House Price Prediction

Introduction

House is one of human life's most essential needs. Demand for houses grew rapidly over the years as people's living standards improved. house price prediction is based on cost and sale price. Therefore, the availability of a house price prediction model helps fill up an important information gap and improve the efficiency of the real estate market. The goal is to predict the prices of houses precisely using machine learning techniques.

DataSet:

area_type : Area type. availability :
location : location of House.
size : Size of House.
society :
total_sqft : Square feet of the House bath :
balcony : price : Price of House.

Objectives:

- Prediction of House Price.
- To identify which location having the highest number of restaurants.

In [3]:
```
# import libraries.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import StandardScaler
from scipy.sparse import hstack
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor
```

Exploratory Data Analysis

In [4]:
```
data=pd.read_csv("C:\\Users\\ppheg\\Downloads\\Bangalore  house data.csv")
data
```

Out[4]:

	area_type	availability	location	size	society	total_sqft	bath	balcony	pri
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2.0	1.0	39.

	area_type	availability	location	size	society	total_sqft	bath	balcony	pri
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5.0	3.0	120.
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	NaN	1440	2.0	3.0	62.
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	Soiewre	1521	3.0	1.0	95.
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	NaN	1200	2.0	1.0	51.
...	...	...	...	...	...	...	...	...	
13315	Built-up Area	Ready To Move	Whitefield	5 Bedroom	ArsiaEx	3453	4.0	0.0	231.
13316	Super built-up Area	Ready To Move	Richards Town	4 BHK	NaN	3600	5.0	NaN	400.
13317	Built-up Area	Ready To Move	Raja Rajeshwari Nagar	2 BHK	Mahla T	1141	2.0	1.0	60.
13318	Super built-up Area	18-Jun	Padmanabhanagar	4 BHK	SollyCl	4689	4.0	1.0	488.
13319	Super built-up Area	Ready To Move	Doddathoguru	1 BHK	NaN	550	1.0	1.0	17.

13320 rows × 9 columns

In [5]:
```
#converting total_sqrt to numerical value

numeric_sqrt = []

for pt in data["total_sqft"]:

    pt = re.sub(r'[aA-zZ]+',  ' ',pt)

    pt = pt.replace(" .", "")

    if "-" in pt:
        vals = pt.split("-")
        #taking average of two values
        value = (float(vals[0])+float(vals[1]))/2
        numeric_sqrt.append(value)

    else:
        numeric_sqrt.append(float(pt))


data["total_sqft"] =  numeric_sqrt
```

In [6]:
```
# Shape of Data:
data.shape
```

Out[6]: (13320, 9)

Number of records present in the data: 13320

Number of columns present in the data: 9

In [7]:
```
# information about Data:
data.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       7818 non-null   object 
 5   total_sqft    13320 non-null  float64
 6   bath          13247 non-null  float64
 7   balcony       12711 non-null  float64
 8   price         13320 non-null  float64
dtypes: float64(4), object(5)
memory usage: 936.7+ KB
```

There are 5 categorical variables, 4 numerical variable. And we have 1 numerical target variable Price.

In [8]:
```
data.describe()
```

Out[8]:

	total_sqft	bath	balcony	price
count	13320.000000	13247.000000	12711.000000	13320.000000
mean	1555.971707	2.692610	1.584376	112.565627
std	1238.902448	1.341458	0.817263	148.971674
min	1.000000	1.000000	0.000000	8.000000
25%	1100.000000	2.000000	1.000000	50.000000
50%	1275.000000	2.000000	2.000000	72.000000
75%	1679.250000	3.000000	2.000000	120.000000
max	52272.000000	40.000000	3.000000	3600.000000

In [9]:
```
# Columns Name:
data.columns
```

Out[9]:
```
Index(['area_type', 'availability', 'location', 'size', 'society',
       'total_sqft', 'bath', 'balcony', 'price'],
      dtype='object')
```

Data Cleaning:

In [10]:
```
#check null values
data.isnull().sum()
```

Out[10]:
```
area_type          0
availability       0
location           1
size              16
society         5502
```

```
total_sqft        0
bath             73
balcony         609
price             0
dtype: int64
```

Society and Balcony columns having lot of null values so fill null values by imputation method.

In [11]:
```
# impute missing values with mean value
data['balcony']=data['balcony'].fillna(round(data['balcony'].mean(),1))
data['balcony'][:5]
```

Out[11]:
```
0    1.0
1    3.0
2    3.0
3    1.0
4    1.0
Name: balcony, dtype: float64
```

In [12]:
```
# impute missing values with mean value
data['bath']=data['bath'].fillna(round(data['bath'].mean(),1))
data['bath'][:5]
```

Out[12]:
```
0    2.0
1    5.0
2    2.0
3    3.0
4    2.0
Name: bath, dtype: float64
```

In [13]:
```
# impute missing values in dish liked with "unknown", so that "unknown" also conside
data['society']=data['society'].fillna("unknown")
data['society']
```

Out[13]:
```
0        Coomee
1        Theanmp
2        unknown
3        Soiewre
4        unknown
          ...
13315    ArsiaEx
13316    unknown
13317    Mahla T
13318    SollyCl
13319    unknown
Name: society, Length: 13320, dtype: object
```

In [14]:
```
data.dropna(inplace=True)
```

In [15]:
```
data.isnull().sum()
```

Out[15]:
```
area_type       0
availability    0
location        0
size            0
society         0
total_sqft      0
bath            0
balcony         0
price           0
dtype: int64
```

There are no missing values in any of the column in this dataframe. The dataframe seems to be clean. There is no much effort needed for cleaning.

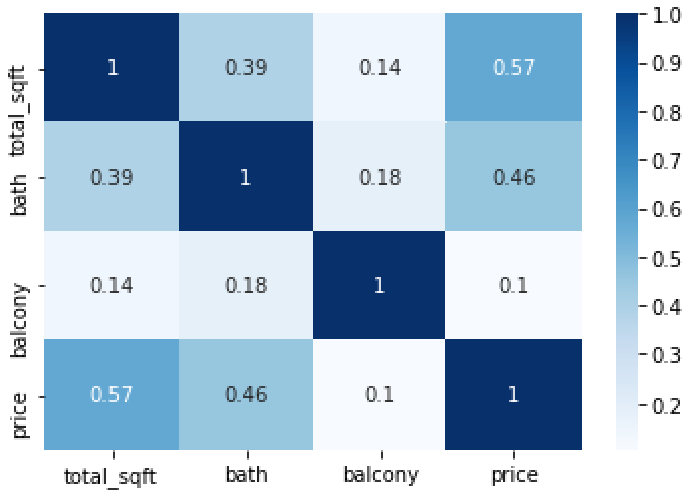
Correlation Analysis:

In [16]:
```
# correlation:
a=data.corr()
a
```

Out[16]:

	total_sqft	bath	balcony	price
total_sqft	1.000000	0.389960	0.137884	0.573469
bath	0.389960	1.000000	0.183847	0.455631
balcony	0.137884	0.183847	1.000000	0.104865
price	0.573469	0.455631	0.104865	1.000000

In [17]:
```
import seaborn as sns
sns.heatmap(a,annot=True,cmap='Blues')
plt.show()
```



Bath and Balcony have moderatly correlated

Analysis of each features:

In [18]:
```
def descriptive(feature):
    """Returns the descriptive statistics of the given feature"""

    descriptive = pd.DataFrame()
    descriptive["minimum"] = [data[feature].min()]
    descriptive["maximum"] = [data[feature].max()]
    descriptive["mean"] = [data[feature].mean()]
    descriptive["median"] = [data[feature].median()]
    descriptive["mode"] = [data[feature].mode()[0]]

    return descriptive
```

In [19]:
```
descriptive("price")
```

Out[19]:

	minimum	maximum	mean	median	mode
0	8.0	3600.0	112.584033	72.0	75.0

mean and median values of price is very low compared to maximum value.

In [20]:
```
descriptive("bath")
```

Out[20]:

	minimum	maximum	mean	median	mode
0	1.0	40.0	2.692618	2.0	2.0

In [21]:
```
descriptive("balcony")
```

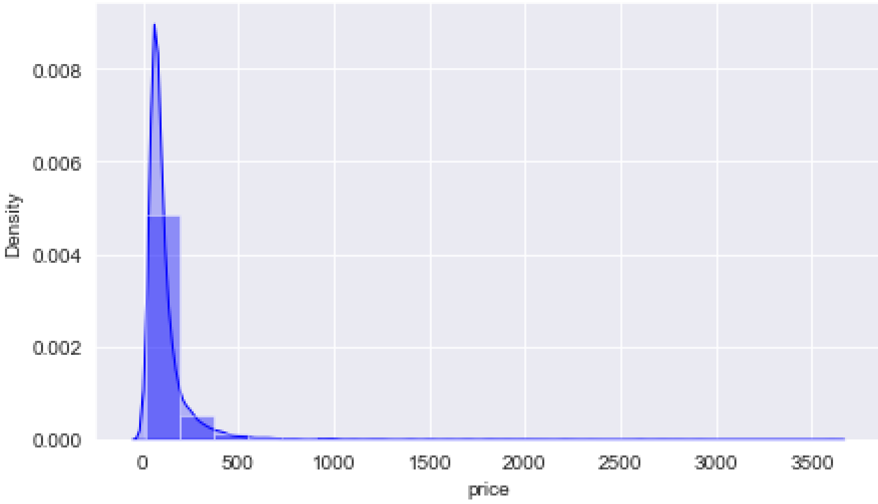
Out[21]:

	minimum	maximum	mean	median	mode
0	0.0	3.0	1.585041	2.0	2.0

In [22]:
```
# Distribution of price of house in Bengalore.
fig = plt.figure(figsize=(7,4))
sns.set_style('darkgrid')
sns.distplot(data['price'], bins = 20,  color= 'blue',kde_kws={"shade": True})
```

```
C:\Users\ppheg\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarn
ing: `distplot` is a deprecated function and will be removed in a future version. Pl
ease adapt your code to use either `displot` (a figure-level function with similar f
lexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[22]: <AxesSubplot:xlabel='price', ylabel='Density'>



Distribution plot of price rightly skewed (positively). There are very few values which are greater than 400. mean value of price is nearly 100.

Data Analysis:

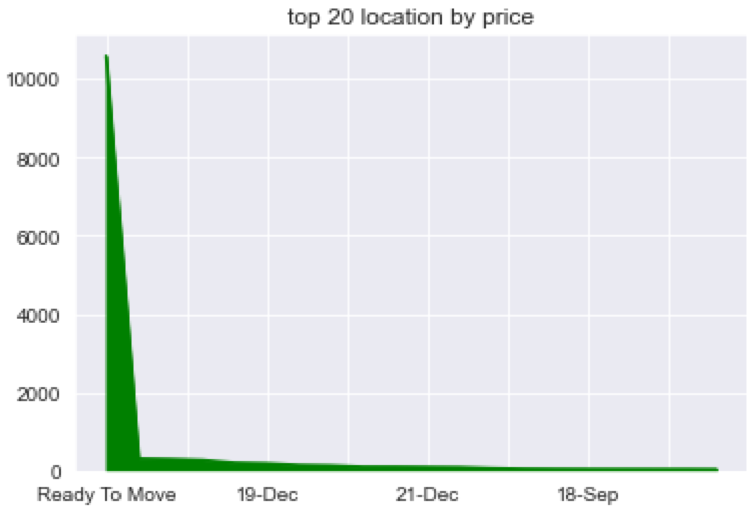
In [23]:
```
plt.figure(figsize=(10,5))
plt.title('area_type vs price\n', fontdict={'color':'red','size':15})
sns.lineplot(x='area_type',y='price', data=data , color='r')
plt.show()
```

plot area type is more price. and carpet area type is less.

In [24]:
```
data.location.value_counts()[:20].plot(kind='barh',color='Purple')
plt.title('top 20 location by price')
plt.show()
```

Whitefield location have highest price.

In [25]:
```
data.availability.value_counts()[:20].plot(kind='area',color='green')
plt.title('top 20 location by price')
plt.show()
```



Ready to move Availability is more.

In [26]:
```
data.groupby(by='size').sum().sort_values('price',ascending=True).head(10)
```

Out[26]:

size	total_sqft	bath	balcony	price
14 BHK	1250.0	15.0	0.0	125.00
18 Bedroom	1200.0	18.0	1.6	200.00
27 BHK	8000.0	27.0	0.0	230.00
13 BHK	5425.0	13.0	0.0	275.00
12 Bedroom	2232.0	6.0	2.0	300.00
11 Bedroom	2400.0	17.0	3.0	320.00
1 RK	6411.5	13.0	6.0	365.59
19 BHK	2000.0	16.0	1.6	490.00
11 BHK	11000.0	21.0	4.6	510.00
16 BHK	10000.0	16.0	1.6	550.00

Here House contains 14 BHK and 15 bath, zero balcony have low price.

Data splitting

In [27]:
```
#train test splitting:
y = data['price']
x = data.drop('price',axis=1)


x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=353)

print("Train size: ",x_train.shape)
print("train size:",y_train.shape)
print("test size:",x_test.shape)
print("test size:",y_test.shape)
```

```
Train size:  (10642, 8)
train size: (10642,)
test size: (2661, 8)
test size: (2661,)
```

Feature Tranformation

one hot encoding for categorical data

id	color
1	red
2	blue
3	green
4	blue

One Hot Encoding

id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

In [28]:

```
enc = OneHotEncoder(handle_unknown='ignore')

enc.fit(x_train["area_type"].values.reshape(-1,1))

x_tr_area = enc.transform(x_train["area_type"].values.reshape(-1,1))
x_te_area = enc.transform(x_test["area_type"].values.reshape(-1,1))

print(x_tr_area.shape)
print(x_te_area.shape)
```

```
(10642, 4)
(2661, 4)
```

In [29]:

```
enc = OneHotEncoder(handle_unknown='ignore')

enc.fit(x_train["availability"].values.reshape(-1,1))

x_tr_availability = enc.transform(x_train["availability"].values.reshape(-1,1))
x_te_availability = enc.transform(x_test["availability"].values.reshape(-1,1))

print(x_tr_availability.shape)
print(x_te_availability.shape)
```

```
(10642, 79)
(2661, 79)
```

In [30]:

```
enc = OneHotEncoder(handle_unknown='ignore')

enc.fit(x_train["location"].values.reshape(-1,1))

x_tr_location = enc.transform(x_train["location"].values.reshape(-1,1))
x_te_location = enc.transform(x_test["location"].values.reshape(-1,1))

print(x_tr_location.shape)
print(x_te_location.shape)
```

```
(10642, 1213)
(2661, 1213)
```

In [31]:

```
enc = OneHotEncoder(handle_unknown='ignore')

enc.fit(x_train["size"].values.reshape(-1,1))

x_tr_size = enc.transform(x_train["size"].values.reshape(-1,1))
x_te_size = enc.transform(x_test["size"].values.reshape(-1,1))

print(x_tr_size.shape)
print(x_te_size.shape)
```

```
(10642, 28)
(2661, 28)
```

In [32]:
```
enc = OneHotEncoder(handle_unknown='ignore')

enc.fit(x_train["society"].values.reshape(-1,1))

x_tr_society = enc.transform(x_train["society"].values.reshape(-1,1))
x_te_society = enc.transform(x_test["society"].values.reshape(-1,1))

print(x_tr_society.shape)
print(x_te_society.shape)
```

```
(10642, 2350)
(2661, 2350)
```

Standardizing numerical features

In [33]:
```
std = StandardScaler()

#finding mean and standrd deviation using train data
std.fit(x_train["total_sqft"].values.reshape(-1,1))

#standardizing train and test data using mean and std calculated using train data
x_train_sqrt = std.transform(x_train["total_sqft"].values.reshape(-1,1))
x_test_sqrt  = std.transform(x_test["total_sqft"].values.reshape(-1,1))

print(x_train_sqrt.shape)
print(x_test_sqrt.shape)
```

```
(10642, 1)
(2661, 1)
```

In [34]:
```
std = StandardScaler()

#finding mean and standrd deviation using train data
std.fit(x_train["bath"].values.reshape(-1,1))

#standardizing train and test data using mean and std calculated using train data
x_train_bath = std.transform(x_train["bath"].values.reshape(-1,1))
x_test_bath = std.transform(x_test["bath"].values.reshape(-1,1))


print(x_train_bath.shape)
print(x_test_bath.shape)
```

```
(10642, 1)
(2661, 1)
```

In [35]:
```
std = StandardScaler()

#finding mean and standrd deviation using train data
std.fit(x_train["balcony"].values.reshape(-1,1))

#standardizing train and test data using mean and std calculated using train data
x_train_balcony = std.transform(x_train["balcony"].values.reshape(-1,1))
x_test_balcony = std.transform(x_test["balcony"].values.reshape(-1,1))


print(x_train_balcony.shape)
print(x_test_balcony.shape)
```

```
(10642, 1)
(2661, 1)
```

In [ ]:

Concatenating all features

In [36]:
```
from scipy.sparse import hstack

train_data = hstack((x_tr_area,x_tr_availability,x_tr_location,x_tr_size,x_tr_societ
test_data =   hstack((x_te_area,x_te_availability,x_te_location,x_te_size,x_te_socie

#Concatenating all features
print("FINAL DATA MATRIX SHAPE IS ........")
print(train_data.shape,y_train.shape)
print(test_data.shape,y_test.shape)
print("*"*100)
```

```
FINAL DATA MATRIX SHAPE IS .......
(10642, 3677) (10642,)
(2661, 3677) (2661,)
*************************************************************************************
***************
```

ML Models:

Linear regression

In [37]:
```
#Linear regression
linear_regression = LinearRegression()
linear_regression.fit(train_data, y_train)
```

Out[37]: LinearRegression()

In [38]:
```
y_1_pred = linear_regression.predict(test_data)
print(r2_score(y_test, y_1_pred))
```

```
0.349703907817825
```

Decicion Tree Regression

In [39]:
```
### Hyper parameter tuning
model=DecisionTreeRegressor()
hyperparametr={"max_depth":[3,5,7,9,11,13,15,19,21]}
search= RandomizedSearchCV(model,hyperparametr,scoring="r2",random_state=0)
search.fit(train_data,y_train)
print("Best hyper parameter: ", search.best_params_)

####################################################################################
model=DecisionTreeRegressor(max_depth=search.best_params_["max_depth"])
model.fit(train_data,y_train)
```

```
C:\Users\ppheg\anaconda3\lib\site-packages\sklearn\model_selection\_search.py:278: U
serWarning: The total space of parameters 9 is smaller than n_iter=10. Running 9 ite
rations. For exhaustive searches, use GridSearchCV.
  warnings.warn(
Best hyper parameter:  {'max_depth': 3}
```

Out[39]: DecisionTreeRegressor(max_depth=3)

In [40]:
```
y_pred_dt = model.predict(test_data)
print(r2_score(y_test, y_pred_dt))
```

```
0.5275030108209662
```