Data Collection

In [1]:
```
import numpy as np
import pandas as pd
import plotly.express as px
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
df = pd.read_csv('House Price Prediction in India.csv')
df.head(10)
```

Out[2]:

	POSTED_BY	UNDER_CONSTRUCTION	RERA	BHK_NO.	BHK_OR_RK	SQUARE_FT	READY_TO_MOV
0	Owner	0	0	2	BHK	1300.236407	
1	Dealer	0	0	2	BHK	1275.000000	
2	Owner	0	0	2	BHK	933.159722	
3	Owner	0	1	2	BHK	929.921143	
4	Dealer	1	0	2	BHK	999.009247	
5	Owner	0	0	3	BHK	1250.000000	
6	Dealer	0	0	3	BHK	1495.053957	
7	Owner	0	1	3	BHK	1181.012946	
8	Dealer	0	1	2	BHK	1040.000000	
9	Owner	0	1	2	BHK	879.120879	

In [3]:
```
df[['BHK_NO.', 'BHK_OR_RK']]
```

Out[3]:

	BHK_NO.	BHK_OR_RK
0	2	BHK
1	2	BHK
2	2	BHK
3	2	BHK
4	2	BHK
...	...	...
29446	3	BHK
29447	2	BHK
29448	2	BHK
29449	2	BHK
29450	2	BHK

29451 rows × 2 columns

In [4]:
```
df['BHK_OR_RK'].value_counts()
```

Out[4]:
```
BHK    29427
RK        24
Name: BHK_OR_RK, dtype: int64
```

In [5]:
```
df.describe()
```

Out[5]:

	UNDER_CONSTRUCTION	RERA	BHK_NO.	SQUARE_FT	READY_TO_MOVE	F
count	29451.000000	29451.000000	29451.000000	2.945100e+04	29451.000000	29451.
mean	0.179756	0.317918	2.392279	1.980217e+04	0.820244	0.
std	0.383991	0.465675	0.879091	1.901335e+06	0.383991	0.
min	0.000000	0.000000	1.000000	3.000000e+00	0.000000	0.
25%	0.000000	0.000000	2.000000	9.000211e+02	1.000000	1.
50%	0.000000	0.000000	2.000000	1.175057e+03	1.000000	1.
75%	0.000000	1.000000	3.000000	1.550688e+03	1.000000	1.
max	1.000000	1.000000	20.000000	2.545455e+08	1.000000	1.

In [6]:
```
df['BHK_NO.'].unique()
```

Out[6]:
```
array([ 2,  3,  1,  4,  5,  6, 12,  8, 20, 10,  7,  9, 13, 17, 15, 11],
      dtype=int64)
```

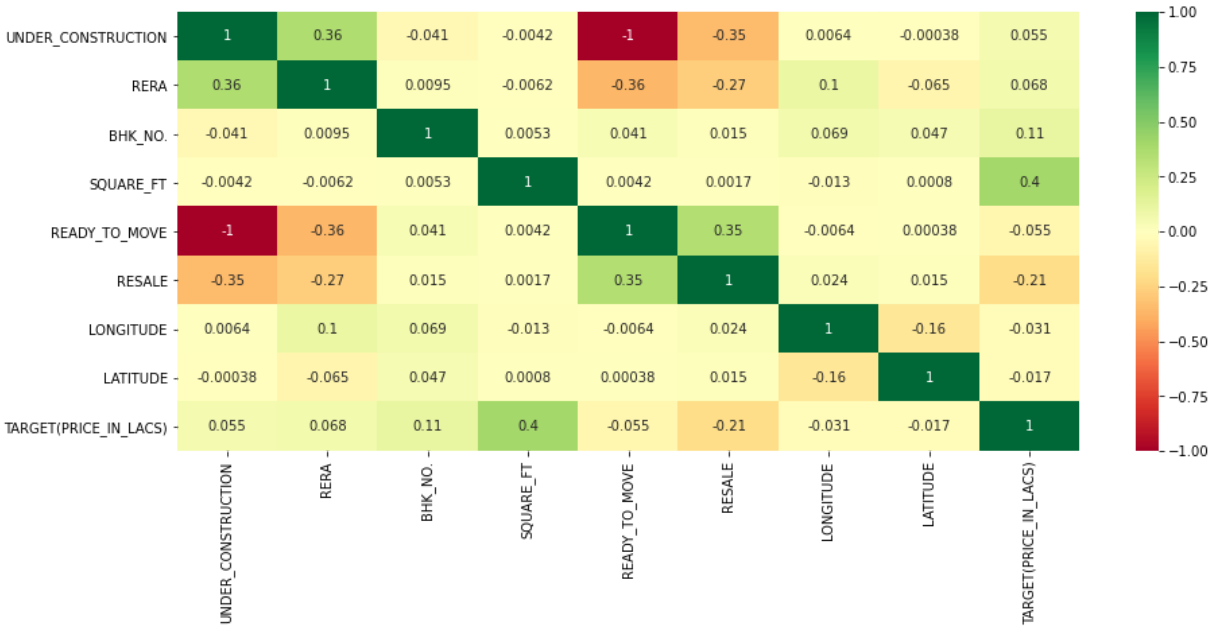
In [7]:
```
df.dtypes
```

Out[7]:
```
POSTED_BY                object
UNDER_CONSTRUCTION        int64
RERA                      int64
BHK_NO.                   int64
BHK_OR_RK                object
SQUARE_FT               float64
READY_TO_MOVE             int64
RESALE                    int64
ADDRESS                  object
LONGITUDE               float64
LATITUDE                float64
TARGET(PRICE_IN_LACS)   float64
dtype: object
```

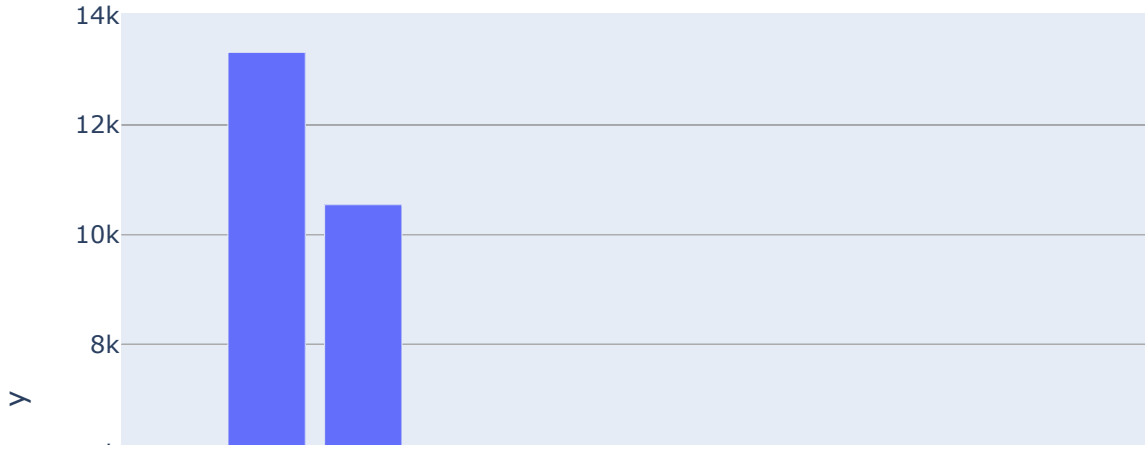
Exploratory Data Analysis

In [8]:
```
plt.figure(figsize = (15, 6))
sns.heatmap(data = df.corr(), annot = True, cmap = 'RdYlGn')
```

Out[8]:
```
<AxesSubplot:>
```

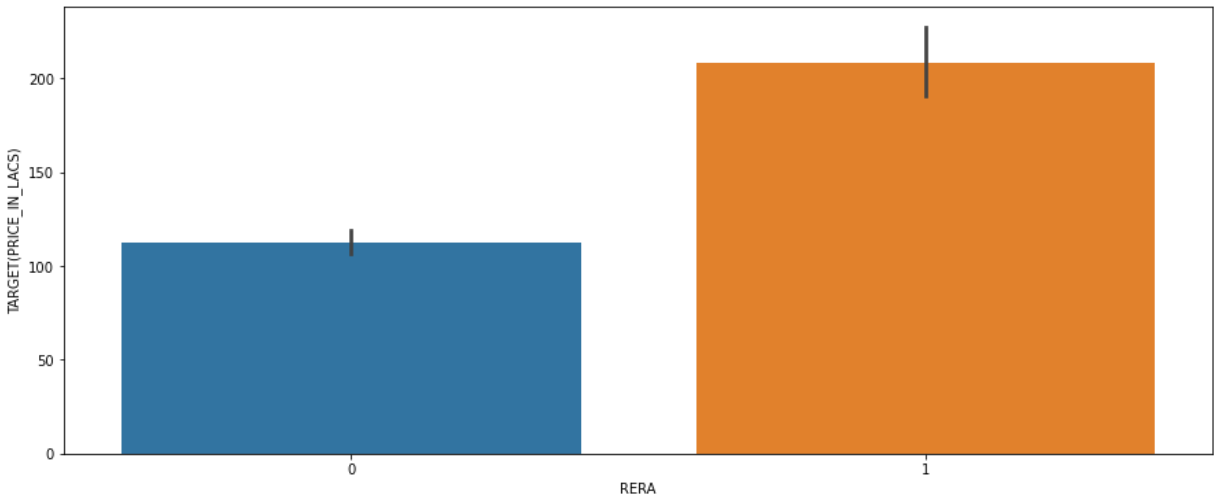


In [9]:
```
fig = px.bar(x=df["BHK_NO."].unique(), y=df["BHK_NO."].value_counts())
fig.show()
```



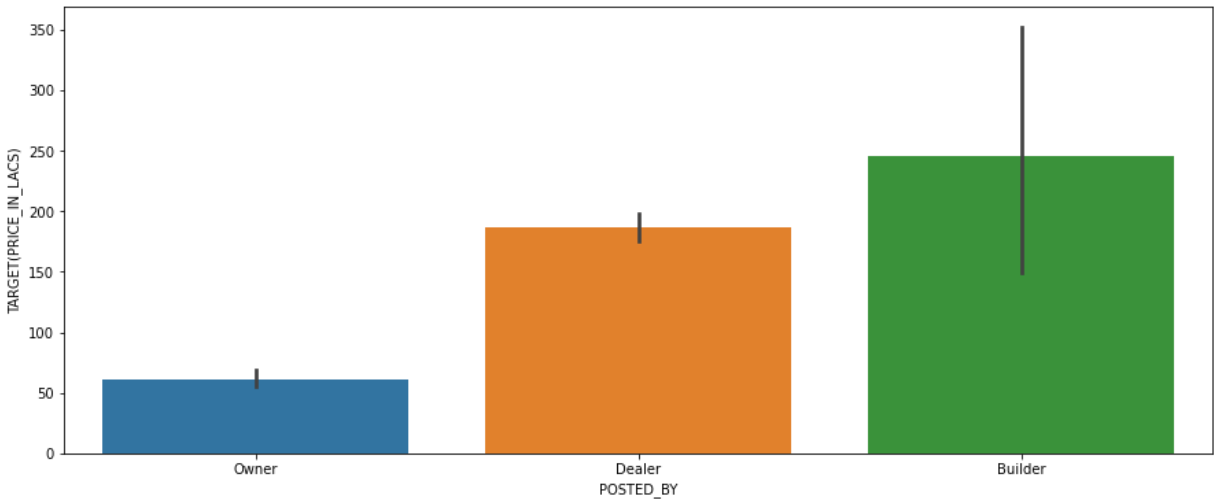
In [10]:
```
plt.figure(figsize = (15, 6))
sns.barplot(data = df, x = 'RERA', y = 'TARGET(PRICE_IN_LACS)')
```

Out[10]: <AxesSubplot:xlabel='RERA', ylabel='TARGET(PRICE_IN_LACS)'>



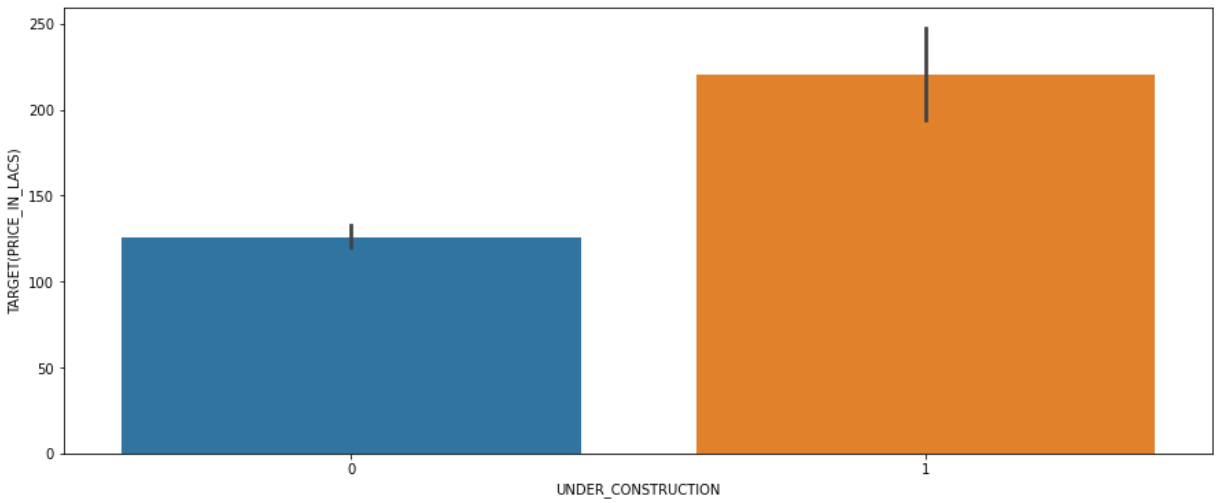
In [11]:
```
plt.figure(figsize = (15, 6))
sns.barplot(data = df, x = 'POSTED_BY', y = 'TARGET(PRICE_IN_LACS)')
```

Out[11]: <AxesSubplot:xlabel='POSTED_BY', ylabel='TARGET(PRICE_IN_LACS)'>



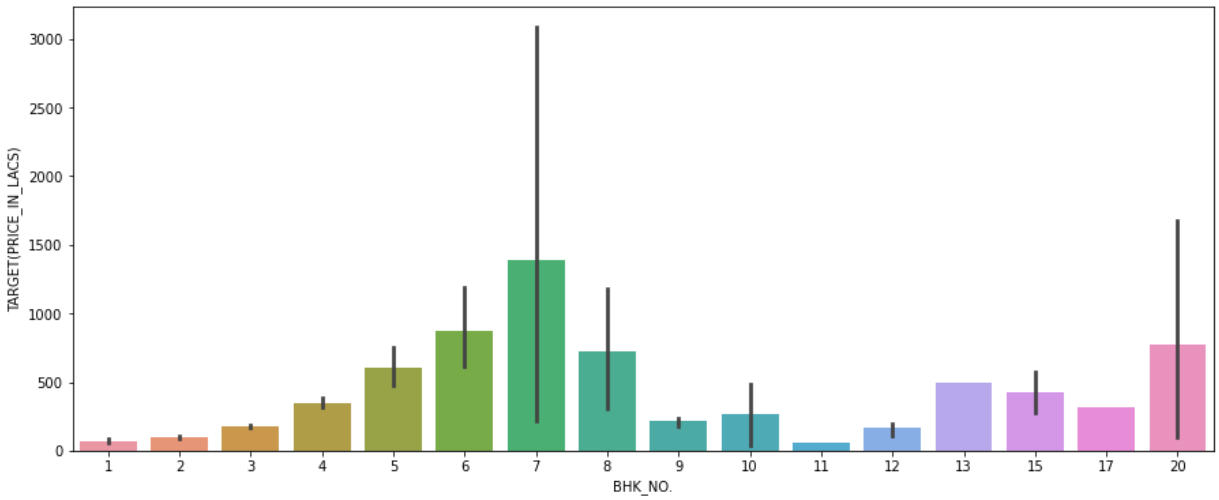
In [12]:
```
plt.figure(figsize = (15, 6))
sns.barplot(data = df, x = 'UNDER_CONSTRUCTION', y = 'TARGET(PRICE_IN_LACS)')
```

Out[12]: <AxesSubplot:xlabel='UNDER_CONSTRUCTION', ylabel='TARGET(PRICE_IN_LACS)'>



In [13]:
```
plt.figure(figsize = (15, 6))
sns.barplot(data = df, x = 'BHK_NO.', y = 'TARGET(PRICE_IN_LACS)')
```

Out[13]: <AxesSubplot:xlabel='BHK_NO.', ylabel='TARGET(PRICE_IN_LACS)'>



OneHotEncoding

In [14]:
```
df = df.drop(['BHK_OR_RK', 'ADDRESS', 'LATITUDE', 'LONGITUDE'], axis = 1)
df.head()
```

Out[14]:

	POSTED_BY	UNDER_CONSTRUCTION	RERA	BHK_NO.	SQUARE_FT	READY_TO_MOVE	RESALE
0	Owner	0	0	2	1300.236407	1	1
1	Dealer	0	0	2	1275.000000	1	1
2	Owner	0	0	2	933.159722	1	1
3	Owner	0	1	2	929.921143	1	1
4	Dealer	1	0	2	999.009247	0	1

In [15]:
```
df = pd.get_dummies(df)
```

In [16]:
```
df.columns
```

Out[16]:
```
Index(['UNDER_CONSTRUCTION', 'RERA', 'BHK_NO.', 'SQUARE_FT', 'READY_TO_MOVE',
       'RESALE', 'TARGET(PRICE_IN_LACS)', 'POSTED_BY_Builder',
       'POSTED_BY_Dealer', 'POSTED_BY_Owner'],
      dtype='object')
```

In [17]:
```
df.head()
```

Out[17]:

	UNDER_CONSTRUCTION	RERA	BHK_NO.	SQUARE_FT	READY_TO_MOVE	RESALE	TARGET(PRICE
0	0	0	2	1300.236407	1	1	
1	0	0	2	1275.000000	1	1	
2	0	0	2	933.159722	1	1	
3	0	1	2	929.921143	1	1	
4	1	0	2	999.009247	0	1	

In [18]:
```
df = df.drop(['POSTED_BY_Builder'], axis = 1)
df.head()
```

Out[18]:

	UNDER_CONSTRUCTION	RERA	BHK_NO.	SQUARE_FT	READY_TO_MOVE	RESALE	TARGET(PRICE
0	0	0	2	1300.236407	1	1	
1	0	0	2	1275.000000	1	1	
2	0	0	2	933.159722	1	1	
3	0	1	2	929.921143	1	1	
4	1	0	2	999.009247	0	1	

Feature Scaling

In [19]:
```
X = df.drop(columns = ['TARGET(PRICE_IN_LACS)'])
y = df['TARGET(PRICE_IN_LACS)']
```

In [20]:
```
X
```

Out[20]:

	UNDER_CONSTRUCTION	RERA	BHK_NO.	SQUARE_FT	READY_TO_MOVE	RESALE	POSTED_I
0	0	0	2	1300.236407	1	1	
1	0	0	2	1275.000000	1	1	
2	0	0	2	933.159722	1	1	
3	0	1	2	929.921143	1	1	
4	1	0	2	999.009247	0	1	
...	...	...	...	...	...	...	
29446	0	0	3	2500.000000	1	1	
29447	0	0	2	769.230769	1	1	
29448	0	0	2	1022.641509	1	1	
29449	0	0	2	927.079009	1	1	
29450	0	1	2	896.774194	1	1	

29451 rows × 8 columns

In [21]:
```
y
```

Out[21]:
```
0        55.0
1        51.0
2        43.0
3        62.5
4        60.5
         ...
29446    45.0
29447    16.0
29448    27.1
29449    67.0
29450    27.8
Name: TARGET(PRICE_IN_LACS), Length: 29451, dtype: float64
```

In [23]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

```
X = sc.fit_transform(X)
```

In [24]:
```
X
```

Out[24]:
```
array([[-0.46813431, -0.68271456, -0.44623962, ...,  0.27523994,
        -1.28022595,  1.33968012],
       [-0.46813431, -0.68271456, -0.44623962, ...,  0.27523994,
         0.78111211, -0.74644684],
       [-0.46813431, -0.68271456, -0.44623962, ...,  0.27523994,
        -1.28022595,  1.33968012],
       ...,
       [-0.46813431, -0.68271456, -0.44623962, ...,  0.27523994,
         0.78111211, -0.74644684],
       [-0.46813431, -0.68271456, -0.44623962, ...,  0.27523994,
        -1.28022595,  1.33968012],
       [-0.46813431,  1.464741  , -0.44623962, ...,  0.27523994,
         0.78111211, -0.74644684]])
```

In [25]:
```
X.shape
```

Out[25]:
```
(29451, 8)
```

In [26]:
```
y.shape
```

Out[26]:
```
(29451,)
```

Splitting the dataset into Training and Testing Set**

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 = 0.2, random_st
```

In [28]:
```
X_train
```

Out[28]:
```
array([[-0.46813431,  1.464741  , -0.44623962, ...,  0.27523994,
         0.78111211, -0.74644684],
       [ 2.13613911,  1.464741  , -0.44623962, ...,  0.27523994,
         0.78111211, -0.74644684],
       [ 2.13613911,  1.464741  , -0.44623962, ..., -3.6331937 ,
         0.78111211, -0.74644684],
       ...,
       [-0.46813431, -0.68271456,  1.82887558, ...,  0.27523994,
        -1.28022595,  1.33968012],
       [-0.46813431,  1.464741  , -0.44623962, ...,  0.27523994,
        -1.28022595,  1.33968012],
       [ 2.13613911,  1.464741  ,  0.69131798, ...,  0.27523994,
         0.78111211, -0.74644684]])
```

Model Building

Multiple Linear Regression

In [29]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[29]:
```
LinearRegression()
```

In [30]:
```
y_pred = regressor.predict(X_test)
```

In [31]:
```
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

R2_Score = r2_score(y_test, y_pred)
Mean_Absolute_Error = mean_absolute_error(y_test, y_pred)
Mean_Square_Error = mean_squared_error(y_test, y_pred)
Root_Mean_Square_Error = np.sqrt(mean_squared_error(y_test, y_pred))

results = pd.DataFrame([['Multiple Linear Regression', R2_Score, Mean_Absolute_Error
                         columns = ['Model', 'R2 Score', 'Mean Absolute Error', 'Mean S
```

In [32]:
```
results
```

Out[32]:

	Model	R2 Score	Mean Absolute Error	Mean Square Error	Root Mean Square Error
0	Multiple Linear Regression	0.372462	133.049526	288060.13234	536.712337

Random Forest

In [33]:
```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 100, random_state = 1)
regressor.fit(X_train, y_train)
```

Out[33]:
```
RandomForestRegressor(random_state=1)
```

In [34]:
```
y_pred = regressor.predict(X_test)
```

In [35]:
```
R2_Score = r2_score(y_test, y_pred)
Mean_Absolute_Error = mean_absolute_error(y_test, y_pred)
Mean_Square_Error = mean_squared_error(y_test, y_pred)
Root_Mean_Square_Error = np.sqrt(mean_squared_error(y_test, y_pred))

model_results = pd.DataFrame([['Random Forest', R2_Score, Mean_Absolute_Error, Mean_
                         columns = ['Model', 'R2 Score', 'Mean Absolute Error', 'Mean S
results = results.append(model_results, ignore_index = True)
```

In [36]:
```
results
```

Out[36]:

	Model	R2 Score	Mean Absolute Error	Mean Square Error	Root Mean Square Error
0	Multiple Linear Regression	0.372462	133.049526	288060.132340	536.712337
1	Random Forest	0.918399	57.487444	37457.249154	193.538754

XGBoost

In [37]:
```
from xgboost import XGBRegressor
regressor = XGBRegressor(random_state = 2)
regressor.fit(X_train, y_train)
```

Out[37]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=2,
```

```
                reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
                tree_method='exact', validate_parameters=1, verbosity=None)
```

In [38]:
```
y_pred = regressor.predict(X_test)
```

In [39]:
```
R2_Score = r2_score(y_test, y_pred)
Mean_Absolute_Error = mean_absolute_error(y_test, y_pred)
Mean_Square_Error = mean_squared_error(y_test, y_pred)
Root_Mean_Square_Error = np.sqrt(mean_squared_error(y_test, y_pred))

model_results = pd.DataFrame([['XGB Regressor', R2_Score, Mean_Absolute_Error, Mean_
                      columns = ['Model', 'R2 Score', 'Mean Absolute Error', 'Mean S
results = results.append(model_results, ignore_index = True)
```

In [40]:
```
results
```

Out[40]:

	Model	R2 Score	Mean Absolute Error	Mean Square Error	Root Mean Square Error
0	Multiple Linear Regression	0.372462	133.049526	288060.132340	536.712337
1	Random Forest	0.918399	57.487444	37457.249154	193.538754
2	XGB Regressor	0.936873	56.221285	28977.345801	170.227336

Hyper-Parameter Tuning Using RandomizedSearchCV

In [41]:
```
parameters = {"learning_rate": [0.05, 0.10, 0.15, 0.20, 0.25, 0.30], "max_depth": [3
              "min_child_weight": [1, 3, 5, 7], "gamma": [0.0, 0.1, 0.2, 0.3, 0.4],
```

In [42]:
```
from sklearn.model_selection import RandomizedSearchCV
random_search = RandomizedSearchCV(estimator = regressor, param_distributions = para
                                   cv = 10, verbose = 3)
```

In [43]:
```
import time

t0 = time.time()
random_search.fit(X_train, y_train)
t1 = time.time()
print("Took %0.2f Seconds" %(t1-t0))
```

```
Fitting 10 folds for each of 5 candidates, totalling 50 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  24 tasks      | elapsed:   32.6s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:   39.8s finished
Took 41.98 Seconds
```

In [44]:
```
random_search.best_estimator_
```

Out[44]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.5, gamma=0.0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.05, max_delta_step=0, max_depth=12,
             min_child_weight=3, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=2,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

In [45]:
```
random_search.best_params_
```

Out[45]:
```
{'min_child_weight': 3,
 'max_depth': 12,
 'learning_rate': 0.05,
 'gamma': 0.0,
 'colsample_bytree': 0.5}
```

In [46]:
```
regressor = XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.3, gamma=0.0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.15, max_delta_step=0, max_depth=6,
             min_child_weight=3, monotone_constraints='()',
             n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=2,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

In [47]:
```
regressor.fit(X_train, y_train)
```

Out[47]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.3, gamma=0.0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.15, max_delta_step=0, max_depth=6,
             min_child_weight=3, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=2,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

In [48]:
```
y_pred = regressor.predict(X_test)
```

In [49]:
```
R2_Score = r2_score(y_test, y_pred)
Mean_Absolute_Error = mean_absolute_error(y_test, y_pred)
Mean_Square_Error = mean_squared_error(y_test, y_pred)
Root_Mean_Square_Error = np.sqrt(mean_squared_error(y_test, y_pred))

model_results = pd.DataFrame([['XGB Regressor(Hyper-Parameter Tuned)', R2_Score, Mea
                      columns = ['Model', 'R2 Score', 'Mean Absolute Error', 'Mean S
results = results.append(model_results, ignore_index = True)
```

In [50]:
```
results
```

Out[50]:

	Model	R2 Score	Mean Absolute Error	Mean Square Error	Root Mean Square Error
0	Multiple Linear Regression	0.372462	133.049526	288060.132340	536.712337
1	Random Forest	0.918399	57.487444	37457.249154	193.538754
2	XGB Regressor	0.936873	56.221285	28977.345801	170.227336
3	XGB Regressor(Hyper-Parameter Tuned)	0.858660	63.394923	64879.348576	254.714249

Hence, we will go with the XGBoost algorithm.