HOUSE PRICE PREDICTION USING ML

Project Description:

House is very important part of everyone's life. Buying house is dream for most of the people. Price of House depends on many parameters like Area, Bedrooms, Parking, Furnishing Status, Main road, etc. We need to predict price of house based on these parameters. We can do that using Machine Learning. We need to train Algorithm based on available Datasets after which we can predict price of house by giving based on parameters values, we give. **Linear Regression Algorithm** is used to train data and build a model from it after which we can predict price of house.

Since the machine learning supports only Numerical Data to get trained, it's necessary to convert categorical data to numerical data before moving to any steps further. We can use **Map method** to convert from categorical data to numerical data which is simple and easy to understand and execute. Initially Dataset we have may have outliers so it's necessary to handle outliers to get accurate predictions. We can find outliers using Boxplot and handle it using **Capping using IQR technique**. Another important part of this project visualization or EDA (Exploratory Data Analysis) which is used to analyse all columns individually and can get relationships between 2 or more columns to get clear picture of Data graphically. We can analyse one column using **Boxplot, distplot, Countplot, etc.,** and we can analyse two or more columns using **Scatter plot, Barchart, Boxplot, Pairplot, etc**. We can find correlation of 2 column or with all column, we can use **Heatmap** and check how much it's corelated and check for positive or negative correlation.

Pre requisites:

To complete this project, you must require following software's, concepts and packages

Anaconda navigator and PyCharm

Python packages:

- Open anaconda prompt as administrator – Go to drive (i.e., J:) – type "Jupyter Notebook"
- Type "pip install NumPy" and click enter.
- Type "pip install Pandas" and click enter.
- Type "pip install scikit-learn" (i.e., sklearn) and click enter.
- Type "pip install matplotlib" and click enter.
- Type "pip install seaborn" and click enter.

Project Objectives:

By the end of this project, you will:

- Know fundamental concepts and techniques used for machine learning.
- Gain a broad understanding about data.
- Have knowledge on pre-processing the data/transformation techniques on outlier and
- some visualization concepts.

Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

1) Data collection

- Collect the dataset or create the dataset

2) Visualizing and analysing the data

- Importing the libraries
- Read the Dataset

3) Data pre-processing

- Converting Datatype (categorical data to numerical data)
- Descriptive analysis
- Checking Correlation
- Splitting the dataset as x and y
- Splitting dataset into training and test set

4) Model building

- Import the model building libraries
- Initializing the model
- Training and testing the model
- Finding Accuracy and Loss
- Predicting with own Data

1) Data Collection

ML depends heavily on data; it is most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset or use dataset available in PC.

Download the dataset

There are many popular open sources for collecting the data.

Eg: kaggle.com, UCI repository, etc.

In this project I have used Housing.csv data which was available with me.

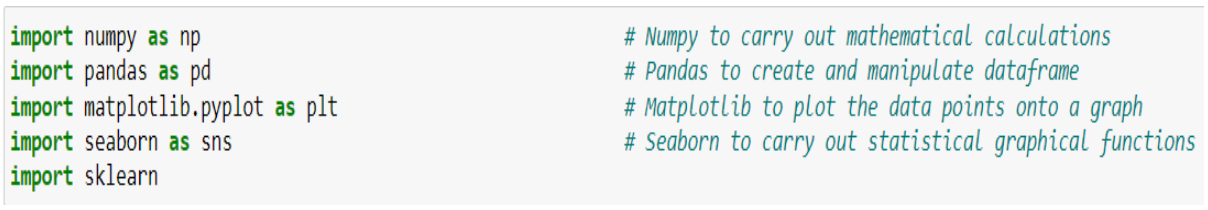
2) Visualizing and analysing the data

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analysing techniques.

(Note: There is n number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.)

Step 1: Importing the libraries

Import the necessary libraries as shown in the image.

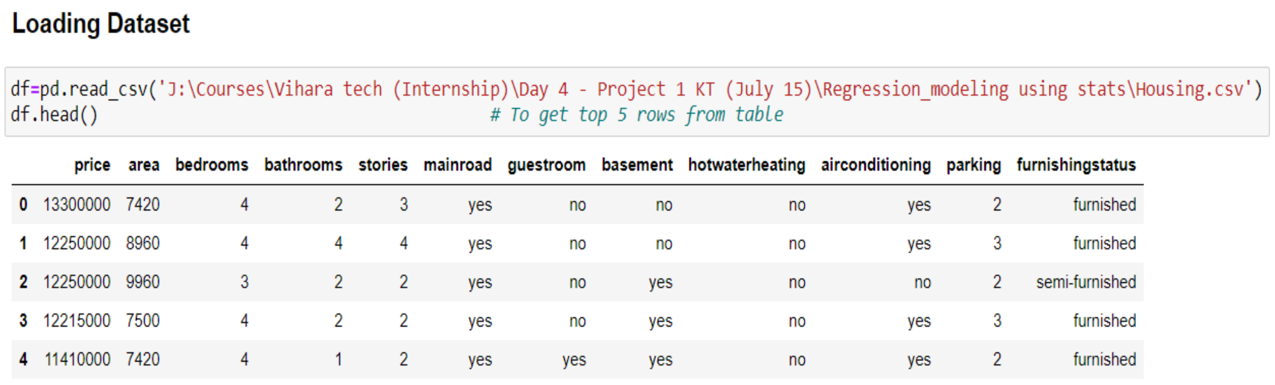


```
import numpy as np                          # Numpy to carry out mathematical calculations
import pandas as pd                         # Pandas to create and manipulate dataframe
import matplotlib.pyplot as plt             # Matplotlib to plot the data points onto a graph
import seaborn as sns                       # Seaborn to carry out statistical graphical functions
import sklearn
```

Step 2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of csv file.



Loading Dataset

```
df=pd.read_csv('J:\Courses\Vihara tech (Internship)\Day 4 - Project 1 KT (July 15)\Regression_modeling using stats\Housing.csv')
df.head()                                   # To get top 5 rows from table
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	furnished

3) Data Pre-processing

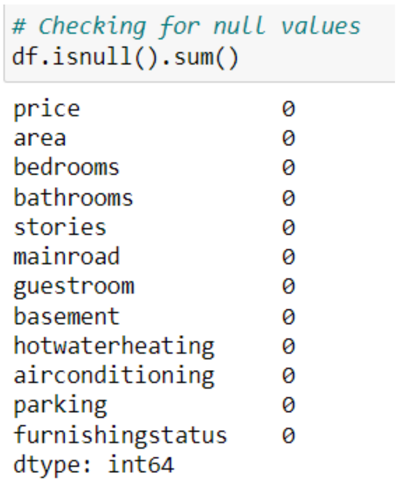
As we have understood how the data is, let’s pre-process the collected data.
The download data set is not suitable for training the machine learning model as it might have so much of randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps:

- Descriptive analysis
- Converting Datatype (categorical data to numerical data)
- Splitting the dataset as x and y
- Checking Correlation
- Splitting dataset into training and test set

(Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.)

Step 1: Checking for null values

For checking the null values, data.isnull() function is used. To sum those null values, we Use .sum() function to it. From the below image we found that there are no null values present in our dataset. So, we can skip handling of missing values step.



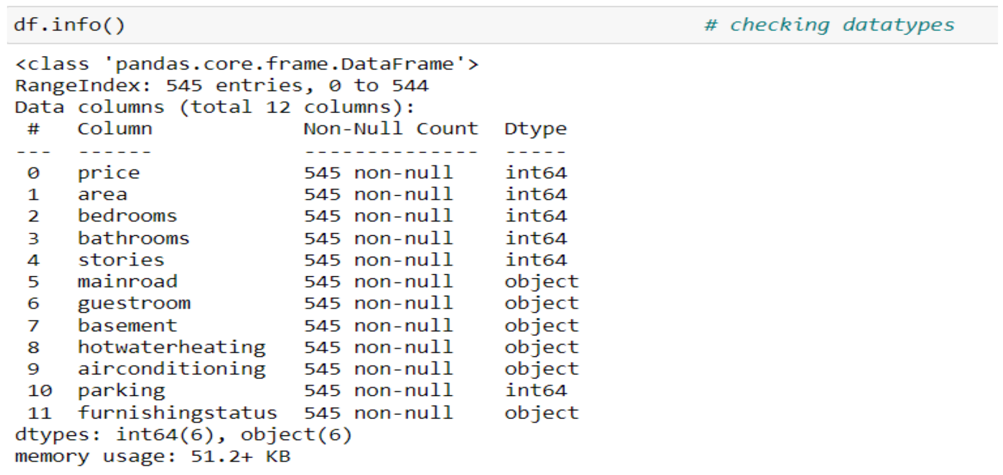
```
# Checking for null values
df.isnull().sum()
```

```
price               0
area                0
bedrooms            0
bathrooms           0
stories             0
mainroad            0
guestroom           0
basement            0
hotwaterheating     0
airconditioning     0
parking             0
furnishingstatus    0
dtype: int64
```

Step 2: Checking for Datatypes

For checking datatypes we can use .info() to get information of all columns belonging to which datatype individually.
We can use “table_name.select_dtypes(exclude = 'object')” to get all numerical data (int).

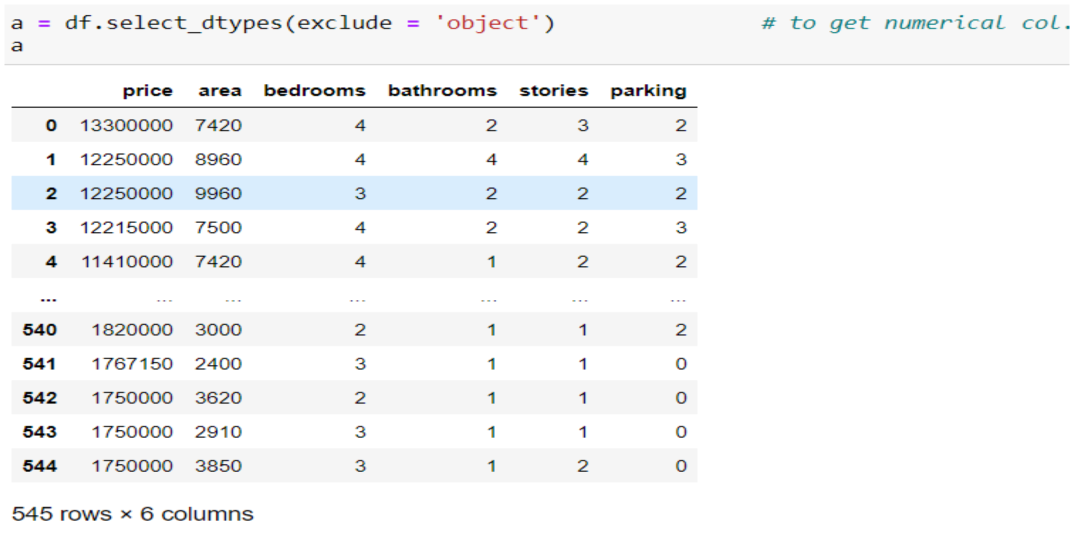
We can use “table_name.select_dtypes(include = 'object')” to get all categorical data (object).



```
df.info()                                          # checking datatypes
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   price             545 non-null    int64 
 1   area              545 non-null    int64 
 2   bedrooms          545 non-null    int64 
 3   bathrooms         545 non-null    int64 
 4   stories           545 non-null    int64 
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64 
 11  furnishingstatus  545 non-null    object
dtypes: int64(6), object(6)
memory usage: 51.2+ KB
```

Fig 2.2.1: Information about datatypes

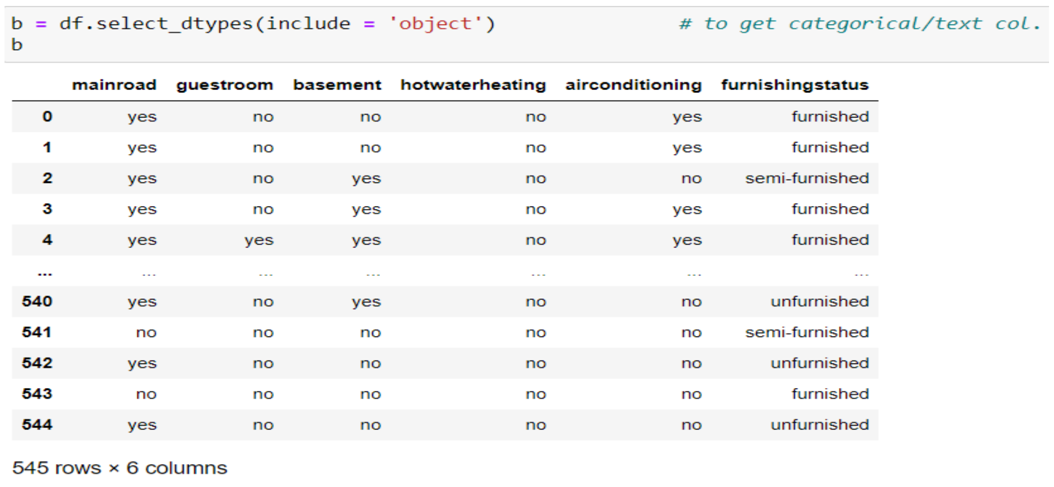


```
a = df.select_dtypes(exclude = 'object')                 # to get numerical col.
a
```

	price	area	bedrooms	bathrooms	stories	parking
0	13300000	7420	4	2	3	2
1	12250000	8960	4	4	4	3
2	12250000	9960	3	2	2	2
3	12215000	7500	4	2	2	3
4	11410000	7420	4	1	2	2
...	...	...	...	...	...	...
540	1820000	3000	2	1	1	2
541	1767150	2400	3	1	1	0
542	1750000	3620	2	1	1	0
543	1750000	2910	3	1	1	0
544	1750000	3850	3	1	2	0

545 rows × 6 columns

Fig 2.2.1: Numerical Datatypes



```
b = df.select_dtypes(include = 'object')              # to get categorical/text col.
b
```

	mainroad	guestroom	basement	hotwaterheating	airconditioning	furnishingstatus
0	yes	no	no	no	yes	furnished
1	yes	no	no	no	yes	furnished
2	yes	no	yes	no	no	semi-furnished
3	yes	no	yes	no	yes	furnished
4	yes	yes	yes	no	yes	furnished
...	...	...	...	...	...	...
540	yes	no	yes	no	no	unfurnished
541	no	no	no	no	no	semi-furnished
542	yes	no	no	no	no	unfurnished
543	no	no	no	no	no	furnished
544	yes	no	no	no	no	unfurnished

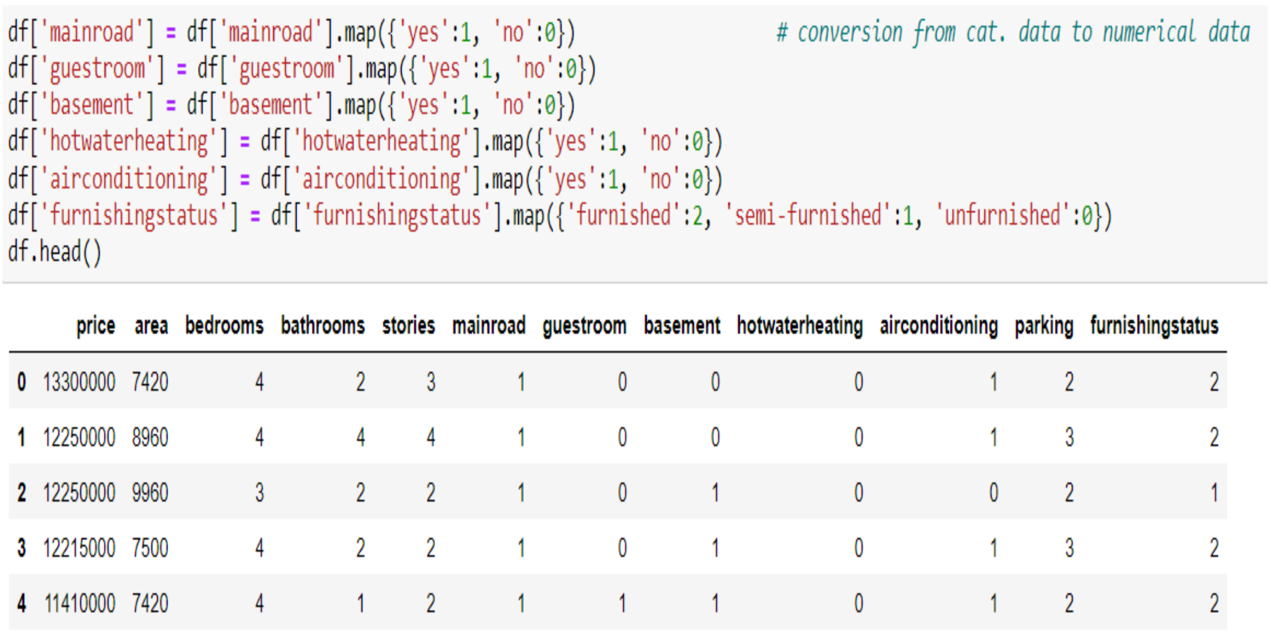
545 rows × 6 columns

Fig 2.2.1: Categorical/Text Datatypes

Step 3: Converting the Datatype

Converting the datatype from object to int. So that we will get output properly.

We need to convert all categorical data to numerical data as we can't train categorical data in ML. We have many methods to convert but I have chosen map() method as it's simpler to understand and use.

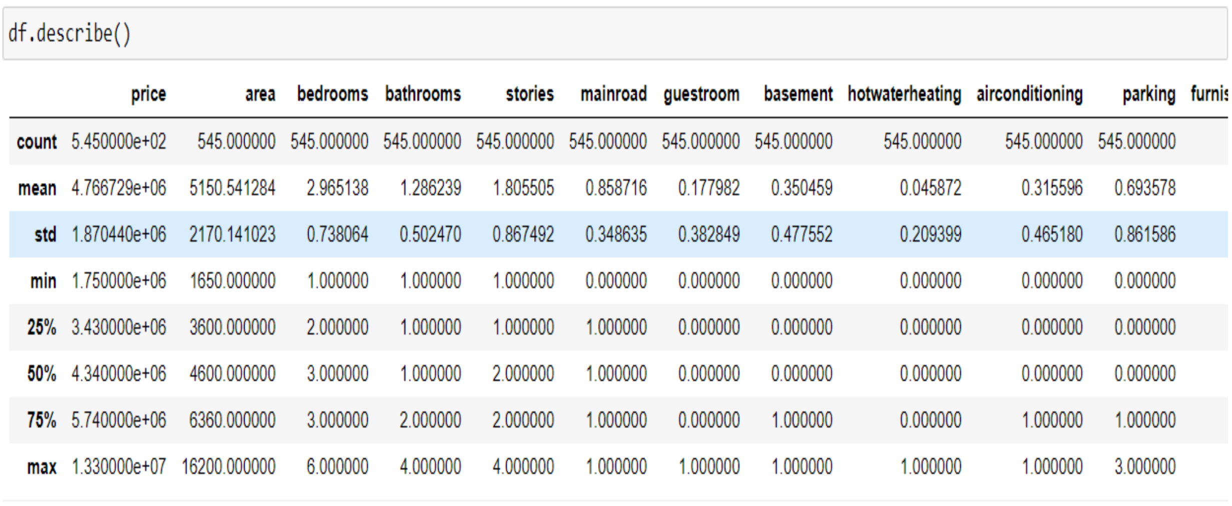


```
df['mainroad'] = df['mainroad'].map({'yes':1, 'no':0})            # conversion from cat. data to numerical data
df['guestroom'] = df['guestroom'].map({'yes':1, 'no':0})
df['basement'] = df['basement'].map({'yes':1, 'no':0})
df['hotwaterheating'] = df['hotwaterheating'].map({'yes':1, 'no':0})
df['airconditioning'] = df['airconditioning'].map({'yes':1, 'no':0})
df['furnishingstatus'] = df['furnishingstatus'].map({'furnished':2, 'semi-furnished':1, 'unfurnished':0})
df.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	furnishingstatus
0	13300000	7420	4	2	3	1	0	0	0	1	2	2
1	12250000	8960	4	4	4	1	0	0	0	1	3	2
2	12250000	9960	3	2	2	1	0	1	0	0	2	1
3	12215000	7500	4	2	2	1	0	1	0	1	3	2
4	11410000	7420	4	1	2	1	1	1	0	1	2	2

Step 4: Descriptive analysis

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas have a worthy function called describe. With this describe function we can find mean, std, min, max and percentile values of continuous features.



```
df.describe()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	furnis
count	5.450000e+02	545.000000	545.000000	545.000000	545.000000	545.000000	545.000000	545.000000	545.000000	545.000000	545.000000	
mean	4.766729e+06	5150.541284	2.965138	1.286239	1.805505	0.858716	0.177982	0.350459	0.045872	0.315596	0.693578	
std	1.870440e+06	2170.141023	0.738064	0.502470	0.867492	0.348635	0.382849	0.477552	0.209399	0.465180	0.861586	
min	1.750000e+06	1650.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	3.430000e+06	3600.000000	2.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	4.340000e+06	4600.000000	3.000000	1.000000	2.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
75%	5.740000e+06	6360.000000	3.000000	2.000000	2.000000	1.000000	0.000000	1.000000	0.000000	1.000000	1.000000	
max	1.330000e+07	16200.000000	6.000000	4.000000	4.000000	1.000000	1.000000	1.000000	1.000000	1.000000	3.000000	

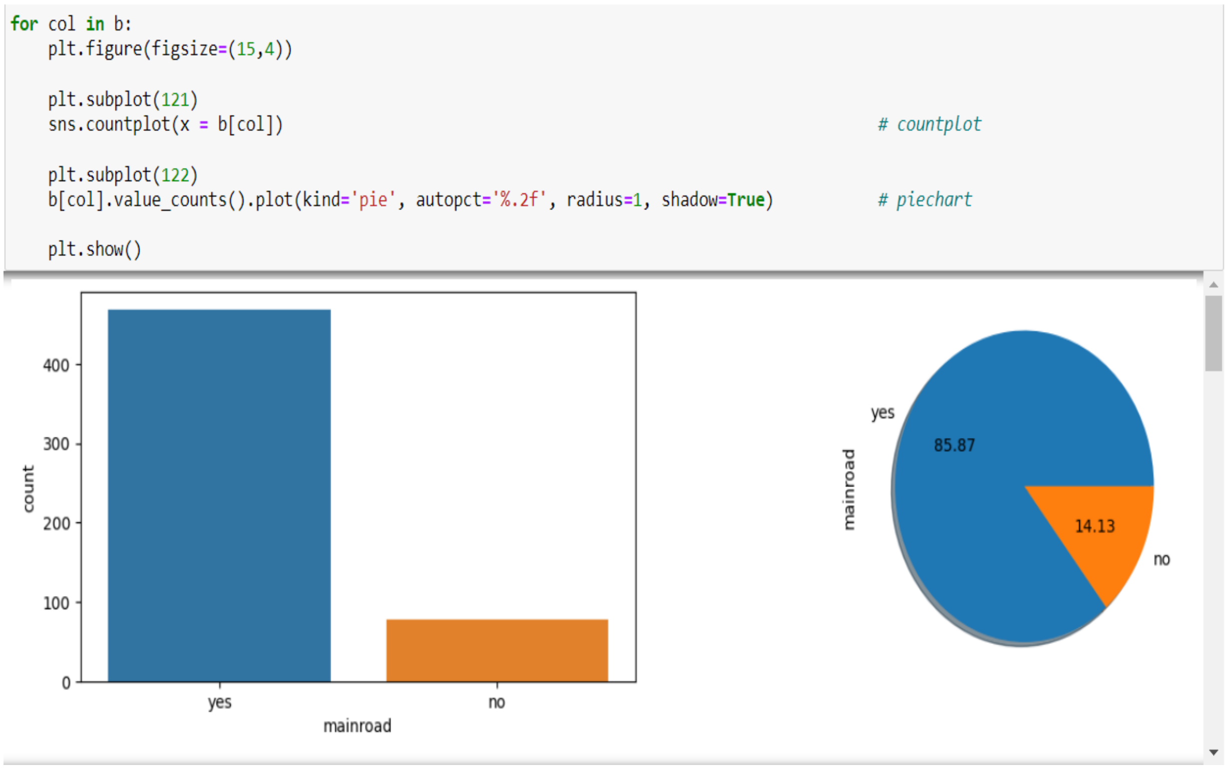
Step 5: EDA (Exploratory Data Analysis)

a) Univariate Analysis

- Univariate analysis is the simplest form of analysing data. "Uni" means "one", so in other words our data has only one variable. It doesn't deal with causes or relationships and its major purpose is to describe. It takes data, summarizes that data and finds patterns in the data.
- There are 2 types of Data, Categorical Data and Numerical Data. For Categorical Data, Countplot and Piechart are best suit. For Numerical Data Boxplot and Distplot are best suit which gives proper distribution of Data

(i) Categorical Data

- Taking 2 plots for this analysis namely Countplot and Piechart.
- Instead of creating plots for each column, we can use for loop which will create all 2 plots for all categorical columns.
- We can use for loop for variable 'b' where all categorical columns are stored, instead of writing it many times for each column we can use for loop which gives the same result, quicker result and less coding. In below fig. I have pasted just one countplot and piechart as an example but it's there for all categorical column.



```
for col in b:
    plt.figure(figsize=(15,4))

    plt.subplot(121)
    sns.countplot(x = b[col])                                               # countplot

    plt.subplot(122)
    b[col].value_counts().plot(kind='pie', autopct='%.2f', radius=1, shadow=True)          # piechart

    plt.show()
```

Fig: Countplot and Piechart

Column 1: Countplot

Col1 shows the Countplot which is used to plot for categorical data in general. We can see differences in categorical data in dataset. We can know how many data has mainroad, how many doesn't like 400+ have mainroad and around 80 doesn't have mainroad.

Similarly, can be checked for other categorical columns as well as shown in above plots.

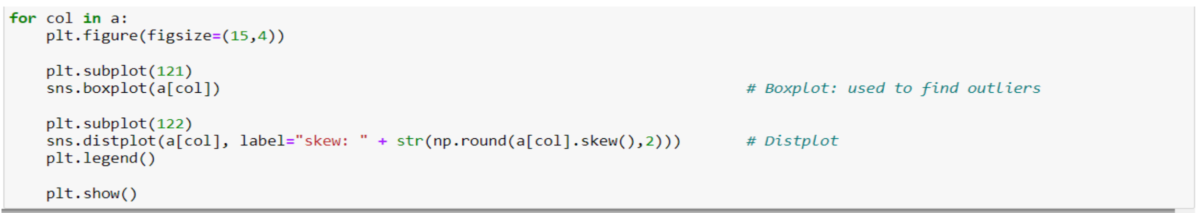
Column 2: Piechart

Col2 shows the piechart which gives precise values in percentage. Like 85.87% data have mainroads and 14.13% data doesn't have mainroad. Similarily, can be checked for all the columns as shown in above plots.

We can observe that form above Countplot and Piechart data, most of the ppl prefer mainroad and doesn't prefer guestroom, basement, hotwaterheating, airconditioning and most of the ppl prefer semi-furnished followed by unfurnished and furnished.

(ii) Numerical Data

- Taking 2 plots for this analysis namely Boxplot and Distplot
- Instead of creating plots for each column, we can use for loop which will create all 2 plots for all numerical columns.
- We can use for loop for variable 'a' where all categorical columns are stored, instead of writing it many times for each column we can use for loop which gives the same result, quicker result and less coding. In below fig. I have pasted just one boxplot and distplot as an example but it's there for all categorical column.



```
for col in a:
    plt.figure(figsize=(15,4))

    plt.subplot(121)
    sns.boxplot(a[col])                                                   # Boxplot: used to find outliers

    plt.subplot(122)
    sns.distplot(a[col], label="skew: " + str(np.round(a[col].skew(),2)))     # Distplot
    plt.legend()

    plt.show()
```

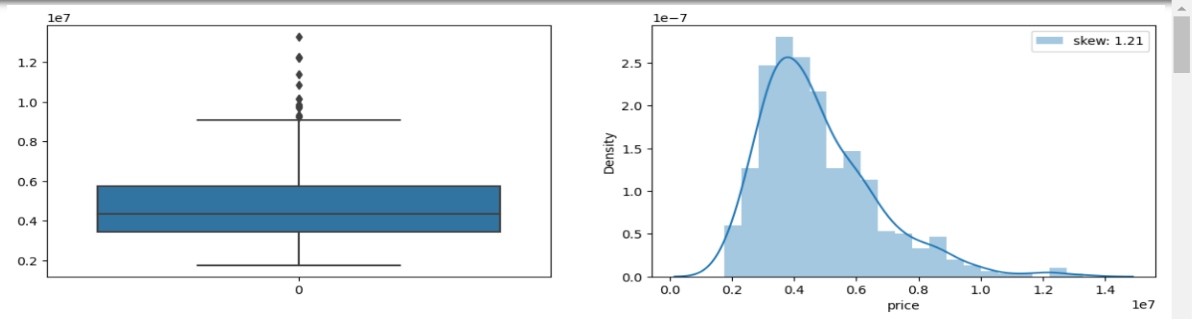


Fig: Boxplot and distplot

Column 1: Boxplot

We can see from above that outliers of all col. are there are hardly 1 outlier except price and area so we can ignore it. Since, there are many outliers present in Price and area we need to handle handle it to increase its accuracy and effeciency.

Column 2: Distplot

Other side of col, we can see distplot which is advanced version of Histogram which gives KDE (Kernell Density Estimator) along with graph which estimate the probability density function of a random variable.

(b) Bivariate Analysis

- Bivariate analysis is a statistical method examining how two different things are related.
- The bivariate analysis aims to determine if there is a statistical link between the two variables. and, if so, how strong and in which direction that link is.

(i) Scatterplot (Num-Num)

- A scatter plot uses dots to represent values for two different numeric variables.
- This plot are majorly suitable for numerical data.

```
# Scatterplot (Num-Num)

plt.figure(figsize=(12,7))
plt.subplot(221)
sns.scatterplot(x = df["area"], y = df["price"])

plt.subplot(222)
sns.scatterplot(x = df["bedrooms"], y = df["price"])

plt.subplot(223)
sns.scatterplot(x = df["stories"], y = df["price"])

plt.subplot(224)
sns.scatterplot(x = df["parking"], y = df["price"])

plt.show()
```

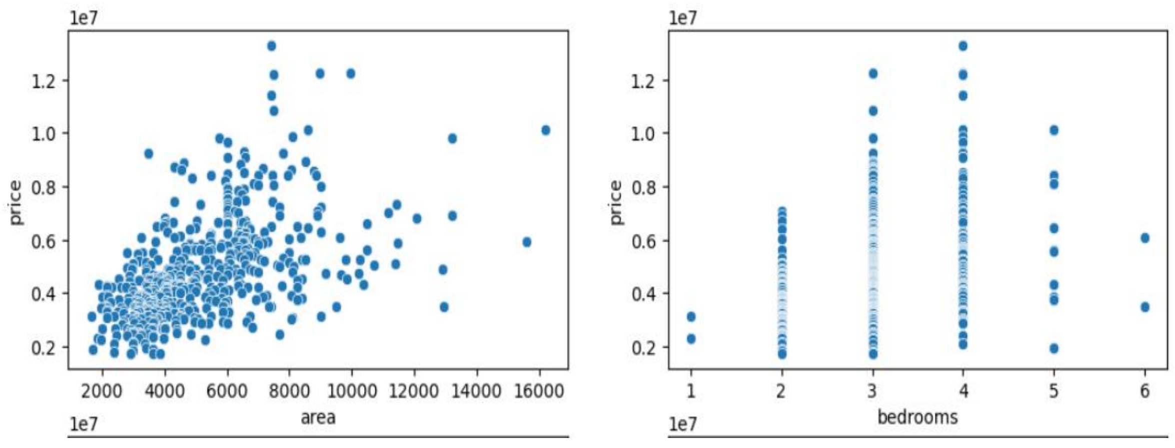
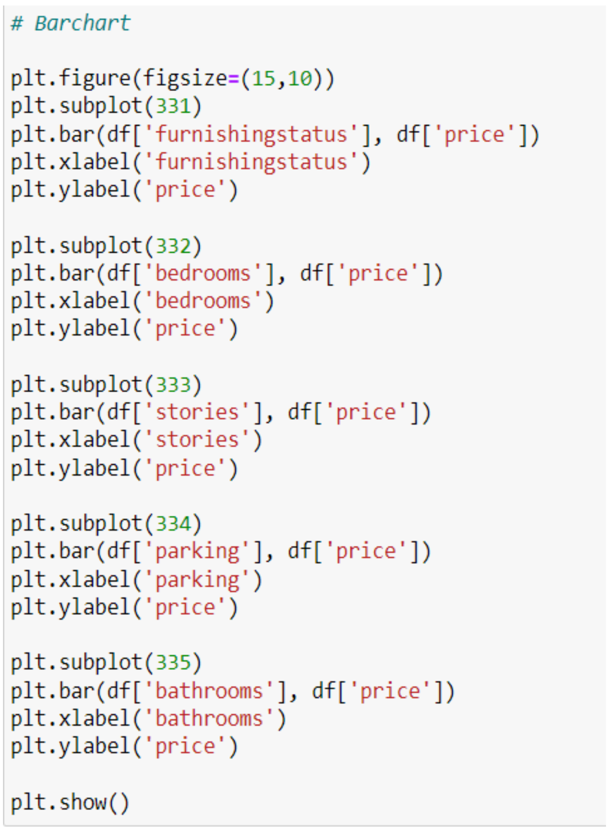


Fig: Scatterplot

(ii) **Barchart**

- Purpose of a bar graph is to convey relational information quickly in a visual manner.
- We can see relationship between price and other variables from below fig (in graphs)



```
# Barchart

plt.figure(figsize=(15,10))
plt.subplot(331)
plt.bar(df['furnishingstatus'], df['price'])
plt.xlabel('furnishingstatus')
plt.ylabel('price')

plt.subplot(332)
plt.bar(df['bedrooms'], df['price'])
plt.xlabel('bedrooms')
plt.ylabel('price')

plt.subplot(333)
plt.bar(df['stories'], df['price'])
plt.xlabel('stories')
plt.ylabel('price')

plt.subplot(334)
plt.bar(df['parking'], df['price'])
plt.xlabel('parking')
plt.ylabel('price')

plt.subplot(335)
plt.bar(df['bathrooms'], df['price'])
plt.xlabel('bathrooms')
plt.ylabel('price')

plt.show()
```

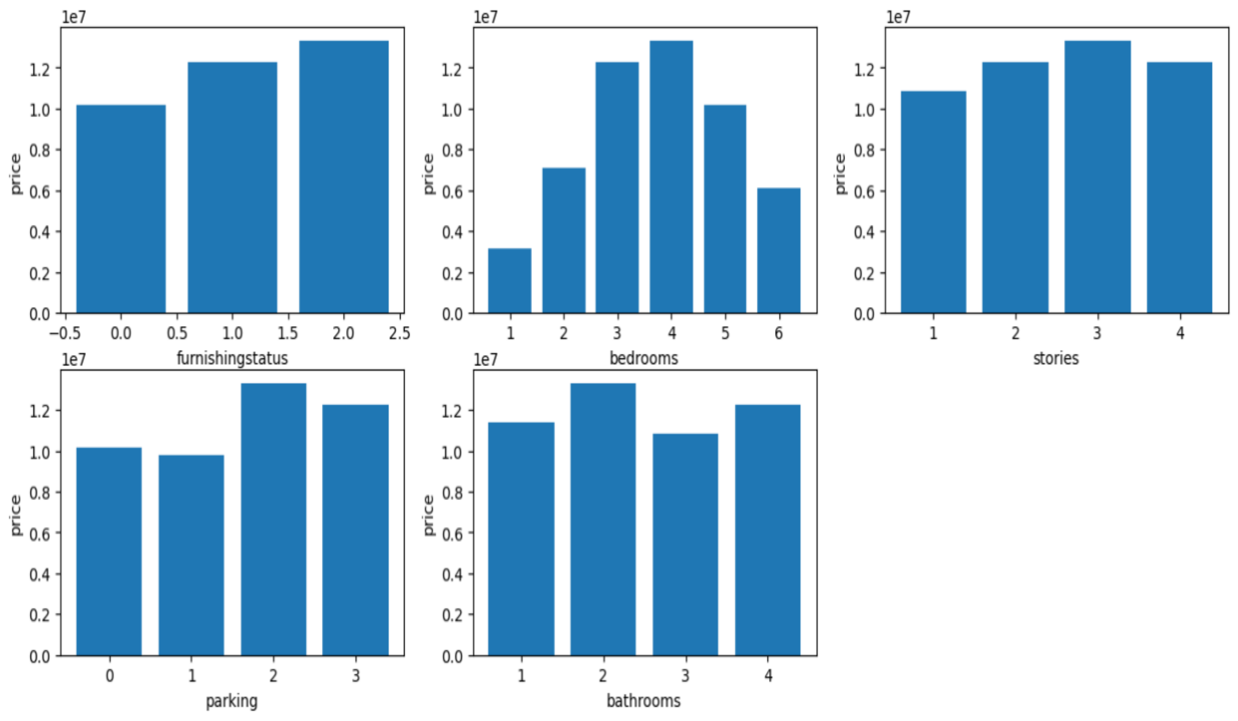
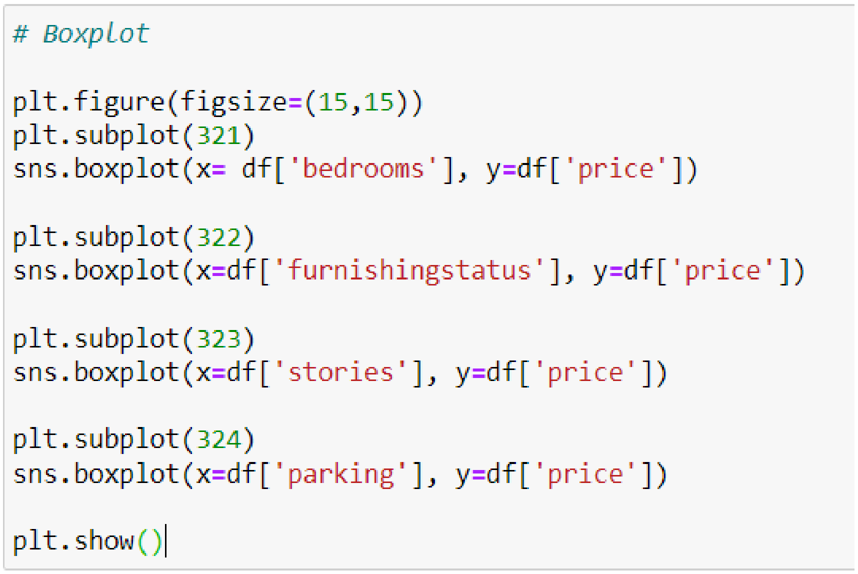


Fig: Barchart

(iii) Boxplot



```
# Boxplot

plt.figure(figsize=(15,15))
plt.subplot(321)
sns.boxplot(x= df['bedrooms'], y=df['price'])

plt.subplot(322)
sns.boxplot(x=df['furnishingstatus'], y=df['price'])

plt.subplot(323)
sns.boxplot(x=df['stories'], y=df['price'])

plt.subplot(324)
sns.boxplot(x=df['parking'], y=df['price'])

plt.show()
```

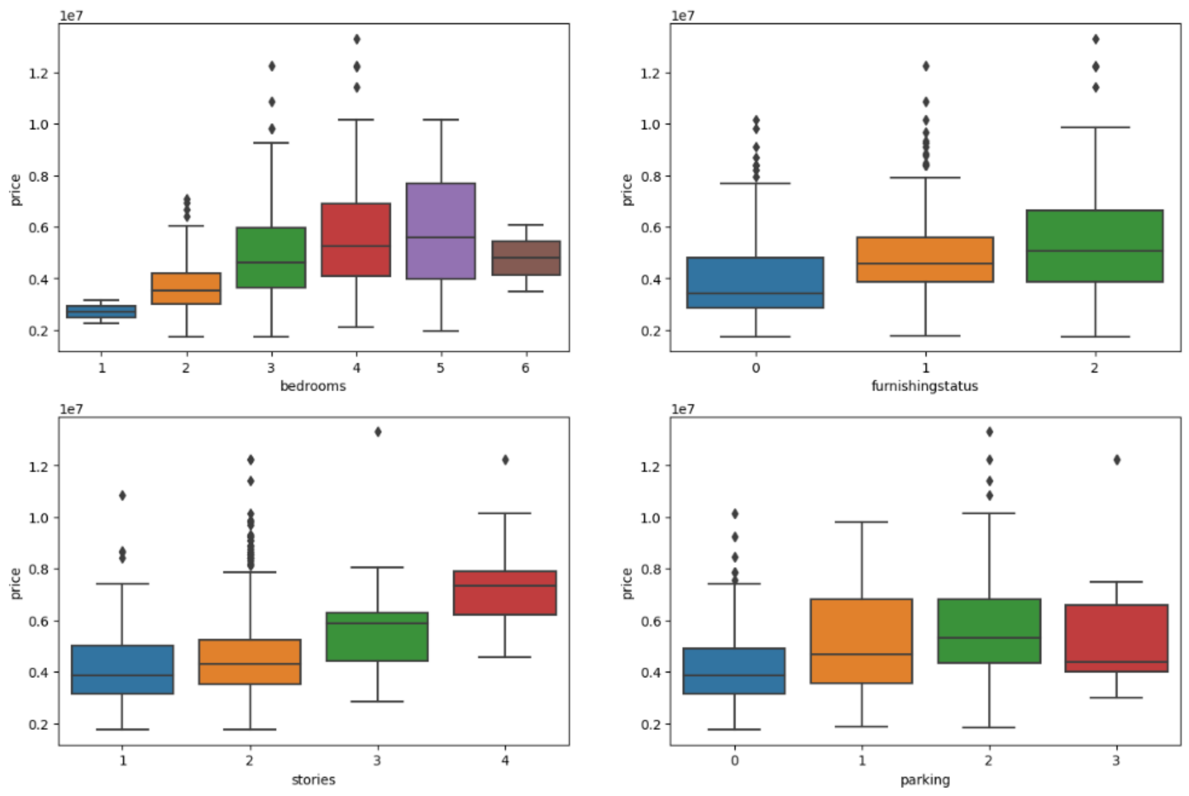
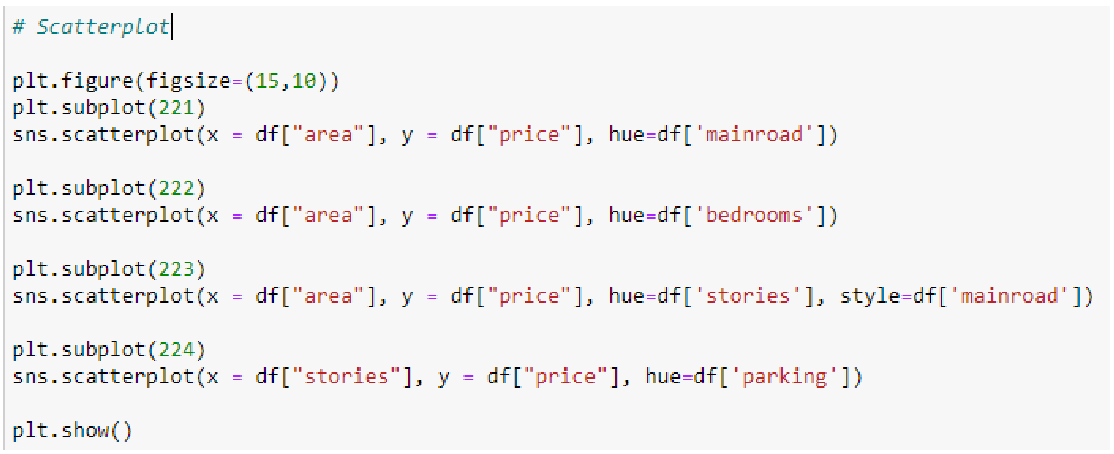


Fig: Boxplot

(c) Multivariate Analysis

- Multivariate analysis involves evaluating multiple variables (more than two) to identify any possible association among them.

(i) <u>Scatterplot</u>



```
# Scatterplot

plt.figure(figsize=(15,10))
plt.subplot(221)
sns.scatterplot(x = df["area"], y = df["price"], hue=df['mainroad'])

plt.subplot(222)
sns.scatterplot(x = df["area"], y = df["price"], hue=df['bedrooms'])

plt.subplot(223)
sns.scatterplot(x = df["area"], y = df["price"], hue=df['stories'], style=df['mainroad'])

plt.subplot(224)
sns.scatterplot(x = df["stories"], y = df["price"], hue=df['parking'])

plt.show()
```

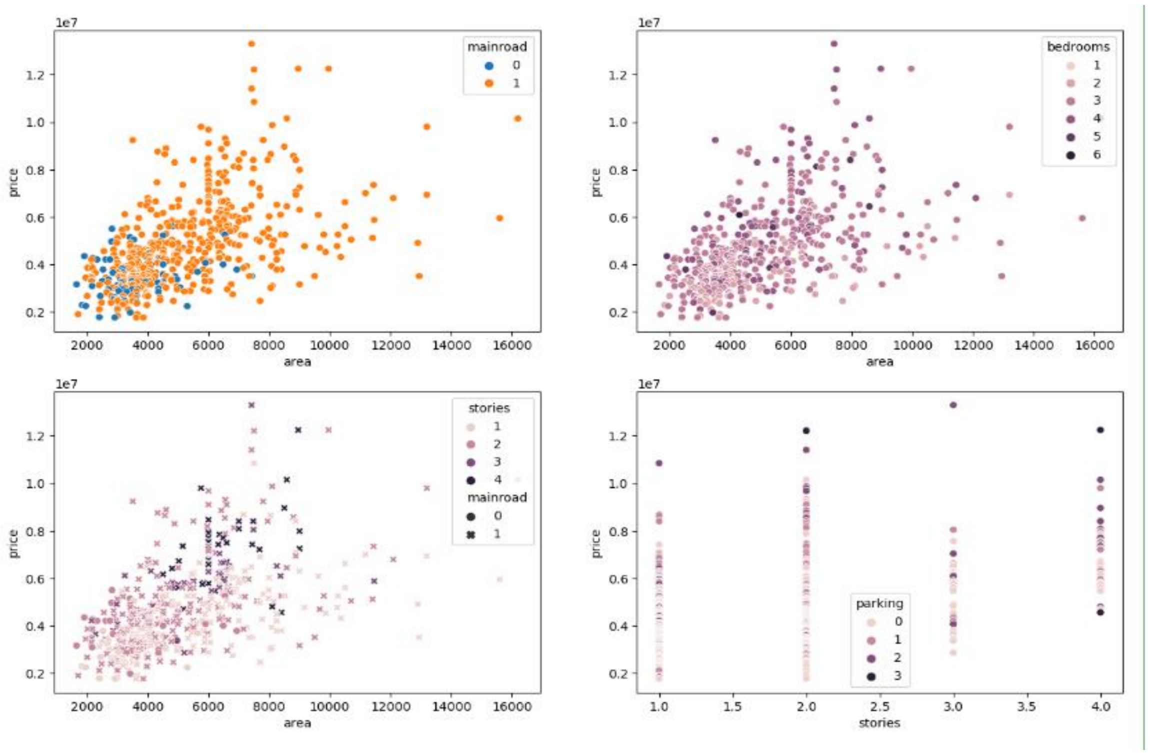


Fig: Scatterplot

- Using scatter plot we can see relationship between more than 3 variables,
- While comparing price and area, we can see how many have mainroad which is in orange color and how many doesn't in blue.
- Similarly, we can see for bedrooms, area and stories which us indicated in different colors.

(ii) **Boxplot**

```
# Boxplot

plt.figure(figsize=(15,15))
plt.subplot(321)
sns.boxplot(x= df['bedrooms'], y=df['price'], hue=df['bathrooms'])

plt.subplot(322)
sns.boxplot(x=df['bedrooms'], y=df['price'], hue=df['furnishingstatus'])

plt.subplot(323)
sns.boxplot(x=df['stories'], y=df['price'], hue=df['parking'])

plt.subplot(324)
sns.boxplot(x=df['stories'], y=df['price'], hue=df['mainroad'])

plt.subplot(325)
sns.boxplot(x=df['bedrooms'], y=df['price'], hue=df['mainroad'])

plt.subplot(326)
sns.boxplot(x=df['bedrooms'], y=df['price'], hue=df['stories'])

plt.show()
```

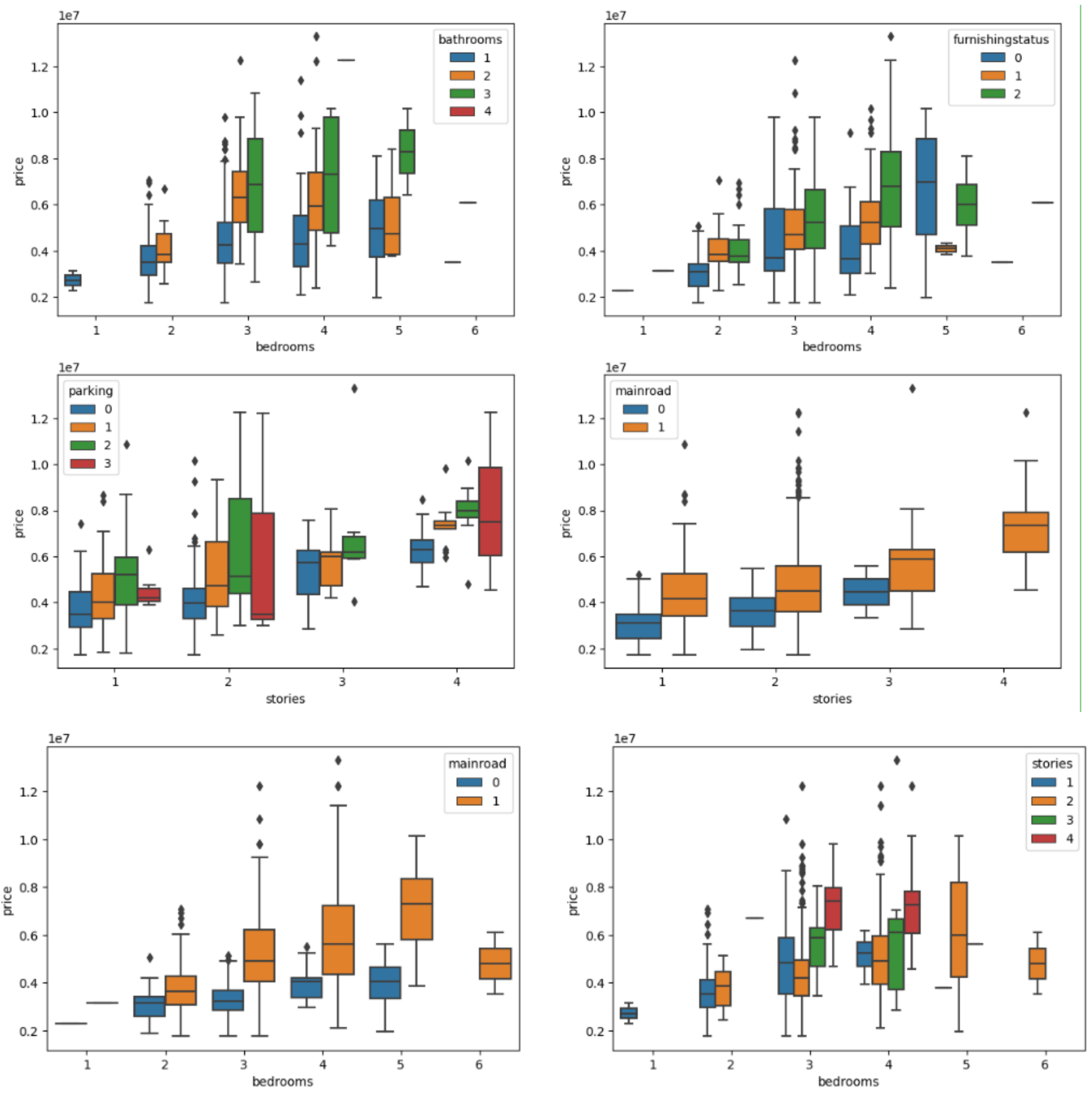


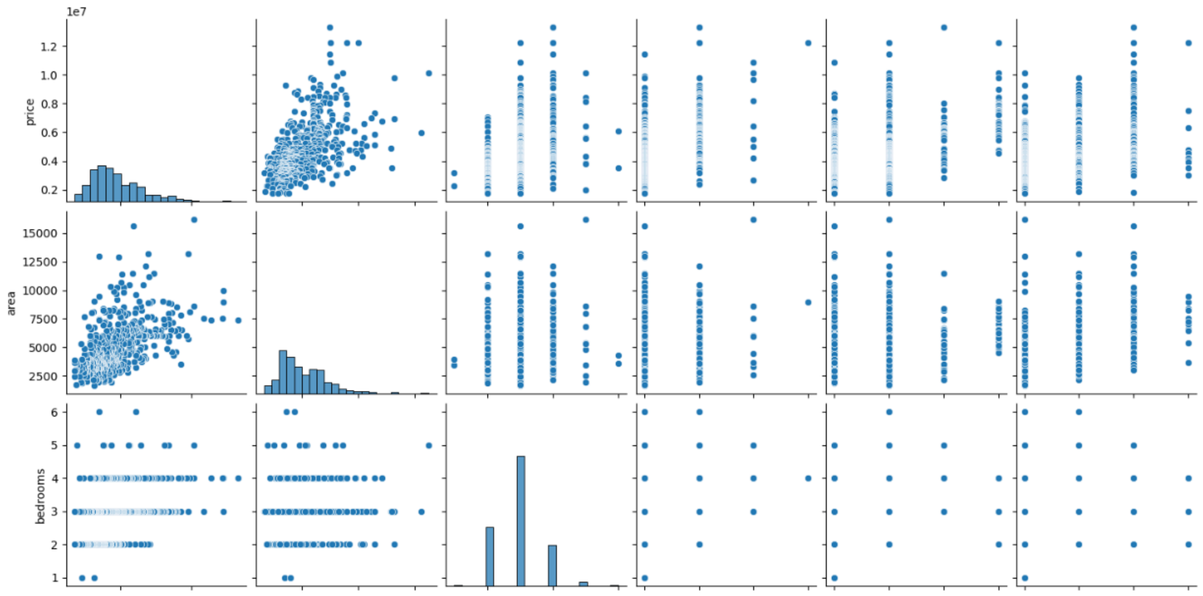
Fig: Boxplot

- We can observe following observations from Boxplot for more than 2 variables,
- For 3 bathrooms 4BHK price is more, for 2 bathrooms 3 & 4BHK prices are more and for 1-bathroom 5BHK price is more.
- If house is not furnished then 5BHK price is high, if semi-furnished then 4BHK and if furnished then 4BHK price is high.
- Price is high for 4stories 3parking followed by 4stories 2parking, 4stories 1parking and 4stories no parking.
- Price depends on stories and mainroad, more the stories high the price and mainroad is included in it.
- Price depends on bedrooms and mainroad, if there is mainroad then price is high depending on stories.
- Price is high if stories and bedrooms are more.

(iii) <u>Pairplot</u>

Pairplot considers only numerical columns from the table and do scatter plot with same col, it will do histogram and relation with another col will be scatter plot.

```
# Pairplot: takes only numerical columns from the table and do scatter plot
# with same col, it will do histogram and relation with other col will be scatter plot

sns.pairplot(df)                    # Visualize each numerical columns (including conversion from cat to num)
```



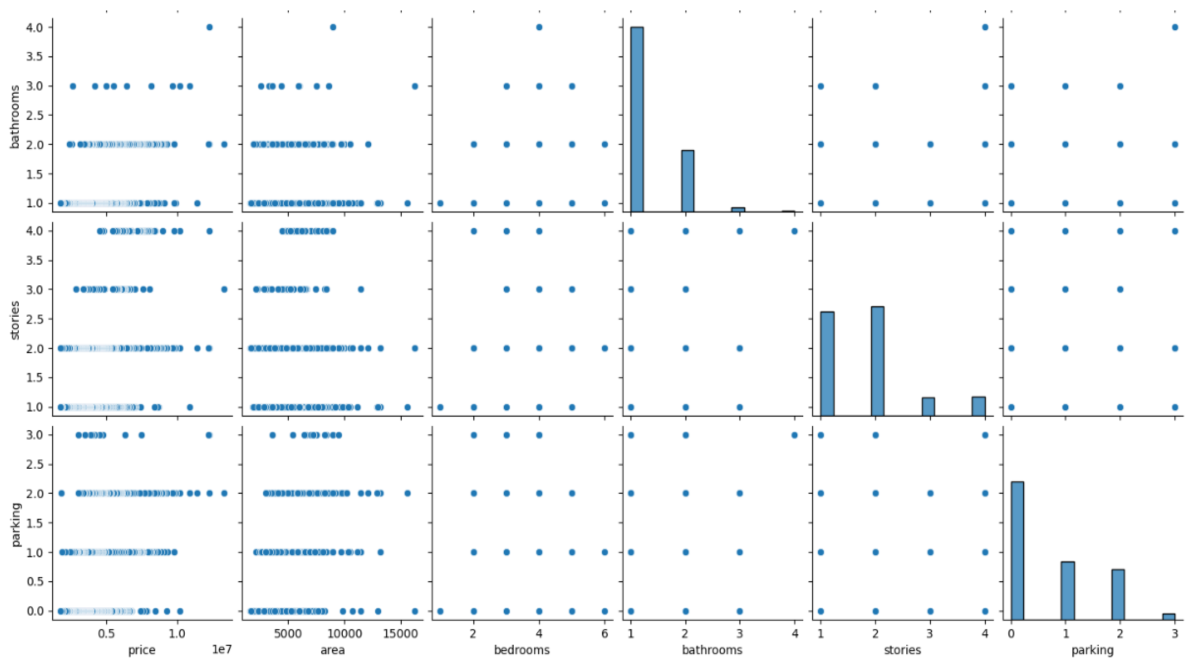
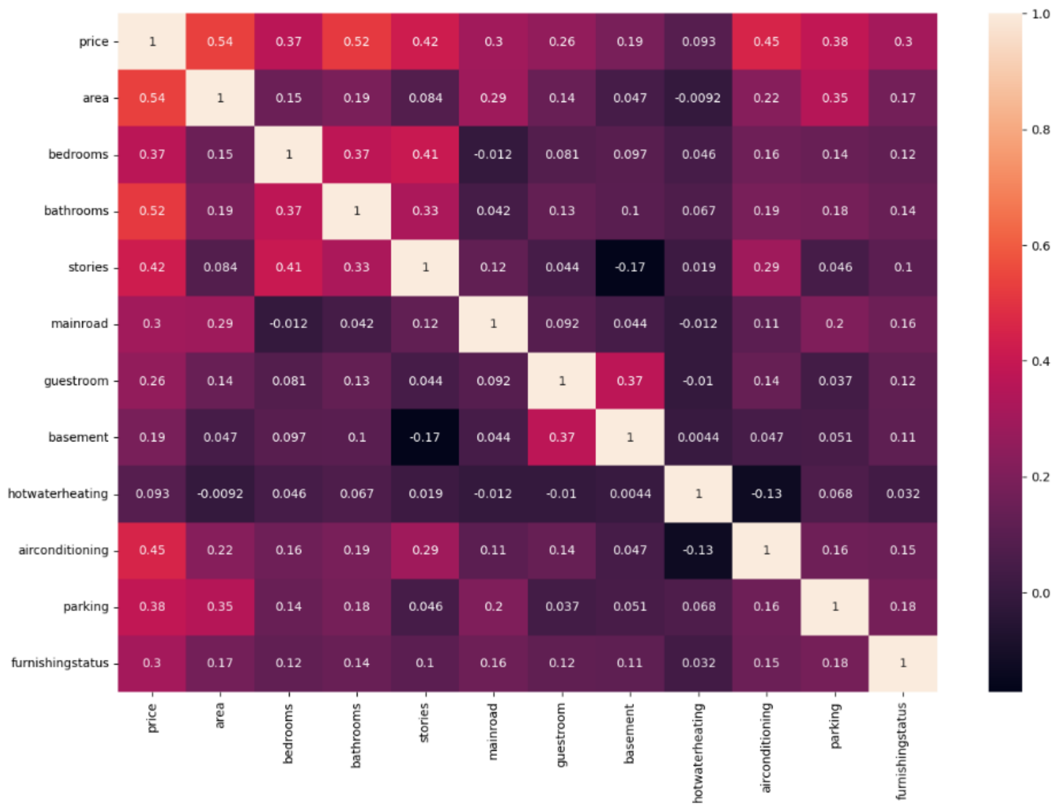


Fig: Pairplot

(iv) **Checking Correlation (Heatmap)**

Heatmap is used to find correlation between the variables of all columns which makes easier to check corelation between columns like how much strong or week, positive or negative correlated.



	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	furnishingstatus
price	1	0.54	0.37	0.52	0.42	0.3	0.26	0.19	0.093	0.45	0.38	0.3
area	0.54	1	0.15	0.19	0.084	0.29	0.14	0.047	-0.0092	0.22	0.35	0.17
bedrooms	0.37	0.15	1	0.37	0.41	-0.012	0.081	0.097	0.046	0.16	0.14	0.12
bathrooms	0.52	0.19	0.37	1	0.33	0.042	0.13	0.1	0.067	0.19	0.18	0.14
stories	0.42	0.084	0.41	0.33	1	0.12	0.044	-0.17	0.019	0.29	0.046	0.1
mainroad	0.3	0.29	-0.012	0.042	0.12	1	0.092	0.044	-0.012	0.11	0.2	0.16
guestroom	0.26	0.14	0.081	0.13	0.044	0.092	1	0.37	-0.01	0.14	0.037	0.12
basement	0.19	0.047	0.097	0.1	-0.17	0.044	0.37	1	0.0044	0.047	0.051	0.11
hotwaterheating	0.093	-0.0092	0.046	0.067	0.019	-0.012	-0.01	0.0044	1	-0.13	0.068	0.032
airconditioning	0.45	0.22	0.16	0.19	0.29	0.11	0.14	0.047	-0.13	1	0.16	0.15
parking	0.38	0.35	0.14	0.18	0.046	0.2	0.037	0.051	0.068	0.16	1	0.18
furnishingstatus	0.3	0.17	0.12	0.14	0.1	0.16	0.12	0.11	0.032	0.15	0.18	1

Fig: Heatmap

We can see from above heatmap, no feature/col has highly positive or negative correlated so all col. is equally important.

Step 6: Splitting the data (Without Handling Outliers)

- Now let's split the Dataset into train and test sets.
- Changes: first split the dataset into x and y and then split the data set into train and test.
- Here x and y variables are created. On x variable, data is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing x, y, test_size and random_state. We can save independent columns in x variable and dependent column in y variable.

```
# Split the data

# Give independent data to x & dependent data to y
x = df.drop(['price'],axis=1)                # independent
y = df['price']                              # dependent
```

```
# Splitting the data using train_test method
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
# test_size indicates percentage of values for test i.e, 20% Data = Test & 80% Data = Train
```

➢ **We can check no. of rows selected for test and train using len()**

```
len(x_train) , len(y_train)                     # 80% Random Data selected for train
```

```
(436, 436)
```

```
len(x_test) , len(y_test)                       # 20% Random Data selected for test
```

```
(109, 109)
```

4) Model Building

Step 1: Train the model

To train the model we need to give test data and train data to the algorithm. Since I am using Linear Regression algorithm, I am importing it which is in sklearn then fitting it to get trained.

```
# Give data to the model

# Now, giving training data to train the algorithm
from sklearn.linear_model import LinearRegression
```

```
reg = LinearRegression()
```

```
reg.fit(x_train , y_train)                    # fit is mediator, gives train data to reg (algorithm: y=mx+c)
```

```
LinearRegression()
```

Step 2: Check Accuracy and Loss of the Model

Since, model is trained we need to test the model so as to find accuracy and loss of model trained.

(i) Train Accuracy and Train Loss

```
# Train accuracy and Train loss

y_train_pred = reg.predict(X_train)
```

```
from sklearn.metrics import r2_score            #r2_score = used to find accuracy of the model
```

```
# find accuracy of train_data
print(f'train accuracy : {r2_score(y_train,y_train_pred)}')
```

```
train accuracy : 0.6652807320087133
```

```
# find loss of train_data
# Formula: loss = 1 - accuracy
print(f'train loss : {1-r2_score(y_train,y_train_pred)}')
```

```
train loss : 0.3347192679912867
```

(ii) Test Accuracy and Test Loss

```
# Test accuracy and Test loss

y_test_pred = reg.predict(X_test)
```

```
# find accuracy of test_data
print(f'test accuracy : {r2_score(y_test,y_test_pred)}')
```

```
test accuracy : 0.6334513682813152
```

```
# find loss of test_data
print(f'test loss : {1-r2_score(y_test,y_test_pred)}')
```

```
test loss : 0.3665486317186848
```

Step 3: Prediction with own data

```
# Prediction with own data

X_train.columns
```

```
Index(['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'guestroom',
       'basement', 'hotwaterheating', 'airconditioning', 'parking',
       'furnishingstatus'],
      dtype='object')
```

```
print('The price is',reg.predict([[7000 , 5 , 3 , 1 , 1, 1, 0, 0, 1, 1, 1]]))
```

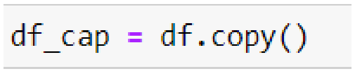
```
The price is [7620635.02407611]
```

Caping using IQR method - Handling Outliers

- We can see from Boxplot that there are outliers in many columns but only in Price and area col there are more outliers. So, we need to handle outliers so as to increase accuracy of the model. Caping using IQR technique is used to handle the outliers.
- Since, there are many outliers in Price and area column, we can handle only these columns. Others columns have 1 or 2 outliers which can be ignored.

Step1: Copy Table

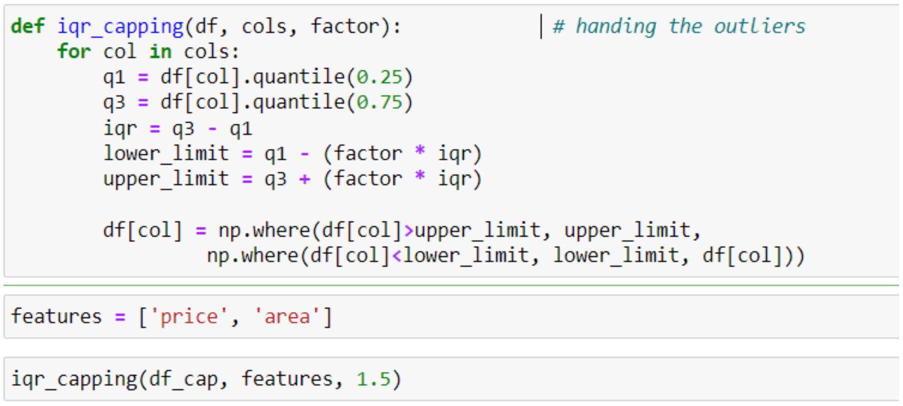
Firstly, we need to copy df (table) instead of disturbing the original table.



```
df_cap = df.copy()
```

Step 2: Handling Outliers

Now, I have used function and for loop so it can be applied for all columns that is selected to handle outliers at once instead of doing it separately. By doing so, it reduces time, increases speed and avoids confusions further.



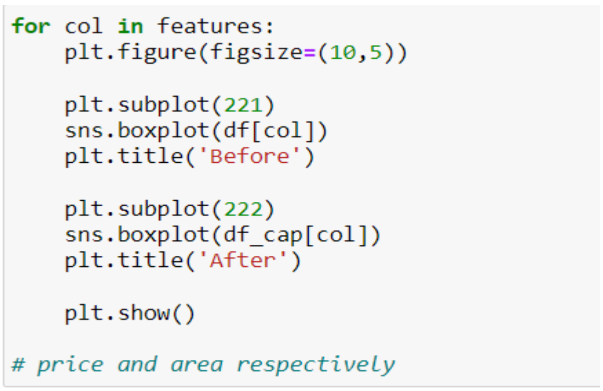
```
def iqr_capping(df, cols, factor):             # handing the outliers
    for col in cols:
        q1 = df[col].quantile(0.25)
        q3 = df[col].quantile(0.75)
        iqr = q3 - q1
        lower_limit = q1 - (factor * iqr)
        upper_limit = q3 + (factor * iqr)

        df[col] = np.where(df[col]>upper_limit, upper_limit,
                  np.where(df[col]<lower_limit, lower_limit, df[col]))
```

```
features = ['price', 'area']
```

```
iqr_capping(df_cap, features, 1.5)
```

Step3: Check for outliers and compare with outliers using Boxplot

I am plotting Boxplot now to see whether outliers are handled and also compare it without and with handling outliers so as to get clear picture of it.



```
for col in features:
    plt.figure(figsize=(10,5))

    plt.subplot(221)
    sns.boxplot(df[col])
    plt.title('Before')

    plt.subplot(222)
    sns.boxplot(df_cap[col])
    plt.title('After')

    plt.show()

# price and area respectively
```

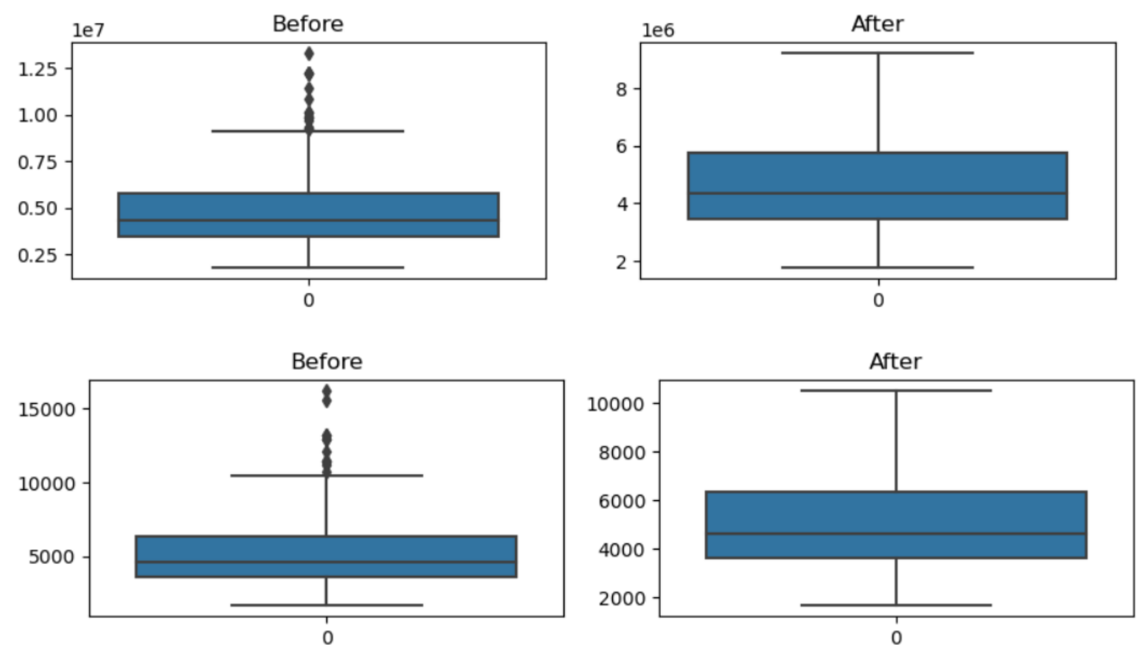


Fig: Boxplot

Step 4: Split the data (After Handling Outliers)

```python
# Give independent data to x & dependent data to y
x = df_cap.drop(['price'],axis=1)                # independent
y = df_cap['price']                              # dependent
```

```python
# Splitting the data using train_test method
from sklearn.model_selection import train_test_split
```

```python
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
# test_size indicates percentage of values for test i.e, 20% Data = Test & 80% Data = Train
```

```python
len(X_train) , len(y_train)                    # 80% Random Data selected for train
```

```
(436, 436)
```

```python
len(X_test) , len(y_test)                      # 20% Random Data selected for test
```

```
(109, 109)
```

Step 4: Train the data

```python
from sklearn.linear_model import LinearRegression
```

```python
reg = LinearRegression()
```

```python
reg.fit(X_train , y_train)
```

```
▾ LinearRegression
LinearRegression()
```

Step5: Check Slope and Intersect values

```
# m values                          # slope
reg.coef_
```

```
array([2.73461262e+02, 1.02972508e+05, 9.44416088e+05, 3.99332730e+05,
       4.91397509e+05, 2.71268056e+05, 4.63930793e+05, 6.92725194e+05,
       7.43934439e+05, 1.76134984e+05, 2.02430873e+05])
```

```
# c value                           # intersect
reg.intercept_
```

```
-137720.08604855184
```

Step6: Find Accuracy and Loss of the model

Train accuracy and Train loss

```
y_train_pred = reg.predict(X_train)
```

```
from sklearn.metrics import r2_score      #r2_score = used to find accuracy of the model
```

```
# find accuracy of train_data
print(f'train accuracy : {r2_score(y_train,y_train_pred)}')
```

```
train accuracy : 0.6703265786548698
```

```
# find loss of train_data
# Formula: loss = 1 - accuracy
print(f'train loss : {1-r2_score(y_train,y_train_pred)}')
```

```
train loss : 0.3296734213451302
```

Test accuracy and Test loss

```
y_test_pred = reg.predict(X_test)
```

```
# find accuracy of test_data
print(f'test accuracy : {r2_score(y_test,y_test_pred)}')
```

```
test accuracy : 0.668128995980866
```

```
# find loss of test_data
print(f'test loss : {1-r2_score(y_test,y_test_pred)}')
```

```
test loss : 0.33187100401913405
```

Step 7: Prediction with own data

```
X_train.columns
```

```
Index(['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'guestroom',
       'basement', 'hotwaterheating', 'airconditioning', 'parking',
       'furnishingstatus'],
      dtype='object')
```

```
print('The price is',reg.predict([[7000 , 5 , 3 , 1 , 1, 1, 0, 0, 1, 1, 1]]))
```

```
The price is [7409118.14925463]
```

- We can observe after and before handling outliers, before handling test accuracy was 63% and after handling it's 66%.
- So, handling outliers has some impact in accuracy which is important for a model to be more precise.

References

[1] https://www.analyticsvidhya.com/blog/2022/07/step-by-step-exploratory-data-analysis-eda-using-python/.

[2] https://medium.com/code-heroku/introduction-to-exploratory-data-analysis-eda-c0257f888676.

[3] https://www.youtube.com/watch?v=4HyTlbHUKSw&t=532s.

[4] https://www.youtube.com/watch?v=6D3VtEfCw7w&t=1113s.