In [ ]:
```
## Data Science Problem
```

Crack this Business Problem

Renting a house or an apartment is never easy. Whether you are a college student or a working professional, renting a place always seems like a daunting task that is often impulsive or risky. Rent is influenced by several factors.

In this challenge, participants will predict the house-rents using data science methods, machine learning, and hyperparameter tuning.

In [ ]:
```
##! unzip /content/Participants_data_DSSC_SZ.zip
```

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [ ]:
```
train = pd.read_csv("/content/train.csv")
```

In [ ]:
```
test = pd.read_csv("/content/test.csv")
```

Data Dimensions

In [ ]:
```
train.columns
```

Out[ ]:
```
Index(['Property_ID', 'room', 'layout_type', 'property_type', 'locality',
       'price', 'area', 'furnish_type', 'bathroom', 'city', 'parking_spaces',
       'floor', 'pet_friendly', 'power_backup', 'washing_machine',
       'air_conditioner', 'geyser/solar', 'security_deposit', 'CCTV/security',
       'lift', 'neighbourhood'],
      dtype='object')
```

In [ ]:
```
test.columns
```

Out[ ]:
```
Index(['Property_ID', 'room', 'layout_type', 'property_type', 'locality',
```

```
'area', 'furnish_type', 'bathroom', 'city', 'parking_spaces', 'floor',
'pet_friendly', 'power_backup', 'washing_machine', 'air_conditioner',
'geyser/solar', 'security_deposit', 'CCTV/security', 'lift',
'neighbourhood', 'price'],
```

In [ ]:
```
train.head(3)
```

Out[ ]:

	Property_ID	room	layout_type	property_type	locality	price	area	furnish_type	bathroom	city	...	floor	pet_friendly	pow
0	42208	3	BHK	Independent House	Palavakkam	33624	1312	Furnished	2	Chennai	...	1	1	
1	90879	1	BHK	Apartment	Manikonda	9655	1474	Unfurnished	2	Hyderabad	...	17	0	
2	99943	3	BHK	Apartment	Jodhpur Park	23699	1837	Semi-Furnished	2	Kolkata	...	10	1	

3 rows × 21 columns

In [ ]:
```
train["price"].head(3)
```

Out[ ]:
```
0    33624
1     9655
2    23699
Name: price, dtype: int64
```

In [ ]:
```
test.head(3)
```

Out[ ]:

	Property_ID	room	layout_type	property_type	locality	area	furnish_type	bathroom	city	parking_spaces	...	pet_friendly	p
0	114342	2	BHK	Independent Floor	Palava	1347	Semi-Furnished	1	Mumbai	0	...	0	
1	88819	1	BHK	Independent House	Somajiguda	634	Semi-Furnished	3	Hyderabad	1	...	0	
2	85623	1	BHK	Apartment	Toli Chowki	524	Unfurnished	1	Hyderabad	1	...	1	

3 rows × 21 columns

In [ ]:
```
test["price"] = 0
```

Data Types

In [ ]:
```
print("The datatypes of the train set", train.dtypes)
print("The datatypes of the test set", test.dtypes)
```

```
The datatypes of the train set Property_ID          int64
room                  int64
layout_type          object
property_type        object
locality             object
price                 int64
area                  int64
furnish_type         object
bathroom              int64
city                 object
parking_spaces        int64
floor                 int64
pet_friendly          int64
power_backup          int64
washing_machine       int64
air_conditioner       int64
geyser/solar          int64
security_deposit      int64
CCTV/security         int64
lift                  int64
neighbourhood         int64
dtype: object
The datatypes of the test set Property_ID          int64
room                  int64
layout_type          object
property_type        object
locality             object
area                  int64
furnish_type         object
bathroom              int64
city                 object
parking_spaces        int64
floor                 int64
pet_friendly          int64
power_backup          int64
washing_machine       int64
```

```
air_conditioner      int64
geyser/solar         int64
security_deposit     int64
CCTV/security        int64
lift                 int64
neighbourhood        int64
price                int64
dtype: object
```

In [ ]:
```
num_var = [feature for feature in train.columns if train[feature].dtypes != 'O']
discrete_var = [feature for feature in num_var if len(train[feature].unique()) <= 25]
cont_var = [feature for feature in num_var if feature not in discrete_var]
categ_var = [feature for feature in train.columns if feature not in num_var]
```

In [ ]:
```
print("The category values are :",train[categ_var].columns)
print("The numerical values are :",train[num_var].columns)
print("The discrete values are :", train[discrete_var].columns)
print("The continuous values are :",train[cont_var].columns)
```

```
The category values are : Index(['layout_type', 'property_type', 'locality', 'furnish_type', 'city'], dtype='object')
The numerical values are : Index(['Property_ID', 'room', 'price', 'area', 'bathroom', 'parking_spaces',
       'floor', 'pet_friendly', 'power_backup', 'washing_machine',
       'air_conditioner', 'geyser/solar', 'security_deposit', 'CCTV/security',
       'lift', 'neighbourhood'],
      dtype='object')
The discrete values are : Index(['room', 'bathroom', 'parking_spaces', 'floor', 'pet_friendly',
       'power_backup', 'washing_machine', 'air_conditioner', 'geyser/solar',
       'CCTV/security', 'lift'],
      dtype='object')
The continuous values are : Index(['Property_ID', 'price', 'area', 'security_deposit', 'neighbourhood'], dtype='object
')
```

In [ ]:
```
def find_var_type(var):

    if var in discrete_var:
        print("{} is a Numerical Variable, Discrete in nature".format(var))
    elif var in cont_var :
        print("{} is a Numerical Variable, Continuous in nature".format(var))
    else :
        print("{} is a Categorical Variable".format(var))
```

In [ ]:
```
train.columns
```

Out[ ]:
```
Index(['Property_ID', 'room', 'layout_type', 'property_type', 'locality',
       'price', 'area', 'furnish_type', 'bathroom', 'city', 'parking_spaces',
       'floor', 'pet_friendly', 'power_backup', 'washing_machine',
       'air_conditioner', 'geyser/solar', 'security_deposit', 'CCTV/security',
       'lift', 'neighbourhood'],
      dtype='object')
```

In [ ]:
```
find_var_type('Property_ID')
find_var_type('room')
find_var_type('layout_type')
find_var_type('property_type')
find_var_type('locality')
find_var_type('price')
find_var_type('area')
find_var_type('furnish_type')
find_var_type('bathroom')
find_var_type('city')
find_var_type('parking_spaces')
find_var_type('floor')
find_var_type('pet_friendly')
find_var_type('power_backup')
find_var_type('washing_machine')
find_var_type('air_conditioner')
find_var_type('geyser/solar')
find_var_type('security_deposit')
find_var_type('CCTV/security')
find_var_type('lift')
find_var_type('neighbourhood')
```

```
Property_ID is a Numerical Variable, Continuous in nature
room is a Numerical Variable, Discrete in nature
layout_type is a Categorical Variable
property_type is a Categorical Variable
locality is a Categorical Variable
price is a Numerical Variable, Continuous in nature
area is a Numerical Variable, Continuous in nature
furnish_type is a Categorical Variable
bathroom is a Numerical Variable, Discrete in nature
city is a Categorical Variable
parking_spaces is a Numerical Variable, Discrete in nature
```

```
floor is a Numerical Variable, Discrete in nature
pet_friendly is a Numerical Variable, Discrete in nature
power_backup is a Numerical Variable, Discrete in nature
washing_machine is a Numerical Variable, Discrete in nature
air_conditioner is a Numerical Variable, Discrete in nature
geyser/solar is a Numerical Variable, Discrete in nature
security_deposit is a Numerical Variable, Continuous in nature
CCTV/security is a Numerical Variable, Discrete in nature
lift is a Numerical Variable, Discrete in nature
neighbourhood is a Numerical Variable, Continuous in nature
```

Handling `Missing` Values

In [ ]:
```
train.isnull().sum()
```

Out[ ]:
```
Property_ID         0
room                0
layout_type         0
property_type       0
locality            0
price               0
area                0
furnish_type        0
bathroom            0
city                0
parking_spaces      0
floor               0
pet_friendly        0
power_backup        0
washing_machine     0
air_conditioner     0
geyser/solar        0
security_deposit    0
CCTV/security       0
lift                0
neighbourhood       0
dtype: int64
```

In [ ]:
```
test.isnull().sum()
```

Out[ ]:
```
Property_ID         0
room                0
```

```
layout_type         0
property_type       0
locality            0
area                0
furnish_type        0
bathroom            0
city                0
parking_spaces      0
floor               0
pet_friendly        0
power_backup        0
washing_machine     0
air_conditioner     0
geyser/solar        0
security_deposit    0
CCTV/security       0
lift                0
neighbourhood       0
price               0
```

In [ ]:
```
### Here there is no missing values
```

Missing Values Imputation

1. We will do Ramdom Sample Imputation for the our variables which are having the most percentage of Null Values

2. Mean Imputation For continuous variables

In [ ]:
```
def RandomSampleImputation(df, feature):
    df[feature]=df[feature]
    random_sample=df[feature].dropna().sample(df[feature].isnull().sum(),random_state=0)
    random_sample.index=df[df[feature].isnull()].index
    df.loc[df[feature].isnull(),feature]=random_sample
```

In [ ]:
```
def MeanImputation(df, feature):
    df[feature]= df[feature]
    mean= df[feature].mean()
    df[feature]= df[feature].fillna(mean)
```

Category Encoding

In [ ]:
```
##! pip install category_encoders
```

In [ ]:
```
import category_encoders as ce
import pandas as pd
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [ ]:
```
print("The category values are :",train[categ_var].columns)
```

```
The category values are : Index(['layout_type', 'property_type', 'locality', 'furnish_type', 'city'], dtype='object')
```

The Category Data can be of two types :

1. Nominal Data :
   Nominal data is data that can be labelled or classified into mutually exclusive categories within a variable. These categories cannot be ordered in a meaningful way.
2. Ordinal Data :
   ordinal data is said to have been collected when a responder inputs his/her financial happiness level on a scale of 1-10.

In [ ]:
```
train.layout_type.value_counts()
```

Out[ ]:
```
BHK    114684
RK      19999
Name: layout_type, dtype: int64
```

In [ ]:
```
print(train.property_type.value_counts())
print(train.locality.value_counts())
print(train.furnish_type.value_counts())
print(train.city.value_counts())
```

```
Apartment            86819
Independent Floor    25850
Independent House    13408
Studio Apartment      5723
Villa                 2391
Penthouse              492
```

```
Name: property_type, dtype: int64
Thane West                          3127
Chembur                             2461
Andheri East                        2377
Bopal                               2054
Kharghar                            1819
                                    ...
Annapoorneshwari Nagar                 1
Kona Expressway                        1
Nandambakkam                           1
Rajpur Khurd Extension                 1
Venkatapuram Alwal Secundrabad         1
Name: locality, Length: 3706, dtype: int64
Semi-Furnished    63646
Unfurnished       41398
Furnished         29639
Name: furnish_type, dtype: int64
Mumbai       46910
Delhi        22826
Bangalore    16092
Pune         15713
Ahmedabad    12976
Hyderabad     7334
Kolkata       6795
Chennai       6037
Name: city, dtype: int64
```

In [ ]:
```
encoder= ce.OrdinalEncoder(cols=['layout_type'],return_df=True, mapping=[{'col':'layout_type', 'mapping':{'BHK': 1, 'R
```

In [ ]:
```
train['layout_type'] = encoder.fit_transform(train['layout_type'])
```

In [ ]:
```
encoder= ce.OrdinalEncoder(cols=['property_type'],return_df=True, mapping=[{'col':'property_type', 'mapping':{'Apartme
```

In [ ]:
```
train['property_type'] = encoder.fit_transform(train['property_type'])
```

In [ ]:
```
encoder= ce.OrdinalEncoder(cols=['furnish_type'],return_df=True, mapping=[{'col':'furnish_type', 'mapping':{'Semi-Furn
```

In [ ]:
```
train['furnish_type'] = encoder.fit_transform(train['furnish_type'])
```

Locality and City , I can use `Label` Encoding or `Mean` Encoding

Unlike label encoding, which gets the work done efficiently but in a random way, **mean encoding tries to approach** the problem more logically. In a nutshell, it uses the target variable as the basis to generate the new encoded feature.

In [ ]:
```
train.groupby(['locality'])['price'].mean()
```

Out[ ]:
```
locality
1 Sector Number 3 Road     188557.0
10 Sector Number 3 Road     28970.0
100 Feet Road               17570.0
11 Sector Number 6 Road     20495.0
14 Road                      8808.0
                              ...
vivekananda Nagar           12998.0
wadebolhai                  11800.0
wakad                       78465.0
worli sea Fase              38462.0
yogi nagar                  27504.0
Name: price, Length: 3706, dtype: float64
```

In [ ]:
```
Mean_encoded_subject = train.groupby(['locality'])['price'].mean().to_dict()

train['locality'] =  train['locality'].map(Mean_encoded_subject)
```

In [ ]:
```
train.groupby(['city'])['price'].mean()
```

Out[ ]:
```
city
Ahmedabad    43099.677250
Bangalore    39863.151007
Chennai      37694.160345
Delhi        36888.325506
Hyderabad    36681.611672
Kolkata      35341.154084
Mumbai       35078.077745
Pune         32872.992045
Name: price, dtype: float64
```

In [ ]:
```
Mean_encoded_subject = train.groupby(['city'])['price'].mean().to_dict()

train['city'] =  train['city'].map(Mean_encoded_subject)
```

In [ ]:
```
print(train.columns)
```

```
Index(['Property_ID', 'room', 'layout_type', 'property_type', 'locality',
       'price', 'area', 'furnish_type', 'bathroom', 'city', 'parking_spaces',
       'floor', 'pet_friendly', 'power_backup', 'washing_machine',
       'air_conditioner', 'geyser/solar', 'security_deposit', 'CCTV/security',
       'lift', 'neighbourhood'],
      dtype='object')
```

Similarly, for the test data Set

In [ ]:
```
test.columns
```

Out[ ]:
```
Index(['Property_ID', 'room', 'layout_type', 'property_type', 'locality',
       'area', 'furnish_type', 'bathroom', 'city', 'parking_spaces', 'floor',
       'pet_friendly', 'power_backup', 'washing_machine', 'air_conditioner',
       'geyser/solar', 'security_deposit', 'CCTV/security', 'lift',
       'neighbourhood', 'price'],
      dtype='object')
```

In [ ]:
```
num_var = [feature for feature in test.columns if test[feature].dtypes != 'O']
discrete_var = [feature for feature in num_var if len(test[feature].unique()) <= 25]
cont_var = [feature for feature in num_var if feature not in discrete_var]
categ_var = [feature for feature in test.columns if feature not in num_var]
```

In [ ]:
```
test[categ_var].columns
```

Out[ ]:
```
Index(['layout_type', 'property_type', 'locality', 'furnish_type', 'city'], dtype='object')
```

In [ ]:
```
encoder= ce.OrdinalEncoder(cols=['layout_type'],return_df=True, mapping=[{'col':'layout_type', 'mapping':{'BHK': 1, 'R
test['layout_type'] = encoder.fit_transform(test['layout_type'])
```

In [ ]:
```
encoder= ce.OrdinalEncoder(cols=['property_type'],return_df=True, mapping=[{'col':'property_type', 'mapping':{'Apartme
test['property_type'] = encoder.fit_transform(test['property_type'])
```

In [ ]:
```
encoder= ce.OrdinalEncoder(cols=['furnish_type'],return_df=True, mapping=[{'col':'furnish_type', 'mapping':{'Semi-Furn
test['furnish_type'] = encoder.fit_transform(test['furnish_type'])
```

In [ ]:
```
from sklearn.preprocessing import OrdinalEncoder

test["city"] = test["city"].astype("category").cat.codes
```

In [ ]:
```
test["locality"] = test["locality"].astype("category").cat.codes
```

In [ ]:
```
train.columns
```

Out[ ]:
```
Index(['Property_ID', 'room', 'layout_type', 'property_type', 'locality',
       'price', 'area', 'furnish_type', 'bathroom', 'city', 'parking_spaces',
       'floor', 'pet_friendly', 'power_backup', 'washing_machine',
       'air_conditioner', 'geyser/solar', 'security_deposit', 'CCTV/security',
       'lift', 'neighbourhood'],
      dtype='object')
```

In [ ]:
```
# Checking for outliers in the continuous variables
num_train_data = train[['Property_ID', 'room', 'layout_type', 'property_type', 'locality',
       'price', 'area', 'furnish_type', 'bathroom', 'city', 'parking_spaces',
       'floor', 'pet_friendly', 'power_backup', 'washing_machine',
       'air_conditioner', 'geyser/solar', 'security_deposit', 'CCTV/security',
       'lift', 'neighbourhood']]
```

In [ ]:
```
# Checking outliers at 25%, 50%, 75%, 90%, 95% and 99%
num_train_data.describe(percentiles=[.25, .5, .75, .90, .95, .99])
```

Out[ ]:

	Property_ID	room	layout_type	property_type	locality	price	area	furnish_type	bathroc
count	134683.000000	134683.000000	134683.000000	134683.000000	134683.000000	134683.000000	134683.000000	134683.000000	134683.0000

	Property_ID	room	layout_type	property_type	locality	price	area	furnish_type	bathroc
mean	96036.100777	2.029677	1.148489	1.607790	36690.033894	36690.033894	1480.388490	1.747503	2.0404
std	55565.228125	0.937308	0.355586	0.989212	11202.374171	62620.364025	1412.464718	0.793017	0.8670
min	2.000000	1.000000	1.000000	1.000000	1850.000000	1583.000000	81.000000	1.000000	1.0000
25%	47940.000000	1.000000	1.000000	1.000000	32926.574359	12035.500000	759.000000	1.000000	1.0000
50%	95950.000000	2.000000	1.000000	1.000000	35546.609707	20856.000000	1114.000000	2.000000	2.0000
75%	144194.500000	3.000000	1.000000	2.000000	39499.064417	36014.000000	1580.000000	2.000000	2.0000
90%	173076.800000	3.000000	2.000000	3.000000	44110.060883	68180.600000	2461.000000	3.000000	3.0000
95%	182719.900000	4.000000	2.000000	4.000000	47027.878378	117600.400000	3833.000000	3.000000	4.0000
99%	190431.360000	4.000000	2.000000	5.000000	66104.862745	343687.580000	8726.000000	3.000000	5.0000
max	192405.000000	5.000000	2.000000	6.000000	738467.000000	799325.000000	13942.000000	3.000000	5.0000

In [ ]:

```
Q1 = train.quantile(0.25)
Q3 = train.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
Property_ID          96254.500000
room                     2.000000
layout_type              0.000000
property_type            1.000000
locality              6572.490058
price                23978.500000
area                   821.000000
furnish_type             1.000000
bathroom                 1.000000
city                  2616.082600
parking_spaces           1.000000
floor                   11.000000
pet_friendly             1.000000
power_backup             1.000000
washing_machine          1.000000
air_conditioner          1.000000
geyser/solar             1.000000
security_deposit    164902.000000
CCTV/security            1.000000
```

```
lift                    1.000000
neighbourhood         1900.000000
dtype: float64
```

In [ ]:

```
print(train.quantile(0.10))
print(train.quantile(0.90))
```

```
Property_ID         19083.200000
room                    1.000000
layout_type             1.000000
property_type           1.000000
locality            29411.795455
price                6086.200000
area                  625.000000
furnish_type            1.000000
bathroom                1.000000
city                32872.992045
parking_spaces          0.000000
floor                   2.000000
pet_friendly            0.000000
power_backup            0.000000
washing_machine         0.000000
air_conditioner         0.000000
geyser/solar            0.000000
security_deposit    28688.200000
CCTV/security           0.000000
lift                    0.000000
neighbourhood         400.000000
Name: 0.1, dtype: float64
Property_ID         173076.800000
room                     3.000000
layout_type              2.000000
property_type            3.000000
locality             44110.060883
price                68180.600000
area                  2461.000000
furnish_type             3.000000
bathroom                 3.000000
city                 39863.151007
parking_spaces           1.000000
floor                   17.000000
pet_friendly             1.000000
power_backup             1.000000
washing_machine          1.000000
air_conditioner          1.000000
geyser/solar             1.000000
```

```
security_deposit    424584.000000
CCTV/security            1.000000
lift                     1.000000
neighbourhood         3700.000000
Name: 0.9, dtype: float64
```

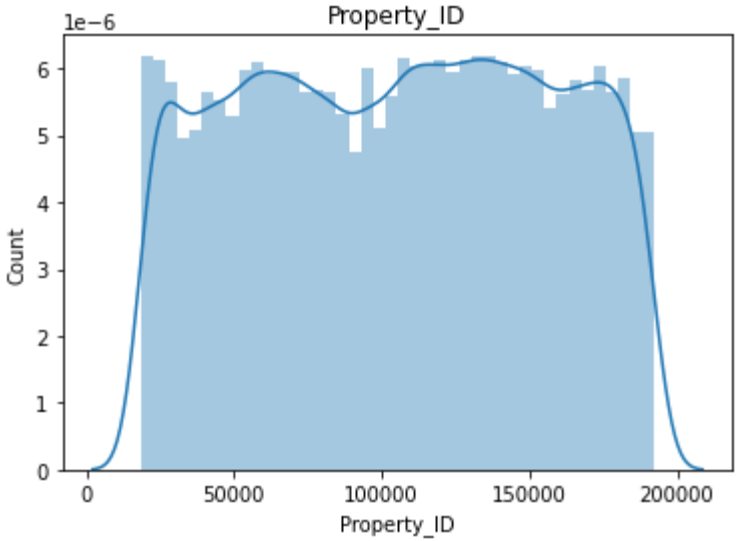
In [ ]:
```
train = train[~((train < (Q1 - 1.5 * IQR)) | (train > (Q3 + 1.5 * IQR))).any(axis=1)]
print(train.shape)
```

```
(68352, 21)
```

Analysis for Continuous variables

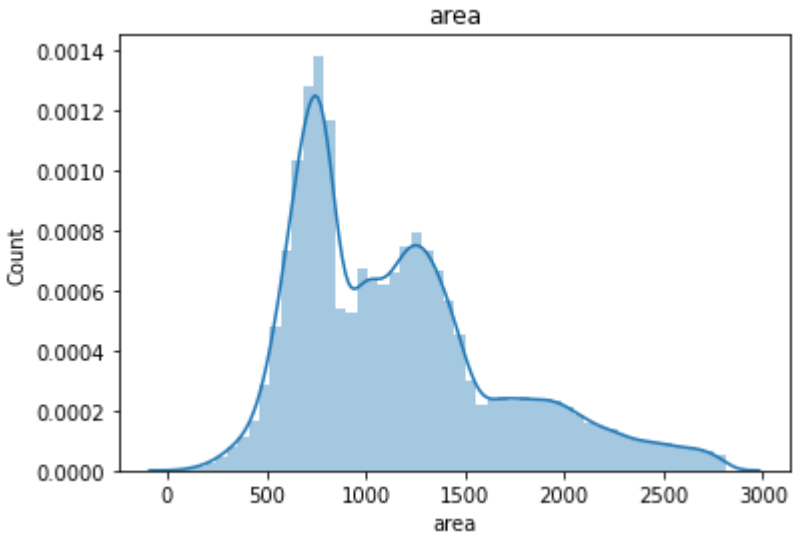
In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

for feature in cont_var:
    data=train.copy()
    sns.distplot(train[feature])
    plt.xlabel(feature)
    plt.ylabel("Count")
    plt.title(feature)
    plt.figure(figsize=(15,15))
    plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated functi
on and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function wi
th similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



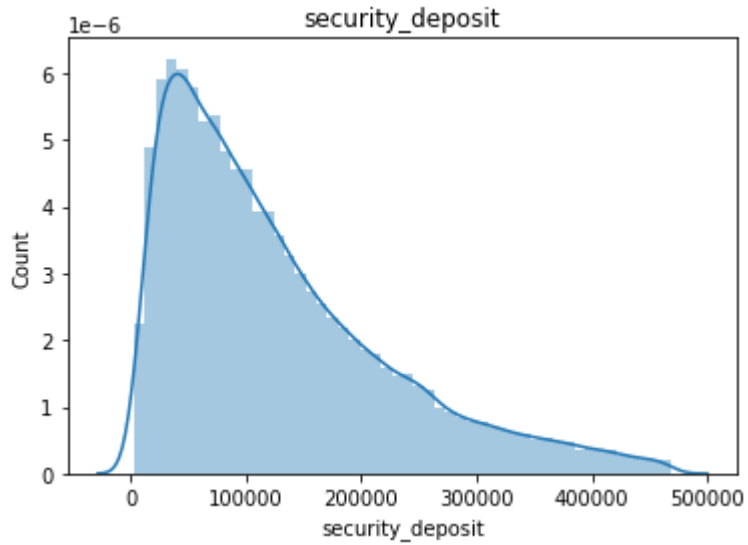

```
<Figure size 1080x1080 with 0 Axes>
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated functi
on and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function wi
th similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```




```
<Figure size 1080x1080 with 0 Axes>
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated functi
on and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function wi
th similar flexibility) or `histplot` (an axes-level function for histograms).
```




```
<Figure size 1080x1080 with 0 Axes>
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated functi
on and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function wi
th similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

The Skewed data is required to be converted to normalised data

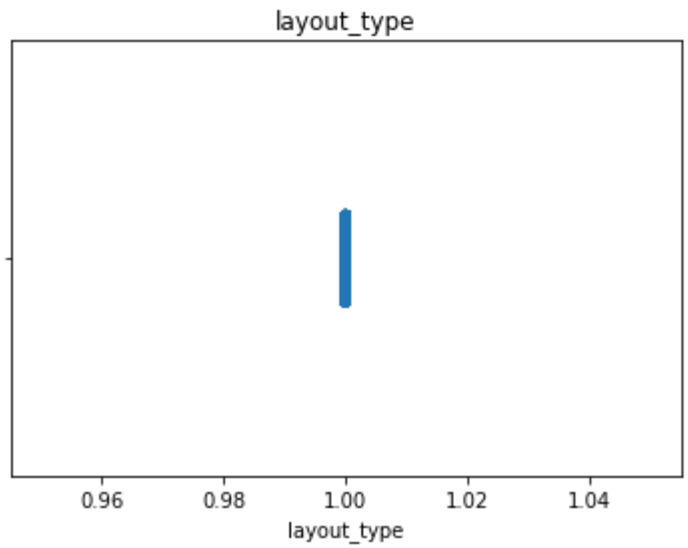
Analysis Of Categorical Variables

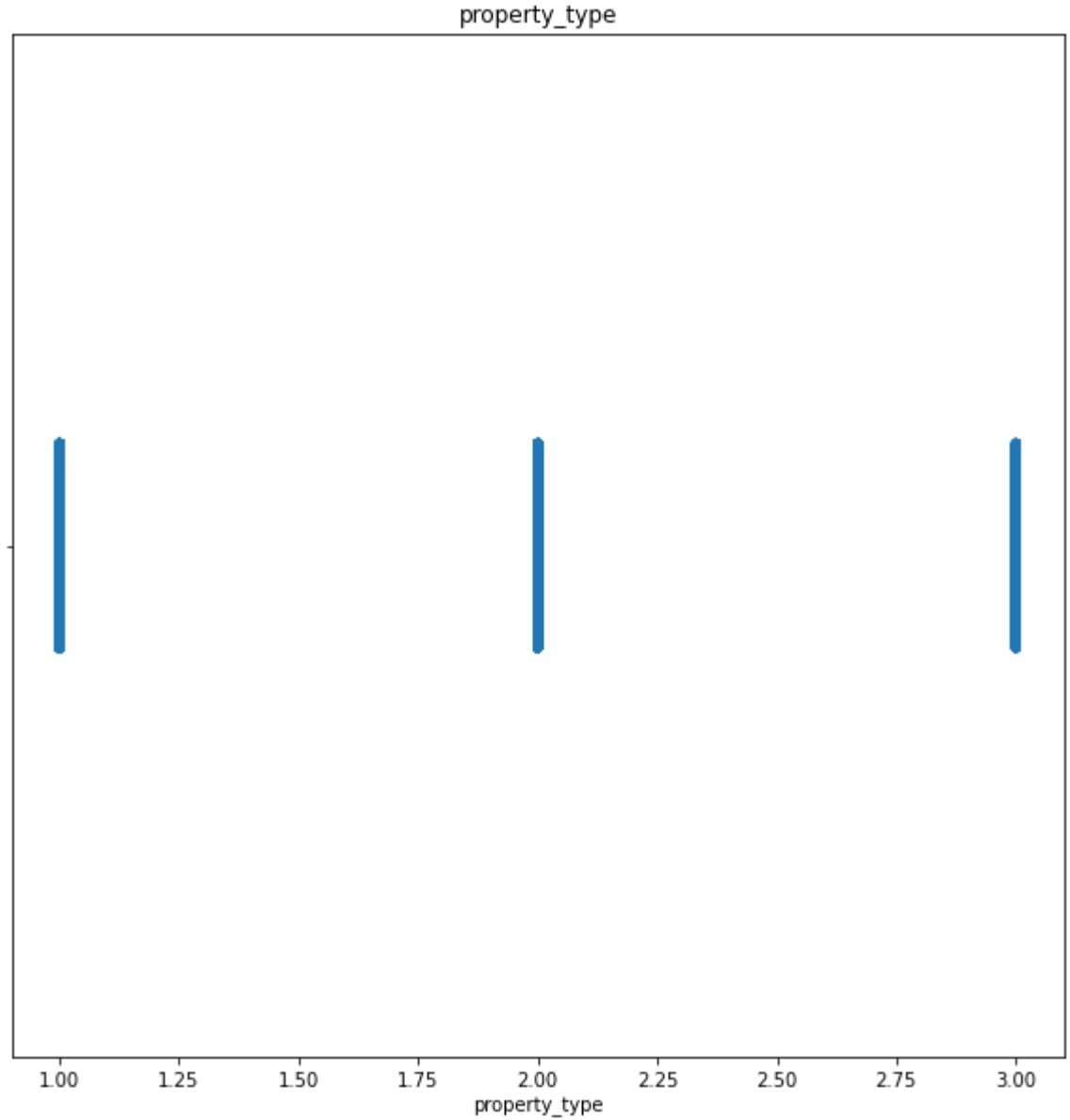
In [ ]:

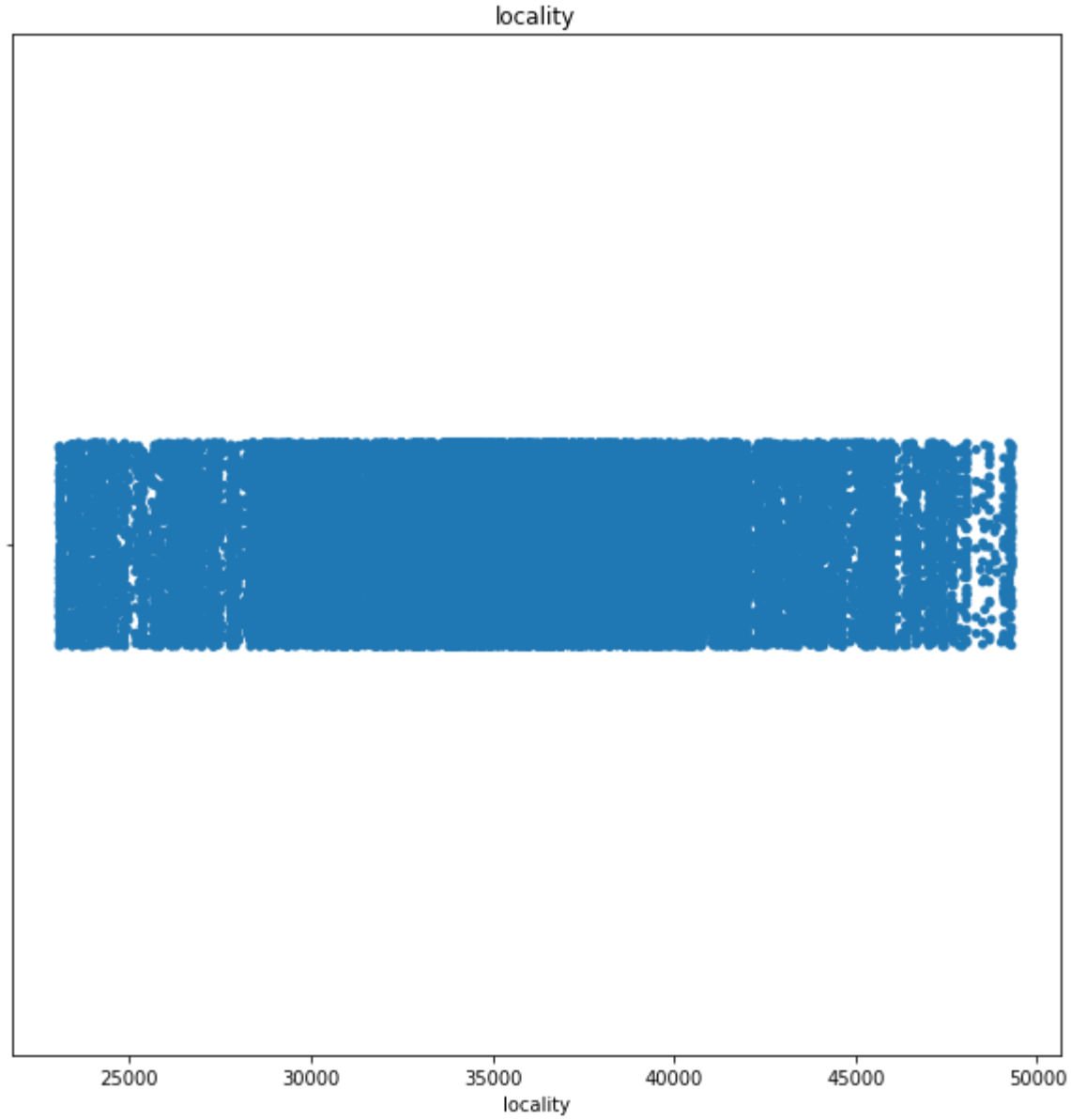
```
for feature in categ_var:
    data=train.copy()
    sns.stripplot(data[feature])
    plt.title(feature)
    plt.figure(figsize=(10,10))
```

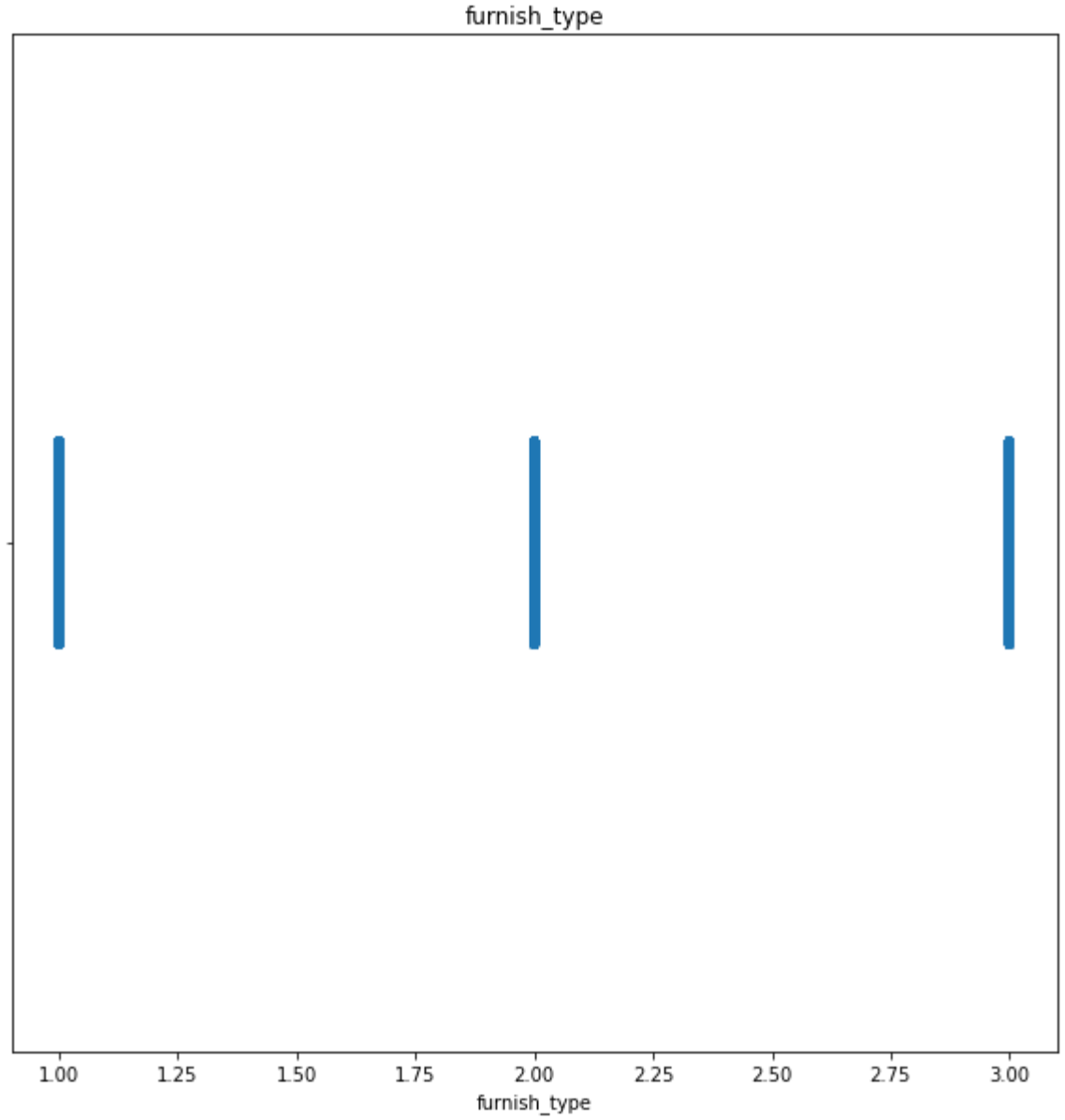
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyw
ord arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without
an explicit keyword will result in an error or misinterpretation.
```

```
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyw
ord arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without
an explicit keyword will result in an error or misinterpretation.
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyw
ord arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without
an explicit keyword will result in an error or misinterpretation.
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyw
ord arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without
an explicit keyword will result in an error or misinterpretation.
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyw
ord arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without
an explicit keyword will result in an error or misinterpretation.
```




property_type
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
property_type


locality
25000
30000
35000
40000
45000
50000
locality

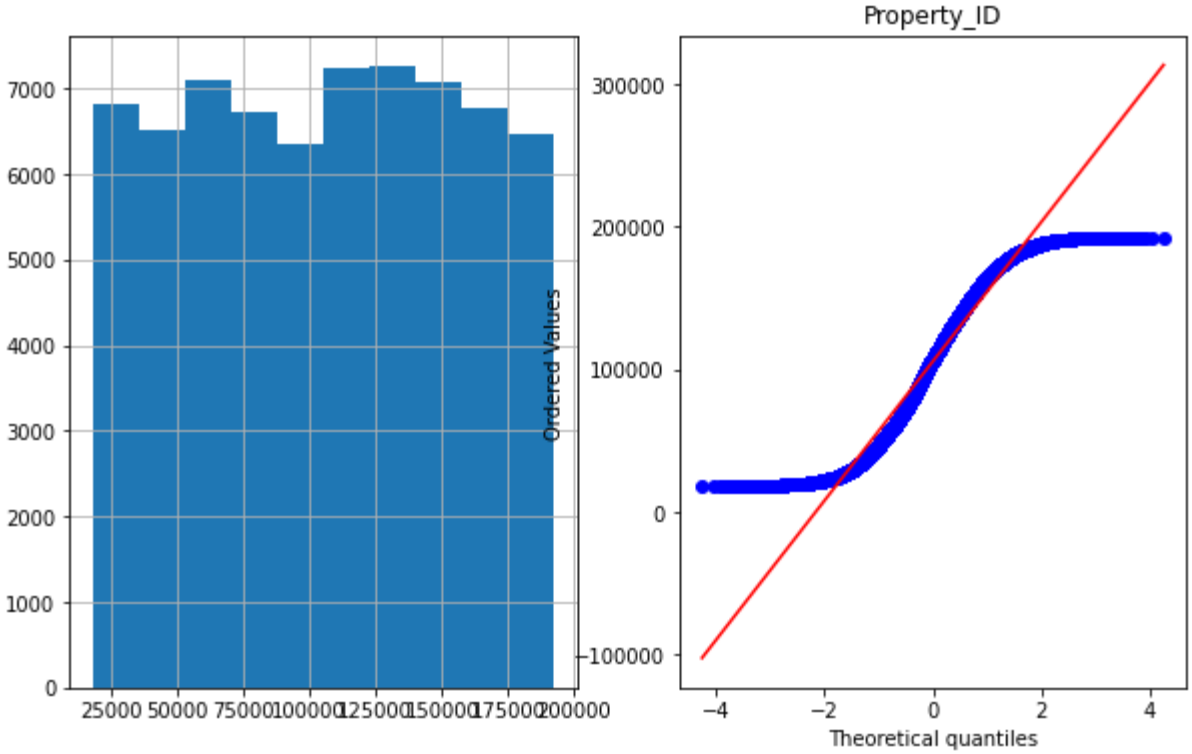

furnish_type
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
furnish_type

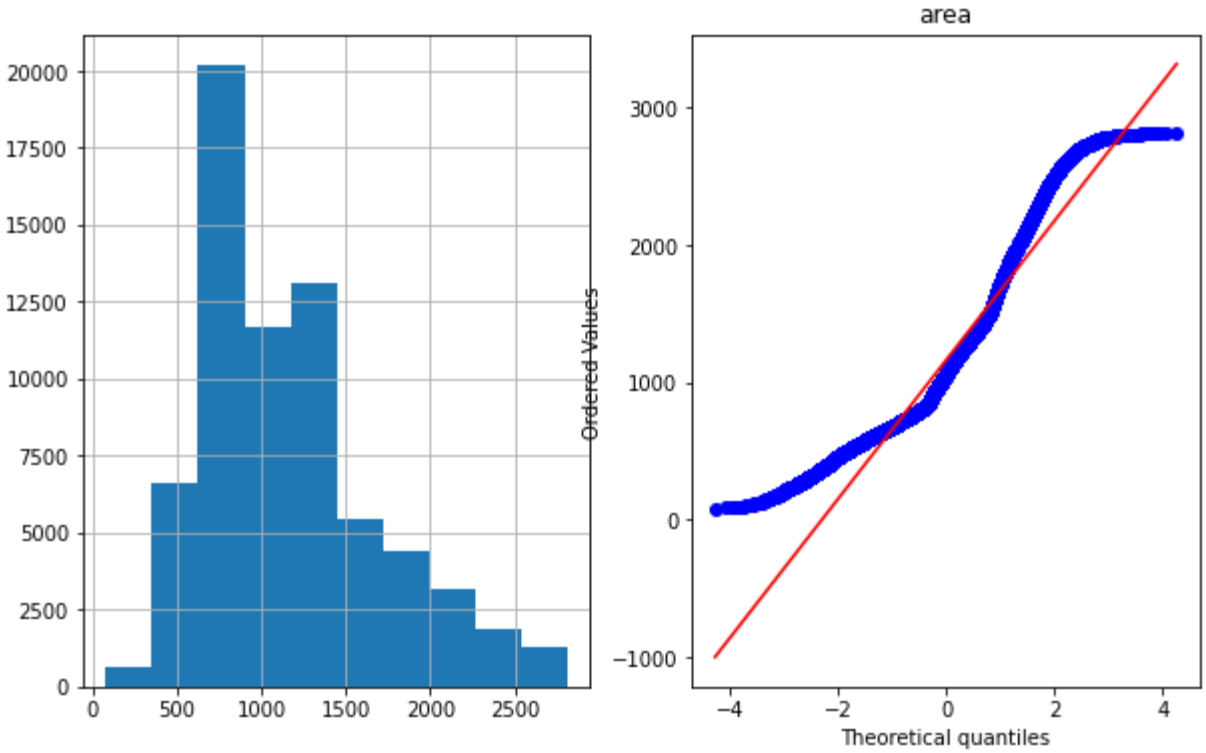
Plotting Q-Q Plot

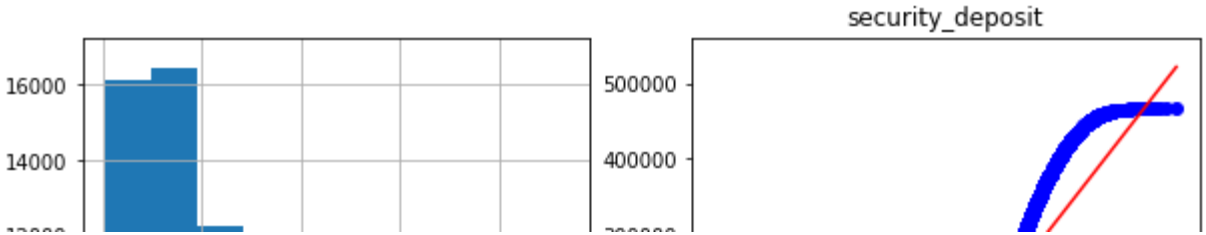
In [ ]:
```
import scipy.stats as stats
import pylab
```

In [ ]:
```
    def plot_curve(healthcare_stroke_data,feature):
        plt.figure(figsize=(10,6))
        plt.subplot(1,2,1)
        healthcare_stroke_data[feature].hist()
        plt.subplot(1,2,2)
        stats.probplot(healthcare_stroke_data[feature],dist='norm',plot=pylab)
        plt.title(feature)
        plt.show()
```

In [ ]:
```
for i in cont_var:
    plot_curve(train, i)
```


7000
6000
5000
4000
3000
2000
1000
0
25000 50000 75000 100000 125000 150000 175000 200000
Property_ID
300000
200000
100000
0
−100000
Ordered Values
−4
−2
0
2
4
Theoretical quantiles


area
Ordered Values
Theoretical quantiles
20000
17500
15000
12500
10000
7500
5000
2500
0
500
1000
1500
2000
2500
3000
2000
1000
0
−1000
−4
−2
0
2
4



Splitting the data w.r.t target columns

In [ ]:
```
train.columns
```

Out[ ]:
```
Index(['Property_ID', 'room', 'layout_type', 'property_type', 'locality',
       'price', 'area', 'furnish_type', 'bathroom', 'city', 'parking_spaces',
       'floor', 'pet_friendly', 'power_backup', 'washing_machine',
       'air_conditioner', 'geyser/solar', 'security_deposit', 'CCTV/security',
       'lift', 'neighbourhood'],
      dtype='object')
```

In [ ]:
```
X_train = train.drop(["price"], axis=1)
y_train = train["price"]
```

In [ ]:
```
X_test = test.drop(["price"], axis=1)
y_test = test["price"]
```

Scaling And Normalizarion

Robust Scaler

It is used to scale the feature to median and quantiles Scaling using median and quantiles consists of substracting the median to all the observations, and then dividing by the interquantile difference. The interquantile difference is the difference between the 75th and 25th quantile:

IQR = 75th quantile - 25th quantile

X_scaled = (X - X.median) / IQR

In [ ]:
```
from sklearn.preprocessing import RobustScaler
```

In [ ]:
```
scale=RobustScaler()
```

In [ ]:
```
scale.fit(X_train)
```

Out[ ]:
```
RobustScaler()
```

In [ ]:
```
X_train= scale.transform(X_train)
```

In [ ]:
```
train.columns
```

Out[ ]:
```
Index(['Property_ID', 'room', 'layout_type', 'property_type', 'locality',
       'price', 'area', 'furnish_type', 'bathroom', 'city', 'parking_spaces',
       'floor', 'pet_friendly', 'power_backup', 'washing_machine',
       'air_conditioner', 'geyser/solar', 'security_deposit', 'CCTV/security',
       'lift', 'neighbourhood'],
      dtype='object')
```

In [ ]:
```
test.columns
```

Out[ ]:
```
Index(['Property_ID', 'room', 'layout_type', 'property_type', 'locality',
       'area', 'furnish_type', 'bathroom', 'city', 'parking_spaces', 'floor',
       'pet_friendly', 'power_backup', 'washing_machine', 'air_conditioner',
       'geyser/solar', 'security_deposit', 'CCTV/security', 'lift',
       'neighbourhood', 'price'],
      dtype='object')
```

In [ ]:
```
X_train=pd.DataFrame(X_train)
```

In [ ]:
```
X_train.head(3)
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	-0.756109	0.5	0.0	2.0	-0.165137	0.375559	1.0	0.0	1.445152	0.0	-0.727273	0.0	0.0	0.0	0.0	0.0	1.596230	-1.0	-1.0	-0.894737
1	-0.188037	-0.5	0.0	0.0	0.276423	0.616990	0.0	0.0	0.885809	-1.0	0.727273	-1.0	1.0	0.0	-1.0	1.0	-0.648747	-1.0	0.0	-0.210526

In [ ]:
```
X_train.rename(columns = {0:'Property_ID',
                          1:'room',
                          2:'layout_type',
                          3:'property_type',
                          4:'locality',
                          5:'area',
                          6:'furnish_type',
                          7:'bathroom',
                          8:'city',
                          9:'parking_spaces',
                          10:'floor',
                          11:'pet_friendly',
                          12:'power_backup',
                          13:'washing_machine',
                          14:'air_conditioner',
                          15:'geyser/solar',
                          16:'security_deposit',
                          17:'CCTV/security',
                          18:'lift',
                          19:'neighbourhood'}, inplace = True)
```

In [ ]:
```
X_train.head(3)
```

Out[ ]:

	Property_ID	room	layout_type	property_type	locality	area	furnish_type	bathroom	city	parking_spaces	floor	pet_frie
0	-0.756109	0.5	0.0	2.0	-0.165137	0.375559	1.0	0.0	1.445152	0.0	-0.727273	
1	-0.188037	-0.5	0.0	0.0	0.276423	0.616990	0.0	0.0	0.885809	-1.0	0.727273	
2	-0.082244	0.5	0.0	0.0	-1.875718	1.157973	-1.0	0.0	0.145326	-1.0	0.090909	

In [ ]:
```
scale.fit(X_test)
```

Out[ ]:
```
RobustScaler()
```

In [ ]:
```
X_test= scale.transform(X_test)
```

In [ ]:
```
X_test=pd.DataFrame(X_test)
```

In [ ]:
```
test.columns
```

Out[ ]:
```
Index(['Property_ID', 'room', 'layout_type', 'property_type', 'locality',
       'area', 'furnish_type', 'bathroom', 'city', 'parking_spaces', 'floor',
       'pet_friendly', 'power_backup', 'washing_machine', 'air_conditioner',
       'geyser/solar', 'security_deposit', 'CCTV/security', 'lift',
       'neighbourhood', 'price'],
      dtype='object')
```

In [ ]:
```
X_test.rename(columns = {0:'Property_ID',
                         1:'room',
                         2:'layout_type',
                         3:'property_type',
                         4:'locality',
                         5:'area',
                         6:'furnish_type',
                         7:'bathroom',
                         8:'city',
                         9:'parking_spaces',
                         10:'floor',
                         11:'pet_friendly',
                         12:'power_backup',
                         13:'washing_machine',
                         14:'air_conditioner',
                         15:'geyser/solar',
                         16:'security_deposit',
                         17:'CCTV/security',
                         18:'lift',
                         19:'neighbourhood'}, inplace = True)
```

In [ ]:
```
X_test.head(3)
```

Out[ ]:

	Property_ID	room	layout_type	property_type	locality	area	furnish_type	bathroom	city	parking_spaces	floor	pet_friendly
0	0.182038	0.0	0.0	1.0	0.372607	0.290886	-1.0	-1.0	0.25	-1.0	-0.636364	-1.0
1	-0.083677	-0.5	0.0	2.0	0.787187	-0.599251	-1.0	1.0	-0.25	0.0	-0.454545	-1.0
2	-0.116950	-0.5	0.0	0.0	0.891753	-0.736579	0.0	-1.0	-0.25	0.0	-0.545455	0.0

Guassian Transformation

Some machine learning algorithms like linear and logistic assume that the features are normally distributed -Accuracy -Performance

- logarithmic transformation
- reciprocal transformation
- square root transformation
- exponential transformation (more general, you can use any exponent)
- boxcox transformation

In [ ]:

```
import scipy.stats as stat
import pylab
```

In [ ]:

```
#### If you want to check whether feature is guassian or normal distributed
#### Q-Q plot
def plot_data(df,feature):
    plt.figure(figsize=(10,6))
    plt.subplot(1,2,1)
    df[feature].hist()
    plt.subplot(1,2,2)
    stat.probplot(df[feature],dist='norm',plot=pylab)
    plt.show()
```

In [ ]:

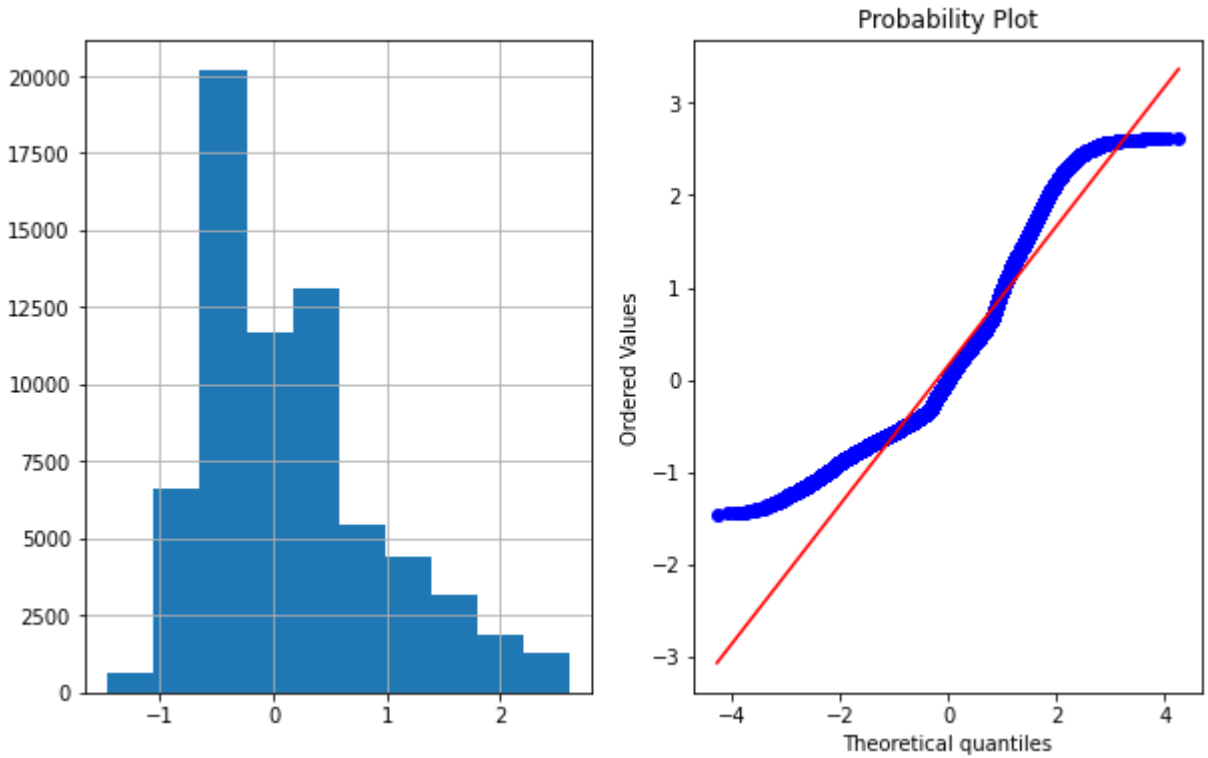
```
X_train.columns
```

Out[ ]:

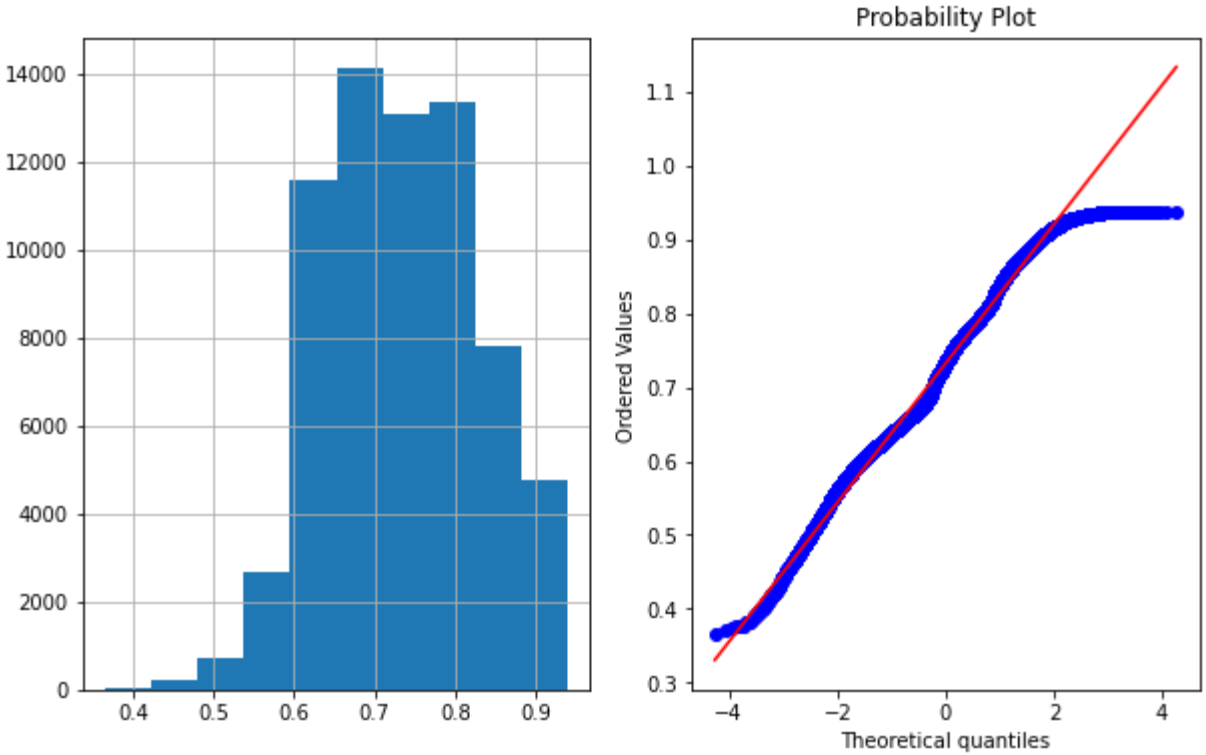
```
Index(['Property_ID', 'room', 'layout_type', 'property_type', 'locality',
       'area', 'furnish_type', 'bathroom', 'city', 'parking_spaces', 'floor',
       'pet_friendly', 'power_backup', 'washing_machine', 'air_conditioner',
```

```
       'geyser/solar', 'security_deposit', 'CCTV/security', 'lift',
       'neighbourhood'],
      dtype='object')
```

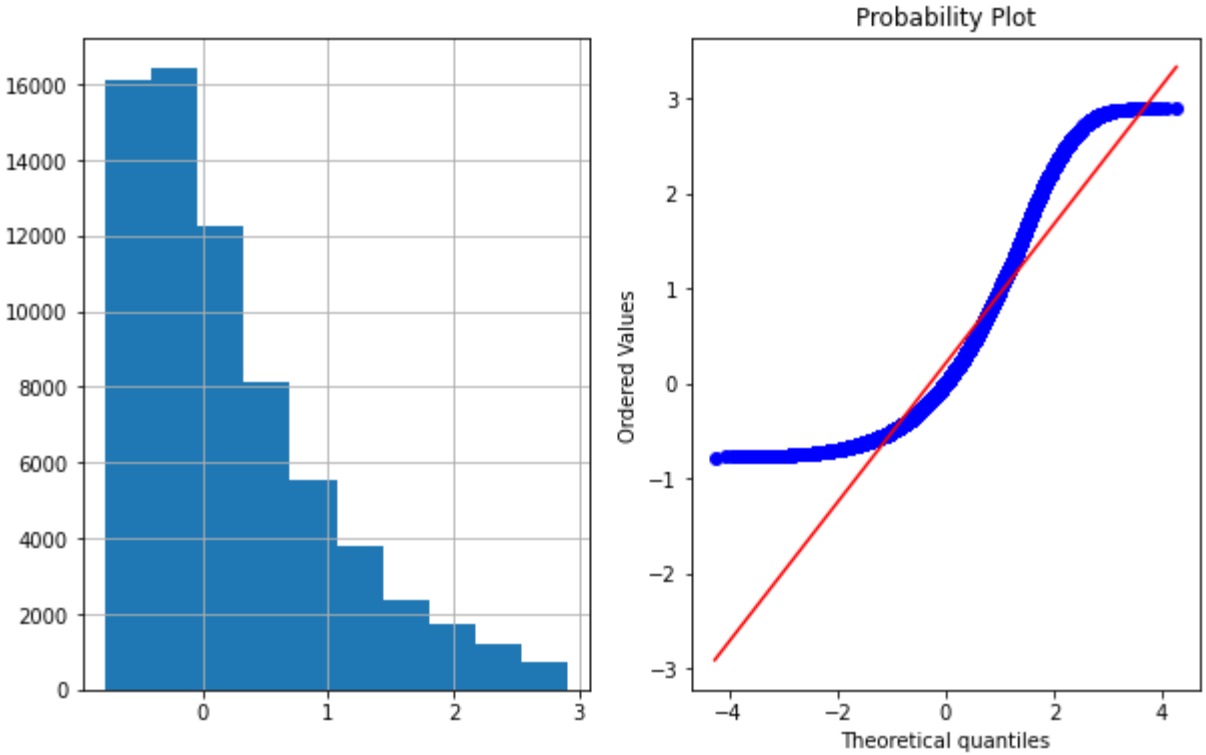
In [ ]:
```
plot_data(X_train,'area')
```



In [ ]:
```
X_train['area'],parameters=stat.boxcox(X_train['area']+3)
plot_data(X_train,'area')
```

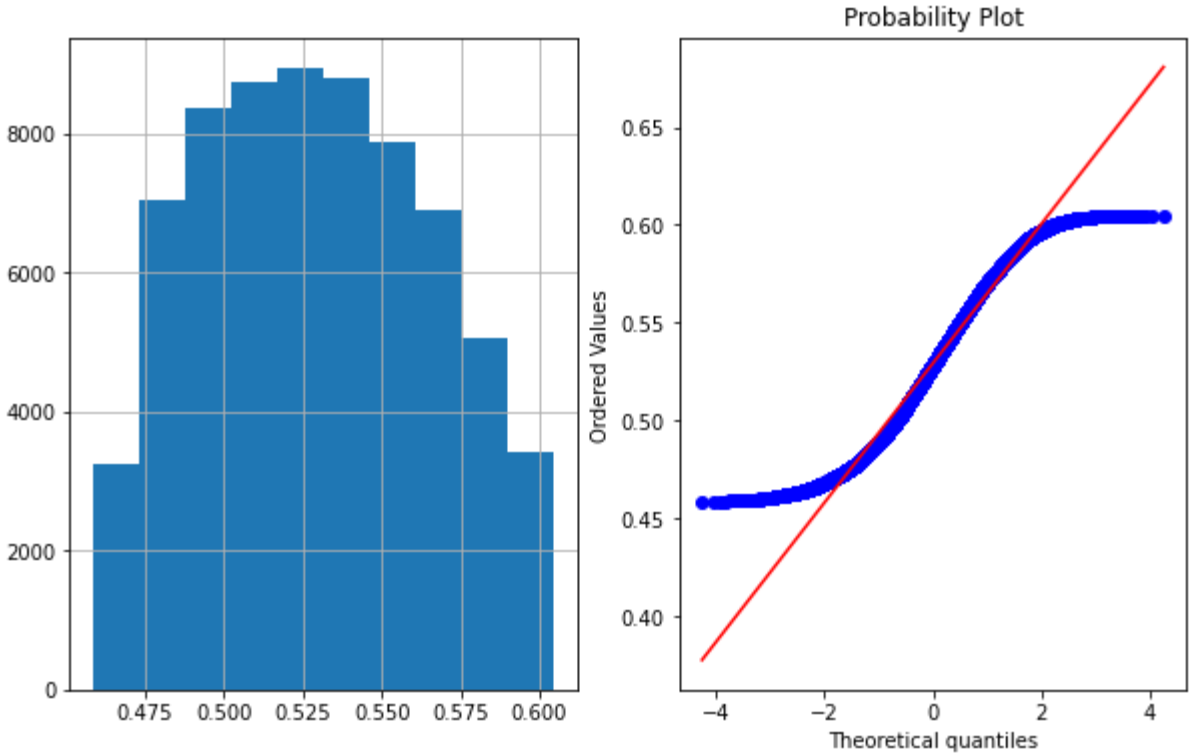


In [ ]:
```
plot_data(X_train,'security_deposit')
```



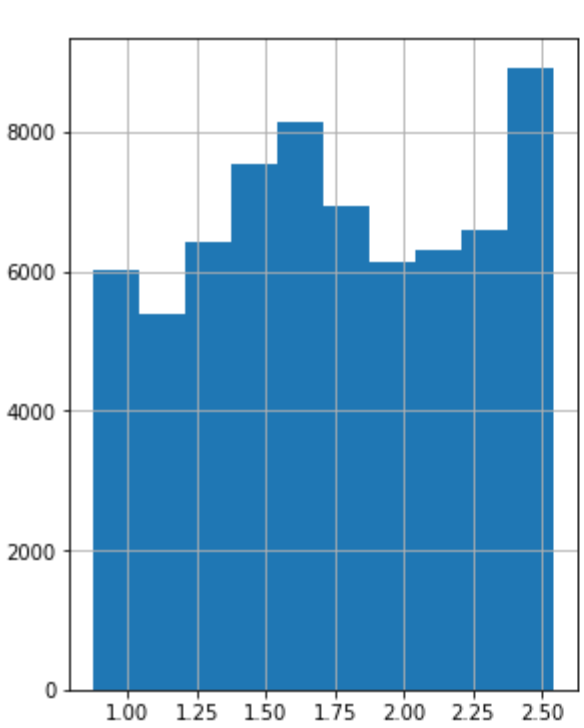
In [ ]:

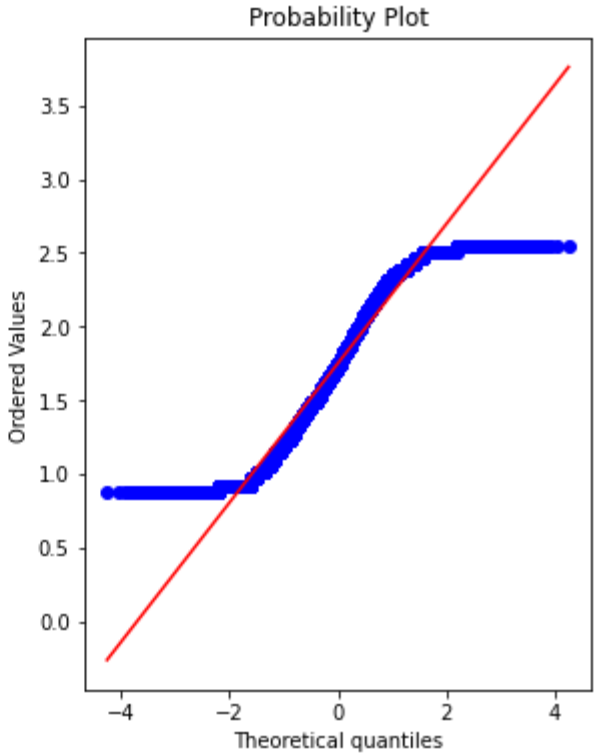
```
X_train['security_deposit'],parameters=stat.boxcox(X_train['security_deposit']+3)
plot_data(X_train,'security_deposit')
```



In [ ]:

```
##### Square Root Transformation
X_train['neighbourhood'],parameters=stat.boxcox(X_train['neighbourhood']+3)
plot_data(X_train,'neighbourhood')
```





In [ ]:
```
print(X_train.shape,X_test.shape)
print(y_train.shape,y_test.shape)
```

```
(68352, 20) (57722, 20)
(68352,) (57722,)
```

In [ ]:
```
from sklearn.linear_model import Lasso

from sklearn.feature_selection import SelectFromModel
```

In [ ]:
```
model = SelectFromModel(Lasso(alpha=0.005,random_state=0))
```

In [ ]:
```
model.fit(X_train,y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:648: ConvergenceWarning: Objective
```

```
did not converge. You might want to increase the number of iterations, check the scale of the features or consider inc
reasing regularisation. Duality gap: 3.153e+10, tolerance: 1.332e+09
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
```

Out[ ]:
```
SelectFromModel(estimator=Lasso(alpha=0.005, random_state=0))
```

In [ ]:
```
model.get_support()
```

Out[ ]:
```
array([ True,  True, False,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True])
```

In [ ]:
```
selected_features = X_train.columns[(model.get_support())]
```

In [ ]:
```
selected_features
```

Out[ ]:
```
Index(['Property_ID', 'room', 'property_type', 'locality', 'area',
       'furnish_type', 'bathroom', 'city', 'parking_spaces', 'floor',
       'pet_friendly', 'power_backup', 'washing_machine', 'air_conditioner',
       'geyser/solar', 'security_deposit', 'CCTV/security', 'lift',
       'neighbourhood'],
      dtype='object')
```

Correlation - To Check `Multicollinearity`

In [ ]:
```
X_train.corr()
```

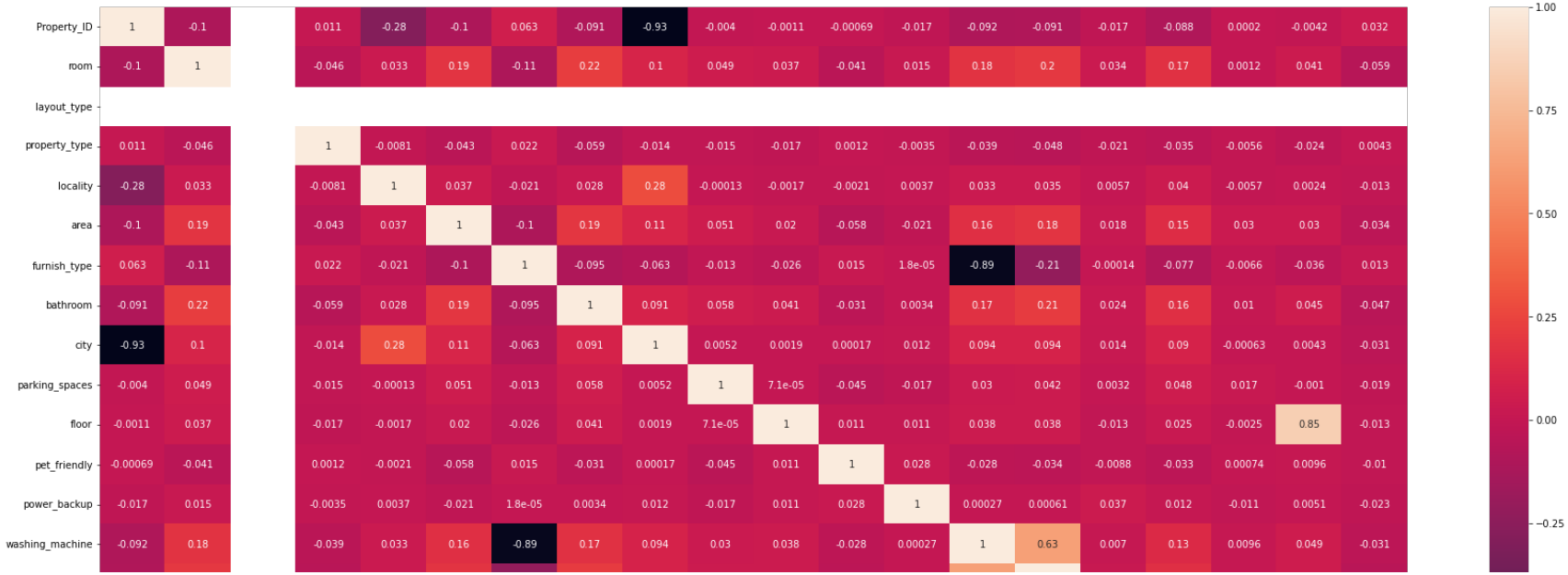
Out[ ]:

	Property_ID	room	layout_type	property_type	locality	area	furnish_type	bathroom	city	parking_space
Property_ID	1.000000	-0.099695	NaN	0.011123	-0.280253	-0.103615	0.062827	-0.090696	-0.926966	-0.00404
room	-0.099695	1.000000	NaN	-0.046251	0.032943	0.186472	-0.105143	0.223951	0.102051	0.04878
layout_type	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
property_type	0.011123	-0.046251	NaN	1.000000	-0.008093	-0.043294	0.021501	-0.059230	-0.013794	-0.01528
locality	-0.280253	0.032943	NaN	-0.008093	1.000000	0.036961	-0.021122	0.028094	0.284135	-0.00013
area	-0.103615	0.186472	NaN	-0.043294	0.036961	1.000000	-0.101525	0.192941	0.105968	0.05128

	Property_ID	room	layout_type	property_type	locality	area	furnish_type	bathroom	city	parking_space
furnish_type	0.062827	-0.105143	NaN	0.021501	-0.021122	-0.101525	1.000000	-0.094522	-0.063494	-0.01313
bathroom	-0.090696	0.223951	NaN	-0.059230	0.028094	0.192941	-0.094522	1.000000	0.090727	0.05754
city	-0.926966	0.102051	NaN	-0.013794	0.284135	0.105968	-0.063494	0.090727	1.000000	0.00524
parking_spaces	-0.004044	0.048786	NaN	-0.015283	-0.000135	0.051283	-0.013132	0.057549	0.005247	1.00000
floor	-0.001097	0.037129	NaN	-0.016527	-0.001679	0.020442	-0.026146	0.041290	0.001859	0.00007
pet_friendly	-0.000686	-0.040641	NaN	0.001205	-0.002054	-0.057660	0.015422	-0.031466	0.000174	-0.04497
power_backup	-0.016903	0.014800	NaN	-0.003481	0.003687	-0.021124	0.000018	0.003421	0.012138	-0.01721
washing_machine	-0.091852	0.175049	NaN	-0.039140	0.032966	0.164374	-0.894251	0.171517	0.093817	0.02964
air_conditioner	-0.091267	0.199302	NaN	-0.048055	0.035237	0.182291	-0.213400	0.210058	0.094398	0.04188
geyser/solar	-0.017237	0.033690	NaN	-0.020548	0.005657	0.018211	-0.000140	0.024340	0.014239	0.00324
security_deposit	-0.087995	0.165611	NaN	-0.034947	0.039670	0.147033	-0.076804	0.163044	0.089832	0.04838
CCTV/security	0.000195	0.001193	NaN	-0.005569	-0.005708	0.029792	-0.006598	0.010218	-0.000626	0.01723
lift	-0.004166	0.041422	NaN	-0.024008	0.002418	0.029739	-0.035730	0.045064	0.004327	-0.00104

In [ ]:
```
import seaborn as sns
corr=X_train.corr()
top_features=corr.index
plt.figure(figsize=(30,15))
sns.heatmap(X_train[top_features].corr(),annot=True)
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fdb43009d90>



In [ ]:
```
threshold=0.7
```

In [ ]:
```
# find and remove correlated features
def correlation(dataset, threshold):
    col_corr = set()  # Set of all the names of correlated columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff value
                colname = corr_matrix.columns[i]  # getting the name of column
                col_corr.add(colname)
    return col_corr
```

In [ ]:
```
correlation(X_train,threshold)
```

Out[ ]:
```
{'city', 'lift', 'washing_machine'}
```

In [ ]:
```
y_train[:100]
```

Out[ ]:
```
0      33624
1       9655
2      23699
3       6306
8      11615
       ...
196    18106
197     8291
200    38619
201    48853
202    24598
Name: price, Length: 100, dtype: int64
```

In [ ]:
```
#### Information Gain
```

In [ ]:
```
#from sklearn.feature_selection import mutual_info_regression
```

In [ ]:
```
#mutual_info=mutual_info_regression(X_train,y_train)
```

In [ ]:
```
#mutual_data=pd.Series(mutual_info,index=X_train.columns)
#mutual_data.sort_values(ascending=False)
```

In [ ]:
```
#mutual_info = pd.Series(mutual_info)
#mutual_info.index = X_train.columns
```

In [ ]:
```
#mutual_info.sort_values(ascending=False).plot.bar(figsize=(15,5))
```

In [ ]:
```
X_train.columns
```

Out[ ]:
```
Index(['Property_ID', 'room', 'layout_type', 'property_type', 'locality',
       'area', 'furnish_type', 'bathroom', 'city', 'parking_spaces', 'floor',
       'pet_friendly', 'power_backup', 'washing_machine', 'air_conditioner',
       'geyser/solar', 'security_deposit', 'CCTV/security', 'lift',
       'neighbourhood'],
      dtype='object')
```

In [ ]:
```
X_test.columns
```

Out[ ]:
```
Index(['Property_ID', 'room', 'layout_type', 'property_type', 'locality',
       'area', 'furnish_type', 'bathroom', 'city', 'parking_spaces', 'floor',
       'pet_friendly', 'power_backup', 'washing_machine', 'air_conditioner',
       'geyser/solar', 'security_deposit', 'CCTV/security', 'lift',
       'neighbourhood'],
      dtype='object')
```

In [ ]:
```
X_train = X_train.drop(columns="Property_ID")
X_test = X_test.drop(columns="Property_ID")
```

Checking the Multi-collinearity

In [ ]:
```
X_train.corr()
```

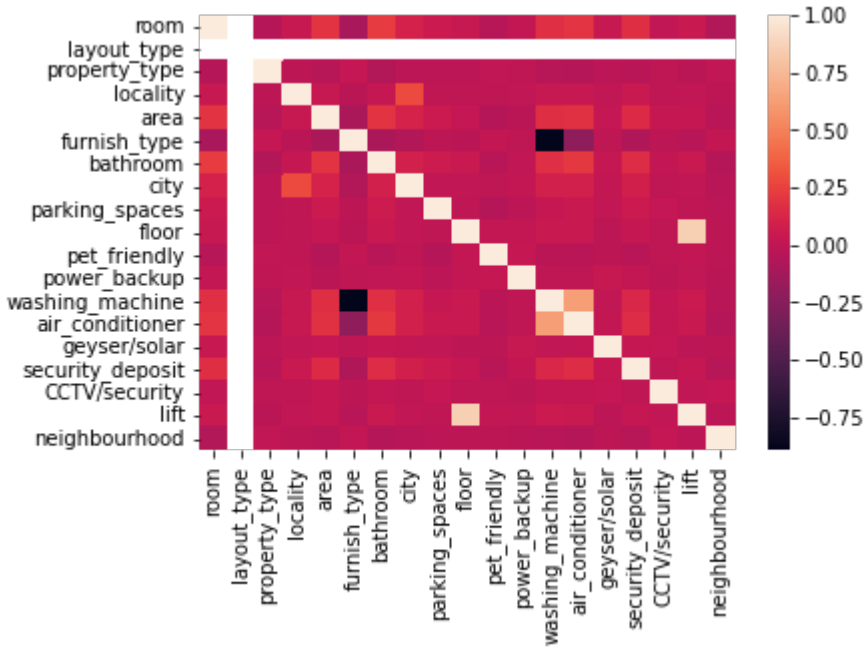
Out[ ]:

	room	layout_type	property_type	locality	area	furnish_type	bathroom	city	parking_spaces	floor
room	1.000000	NaN	-0.046251	0.032943	0.186472	-0.105143	0.223951	0.102051	0.048786	0.037129
layout_type	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
property_type	-0.046251	NaN	1.000000	-0.008093	-0.043294	0.021501	-0.059230	-0.013794	-0.015283	-0.016527
locality	0.032943	NaN	-0.008093	1.000000	0.036961	-0.021122	0.028094	0.284135	-0.000135	-0.001679
area	0.186472	NaN	-0.043294	0.036961	1.000000	-0.101525	0.192941	0.105968	0.051283	0.020442
furnish_type	-0.105143	NaN	0.021501	-0.021122	-0.101525	1.000000	-0.094522	-0.063494	-0.013132	-0.026146
bathroom	0.223951	NaN	-0.059230	0.028094	0.192941	-0.094522	1.000000	0.090727	0.057549	0.041290
city	0.102051	NaN	-0.013794	0.284135	0.105968	-0.063494	0.090727	1.000000	0.005247	0.001859
parking_spaces	0.048786	NaN	-0.015283	-0.000135	0.051283	-0.013132	0.057549	0.005247	1.000000	0.000071
floor	0.037129	NaN	-0.016527	-0.001679	0.020442	-0.026146	0.041290	0.001859	0.000071	1.000000
pet_friendly	-0.040641	NaN	0.001205	-0.002054	-0.057660	0.015422	-0.031466	0.000174	-0.044972	0.010832
power_backup	0.014800	NaN	-0.003481	0.003687	-0.021124	0.000018	0.003421	0.012138	-0.017219	0.011336
washing_machine	0.175049	NaN	-0.039140	0.032966	0.164374	-0.894251	0.171517	0.093817	0.029648	0.038329

	room	layout_type	property_type	locality	area	furnish_type	bathroom	city	parking_spaces	floor
air_conditioner	0.199302	NaN	-0.048055	0.035237	0.182291	-0.213400	0.210058	0.094398	0.041886	0.038207
geyser/solar	0.033690	NaN	-0.020548	0.005657	0.018211	-0.000140	0.024340	0.014239	0.003249	-0.012572
security_deposit	0.165611	NaN	-0.034947	0.039670	0.147033	-0.076804	0.163044	0.089832	0.048389	0.025090
CCTV/security	0.001193	NaN	-0.005569	-0.005708	0.029792	-0.006598	0.010218	-0.000626	0.017230	-0.002496
lift	0.041422	NaN	-0.024008	0.002418	0.029739	-0.035730	0.045064	0.004327	-0.001042	0.854527

In [ ]:
```
sns.heatmap(X_train.corr())
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdb3d63c7d0>
```



In [ ]:
```
threshold = 0.7
```

In [ ]:
```
correlation(X_train,threshold)
```

Out[ ]: {'lift', 'washing_machine'}

PIPELINE CREATION TO CHECK WHICH MODEL WORKS BETTER

In [ ]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
```

In [ ]:
```
pipeline_lr=Pipeline([('scalar1',RobustScaler()),
                      ('pca1',PCA(n_components=2)),
                      ('lr_classifier',LinearRegression())])
```

In [ ]:
```
pipeline_dt=Pipeline([('scalar2',RobustScaler()),
                      ('pca2',PCA(n_components=2)),
                      ('dt_classifier',DecisionTreeRegressor())])
```

In [ ]:
```
pipeline_randomforest=Pipeline([('scalar3',RobustScaler()),
                      ('pca3',PCA(n_components=2)),
                      ('rf_classifier',RandomForestRegressor())])
```

In [ ]:
```
pipeline_gradient_boost=Pipeline([('scalar4',RobustScaler()),
                      ('pca4',PCA(n_components=2)),
                      ('gb_classifier',GradientBoostingRegressor())])
```

In [ ]:
```
pipeline_XGboost=Pipeline([('scalar5',RobustScaler()),
                           ('pca5',PCA(n_components=2)),
                           ('xgb_classifier',XGBRegressor())])
```

In [ ]:
```
## LEts make the list of pipelines
pipelines = [pipeline_lr, pipeline_dt, pipeline_randomforest,pipeline_gradient_boost,pipeline_XGboost]
```

In [ ]:
```
best_accuracy=0.0
best_regressor=0
best_pipeline=""
```

In [ ]:
```
from sklearn.metrics import r2_score
```

In [ ]:
```
# Dictionary of pipelines and classifier types for ease of reference
pipe_dict = {0: 'Logistic Regression', 1: 'Decision Tree', 2: 'RandomForest', 3: 'Gradient Boost', 4: 'Extreme Gradien

# Fit the pipelines
for pipe in pipelines:
        pipe.fit(X_train, y_train)
```

```
[11:25:00] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squar
ederror.
```

In [ ]:
```
for i,model in enumerate(pipelines):
    print("{} Test Accuracy: {}".format(pipe_dict[i],model.score(X_train, y_train)))
```

```
Logistic Regression Test Accuracy: 0.06815937692973595
Decision Tree Test Accuracy: 0.4773327185635178
RandomForest Test Accuracy: 0.6599476639548443
Gradient Boost Test Accuracy: 0.1319582580960763
Extreme Gradient Boost Test Accuracy: 0.13279612307271726
```

Random Forest Regressor

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
ranfor= RandomForestRegressor()
ranfor.fit(X_train,y_train)
y_pred_rf= ranfor.predict(X_test)
from sklearn.metrics import r2_score
print("The train score is ", ranfor.score(X_train, y_train))
print("The test score is ", ranfor.score(X_test, y_pred_rf))
```

```
The train score is  0.9414880547956835
The test score is  1.0
```

XG Boost Regressor

In [ ]:
```
from xgboost import XGBRegressor
xgb= XGBRegressor()
xgb.fit(X_train,y_train)
ypred_xgb= xgb.predict(X_test)
print("The train score is ", xgb.score(X_train, y_train))
print("The test score is ", xgb.score(X_test, ypred_xgb))
```

```
[11:26:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squar
ederror.
The train score is  0.6060878382021276
The test score is  1.0
```

In [ ]:
```
#! pip install scikit_optimize
```

In [ ]:
```
import lightgbm as lgb
from skopt import BayesSearchCV
from sklearn.model_selection import StratifiedKFold
```

In [ ]:
```
import xgboost as xgb
from xgboost.sklearn import XGBRegressor
from sklearn.ensemble import GradientBoostingRegressor
```

In [ ]:
```
#! pip install catboost
```

In [ ]:
```
# Import packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from keras.models import Sequential
from keras.layers import Dense, BatchNormalization, Dropout
from tensorflow.keras.optimizers import Adam, SGD, RMSprop, Adadelta, Adagrad, Adamax, Nadam, Ftrl
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.wrappers.scikit_learn import KerasClassifier
from math import floor
from sklearn.metrics import make_scorer, accuracy_score
from bayes_opt import BayesianOptimization
from sklearn.model_selection import StratifiedKFold
from keras.layers import LeakyReLU
LeakyReLU = LeakyReLU(alpha=0.1)
import warnings
warnings.filterwarnings('ignore')
pd.set_option("display.max_columns", None)
```

In [ ]:
```
##!pip install bayesian-optimization
```

In [ ]:
```
###!pip install keras-tuner
```

In [ ]:
```
# SETTINGS - CHANGE THESE TO GET SOMETHING MEANINGFUL
ITERATIONS = 19 # 1000
```

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

In [ ]:
```
import lightgbm as lgb
from skopt import BayesSearchCV
from sklearn.model_selection import StratifiedKFold
```

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

In [ ]:
```
X_train.head(3)
```

Out[ ]:

	room	layout_type	property_type	locality	area	furnish_type	bathroom	city	parking_spaces	floor	pet_friendly	power_l
0	0.5	0.0	2.0	-0.165137	0.779641	1.0	0.0	1.445152	0.0	-0.727273	0.0	
1	-0.5	0.0	0.0	0.276423	0.805192	0.0	0.0	0.885809	-1.0	0.727273	-1.0	
2	0.5	0.0	0.0	-1.875718	0.852664	-1.0	0.0	0.145326	-1.0	0.090909	0.0	

In [ ]:
```
X_train.columns
```

Out[ ]:
```
Index(['room', 'layout_type', 'property_type', 'locality', 'area',
       'furnish_type', 'bathroom', 'city', 'parking_spaces', 'floor',
       'pet_friendly', 'power_backup', 'washing_machine', 'air_conditioner',
       'geyser/solar', 'security_deposit', 'CCTV/security', 'lift',
       'neighbourhood'],
      dtype='object')
```

In [ ]:
```
X_test.head(3)
```

Out[ ]:

	room	layout_type	property_type	locality	area	furnish_type	bathroom	city	parking_spaces	floor	pet_friendly	power_back
0	0.0	0.0	1.0	0.372607	0.290886	-1.0	-1.0	0.25	-1.0	-0.636364	-1.0	
1	-0.5	0.0	2.0	0.787187	-0.599251	-1.0	1.0	-0.25	0.0	-0.454545	-1.0	
2	-0.5	0.0	0.0	0.891753	-0.736579	0.0	-1.0	-0.25	0.0	-0.545455	0.0	

In [ ]:
```
# Make scorer accuracy
score_acc = make_scorer(accuracy_score)
```

In [ ]:
```
from keras.wrappers.scikit_learn import KerasRegressor
from tensorflow.keras.optimizers import RMSprop

from tensorflow.keras import Sequential # import Sequential from tensorflow.keras
from tensorflow.keras.layers import Dense # import Dense from tensorflow.keras.layers
from numpy.random import seed # seed helps you to fix the randomness in the neural network.
import tensorflow
```

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

In [ ]:
```
from tensorflow.keras.losses import mean_squared_error
```

In [ ]:
```
from numpy.random import seed # seed helps you to fix the randomness in the neural network.
```

In [ ]:
```
# Import packages
# Basic packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import pickle
from math import floor
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import MinMaxScaler

# Evaluation and bayesian optimization
from sklearn.metrics import make_scorer, mean_absolute_error
from sklearn.metrics import mean_squared_error as MSE
from hyperopt import hp, fmin, tpe
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from bayes_opt import BayesianOptimization

import warnings
warnings.filterwarnings('ignore')
pd.set_option("display.max_columns", None)
```

Bayesian Optimization Of Neural Network

In [ ]:
```
# Make scorer: MSE
mse = make_scorer(MSE, greater_is_better=False)
```

In [189…
```
print(X_train.shape)
```

```
(68352, 19)
```

In [190…
```
print(X_test.shape)
```

```
(57722, 19)
```

In [191…

```
    # Hyperparameter-tuning: Bayesian Optimization, bayes_opt
def nn_re_bo(neurons, activation, optimizer, batch_size, epochs, layers1, layers2, dropout, dropout_rate):
    optimizerL = ['Adam', 'RMSprop', 'Adadelta', 'Adagrad', 'Adamax', 'Nadam', 'Ftrl', 'SGD']
    activationL = ['relu', 'sigmoid', 'softplus', 'softsign', 'tanh', 'selu',
                   'elu', 'exponential', 'LeakyReLU']

    neurons = round(neurons)
    activation = activationL[floor(activation)]
    optimizer = optimizerL[floor(optimizer)]
    batch_size = round(batch_size)
    epochs = round(epochs)
    layers1 = round(layers1)
    layers2 = round(layers2)

    def nn_re_fun():
        nn = Sequential()
        nn.add(Dense(neurons, input_dim=19, activation=activation))
        for i in range(layers1):
            nn.add(Dense(neurons, activation=activation))
        if dropout > 0.5:
            nn.add(Dropout(dropout_rate, seed=123))
        for i in range(layers2):
            nn.add(Dense(neurons, activation=activation))
        nn.add(Dense(1, activation='linear'))
        nn.compile(loss='mean_squared_error', optimizer=optimizer)
        return nn

    nn = KerasRegressor(build_fn=nn_re_fun, epochs=epochs, batch_size=batch_size, verbose=0)

    kfold = KFold(n_splits=12, shuffle=True, random_state=123)
    scores =  cross_val_score(nn, X_train, y_train, cv=kfold).mean()
    score = ((scores*-1)**0.5)*-1
    return score
```

In [ ]:

```
# Set paramaters
# Set hyperparameters spaces
params_nn ={
    'neurons': (10, 280),
    'activation':(0, 9),
    'optimizer':(0,7),
    'batch_size':(200, 500),
    'epochs':(10, 60),
    'layers1':(1,3),
    'layers2':(1,3),
    'dropout':(0,1),
    'dropout_rate':(0,0.3)
}

# Run Bayesian Optimization
nn_bo = BayesianOptimization(nn_re_bo, params_nn, random_state=123)
nn_bo.maximize(init_points=11, n_iter=5)
```

```
|   iter    |  target   | activa... | batch_... |  dropout  | dropou... |  epochs   |  layers1  |  layers2  |  neurons
| optimizer |
-------------------------------------------------------------------------------------------------------------------------
--------------
```

In [174…

```
# Best hyperparameters
params_nn = nn_bo.max['params']

optimizerL = ['Adam', 'RMSprop', 'Adadelta', 'Adagrad', 'Adamax', 'Nadam', 'Ftrl','SGD', 'SGD']
activationL = ['relu', 'sigmoid', 'softplus', 'softsign', 'tanh', 'selu',
               'elu', 'exponential', 'LeakyReLU']

params_nn['neurons'] = round(params_nn['neurons'])
params_nn['activation'] = activationL[floor(params_nn['activation'])]
params_nn['optimizer'] = optimizerL[round(params_nn['optimizer'])]
params_nn['batch_size'] = round(params_nn['batch_size'])
params_nn['epochs'] = round(params_nn['epochs'])
params_nn['layers1'] = round(params_nn['layers1'])
params_nn['layers2'] = round(params_nn['layers2'])
params_nn
```

Out[174…

```
{'activation': 'relu',
```

```
'batch_size': 406,
'dropout': 0.7922002190758349,
'dropout_rate': 0.13381004700866558,
'epochs': 21,
'layers1': 3,
'layers2': 2,
'neurons': 117,
```

In [175…

```
params_nn['neurons']
```

Out[175… 117

In [176…

```
# Fitting the training data
def nn_re_fun():
    nn = Sequential()
    nn.add(Dense(params_nn['neurons'], input_dim=19, activation=params_nn['activation']))
    for i in range(params_nn['layers1']):
        nn.add(Dense(params_nn['neurons'], activation=params_nn['activation']))
    if params_nn['dropout'] > 0.5:
        nn.add(Dropout(params_nn['dropout_rate'], seed=123))
    for i in range(params_nn['layers2']):
        nn.add(Dense(params_nn['neurons'], activation=params_nn['activation']))
    nn.add(Dense(1, activation='linear'))
    nn.compile(loss='mean_squared_error', optimizer=params_nn['optimizer'], metrics=['mse'])
    return nn
```

In [177…

```
nn_hyp = KerasRegressor(build_fn=nn_re_fun, epochs=params_nn['epochs'], batch_size=params_nn['batch_size'], verbose=0)

nn_hyp.fit(X_train, y_train, validation_data=(X_test, y_test), verbose=0)

# Predict the validation data
pred_nn = nn_hyp.predict(X_test)

# Compute RMSE
rmse_nn = MSE(y_test, pred_nn)**0.5
```

In [178…

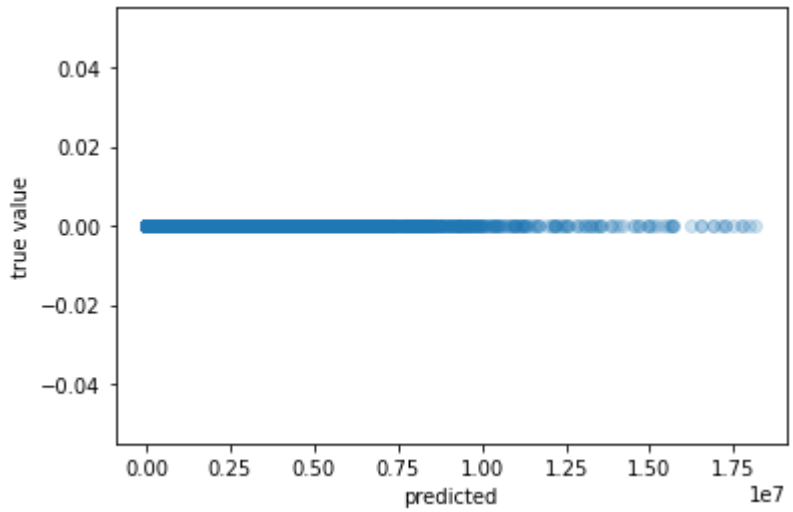
```
rmse_nn
```

Out[178… 990368.5831882139

In [179…
```
MSE(y_test, pred_nn)
```

Out[179… 980829930566.2301

In [180…
```
# Scatter plot true and predicted values
plt.scatter(pred_nn, y_test, alpha=0.2)
plt.xlabel('predicted')
plt.ylabel('true value')

plt.show()
```



In [181…
```
train_error =  np.abs(y_test - pred_nn)
mean_error = np.mean(train_error)
min_error = np.min(train_error)
max_error = np.max(train_error)
std_error = np.std(train_error)
```

In [182…

```
print(train_error)
print(mean_error)
print(min_error)
print(max_error)
print(std_error)
```

```
0        1577.087158
1        1545.876587
2        1689.401978
3        1959.959595
4        1408.594849
            ...
57717    1591.878662
57718    1525.145752
57719    1492.589966
57720    2002.231934
57721    1514.458740
Name: price, Length: 57722, dtype: float64
259397.55647654214
765.1094970703125
18146916.0
955794.3493556759
```

In [183…

```
y_pred = pd.DataFrame(pred_nn)
```

In [184…

```
y_pred = y_pred.rename(columns={0: "price"})
```

In [185…

```
y_pred.head(10)
```

Out[185…

	price
0	1577.087158
1	1545.876587
2	1689.401978
3	1959.959595
4	1408.594849

	price
5	202640.125000
6	1218.831421
7	1705.651123
8	11003.796875

In [186…

```
test.columns
```

Out[186…

```
Index(['Property_ID', 'room', 'layout_type', 'property_type', 'locality',
       'area', 'furnish_type', 'bathroom', 'city', 'parking_spaces', 'floor',
       'pet_friendly', 'power_backup', 'washing_machine', 'air_conditioner',
       'geyser/solar', 'security_deposit', 'CCTV/security', 'lift',
       'neighbourhood', 'price'],
      dtype='object')
```

In [187…

```
test.head(3)
```

Out[187…

	Property_ID	room	layout_type	property_type	locality	area	furnish_type	bathroom	city	parking_spaces	floor	pet_friendly	power_ba
0	114342	2	1	2	1753	1347	1	1	6	0	2	0	
1	88819	1	1	3	2316	634	1	3	4	1	4	0	
2	85623	1	1	1	2458	524	2	1	4	1	3	1	

In [188…

```
y_pred.to_csv("/content/predict_price_hpt_latest.csv")
```

In [ ]: