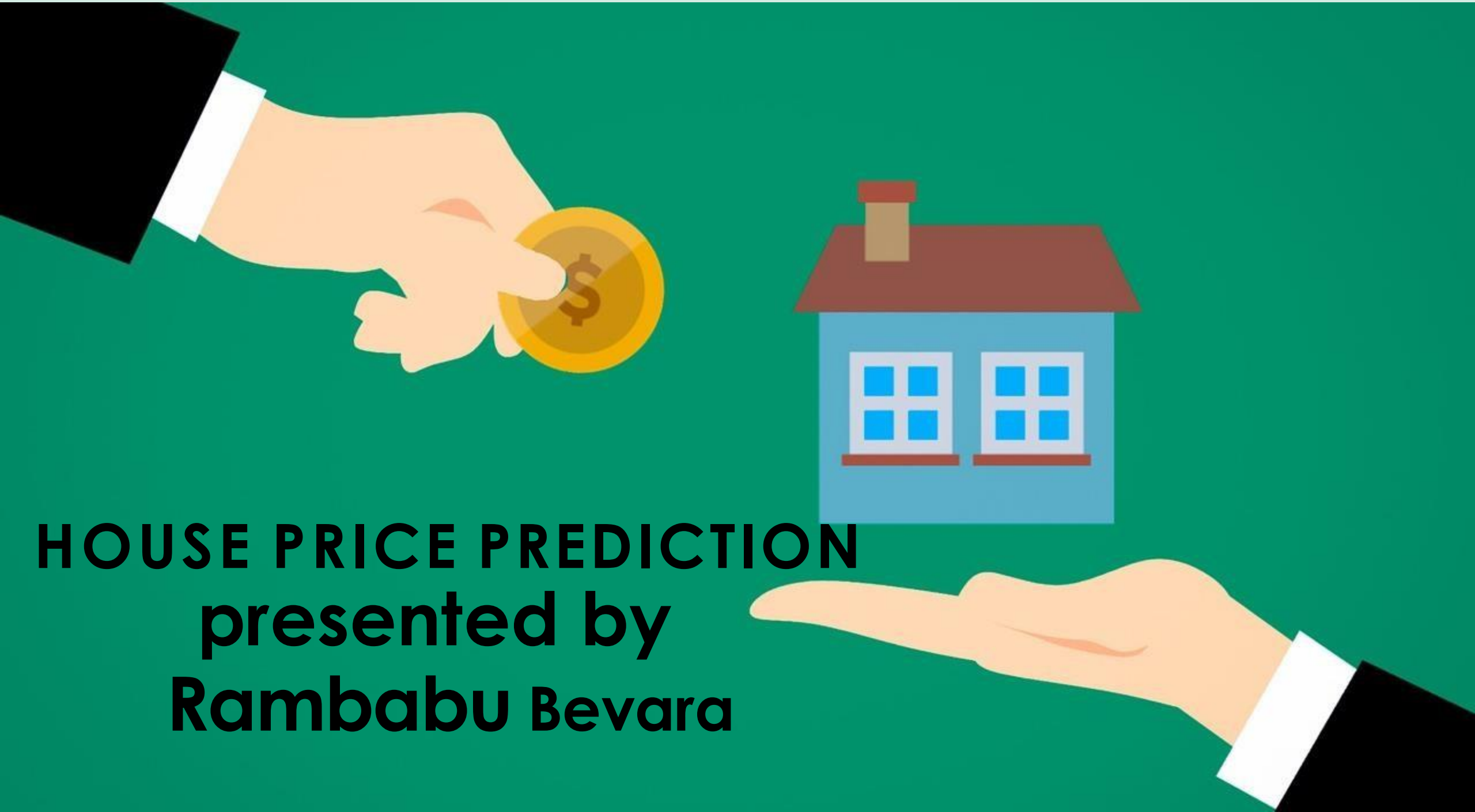


HOUSE PRICE PREDICTION

presented by

Rambabu Bevara

OUTLINE

- Introduction
- Problem statement
- Dataset & Cleaning
- Feature Engineering
- Outliers Detection & Filling
- One Hot Encoding
- Model Creation
- Output

INTRODUCTION

- This Project ,we have built a machine learning model to predict the house prices of an dataset from cities in india.
- This project will very helpful for the real estate market.
- Our model can be used by both house sellers and house buyers.
- Multiple Linear Regression algorithim is used to creat a model with great accuracy score
- We will aplly Lasso & Ridge models to find out the most optimum model with Maximum R-squared value.

PROBLEM STATEMENT

- Real estste is not only the key sector of the national economy,but also one of the citizen's major concerns.
- With increasing demand of housing ,prices of houses are also going up.
- Delivering precise forecasts for housing market prices is essential to address this growing challenge.
- Establishing fair and well-founded property valuations is crucial for restoring transparency and building consumer confidence, particularly in markets like India where trust is paramount.

SPECIFICATIONS

- This project aims to forecast housing prices in c by anities in india alyzing key attributes.
- The predictive model is built using a dataset specifically containing some cities in india housing market data.
- We employ machine learning techniques to develop this forecasting tool.
- Specifically, the model is trained and evaluated using the Multiple Linear Regression algorithm and Ridge & Lasso .

DATASET

- The dataset is about House price Prediction
- Its contains 187531 rows × 21 columns
- The columns are :[Index, Title,Description,Amount(in rupees),Price (in rupees), location,Carpet Area,Status,Floor,Transaction,Furnishing,facing,overlooking, Society,Bathroom,Balcony,Car Parking,Ownership,Super Area,Dimensions,Plot Area]
- We have used a total of 9 features to train our machine learning model

DATA LOADING

jupyter House Price Prediction Last Checkpoint: 5 hours ago

File Edit View Run Kernel Settings Help

Code JupyterLab Python [conda env:base] *

```
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
house=pd.read_csv("C://Users//nisha//Downloads//house_prices.csv (1)//house_prices.csv")
house
```

[2]:

	Index	Title	Description	Amount(in rupees)	Price (in rupees)	location	Carpet Area	Status	Floor	Transaction	...	facing	overlooking	Society	Bathroom	Ba
0	0	1 BHK Ready to Occupy Flat for sale in Srushti...	Bhiwandi, Thane has an attractive 1 BHK Flat f...	42 Lac	6000.0	thane	500 sqft	Ready to Move	10 out of 11	Resale	...	NaN	NaN	Srushti Siddhi Mangal Murti Complex	1	
1	1	2 BHK Ready to Occupy Flat for	One can find this stunning 2 BHK flat for	98 Lac	13799.0	thane	473 sqft	Ready to Move	3 out of 22	Resale	...	East	Garden/Park	Dosti Vihar	2	

DATA CLEANING

- The core purpose of data cleaning is to uncover and remove inaccuracies and repeated data points, ensuring the dataset is trustworthy and precise.
- This task is commonly performed with the help of the popular Python library, pandas.
- At the outset, columns irrelevant to predicting the final price are excluded from the Dataset
- Any rows containing missing information in one or more fields are eliminated to preserve data quality.
- Columns mixing textual and numerical data are cleaned by converting all values into integers for consistency.

➢ Checking null values

```
[5]: house.isnull().mean()*100
```

```
[5]: Index                  0.000000
     Title                  0.000000
     Description            1.612000
     Amount(in rupees)      0.000000
     Price (in rupees)      9.419776
     location               0.000000
     Carpet Area           43.018488
     Status                 0.327946
     Floor                  3.773776
     Transaction            0.044259
     Furnishing             1.544811
     facing                37.451408
     overlooking           43.425354
     Society               58.485264
     Bathroom               0.441527
     Balcony               26.094352
     Car Parking           55.114621
     Ownership             34.936624
```

➢ Dropping unnecessary columns

```
8]: house['Title']=house['Title'].str[:5]
```

```
9]: house['Title']=house['Title'].replace({' Apar':1,'> 10 ':10,' Buil':3,' Stud':11})
    house
```

9]:

	Title	Amount(in rupees)	location	Floor	Transaction	Furnishing	facing	Bathroom	Balcony
0	1 BHK	42 Lac	thane	10 out of 11	Resale	Unfurnished	NaN	1	2
1	2 BHK	98 Lac	thane	3 out of 22	Resale	Semi-Furnished	East	2	NaN
2	2 BHK	1.40 Cr	thane	10 out of 29	Resale	Unfurnished	East	2	NaN
3	1 BHK	25 Lac	thane	1 out of 3	Resale	Unfurnished	NaN	1	1
4	2 BHK	1.60 Cr	thane	20 out of 42	Resale	Unfurnished	West	2	NaN
...	...	...	...	...	...	...	...	...	...

➢ Extracting data & replacing

```
11]: house['BHK'] = house['Title'].str.extract(r'(\d+)', expand=False)
     house['BHK'] = pd.to_numeric(house['BHK'], errors='coerce')
     house
```

11]:

	Title	Amount(in rupees)	location	Floor	Transaction	Furnishing	facing	Bathroom	Balcony	BHK
0	1 BHK	42 Lac	thane	10 out of 11	Resale	Unfurnished	NaN	1	2	1.0
1	2 BHK	98 Lac	thane	3 out of 22	Resale	Semi-Furnished	East	2	NaN	2.0
2	2 BHK	1.40 Cr	thane	10 out of 29	Resale	Unfurnished	East	2	NaN	2.0

FEATURE ENGINEERING

- Feature engineering involves leveraging expert knowledge to derive meaningful attributes from unprocessed data using data mining methods. These crafted features play a crucial role in boosting the effectiveness of machine learning models. In essence, feature engineering can be viewed as a practical application of machine learning principles.

- To streamline our dataset, dimensionality reduction techniques are applied to eliminate less significant data points, focusing on those most relevant for predicting house prices.

➢ Filling Null values and coverting data types

House Price Prediction Last Checkpoint: 5 hours ago

lit View Run Kernel Settings Help

Code

```
house['Amount(in rupees)'].fillna(house['Amount(in rupees)'].median(),inplace=True)
house['Amount(in rupees)']=pd.to_numeric(house['Amount(in rupees)'],errors='coerce'

##2.Extracting & converting to_numeric Floor
# Extract leading number from the floor string
house['Floor']=house['Floor'].astype(str)
house['Floor'] = house['Floor'].str.extract(r'(\d+)')
house['Floor'] = pd.to_numeric(house['Floor'], errors='coerce')
house['Floor'].fillna(house['Floor'].median(), inplace=True)

3.## filling Null values of Location with mode
house['location'].fillna(house['location'].mode()[0],inplace=True)

4.##Filling Null values of Transaction with mode
house['Transaction'].fillna(house['Transaction'].mode()[0],inplace=True)

##5.Filling Null Values Of Furnishing with mode
house['Furnishing'].fillna(house['Furnishing'].mode()[0],inplace=True)

##7.Filling Null Values of Bathrooms with Mode & Converting this into numeric
house['Bathroom'].fillna(house['Bathroom'].mode()[0],inplace=True)
house['Bathroom']=pd.to_numeric(house['Bathroom'],errors='coerce')

##8.Filing Null Values Of Balcony with Mode
house['Balcony'].fillna(house['Balcony'].mode()[0],inplace=True)
house['Balcony']=pd.to_numeric(house['Balcony'],errors='coerce')

##9.Fiulling Null VAlues of BHK with Mode
house['BHK'].fillna(house['BHK'].mode()[0],inplace=True)

house['facing'].fillna(house['facing'].mode()[0],inplace=True)
```

➢ Code for convert values

[13]:
```
def convert(val):
    val=val.strip()
    if 'Lac' in val:
        return float(val.replace('Lac','').strip())*1e5
    elif 'Cr' in val:
        return float(val.replace('Cr','').strip())*1e7
    else:
        return None
```

[14]:
```
house['Amount(in rupees)']=house['Amount(in rupees)'].apply(convert)
```

DETECTING OUTLIERS & FILLING

- Once an outlier is identified, attempt to correct the error if feasible; if not, exclude that data point from the dataset.
- Our dataset revealed inconsistencies in the relationships among certain attribute values.
- Consequently, such anomalous records are removed to ensure data integrity.
- Scatter plots serve as a visual tool to spot additional outliers, which are also eliminated from the dataset.

➢ Checking Outliers & filling

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python [conda env:base] *

BHK

[18]:
```
q1=house['BHK'].quantile(0.25)
q2=house['BHK'].quantile(0.75)
iqr=q2-q1
upper=q2+(1.5)*iqr
lower=q1-(1.5)*iqr
upper,lower
```

[18]:
```
(4.5, 0.5)
```

[19]:
```
mde=house['BHK'].median()
house['BHK']=house['BHK'].apply(lambda x:mde if x <lower or x >upper else x )
```

[20]:
```
house.boxplot(column='Amount(in rupees)')
plt.show()
```

[22]:
```
q1=house['Amount(in rupees)'].quantile(0.25)
q2=house['Amount(in rupees)'].quantile(0.75)
iqr=q2-q1
upper=q2+(1.5)*iqr
lower=q1-(1.5)*iqr
upper,lower
```

[22]:
```
(28990000.0, -9650000.0)
```

[23]:
```
mde=house['Amount(in rupees)'].median()
```

➢ Filling Null values and converting datatypes

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python [conda env:base] *

[69]:
```
house['Balcony']=house['Balcony'].fillna(0).astype(int)
house['Balcony']=house['Balcony'].fillna(house['Balcony'].mean()).round().astype(int)
house
```

[69]:

	Amount(in rupees)	location	Floor	Transaction	Furnishing	facing	Bathroom	Balcony	BHK
0	4200000.0	thane	10.0	Resale	Unfurnished	East	1.0	2	1.0
1	9800000.0	thane	3.0	Resale	Semi-Furnished	East	2.0	2	2.0

[71]:
```
house['Bathroom']=house['Bathroom'].fillna(0).astype(int)
house['Bathroom']=house['Bathroom'].fillna(house['Bathroom'].mean()).round().astype(int)
house
```

[71]:

	Amount(in rupees)	location	Floor	Transaction	Furnishing	facing	Bathroom	Balcony	BHK
0	4200000.0	thane	10.0	Resale	Unfurnished	East	1	2	1.0
1	9800000.0	thane	3.0	Resale	Semi-Furnished	East	2	2	2.0

➢ ONE HOT CODING

- This approach transforms categorical data into numerical format, making it suitable for machine learning models.
- In our dataset, the categorical feature we focus on is "location."
- We applied one hot encoding to convert each location into a numeric representation.

➢ ONE HOT CODING

[41]:
```
x=house[['Floor','location',
       'Transaction', 'Furnishing', 'Bathroom', 'Balcony','BHK']]
y=house['Amount(in rupees)']
```

[42]:
```
house['location'].unique()
```

[42]:
```
array(['thane', 'other', 'mumbai', 'ahmedabad', 'bangalore', 'chennai',
       'gurgaon', 'hyderabad', 'jaipur', 'kolkata', 'new-delhi', 'noida',
       'pune', 'bhiwadi', 'chandigarh', 'faridabad', 'goa',
       'greater-noida', 'kochi', 'mohali', 'ranchi', 'surat', 'vadodara',
       'visakhapatnam', 'zirakpur'], dtype=object)
```

[102]:
```
x=x['location_encoded'] = house['location'].map(house.groupby('location')['Amount(in rupees)'].mean())
x
```

jupyter House Price Prediction Last Checkpoint: 3 hours ago

File Edit View Run Kernel Settings Help Trusted

Markdown JupyterLab Python [conda env:base] *

[108]:
```
cat=x.select_dtypes(exclude='number')
num=x.select_dtypes(include='number')
```

[110]:
```
cat_x=pd.get_dummies(cat)
cat_x
```

[110]:

	Transaction_New Property	Transaction_Other	Transaction_Rent/Lease	Transaction_Resale	Furnishing_Furnished	Furnishing_Semi-Furnished	Furnishing_Unfurnished	facing_East	facin
0	False	False	False	True	False	False	True	True	
1	False	False	False	True	False	True	False	True	

➢ MODEL BUILDING

- **Modeling involves training a machine learning algorithm to accurately forecast labels based on input features.**
- **We applied the Linear Regression technique for both training and evaluating our model.**
- **Our model achieved an accuracy of 100%, reflecting strong predictive performance.**

model training

```
[129]: from sklearn.linear_model import LinearRegression,Ridge,Lasso
       from sklearn.metrics import mean_squared_error,r2_score

[131]: model=LinearRegression()
       model.fit(x_train,y_train)

[131]: LinearRegression
       LinearRegression()
```

Home | COrreleation | House Price Prediction | Cat Variable Encoding

localhost:8888/notebooks/House%20Price%20Prediction.ipynb?

Gmail YouTube All Bookmarks

jupyter House Price Prediction Last Checkpoint: 3 hours ago

File Edit View Run Kernel Settings Help Trusted

Markdown JupyterLab Python [conda env:base] *

[142]:
```
print('Test Mean squared error :',mean_squared_error(y_test,test_pred))
```
```
Test Mean squared error : 2.7900281376731636e-13
```

[144]:
```
print('Train Mean squared error :',mean_squared_error(y_train,train_pred))
```
```
Train Mean squared error : 2.805622538822866e-13
```

[146]:
```
print('Test R2 error :',r2_score(y_test,test_pred))
```
```
Test R2 error : 1.0
```

[148]:
```
print('Train R2 error :',r2_score(y_train,train_pred))
```
```
Train R2 error : 1.0
```

[134]:
```
test_pred=model.predict(x_test)
test_pred
```

[134]:
```
array([12100000.00000015,  7200000.00000045, 14999999.9999998 , ...,
        5100000.0000002 ,  7799999.99999993,  6400000.00000052])
```

[136]:
```
y_test
```

[136]:
```
94353     12100000.0
20969      7200000.0
90081     15000000.0
160714     4420000.0
80704      7800000.0
              ...
```

79°F Haze | Search | 11:04 PM 6/10/2025

Ridge Model

[151]:
```
ridge_model=Ridge(alpha=0.1)
ridge_model.fit(x_train,y_train)
```

```
C:\Users\nisha\anaconda3\Lib\site-packages\sklearn\linear_model\_ridge.py:216: LinAlgWarning: Ill-conditioned matrix (rcond=1.66969e-20):
result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

[151]:
```
Ridge
Ridge(alpha=0.1)
```

[153]:
```
ridge_test=ridge_model.predict(x_test)
ridge_train=ridge_model.predict(x_train)
```

[155]:
```
print('Ridge Model of MSE of train :',mean_squared_error(y_train,ridge_train))
```

```
Ridge Model of MSE of train : 2.114008490639501e-11
```

[159]:
```
print('Ridge model of R2 of train :',r2_score(y_train,ridge_train))
```

```
Ridge model of R2 of train : 1.0
```

[161]:
```
print('Ridge Model of R2 of test :',r2_score(y_test,ridge_test))
```

```
Ridge Model of R2 of test : 1.0
```

Lasso model

```
[ ]: Lasso_model=Lasso(alpha=0.1)
     Lasso_model.fit(x_train,y_train)
```

```
[ ]: Lasso_test=Lasso_model.predict(x_test)
     Lasso_train=Lasso_model.predict(x_train)
```

```
[ ]: print('Lasso Model of MSE of train :',mean_squared_error(y_train,Lasso_train))
```

```
[ ]: print('Lasso Model of MSE of test:',mean_squared_error(y_test,Lasso_test))
```

```
Lasso(alpha=0.1)
```

```
[166]: Lasso_test=Lasso_model.predict(x_test)
       Lasso_train=Lasso_model.predict(x_train)
```

```
[168]: print('Lasso Model of MSE of train :',mean_squared_error(y_train,Lasso_train))
```

```
Lasso Model of MSE of train : 1.3580493466373543e-14
```

```
[170]: print('Lasso Model of MSE of test:',mean_squared_error(y_test,Lasso_test))
```

```
Lasso Model of MSE of test: 1.3522017256084456e-14
```

```
[172]: print('Lasso model of R2 of train :',r2_score(y_train,Lasso_train))
```

```
Lasso model of R2 of train : 1.0
```

```
[174]: print('Lasso model of R2 of test :',r2_score(y_test,Lasso_test))
```

```
Lasso model of R2 of test : 1.0
```

➢ OUTPUT

- We created a function to predict the house price.
- Using Linear Regression & Lasso & Ridge Completed Building Prediction .
- When we pass the values into our function,it will predict house price for us.
- Where we got R-Squared value as 100% Accuracy.
- The Mean Squared error also in good .
- In real world it's Rare to come this type of values .
- It’s a good fit.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

house=pd.read_csv("C://Users//nisha//Downloads//house_prices.csv
(1)//house_prices.csv")
house
```

```
         Index                                                Title  \
0            0  1 BHK Ready to Occupy Flat for sale in Srushti...
1            1  2 BHK Ready to Occupy Flat for sale in Dosti V...
2            2  2 BHK Ready to Occupy Flat for sale in Sunrise...
3            3        1 BHK Ready to Occupy Flat for sale Kasheli
4            4  2 BHK Ready to Occupy Flat for sale in TenX Ha...
...        ...                                                ...
187526  187526  3 BHK Ready to Occupy Flat for sale in Bollywo...
187527  187527  3 BHK Ready to Occupy Flat for sale in Sushma ...
187528  187528  3 BHK Ready to Occupy Flat for sale in Bollywo...
187529  187529  2 BHK Ready to Occupy Flat for sale in Friends...
187530  187530  3 BHK Ready to Occupy Flat for sale in Affinit...

                                           Description Amount(in
rupees)  \
0       Bhiwandi, Thane has an attractive 1 BHK Flat f...           42
Lac
1       One can find this stunning 2 BHK flat for sale...           98
Lac
2       Up for immediate sale is a 2 BHK apartment in ...
1.40 Cr
3       This beautiful 1 BHK Flat is available for sal...           25
Lac
4       This lovely 2 BHK Flat in Pokhran Road, Thane ...
1.60 Cr
...                                                   ...
...
187526  This magnificent 3 BHK Flat is available for s...           63
Lac
187527  Have a look at this immaculate 3 BHK flat for ...           55
Lac
187528  Gazipur, Zirakpur has an appealing 3 BHK flat ...           76
Lac
187529  Up for immediate sale is a 2 BHK apartment in ...           30
Lac
187530  This exquisite 3 BHK Flat is offered for sale ...
1.18 Cr

        Price (in rupees)  location Carpet Area         Status
Floor  \
0                  6000.0     thane    500 sqft  Ready to Move  10 out
```

```
of 11
1                13799.0     thane    473 sqft  Ready to Move   3 out
of 22
2                17500.0     thane    779 sqft  Ready to Move  10 out
of 29
3                    NaN     thane    530 sqft  Ready to Move    1
out of 3
4                18824.0     thane    635 sqft  Ready to Move  20 out
of 42
...                  ...       ...         ...            ...
...
187526            3225.0  zirakpur         NaN  Ready to Move    2
out of 4
187527            3274.0  zirakpur         NaN  Ready to Move    4
out of 6
187528            4343.0  zirakpur   1250 sqft  Ready to Move    1
out of 3
187529            4231.0  zirakpur         NaN  Ready to Move    2
out of 2
187530            6162.0  zirakpur         NaN  Ready to Move   5 out
of 13

         Transaction  ...        facing             overlooking  \
0             Resale  ...           NaN                    NaN
1             Resale  ...          East             Garden/Park
2             Resale  ...          East             Garden/Park
3             Resale  ...           NaN                    NaN
4             Resale  ...          West  Garden/Park, Main Road
...              ...  ...           ...                    ...
187526  New Property  ...          East             Garden/Park
187527        Resale  ...  North - East  Garden/Park, Main Road
187528        Resale  ...          East  Garden/Park, Main Road
187529        Resale  ...           NaN              Main Road
187530        Resale  ...  North - East       Garden/Park, Pool

                                    Society Bathroom Balcony Car
Parking  \
0       Srushti Siddhi Mangal Murti Complex        1       2
NaN
1                              Dosti Vihar        2     NaN      1
Open
2                      Sunrise by Kalpataru        2     NaN   1
Covered
3                                       NaN        1       1
NaN
4               TenX Habitat Raymond Realty        2     NaN   1
Covered
...                                     ...      ...     ...         .
..
```

```
187526                      Bollywood Esencia        3       3   1
Covered
187527                     Sushma Urban Views        3     NaN   1
Covered
187528                      Bollywood Esencia        3       2  1
Covered,
187529                        Friends Enclave        2     NaN
NaN
187530                        Affinity Greens        4       4   1
Covered

                    Ownership Super Area Dimensions  Plot Area
0                         NaN        NaN        NaN        NaN
1                    Freehold        NaN        NaN        NaN
2                    Freehold        NaN        NaN        NaN
3                         NaN        NaN        NaN        NaN
4       Co-operative Society        NaN        NaN        NaN
...                        ...        ...        ...        ...
187526               Freehold  1953 sqft        NaN        NaN
187527                    NaN  1680 sqft        NaN        NaN
187528               Freehold        NaN        NaN        NaN
187529                    NaN   709 sqft        NaN        NaN
187530               Freehold  1915 sqft        NaN        NaN

[187531 rows x 21 columns]
```

```
house.columns
```

```
Index(['Index', 'Title', 'Description', 'Amount(in rupees)',
       'Price (in rupees)', 'location', 'Carpet Area', 'Status',
'Floor',
       'Transaction', 'Furnishing', 'facing', 'overlooking',
'Society',
       'Bathroom', 'Balcony', 'Car Parking', 'Ownership', 'Super
Area',
       'Dimensions', 'Plot Area'],
      dtype='object')
```

```
house.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 187531 entries, 0 to 187530
Data columns (total 21 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   Index              187531 non-null  int64
 1   Title              187531 non-null  object
 2   Description        184508 non-null  object
 3   Amount(in rupees)  187531 non-null  object
 4   Price (in rupees)  169866 non-null  float64
```

```
 5   location           187531 non-null  object
 6   Carpet Area        106858 non-null  object
 7   Status             186916 non-null  object
 8   Floor              180454 non-null  object
 9   Transaction        187448 non-null  object
 10  Furnishing         184634 non-null  object
 11  facing             117298 non-null  object
 12  overlooking        106095 non-null  object
 13  Society            77853 non-null   object
 14  Bathroom           186703 non-null  object
 15  Balcony            138596 non-null  object
 16  Car Parking        84174 non-null   object
 17  Ownership          122014 non-null  object
 18  Super Area         79846 non-null   object
 19  Dimensions         0 non-null       float64
 20  Plot Area          0 non-null       float64
dtypes: float64(3), int64(1), object(17)
memory usage: 30.0+ MB
```

```
house.isnull().mean()*100
```

```
Index                   0.000000
Title                   0.000000
Description             1.612000
Amount(in rupees)       0.000000
Price (in rupees)       9.419776
location                0.000000
Carpet Area            43.018488
Status                  0.327946
Floor                   3.773776
Transaction             0.044259
Furnishing              1.544811
facing                 37.451408
overlooking            43.425354
Society                58.485264
Bathroom                0.441527
Balcony                26.094352
Car Parking            55.114621
Ownership              34.936624
Super Area             57.422506
Dimensions            100.000000
Plot Area             100.000000
dtype: float64
```

```
house.shape
```

```
(187531, 21)
```

```
## Dropping columns
house.drop(['Index','Price (in
```

```
rupees)','Description','Status','overlooking','Car
Parking','Society','Dimensions','Ownership','Carpet Area','Super
Area','Plot Area'],axis=1,inplace=True)

house['Title']=house['Title'].str[:5]

house['Title']=house['Title'].replace({' Apar':1,'> 10 ':10,'
Buil':3,' Stud':11})
house
```

	Title	Amount(in rupees)	location	Floor	Transaction
0	1 BHK	42 Lac	thane	10 out of 11	Resale
1	2 BHK	98 Lac	thane	3 out of 22	Resale
2	2 BHK	1.40 Cr	thane	10 out of 29	Resale
3	1 BHK	25 Lac	thane	1 out of 3	Resale
4	2 BHK	1.60 Cr	thane	20 out of 42	Resale
...	...	...	...	...	...
187526	3 BHK	63 Lac	zirakpur	2 out of 4	New Property
187527	3 BHK	55 Lac	zirakpur	4 out of 6	Resale
187528	3 BHK	76 Lac	zirakpur	1 out of 3	Resale
187529	2 BHK	30 Lac	zirakpur	2 out of 2	Resale
187530	3 BHK	1.18 Cr	zirakpur	5 out of 13	Resale

	Furnishing	facing	Bathroom	Balcony
0	Unfurnished	NaN	1	2
1	Semi-Furnished	East	2	NaN
2	Unfurnished	East	2	NaN
3	Unfurnished	NaN	1	1
4	Unfurnished	West	2	NaN
...	...	...	...	...
187526	Semi-Furnished	East	3	3
187527	Unfurnished	North - East	3	NaN
187528	Furnished	East	3	2
187529	Semi-Furnished	NaN	2	NaN
187530	Semi-Furnished	North - East	4	4

[187531 rows x 9 columns]

```
house['Title'].unique()
```

```
array(['1 BHK', '2 BHK', '3 BHK', '4 BHK', '5 BHK', 11, '6 BHK', 1,
       '8 BHK', '7 BHK', 10, '9 BHK', '10 BH', 3], dtype=object)
```

```
house['BHK'] = house['Title'].str.extract(r'(\d+)', expand=False)
house['BHK'] = pd.to_numeric(house['BHK'], errors='coerce')
house
```

```
        Title Amount(in rupees)  location         Floor   Transaction
\
0       1 BHK           42 Lac      thane  10 out of 11        Resale

1       2 BHK           98 Lac      thane   3 out of 22        Resale

2       2 BHK          1.40 Cr      thane  10 out of 29        Resale

3       1 BHK           25 Lac      thane    1 out of 3        Resale

4       2 BHK          1.60 Cr      thane  20 out of 42        Resale

...       ...              ...       ...           ...           ...

187526  3 BHK           63 Lac   zirakpur    2 out of 4  New Property

187527  3 BHK           55 Lac   zirakpur    4 out of 6        Resale

187528  3 BHK           76 Lac   zirakpur    1 out of 3        Resale

187529  2 BHK           30 Lac   zirakpur    2 out of 2        Resale

187530  3 BHK          1.18 Cr   zirakpur   5 out of 13        Resale


             Furnishing        facing Bathroom Balcony  BHK
0          Unfurnished           NaN        1       2  1.0
1       Semi-Furnished          East        2     NaN  2.0
2          Unfurnished          East        2     NaN  2.0
3          Unfurnished           NaN        1       1  1.0
4          Unfurnished          West        2     NaN  2.0
...                ...           ...      ...     ...  ...
187526  Semi-Furnished          East        3       3  3.0
187527     Unfurnished  North - East        3     NaN  3.0
187528       Furnished          East        3       2  3.0
187529  Semi-Furnished           NaN        2     NaN  2.0
187530  Semi-Furnished  North - East        4       4  3.0

[187531 rows x 10 columns]
```

```
house.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 187531 entries, 0 to 187530
```

```
Data columns (total 10 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   Title               187531 non-null  object 
 1   Amount(in rupees)   187531 non-null  object 
 2   location            187531 non-null  object 
 3   Floor               180454 non-null  object 
 4   Transaction         187448 non-null  object 
 5   Furnishing          184634 non-null  object 
 6   facing              117298 non-null  object 
 7   Bathroom            186703 non-null  object 
 8   Balcony             138596 non-null  object 
 9   BHK                 186578 non-null  float64
dtypes: float64(1), object(9)
memory usage: 14.3+ MB
```

```
def convert(val):
    val=val.strip()
    if 'Lac' in val:
        return float(val.replace('Lac','').strip())*1e5
    elif 'Cr' in val:
        return float(val.replace('Cr','').strip())*1e7
    else:
        return None
```

```
house['Amount(in rupees)']=house['Amount(in rupees)'].apply(convert)
```

```
house
```

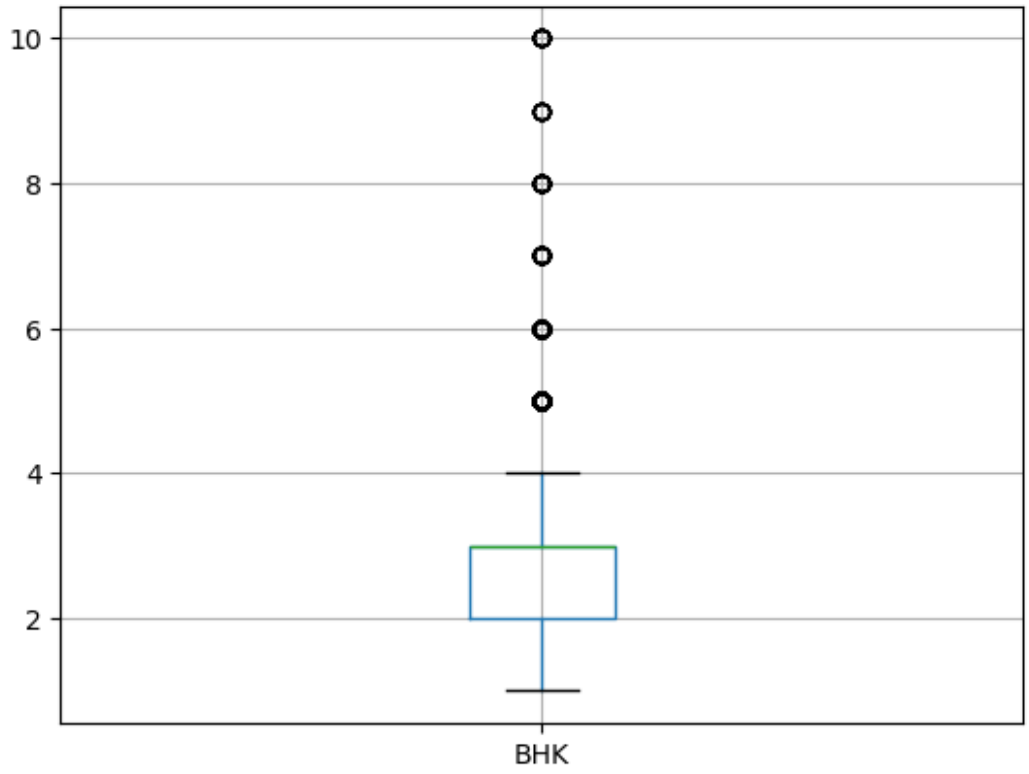
	Title	Amount(in rupees)	location	Floor	Transaction
0	1 BHK	4200000.0	thane	10 out of 11	Resale
1	2 BHK	9800000.0	thane	3 out of 22	Resale
2	2 BHK	14000000.0	thane	10 out of 29	Resale
3	1 BHK	2500000.0	thane	1 out of 3	Resale
4	2 BHK	16000000.0	thane	20 out of 42	Resale
...	...	...	...	...	...
187526	3 BHK	6300000.0	zirakpur	2 out of 4	New Property
187527	3 BHK	5500000.0	zirakpur	4 out of 6	Resale
187528	3 BHK	7600000.0	zirakpur	1 out of 3	Resale
187529	2 BHK	3000000.0	zirakpur	2 out of 2	Resale

```
187530  3 BHK         11800000.0  zirakpur   5 out of 13        Resale


             Furnishing        facing Bathroom Balcony  BHK
0          Unfurnished           NaN        1       2  1.0
1       Semi-Furnished          East        2     NaN  2.0
2          Unfurnished          East        2     NaN  2.0
3          Unfurnished           NaN        1       1  1.0
4          Unfurnished          West        2     NaN  2.0
...                ...           ...      ...     ...  ...
187526  Semi-Furnished          East        3       3  3.0
187527     Unfurnished  North - East        3     NaN  3.0
187528       Furnished          East        3       2  3.0
187529  Semi-Furnished           NaN        2     NaN  2.0
187530  Semi-Furnished  North - East        4       4  3.0

[187531 rows x 10 columns]
```

```
house['Amount(in rupees)'].nunique()
```

```
1559
```

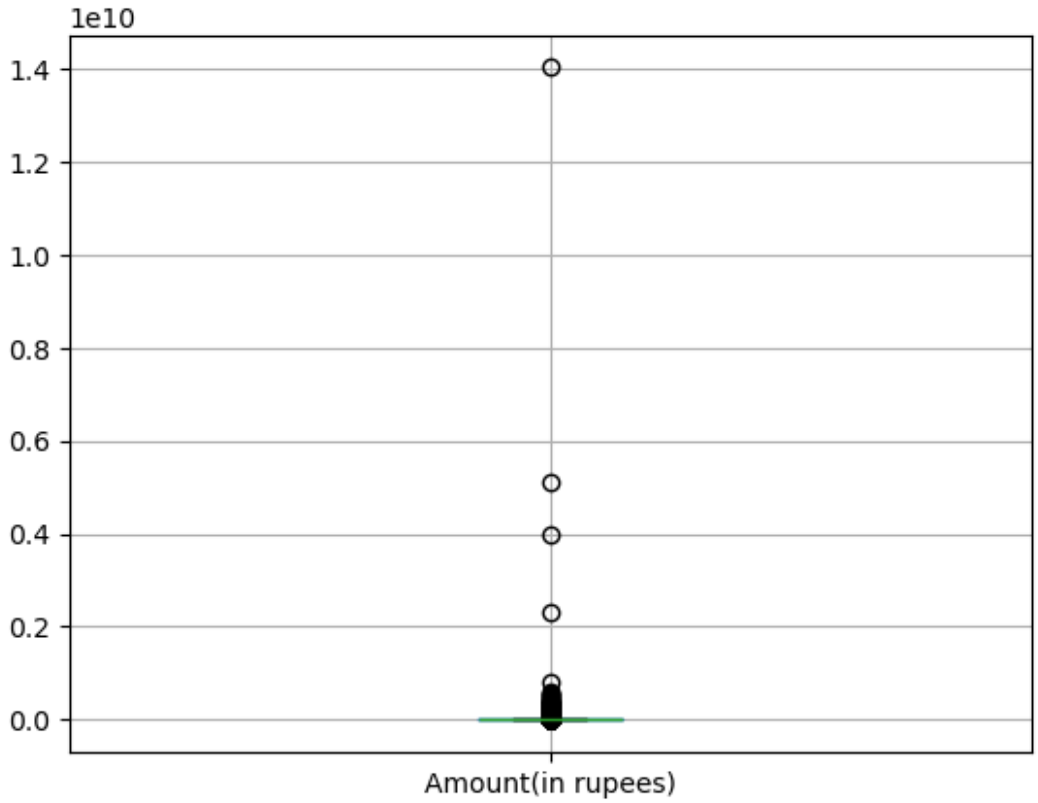
```
house.boxplot(column='BHK')
plt.show()
```



```
q1=house['BHK'].quantile(0.25)
q2=house['BHK'].quantile(0.75)
iqr=q2-q1
upper=q2+(1.5)*iqr
lower=q1-(1.5)*iqr
upper,lower
```

```
(4.5, 0.5)
```

```
mde=house['BHK'].median()
house['BHK']=house['BHK'].apply(lambda x:mde if x <lower or x >upper
else x )

house.boxplot(column='Amount(in rupees)')
plt.show()
```




```
house['Amount(in rupees)'].describe()
```

```
count    1.778470e+05
mean     1.198134e+07
std      3.943827e+07
min      1.000000e+05
25%      4.840000e+06
50%      7.800000e+06
75%      1.450000e+07
```

```
max      1.400300e+10
Name: Amount(in rupees), dtype: float64
```

```
q1=house['Amount(in rupees)'].quantile(0.25)
q2=house['Amount(in rupees)'].quantile(0.75)
iqr=q2-q1
upper=q2+(1.5)*iqr
lower=q1-(1.5)*iqr
upper,lower
```

```
(28990000.0, -9650000.0)
```

```
mde=house['Amount(in rupees)'].median()
house['Amount(in rupees)']=house['Amount(in rupees)'].apply(lambda
x:mde if x <lower or x >upper else x )
```

```
house.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 187531 entries, 0 to 187530
Data columns (total 10 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   Title              187531 non-null  object 
 1   Amount(in rupees)  177847 non-null  float64
 2   location           187531 non-null  object 
 3   Floor              180454 non-null  object 
 4   Transaction        187448 non-null  object 
 5   Furnishing         184634 non-null  object 
 6   facing             117298 non-null  object 
 7   Bathroom           186703 non-null  object 
 8   Balcony            138596 non-null  object 
 9   BHK                186578 non-null  float64
dtypes: float64(2), object(8)
memory usage: 14.3+ MB
```

```
house.isnull().mean()*100
```

```
Title                 0.000000
Amount(in rupees)     5.163946
location              0.000000
Floor                 3.773776
Transaction           0.044259
Furnishing            1.544811
facing               37.451408
Bathroom              0.441527
Balcony              26.094352
BHK                   0.508183
dtype: float64
```

```
house.columns
```

```
Index(['Title', 'Amount(in rupees)', 'location', 'Floor',
'Transaction',
       'Furnishing', 'facing', 'Bathroom', 'Balcony', 'BHK'],
      dtype='object')
```

```
1.## converting 'Amount(in rupees)' to numeric & filing Null values
with median
house['Amount(in rupees)'].fillna(house['Amount(in
rupees)'].median(),inplace=True)
house['Amount(in rupees)']=pd.to_numeric(house['Amount(in
rupees)'],errors='coerce')

##2.Extracting & converting to_numeric Floor
# Extract leading number from the floor string
house['Floor']=house['Floor'].astype(str)
house['Floor'] = house['Floor'].str.extract(r'(\d+)')
house['Floor'] = pd.to_numeric(house['Floor'], errors='coerce')
house['Floor'].fillna(house['Floor'].median(), inplace=True)

3.## filling Null values of Location with mode
house['location'].fillna(house['location'].mode()[0],inplace=True)

4.##Filling Null values of Transaction with mode
house['Transaction'].fillna(house['Transaction'].mode()
[0],inplace=True)

##5.Filling Null Values Of Furnishing with mode
house['Furnishing'].fillna(house['Furnishing'].mode()[0],inplace=True)

##7.Filling Null Values of Bathrooms with Mode & Converting this into
numeric
house['Bathroom'].fillna(house['Bathroom'].mode()[0],inplace=True)
house['Bathroom']=pd.to_numeric(house['Bathroom'],errors='coerce')

##8.Filing Null Values Of Balcony with Mode
house['Balcony'].fillna(house['Balcony'].mode()[0],inplace=True)
house['Balcony']=pd.to_numeric(house['Balcony'],errors='coerce')

##9.Fiulling Null VAlues of BHK with Mode
house['BHK'].fillna(house['BHK'].mode()[0],inplace=True)

house['facing'].fillna(house['facing'].mode()[0],inplace=True)
```

```
C:\Users\nisha\AppData\Local\Temp\ipykernel_2572\2565815676.py:2:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
```

```
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  house['Amount(in rupees)'].fillna(house['Amount(in
rupees)'].median(),inplace=True)
C:\Users\nisha\AppData\Local\Temp\ipykernel_2572\2565815676.py:10:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  house['Floor'].fillna(house['Floor'].median(), inplace=True)
C:\Users\nisha\AppData\Local\Temp\ipykernel_2572\2565815676.py:13:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  house['location'].fillna(house['location'].mode()[0],inplace=True)
C:\Users\nisha\AppData\Local\Temp\ipykernel_2572\2565815676.py:16:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  house['Transaction'].fillna(house['Transaction'].mode()
[0],inplace=True)
```

```
C:\Users\nisha\AppData\Local\Temp\ipykernel_2572\2565815676.py:19:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  house['Furnishing'].fillna(house['Furnishing'].mode()
[0],inplace=True)
C:\Users\nisha\AppData\Local\Temp\ipykernel_2572\2565815676.py:22:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  house['Bathroom'].fillna(house['Bathroom'].mode()[0],inplace=True)
C:\Users\nisha\AppData\Local\Temp\ipykernel_2572\2565815676.py:26:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  house['Balcony'].fillna(house['Balcony'].mode()[0],inplace=True)
C:\Users\nisha\AppData\Local\Temp\ipykernel_2572\2565815676.py:30:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.
```

```
For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  house['BHK'].fillna(house['BHK'].mode()[0],inplace=True)
C:\Users\nisha\AppData\Local\Temp\ipykernel_2572\2565815676.py:32:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  house['facing'].fillna(house['facing'].mode()[0],inplace=True)
```

```
house.drop(['Title'],inplace=True,axis=1)
```

```
house
```

```
        Amount(in rupees)  location  Floor   Transaction
Furnishing  \
0               4200000.0     thane   10.0        Resale
Unfurnished
1               9800000.0     thane    3.0        Resale  Semi-
Furnished
2              14000000.0     thane   10.0        Resale
Unfurnished
3               2500000.0     thane    1.0        Resale
Unfurnished
4              16000000.0     thane   20.0        Resale
Unfurnished
...                   ...       ...    ...           ...              .
..
187526          6300000.0  zirakpur    2.0  New Property  Semi-
Furnished
187527          5500000.0  zirakpur    4.0        Resale
Unfurnished
187528          7600000.0  zirakpur    1.0        Resale
Furnished
187529          3000000.0  zirakpur    2.0        Resale  Semi-
Furnished
187530         11800000.0  zirakpur    5.0        Resale  Semi-
Furnished
```

```
               facing  Bathroom  Balcony  BHK
0                East         1        2  1.0
1                East         2        2  2.0
2                East         2        2  2.0
3                East         1        1  1.0
4                West         2        2  2.0
...               ...       ...      ...  ...
187526           East         3        3  3.0
187527  North - East          3        2  3.0
187528           East         3        2  3.0
187529           East         2        2  2.0
187530  North - East          4        4  3.0

[187531 rows x 9 columns]
```

```
house.duplicated().sum()
```

```
129323
```

```
house.isnull().mean()*100
```

```
Amount(in rupees)    0.0
location             0.0
Floor                0.0
Transaction          0.0
Furnishing           0.0
facing               0.0
Bathroom             0.0
Balcony              0.0
BHK                  0.0
dtype: float64
```

```
house['Amount(in rupees)'].describe()
```

```
count    1.875310e+05
mean     9.023123e+06
std      5.955705e+06
min      1.000000e+05
25%      5.000000e+06
50%      7.800000e+06
75%      1.140000e+07
max      2.890000e+07
Name: Amount(in rupees), dtype: float64
```

```
loc=house['location'].value_counts()
cl=loc[loc >=1000].index
house['location']=house['location'].map(lambda x:x if x in cl else
'other' )
house
```

```
        Amount(in rupees)  location  Floor   Transaction
Furnishing  \
0               4200000.0     thane   10.0        Resale
Unfurnished
1               9800000.0     thane    3.0        Resale  Semi-
Furnished
2              14000000.0     thane   10.0        Resale
Unfurnished
3               2500000.0     thane    1.0        Resale
Unfurnished
4              16000000.0     thane   20.0        Resale
Unfurnished
...                   ...       ...    ...           ...              .
..
187526          6300000.0  zirakpur    2.0  New Property  Semi-
Furnished
187527          5500000.0  zirakpur    4.0        Resale
Unfurnished
187528          7600000.0  zirakpur    1.0        Resale
Furnished
187529          3000000.0  zirakpur    2.0        Resale  Semi-
Furnished
187530         11800000.0  zirakpur    5.0        Resale  Semi-
Furnished

               facing  Bathroom  Balcony  BHK
0                East         1        2  1.0
1                East         2        2  2.0
2                East         2        2  2.0
3                East         1        1  1.0
4                West         2        2  2.0
...               ...       ...      ...  ...
187526           East         3        3  3.0
187527  North - East         3        2  3.0
187528           East         3        2  3.0
187529           East         2        2  2.0
187530  North - East         4        4  3.0

[187531 rows x 9 columns]
```

```
house['location'].value_counts()
house['location'].nunique()
```

```
25
```

```
house['Balcony']=house['Balcony'].fillna(0).astype(int)
house['Balcony']=house['Balcony'].fillna(house['Balcony'].mean()).roun
d().astype(int)
house
```

```
        Amount(in rupees)  location  Floor   Transaction
Furnishing  \
0               4200000.0     thane   10.0        Resale
Unfurnished
1               9800000.0     thane    3.0        Resale  Semi-
Furnished
2              14000000.0     thane   10.0        Resale
Unfurnished
3               2500000.0     thane    1.0        Resale
Unfurnished
4              16000000.0     thane   20.0        Resale
Unfurnished
...                   ...       ...    ...           ...               .
..
187526          6300000.0  zirakpur    2.0  New Property  Semi-
Furnished
187527          5500000.0  zirakpur    4.0        Resale
Unfurnished
187528          7600000.0  zirakpur    1.0        Resale
Furnished
187529          3000000.0  zirakpur    2.0        Resale  Semi-
Furnished
187530         11800000.0  zirakpur    5.0        Resale  Semi-
Furnished

               facing  Bathroom  Balcony  BHK
0                East         1        2  1.0
1                East         2        2  2.0
2                East         2        2  2.0
3                East         1        1  1.0
4                West         2        2  2.0
...               ...       ...      ...  ...
187526           East         3        3  3.0
187527  North - East         3        2  3.0
187528           East         3        2  3.0
187529           East         2        2  2.0
187530  North - East         4        4  3.0

[187531 rows x 9 columns]
```

```
house['Bathroom']=house['Bathroom'].fillna(0).astype(int)
house['Bathroom']=house['Bathroom'].fillna(house['Bathroom'].mean()).r
ound().astype(int)
house
```

```
        Amount(in rupees)  location  Floor   Transaction
Furnishing  \
0               4200000.0     thane   10.0        Resale
Unfurnished
1               9800000.0     thane    3.0        Resale  Semi-
```

```
Furnished
2              14000000.0     thane   10.0        Resale
Unfurnished
3               2500000.0     thane    1.0        Resale
Unfurnished
4              16000000.0     thane   20.0        Resale
Unfurnished
...                   ...       ...    ...           ...               .
..
187526          6300000.0  zirakpur    2.0  New Property  Semi-
Furnished
187527          5500000.0  zirakpur    4.0        Resale
Unfurnished
187528          7600000.0  zirakpur    1.0        Resale
Furnished
187529          3000000.0  zirakpur    2.0        Resale  Semi-
Furnished
187530         11800000.0  zirakpur    5.0        Resale  Semi-
Furnished

               facing  Bathroom  Balcony  BHK
0               East         1        2  1.0
1               East         2        2  2.0
2               East         2        2  2.0
3               East         1        1  1.0
4               West         2        2  2.0
...              ...       ...      ...  ...
187526          East         3        3  3.0
187527  North - East         3        2  3.0
187528          East         3        2  3.0
187529          East         2        2  2.0
187530  North - East         4        4  3.0

[187531 rows x 9 columns]
```

```
house['Balcony'].unique()
```

```
array([2, 1, 3, 4, 0, 6, 5])
```

```
house['Balcony'] = house['Balcony'].map(lambda x: '6' if int(x) > 5
else str(x))
house['Balcony']=house['Balcony'].astype(int)
house
```

```
        Amount(in rupees)  location  Floor   Transaction
Furnishing  \
0               4200000.0     thane   10.0        Resale
Unfurnished
1               9800000.0     thane    3.0        Resale  Semi-
Furnished
```

```
2               14000000.0     thane   10.0        Resale
Unfurnished
3                2500000.0     thane    1.0        Resale
Unfurnished
4               16000000.0     thane   20.0        Resale
Unfurnished
...                    ...       ...    ...           ...               .
..
187526           6300000.0  zirakpur    2.0  New Property  Semi-
Furnished
187527           5500000.0  zirakpur    4.0        Resale
Unfurnished
187528           7600000.0  zirakpur    1.0        Resale
Furnished
187529           3000000.0  zirakpur    2.0        Resale  Semi-
Furnished
187530          11800000.0  zirakpur    5.0        Resale  Semi-
Furnished

               facing  Bathroom  Balcony  BHK
0                East         1        2  1.0
1                East         2        2  2.0
2                East         2        2  2.0
3                East         1        1  1.0
4                West         2        2  2.0
...               ...       ...      ...  ...
187526           East         3        3  3.0
187527  North - East          3        2  3.0
187528           East         3        2  3.0
187529           East         2        2  2.0
187530  North - East          4        4  3.0

[187531 rows x 9 columns]
```

```
house.isnull().sum()
```

```
Amount(in rupees)    0
location             0
Floor                0
Transaction          0
Furnishing           0
facing               0
Bathroom             0
Balcony              0
BHK                  0
dtype: int64
```

```
x=house[['Floor','location',
       'Transaction', 'Furnishing', 'Bathroom', 'Balcony','BHK']]
y=house['Amount(in rupees)']
```

```
house['location'].unique()
```

```
array(['thane', 'other', 'mumbai', 'ahmedabad', 'bangalore',
'chennai',
       'gurgaon', 'hyderabad', 'jaipur', 'kolkata', 'new-delhi',
'noida',
       'pune', 'bhiwadi', 'chandigarh', 'faridabad', 'goa',
       'greater-noida', 'kochi', 'mohali', 'ranchi', 'surat',
'vadodara',
       'visakhapatnam', 'zirakpur'], dtype=object)
```

```
x=x['location_encoded'] =
house['location'].map(house.groupby('location')['Amount(in
rupees)'].mean())
x
```

```
---------------------------------------------------------------------
-----
RecursionError                            Traceback (most recent call
last)
File ~\anaconda3\Lib\site-packages\IPython\core\formatters.py:711, in
PlainTextFormatter.__call__(self, obj)
    704 stream = StringIO()
    705 printer = pretty.RepresentationPrinter(stream, self.verbose,
    706     self.max_width, self.newline,
    707     max_seq_length=self.max_seq_length,
    708     singleton_pprinters=self.singleton_printers,
    709     type_pprinters=self.type_printers,
    710     deferred_pprinters=self.deferred_printers)
--> 711 printer.pretty(obj)
    712 printer.flush()
    713 return stream.getvalue()

File ~\anaconda3\Lib\site-packages\IPython\lib\pretty.py:419, in
RepresentationPrinter.pretty(self, obj)
    408                         return meth(obj, self, cycle)
    409                 if (
    410                     cls is not object
    411                     # check if cls defines __repr__
   (...)
    417                     and callable(_safe_getattr(cls,
"__repr__", None))
    418                 ):
--> 419                     return _repr_pprint(obj, self, cycle)
    421     return _default_pprint(obj, self, cycle)
    422 finally:

File ~\anaconda3\Lib\site-packages\IPython\lib\pretty.py:787, in
_repr_pprint(obj, p, cycle)
    785 """A pprint that just redirects to the normal repr
```

```
function."""
    786 # Find newlines and replace them with p.break_()
--> 787 output = repr(obj)
    788 lines = output.splitlines()
    789 with p.group():

File ~\anaconda3\Lib\site-packages\pandas\core\series.py:1784, in
Series.__repr__(self)
   1782 # pylint: disable=invalid-repr-returned
   1783 repr_params = fmt.get_series_repr_params()
-> 1784 return self.to_string(**repr_params)

File ~\anaconda3\Lib\site-packages\pandas\core\series.py:1883, in
Series.to_string(self, buf, na_rep, float_format, header, index,
length, dtype, name, max_rows, min_rows)
   1831 """
   1832 Render a string representation of the Series.
   1833
   (...)
   1869 '0    1\\n1    2\\n2    3'
   1870 """
   1871 formatter = fmt.SeriesFormatter(
   1872     self,
   1873     name=name,
   (...)
   1881     max_rows=max_rows,
   1882 )
-> 1883 result = formatter.to_string()
   1885 # catch contract violations
   1886 if not isinstance(result, str):

File ~\anaconda3\Lib\site-packages\pandas\io\formats\format.py:320, in
SeriesFormatter.to_string(self)
    318 else:
    319     fmt_index = index._format_flat(include_name=True)
--> 320 fmt_values = self._get_formatted_values()
    322 if self.is_truncated_vertically:
    323     n_header_rows = 0

File ~\anaconda3\Lib\site-packages\pandas\io\formats\format.py:297, in
SeriesFormatter._get_formatted_values(self)
    296 def _get_formatted_values(self) -> list[str]:
--> 297     return format_array(
    298         self.tr_series._values,
    299         None,
    300         float_format=self.float_format,
    301         na_rep=self.na_rep,
    302         leading_space=self.index,
    303     )
```

```
File ~\anaconda3\Lib\site-packages\pandas\io\formats\format.py:1161,
in format_array(values, formatter, float_format, na_rep, digits,
space, justify, decimal, leading_space, quoting, fallback_formatter)
   1145     digits = get_option("display.precision")
   1147 fmt_obj = fmt_klass(
   1148     values,
   1149     digits=digits,
   (...)
   1158     fallback_formatter=fallback_formatter,
   1159 )
-> 1161 return fmt_obj.get_result()

File ~\anaconda3\Lib\site-packages\pandas\io\formats\format.py:1194,
in _GenericArrayFormatter.get_result(self)
   1193 def get_result(self) -> list[str]:
-> 1194     fmt_values = self._format_strings()
   1195     return _make_fixed_width(fmt_values, self.justify)

File ~\anaconda3\Lib\site-packages\pandas\io\formats\format.py:1259,
in _GenericArrayFormatter._format_strings(self)
   1257 for i, v in enumerate(vals):
   1258     if (not is_float_type[i] or self.formatter is not None)
and leading_space:
-> 1259         fmt_values.append(f" {_format(v)}")
   1260     elif is_float_type[i]:
   1261         fmt_values.append(float_format(v))

File ~\anaconda3\Lib\site-packages\pandas\io\formats\format.py:1234,
in _GenericArrayFormatter._format_strings.<locals>._format(x)
   1232     return self.na_rep
   1233 elif isinstance(x, PandasObject):
-> 1234     return str(x)
   1235 elif isinstance(x, StringDtype):
   1236     return repr(x)

File ~\anaconda3\Lib\site-packages\pandas\core\series.py:1784, in
Series.__repr__(self)
   1782 # pylint: disable=invalid-repr-returned
   1783 repr_params = fmt.get_series_repr_params()
-> 1784 return self.to_string(**repr_params)

File ~\anaconda3\Lib\site-packages\pandas\core\series.py:1883, in
Series.to_string(self, buf, na_rep, float_format, header, index,
length, dtype, name, max_rows, min_rows)
   1831 """
   1832 Render a string representation of the Series.
   1833
   (...)
   1869 '0    1\\n1    2\\n2    3'
   1870 """
```

```
   1871 formatter = fmt.SeriesFormatter(
   1872     self,
   1873     name=name,
   (...)
   1881     max_rows=max_rows,
   1882 )
-> 1883 result = formatter.to_string()
   1885 # catch contract violations
   1886 if not isinstance(result, str):

File ~\anaconda3\Lib\site-packages\pandas\io\formats\format.py:320, in
SeriesFormatter.to_string(self)
    318 else:
    319     fmt_index = index._format_flat(include_name=True)
--> 320 fmt_values = self._get_formatted_values()
    322 if self.is_truncated_vertically:
    323     n_header_rows = 0

File ~\anaconda3\Lib\site-packages\pandas\io\formats\format.py:297, in
SeriesFormatter._get_formatted_values(self)
    296 def _get_formatted_values(self) -> list[str]:
--> 297     return format_array(
    298         self.tr_series._values,
    299         None,
    300         float_format=self.float_format,
    301         na_rep=self.na_rep,
    302         leading_space=self.index,
    303     )

File ~\anaconda3\Lib\site-packages\pandas\io\formats\format.py:1161,
in format_array(values, formatter, float_format, na_rep, digits,
space, justify, decimal, leading_space, quoting, fallback_formatter)
   1145     digits = get_option("display.precision")
   1147 fmt_obj = fmt_klass(
   1148     values,
   1149     digits=digits,
   (...)
   1158     fallback_formatter=fallback_formatter,
   1159 )
-> 1161 return fmt_obj.get_result()

File ~\anaconda3\Lib\site-packages\pandas\io\formats\format.py:1194,
in _GenericArrayFormatter.get_result(self)
   1193 def get_result(self) -> list[str]:
-> 1194     fmt_values = self._format_strings()
   1195     return _make_fixed_width(fmt_values, self.justify)

File ~\anaconda3\Lib\site-packages\pandas\io\formats\format.py:1259,
in _GenericArrayFormatter._format_strings(self)
   1257 for i, v in enumerate(vals):
```

```
   1258     if (not is_float_type[i] or self.formatter is not None)
and leading_space:
-> 1259         fmt_values.append(f" {_format(v)}")
   1260     elif is_float_type[i]:
   1261         fmt_values.append(float_format(v))

File ~\anaconda3\Lib\site-packages\pandas\io\formats\format.py:1234,
in _GenericArrayFormatter._format_strings.<locals>._format(x)
   1232     return self.na_rep
   1233 elif isinstance(x, PandasObject):
-> 1234     return str(x)
   1235 elif isinstance(x, StringDtype):
   1236     return repr(x)

    [... skipping similar frames: Series.__repr__ at line 1784 (368
times), _GenericArrayFormatter._format_strings.<locals>._format at
line 1234 (367 times), _GenericArrayFormatter._format_strings at line
1259 (367 times), SeriesFormatter._get_formatted_values at line 297
(367 times), format_array at line 1161 (367 times),
_GenericArrayFormatter.get_result at line 1194 (367 times),
Series.to_string at line 1883 (367 times), SeriesFormatter.to_string
at line 320 (367 times)]

File ~\anaconda3\Lib\site-packages\pandas\core\series.py:1883, in
Series.to_string(self, buf, na_rep, float_format, header, index,
length, dtype, name, max_rows, min_rows)
   1831 """
   1832 Render a string representation of the Series.
   1833
   (...)
   1869 '0    1\\n1    2\\n2    3'
   1870 """
   1871 formatter = fmt.SeriesFormatter(
   1872     self,
   1873     name=name,
   (...)
   1881     max_rows=max_rows,
   1882 )
-> 1883 result = formatter.to_string()
   1885 # catch contract violations
   1886 if not isinstance(result, str):

File ~\anaconda3\Lib\site-packages\pandas\io\formats\format.py:320, in
SeriesFormatter.to_string(self)
    318 else:
    319     fmt_index = index._format_flat(include_name=True)
--> 320 fmt_values = self._get_formatted_values()
    322 if self.is_truncated_vertically:
    323     n_header_rows = 0
```

```
File ~\anaconda3\Lib\site-packages\pandas\io\formats\format.py:297, in
SeriesFormatter._get_formatted_values(self)
    296 def _get_formatted_values(self) -> list[str]:
--> 297     return format_array(
    298         self.tr_series._values,
    299         None,
    300         float_format=self.float_format,
    301         na_rep=self.na_rep,
    302         leading_space=self.index,
    303     )

File ~\anaconda3\Lib\site-packages\pandas\io\formats\format.py:1161,
in format_array(values, formatter, float_format, na_rep, digits,
space, justify, decimal, leading_space, quoting, fallback_formatter)
   1145     digits = get_option("display.precision")
   1147 fmt_obj = fmt_klass(
   1148     values,
   1149     digits=digits,
   (...)
   1158     fallback_formatter=fallback_formatter,
   1159 )
-> 1161 return fmt_obj.get_result()

File ~\anaconda3\Lib\site-packages\pandas\io\formats\format.py:1194,
in _GenericArrayFormatter.get_result(self)
   1193 def get_result(self) -> list[str]:
-> 1194     fmt_values = self._format_strings()
   1195     return _make_fixed_width(fmt_values, self.justify)

File ~\anaconda3\Lib\site-packages\pandas\io\formats\format.py:1259,
in _GenericArrayFormatter._format_strings(self)
   1257 for i, v in enumerate(vals):
   1258     if (not is_float_type[i] or self.formatter is not None)
and leading_space:
-> 1259         fmt_values.append(f" {_format(v)}")
   1260     elif is_float_type[i]:
   1261         fmt_values.append(float_format(v))

File ~\anaconda3\Lib\site-packages\pandas\io\formats\format.py:1234,
in _GenericArrayFormatter._format_strings.<locals>._format(x)
   1232     return self.na_rep
   1233 elif isinstance(x, PandasObject):
-> 1234     return str(x)
   1235 elif isinstance(x, StringDtype):
   1236     return repr(x)

File ~\anaconda3\Lib\site-packages\pandas\core\series.py:1784, in
Series.__repr__(self)
   1782 # pylint: disable=invalid-repr-returned
   1783 repr_params = fmt.get_series_repr_params()
```

```
-> 1784 return self.to_string(**repr_params)

File ~\anaconda3\Lib\site-packages\pandas\core\series.py:1871, in
Series.to_string(self, buf, na_rep, float_format, header, index,
length, dtype, name, max_rows, min_rows)
   1818 def to_string(
   1819     self,
   1820     buf: FilePath | WriteBuffer[str] | None = None,
   (...)
   1829     min_rows: int | None = None,
   1830 ) -> str | None:
   1831     """
   1832     Render a string representation of the Series.
   1833
   (...)
   1869     '0    1\\n1    2\\n2    3'
   1870     """
-> 1871     formatter = fmt.SeriesFormatter(
   1872         self,
   1873         name=name,
   1874         length=length,
   1875         header=header,
   1876         index=index,
   1877         dtype=dtype,
   1878         na_rep=na_rep,
   1879         float_format=float_format,
   1880         min_rows=min_rows,
   1881         max_rows=max_rows,
   1882     )
   1883     result = formatter.to_string()
   1885     # catch contract violations

File ~\anaconda3\Lib\site-packages\pandas\io\formats\format.py:223, in
SeriesFormatter.__init__(self, series, length, header, index, na_rep,
name, float_format, dtype, max_rows, min_rows)
    221 self.float_format = float_format
    222 self.dtype = dtype
--> 223 self.adj = printing.get_adjustment()
    225 self._chk_truncate()

File ~\anaconda3\Lib\site-packages\pandas\io\formats\printing.py:572,
in get_adjustment()
    570     return _EastAsianTextAdjustment()
    571 else:
--> 572     return _TextAdjustment()

File ~\anaconda3\Lib\site-packages\pandas\io\formats\printing.py:508,
in _TextAdjustment.__init__(self)
    507 def __init__(self) -> None:
--> 508     self.encoding = get_option("display.encoding")
```

```
File ~\anaconda3\Lib\site-packages\pandas\_config\config.py:274, in
CallableDynamicDoc.__call__(self, *args, **kwds)
    273 def __call__(self, *args, **kwds) -> T:
--> 274     return self.__func__(*args, **kwds)

File ~\anaconda3\Lib\site-packages\pandas\_config\config.py:146, in
_get_option(pat, silent)
    145 def _get_option(pat: str, silent: bool = False) -> Any:
--> 146     key = _get_single_key(pat, silent)
    148     # walk the nested dict
    149     root, k = _get_root(key)

File ~\anaconda3\Lib\site-packages\pandas\_config\config.py:138, in
_get_single_key(pat, silent)
    135 key = keys[0]
    137 if not silent:
--> 138     _warn_if_deprecated(key)
    140 key = _translate_key(key)
    142 return key

File ~\anaconda3\Lib\site-packages\pandas\_config\config.py:696, in
_warn_if_deprecated(key)
    688 def _warn_if_deprecated(key: str) -> bool:
    689     """
    690     Checks if `key` is a deprecated option and if so, prints a
warning.
    691
   (...)
    694     bool - True if `key` is deprecated, False otherwise.
    695     """
--> 696     d = _get_deprecated_option(key)
    697     if d:
    698         if d.msg:

RecursionError: maximum recursion depth exceeded
```

```
x=house.drop('location',axis=1,inplace=False)
x
```

```
        Amount(in rupees)  Floor   Transaction      Furnishing
facing  \
0               4200000.0   10.0        Resale     Unfurnished
East
1               9800000.0    3.0        Resale  Semi-Furnished
East
2              14000000.0   10.0        Resale     Unfurnished
East
3               2500000.0    1.0        Resale     Unfurnished
East
```

```
4              16000000.0   20.0        Resale     Unfurnished
West
...                   ...    ...           ...             ...
...
187526          6300000.0    2.0  New Property  Semi-Furnished
East
187527          5500000.0    4.0        Resale     Unfurnished  North
- East
187528          7600000.0    1.0        Resale       Furnished
East
187529          3000000.0    2.0        Resale  Semi-Furnished
East
187530         11800000.0    5.0        Resale  Semi-Furnished  North
- East

        Bathroom  Balcony  BHK
0              1        2  1.0
1              2        2  2.0
2              2        2  2.0
3              1        1  1.0
4              2        2  2.0
...          ...      ...  ...
187526         3        3  3.0
187527         3        2  3.0
187528         3        2  3.0
187529         2        2  2.0
187530         4        4  3.0

[187531 rows x 8 columns]
```

```
house['location']=x['location']
```

```
---------------------------------------------------------------------
-----
KeyError                                  Traceback (most recent call
last)
File ~\anaconda3\Lib\site-packages\pandas\core\indexes\base.py:3805,
in Index.get_loc(self, key)
   3804 try:
-> 3805     return self._engine.get_loc(casted_key)
   3806 except KeyError as err:

File index.pyx:167, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:196, in pandas._libs.index.IndexEngine.get_loc()

File pandas\\_libs\\hashtable_class_helper.pxi:7081, in
pandas._libs.hashtable.PyObjectHashTable.get_item()

File pandas\\_libs\\hashtable_class_helper.pxi:7089, in
```

```
pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'location'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call
last)
Cell In[104], line 1
----> 1 house['location']=x['location']

File ~\anaconda3\Lib\site-packages\pandas\core\series.py:1121, in
Series.__getitem__(self, key)
   1118     return self._values[key]
   1120 elif key_is_scalar:
-> 1121     return self._get_value(key)
   1123 # Convert generator to list before going through hashable part
   1124 # (We will iterate through the generator there to check for
slices)
   1125 if is_iterator(key):

File ~\anaconda3\Lib\site-packages\pandas\core\series.py:1237, in
Series._get_value(self, label, takeable)
   1234     return self._values[label]
   1236 # Similar to Index.get_value, but we do not fall back to
positional
-> 1237 loc = self.index.get_loc(label)
   1239 if is_integer(loc):
   1240     return self._values[loc]

File ~\anaconda3\Lib\site-packages\pandas\core\indexes\base.py:3812,
in Index.get_loc(self, key)
   3807     if isinstance(casted_key, slice) or (
   3808         isinstance(casted_key, abc.Iterable)
   3809         and any(isinstance(x, slice) for x in casted_key)
   3810     ):
   3811         raise InvalidIndexError(key)
-> 3812     raise KeyError(key) from err
   3813 except TypeError:
   3814     # If we have a listlike key, _check_indexing_error will
raise
   3815     #  InvalidIndexError. Otherwise we fall through and re-
raise
   3816     #  the TypeError.
   3817     self._check_indexing_error(key)

KeyError: 'location'
```

```
cat=x.select_dtypes(exclude='number')
num=x.select_dtypes(include='number')

cat_x=pd.get_dummies(cat)
cat_x
```

```
        Transaction_New Property  Transaction_Other
Transaction_Rent/Lease  \
0                          False              False
False
1                          False              False
False
2                          False              False
False
3                          False              False
False
4                          False              False
False
...                          ...                ...
...
187526                      True              False
False
187527                     False              False
False
187528                     False              False
False
187529                     False              False
False
187530                     False              False
False

        Transaction_Resale  Furnishing_Furnished  Furnishing_Semi-
Furnished  \
0                     True                 False
False
1                     True                 False
True
2                     True                 False
False
3                     True                 False
False
4                     True                 False
False
...                    ...                   ...
...
187526               False                 False
True
187527                True                 False
False
187528                True                  True
```

```
False
187529                 True                  False
True
187530                 True                  False
True

        Furnishing_Unfurnished  facing_East  facing_North  \
0                         True         True         False
1                        False         True         False
2                         True         True         False
3                         True         True         False
4                         True        False         False
...                        ...          ...           ...
187526                   False         True         False
187527                    True        False         False
187528                   False         True         False
187529                   False         True         False
187530                   False        False         False

        facing_North - East  facing_North - West  facing_South  \
0                     False                False         False
1                     False                False         False
2                     False                False         False
3                     False                False         False
4                     False                False         False
...                     ...                  ...           ...
187526                False                False         False
187527                 True                False         False
187528                False                False         False
187529                False                False         False
187530                 True                False         False

        facing_South - East  facing_South -West  facing_West
0                     False               False        False
1                     False               False        False
2                     False               False        False
3                     False               False        False
4                     False               False         True
...                     ...                 ...          ...
187526                False               False        False
187527                False               False        False
187528                False               False        False
187529                False               False        False
187530                False               False        False

[187531 rows x 15 columns]

cat_x
```

```
        Transaction_New Property  Transaction_Other
Transaction_Rent/Lease  \
0                           False               False
False
1                           False               False
False
2                           False               False
False
3                           False               False
False
4                           False               False
False
...                           ...                 ...
...
187526                       True               False
False
187527                      False               False
False
187528                      False               False
False
187529                      False               False
False
187530                      False               False
False

        Transaction_Resale  Furnishing_Furnished  Furnishing_Semi-
Furnished  \
0                     True                  False
False
1                     True                  False
True
2                     True                  False
False
3                     True                  False
False
4                     True                  False
False
...                    ...                    ...
...
187526               False                  False
True
187527                True                  False
False
187528                True                   True
False
187529                True                  False
True
187530                True                  False
True
```

```
        Furnishing_Unfurnished  facing_East  facing_North  \
0                         True         True         False
1                        False         True         False
2                         True         True         False
3                         True         True         False
4                         True        False         False
...                        ...          ...           ...
187526                   False         True         False
187527                    True        False         False
187528                   False         True         False
187529                   False         True         False
187530                   False        False         False

        facing_North - East  facing_North - West  facing_South  \
0                     False                False         False
1                     False                False         False
2                     False                False         False
3                     False                False         False
4                     False                False         False
...                     ...                  ...           ...
187526                False                False         False
187527                 True                False         False
187528                False                False         False
187529                False                False         False
187530                 True                False         False

        facing_South - East  facing_South -West  facing_West
0                     False               False        False
1                     False               False        False
2                     False               False        False
3                     False               False        False
4                     False               False         True
...                     ...                 ...          ...
187526                False               False        False
187527                False               False        False
187528                False               False        False
187529                False               False        False
187530                False               False        False

[187531 rows x 15 columns]

cat_x.describe()

       Transaction_New Property Transaction_Other
Transaction_Rent/Lease  \
count                     187531            187531
187531
unique                         2                 2
2
```

```
top                         False              False
False
freq                       144966             186822
187529

       Transaction_Resale Furnishing_Furnished Furnishing_Semi-
Furnished  \
count              187531               187531
187531
unique                  2                    2
2
top                  True                 False
False
freq               144255               167369
96316

       Furnishing_Unfurnished facing_East facing_North facing_North -
East  \
count                  187531      187531       187531
187531
unique                      2           2            2
2
top                     False        True         False
False
freq                   111377      124974       170998
163311

       facing_North - West facing_South facing_South - East  \
count                187531       187531               187531
unique                    2            2                    2
top                   False        False                False
freq                 183688       182837               184909

       facing_South -West facing_West
count               187531      187531
unique                   2           2
top                  False       False
freq                185460      178957

x=pd.concat([cat_x,num],axis=1)
x

        Transaction_New Property  Transaction_Other
Transaction_Rent/Lease  \
0                           False               False
False
1                           False               False
False
2                           False               False
False
```

```
3                           False              False
False
4                           False              False
False
...                           ...                ...
...
187526                       True              False
False
187527                      False              False
False
187528                      False              False
False
187529                      False              False
False
187530                      False              False
False

        Transaction_Resale  Furnishing_Furnished  Furnishing_Semi-
Furnished  \
0                     True                 False
False
1                     True                 False
True
2                     True                 False
False
3                     True                 False
False
4                     True                 False
False
...                    ...                   ...
...
187526               False                 False
True
187527                True                 False
False
187528                True                  True
False
187529                True                 False
True
187530                True                 False
True

        Furnishing_Unfurnished  facing_East  facing_North  \
0                         True         True         False
1                        False         True         False
2                         True         True         False
3                         True         True         False
4                         True        False         False
...                        ...          ...           ...
```

```
187526                   False         True         False
187527                    True        False         False
187528                   False         True         False
187529                   False         True         False
187530                   False        False         False

        facing_North - East  facing_North - West  facing_South  \
0                     False                False         False
1                     False                False         False
2                     False                False         False
3                     False                False         False
4                     False                False         False
...                     ...                  ...           ...
187526                False                False         False
187527                 True                False         False
187528                False                False         False
187529                False                False         False
187530                 True                False         False

        facing_South - East  facing_South -West  facing_West  \
0                     False               False        False
1                     False               False        False
2                     False               False        False
3                     False               False        False
4                     False               False         True
...                     ...                 ...          ...
187526                False               False        False
187527                False               False        False
187528                False               False        False
187529                False               False        False
187530                False               False        False

        Amount(in rupees)  Floor  Bathroom  Balcony  BHK
0               4200000.0   10.0         1        2  1.0
1               9800000.0    3.0         2        2  2.0
2              14000000.0   10.0         2        2  2.0
3               2500000.0    1.0         1        1  1.0
4              16000000.0   20.0         2        2  2.0
...                   ...    ...       ...      ...  ...
187526          6300000.0    2.0         3        3  3.0
187527          5500000.0    4.0         3        2  3.0
187528          7600000.0    1.0         3        2  3.0
187529          3000000.0    2.0         2        2  2.0
187530         11800000.0    5.0         4        4  3.0

[187531 rows x 20 columns]
```

```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.1,rando
m_state=7)

y_train
```

```
9680       5100000.0
169371    15500000.0
54600     12500000.0
105030     2500000.0
101109     3100000.0
             ...
66455      3500000.0
53459      8500000.0
10742      5000000.0
49689      6900000.0
61615      7800000.0
Name: Amount(in rupees), Length: 168777, dtype: float64
```

```
print(x_train.shape,x_test.shape,y_train.shape,y_test.shape)
```

```
(168777, 20) (18754, 20) (168777,) (18754,)
```

model training

```
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.metrics import mean_squared_error,r2_score

model=LinearRegression()
model.fit(x_train,y_train)
```

```
LinearRegression()
```

model testing

```
test_pred=model.predict(x_test)
test_pred
```

```
array([12100000.00000015,  7200000.00000045, 14999999.9999998 , ...,
        5100000.0000002 ,  7799999.99999993,  6400000.00000052])
```

```
y_test
```

```
94353     12100000.0
20969      7200000.0
90081     15000000.0
160714     4420000.0
80704      7800000.0
             ...
33483      7800000.0
```

```
3148       6500000.0
87503      5100000.0
20526      7800000.0
25629      6400000.0
Name: Amount(in rupees), Length: 18754, dtype: float64
```

```
train_pred=model.predict(x_train)
train_pred
```

```
array([ 5100000.00000017, 15499999.99999926, 12499999.99999952, ...,
        5000000.0000002 ,  6900000.00000039,  7800000.0000006 ])
```

```
(y_train)
```

```
9680       5100000.0
169371    15500000.0
54600     12500000.0
105030     2500000.0
101109     3100000.0
             ...
66455      3500000.0
53459      8500000.0
10742      5000000.0
49689      6900000.0
61615      7800000.0
Name: Amount(in rupees), Length: 168777, dtype: float64
```

```
print('Test Mean squared
error :',mean_squared_error(y_test,test_pred))
```

```
Test Mean squared error : 2.7900281376731636e-13
```

```
print('Train Mean squared
error :',mean_squared_error(y_train,train_pred))
```

```
Train Mean squared error : 2.805622538822866e-13
```

```
print('Test R2 error :',r2_score(y_test,test_pred))
```

```
Test R2 error : 1.0
```

```
print('Train R2 error :',r2_score(y_train,train_pred))
```

```
Train R2 error : 1.0
```

Ridge Model

```
ridge_model=Ridge(alpha=0.1)
ridge_model.fit(x_train,y_train)
```

```
C:\Users\nisha\anaconda3\Lib\site-packages\sklearn\linear_model\
_ridge.py:216: LinAlgWarning: Ill-conditioned matrix (rcond=1.66969e-
```

```
20): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

```
Ridge(alpha=0.1)
```

```
ridge_test=ridge_model.predict(x_test)
ridge_train=ridge_model.predict(x_train)
```

```
print('Ridge Model of MSE of
train :',mean_squared_error(y_train,ridge_train))
```

```
Ridge Model of MSE of train : 2.114008490639501e-11
```

```
print('Ridge Model of MSE of
test :',mean_squared_error(y_test,ridge_test))
```

```
Ridge Model of MSE of test : 1.4487198345135523e-11
```

```
print('Ridge model of R2 of train :',r2_score(y_train,ridge_train))
```

```
Ridge model of R2 of train : 1.0
```

```
print('Ridge Model of R2 of test :',r2_score(y_test,ridge_test))
```

```
Ridge Model of R2 of test : 1.0
```

Lasso model

```
Lasso_model=Lasso(alpha=0.1)
Lasso_model.fit(x_train,y_train)
```

```
Lasso(alpha=0.1)
```

```
Lasso_test=Lasso_model.predict(x_test)
Lasso_train=Lasso_model.predict(x_train)
```

```
print('Lasso Model of MSE of
train :',mean_squared_error(y_train,Lasso_train))
```

```
Lasso Model of MSE of train : 1.3580493466373543e-14
```

```
print('Lasso Model of MSE of
test:',mean_squared_error(y_test,Lasso_test))
```

```
Lasso Model of MSE of test: 1.3522017256084456e-14
```

```
print('Lasso model of R2 of train :',r2_score(y_train,Lasso_train))
```

```
Lasso model of R2 of train : 1.0
```

```
print('Lasso model of R2 of test :',r2_score(y_test,Lasso_test))
```

```
Lasso model of R2 of test : 1.0
```

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
x_scaled

from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV

alphas = [0.0001, 0.001, 0.01, 0.1, 1, 10]
grid = GridSearchCV(Lasso(), param_grid={'alpha': alphas}, cv=5)
grid.fit(x_train, y_train)

print("Best alpha:", grid.best_params_)
print("Train R²:", grid.score(x_train, y_train))
print("Test R²:", grid.score(x_test, y_test))
```