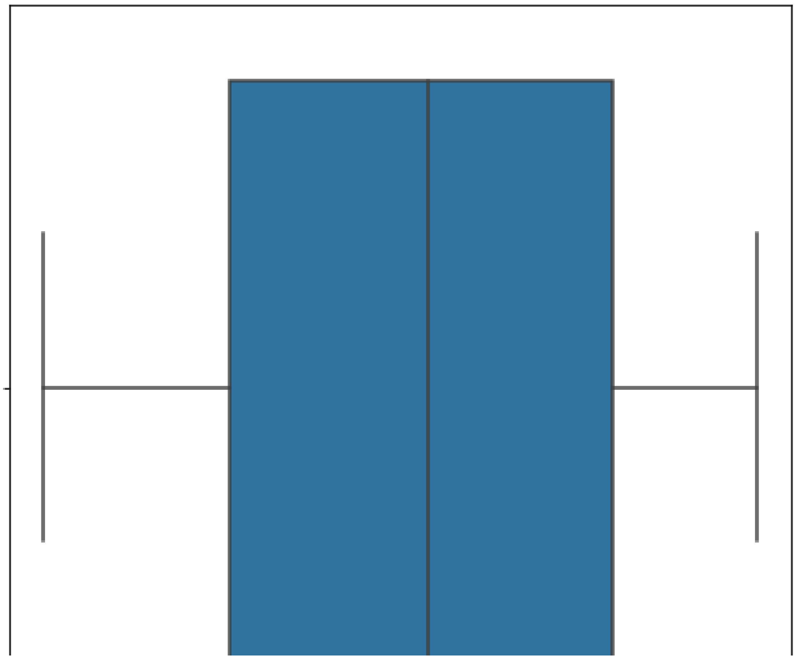
```
df_num=df.select_dtypes(['int64','float64'])
```

In [10]:

```
df_cat=df.select_dtypes(['object'])
```

In [11]:

```
for i in df_num:
    plt.figure(figsize=(7,7))
    sns.boxplot(data=df_num,x=i,whis=3)
    # upper whisker = q3+1.5*IQR
    # lower whisker = q1 - 1.5*IQR
    # boxplot will calculate upper whisker and lower whisker by it's own and the nit will p
    plt.show()
```



In [12]:

```
df_num.shape
```

Out[12]:

```
(48895, 7)
```

In [13]:

```
# price Column Outlier Treatment
q1=np.quantile(df_num["price"],0.25)
q3=np.quantile(df_num["price"],0.75)
iqr=q3-q1
print("Quantile1 for price is => ",q1)
print("Quantile3 for price is => ",q3)
print("IQR for price column is => ",iqr)
#as we know we have higher extream values so no need to calculate lower whisker will only g
up_whs=q3+3*iqr
print("upper whisker with 3 penalty is => ",up_whs)

# accept all those records which come below given whisker values
df_num=df_num[df_num["price"]<up_whs]
df_num.shape
```

```
Quantile1 for price is =>  69.0
Quantile3 for price is =>  175.0
IQR for price column is =>  106.0
upper whisker with 3 penalty is =>  493.0
```

Out[13]:

```
(47567, 7)
```

In [14]:

```
# minimum_nights Column Outlier Treatment
q1=np.quantile(df_num["minimum_nights"],0.25)
q3=np.quantile(df_num["minimum_nights"],0.75)
iqr=q3-q1
print("Quantile1 for minimum_nights is => ",q1)
print("Quantile3 for minimum_nights is => ",q3)
print("IQR for minimum_nights column is => ",iqr)
#as we know we have higher extream values so no need to calculate lower whisker will only g
up_whs=q3+3*iqr
print("upper whisker with 3 penalty is => ",up_whs)

# accept all those records which come below given whisker values
df_num=df_num[df_num["minimum_nights"]<up_whs]
df_num.shape
```

```
Quantile1 for minimum_nights is =>  1.0
Quantile3 for minimum_nights is =>  5.0
IQR for minimum_nights column is =>  4.0
upper whisker with 3 penalty is =>  17.0
```

Out[14]:

```
(42156, 7)
```

In [15]:

```
# number_of_reviews Column Outlier Treatment
q1=np.quantile(df_num["number_of_reviews"],0.25)
q3=np.quantile(df_num["number_of_reviews"],0.75)
iqr=q3-q1
print("Quantile1 for number_of_reviews is => ",q1)
print("Quantile3 for number_of_reviews is => ",q3)
print("IQR for number_of_reviews column is => ",iqr)
#as we know we have higher extream values so no need to calculate lower whisker will only g
up_whs=q3+3*iqr
print("upper whisker with 3 penalty is => ",up_whs)

# accept all those records which come below given whisker values
df_num=df_num[df_num["number_of_reviews"]<up_whs]
df_num.shape
```

```
Quantile1 for number_of_reviews is =>  1.0
Quantile3 for number_of_reviews is =>  27.0
IQR for number_of_reviews column is =>  26.0
upper whisker with 3 penalty is =>  105.0
```

Out[15]:

```
(39430, 7)
```

In [16]:

```
# calculated_host_listings_count Column Outlier Treatment
q1=np.quantile(df_num["calculated_host_listings_count"],0.25)
q3=np.quantile(df_num["calculated_host_listings_count"],0.75)
iqr=q3-q1
print("Quantile1 for calculated_host_listings_count is => ",q1)
print("Quantile3 for calculated_host_listings_count is => ",q3)
print("IQR for calculated_host_listings_count column is => ",iqr)
#as we know we have higher extream values so no need to calculate lower whisker will only g
up_whs=q3+3*iqr
print("upper whisker with 3 penalty is => ",up_whs)

# accept all those records which come below given whisker values
df_num=df_num[df_num["calculated_host_listings_count"]<up_whs]
df_num.shape
```

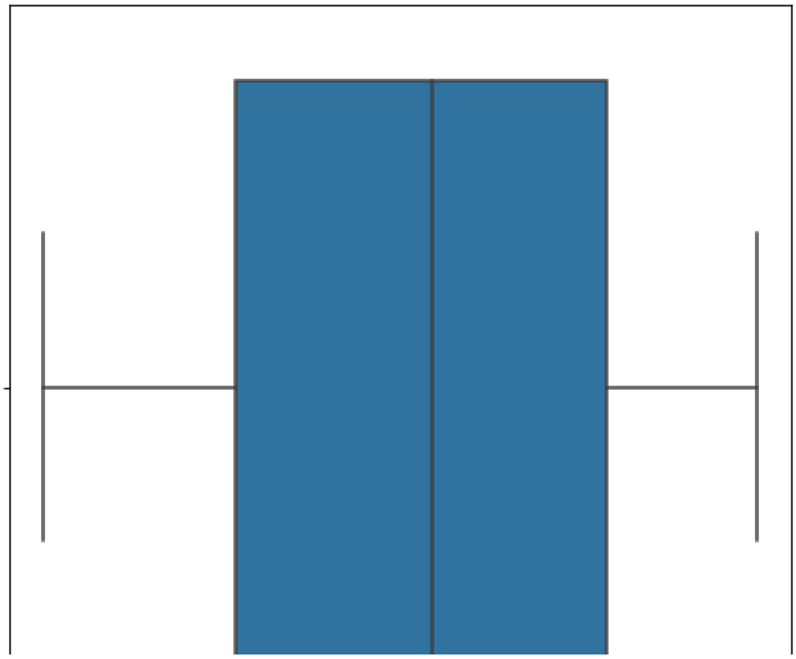
```
Quantile1 for calculated_host_listings_count is =>  1.0
Quantile3 for calculated_host_listings_count is =>  2.0
IQR for calculated_host_listings_count column is =>  1.0
upper whisker with 3 penalty is =>  5.0
```

Out[16]:

```
(36797, 7)
```

In [17]:

```
for i in df_num:
    plt.figure(figsize=(7,7))
    sns.boxplot(data=df_num,x=i,whis=3)
    # upper whisker = q3+1.5*IQR
    # lower whisker = q1 - 1.5*IQR
    # boxplot will calculate upper whisker and lower whisker by it's own and the nit will p
    plt.show()
```



In [18]:

```
from sklearn.preprocessing import LabelEncoder
```

In [20]:

```
df_cat.columns
```

Out[20]:

```
Index(['host_name', 'neighbourhood', 'room_type'], dtype='object')
```

In [22]:

```
df_cat
```

Out[22]:

ID	host_name	neighbourhood	room_type
1	John	Kensington	Private room
2	Jennifer	Midtown	Entire home/apt
3	Elisabeth	Harlem	Private room
4	LisaRoxanne	Clinton Hill	Entire home/apt
5	Laura	East Harlem	Entire home/apt
...	...	...	...
48891	Sabrina	Bedford-Stuyvesant	Private room
48892	Marisol	Bushwick	Private room
48893	Ilgar & Aysel	Harlem	Entire home/apt
48894	Taz	Hell's Kitchen	Shared room
48895	Christophe	Hell's Kitchen	Private room

48895 rows × 3 columns

In [24]:

```
df_cat["host_name"].unique
```

Out[24]:

```
<bound method Series.unique of ID
1                 John
2             Jennifer
3            Elisabeth
4          LisaRoxanne
5                Laura
             ...
48891          Sabrina
48892          Marisol
48893      Ilgar & Aysel
48894              Taz
48895       Christophe
Name: host_name, Length: 48895, dtype: object>
```

In [25]:

```
df_cat['room_type']=pd.to_numeric(df['room_type'].str.replace('/','').str.replace(',', ''),
```

In [27]:

```
le=LabelEncoder()
for col in df_cat:
    df_cat[col]=le.fit_transform(df_cat[col].astype(str))
```

In [29]:

```
df_new=pd.merge(df_num,df_cat,on="ID")
```

In [30]:

```
df_new
```

Out[30]:

	id	host_id	price	minimum_nights	number_of_reviews	calculated_host_listing
ID						
2	2595	2845	225	1	45	
3	3647	4632	150	3	0	
5	5022	7192	80	10	9	
6	5099	7322	200	3	74	
11	5295	7702	135	5	53	
...	...	...	...	...	...	
48890	36484363	107716952	65	1	0	
48891	36484665	8232441	70	2	0	
48892	36485057	6570630	40	4	0	
48893	36485431	23492952	115	10	0	
48895	36487245	68119814	90	7	0	

36797 rows × 10 columns

In [31]:

```
# import required modules
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

In [32]:

```
x=df_new.drop("price",axis=1)
y=df_new["price"]
```

In [33]:

```
#test Train model
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=1)
```

In [34]:

```
# Model Training
lr=LinearRegression() # object will be created
# llearning model on given data
# use theta0+theta1x to calculate bestfit prediction line
lr.fit(x_train,y_train)
```

Out[34]:

```
LinearRegression()
```

In [37]:

```
# now model is trained and ready to predict
# but before we use it for prediction let's check it's accuracy
y_pred=lr.predict(x_test)
```

In [35]:

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

In [38]:

```
mse=mean_squared_error(y_test,y_pred)
r2score=r2_score(y_test,y_pred)
print(" accuracy MSE is {} ".format(mse))
print(" accuracy R2 score is {} ".format(r2score))
```

```
 accuracy MSE is 6608.72882250768
 accuracy R2 score is 0.04407718539536687
```

Deep Learning

In [53]:

```
ss = StandardScaler()
x = ss.fit_transform(x)
```

In [54]:

```
x.shape[1]
```

Out[54]:

```
9
```

In [55]:

```
model = tf.keras.Sequential(
    [tf.keras.layers.Dense(2, activation="relu", input_shape=(x.shape[1], )),
     tf.keras.layers.Dense(5, activation="relu"),
     tf.keras.layers.Dense(3, activation="relu"),
     tf.keras.layers.Dense(1)]
)
```

In [56]:

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_7 (Dense)              (None, 2)                 20        
_________________________________________________________________
dense_8 (Dense)              (None, 5)                 15        
_________________________________________________________________
dense_9 (Dense)              (None, 3)                 18        
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 4         
=================================================================
Total params: 57
Trainable params: 57
Non-trainable params: 0
_________________________________________________________________
```

In [57]:

```
model.compile(optimizer="sgd", loss="mse")
```

In [58]:

```
trained_model = model.fit(x_train, y_train, epochs=50, batch_size=20)
```

```
Epoch 1/50
1288/1288 [==============================] - 1s 480us/step - loss: 1799912
186812275978403840.0000
Epoch 2/50
1288/1288 [==============================] - 1s 463us/step - loss: 6686.18
05
Epoch 3/50
1288/1288 [==============================] - 1s 475us/step - loss: 6937.95
41
Epoch 4/50
1288/1288 [==============================] - 1s 511us/step - loss: 6755.24
10
Epoch 5/50
1288/1288 [==============================] - 1s 487us/step - loss: 6791.57
30
Epoch 6/50
1288/1288 [==============================] - 1s 453us/step - loss: 6914.37
50
Epoch 7/50
1288/1288 [==============================] - 1s 544us/step - loss: 6829.99
21
Epoch 8/50
1288/1288 [==============================] - 1s 483us/step - loss: 6834.89
13
Epoch 9/50
1288/1288 [==============================] - 1s 518us/step - loss: 6797.69
94
Epoch 10/50
1288/1288 [==============================] - 1s 505us/step - loss: 6812.22
18
Epoch 11/50
1288/1288 [==============================] - 1s 493us/step - loss: 6892.35
65
Epoch 12/50
1288/1288 [==============================] - 1s 469us/step - loss: 6806.68
80
Epoch 13/50
1288/1288 [==============================] - 1s 578us/step - loss: 6764.08
81
Epoch 14/50
1288/1288 [==============================] - 1s 468us/step - loss: 6893.10
65
Epoch 15/50
1288/1288 [==============================] - 1s 473us/step - loss: 6726.30
60
Epoch 16/50
1288/1288 [==============================] - 1s 511us/step - loss: 6861.77
27
Epoch 17/50
1288/1288 [==============================] - 1s 498us/step - loss: 6870.99
91
Epoch 18/50
1288/1288 [==============================] - 1s 615us/step - loss: 6848.58
89
Epoch 19/50
1288/1288 [==============================] - 1s 524us/step - loss: 6847.62
```

```
82
Epoch 20/50
1288/1288 [==============================] - 1s 446us/step - loss: 6928.36
29
Epoch 21/50
1288/1288 [==============================] - 1s 501us/step - loss: 6720.67
22
Epoch 22/50
1288/1288 [==============================] - 1s 643us/step - loss: 6702.07
88
Epoch 23/50
1288/1288 [==============================] - 1s 659us/step - loss: 6848.17
73
Epoch 24/50
1288/1288 [==============================] - 1s 578us/step - loss: 6796.68
92
Epoch 25/50
1288/1288 [==============================] - 1s 584us/step - loss: 6976.03
14
Epoch 26/50
1288/1288 [==============================] - 1s 603us/step - loss: 6813.49
94
Epoch 27/50
1288/1288 [==============================] - 1s 595us/step - loss: 6780.63
73
Epoch 28/50
1288/1288 [==============================] - 1s 540us/step - loss: 6823.94
08
Epoch 29/50
1288/1288 [==============================] - 1s 501us/step - loss: 6898.52
23
Epoch 30/50
1288/1288 [==============================] - 1s 526us/step - loss: 6848.43
61
Epoch 31/50
1288/1288 [==============================] - 1s 501us/step - loss: 6857.61
88
Epoch 32/50
1288/1288 [==============================] - 1s 607us/step - loss: 6765.04
99
Epoch 33/50
1288/1288 [==============================] - 1s 499us/step - loss: 6844.51
86
Epoch 34/50
1288/1288 [==============================] - 1s 490us/step - loss: 6769.03
32
Epoch 35/50
1288/1288 [==============================] - 1s 513us/step - loss: 6825.89
67
Epoch 36/50
1288/1288 [==============================] - 1s 525us/step - loss: 6840.13
74
Epoch 37/50
1288/1288 [==============================] - 1s 529us/step - loss: 6842.95
11
Epoch 38/50
1288/1288 [==============================] - 1s 460us/step - loss: 6738.22
09
Epoch 39/50
1288/1288 [==============================] - 1s 467us/step - loss: 6876.55
66
```

```
Epoch 40/50
1288/1288 [==============================] - 1s 467us/step - loss: 6747.39
46
Epoch 41/50
1288/1288 [==============================] - 1s 456us/step - loss: 6878.53
31
Epoch 42/50
1288/1288 [==============================] - 1s 462us/step - loss: 6788.87
75
Epoch 43/50
1288/1288 [==============================] - 1s 467us/step - loss: 6862.65
99
Epoch 44/50
1288/1288 [==============================] - 1s 475us/step - loss: 6697.66
99
Epoch 45/50
1288/1288 [==============================] - 1s 511us/step - loss: 6795.49
15
Epoch 46/50
1288/1288 [==============================] - 1s 508us/step - loss: 6925.93
78
Epoch 47/50
1288/1288 [==============================] - 1s 477us/step - loss: 7039.74
82
Epoch 48/50
1288/1288 [==============================] - 1s 628us/step - loss: 6862.60
60
Epoch 49/50
1288/1288 [==============================] - 1s 469us/step - loss: 6816.63
75
Epoch 50/50
1288/1288 [==============================] - 1s 482us/step - loss: 6802.34
60
```

In [59]:

```
# testing
y_hat = model.predict(x_test)
```

In [60]:

```
r2_score(y_test, y_hat)
```

Out[60]:

```
-0.00013958187525808796
```

In [ ]:

```
```