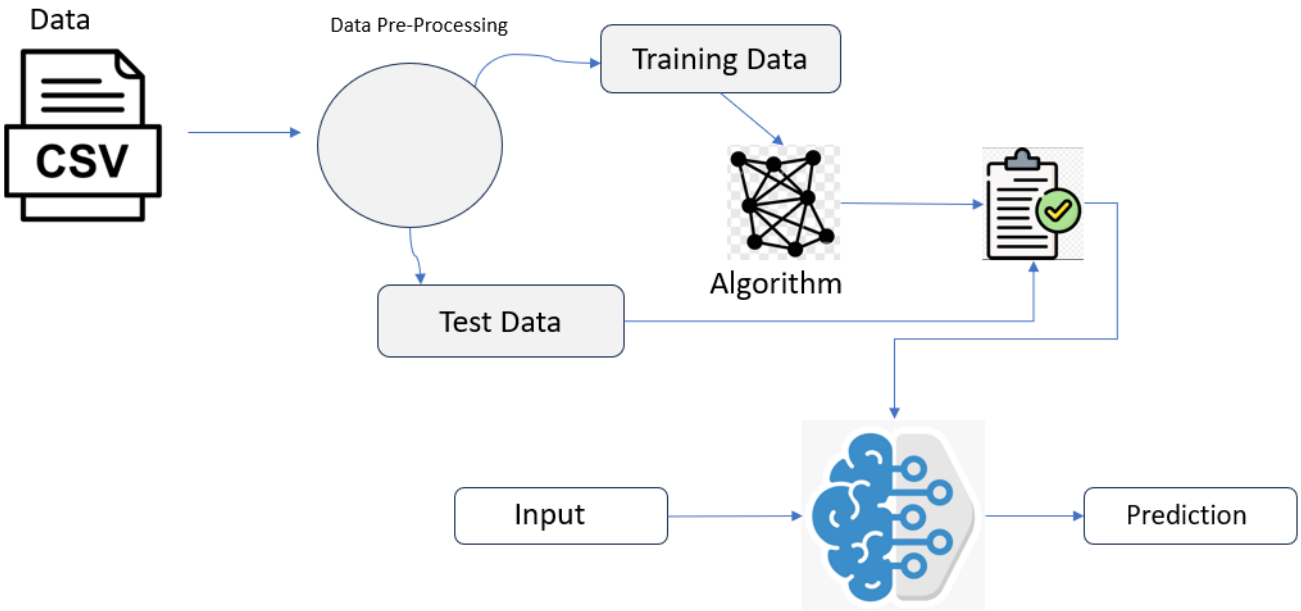
House Price Prediction Using Linear Regression – ML

Project Description:

House selling in different locations is very different and the market has changed drastically these days. Like every other market, house selling is also getting data driven each passing day. We've the historical data of house sold in past to which we have utilized in predicting the sell price of the houses. ML plays a pivotal role in house sell price prediction. ML Algorithms used are Linear Regression to predict the selling price of house.

Technical Architecture:



Pre requisites:

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

Anaconda navigator

PyCharm

Python packages:

- Open anaconda prompt as administrator
- Type "pip install numpy" and click enter.
- Type "pip install pandas" and click enter.
- Type "pip install scikit-learn" and click enter.
- Type "pip install matplotlib.pyplot" and click enter.
- Type "pip install seaborn" and click enter.
- Type "from sklearn.model_selection import train_test_split" and click enter.
- Type "from sklearn.linear_model import LinearRegression" and click enter.
- Type "from sklearn.metrics import r2_score" 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

Project Flow:

- We enter the data as input
- Entered input is analyzed by the model which is integrated.
- Once model analyses the input the prediction is showcased as output

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

- Dataset
- Visualizing and analysing the data
- Importing the libraries
- Read the Dataset
- Data pre-processing
- Handling missing values
- Descriptive analysis
- Splitting the dataset as x and y
- Handling Categorical Values
- Checking Correlation
- Converting Datatype
- Splitting dataset into training and test set
- Model building
- Import the model building libraries
- Initializing the model
- Training and testing the model
- Evaluating performance of model
- Save the model

Milestone 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. Dataset was given to us already for this project.

Activity 1: Download the dataset: We're already given the dataset.

Milestone 2: Visualizing and analysing the data

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

Activity 2: Importing the libraries

Import the necessary libraries as shown in the image.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

Activity 3: Read the Dataset

Our dataset format is .csv. I have 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.

I had created a class data_import and given the dataset path to this class. Under this class I had created a constructor and given the path to the class. From here using pandas read_csv function, I had read the dataset.

```
        price  area  bedrooms  ...  airconditioning  parking furnishingstatus
0    13300000  7420         4  ...              yes        2        furnished
1    12250000  8960         4  ...              yes        3        furnished
2    12250000  9960         3  ...               no        2   semi-furnished
3    12215000  7500         4  ...              yes        3        furnished
4    11410000  7420         4  ...              yes        2        furnished
..        ...   ...       ...  ...              ...      ...              ...
540   1820000  3000         2  ...               no        2      unfurnished
541   1767150  2400         3  ...               no        0   semi-furnished
542   1750000  3620         2  ...               no        0      unfurnished
543   1750000  2910         3  ...               no        0        furnished
544   1750000  3850         3  ...               no        0      unfurnished

[545 rows x 12 columns]
```

Milestone 3: Data Pre-processing

As we have understood how the data is lets pre-process the collected data. The given 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.

I had created a preprocessing function to perform all the preprocessing steps.

This activity includes the following steps.

- Handling missing values
- Descriptive analysis
- Splitting the dataset as x and y
- Handling Categorical Values
- Checking Correlation
- Converting Datatype
- Splitting dataset into training and test set

Activity 4: 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.

```
def preprocessing(self):
    try:
        |
        print(self.df.isnull().sum())
```

For which the result came as shown in below image,

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

Since there are no null values, I had checked the info of the data and the result is as shown below,I have used below command in the function to check this,
self.df.info()

```
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
```

Since, there are 6 categorical columns, I had converted these into numerical columns using below commands,

```
self.df["airconditioning"] = self.df["airconditioning"].replace({"yes": 1, "no": 0})
self.df["mainroad"] = self.df["mainroad"].replace({"yes": 1, "no": 0})
self.df["guestroom"] = self.df["guestroom"].replace({"yes": 1, "no": 0})
self.df["basement"] = self.df["basement"].replace({"yes": 1, "no": 0})
self.df["hotwaterheating"] = self.df["hotwaterheating"].replace({"yes": 1, "no": 0})
a = pd.get_dummies(self.df["furnishingstatus"], dtype=np.int64)
```

Since the furnishing status column had 3 different statuses, I had used get_dummies functions of pandas to convert this info numerical columns. It is shown in below image,

```
a = pd.get_dummies(self.df["furnishingstatus"], dtype=np.int64)
```

I had saved this in a variable a and then I had added this a to our dataframe using concat function of pandas.
I had also removed the categorical furnishing status columns using pandas drop function.

```
self.df = pd.concat([self.df, a], axis=1)
self.df = self.df.drop(["furnishingstatus"], axis=1)
```

Activity 5: Descriptive analysis

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

```
In [3]: df.describe()
```

Out[3]:

	price	area	bedrooms	bathrooms	stories	parking
count	5.450000e+02	545.000000	545.000000	545.000000	545.000000	545.000000
mean	4.766729e+06	5150.541284	2.965138	1.286239	1.805505	0.693578
std	1.870440e+06	2170.141023	0.738064	0.502470	0.867492	0.861586
min	1.750000e+06	1650.000000	1.000000	1.000000	1.000000	0.000000
25%	3.430000e+06	3600.000000	2.000000	1.000000	1.000000	0.000000
50%	4.340000e+06	4600.000000	3.000000	1.000000	2.000000	0.000000
75%	5.740000e+06	6360.000000	3.000000	2.000000	2.000000	1.000000
max	1.330000e+07	16200.000000	6.000000	4.000000	4.000000	3.000000

The below image shows how many unique values are there in each columns.

```
: df.nunique() #Checking the unique values in each columns of the dataset
```

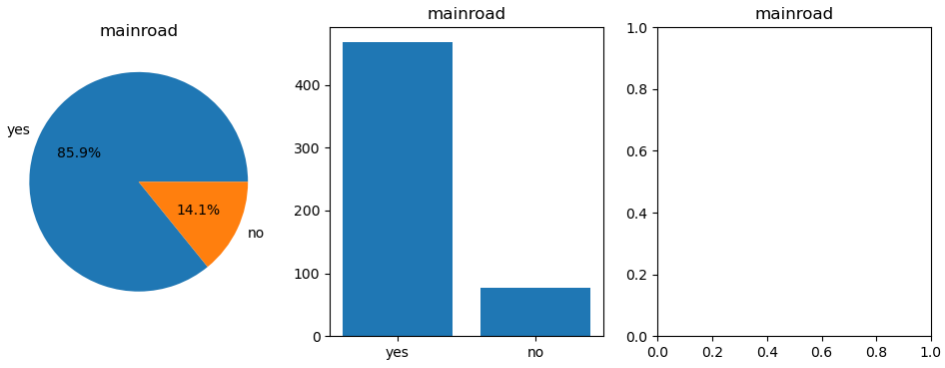
```
price              219
area               284
bedrooms             6
bathrooms            4
stories              4
mainroad             2
guestroom            2
basement             2
hotwaterheating      2
airconditioning      2
parking              4
furnishingstatus     3
dtype: int64
```

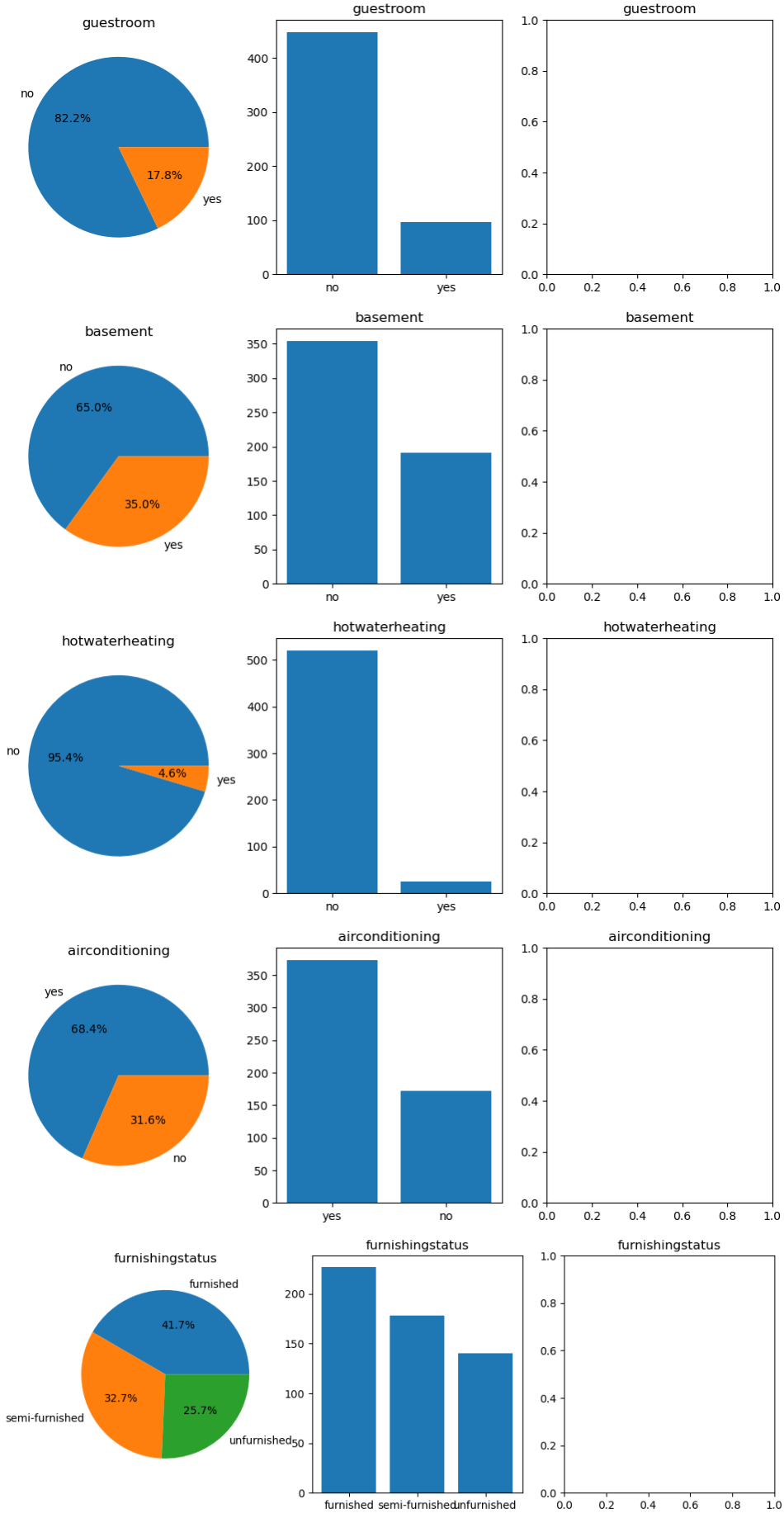
Activity 6: Univariate Analysis

To perform this analysis I had used below code,

```
In [10]: for col in df_cat.columns:
             plt.figure(figsize=(12,4))
             plt.subplot(1, 3, 1)
             plt.pie(df_cat[col].value_counts(), autopct="%1.1f%%", labels=df_cat[col].unique())
             plt.title(col)
             plt.subplot(1, 3, 2)
             plt.bar(df_cat[col].unique(), df_cat[col].value_counts())
             plt.title(col)
             plt.subplot(1, 3, 3)
             #col.plot.kde(df_cat[col].unique(), df_cat[col].value_counts())
             plt.title(col)
             plt.show()
```

Below is the result of this code,




guestroom
no
82.2%
17.8%
yes
guestroom
400
300
200
100
0
no
yes
guestroom
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
basement
no
65.0%
35.0%
yes
basement
350
300
250
200
150
100
50
0
no
yes
basement
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
hotwaterheating
no
95.4%
4.6%
yes
hotwaterheating
500
400
300
200
100
0
no
yes
hotwaterheating
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
airconditioning
yes
68.4%
31.6%
no
airconditioning
350
300
250
200
150
100
50
0
yes
no
airconditioning
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
furnishingstatus
furnished
41.7%
32.7%
semi-furnished
25.7%
unfurnished
furnishingstatus
200
150
100
50
0
furnished
semi-furnished
unfurnished
furnishingstatus
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Here I had applied a pie chart and countplot on the categorical data to check the values of each label in these features. I had used a for loop and applied it on dataset categorical data columns.

Mainroad:

85.9% houses are touching the mainroad and only 14.1% are not touching the mainroad. The same is shown in the pie chart. As the countplot is showing almost 470 Houses are touching mainroad and only 75 houses are not on mainroad.

Guestroom:

82.2% houses doesnt have a guestroom whereas 17.8% houses have it. The same is shown in the pie chart. As the countplot is showing almost 450 Houses doesnt have a guestroom, whereas approx 95 houses does have it.

Basement:

65% houses doesnt have a basement whereas 35% houses have it. The same is shown in the pie chart. As the countplot is also showing almost 355 Houses doesnt have a basement, whereas approx 190 houses does have it.

HotWaterHeating:

95.4% houses doesnt have HotWaterHeating facility, whereas only 4.6% houses have it. The same is shown in the pie chart. The countplot is also showing almost 520 Houses doesnt have HotWaterHeating facility, only 25 houses have it.

Airconditioning:

68.4% houses does have Air_Conditioning installed, whereas 31.6% doesn't. The same is shown in the pie chart. As the countplot is showing almost 375 Houses have Air_Conditioning installed, whereas approxx 170 houses doesnt have it.

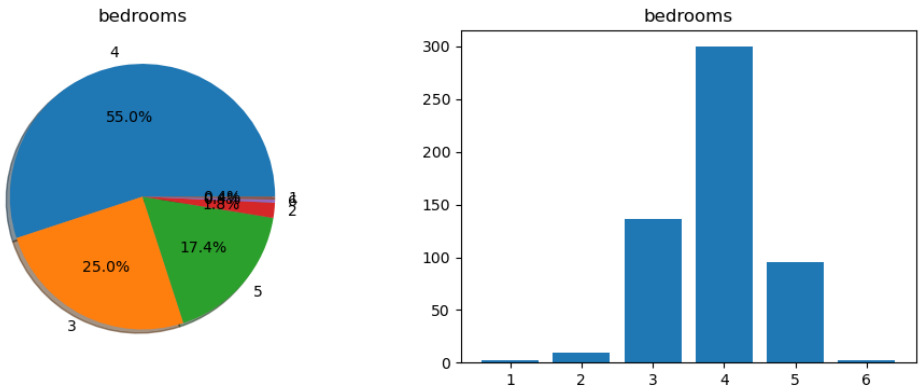
Furnishing_Status:

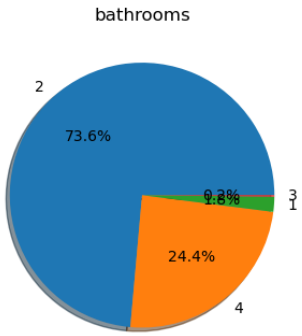
41.7% houes are furnished, 32.7% are semi-furnished, whereas 25.7% are unfurnished. The same is shown in the pie chart. The countplot is approxx 230 houes are furnished, approxx 180 are semi-furnished, whereas approxx 135 houses are unfurnished.

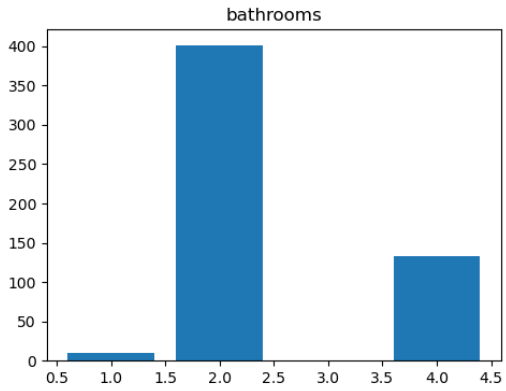
To analyze the numerical data columns, I had used below code:

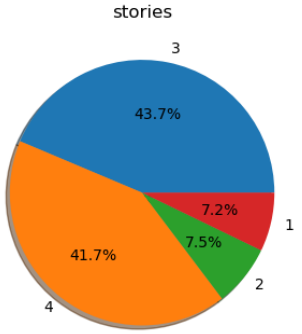
```
for col in df_int_1.columns:
    plt.figure(figsize=(12,4))
    plt.subplot(1, 2, 1)
    plt.pie(df_int_1[col].value_counts(), autopct="%1.1f%%", labels=df_int_1[col].unique(), shadow=True)
    plt.title(col)
    plt.subplot(1, 2, 2)
    plt.bar(df_int_1[col].unique(), df_int_1[col].value_counts())
    plt.title(col)
    plt.show()
```

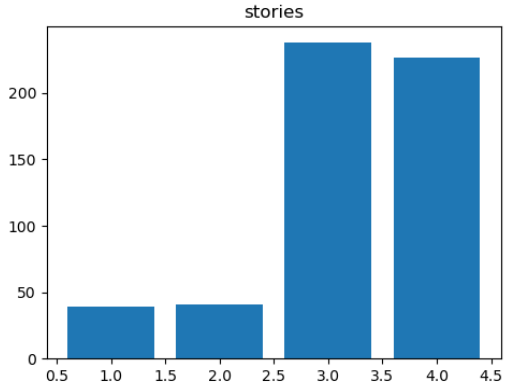
To which below is the result,

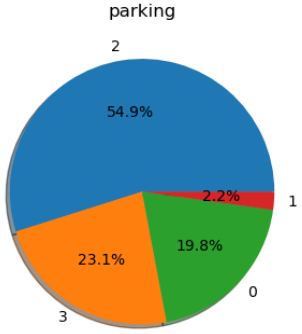


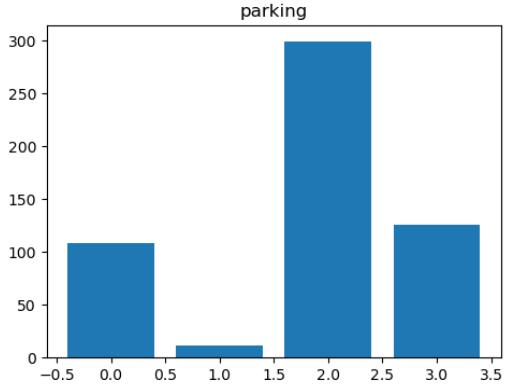












Bedroom ¶

In our dataset. maximum houses have 4 bedrooms, which takes 55% of total houses. At second place we have the demand of 3 Bedroom houses and then 5 bedroom. 1, 2 and 6 bedroom demands were too low.

Bathrooms

In our dataset. maximum houses have 2 bathrooms, which takes 73.6% of total houses. At second place we have the demand of 4 bathrooms houses. Demand for 1 and 3 bathroom houses were too low.

Stories

Majority of the customeres have preferred 3 & 4 story houses, which conclude almost 85% people went for 3 and 4 story houses. Demand for 1 and 2 story houses were too low.

Parking

Majority of the customeres have preferred at least 2 OR 3 parking spaces in their house, which include almost 78% of the people went for 2 and 3 parking spaces. There are people who went for house with 0 parking space. Whereas only 2.2% of the people selected at least 1 parking space in their house.

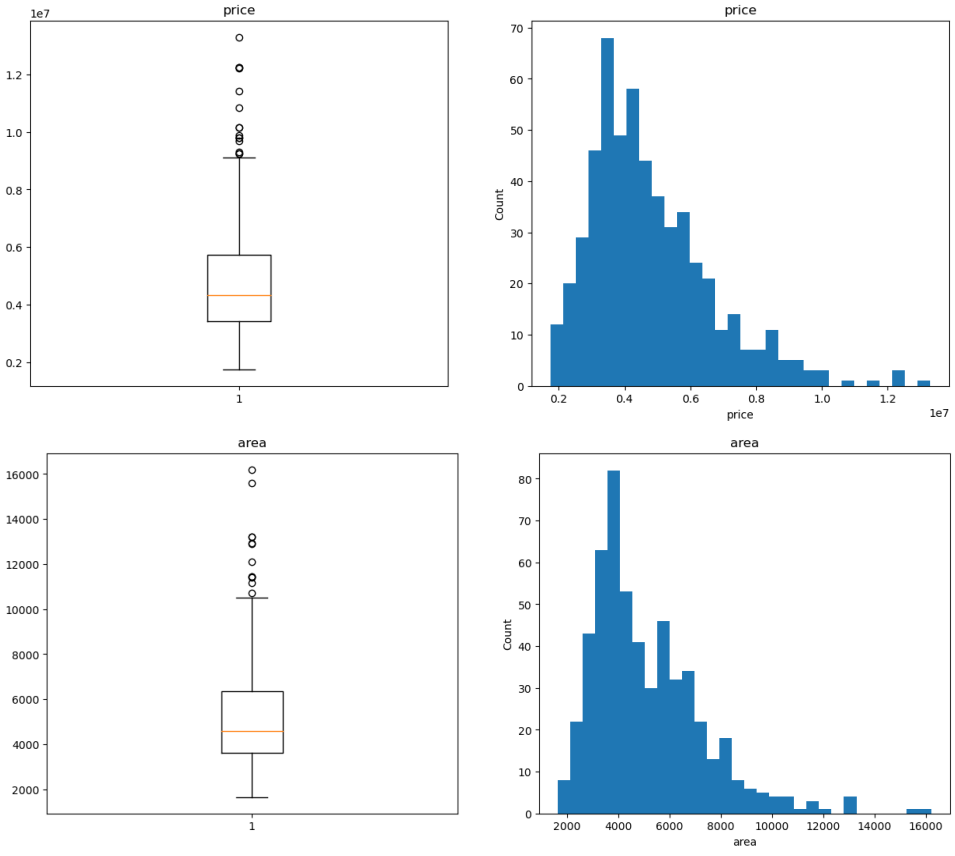
To do the analysis between price and area, I had used below code

```
df_int_2 = df_int.iloc[:, :2]
df_int_2.head()
```

	price	area
0	13300000	7420
1	12250000	8960
2	12250000	9960
3	12215000	7500
4	11410000	7420

```
for col in df_int_2.columns:
    plt.figure(figsize=(15,6))
    plt.subplot(1, 2, 1)
    plt.boxplot(x=df_int_2[col])
    plt.title(col)
    plt.subplot(1, 2, 2)
    plt.hist(df_int_2[col], bins=30)
    plt.xlabel(col)
    plt.ylabel("Count")
    plt.title(col)
    plt.show()
```

Below are the results of it,



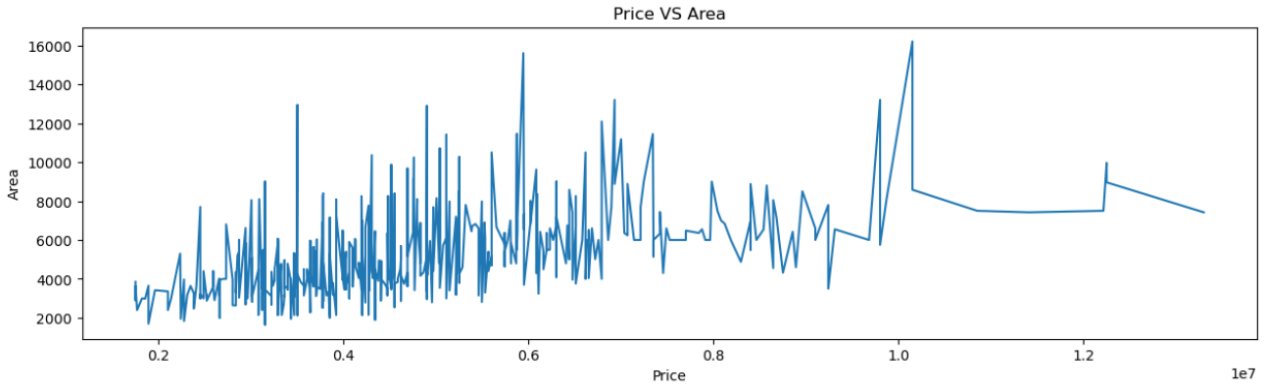
Price

Majority of the customers purchased the house between 3.5 Million to (almost) 6 Million.

Area

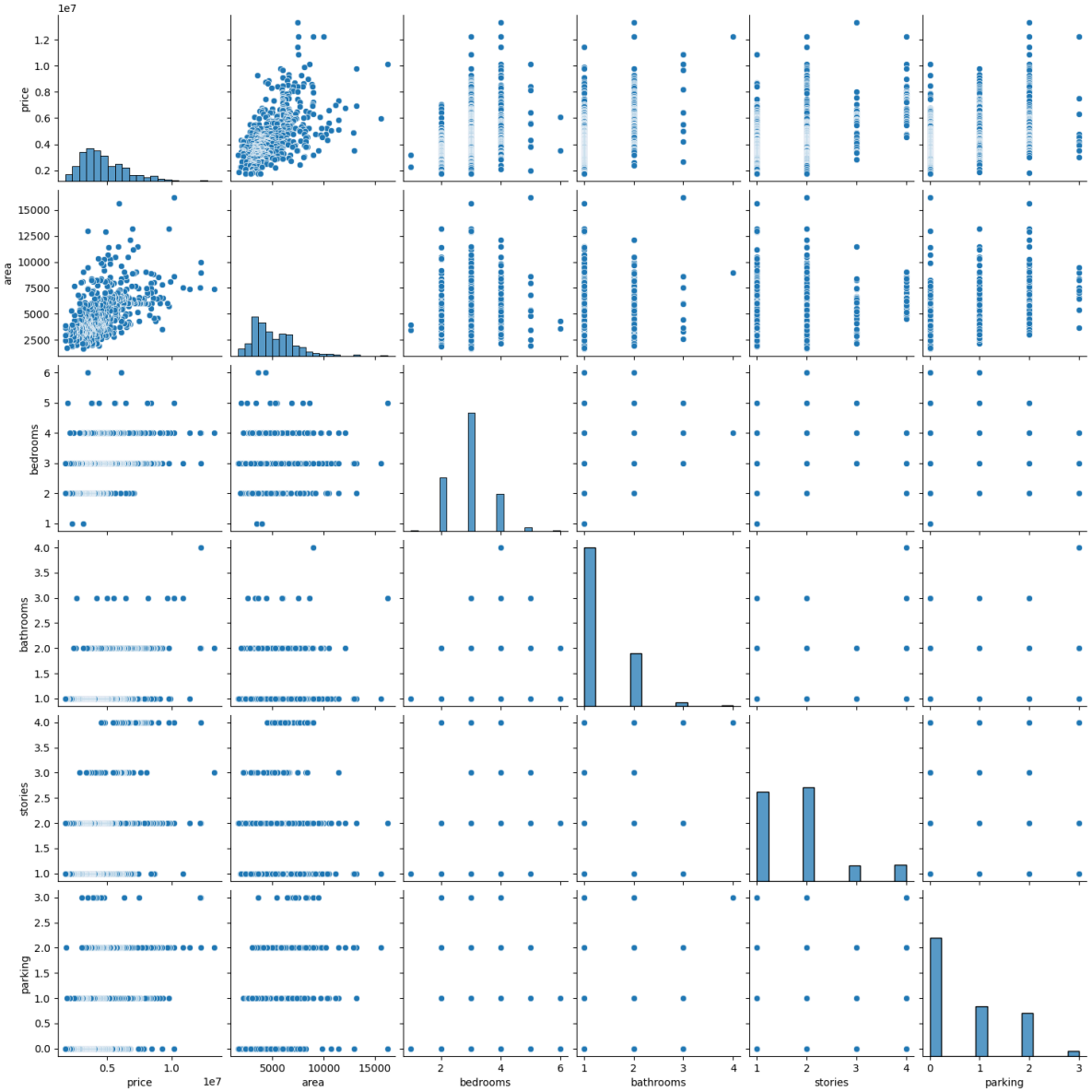
Majority of the customers purchased the house which has the area betwen 3500 SQFT - 7000 SQFT.

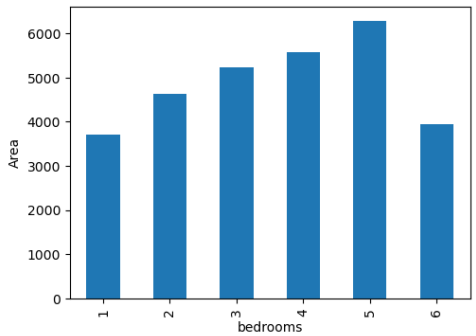
```
plt.figure(figsize=(15,4))
plt.plot(df_int_2["price"], df_int_2["area"])
plt.xlabel("Price")
plt.ylabel("Area")
plt.title("Price VS Area")
plt.show()
```

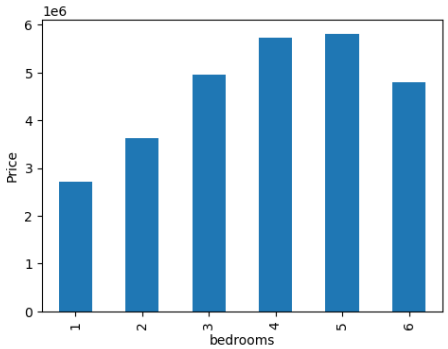


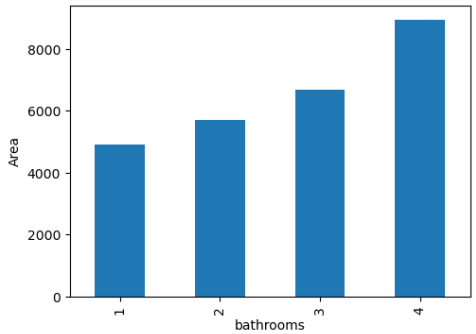

```
As the above comparision depicts that majority of the customers has chooses the house between 3.5 M - 6.5 M (Approxx). In the
same range the available area is betweeen 3500-7K SQFT.
```

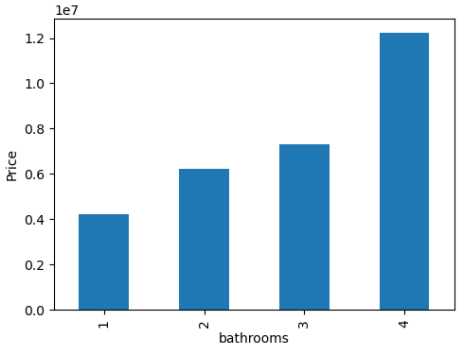
Bi-Variate Analysis:

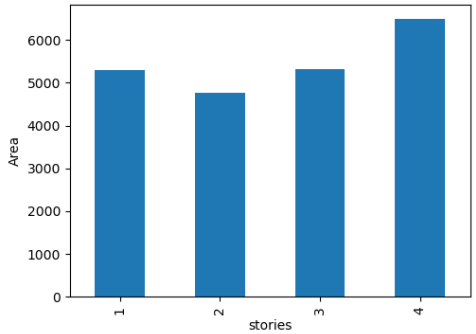


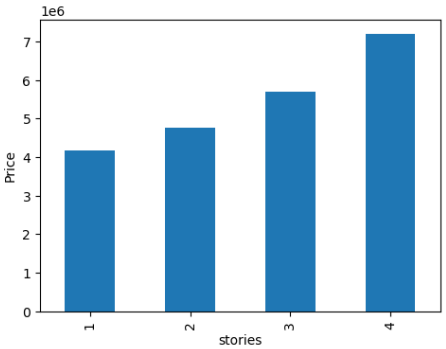


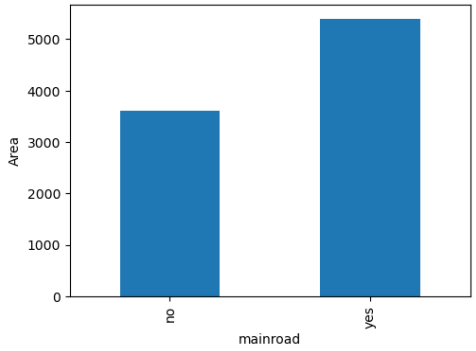


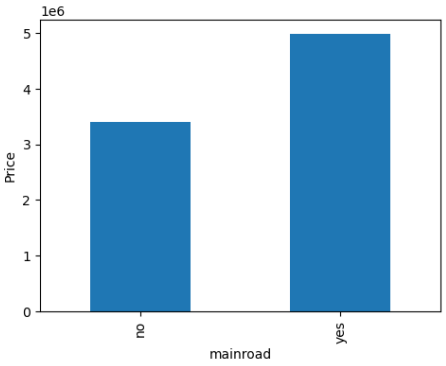

Area
0
2000
4000
6000
8000
1
2
3
4
bathrooms

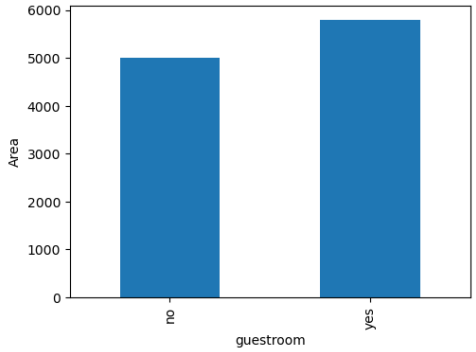

1e7
Price
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1
2
3
4
bathrooms

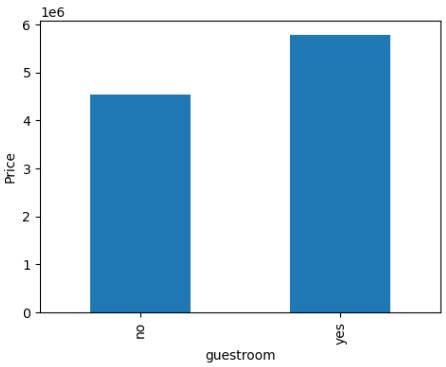

Area
0
1000
2000
3000
4000
5000
6000
1
2
3
4
stories

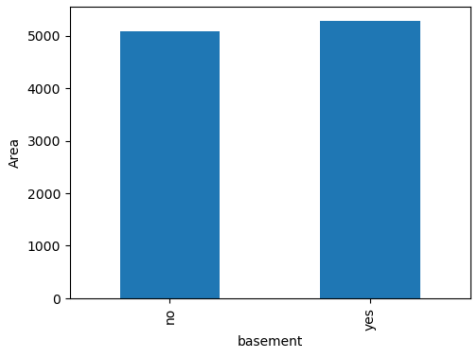

1e6
Price
0
1
2
3
4
5
6
7
1
2
3
4
stories

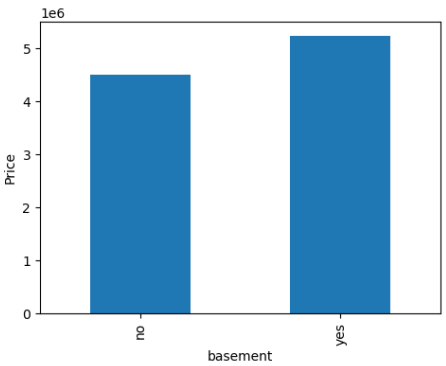

Area
0
1000
2000
3000
4000
5000
no
yes
mainroad

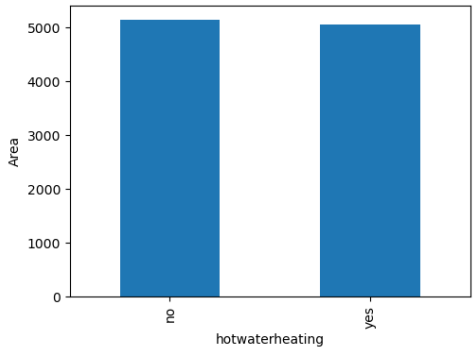

1e6
Price
0
1
2
3
4
5
no
yes
mainroad

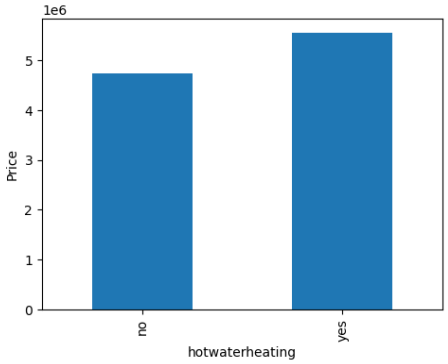

Area
0
1000
2000
3000
4000
5000
6000
no
yes
guestroom

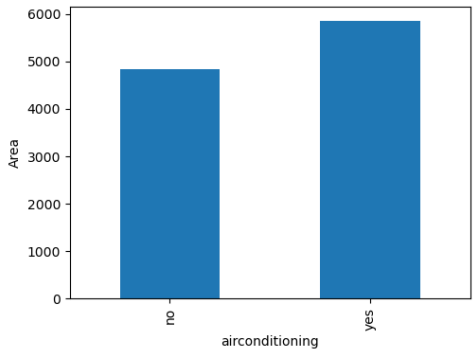

1e6
Price
0
1
2
3
4
5
6
no
yes
guestroom

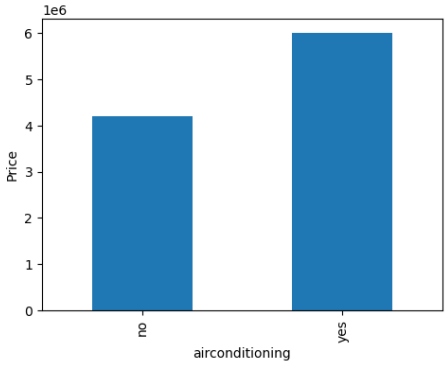

Area
0
1000
2000
3000
4000
5000
no
yes
basement

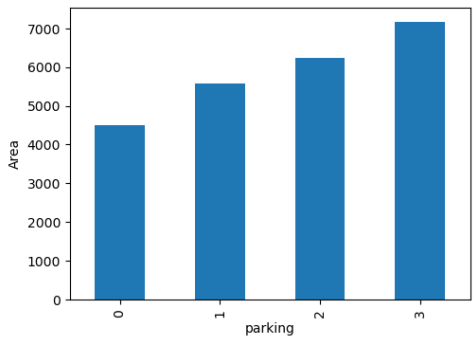

1e6
Price
0
1
2
3
4
5
no
yes
basement

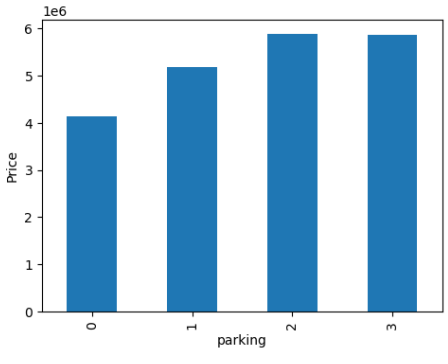

Area
0
1000
2000
3000
4000
5000
no
yes
hotwaterheating

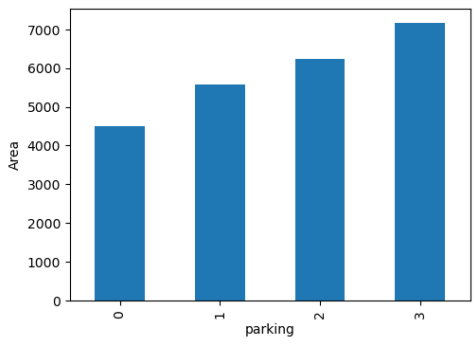

1e6
Price
0
1
2
3
4
5
no
yes
hotwaterheating

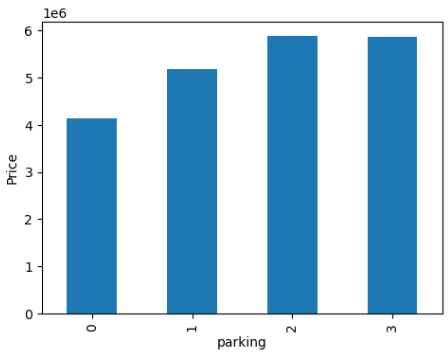

Area
0
1000
2000
3000
4000
5000
6000
no
yes
airconditioning

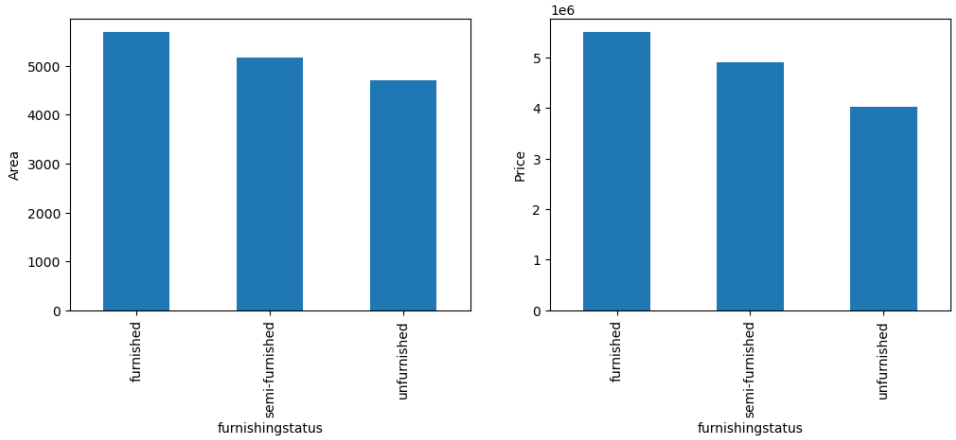

1e6
Price
0
1
2
3
4
5
6
no
yes
airconditioning


Area
0
1000
2000
3000
4000
5000
6000
7000
0
1
2
3
parking


1e6
Price
0
1
2
3
4
5
6
0
1
2
3
parking


Area
0
1000
2000
3000
4000
5000
6000
7000
0
1
2
3
parking


1e6
Price
0
1
2
3
4
5
6
0
1
2
3
parking



Bedroom vs area vs price:

By checking the bedroom with mean value of area and price, I found that 1 Bedroom housee has taken an area of approxx 3600 SQFT, whereas the prces for it is in the range of 2.7 - 2.8 M. 2 Bedroom house has the average area of 4500 SQFT annd the price range is 3.6 M. 3 Bedroom house has an average area of 5200 SQFT, whereas the price for it in the range of 5M (Approx). 4 Bedroomm house has an aread of 5500 SQFT, whereas the prices for this is 5.7 M (Approxx), 5 Bedroom house has the average area above 6K SQFT, wheras the prices for it in the range of 5.8-5.9 M, 6 Bedroom houses are built on an average area of 3800 SQFT, whereas the price for it lies around 5M.

Bathrooms vs area vs price:

The house having 1 bathroom has an average area of 4800 SQFT and the price range for this is around 4M. 2 bathroom house are having an average area of around 6K SQFT and prices in the range of 6M, 3 Bathroom house have an area of around 6700 SQFT, whereas the price is in the range of 7.5M, 4 Bathrroom house are built on around 9500 SQFT area, whereas the price is highest of around 12M.

Stories vs area vs price:

Majority of the house built with 1 story have an area around 5200 SQFT, wheereas the 2 story houses have 4800 SQFT, the price for 1 Sotry house is ranign 4M, whereas for story it is around 4.8M. 3 Story houses are havin an area of 5300 SQFT with the pricce range of 5.8M, 4 story house are occupying the highest area with higher price, the area is 6500+ whereas the price is 7.4M+.

Mainroad vs area vs price:

Houses bulit on an average area of 3500, the mainroad connectivity is not there and price range of house with no mainroad connectivity is around 3.5M, whereas the houses having mainroad connectivity have an average area of 5300 with the price rangin 5M+.

Multi-Variate Analysis: Checking the correlation using heatmap

```
plt.figure(figsize=(8,5))
sns.heatmap(data=df_int.corr(method="pearson"), annot=True)
plt.show()
```

Plotting the correlation of the int columns it shows that no column is highly correlated. Hence, we need to keep all the columns to train our model.

Activity 7: Splitting data into train and test

To perform this action, I had used below code,

```
def split_data(self):
    try:
        x = self.df.iloc[:, 1:]
        y = self.df.iloc[:, 0]
        x_train, x_test, y_train, y_test = train_test_split(x,y, random_state=42, test_size=0.2)
        return x_train, x_test, y_train, y_test
    except Exception as e:
        print(f"The error is in main {e.__str__()}")
```

Here I had created a function name split_data. I followed below steps here,

1. I separated the dataset in x & y to split it.
2. Using tra_test_split function, I had split the data keeping the test data size 20%, whereas the training data as 80% from the original dataset. I also passed random_sate to keep the output stable.
3. I returned the x_train, x_test, y_train, y_test values to reuse those while training and testing the modal later.

X values:

	area	bedrooms	bathrooms	...	furnished	semi-furnished	unfurnished
0	7420	4	2	...	1	0	0
1	8960	4	4	...	1	0	0
2	9960	3	2	...	0	1	0
3	7500	4	2	...	1	0	0
4	7420	4	1	...	1	0	0
..	...	...	...	...	...	...	...
540	3000	2	1	...	0	0	1
541	2400	3	1	...	0	1	0
542	3620	2	1	...	0	0	1
543	2910	3	1	...	1	0	0
544	3850	3	1	...	0	0	1

Y values:

0	13300000
1	12250000
2	12250000
3	12215000
4	11410000
	...
540	1820000
541	1767150
542	1750000
543	1750000
544	1750000

X_train values:

	area	bedrooms	bathrooms	...	furnished	semi-furnished	unfurnished
543	2910	3	1	...	1	0	0
9	5750	3	2	...	0	0	1
533	2400	3	1	...	0	0	1
274	6450	4	1	...	0	1	0
465	3800	2	1	...	0	0	1
..	...	...	...	...	...	...	...
71	6000	4	2	...	0	0	1
106	5450	4	2	...	0	1	0
270	4500	3	2	...	1	0	0
435	4040	2	1	...	0	0	1
102	5500	3	2	...	0	1	0

[490 rows x 13 columns]

Y_train values:

```
543    1750000
9      9800000
533    2100000
274    4340000
465    3045000
        ...
71     6755000
106    6160000
270    4340000
435    3290000
102    6195000
Name: price, Length: 490, dtype: int64
```

X_test values:

	area	bedrooms	bathrooms	...	furnished	semi-furnished	unfurnished
316	5900	4	2	...	0	0	1
77	6500	3	2	...	1	0	0
360	4040	2	1	...	0	1	0
90	5000	3	1	...	0	1	0
493	3960	3	1	...	1	0	0
209	6720	3	1	...	0	0	1
176	8520	3	1	...	1	0	0
249	4990	4	2	...	1	0	0
516	3240	2	1	...	0	0	1
426	2700	3	1	...	1	0	0
6	8580	4	3	...	0	1	0
497	3934	2	1	...	0	0	1
422	3720	2	1	...	0	0	1
424	3100	3	1	...	0	1	0
529	3970	3	1	...	0	0	1
499	3630	3	3	...	0	0	1
498	2000	2	1	...	0	1	0
55	6000	3	1	...	0	0	1
476	5850	3	1	...	0	0	1
486	6000	2	1	...	0	1	0
72	5020	3	1	...	0	0	1
163	6825	3	1	...	0	1	0
538	3649	2	1	...	0	0	1
174	3800	3	1	...	0	0	1
304	8250	3	1	...	0	1	0
2	9960	3	2	...	0	1	0
463	3090	2	1	...	0	0	1

463	3090	2	1	...	0	0	1
184	3000	3	2	...	1	0	0
10	13200	3	1	...	1	0	0
512	3000	2	1	...	1	0	0
70	4000	3	2	...	0	1	0
398	3120	3	1	...	0	0	1
79	6000	3	2	...	1	0	0
483	6615	3	1	...	0	1	0
429	4775	4	1	...	0	0	1
296	4600	3	2	...	0	1	0
210	4646	3	1	...	0	1	0
431	3180	4	1	...	0	0	1
394	3480	3	1	...	0	0	1
523	2787	4	2	...	1	0	0
158	7980	3	1	...	0	1	0
367	3630	2	1	...	0	0	1
76	6420	3	2	...	1	0	0
199	4200	3	1	...	1	0	0
451	6750	2	1	...	0	1	0
255	5885	2	1	...	0	0	1
83	6000	3	2	...	0	1	0
137	4640	4	1	...	0	1	0
473	8050	2	1	...	0	0	1
540	3000	2	1	...	0	0	1
30	7475	3	2	...	0	0	1
517	3000	2	1	...	0	0	1
284	7770	2	1	...	1	0	0
324	4500	3	1	...	0	1	0
440	3640	4	1	...	0	0	1
440	3640	4	1	...	0	0	1

[55 rows x 13 columns]

Y_test values:

316	4060000
77	6650000
360	3710000
90	6440000
493	2800000
209	4900000
176	5250000
249	4543000
516	2450000
426	3353000
6	10150000
497	2660000
422	3360000
424	3360000
529	2275000
499	2660000
498	2660000
55	7350000
476	2940000
486	2870000
72	6720000
163	5425000
538	1890000
174	5250000
304	4193000
2	12250000
463	3080000
184	5110000

```
184     5110000
10      9800000
512     2520000
70      6790000
398     3500000
79      6650000
483     2940000
429     3325000
296     4200000
210     4900000
431     3290000
394     3500000
523     2380000
158     5495000
367     3675000
76      6650000
199     4907000
451     3150000
255     4480000
83      6580000
137     5740000
473     3003000
540     1820000
30      8400000
517     2450000
284     4270000
324     4007500
440     3234000
Name: price, dtype: int64
```

Milestone 4: Model Building

Activity 1:
Linear_Regression Modal:

A function named training_modal is created and training data are passed as the parameters. Inside the function, Linear_Regression algorithm is called which was initialized under the class. Training data is passed to the model with .fit() function. Training data predction is calculated using modal.predict() function and saved in a new variable. For evaluating the model training accuracy, we've used r2_score and passed the actualy training data and predicted data.

```
def training_modal(self, x_train, x_test, y_train, y_test):
    try:
        self.modal.fit(x_train, y_train)
        y_train_pred = self.modal.predict(x_train)
        print(f" The training accuracy is: {r2_score(y_train, y_train_pred)}")
    except Exception as e:
        print(f"Error in Training Modal {e.__str__()}")
```

Testing the modal:

To test the modal I had created a test_moda function and using modal.predict() function, I had created the test prediction data and saved it in a variable. Post that I had calculated the accuracy using r2_score.

```
def test_modal(self, x_test, y_test):
    try:
        y_test_pred = self.modal.predict(x_test)
        print(f"The Test Accuracy of the Modal is: {r2_score(y_test, y_test_pred)}")

    except Exception as e:
        print(f"The error is in Test Modal {e.__str__()}")
```

Below is the accuracy score of the modal,

```
 The training accuracy is: 0.6614271587596954
The Test Accuracy of the Modal is: 0.6561849937714236
```