bengaluru-linearr

October 3, 2024

LinearRegression

Linear regression is a supervised learning algorithm used in machine learning to model the relationship between a dependent variable (target) and one or more independent variables (features). The goal is to find the best-fitting line (or hyperplane in higher dimensions) that minimizes the difference between the predicted and actual values of the target variable.

Key Concepts: Simple Linear Regression: Involves one independent variable and one dependent variable. The model fits a straight line (y = mx + c) where:

y is the dependent variable (prediction), x is the independent variable (input), m is the slope (weights in machine learning), c is the intercept (bias term).

[385]:
```
# load the essential libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

[142]:
```
df=pd.read_csv(r"C:\Users\NF\Downloads\Bengaluru_House_Data.csv")
```

[144]:
```
df
```

[144]:

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2.0	1.0	39.07
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5.0	3.0	120.00
2	Built-up Area	Ready To Move	Uttarahalli						
3	Super built-up Area	Ready To Move	Lingadheeranahalli						
4	Super built-up Area	Ready To Move	Kothanur						
…	…	…	…						
13315	Built-up Area	Ready To Move	Whitefield						
13316	Super built-up Area	Ready To Move	Richards Town						
13317	Built-up Area	Ready To Move	Raja Rajeshwari Nagar						
13318	Super built-up Area	18-Jun	Padmanabhanagar						
13319	Super built-up Area	Ready To Move	Doddathoguru						

```
2          3 BHK   @@@@@@       1440   2.0      3.0   62.00
3          3 BHK  Soiewre       1521   3.0      1.0   95.00
4          2 BHK      NaN       1200   2.0      1.0   51.00
…          …      …         …   …      …      …
13315  5 Bedroom  ArsiaEx       3453   4.0      0.0  231.00
13316      4 BHK      NaN       3600   5.0      NaN  400.00
13317      2 BHK  Mahla T       1141   2.0      1.0   60.00
13318      4 BHK  SollyCl       4689   4.0      1.0  488.00
13319      1 BHK      NaN        550   1.0      1.0   17.00

[13320 rows x 9 columns]
```

[146]:
```
df.head()
```

[146]:
```
              area_type   availability                  location       size  \
0  Super built-up  Area         19-Dec  Electronic City Phase II      2 BHK
1            Plot  Area  Ready To Move          Chikka Tirupathi  4 Bedroom
2        Built-up  Area  Ready To Move               Uttarahalli      3 BHK
3  Super built-up  Area  Ready To Move        Lingadheeranahalli      3 BHK
4  Super built-up  Area  Ready To Move                  Kothanur      2 BHK

   society total_sqft  bath  balcony   price
0  Coomee        1056   2.0      1.0   39.07
1  Theanmp       2600   5.0      3.0  120.00
2   @@@@@@       1440   2.0      3.0   62.00
3  Soiewre       1521   3.0      1.0   95.00
4      NaN       1200   2.0      1.0   51.00
```

[148]:
```
df.tail()
```

[148]:
```
                 area_type   availability                  location       size  \
13315        Built-up  Area  Ready To Move             Whitefield  5 Bedroom
13316  Super built-up  Area  Ready To Move           Richards Town      4 BHK
13317        Built-up  Area  Ready To Move  Raja Rajeshwari Nagar      2 BHK
13318  Super built-up  Area         18-Jun        Padmanabhanagar      4 BHK
13319  Super built-up  Area  Ready To Move           Doddathoguru      1 BHK

       society total_sqft  bath  balcony  price
13315  ArsiaEx       3453   4.0      0.0  231.0
13316      NaN       3600   5.0      NaN  400.0
13317  Mahla T       1141   2.0      1.0   60.0
13318  SollyCl       4689   4.0      1.0  488.0
13319      NaN        550   1.0      1.0   17.0
```

[150]:
```
df.sample()
```

[150]:
```
                area_type availability         location   size  society  \
6909  Super built-up  Area       20-Dec  Electronic City  4 BHK  KonteiT

     total_sqft  bath  balcony  price
6909       2093   4.0      1.0  104.0
```

[152]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13320 entries, 0 to 13319
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   area_type     13320 non-null  object
 1   availability  13320 non-null  object
 2   location      13319 non-null  object
 3   size          13304 non-null  object
 4   society       7819 non-null   object
 5   total_sqft    13320 non-null  object
 6   bath          13247 non-null  float64
 7   balcony       12711 non-null  float64
 8   price         13320 non-null  float64
dtypes: float64(3), object(6)
memory usage: 936.7+ KB
```

[154]:
```
df.describe()
```

[154]:
```
                bath       balcony         price
count  13247.000000  12711.000000  13320.000000
mean       2.692610      1.584376    112.565627
std        1.341458      0.817263    148.971674
min        1.000000      0.000000      8.000000
25%        2.000000      1.000000     50.000000
50%        2.000000      2.000000     72.000000
75%        3.000000      2.000000    120.000000
max       40.000000      3.000000   3600.000000
```

[156]:
```
df.columns
```

[156]:
```
Index(['area_type', 'availability', 'location', 'size', 'society',
       'total_sqft', 'bath', 'balcony', 'price'],
      dtype='object')
```

[158]:
```
df.shape
```

[158]:
```
(13320, 9)
```

[160]:
```
# Finding null values
df.isnull().sum()
```

[160]:
```
area_type          0
availability       0
location           1
size              16
society         5501
total_sqft         0
bath              73
balcony          609
price              0
dtype: int64
```

[269]:
```
df=df.fillna(0)
df.isnull().sum()
```

[269]:
```
area_type       0
availability    0
location        0
size            0
society         0
total_sqft      0
bath            0
balcony         0
price           0
dtype: int64
```

[271]:
```
df.shape
```

[271]:
```
(13320, 9)
```

[275]:
```
# to find average price
df["price"].mean()
```

[275]:
```
112.5656265015015
```

[278]:
```
# to find average bathroom in the data
df["bath"].mean()
```

[278]:
```
2.677852852852853
```

[280]:
```
# to find how many balcony a house have in the data
df["balcony"].mean()
```

[280]:
```
1.5119369369369369
```

[282]:
```
df["bath"].mode()
```

[282]:
```
0    2.0
Name: bath, dtype: float64
```

[284]:
```
df["balcony"].mode()
```

[284]:
```
0    2.0
Name: balcony, dtype: float64
```

[286]:
```
df["price"].mode()
```

[286]:
```
0    75.0
Name: price, dtype: float64
```

[288]:
```
df["price"].median()
```

[288]:
```
72.0
```

[290]:
```
df["area_type"].unique()
# to check the unique area type
```

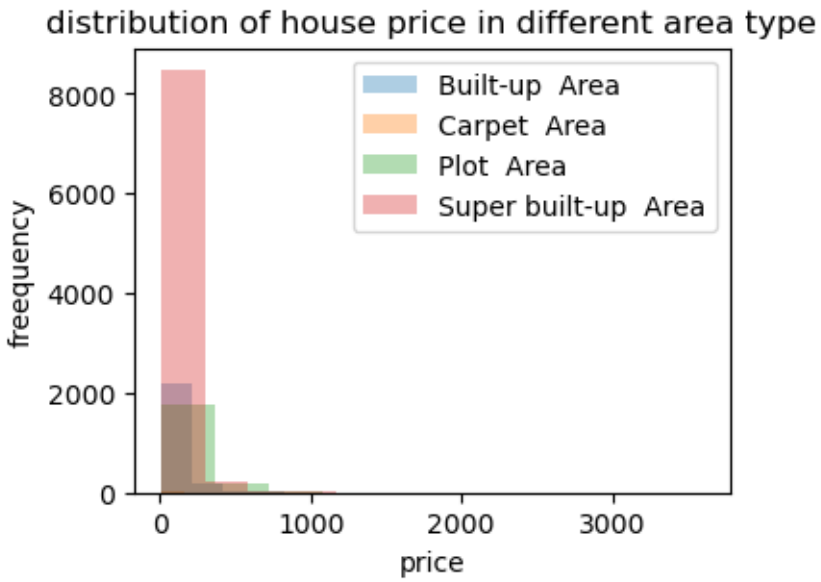
[290]:
```
array(['Super built-up  Area', 'Plot  Area', 'Built-up  Area',
       'Carpet  Area'], dtype=object)
```

[292]:
```
df["location"].unique()
```

[292]:
```
array([ 664,  550,  203, …,  407,  767, 1173])
```

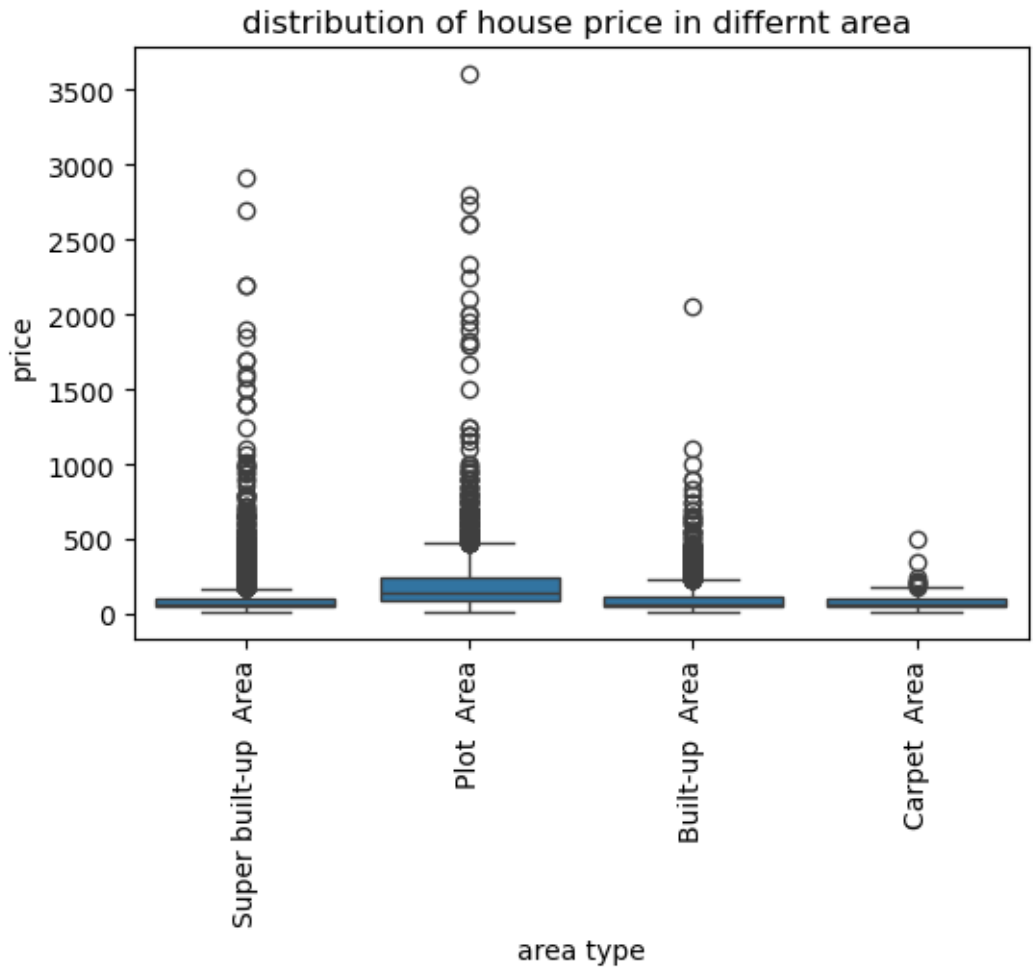
[294]:
```
# average house price according to the area type
plt.figure(figsize=(4,3))
df.groupby("area_type")["price"].plot(kind="hist",alpha=0.36,legend=True)
plt.title("distribution of house price in different area type")
plt.xlabel("price")
plt.ylabel("freequency")
```

[294]:
```
Text(0, 0.5, 'freequency')
```



[295]:
```
plt.figure(figsize=(6,4))
sns.boxplot(x="area_type",y="price",data=df)
plt.title("distribution of house price in differnt area ")
plt.xlabel("area type")
plt.ylabel("price")
plt.xticks(rotation=90)
```

[295]:
```
([0, 1, 2, 3],
 [Text(0, 0, 'Super built-up  Area'),
  Text(1, 0, 'Plot  Area'),
  Text(2, 0, 'Built-up  Area'),
  Text(3, 0, 'Carpet  Area')])
```



[296]:
```
# find highest and lowest price according to the location in the data set
print("location analysis")
print("location with highest price")
print(df.groupby("location")["price"].mean().nlargest(5))
print("\n")
print("location with lowest price ")
print(df.groupby("location")["price"].mean().nsmallest(5))
```

```
location analysis
location with highest price
location
588    1900.000000
359    1486.000000
600    1167.714286
298    1093.388889
575    1068.000000
Name: price, dtype: float64
```

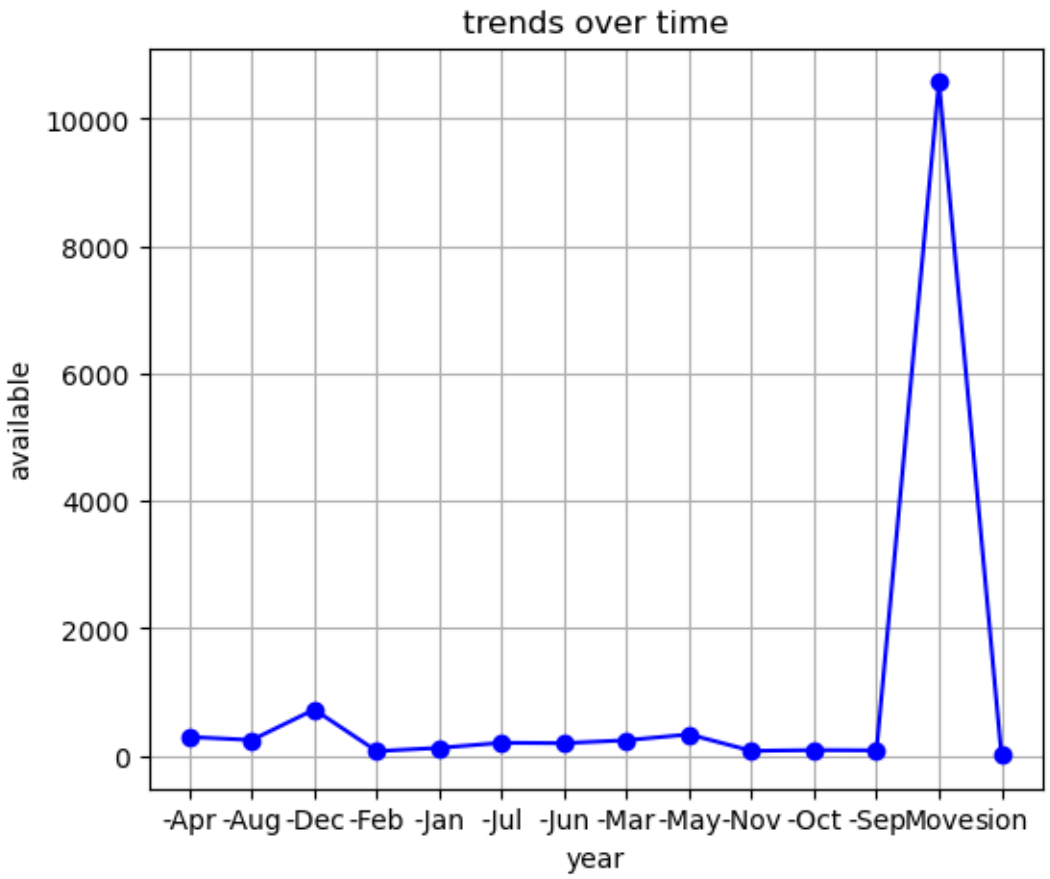
```
location with lowest price
location
69      15.000
0       16.000
1064    16.000
378     17.000
528     19.245
Name: price, dtype: float64
```

[298]:
```
# to find availibility of a house in the dataset
print("average availability of a house")
df["availability"].value_counts()
```

```
average availability of a house
```

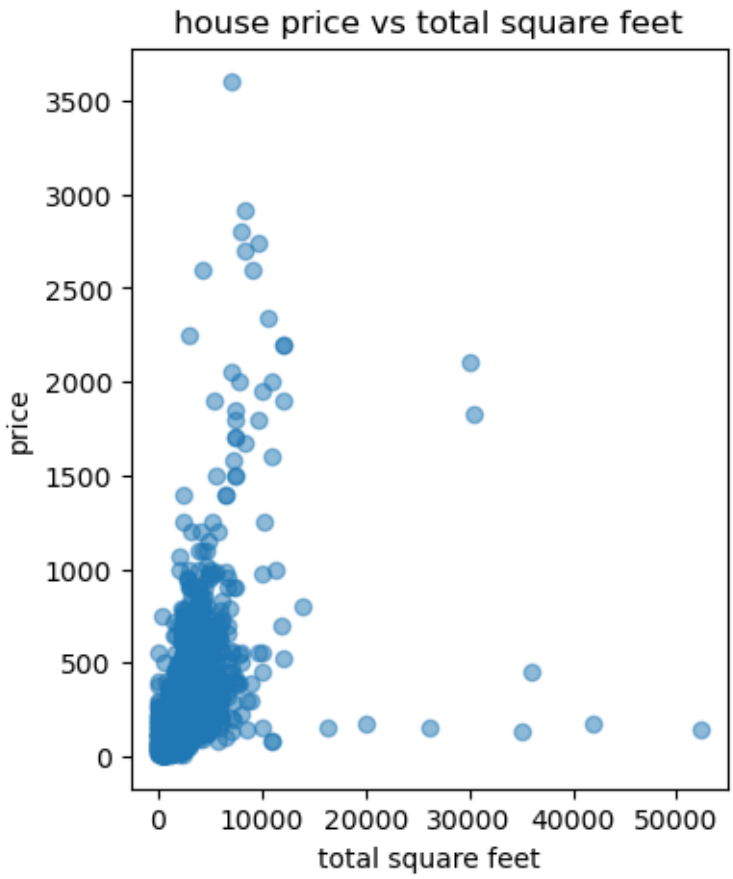
[298]:
```
availability
Ready To Move    10581
18-Dec             307
18-May             295
18-Apr             271
18-Aug             200
                  ...
15-Aug               1
17-Jan               1
16-Nov               1
16-Jan               1
14-Jul               1
Name: count, Length: 81, dtype: int64
```

[301]:
```
# trend availibility of a house
avail=df.groupby(df["availability"].str[-4:])["availability"].count()
# availability trend over time
plt.figure(figsize=(6,5))
plt.plot(avail.index,avail.
  ↪values,marker="o",linestyle="-",color="b",label="availibility over time")
plt.title("trends over time")
plt.xlabel("year")
plt.ylabel("available")
plt.grid()
```



[305]:
```
# to plot house price vs square feet area
plt.figure(figsize=(4,5))
plt.scatter(df["total_sqft"],df["price"],alpha=0.5)
plt.title("house price vs total square feet")
plt.xlabel("total square feet")
plt.ylabel("price")
# plt.grid(True)
```

[305]: Text(0, 0.5, 'price')



[307]:
```
df["total_sqft"].unique()
```

[307]:
```
array([1056., 2600., 1440., …, 2758.,  774., 4689.])
```

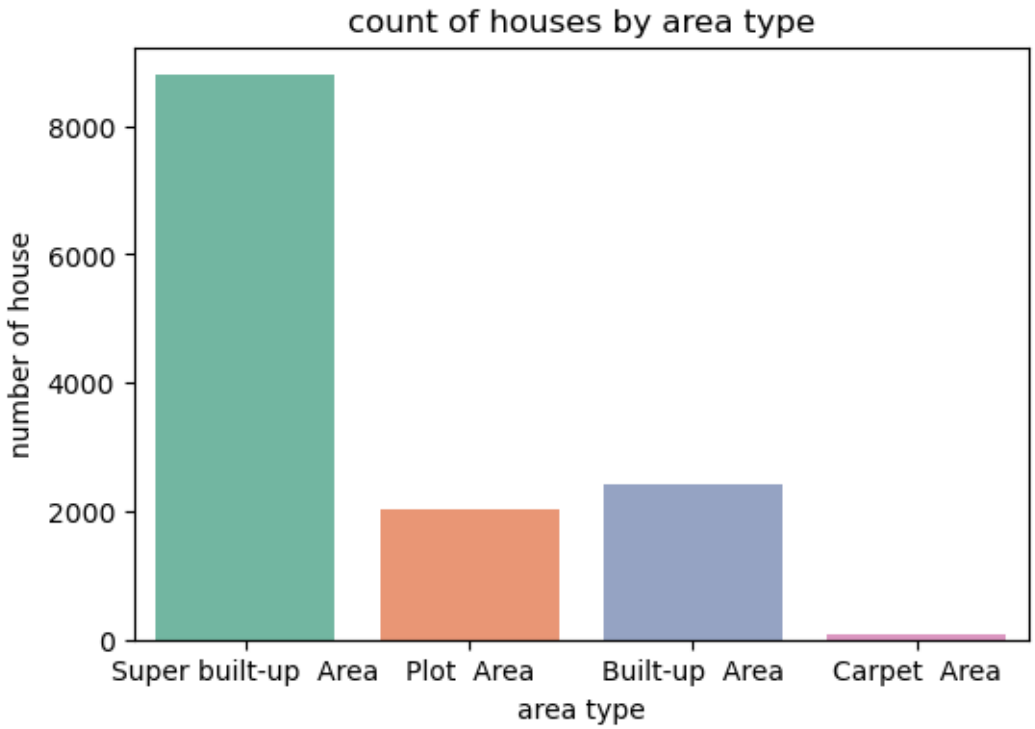
[310]:
```
# to find count of houses by area type
plt.figure(figsize=(6,4))
sns.countplot(data=df,x="area_type",palette="Set2")
plt.title("count of houses by area type")
plt.xlabel("area type")
plt.ylabel("number of house")
```

```
C:\Users\NF\AppData\Local\Temp\ipykernel_3904\4124569865.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(data=df,x="area_type",palette="Set2")
```

[310]: Text(0, 0.5, 'number of house')



[316]:
```
df.columns
```

[316]:
```
Index(['area_type', 'availability', 'location', 'size', 'society',
       'total_sqft', 'bath', 'balcony', 'price'],
      dtype='object')
```

[322]:
```
# convert all the values in location and size column to strings to handle mixed type values
df["location"]=df["location"].astype(str)
df["size"]=df["size"].astype(str)
```

[324]:
```
# convert total_sqft to numerical
def convert_sqft_to_num(x):
    try:
        return float(x)
    except:
        if "-" in x:
            temps=x.split("-")
            return (float(temps[0])+float(temps[1]))/2
        else:
            return None
```

```
df["total_sqft"]=df["total_sqft"].apply(convert_sqft_to_num)
df.dropna(subset=["total_sqft"])
```

[324]:

	area_type	availability	location	size	society	total_sqft	\
0	Super built-up Area	19-Dec	664	6	Coomee	1056.0	
1	Plot Area	Ready To Move	550	13	Theanmp	2600.0	
2	Built-up Area	Ready To Move	203	9	@@@@@@	1440.0	
3	Super built-up Area	Ready To Move	1038	9	Soiewre	1521.0	
4	Super built-up Area	Ready To Move	993	6	0	1200.0	
…	…	…	…	…	…	…	
13315	Built-up Area	Ready To Move	284	16	ArsiaEx	3453.0	
13316	Super built-up Area	Ready To Move	9	11	0	3600.0	
13317	Built-up Area	Ready To Move	1277	6	Mahla T	1141.0	
13318	Super built-up Area	18-Jun	1205	11	SollyCl	4689.0	
13319	Super built-up Area	Ready To Move	637	1	0	550.0	

	bath	balcony	price
0	2.0	1.0	39.07
1	5.0	3.0	120.00
2	2.0	3.0	62.00
3	3.0	1.0	95.00
4	2.0	1.0	51.00
…	…	…	…
13315	4.0	0.0	231.00
13316	5.0	0.0	400.00
13317	2.0	1.0	60.00
13318	4.0	1.0	488.00
13319	1.0	1.0	17.00

[13320 rows x 9 columns]

[326]:
```
# use encoding technique to convert categorical values into numerical values
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

[328]:
```
# apply label encoder to location and size to convert numerical
df["location"]=le.fit_transform(df["location"])
df["size"]=le.fit_transform(df["size"])
```

[330]:
```
df1=df.fillna(0)
df1.isnull().sum()
# split the data in x and y
x=df[["location","size","total_sqft","bath"]]#features
y=df["price"]#target
```

[361]:
```
# import train test split
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
↪2,random_state=101)
```

[365]:
```
# import linear regression and fit the model
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)
```

[365]:
```
LinearRegression()
```

[367]:
```
# store the predicted data into y_prediction
y_pred=model.predict(x_test)
```

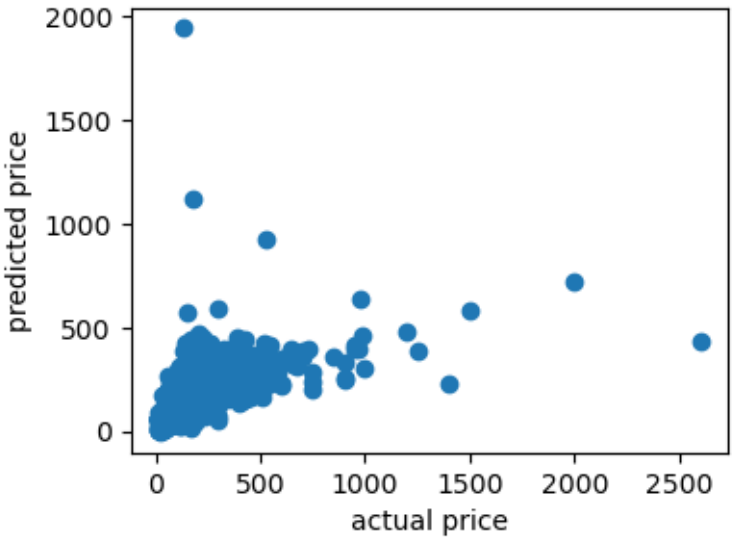
[371]:
```
# import mse and r2 score
from sklearn.metrics import mean_squared_error,r2_score
mse=mean_squared_error(y_test,y_pred)
rmse=np.sqrt(mse)
r2=r2_score(y_test,y_pred)
```

[373]:
```
# print the value of mse and r2 score
print(f"root mean square error (rsme) {rmse}")
print(f"R-squared (R2) {r2}")
```

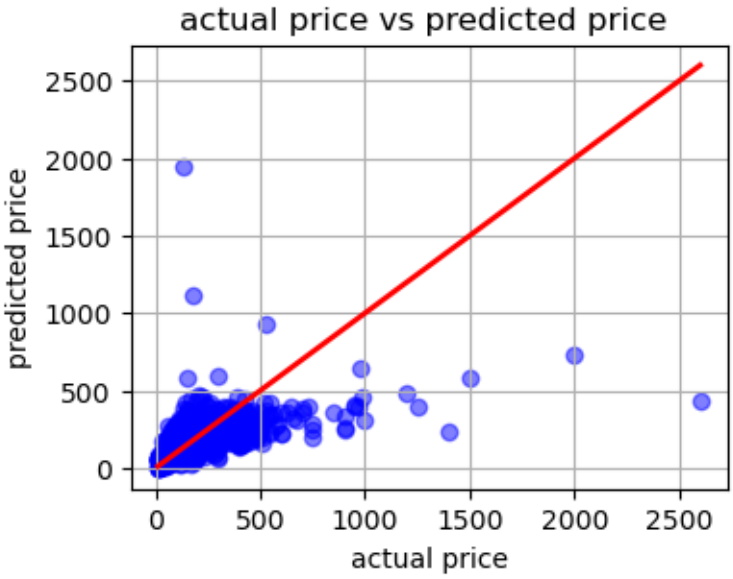
```
root mean square error (rsme) 102.72103957997116
R-squared (R2) 0.4086255285644287
```

[375]:
```
# plot actual vs predicted values
plt.figure(figsize=(4,3))
plt.scatter(y_test,y_pred)
plt.xlabel("actual price")
plt.ylabel("predicted price")
```

[375]:
```
Text(0, 0.5, 'predicted price')
```



[379]:
```
plt.figure(figsize=(4,3))
plt.scatter(y_test,y_pred,color="blue",alpha=0.5)
plt.plot([y_test.min(),y_test.max()],[y_test.min(),y_test.
  ↪max()],color="red",linewidth=2)
plt.title("actual price vs predicted price ")
plt.xlabel("actual price")
plt.ylabel("predicted price")
plt.grid(True)
```



[ ]: