In [1]:

```
import pandas as pd
import numpy as np
```

In [2]:

```
#Preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
```

In [3]:

```
#Algorithms
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBRegressor
```

In [4]:

```
#Tuning
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
```

In [5]:

```
#Metrics
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
```

In [6]:

```
df = pd.read_csv('Properties_links_olx.csv')
```

In [7]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22120 entries, 0 to 22119
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Type            22120 non-null  object
 1   Price           22104 non-null  object
 2   Bedrooms        22118 non-null  object
 3   Bathrooms       22119 non-null  object
 4   Area            22087 non-null  object
 5   Furnished       22120 non-null  object
 6   Level           22120 non-null  object
 7   Payment_Option  22120 non-null  object
 8   Delivery_Term   22120 non-null  object
 9   City            22118 non-null  object
dtypes: object(10)
memory usage: 1.7+ MB
```

In [8]:

```
df['Area'] = df['Area'].replace('Unknown', np.nan)
df['Bedrooms'] = df['Bedrooms'].replace('Unknown', np.nan)
```

```
df['Bathrooms'] = df['Bathrooms'].replace('Unknown', np.nan)
df['Price'] = df['Price'].replace('Unknown', np.nan)
```

In [9]:

```
df.dropna(inplace=True)
#df.drop_duplicates(inplace=True)
```

In [10]:

```
df['Bedrooms'] = df['Bedrooms'].replace('10+',11)
df['Bathrooms'] = df['Bathrooms'].replace('10+',11)
df['Furnished'] = df['Furnished'].replace('Unknown','No')
df['Payment_Option'] = df['Payment_Option'].replace('Unknown','Cash')
df['Delivery_Term'] = df['Delivery_Term'].replace('Unknown ', 'Not Finished')
```

In [11]:

```
df['Bedrooms'] = df['Bedrooms'].astype(int)
df['Bathrooms'] = df['Bathrooms'].astype(int)
df['Area'] = df['Area'].astype(float).astype(int)
df['Price'] = df['Price'].astype(int)
```

In [12]:

```
print(df['Level'].unique())
print(df['Type'].unique())
print(df['Payment_Option'].unique())
```

```
['3' 'Ground' '2' '4' '1' '9' 'Highest' '7' '5' '6' '10' '10+' '8']
['Penthouse' 'Duplex' 'Apartment' 'Studio']
['Cash' 'Cash or Installment' 'Installment']
```

In [13]:

```
df=df.drop(df[(df['Level']=='Unknown')&(df['Type']=='Duplex')].index)
df=df.drop(df[(df['Level']=='Unknown')&(df['Type']=='Apartment')].index)
df=df.drop(df[(df['Level']=='Unknown')&(df['Type']=='Studio')].index)
```

In [14]:

```
df.loc[(df['Level']=='10+'),'Level'] = 11
df.loc[(df['Level']=='Highest'),'Level'] = 12
df.loc[(df['Level']=='Ground'),'Level'] = 0

df.loc[(df['Type']=='Penthouse'),'Level'] = 12
```

In [15]:

```
df['Level'] = df['Level'].astype(int)
```

In [16]:

```
print(df['Level'].unique())
```

```
[12  0  2  4  1  3  9  7  5  6 10 11  8]
```

In [17]:

```
df.reset_index(inplace=True)
df.drop(['index'],axis=1,inplace=True)
```

In [18]:

```
df['Type'].value_counts()
```

Out[18]:

```
Apartment    19225
Duplex        1691
```

```
Penthouse      762
Studio         386
Name: Type, dtype: int64
```

In [19]:

```
city_name = df['City'].value_counts(dropna=False).keys().tolist()
val = df['City'].value_counts(dropna=False).tolist()
value_dict = list(zip(city_name, val))
```

In [20]:

```
Low_frequency_city = []
y = 'Less'
for city_name,val in value_dict:
    if val <= 5:
        Low_frequency_city.append(city_name)
    else :
        pass
def lcdlt(x):
    if x in Low_frequency_city:
        return y
    else :
        return x
df['City'] = df['City'].apply(lcdlt)
df=df.drop(df[(df['City']=='Less')].index)
```

In [21]:

```
lcc = df['City'].unique()
for x in lcc:
    Q1= df[(df['City']==x)]['Price'].quantile(0.25)
    Q3= df[(df['City']==x)]['Price'].quantile(0.75)
    IQR = Q3 - Q1
    upper_bound = Q3 + 1.2 * IQR
    lower_bound = Q1 - 1.2 * IQR
    df=df.drop(df[(df['City']==x)&(df['Price']>=upper_bound)].index)
    df=df.drop(df[(df['City']==x)&(df['Price']<=lower_bound)].index)
```

In [22]:

```
df['City'].unique()
```

Out[22]:

```
array(['Stone Residence', 'Taj City', 'New Cairo - El Tagamoa',
       'Garden Hills', 'Nakheel', '6th of October', 'October Plaza Sodic',
       'Sheikh Zayed', 'The Brooks', 'Badr City', 'Mountain View iCity',
       'Beverly Hills', 'Trio Gardens', 'Sun Capital', 'Nasr City',
       'EL Patio Casa', 'Sodic Westown', 'Asafra', 'Zahraa Al Maadi',
       'Seyouf', 'Madinaty', 'New Capital City', 'Century City', 'Murooj',
       'Sidi Beshr', 'Al Maqsad', 'Sidi Gaber', 'Hadayek al-Ahram',
       'Smoha', 'Roushdy', 'Fifth Square', 'Shorouk City', 'Alma',
       'Gianaclis', 'Hadayek 6th of October', '90 Avenue',
       'Palm Hills New Cairo', 'Mountain View Chillout Park',
       'Village West', 'Mandara', 'Rehab City', 'Obour City',
       'Hyde Park New Cairo', 'Faisal', 'Zed East', 'Uptown Cairo',
       'Ashgar City', 'Monte Napoleon', 'Montazah', 'Maadi', 'Laurent',
       'Galleria Moon Valley', 'New Giza', 'Heliopolis', 'Saba Pasha',
       'Kenz', 'Joulz', 'Isola', 'Sarai', 'Atika', 'Agami', 'Dokki',
       'Raml Station', 'Capital Gardens', 'Mivida', 'Haptown', 'Sheraton',
       'El Khamayel', 'El Fardous', 'Mohandessin', 'Capital Heights 2',
       'L’Avenir', 'Tala', 'VYE Sodic', 'Giza District', 'Degla Gardens',
       'Rock Eden', 'Victoria', 'The Estates',
       'Mountain View - October Park', 'Al Ibrahimiyyah', 'EL Patio 5',
       'Degla Palms', 'The Waterway Compound', 'Green Square', 'The Loft',
       'Haram', 'Stanley', 'Midtown', 'Badya Palm Hills', 'Midtown Sky',
       'Venia', 'Camp Caesar', 'Palm Parks', 'El Ashgar District',
       'Moharam Bik', 'Mountain View Hyde Park', 'Bahray - Anfoshy',
       'Glim', 'Azarita', 'De Joya', 'ZED Towers', 'ABHA',
       'Zizinia El Mostakbal', 'The Icon Residence', 'Kardia', 'Mokattam',
       'Eastown', 'Pyramids Hills', 'Vinci', 'Miami', 'Al Hadrah',
       'Kafr Abdo', 'Noor City', 'Villette', 'El Karma', 'Hexa',
       'West Somid', 'O West', 'Kayan', 'San Stefano', 'AZAD', 'Amorada',
       'Tiba Gardens', 'Swan Lake', 'Mostakbal City', 'Granda', 'Kanarya',
       'Karmell', 'Tag Sultan', 'Palm View', 'The Village',
       'Village Gate', 'Mena Garden City', 'VGK', 'Al Burouj',
       'Italian Square', 'Cairo Gate', 'Bloomfields', 'Sky Condos Sodic',
       'Park View', 'Maamoura', 'EL Patio 7', 'One Kattameya',
       'Hadayek Helwan', 'Dreamland', 'Zayed 2000', 'Zezenia', 'Bolkly',
       'Mountain View 1', 'Lake view Residence', 'EL Patio ORO',
       'New Heliopolis', 'Terrace Smouha', 'Beit Al Watan', 'Bedaya',
       'Aeon', 'Borg al-Arab', 'IL Bosco', 'Gesr Al Suez', 'Maadi View',
       'IL Bosco City', 'Zayed Dunes', 'Jewar', 'Sodic East', 'Eco West',
       'Katameya', 'Shubra', 'Sarayat El Kattameya', 'Jayd', 'Galleria',
       'Etapa', 'Stau', 'Andalus', 'Maryotaya', 'Dar Misr', 'Lake View',
       'Porto October', 'Amreya', 'Maadi V', 'Rock Vera', 'Fleming',
       'Sporting', 'Floria', 'La Mirada', 'District 5 Compound',
       'Gardenia', 'Helmeyat El Zaytoun', 'Baet El Masria', 'Sun Gate',
       'Zayed Regency', 'Regents Park', 'Cleopatra', 'Zamalek', 'Odyssia',
       'Helwan', 'Next Point', 'OIA', 'Hadayek al-Kobba', 'Manshiyya',
       'Cleopatra Palace', 'Agouza', 'Al Manial', 'Sephora Heights',
       'Karma Kay', 'Beta Greens', 'Midtown Condo', 'Ain Shams', 'Joya',
       'The Median', 'Imbaba', 'Mountain View - Giza Plateau', 'Shatby',
       'Belle Vie', 'New Nozha', 'Creek Town', 'Opera City', 'Green Yard',
       'Allegria'], dtype=object)
```

In [23]:

```python
df=df.drop(df[(df['Area']<=100)&(df['Bedrooms']>=4)].index)
df=df.drop(df[(df['Area']<=30)&(df['Type']!='Studio')].index)
```

In [24]:

```python
df.reset_index(inplace=True)
df.drop(['index'],axis=1,inplace=True)
```

In [25]:

```python
for col in df.columns:
    print(col,':',df[col].nunique())
    print(df[col].value_counts().nlargest(7))
    print('\n'+'*'*20+'\n')
```

```
Type : 4
Apartment    18024
Duplex        1407
Penthouse      664
Studio         370
Name: Type, dtype: int64

********************

Price : 3431
3500000    358
2500000    327
3000000    292
```

```
1500000    268
4000000    264
4500000    256
5000000    252
Name: Price, dtype: int64

********************

Bedrooms : 9
3    12909
2     4851
4     1804
1      642
5      218
6       30
7        7
Name: Bedrooms, dtype: int64

********************

Bathrooms : 8
2    8374
3    7212
1    4074
4     700
5      91
6      12
8       1
Name: Bathrooms, dtype: int64

********************

Area : 352
120    705
140    666
150    658
130    606
200    575
160    553
165    525
Name: Area, dtype: int64

********************

Furnished : 2
No     19892
Yes      573
Name: Furnished, dtype: int64

********************

Level : 13
2     5549
0     3364
3     3308
1     2535
4     1478
5     1060
12     860
Name: Level, dtype: int64
```

```
********************

Payment_Option : 3
Cash                     10020
Cash or Installment       7384
Installment               3061
Name: Payment_Option, dtype: int64

********************

Delivery_Term : 4
Finished         10022
Semi Finished     4980
Not Finished      4616
Core & Shell       847
Name: Delivery_Term, dtype: int64

********************

City : 219
Sheikh Zayed               2289
6th of October             1832
New Cairo - El Tagamoa     1668
Madinaty                   1126
Nakheel                    1105
Mountain View iCity         713
Smoha                       606
Name: City, dtype: int64

********************
```

In [26]:

```
df = pd.get_dummies(df, columns = ['Type','Delivery_Term','Furnished','City' ,'Payment_
X = df.drop(columns = ['Price'])
y = df[['Price']]
```

In [27]:

```
from sklearn.model_selection import train_test_split
X_train,X_test, y_train,y_test = train_test_split(X,y,test_size = 0.20,shuffle = True ,

X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[27]:

```
((16372, 236), (4093, 236), (16372, 1), (4093, 1))
```

In [28]:

```
def performance(model,X_train,y_train,y_pred,y_test):
    print('Training Score:',model.score(X_train,y_train))
    print('Testing Score:',r2_score(y_test,y_pred))
    print('Other Metrics In Testing Data: ')
    print('MSE:',mean_squared_error(y_test,y_pred))
    print('MAE:',mean_absolute_error(y_test,y_pred))
```

In [29]:

```
lr = LinearRegression()
lr.fit(X_train,y_train)

lr_pred = lr.predict(X_test)
```

```
performance(lr,X_train,y_train,lr_pred,y_test)
```

```
Training Score: 0.5584429410355207
Testing Score: 0.534537447825368
Other Metrics In Testing Data: 
MSE: 3093023094330.313
MAE: 1223224.5145370143
```

In [30]:

```
ridge = Ridge(alpha = 1)
ridge.fit(X_train,y_train)

#The predicted data
ridge_pred = ridge.predict(X_test)


#The performance
performance(ridge,X_train,y_train,ridge_pred,y_test)
```

```
Training Score: 0.5579257416543518
Testing Score: 0.5344253345710528
Other Metrics In Testing Data: 
MSE: 3093768092790.7856
MAE: 1222851.01501513
```

In [31]:

```
dt = DecisionTreeRegressor()
dt.fit(X_train,y_train)

#The predicted data
dt_pred = dt.predict(X_test)


#The performance
performance(dt,X_train,y_train,dt_pred,y_test)
```

```
Training Score: 0.9705670892893117
Testing Score: 0.4718156183555877
Other Metrics In Testing Data: 
MSE: 3509812943830.1567
MAE: 1085909.8902507357
```

In [32]:

```
rf = RandomForestRegressor()
rf.fit(X_train,y_train.values.ravel())

#The predicted data
rf_pred = rf.predict(X_test)

#The performance
performance(rf,X_train,y_train,rf_pred,y_test)
```

```
Training Score: 0.9267265691835948
Testing Score: 0.6438044346739304
Other Metrics In Testing Data: 
MSE: 2366938230593.107
MAE: 953953.9177945929
```

In [33]:

```
xgb = XGBRegressor()
xgb.fit(X_train,y_train)
```

```
#The predicted data
xgb_pred = xgb.predict(X_test)

#The performance
performance(xgb,X_train,y_train,xgb_pred,y_test)
```

```
Training Score: 0.7452969903216841
Testing Score: 0.6377305723180176
Other Metrics In Testing Data:
MSE: 2407299364804.3374
MAE: 1063096.9965108112
```

In [34]:

```
params = [{'max_depth':list(range(5,20)),'min_samples_split':list(range(2,15)),"min_sam

grid_search = GridSearchCV(estimator=DecisionTreeRegressor(),param_grid=params,cv=10,n_

grid_search.fit(X_train,y_train)

print('Best Estimator:',grid_search.best_estimator_)
print('Best Params:',grid_search.best_params_)

grid_pred = grid_search.predict(X_test)

performance(grid_search,X_train,y_train,grid_pred,y_test)
```

```
Best Estimator: DecisionTreeRegressor(max_depth=19, min_samples_leaf=4, min_samples_spli
t=3)
Best Params: {'max_depth': 19, 'min_samples_leaf': 4, 'min_samples_split': 3}
Training Score: 0.6492374229905005
Testing Score: 0.5405312070474122
Other Metrics In Testing Data:
MSE: 3053194249648.77
MAE: 1198452.0478677144
```

In [35]:

```
params = [{'n_estimators':[100,200,3000,400,500,600],
           'max_depth':list(range(5,20)),'min_samples_split':list(range(2,15))
           ,"min_samples_leaf":[2,3,4,5]}]

rand_search = RandomizedSearchCV(RandomForestRegressor(),params,cv=10,n_jobs=-1)
rand_search.fit(X_train,y_train.values.ravel())

print('Best Estimator:',rand_search.best_estimator_)
print('Best Params:',rand_search.best_params_)

rand_pred = rand_search.predict(X_test)

performance(rand_search,X_train,y_train,rand_pred,y_test)
```

```
Best Estimator: RandomForestRegressor(max_depth=19, min_samples_leaf=2, n_estimators=60
0)
Best Params: {'n_estimators': 600, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_d
epth': 19}
Training Score: 0.7117814675933665
Testing Score: 0.6080488721395514
Other Metrics In Testing Data:
MSE: 2604535820674.025
MAE: 1124950.0387492555
```

In [36]:

```
params = {'max_depth': list(range(5,15)),'n_estimators': [300,400,500,600,700]
          ,'learning_rate': [0.01,0.1,0.2,0.9]}

rand_search = RandomizedSearchCV(XGBRegressor(),params,cv=10,n_jobs=-1)
rand_search.fit(X_train,y_train)

print('Best Estimator:',rand_search.best_estimator_)
print('Best Params:',rand_search.best_params_)

rand_pred = rand_search.predict(X_test)

performance(rand_search,X_train,y_train,rand_pred,y_test)
```

```
Best Estimator: XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=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=10, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=600, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)
Best Params: {'n_estimators': 600, 'max_depth': 10, 'learning_rate': 0.1}
Training Score: 0.8660357632548557
Testing Score: 0.6730094341947591
Other Metrics In Testing Data:
MSE: 2172869475618.5105
MAE: 940733.9736211214
```