HOUSE PRICE PREDICTION

PROJECT
BY
CH.MONIKA
S180461
CSE-2A

PREFACE

About the project:

House price prediction using machine learning helps you extract useful information according to house prices in localty.

The entire idea of project is to analyze the accuracy of predicting house price.

INTRODUCTION

With the rapid development of the country's economy in the past few years, housing price, which cover many livelihood issues, has become a concerning domestic economic problem. People buy houses at different prices because they do not thoroughly understand the house price system

Different people buy houses with the same value at different prices, which usually leads to dissatisfaction with housing prices and unfair housing prices. To solve this problem, we designed an objective housing price prediction scheme based on a decision tree.

STEPS TO BULID A MODEL:

1. IMPORTING DATASET
2. ANALYZING THE DATASET
3. CLEANING AND PREPARING DATASET
4. DATA VISUALIZATION
5. MODEL BUILDING
6. MODEL EVALUATION.

INPUT TO THE MODEL:

As for my model inputs are taken from the dataset colunms. Based on the my model I have trained my model using specific colunms those are sqft_living yr_bulit , yr_renovated, and bedrooms based on these input my model will make predictions.

OUTPUT OF THE MODEL:

It will predict the price of the house based on the input colunms

Analysis

Taken Columns: price, bedrooms ,bathrooms, sqft _living , sqft_lot, floors, waterfront, view, condition, sqft_above,sqft_basement,yr_built,yr_renovated,street,city,statezip,country,Day,month, year

In [2]:
```
df=pd.read_csv('data.csv')
df.head()
```

Out[2]:

price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	street	city	st
3000.0	3	1.50	1340	7912	1.5	0	0	3	1340	0	1955	2005	18810 Densmore Ave N	Shoreline	
4000.0	5	2.50	3650	9050	2.0	0	4	5	3370	280	1921	0	709 W Blaine St	Seattle	
2000.0	3	2.00	1930	11947	1.0	0	0	4	1930	0	1966	0	26206-26214 143rd Ave SE	Kent	
0000.0	3	2.25	2000	8030	1.0	0	0	4	1000	1000	1963	0	857 170th Pl NE	Bellevue	
0000.0	4	2.50	1940	10500	1.0	0	0	4	1140	800	1976	1992	9105 170th Ave NE	Redmond	

Dataset contains:

Date: Date house was sold

Price: Price is prediction target

Bedrooms: Number of Bedrooms/House

Bathrooms: Number of bathrooms/House

Sqft_Living: square footage of the home

Sqft_Lot: square footage of the lot

Floors: Total floors (levels) in house

Waterfront: House which has a view to a waterfront

View: Has been viewed

Condition: How good the condition is (Overall)

Sqft_Above: square footage of house apart from basement

Sqft_Basement: square footage of the basement

Yr_Built: Built Year

Yr_Renovated: Year when house was renovated

Zipcode: Zip

Sqft_Living15: Living room area in 2015(implies-- some renovations) This might or might not have affected the lotsize area
Sqft_Lot15: lotSize area in 2015(implies-- some renovations)

CODE:

Importing the libraries

In [135]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_absolute_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.metrics import accuracy_score
```

Exploratory Data Analysis:

In [50]:
```
df=pd.read_csv("data.csv")
df.head()
```

Out[50]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	st
0	5/2/2014 0:00	313000.0	3	1.50	1340	7912	1.5	0	0	3	1340	0	1955	2005	18 Densr A
1	5/2/2014 0:00	2384000.0	5	2.50	3650	9050	2.0	0	4	5	3370	280	1921	0	70 Blair
2	5/2/2014 0:00	342000.0	3	2.00	1930	11947	1.0	0	0	4	1930	0	1966	0	26 26 143rd
3	5/2/2014 0:00	420000.0	3	2.25	2000	8030	1.0	0	0	4	1000	1000	1963	0	857 1 P
4	5/2/2014 0:00	550000.0	4	2.50	1940	10500	1.0	0	0	4	1140	800	1976	1992	9 170th

In [51]:
```
df.tail()
```
Out[51]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated
4595	7/9/2014 0:00	308166.6667	3	1.75	1510	6360	1.0	0	0	4	1510	0	1954	1979
4596	7/9/2014 0:00	534333.3333	3	2.50	1460	7573	2.0	0	0	3	1460	0	1983	2009
4597	7/9/2014 0:00	416904.1667	3	2.50	3010	7014	2.0	0	0	3	3010	0	2009	0
4598	7/10/2014 0:00	203400.0000	4	2.00	2090	6630	1.0	0	0	3	1070	1020	1974	0
4599	7/10/2014 0:00	220600.0000	3	2.50	1490	8102	2.0	0	0	4	1490	0	1990	0

In [52]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4600 entries, 0 to 4599
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   date           4600 non-null   object 
 1   price          4600 non-null   float64
 2   bedrooms       4600 non-null   int64  
 3   bathrooms      4600 non-null   float64
 4   sqft_living    4600 non-null   int64  
 5   sqft_lot       4600 non-null   int64  
 6   floors         4600 non-null   float64
 7   waterfront     4600 non-null   int64  
 8   view           4600 non-null   int64  
 9   condition      4600 non-null   int64  
 10  sqft_above     4600 non-null   int64  
 11  sqft_basement  4600 non-null   int64  
 12  yr_built       4600 non-null   int64  
 13  yr_renovated   4600 non-null   int64  
 14  street         4600 non-null   object 
 15  city           4600 non-null   object 
 16  statezip       4600 non-null   object 
 17  country        4600 non-null   object 
dtypes: float64(3), int64(10), object(5)
memory usage: 647.0+ KB
```

Checking for the missing values:

Null Value Detection Let's Check for null values in the dataset

```
In [53]: df.isnull()
```

Out[53]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	street	
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	F
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	F
2	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	F
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	F
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	F
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4595	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	F
4596	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	F
4597	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	F
4598	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	F
4599	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	F

4600 rows × 18 columns

```
In [8]: df.isnull().sum()
```

```
Out[8]: price            0
        bedrooms         0
        bathrooms        0
        sqft_living      0
        sqft_lot         0
        floors           0
        waterfront       0
        view             0
        condition        0
        sqft_above       0
        sqft_basement    0
        yr_built         0
        yr_renovated     0
        street           0
        city             0
        statezip         0
        country          0
        Day              0
        month            0
        year             0
        dtype: int64
```

```
In [55]: df.duplicated().sum()
```

```
Out[55]: 0
```

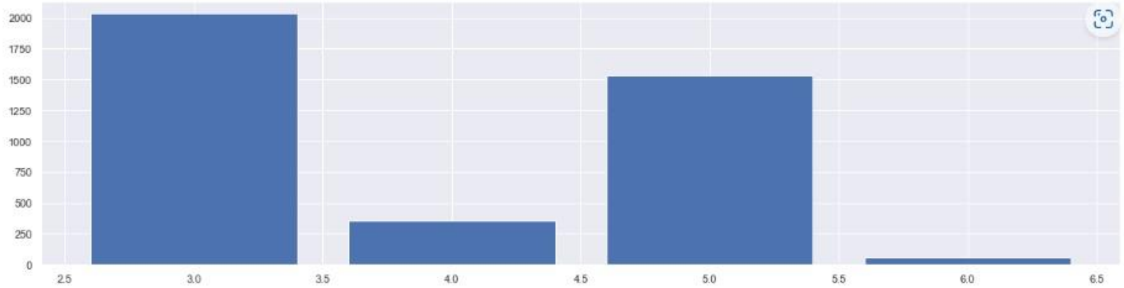
Data Visualization:

Visualizing the correlations between the numerical variables using ploting visualizations.

BARGRAPHS:

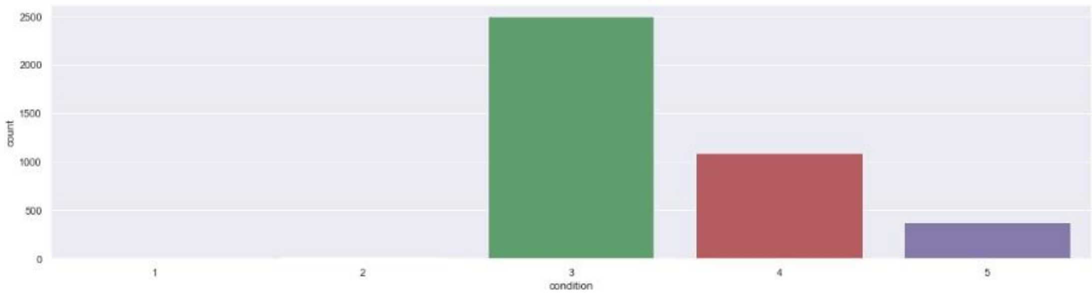
Bargraph for bedrooms value count:

```
In [93]: import matplotlib.pyplot as plt
         x=df['bedrooms'].unique()
         plt.bar(x=x,height=df['bedrooms'].value_counts())
Out[93]: <BarContainer object of 4 artists>
```



Bargraph for conditions

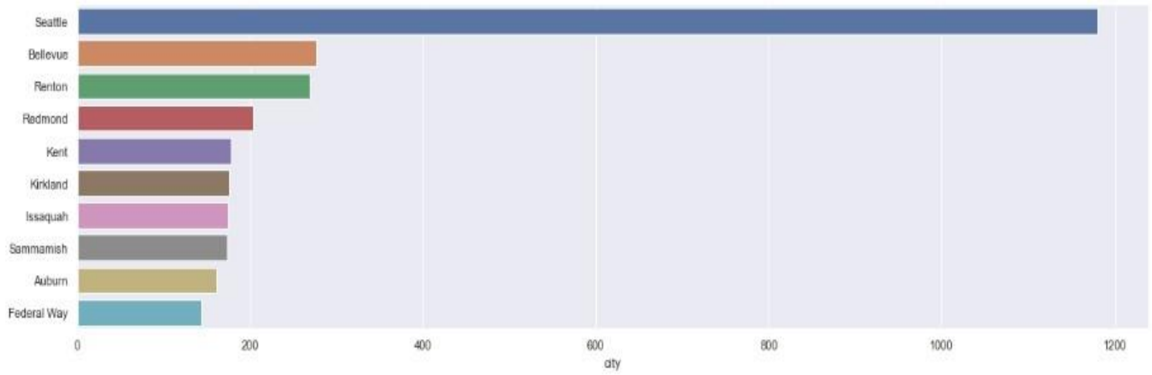
```
In [104]: sns.countplot(df['condition'])
Out[104]: <AxesSubplot:xlabel='condition', ylabel='count'>
```



Bargraph for city&Street:

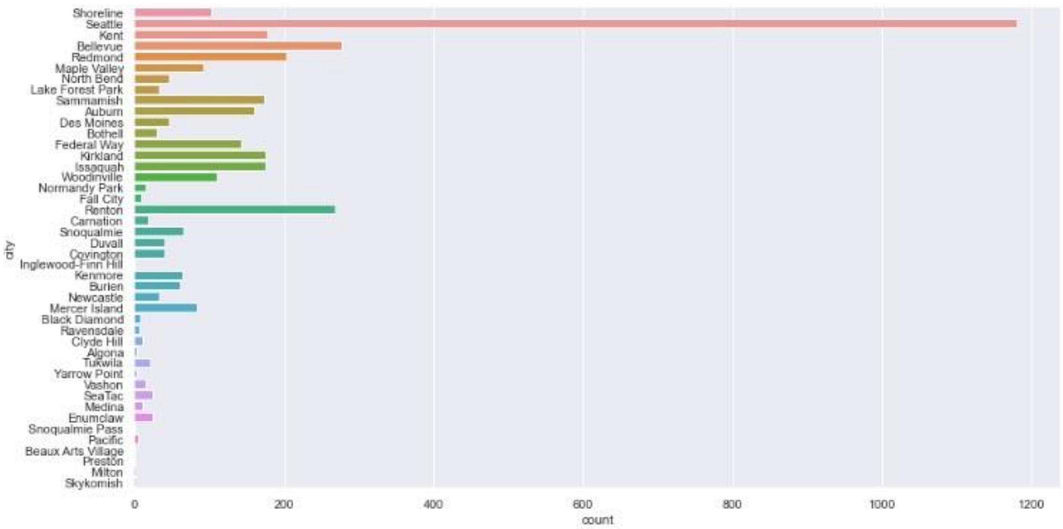
```
In [140]: top_10_cities = df['city'].value_counts().head(10)
          sns.barplot(x = top_10_cities, y=top_10_cities.index)
```

Out[140]: <AxesSubplot:xlabel='city'>



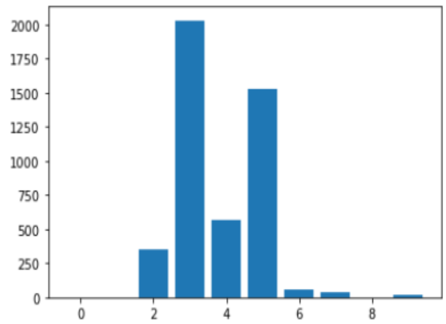

```
In [106]: plt.figure(figsize=(15,8))

          sns.countplot(y=df['city'])
```

Out[106]: <AxesSubplot:xlabel='count', ylabel='city'>



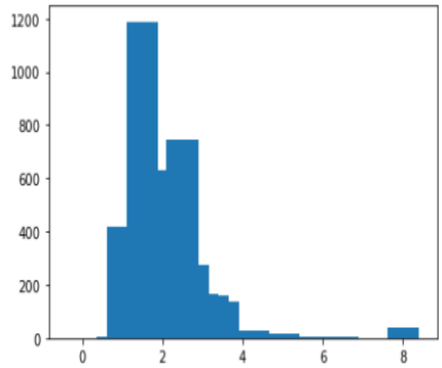
In [65]:
```
x=df['bedrooms'].unique()
plt.bar(x=x,height=df['bedrooms'].value_counts())
```

Out[65]: <BarContainer object of 10 artists>



In [66]:
```
x=df['bathrooms'].unique()
plt.bar(x=x,height=df['bathrooms'].value_counts())
```

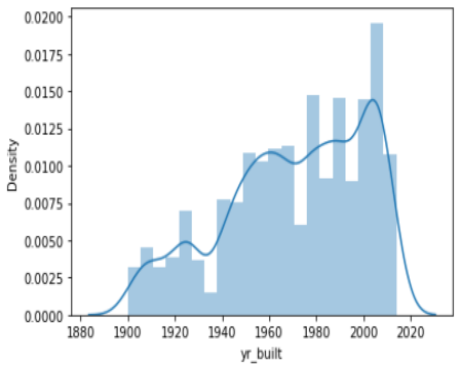
Out[66]: <BarContainer object of 26 artists>



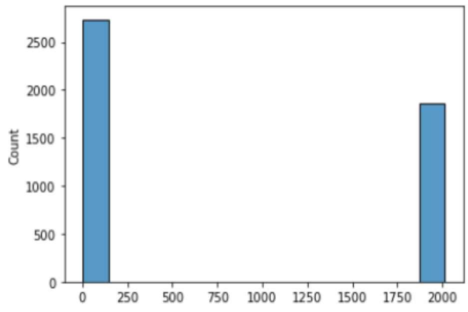
In [69]:
```
sns.distplot(df['yr_built'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function an
d will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[69]: <AxesSubplot:xlabel='yr_built', ylabel='Density'>

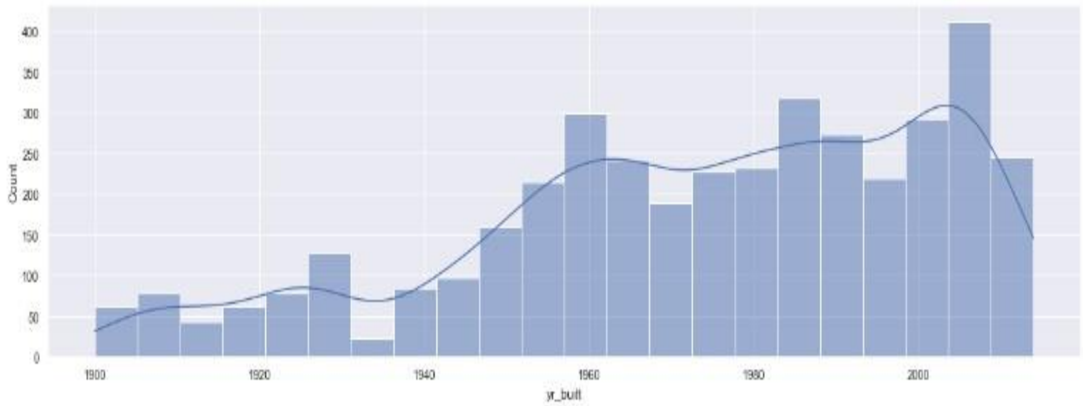


```
In [70]: sns.histplot(df['yr_renovated'])
Out[70]: <AxesSubplot:xlabel='yr_renovated', ylabel='Count'>
```

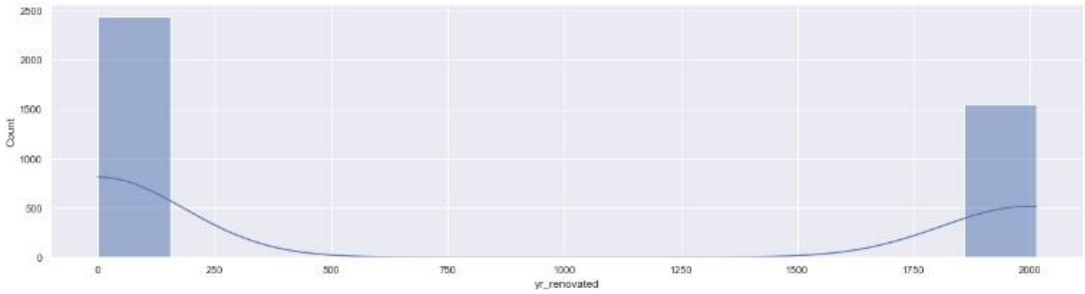


HISTOGRAMS:

```
In [117]: sns.histplot(df.yr_built, kde=True)
Out[117]: <AxesSubplot:xlabel='yr_built', ylabel='Count'>
```

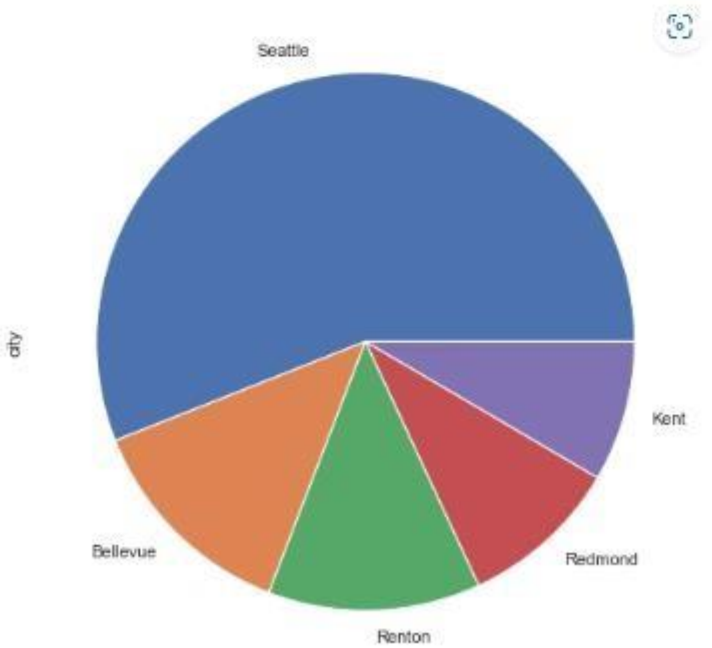



```
In [119]: sns.histplot(df.yr_renovated, kde=True)
Out[119]: <AxesSubplot:xlabel='yr_renovated', ylabel='Count'>
```



PIECHART:

```
In [126]: fig = plt.figure(figsize=(12, 8))
          # Top 5 cities
          df.city.value_counts().head(5).plot.pie()

Out[126]: <AxesSubplot:ylabel='city'>
```



X, y Split

Splitting the data into X and y chunks

In [121]:
```
x=df[['sqft_living','yr_built','yr_renovated','bedrooms','bathrooms']]
y=df['price']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

TESTING AND TRAINING THE MODEL:

In [122]:
```
x_train
```

Out[122]:

	sqft_living	yr_built	yr_renovated	bedrooms	bathrooms
1144	800	1918	0	2	1.00
36	800	1944	0	2	1.00
706	2240	1983	2009	4	2.50
1559	1250	1949	0	3	1.00
1349	2330	1941	1998	4	2.00
...	...	...	...	...	...
1033	1270	2007	0	3	1.50
3264	970	1956	2001	2	1.00
1653	2080	1987	2000	5	2.75
2607	3070	1950	1983	4	2.50
2732	1700	1977	2004	3	1.75

3680 rows × 5 columns

In [123]:
```
x_test
```

Out[123]:

	sqft_living	yr_built	yr_renovated	bedrooms	bathrooms
991	2090	2002	0	3	2.50
2824	2640	1987	2000	4	2.50
1906	650	1967	0	1	1.00
1471	2510	1960	2012	4	2.00
1813	2790	1985	0	4	3.50
...	...	...	...	...	...
1533	1470	1958	1972	3	1.50
463	998	2007	0	3	2.25
4415	1370	1964	0	3	2.00
1927	1540	2011	0	3	3.25
2477	710	1943	2002	2	1.00

920 rows × 5 columns

In [124]:
```
y_train
```

Out[124]:
```
1144     373500.0
36       440000.0
706      592500.0
1559     155000.0
1349     344950.0
           ...
1033     440000.0
3264     210000.0
1653     538888.0
2607    1920000.0
2732     475000.0
Name: price, Length: 3680, dtype: float64
```

In [125]:
```
y_test
```

Out[125]:
```
991      289000.0
2824     429900.0
1906     129000.0
1471     600000.0
1813    1298000.0
           ...
1533     264000.0
463      324000.0
4415      83300.0
1927     520000.0
2477     215000.0
Name: price, Length: 920, dtype: float64
```

In [126]:
```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(x_train.shape)
```

```
(3680, 5)
(3680,)
(920, 5)
(3680, 5)
```

In [127]:
```
model = DecisionTreeRegressor()
my_model=model.fit(x_train,y_train)
y_pred=my_model.predict(x_test)
y_pred
```

Out[127]:
```
array([  415000.     ,   600000.     ,   398000.     ,   970500.     ,
        1008000.     ,   230000.     ,   610000.     ,  1240000.     ,
         369990.     ,  1400000.     ,  1010000.     ,   442000.     ,
         435000.     ,   693000.     ,   625000.     ,   167500.     ,
         268971.875  ,   225000.     ,   595000.     ,   650000.     ,
         488000.     ,   755000.     ,   427000.     ,   280500.     ,
         600000.     ,   460000.     ,   580000.     ,   570000.     ,
         275000.     ,   557000.     ,   490000.     ,   405500.     ,
        1200000.     ,   550000.     ,   640000.     ,   465000.     ,
         275000.     ,   560000.     ,   315000.     ,   639500.     ,
         379900.     ,   371000.     ,   395000.     ,   812000.     ,
         473000.     ,   440825.     ,  1127000.     ,   675000.     ,
         386380.     ,   735000.     ,   234975.     ,   475000.     ,
         330000.     ,   268000.     ,   720000.     ,   265000.     ,
         560000.     ,   148612.5    ,   264000.     ,   845000.     ,
         673000.     ,   436500.     ,   900000.     ,   481000.     ,
         587000.     ,   300000.     ,   175000.     ,   176400.     ,
         480000.     ,   749000.     ,   660000.     ,   950000.     ,
         442000.     ,  1160000.     ,   254600.     ,   760000.     ,
```

DECISION TREE:

DECISION TREE

```
In [128]: train_x, test_x, train_y, test_y = train_test_split(x, y, random_state = 0)
          # Define model
          x = DecisionTreeRegressor()
          # Fit model
          x.fit(train_x, train_y)

Out[128]: DecisionTreeRegressor()
```

PREDICTIONS:

```
In [129]: print("Making predictions for the houses:")
          print(test_x)
          print("The predictions are")
          print(x.predict(test_x))

          Making predictions for the houses:
                sqft_living  yr_built  yr_renovated  bedrooms  bathrooms
          991          2090      2002             0         3        2.5
          2824         2640      1987          2000         4        2.5
          1906          650      1967             0         1        1.0
          1471         2510      1960          2012         4        2.0
          1813         2790      1985             0         4        3.5
          ...           ...       ...           ...       ...        ...
          196          2000      1996             0         4        2.5
          4315         1470      1965             0         4        2.0
          4550         1090      1967          2014         3        1.5
          2941         2200      1916             0         5        2.5
          3987         3000      1979          2014         6        3.5

          [1150 rows x 5 columns]
          The predictions are
          [ 415000.  600000.  299000. ... 2199900.  750500.  574950.]
```

```
In [130]: data =  {'sqft_living':[2640],'yr_built':[1987],'yr_renovated':[2000],'bedrooms':[4],'bathrooms':[2.5]}
          new_input_df = pd.DataFrame(data)
          #Showing data frame of the new input
          new_input_df
          print("The predictions price is")
          print(x.predict(new_input_df))

          The predictions price is
          [600000.]
```

MEAN ABSOLUTE ERROR:

In [131]:

```
val_predictions = x.predict(test_x)
#Calculation of Mean Abosulte Error
print(mean_absolute_error(test_y, val_predictions))
```

```
294878.34679630434
```

ACCURACY SCORE:

In [137]:

```
x=model.score(x_test,y_test)
print(-x)
```

```
7.383932187351807
```

Conclusion:

As you can see above , the model can predict the trend of actual house prices very closely the accuracy of the model can be enhanced by training with data.

References:

https://www.kaggle.com/code/emrearslan123/houseprice-prediction/data

THE END