Bangalore House Price Prediction

Intrduction

In this project, we will use the Bengaluru House price data from Kaggle . The objective is to develop a machine learning model for predicting home prices in Banglore

Dataset Description:

The prediction of house prices is based on the following variables:

- **Location**: The neighborhood or area where the house is located.
- **Total Sq. Ft:** The total square footage of the property.
- **Bedrooms:** The number of bedrooms in the house in Terms of BHK
- **Bathrooms**: The number of bathrooms in the house.
- **Price**: The target variable, which is the price of the house.

import libraries:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
```

load the dataset using pandas

```
df1=pd.read_csv("/kaggle/input/bengaluru-house-price-data/Bengaluru_House_Data.csv")
```

Dropping unnecessary data is an essential task that holds significant importance in the modeling process. This step involves removing irrelevant or redundant features from the dataset, ensuring that the model focuses only on the most relevant and informative data points

```
df2=df1.drop(['area_type','availability','society','balcony'],axis=1)
```

Data preprocessing:

Checking for missing values:

To find missing values, we use a method called **isna()**. Then, we use another method called **sum()** to count how many missing values are in each column.

In this case, we found that :

- location has 1 missing value
- size has 16 missing values
- total_sqft has 0 missing values
- bath has 73 missing values
- price has 0 missing values

```
df2.isna().sum()
```

```
location       1
size          16
total_sqft     0
bath          73
price          0
dtype: int64
```

In our dataset, there are only a few missing values To address this, we've opted to handle them by simply removing the affected rows with **dropna()** method

```
df3 = df2.dropna()
df3.isnull().sum()
```

```
location      0
size          0
total_sqft    0
bath          0
price         0
dtype: int64
```

To understand the different categories present in the 'size' column of the dataset, we use the .unique() method. This method returns an array containing all the unique values found in that column. In the 'size' column, we encounter values such as '2 BHK', '4 Bedroom', '3 BHK', and so

forth. Some entries indicate the number of bedrooms ('**Bedroom**'), while others specify the number of bedrooms along with a hall and a kitchen ('**BHK**'). This insight aids in comprehending the various types of property configurations represented in the dataset.

```
df3['size'].unique()
```

```
array(['2 BHK', '4 Bedroom', '3 BHK', '4 BHK', '6 Bedroom', '3 Bedroom',
       '1 BHK', '1 RK', '1 Bedroom', '8 Bedroom', '2 Bedroom',
       '7 Bedroom', '5 BHK', '7 BHK', '6 BHK', '5 Bedroom', '11 BHK',
       '9 BHK', '9 Bedroom', '27 BHK', '10 Bedroom', '11 Bedroom',
       '10 BHK', '19 BHK', '16 BHK', '43 Bedroom', '14 BHK', '8 BHK',
       '12 Bedroom', '13 BHK', '18 Bedroom'], dtype=object)
```

Feature Engineering

We created a new column 'bhk' which contains the number of bedrooms (BHK) for each property by extracting this information from the 'size' column.

```
df3['bhk']= df3['size'].apply(lambda x : int(x.split(' ')[0]))
```

```
df3.head()
```

	location	size	total_sqft	bath	price	bhk
0	Electronic City Phase II	2 BHK	1056	2.0	39.07	2
1	Chikka Tirupathi	4 Bedroom	2600	5.0	120.00	4
2	Uttarahalli	3 BHK	1440	2.0	62.00	3
3	Lingadheeranahalli	3 BHK	1521	3.0	95.00	3
4	Kothanur	2 BHK	1200	2.0	51.00	2

Check the unique value in total_sqft:

The result indicates that the 'total_sqft' column contains both single numerical values and ranges of values. This suggests that some entries represent a range of square footage rather

than a single value. Handling these range values will be important for accurate analysis.

```
df3.total_sqft.unique()
```

```
array(['1056', '2600', '1440', ..., '1133 - 1384', '774', '4689'],
      dtype=object)
```

The function **isfloat** attempts to convert the input x into a float .If the conversion is successful, it returns True, otherwise, it returns False within the except block.

```
def isfloat(x):
    try:
        float(x)
    except:
        return False
    return True
```

We select rows where the 'total_sqft' column contains values that could not be successfully converted to floats.

```
df3[~df3['total_sqft'].apply(isfloat)].head()
```

	location	size	total_sqft	bath	price	bhk
30	Yelahanka	4 BHK	2100 - 2850	4.0	186.000	4
122	Hebbal	4 BHK	3067 - 8156	4.0	477.000	4
137	8th Phase JP Nagar	2 BHK	1042 - 1105	2.0	54.005	2
165	Sarjapur	2 BHK	1145 - 1340	2.0	43.490	2
188	KR Puram	2 BHK	1015 - 1540	2.0	56.800	2

This function effectively handles both single numerical values and ranges of values in the 'total_sqft' column, returning either the average of a range or a float value if possible, and None if conversion fails

```
def convert_sqft_to_num(x):
    tokens = x.split('-')
    if len(tokens) == 2:
        return(float(tokens[0])+float(tokens[1]))/2
    try:
        return float(x)
    except:
        return None
```

We applied the convert_sqft_to_num function to each element in the 'total_sqft' column using the .apply() method to convert the values in the 'total_sqft' column to numerical format.

```
df4= df3.copy()
df4['total_sqft']= df4['total_sqft'].apply(convert_sqft_to_num)
df4.head(3)
```

	location	size	total_sqft	bath	price	bhk
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3

we calculate the price per square foot for each property in the DataFrame df5 and assigns the result to a new column named 'price_per_sqft'. It multiplies the price by 100,000 to convert it to a per square foot basis and then divides it by the total square footage of the property.

```
df5 = df4.copy()
df5['price_per_sqft'] = df5['price']*100000/df5['total_sqft']
df5.head()
```

	location	size	total_sqft	bath	price	bhk	price_per_sqft
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2	3699.810606
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4	4615.384615
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3	4305.555556
3	Lingadheeranahalli	3 BHK	1521.0	3.0	95.00	3	6245.890861
4	Kothanur	2 BHK	1200.0	2.0	51.00	2	4250.000000

We grouped the data by location using groupby('location'), then counted the number of properties available in each location using .agg('count'). By subsequently sorting these statistics in descending order with sort_values(ascending=False), we obtain the count of properties for each location, starting with the location having the highest number of properties. This provides a clear insight into the distribution of properties by location, highlighting the most densely populated areas in terms of property listings.

```
df5.location = df5.location.apply(lambda x : x.strip())

location_stats = df5.groupby('location')['location'].agg('count').sort_values(ascending = False)
```

location_stats_less_than_10,contains location statistics where the count of properties is less than or equal to 10. In other words, it identifies locations with 10 or fewer properties listed.

```
location_stats_less_than_10 = location_stats[location_stats <=10]
location_stats_less_than_10
```

We update the 'location' column. If a location is found in the location_stats_less_than_10 Series, we replace it with 'other'; otherwise, it remains unchanged

```
df5.location = df5.location.apply(lambda x: 'other' if x in location_stats_less_than_10 else x)
len(df5.location.unique())
```

Outlier Removal

We'll eliminate such outliers by setting our minimum threshold per bedroom to 300 square feet

```
df5[df5.total_sqft/df5.bhk<300].head()
```

	location	size	total_sqft	bath	price	bhk	price_per_sqft
9	other	6 Bedroom	1020.0	6.0	370.0	6	36274.509804
45	HSR Layout	8 Bedroom	600.0	9.0	200.0	8	33333.333333
58	Murugeshpalya	6 Bedroom	1407.0	4.0	150.0	6	10660.980810
68	Devarachikkanahalli	8 Bedroom	1350.0	7.0	85.0	8	6296.296296
70	other	3 Bedroom	500.0	3.0	100.0	3	20000.000000

Here, we observe that the minimum price per square foot is 267 Rs/sqft, while the maximum is 12,000,000 Rs/sqft, indicating a significant variation in property prices. To address this, we will remove outliers on a per-location basis using the mean and one standard deviation.

```
def remove_pps_outliers(df):
    df_out = pd.DataFrame()
    for key, subdf in df.groupby('location'):
        m = np.mean(subdf.price_per_sqft)
        st = np.std(subdf.price_per_sqft)
        reduced_df = subdf[(subdf.price_per_sqft>(m-st)) & (subdf.price_per_sqft<=(m+st))]
        df_out = pd.concat([df_out,reduced_df],ignore_index=True)
    return df_out
df7 = remove_pps_outliers(df6)
df7.shape
```

We can now eliminate those 2 BHK apartments whose price per square foot is lower than the mean price per square foot of 1 BHK apartments.

```
def remove_bhk_outliers(df):
    exclude_indices = np.array([])
    for location, location_df in df.groupby('location'):
        bhk_stats = {}
        for bhk, bhk_df in location_df.groupby('bhk'):
            bhk_stats[bhk] = {
                'mean': np.mean(bhk_df.price_per_sqft),
                'std': np.std(bhk_df.price_per_sqft),
                'count': bhk_df.shape[0]
            }
        for bhk, bhk_df in location_df.groupby('bhk'):
            stats = bhk_stats.get(bhk-1)
            if stats and stats['count']>5:
                exclude_indices = np.append(exclude_indices, bhk_df[bhk_df.price_per_sqft<(stats['mean'])].index.values)
    return df.drop(exclude_indices,axis='index')
df8 = remove_bhk_outliers(df7)
# df8 = df7.copy()
df8.shape
```

Having two or more bathrooms than the number of bedrooms in a home is uncommon. We filter the DataFrame df8 to include only those rows where the number of bathrooms (bath) is

greater than the number of bedrooms (bhk) plus 2.

```
df8[df8.bath>df8.bhk+2]
```

	location	size	total_sqft	bath	price	bhk	price_per_sqft
1626	Chikkabanavar	4 Bedroom	2460.0	7.0	80.0	4	3252.032520
5238	Nagasandra	4 Bedroom	7000.0	8.0	450.0	4	6428.571429
6711	Thanisandra	3 BHK	1806.0	6.0	116.0	3	6423.034330
8411	other	6 BHK	11338.0	9.0	1000.0	6	8819.897689

We filter the DataFrame df9 to include only those rows where the number of bathrooms (bath) is less than the number of bedrooms (bhk) plus 2.

```
df9 = df8[df8.bath<df8.bhk+2]
df9.shape
```

We dropped the columns 'price_per_sqft' and 'bhk'. We initially created these two columns for analyzing the data. However, we already have the 'size' column for the number of bedrooms and the 'price' column for the property price.

```
df10 = df9.drop(['size','price_per_sqft'],axis='columns')
df10.head(100)
```

	location	total_sqft	bath	price	bhk
0	1st Block Jayanagar	2850.0	4.0	428.0	4
1	1st Block Jayanagar	1630.0	3.0	194.0	3
2	1st Block Jayanagar	1875.0	2.0	235.0	3
3	1st Block Jayanagar	1200.0	2.0	130.0	3
4	1st Block Jayanagar	1235.0	2.0	148.0	2
...	...	...	...	...	...
106	7th Phase JP Nagar	1180.0	2.0	72.0	2
110	7th Phase JP Nagar	1400.0	3.0	115.0	3
111	7th Phase JP Nagar	1270.0	2.0	83.0	2
113	7th Phase JP Nagar	2503.0	4.0	188.0	4
114	7th Phase JP Nagar	2200.0	3.0	190.0	3

This code uses one-hot encoding to create dummy variables for the 'location' column in the DataFrame df10. Each unique location is transformed into a binary column, where 1 indicates the presence of the location and 0 indicates absence.

```
dummies = pd.get_dummies(df10.location,dtype=int)
dummies.head(3)
```

	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	5th Phase JP Nagar	6th Phase JP Nagar	7th Phase JP Nagar	8th Phase JP Nagar	9th Phase JP Nagar	...	Vishveshwarya Layout	Vishwapriya Layout	Vittasandra	Whitefield	Yelachenahalli	Yelahanka	Yelahanka New Town	Yelenahalli	Yeshwanthpur
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0

3 rows × 242 columns

we concatenate the DataFrame df10 with the dummy variables created from the 'location' column .It drops the 'other' column then combines the remaining dummy variables with df10 along the columns axis to create a new DataFrame df11.

```
df11 = pd.concat([df10,dummies.drop('other',axis='columns')],axis='columns')
df11.head()
```

[56]:

	location	total_sqft	bath	price	bhk	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	...	Vijayanagar	Vishveshwarya Layout	Vishwapriya Layout	Vittasandra	Whitefield	Yelachenahalli	Yelahanka	Yelahanka New Town	Yelenahalli
0	1st Block Jayanagar	2850.0	4.0	428.0	4	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0
1	1st Block Jayanagar	1630.0	3.0	194.0	3	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0
2	1st Block Jayanagar	1875.0	2.0	235.0	3	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0
3	1st Block Jayanagar	1200.0	2.0	130.0	3	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0
4	1st Block Jayanagar	1235.0	2.0	148.0	2	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0

We dropped the 'location' column.

:

	total_sqft	bath	price	bhk	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	5th Phase JP Nagar	...	Vijayanagar	Vishveshwarya Layout	Vishwapriya Layout	Vittasandra	Whitefield	Yelachenahalli	Yelahanka	Yelahanka New Town	Yelenahalli	Ye
0	2850.0	4.0	428.0	4	1	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
1	1630.0	3.0	194.0	3	1	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	

2 rows × 245 columns

Build a Model

we assigned the DataFrame df12 excluding the 'price' column to the variable X.

```
X = df12.drop(['price'],axis='columns')
```

we assigned the 'price' column of the DataFrame df12 to the variable y.

```
y=df12.price
y.head()
```

```
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.2,random_state=10)
```

Model Building:

We utilize **linear regression** for modeling, and the model's performance is evaluated by calculating the coefficient of determination (R^2 score) on a test dataset using the score() method. In this particular instance, the R^2 score obtained is 0.85. The R^2 score measures the proportion of the variance in the dependent variable that is predictable from the independent variables.

```
from sklearn.linear_model import LinearRegression
lr_clf = LinearRegression()
lr_clf.fit(X_train,y_train)
lr_clf.score(X_test,y_test)
```

```
0.8452277697874291
```

Conclusion:

In this project, we've navigated through various stages of a machine learning project aimed at predicting house prices in Bangalore. Our journey began with thorough data exploration and preprocessing, ensuring our dataset is ready for modeling. We then proceeded to model selection, where we opted for linear regression due to its simplicity and interpretability. After selecting the model, we trained it on our preprocessed data