```
from google.colab import files

# Upload file
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to

```
#importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
#loading data
df=pd.read_csv('house_dataset.csv')
```

```
df.head(10)  #printing first 10 rows of the data
```

	Area (sqft)	Bedrooms	Bathrooms	Distance to City	Parking Space	House Age (yrs)	Price (in lakhs)
0	1360	2	3	28.51	2	37	11.24
1	1794	1	1	23.14	2	14	57.01
2	1630	4	2	5.06	1	20	124.21
3	1595	2	1	26.19	0	9	49.17
4	2138	4	2	15.14	0	22	126.91
5	2669	3	3	26.94	2	38	118.76
6	966	2	3	24.20	0	4	34.22
7	1738	2	1	13.33	0	0	80.27
8	830	3	1	1.65	0	12	55.59
9	1982	2	2	8.79	2	20	86.36

```
df.shape  #printing dimension of dataframe
```

```
(500, 7)
```

```
df.describe()
```

	Area (sqft)	Bedrooms	Bathrooms	Distance to City	Parking Space	House Age (yrs)	Price (in lakhs)
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000
mean	2022.420000	2.424000	2.022000	15.758340	0.976000	24.950000	86.536240
std	845.049606	1.148417	0.819059	8.624099	0.800441	14.185978	50.868419
min	501.000000	1.000000	1.000000	1.190000	0.000000	0.000000	-49.710000
25%	1315.000000	1.000000	1.000000	7.880000	0.000000	13.750000	49.387500
50%	2009.500000	2.000000	2.000000	16.260000	1.000000	24.000000	85.120000
75%	2753.500000	3.000000	3.000000	23.345000	2.000000	37.000000	125.167500
max	3499.000000	4.000000	3.000000	29.940000	2.000000	49.000000	220.980000

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 7 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Area (sqft)       500 non-null    int64  
 1   Bedrooms          500 non-null    int64  
 2   Bathrooms         500 non-null    int64  
 3   Distance to City  500 non-null    float64
 4   Parking Space     500 non-null    int64  
 5   House Age (yrs)   500 non-null    int64  
```

```
 6   Price (in lakhs)  500 non-null    float64
dtypes: float64(2), int64(5)
memory usage: 27.5 KB
```

```
df.dtypes  #datatypes of each column type is printed
```

	0
Area (sqft)	int64
Bedrooms	int64
Bathrooms	int64
Distance to City	float64
Parking Space	int64
House Age (yrs)	int64
Price (in lakhs)	float64

```
df.isna()  #finding Nan values if any
```

	Area (sqft)	Bedrooms	Bathrooms	Distance to City	Parking Space	House Age (yrs)	Price (in lakhs)
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
495	False	False	False	False	False	False	False
496	False	False	False	False	False	False	False
497	False	False	False	False	False	False	False
498	False	False	False	False	False	False	False
499	False	False	False	False	False	False	False

500 rows × 7 columns

```
df.isna().sum()  #sum of Nan values if any
```

	0
Area (sqft)	0
Bedrooms	0
Bathrooms	0
Distance to City	0
Parking Space	0
House Age (yrs)	0
Price (in lakhs)	0

```
df.duplicated()  #checking for duplicated rows
```

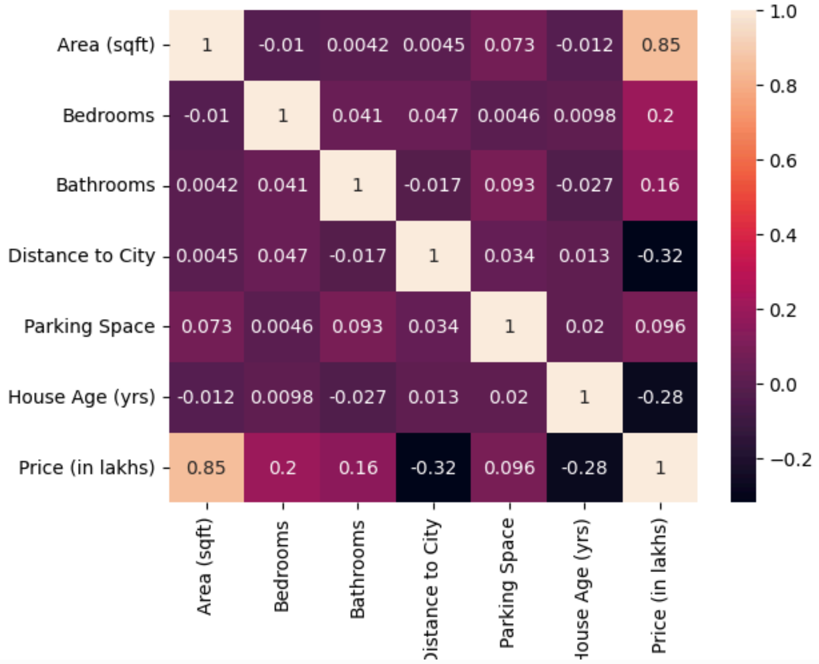
	0
0	False
1	False
2	False
3	False
4	False
...	...
495	False
496	False
497	False
498	False
499	False

500 rows × 1 columns

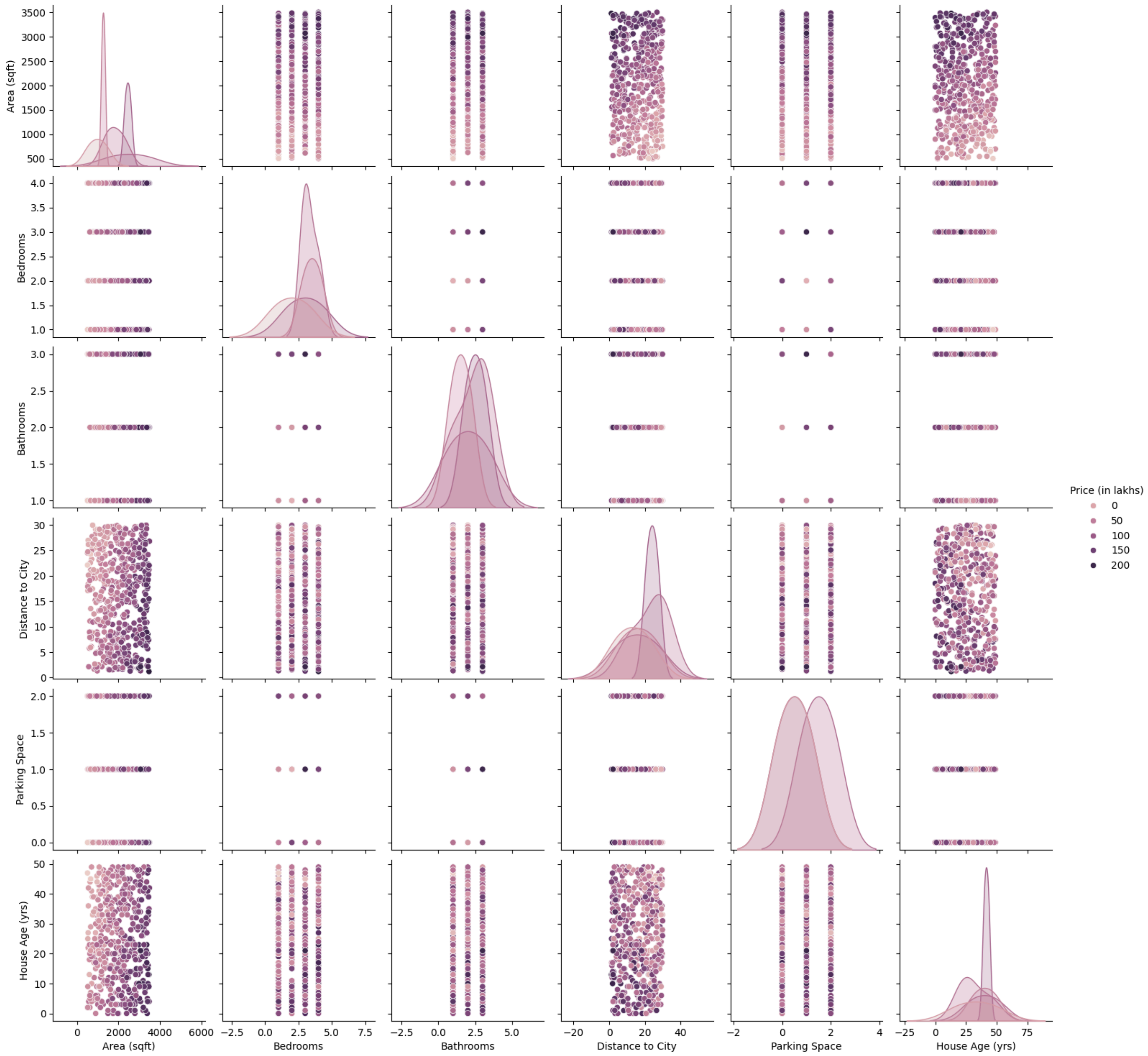
```
df.duplicated().sum()
```

```
np.int64(0)
```

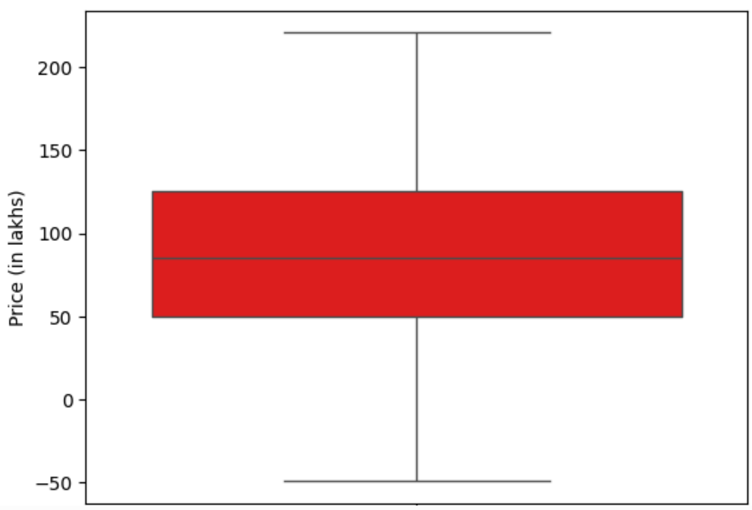
```
corr=sns.heatmap(df.corr(),annot=True)  #created a heatmap using correlation matrix which can be used to display relationship between di
```




```
sns.pairplot(data=df,hue='Price (in lakhs)')  #pairplot to display relationship of each columns
plt.show()
```




```
sns.boxplot(data=df['Price (in lakhs)'], color='red')  #boxplot to identify any outliers exist in price of houses
plt.show()
```



We can see that the house price distribution goes upto -50 which is practicaly impossible.

```
df[df['Price (in lakhs)']<=0] #checking for price values which are below zero
```

	Area (sqft)	Bedrooms	Bathrooms	Distance to City	Parking Space	House Age (yrs)	Price (in lakhs)
24	521	4	1	17.07	2	25	-7.69
37	1062	1	1	13.14	0	41	-3.46
51	741	2	2	17.43	1	39	-2.16
85	837	1	1	26.32	0	20	-10.79
106	891	3	1	29.16	0	27	-11.53
110	878	2	1	25.49	1	39	-17.28
148	595	1	1	22.39	1	13	-14.27
236	686	4	1	20.72	1	45	-3.68
246	646	1	2	22.56	1	21	-6.54
292	753	1	2	29.92	0	33	-14.55
308	897	3	2	19.52	1	35	-0.34
319	959	1	3	26.07	0	46	-15.37
321	969	2	1	29.08	2	44	-20.79
354	680	3	1	25.94	0	45	-30.61
377	654	1	1	24.22	1	45	-42.60
390	1024	1	1	24.25	2	21	-1.87
394	535	1	1	26.95	2	27	-22.30
426	504	1	1	22.09	0	47	-49.71
433	546	2	3	26.14	1	36	-3.59
465	1059	2	1	29.11	1	33	-11.56

```
df[df['Price (in lakhs)']<=0].shape  #finding shape of data having negative price values
```

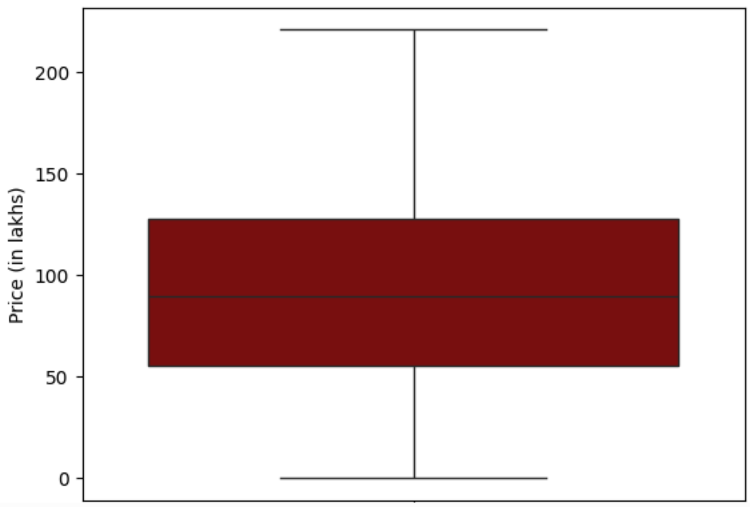
```
(20, 7)
```

```
df = df[df['Price (in lakhs)'] > 0]  # only positive price values are persisted
```

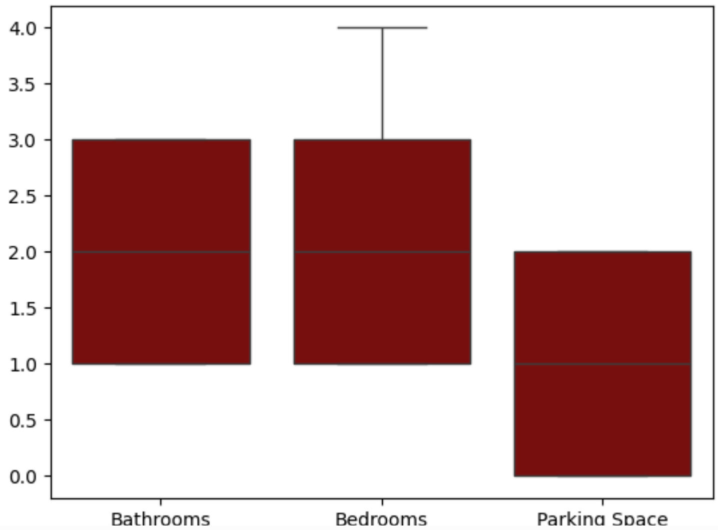
```
df['Price (in lakhs)'].shape   #size of data with postive price values
```

```
(480,)
```

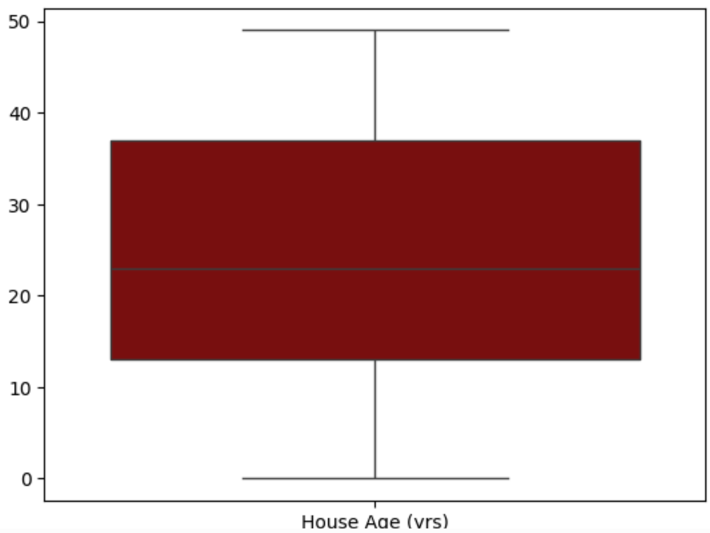
```
sns.boxplot(data=df['Price (in lakhs)'], color='darkred')
plt.show()              #visualizing price column after removing outliers
```




```
sns.boxplot(data=[df['Bathrooms'],df['Bedrooms'],df['Parking Space']], color='darkred')

plt.show()
```




```
sns.boxplot(data=[df['House Age (yrs)']], color='darkred')

plt.show()
```




```
sns.boxplot(data=[df['Distance to City']], color='darkred')

plt.show()
```

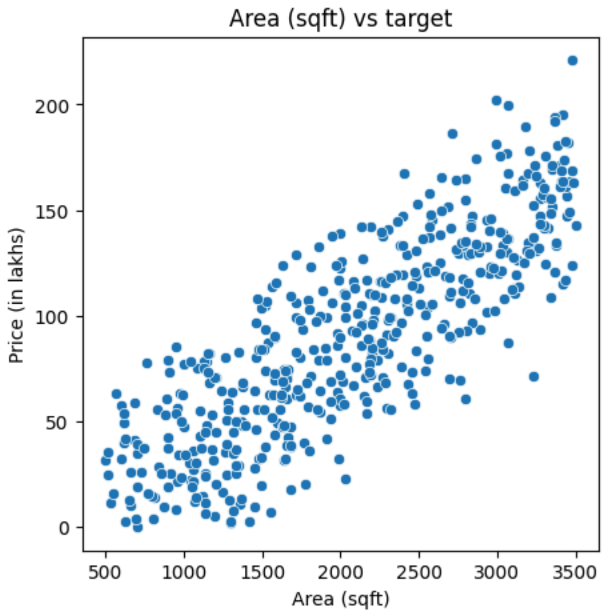
Linear Regression Model

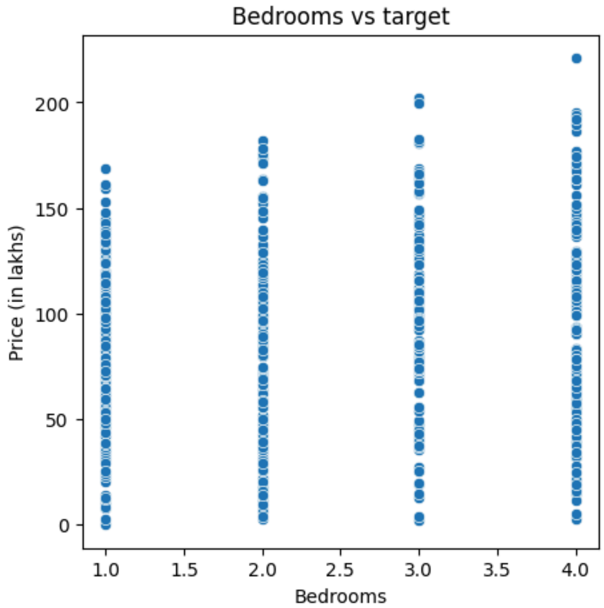
```
#importing libraries for linear regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,r2_score
```

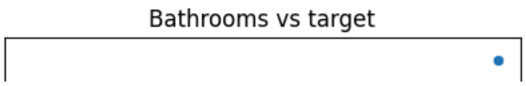
```
X = df.drop('Price (in lakhs)', axis=1)  # Features
y = df['Price (in lakhs)']  #target or label
```

```
X_train, X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

```
for column in X.columns:
  plt.figure(figsize=(5,5))
  sns.scatterplot(x=X[column],y=y)
  plt.title(f'{column} vs target')
  plt.show()
```








```
correlation=pd.concat([X,y],axis=1).corr()
print(correlation[('Price (in lakhs)')].sort_values(ascending=False))
#it shows correlation between target and features after removing outliers
```

```
Price (in lakhs)      1.000000
Area (sqft)           0.833482
```

```
Bedrooms            0.169484
Bathrooms           0.102740
Parking Space       0.090097
House Age (yrs)    -0.247302
Distance to City   -0.269288
Name: Price (in lakhs), dtype: float64
```

this results shows only area and price variable have strong linear relationship, hence linear regression cannot be considered as an effective method

```
lg=LinearRegression()
lg.fit(X_train,y_train)
#applied linear regression model
```

```
LinearRegression
LinearRegression()
```

```
y_pred=lg.predict(X_test)
y_pred
#printing predicted output
```

```
array([ 94.56508563, 203.23123494,  13.07692451,  78.47387607,
       116.5846289 ,  78.78947913, 108.55559977, 150.58982951,
        27.52806592,  44.22878273, 100.38992778, 132.16155817,
        61.05540177,  83.42454379,  91.36593427, 168.29936927,
       110.5784902 ,  69.53864677,  76.31359146,  77.26568666,
        47.56452514,  75.94587236,  63.4206808 ,  50.06468059,
       100.28702794, 120.68979067, 138.22944538, 171.20627432,
        95.52661629,  22.62640103, 138.83663726, 105.73975322,
       149.83819464,  63.3075822 , 174.34522857, 142.69292617,
        51.27565633, 185.14188609, 127.20323885, 149.11454607,
       132.01848852,  78.13721311,  26.02736035,  35.26507095,
       104.48407593, 125.66752273,  74.54393762,  12.58641718,
       127.26502237, 110.40925526,  51.67678445,   1.27196068,
       173.12303982,  91.85920757,  56.6208141 ,  44.49495021,
        46.32723429,  94.68105423, 141.67771192,  67.67649035,
       117.43906536,  71.34439493,  34.02292755,  39.13123423,
       129.66733275,  89.00819898,  63.0094442 , 115.7230588 ,
        31.07639485,  36.50773337, 134.85123761,  78.19256713,
        77.5626971 ,  36.56180953, 133.54634351,  46.6451067 ,
        95.73286793, 132.72389611, 127.16942701, 122.27455525,
        38.89930749, 104.27229832,  76.30210447,  90.71053802,
       112.13036874,  87.02158803, 133.30795139,  83.25264801,
       112.65852732,  99.72966903,  58.22839687, 114.10206337,
        80.13796716,  93.24730748,  29.86817726,   5.65082487])
```

```
mean_squared_error(y_test,y_pred)  #finding mean_squared_value error between actual test output and predicted value
```

```
100.90070866927128
```

```
value=np.sqrt(mean_squared_error(y_test,y_pred))
print(value)
```

```
10.044934478097469
```

```
r2_score(y_test,y_pred)  #r2 score is calculated between actual and predicted
```

```
0.9497208092801454
```

```
#checking if values are overfitted or underfitted

y_train_pred=lg.predict(X_train)
r2_score(y_train,y_train_pred)
```

```
0.9543926994489432
```

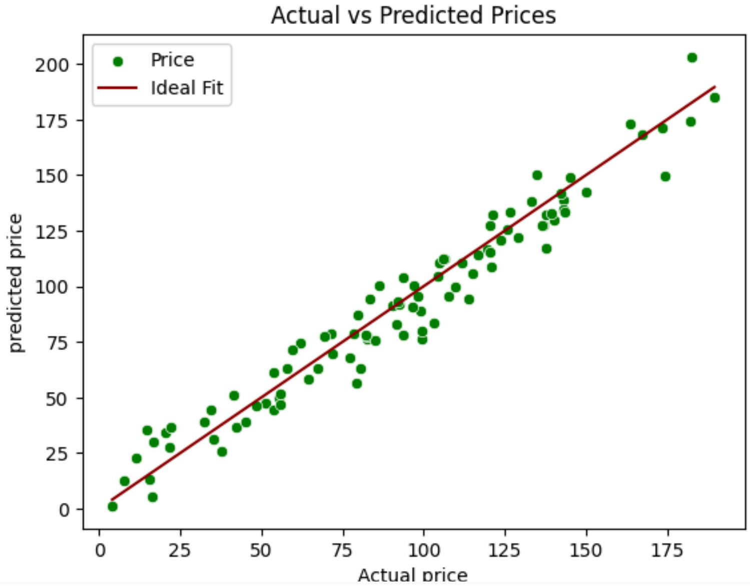
since train r2 score > test r2 score there is neither overfit nor underfit, hence regularization is not required

```
lg.coef_
```

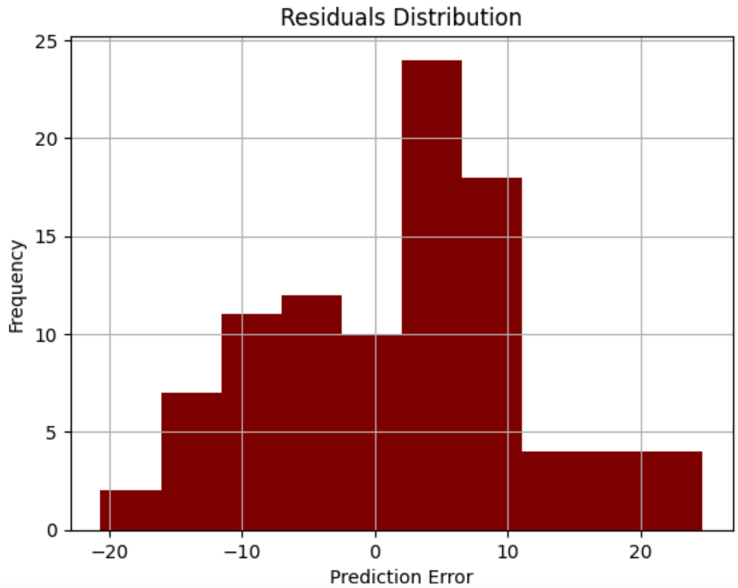
```
array([ 0.05061193,  9.98933784,  7.37998883, -1.92185377,  2.64438886,
       -0.93905505])
```

```
#plot showing actual price vs predicted price along with ideal fit line
ax=sns.scatterplot(x=y_test,y=y_pred, color='green',label='Price')
ax.set_xlabel('Actual price')
ax.set_ylabel('predicted price')
```

```
plt.title('Actual vs Predicted Prices')

plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='darkred',label='Ideal Fit')
plt.legend(loc='best')
plt.show()
```




```
#plot of residual distribution
residuals = y_test - y_pred

plt.hist(residuals, bins=10, color='maroon')
plt.title("Residuals Distribution")
plt.xlabel("Prediction Error")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()
```



Decision Tree regression model

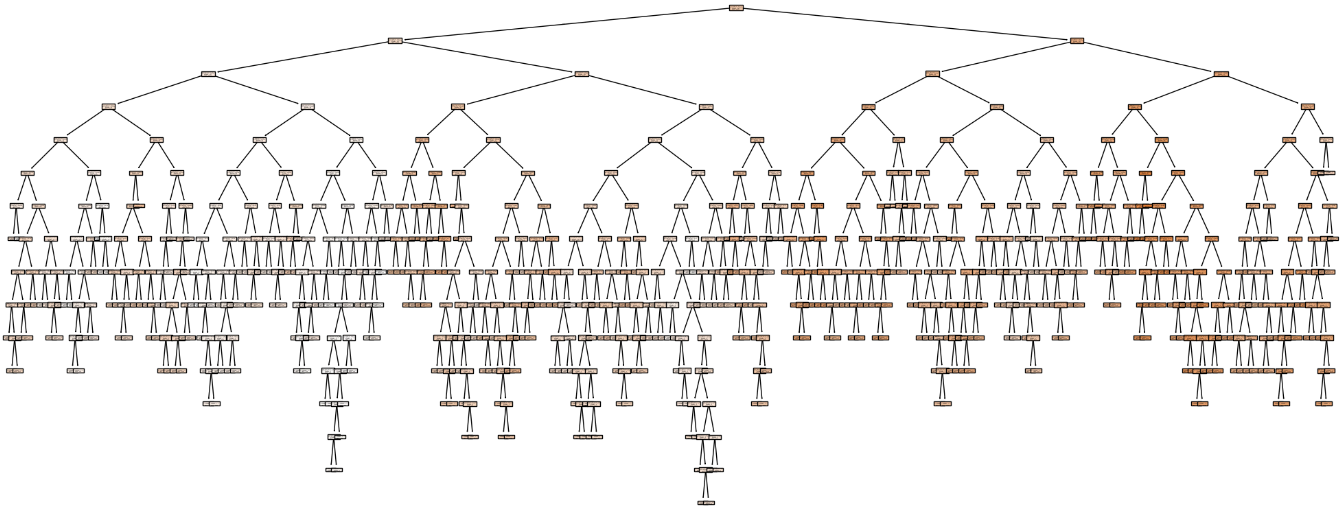
```
#importing decision tree regressor from scikit learn library
from sklearn.tree import DecisionTreeRegressor


X=df.drop('Price (in lakhs)',axis=1)
y=df['Price (in lakhs)']

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

```
dr=DecisionTreeRegressor(random_state=42)
dr.fit(X_train,y_train)
```

```
DecisionTreeRegressor
DecisionTreeRegressor(random_state=42)
```

```
from sklearn.tree import plot_tree  #to plot tree
```

```
#This tree plot gives an idea of how many decision trees are made at each level.
plt.figure(figsize=(25,10))
plot_tree(dr,filled=True,feature_names=X.columns)
plt.show()
```



```
y_pred=dr.predict(X_test)
```

```
r2_score(y_pred,y_test)
```

```
0.7401336679152561
```

```
y_pred_train=dr.predict(X_train)
```

```
r2_score(y_pred_train,y_train)
```

```
1.0
```

```
sample=X_test.iloc[[0]]
prediction=dr.predict(sample)
sample
```

	Area (sqft)	Bedrooms	Bathrooms	Distance to City	Parking Space	House Age (yrs)
76	2481	4	1	18.12	0	42

```
print(prediction[0])  #predicting value for a house in lakhs
```

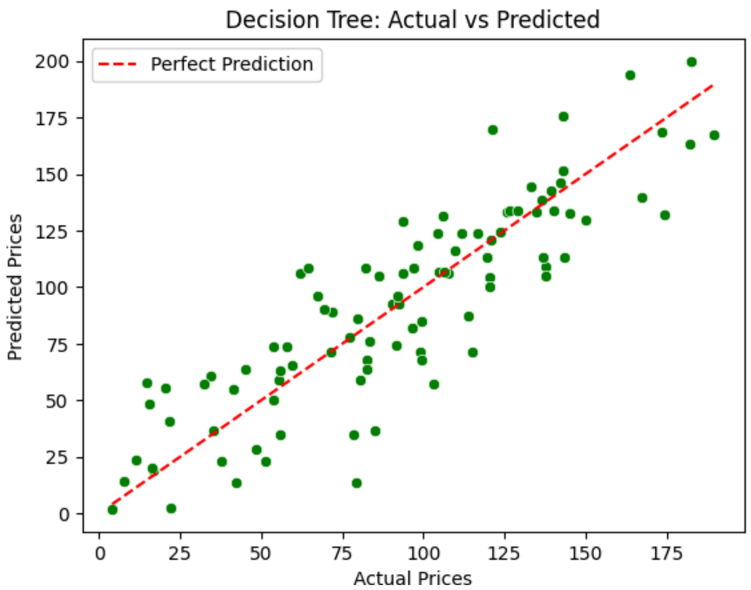
```
76.15
```

```
actual_val=y_test.iloc[0]
print(actual_val)   # actual value for that particular house in lakhs
```

```
83.58
```

```
#actual price vs predicted price along with ideal line
sns.scatterplot(x=y_test, y=y_pred, color='green')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], '--r', label='Perfect Prediction')

plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
```

```
plt.title('Decision Tree: Actual vs Predicted')
plt.legend()
plt.show()
```



Random Forest Regressor model

```
from sklearn.ensemble import RandomForestRegressor
X=df.drop('Price (in lakhs)',axis=1)
y=df['Price (in lakhs)']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

```
rg=RandomForestRegressor(
    n_estimators=100,
    max_features='sqrt',
    random_state=42
)  #sqrt is for selecting root number of  features from square number of  features in total (eg:- 2 from 4)
rg.fit(X_train,y_train)
```

```
RandomForestRegressor
RandomForestRegressor(max_features='sqrt', random_state=42)
```

```
y_pred=rg.predict(X_test)
y_pred_train=rg.predict(X_train)
```

```
r2_score(y_pred,y_test)
```

```
0.7737628340341799
```

```
r2_score(y_pred_train,y_train)
```

```
0.9798616366544238
```

```
sample=X_test.iloc[[0]]
print('Features of test data')
sample
```

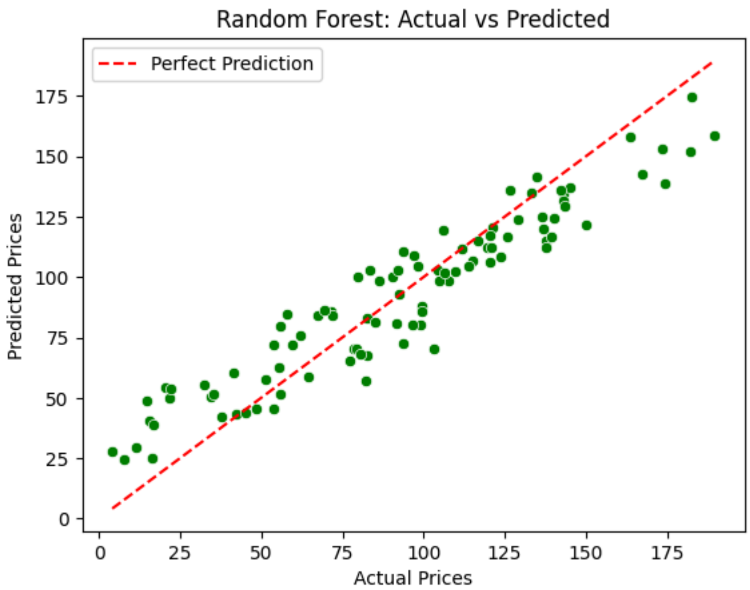
Features of test data

	Area (sqft)	Bedrooms	Bathrooms	Distance to City	Parking Space	House Age (yrs)
76	2481	4	1	18.12	0	42

```
#printing actual and predicted value for a house
prediction=rg.predict(sample)
print(prediction[0])
print(y_test.iloc[0])
```

```
102.8536
83.58
```

```
#actual price vs predicted price along with ideal line
sns.scatterplot(x=y_test, y=y_pred, color='green')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], '--r', label='Perfect Prediction')

plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.title('Random Forest: Actual vs Predicted')
plt.legend()
plt.show()
```



XG boost regressor model

```
import xgboost as xgb
```

```
X=df.drop('Price (in lakhs)',axis=1)
y=df['Price (in lakhs)']
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

```
xb = xgb.XGBRegressor(n_estimators=500,max_depth=2,learning_rate=0.1,random_state=42,reg_alpha=0.5,reg_lambda=1.0)
xb.fit(X_train,y_train)
```

```
XGBRegressor

XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=2, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=500, n_jobs=None,
             num_parallel_tree=None, random_state=42, ...)
```

```
y_pred=xb.predict(X_test)
y_predmodel=xb.predict(X_train)
```

```
r2_score(y_pred,y_test)
```

```
0.8957925225269913
```

```
r2_score(y_predmodel,y_train)
```

```
0.9840777576056675
```

```
mean_squared_error(y_pred,y_test)
```

```
181.33961824616154
```

```
mean_squared_error(y_predmodel,y_train)
```

```
35.56950211366625
```