house-price-prediction

January 17, 2026

IMPORTING NECESSARY LIBS

[517]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import drive
drive.mount('/content/drive') # Mount Google Drive
df=pd.read_csv('/content/drive/MyDrive/House Price Prediction Dataset.csv')
df.head()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[517]:
```
   Id  Area  Bedrooms  Bathrooms  Floors  YearBuilt  Location  Condition  \
0   1  1360         5          4       3       1970  Downtown  Excellent
1   2  4272         5          4       3       1958  Downtown  Excellent
2   3  3592         2          2       3       1938  Downtown       Good
3   4   966         4          2       2       1902  Suburban       Fair
4   5  4926         1          4       2       1975  Downtown       Fair

  Garage   Price
0     No  149919
1     No  424998
2     No  266746
3    Yes  244020
4    Yes  636056
```

DATA CLEANING AND PREPROCESSING

[518]:
```
df.shape
#no.of rows=2000 and columns = 10
```

[518]: (2000, 10)

[519]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
```

```
Data columns (total 10 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Id         2000 non-null   int64
 1   Area       2000 non-null   int64
 2   Bedrooms   2000 non-null   int64
 3   Bathrooms  2000 non-null   int64
 4   Floors     2000 non-null   int64
 5   YearBuilt  2000 non-null   int64
 6   Location   2000 non-null   object
 7   Condition  2000 non-null   object
 8   Garage     2000 non-null   object
 9   Price      2000 non-null   int64
dtypes: int64(7), object(3)
memory usage: 156.4+ KB
```

[520]:

```
df.isnull().sum()
#No null value found
```

[520]:

```
Id           0
Area         0
Bedrooms     0
Bathrooms    0
Floors       0
YearBuilt    0
Location     0
Condition    0
Garage       0
Price        0
dtype: int64
```

[521]:

```
df.describe()
```

[521]:

```
                Id         Area     Bedrooms   Bathrooms       Floors  \
count  2000.000000  2000.000000  2000.000000  2000.00000  2000.000000
mean   1000.500000  2786.209500     3.003500     2.55250     1.993500
std     577.494589  1295.146799     1.424606     1.10899     0.809188
min       1.000000   501.000000     1.000000     1.00000     1.000000
25%     500.750000  1653.000000     2.000000     2.00000     1.000000
50%    1000.500000  2833.000000     3.000000     3.00000     2.000000
75%    1500.250000  3887.500000     4.000000     4.00000     3.000000
max    2000.000000  4999.000000     5.000000     4.00000     3.000000

         YearBuilt          Price
count  2000.000000    2000.000000
mean   1961.446000  537676.855000
std      35.926695  276428.845719
```

```
min    1900.000000   50005.000000
25%    1930.000000  300098.000000
50%    1961.000000  539254.000000
75%    1993.000000  780086.000000
max    2023.000000  999656.000000
```

[522]:
```
df.duplicated().sum()
#No duplicate value found
```

[522]:
```
np.int64(0)
```

[523]:
```
df.columns
```

[523]:
```
Index(['Id', 'Area', 'Bedrooms', 'Bathrooms', 'Floors', 'YearBuilt',
       'Location', 'Condition', 'Garage', 'Price'],
      dtype='object')
```

REMOVING UNNECESSARY COLUMNS

[524]:
```
df=df.drop(columns=[ 'Bedrooms', 'Bathrooms','Id'])
df.head()
```

[524]:

	Area	Floors	YearBuilt	Location	Condition	Garage	Price
0	1360	3	1970	Downtown	Excellent	No	149919
1	4272	3	1958	Downtown	Excellent	No	424998
2	3592	3	1938	Downtown	Good	No	266746
3	966	2	1902	Suburban	Fair	Yes	244020
4	4926	2	1975	Downtown	Fair	Yes	636056

[525]:
```
df['Area'].unique()
```

[525]:
```
array([1360, 4272, 3592, …,  865, 2174, 4062])
```

[526]:
```
df['Location'].unique()
```

[526]:
```
array(['Downtown', 'Suburban', 'Urban', 'Rural'], dtype=object)
```

[527]:
```
df['Location'].isnull().sum()
```

[527]:
```
np.int64(0)
```

[528]:
```
df['Floors'].unique()
```

[528]:
```
array([3, 2, 1])
```

[529]:
```
df['Garage'].unique()
```

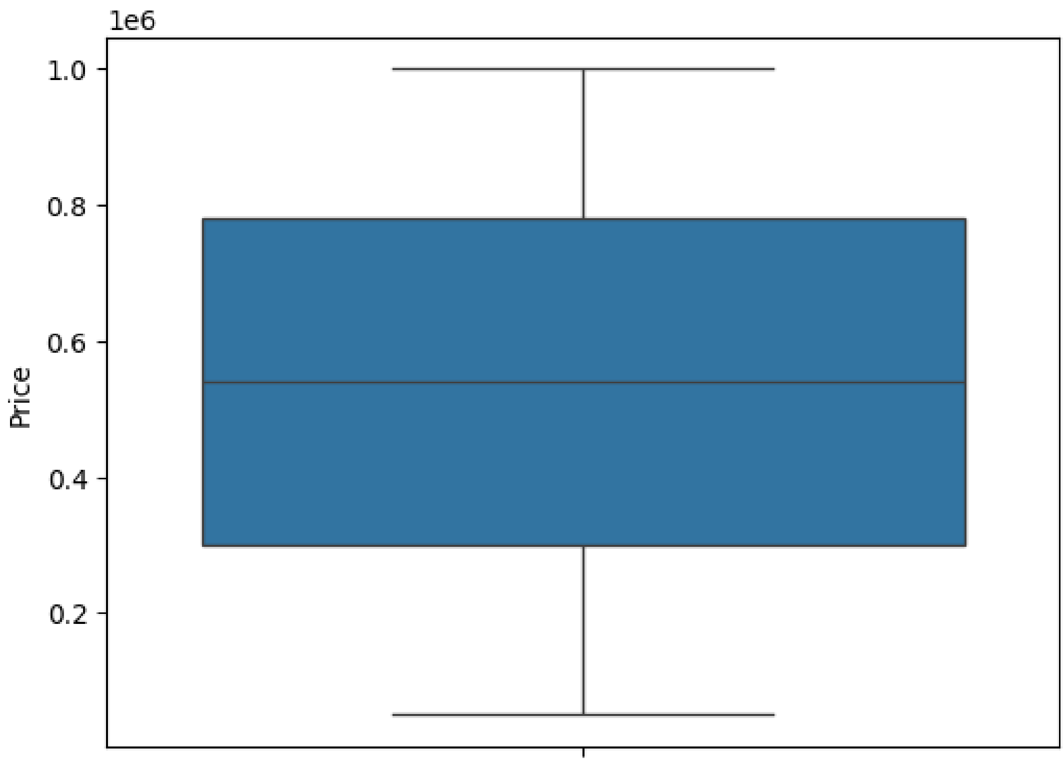
[529]:
```
array(['No', 'Yes'], dtype=object)
```

[530]:
```
df['YearBuilt'].unique()
```

[530]:
```
array([1970, 1958, 1938, 1902, 1975, 1906, 1948, 1925, 1932, 2000, 1947,
       1978, 1901, 2004, 1931, 1903, 1919, 2013, 2016, 1935, 1927, 1976,
       1900, 1959, 1955, 1934, 2011, 1929, 1953, 2020, 1954, 1988, 1979,
       1957, 1982, 1964, 1968, 1950, 1921, 1987, 2006, 2008, 2015, 1952,
       1999, 1967, 1951, 1981, 1949, 1940, 1917, 1965, 1920, 1943, 2002,
       1946, 1928, 1989, 1984, 1916, 1930, 2014, 1972, 1994, 1977, 2009,
       1913, 1996, 1998, 2010, 1983, 2022, 1915, 1911, 2018, 1904, 1980,
       2021, 2005, 1973, 1942, 1944, 1908, 1961, 1956, 1924, 1914, 1905,
       2019, 1941, 1992, 1974, 1963, 2001, 1991, 1936, 1907, 1997, 2007,
       2017, 1966, 1945, 1912, 1986, 1960, 1995, 1933, 1969, 1923, 2012,
       1910, 2003, 1993, 2023, 1918, 1971, 1926, 1939, 1922, 1937, 1909,
       1990, 1962, 1985])
```

[531]:
```
sns.boxplot(df['Price'])
```

[531]: <Axes: ylabel='Price'>



[532]:
```
df['Price'].max()
```

[532]: 999656

[533]:
```
df['Price'].min()
```

[533]:
```
50005
```

[534]:
```
df['Price'].mean()
```

[534]:
```
np.float64(537676.855)
```

DATA EXTRACTION

[535]:
```
x=df[['Area', 'Floors', 'YearBuilt', 'Location', 'Condition', 'Garage']]
y=df['Price']
x
```

[535]:
```
      Area  Floors  YearBuilt  Location  Condition Garage
0     1360       3       1970  Downtown  Excellent     No
1     4272       3       1958  Downtown  Excellent     No
2     3592       3       1938  Downtown       Good     No
3      966       2       1902  Suburban       Fair    Yes
4     4926       2       1975  Downtown       Fair    Yes
…      …      …         …         …        …      …
1995  4994       3       1923  Suburban       Poor     No
1996  3046       1       2019  Suburban       Poor    Yes
1997  1062       2       1903     Rural       Poor     No
1998  4062       2       1936     Urban  Excellent    Yes
1999  2989       3       1903  Suburban       Fair     No

[2000 rows x 6 columns]
```

[536]:
```
y
```

[536]:
```
0       149919
1       424998
2       266746
3       244020
4       636056
         …
1995    295620
1996    580929
1997    476925
1998    161119
1999    482525
Name: Price, Length: 2000, dtype: int64
```

APPLYING TRAIN SPLIT

[537]:
```
#IMPORTING LIBRARY
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
x_train
```

[537]:

```
      Area  Floors  YearBuilt  Location  Condition Garage
368   4230       3       2018  Suburban       Good     No
488   2714       1       1966     Urban       Good    Yes
657   3378       3       1998     Rural  Excellent     No
1699  3041       2       1987  Downtown       Fair     No
338   1581       3       2006     Rural  Excellent     No
…     …     …        …      …         …    …
810   4518       1       1998     Urban  Excellent     No
1560  4068       2       1956     Urban  Excellent     No
1843  3767       1       1901  Downtown       Fair     No
686   3783       2       1961  Suburban       Good    Yes
887   2354       3       2007  Suburban  Excellent     No

[1400 rows x 6 columns]
```

[538]:

```
x_test
```

[538]:

```
      Area  Floors  YearBuilt  Location  Condition Garage
1180  2734       3       2009  Suburban  Excellent     No
1271  1325       2       2008     Rural       Poor     No
1631  4328       2       1970  Suburban  Excellent    Yes
883   1705       1       1985     Rural       Fair    Yes
946   4650       2       1923     Rural       Fair    Yes
…     …     …        …      …         …    …
1711  1786       1       1945     Urban       Fair     No
1973  4526       3       1936     Rural       Fair     No
728    503       3       2012  Suburban       Poor     No
613   1858       1       2013     Rural       Good     No
1007  1804       1       1991     Urban       Good    Yes

[600 rows x 6 columns]
```

ENCODING NOMINAL DATA

[539]:

```
from sklearn.preprocessing import OneHotEncoder,StandardScaler
ohe=OneHotEncoder()
ohe.fit(x[['Location','Garage']])
```

[539]:

```
OneHotEncoder()
```

ORDINAL ENCODING

[540]:

```
from sklearn.preprocessing import OrdinalEncoder
oe=OrdinalEncoder()
oe.fit(x[['Condition']])
```

[540]: OrdinalEncoder()

SCALING DATA

[541]:
```
scaler=StandardScaler()
scaler.fit(x[['Area']])
```

[541]: StandardScaler()

[542]:
```
df['Condition'].unique()
```

[542]: array(['Excellent', 'Good', 'Fair', 'Poor'], dtype=object)

COLUMN TRANSFORMATION

[543]:
```
from sklearn.compose import make_column_transformer
column_transform=make_column_transformer((OneHotEncoder(categories=ohe.
 ↪categories_),['Location','Garage']),
                                         ␣
 ↪(OrdinalEncoder(categories=[['Excellent', 'Good', 'Fair',␣
 ↪'Poor']]),['Condition']),
                                         (StandardScaler(),['Area']),
                                         remainder='passthrough')
column_transform
```

[543]:
```
ColumnTransformer(remainder='passthrough',
                  transformers=[('onehotencoder',
                                 OneHotEncoder(categories=[array(['Downtown',
'Rural', 'Suburban', 'Urban'], dtype=object),
                                                           array(['No', 'Yes'],
dtype=object)]),
                                 ['Location', 'Garage']),
                                ('ordinalencoder',
                                 OrdinalEncoder(categories=[['Excellent',
                                                             'Good', 'Fair',
                                                             'Poor']]),
                                 ['Condition']),
                                ('standardscaler', StandardScaler(), ['Area'])])
```

[544]:
```
x_train_trans=column_transform.fit_transform(x_train)
x_test_trans=column_transform.transform(x_test)
```

MODELING

[545]:
```
#LINEAR REGRESSION
from sklearn.linear_model import LinearRegression

lr=LinearRegression()
```

```
#lr.fit(x_train_trans,y_train)

#TREE BASED REGRESSION
from sklearn.tree import DecisionTreeRegressor
dt=DecisionTreeRegressor()
#dt.fit(x_train_trans,y_train)

#DISTANCE BASED MODELS
from sklearn.neighbors import KNeighborsRegressor
knn=KNeighborsRegressor()
#knn.fit(x_train_trans,y_train)

#BOOSTING
from lightgbm import LGBMRegressor
lgb=LGBMRegressor()
lgb
```

[545]: LGBMRegressor()

Linear Regression: It imports the LinearRegression model from sklearn.linear_model, creates an instance named lr, and then trains (fits) this model using your transformed training features (x_train_trans) and the corresponding training prices (y_train). Decision Tree Regressor: Similarly, it imports DecisionTreeRegressor from sklearn.tree, creates an instance dt, and fits it to the same training data. Decision trees are non-linear models that learn decision rules from data. K-Neighbors Regressor: This part imports KNeighborsRegressor from sklearn.neighbors, creates an instance knn, and fits it. This is a distance-based model that predicts the target based on the 'k' nearest data points in the training set. LGBM Regressor (Light Gradient Boosting Machine): Finally, it imports LGBMRegressor from lightgbm and creates an instance lgb. LightGBM is a gradient boosting framework that uses tree-based learning algorithms. However, this specific line only initializes the lgb object; it does not train (fit) the lgb model in this cell.

[546]:
```
regressors={
    'LR':lr,
    'DT':dt,
    'KNN':knn,
    'LGB':lgb
}
```

This code block creates a Python dictionary called regressors. It's essentially a container to store the different machine learning models you've initialized earlier, making them easily accessible using simple string keys.

'LR' (Linear Regression) is mapped to the lr model instance. 'DT' (Decision Tree Regressor) is mapped to the dt model instance. 'KNN' (K-Neighbors Regressor) is mapped to the knn model instance. 'LGB' (LightGBM Regressor) is mapped to the lgb model instance. This dictionary structure is very useful for iterating through your models, making predictions with each, and evaluating their performance.

TRAINING THE REGRESSOR

[547]:
```python
def train_regressor(regressor,x_train,y_train,x_test,y_test,name):
  regressor.fit(x_train,y_train)
  regressor
  y_pred=regressor.predict(x_test)
  y_pred
```

CALCULATING METRICS

[548]:
```python
#IMPORTING LIB FOR METRICS
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error

metrics_results = {}

for name, regressor in regressors.items():
    print(f"Evaluating {name}:")
    # Fit the regressor before making predictions
    regressor.fit(x_train_trans, y_train)

    # Make predictions on the transformed test set
    y_pred = regressor.predict(x_test_trans)

    # Calculate metrics
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    metrics_results[name] = {
        'MSE': mse,
        'MAE': mae,
        'R2': r2
    }
    print(f"  MSE: {mse:.2f}")
    print(f"  MAE: {mae:.2f}")
    print(f"  R2 Score: {r2:.2f}")
    print("\n")

# Optional: Display all results at once
# import pandas as pd
# print(pd.DataFrame(metrics_results).T)
```

```
Evaluating LR:
  MSE: 73973479746.08
  MAE: 237431.98
  R2 Score: 0.00



Evaluating DT:
  MSE: 154873056519.03
```

```
  MAE: 325111.43
  R2 Score: -1.09


Evaluating KNN:
  MSE: 91532710459.61
  MAE: 255885.91
  R2 Score: -0.24


Evaluating LGB:
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.000181 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 400
[LightGBM] [Info] Number of data points in the train set: 1400, number of used
features: 10
[LightGBM] [Info] Start training from score 533118.275000
  MSE: 86904992222.44
  MAE: 251961.90
  R2 Score: -0.17



/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2739:
UserWarning: X does not have valid feature names, but LGBMRegressor was fitted
with feature names
  warnings.warn(
```

This code block is crucial for evaluating the performance of your trained regression models. Here's what it does step-by-step:

Import Metrics Libraries: It starts by importing r2_score, mean_absolute_error, and mean_squared_error from sklearn.metrics. These functions are used to quantify how well your models perform.

Initialize metrics_results Dictionary: An empty dictionary named metrics_results is created. This dictionary will store the calculated evaluation metrics for each of your models.

Iterate Through Models: The code then loops through each model stored in the regressors dictionary (which you defined earlier, containing lr, dt, knn, and lgb). In each iteration, name gets the model's key (e.g., 'LR', 'DT') and regressor gets the actual trained model object.

Make Predictions: Inside the loop, y_pred = regressor.predict(x_test_trans) uses the current trained regressor to make predictions on your transformed test features (x_test_trans). y_pred will be an array of predicted house prices.

Calculate Metrics:

mse = mean_squared_error(y_test, y_pred): Calculates the Mean Squared Error, which measures the average of the squares of the errors. Larger errors are penalized more heavily. mae = mean_absolute_error(y_test, y_pred): Calculates the Mean Absolute Error, which is the average of the absolute differences between the actual and predicted values. It gives a more direct sense of the average prediction error. r2 = r2_score(y_test, y_pred): Calculates the R-squared (coefficient of determination) score. This indicates the proportion of the variance in the target variable that your model can explain. A higher R2 score (closer to 1) means a better fit. Store and Print Results: The calculated mse, mae, and r2 for the current model are stored in the metrics_results dictionary, and then printed to the console, formatted to two decimal places, to give you an immediate overview of each model's performance.

MAKING PIPELINES

[549]:
```
import sklearn.pipeline
from sklearn.pipeline import make_pipeline
pipe1=make_pipeline(column_transform,lr)
pipe2=make_pipeline(column_transform,dt)
pipe3=make_pipeline(column_transform,knn)
pipe4=make_pipeline(column_transform,lgb)
```

FITTING THESE PIPELINES

[550]:
```
pipe1.fit(x_train,y_train)
pipe2.fit(x_train,y_train)
pipe3.fit(x_train,y_train)
pipe4.fit(x_train,y_train)
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000075 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 400
[LightGBM] [Info] Number of data points in the train set: 1400, number of used
features: 10
[LightGBM] [Info] Start training from score 533118.275000
```

```
/usr/local/lib/python3.12/dist-
packages/sklearn/compose/_column_transformer.py:1667: FutureWarning:
The format of the columns of the 'remainder' transformer in
ColumnTransformer.transformers_ will change in version 1.7 to match the format
of the other transformers.
At the moment the remainder columns are stored as indices (of type int). With
the same ColumnTransformer configuration, in the future they will be stored as
column names (of type str).
To use the new behavior now and suppress this warning, use
ColumnTransformer(force_int_remainder_cols=False).

  warnings.warn(
```

[550]:
```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('onehotencoder',
```

```
OneHotEncoder(categories=[array(['Downtown', 'Rural', 'Suburban', 'Urban'],
dtype=object),
array(['No', 'Yes'], dtype=object)]),
                                                  ['Location', 'Garage']),
                                                 ('ordinalencoder',
OrdinalEncoder(categories=[['Excellent',
'Good',
'Fair',
'Poor']]),
                                                  ['Condition']),
                                                 ('standardscaler',
                                                  StandardScaler(),
                                                  ['Area'])])),
                ('lgbmregressor', LGBMRegressor())])
```

[551]:
```
y_pred1=pipe1.predict(x_test)
y_pred2=pipe2.predict(x_test)
y_pred3=pipe3.predict(x_test)
y_pred4=pipe4.predict(x_test)
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2739:
UserWarning: X does not have valid feature names, but LGBMRegressor was fitted
with feature names
  warnings.warn(
```

Essentially, for each pipeline, the x_test data first goes through the column_transform (applying one-hot encoding, ordinal encoding, and scaling), and then the preprocessed data is fed into the respective regression model to produce price predictions.

CHECKING R2_SCORE

[552]:
```
print(r2_score(y_test,y_pred1))
print(r2_score(y_test,y_pred2))
print(r2_score(y_test,y_pred3))
print(r2_score(y_test,y_pred4))
```

```
0.0006983615451935377
-1.146671395077441
-0.23650783832897426
-0.1739923742384868
```

[553]:
```
scores=[]
for i in range(1,10000):
  x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
↪2,random_state=i)
dt=DecisionTreeRegressor()
pipe=make_pipeline(column_transform,dt)
pipe.fit(x_train,y_train)
```

```
y_pred=pipe.predict(x_test)
scores.append(r2_score(y_test,y_pred))
```

[554]:
```
np.argmax(scores)
```

[554]: np.int64(0)

[555]:
```
scores[np.argmax(scores)]
```

[555]: -1.179504350151821

np.argmax(scores): This function from the NumPy library finds the index of the maximum value within the scores list. The scores list contains the R2 scores calculated during multiple iterations of training and evaluating the Decision Tree Regressor with different random train-test splits. scores[…]: Once np.argmax(scores) returns the index of the best score, this index is then used to access the scores list itself, effectively retrieving the actual maximum R2 score that was found.

[556]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
 ↪1,random_state=np.argmax(scores))
dt = DecisionTreeRegressor()
pipe=make_pipeline(column_transform,dt)
pipe.fit(x_train,y_train)
y_pred=pipe.predict(x_test)
r2_score(y_test,y_pred)
```

[556]: -1.0385103090629602

[557]:
```
pipe.predict(pd.DataFrame([[1360, 3, 1970, 'Downtown', 'Excellent', 'No']],␣
 ↪columns=['Area', 'Floors', 'YearBuilt', 'Location', 'Condition', 'Garage']))
```

[557]: array([149919.])