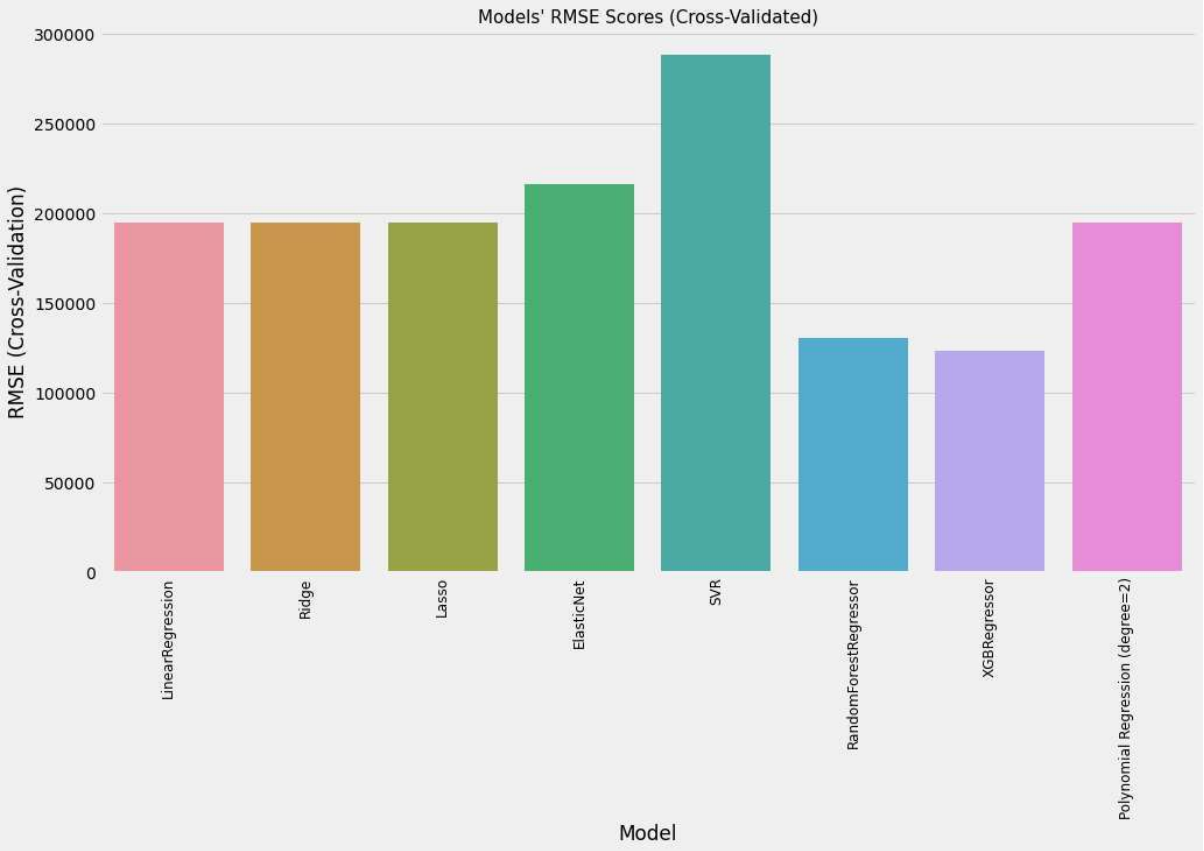


Here we can see that XGB Regressor gives us the best model among all.

---------- Moving on to Phase 2.

Phase.2 Modeling after treating the outliers and multicollinearity

Data Preparation before Modeling

1. Outliers Treatment

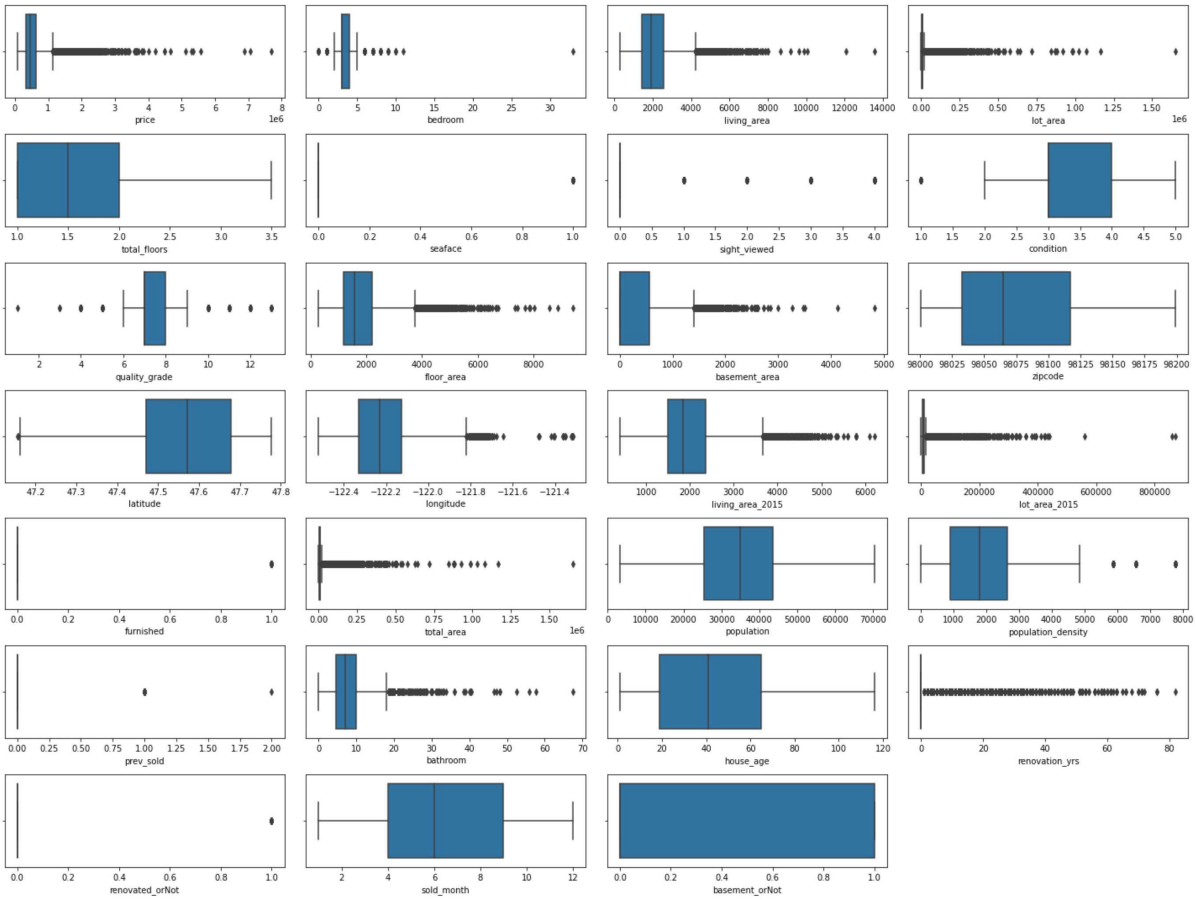
In [54]:
```
df_phase2.columns
```

Out[54]:
```
Index(['price', 'bedroom', 'living_area', 'lot_area', 'total_floors',
       'seaface', 'sight_viewed', 'condition', 'quality_grade', 'floor_area',
       'basement_area', 'zipcode', 'latitude', 'longitude', 'living_area_2015',
       'lot_area_2015', 'furnished', 'total_area', 'city', 'population',
       'population_density', 'prev_sold', 'bathroom', 'house_age',
       'renovation_yrs', 'renovated_orNot', 'sold_month', 'basement_orNot'],
      dtype='object')
```

In [55]:
```
## let's boxplot all the numerical columns and see if there any outliers

data_plot=df_phase2[['price', 'bedroom', 'living_area', 'lot_area', 'total_floors',
       'seaface', 'sight_viewed', 'condition', 'quality_grade', 'floor_area',
       'basement_area', 'zipcode', 'latitude', 'longitude', 'living_area_2015',
       'lot_area_2015', 'furnished', 'total_area', 'population',
       'population_density', 'prev_sold', 'bathroom', 'house_age',
       'renovation_yrs', 'renovated_orNot', 'sold_month', 'basement_orNot']]

fig=plt.figure(figsize=(20,15));
for i in range(0,len(data_plot.columns)):
    ax=fig.add_subplot(7,4,i+1)
    sns.boxplot(data_plot[data_plot.columns[i]])
    plt.tight_layout()
```



Note: We can see that we can see the outliers that might affect our results. The variables in which outliers might affect our model are the following:

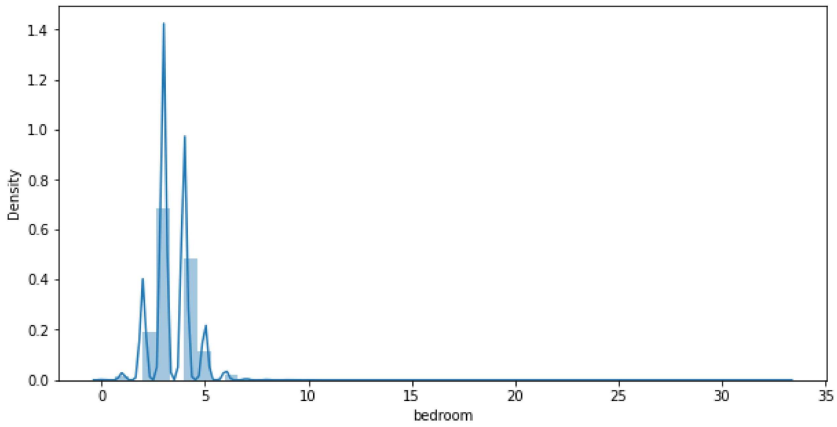
1. price
2. bedroom
3. bathroom
4. living area
5. lot area
6. floor area
7. basement_area
8. total area
9. living area 2015
10. lot area 2015
11. total area

Here, we can refer the correlation between these and treat outliers based on that. so that we wont have a significant data loss.

Trimming the extreme ouliers without affecting any significant loss of data

In [56]:
```
## bedroom distribution
print("Skewness", df_phase2.bedroom.skew())
plt.figure(figsize=(10,5))
sns.distplot(df_phase2.bedroom)
df_phase2.bedroom.describe()
```

Out[56]:
```
Skewness 1.989212330597263
count    21387.000000
mean         3.370880
std          0.930488
min          0.000000
25%          3.000000
50%          3.000000
75%          4.000000
max         33.000000
Name: bedroom, dtype: float64
```



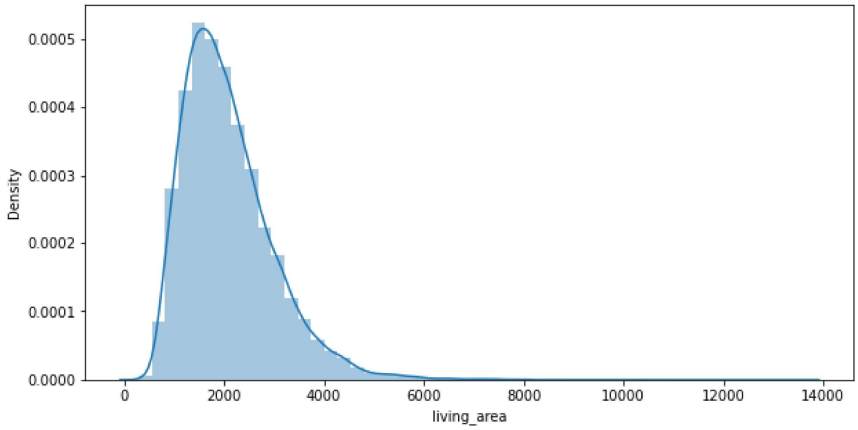
In [57]:
```
## trimming extreme outliers where number of bedroom is more than 10
df_phase2=df_phase2[(df_phase2['bedroom']<=10)]
print('loss of data is', (1-(df_phase2.index.size/dataset.index.size))*100, 'percent')
```

```
loss of data is 0.009351475195207293 percent
```

In [58]:
```
## living area distribution
print("Skewness", df_phase2.living_area.skew())
plt.figure(figsize=(10,5))
sns.distplot(df_phase2.living_area)
df_phase2.living_area.describe()
```

```
Skewness 1.473317409009823
```

Out[58]:
```
count    21385.000000
mean      2080.451812
std        918.959032
min        290.000000
25%       1430.000000
50%       1910.000000
75%       2550.000000
max      13540.000000
Name: living_area, dtype: float64
```



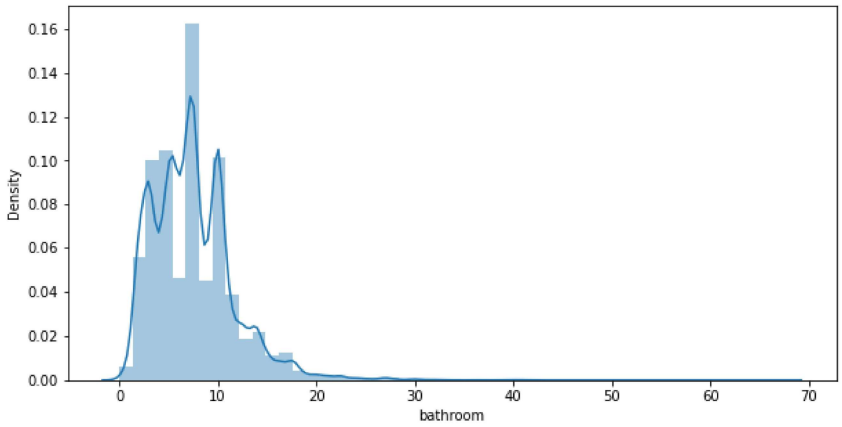
In [59]:
```
## trimming extreme outliers
df_phase2=df_phase2[(df_phase2['living_area']<=8000)]
print('loss of data is', (1-(df_phase2.index.size/dataset.index.size))*100, 'percent')
```

```
loss of data is 0.051433113573662315 percent
```

In [60]:
```
## bathroom distribution
print("Skewness", df_phase2.bathroom.skew())
plt.figure(figsize=(10,5))
sns.distplot(df_phase2.bathroom)
df_phase2.bathroom.describe()
```

```
Skewness 1.5465543349594382
```

Out[60]:
```
count    21376.000000
mean         7.482890
std          4.163702
min          0.000000
25%          4.500000
50%          7.000000
75%         10.000000
max         67.500000
Name: bathroom, dtype: float64
```

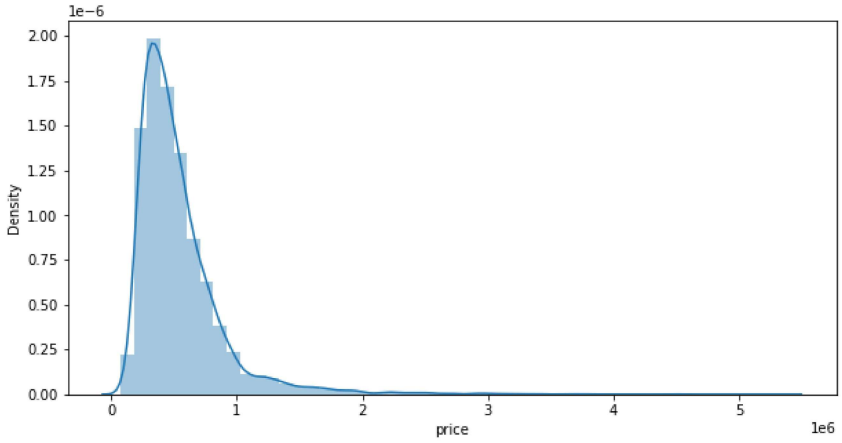


In [61]:
```
## trimming extreme outliers
df_phase2=df_phase2[(df_phase2['bathroom']<=30)]
print('loss of data is', (1-(df_phase2.index.size/dataset.index.size))*100, 'percent')
```

```
loss of data is 0.15429934072099805 percent
```

In [62]:
```
## price distribution
print("Skewness", df_phase2.price.skew())
plt.figure(figsize=(10,5))
sns.distplot(df_phase2.price)
df_phase2.price.describe()
```

Out[62]:
```
Skewness 3.156592280688759
count    2.135400e+04
mean     5.375164e+05
std      3.513037e+05
min      7.500000e+04
25%      3.210000e+05
50%      4.500000e+05
75%      6.430015e+05
max      5.350000e+06
Name: price, dtype: float64
```



In [63]:
```
## trimming extreme outliers
df_phase2=df_phase2[(df_phase2['price']<=4000000)]
print('loss of data is', (1-(df_phase2.index.size/dataset.index.size))*100, 'percent')
```

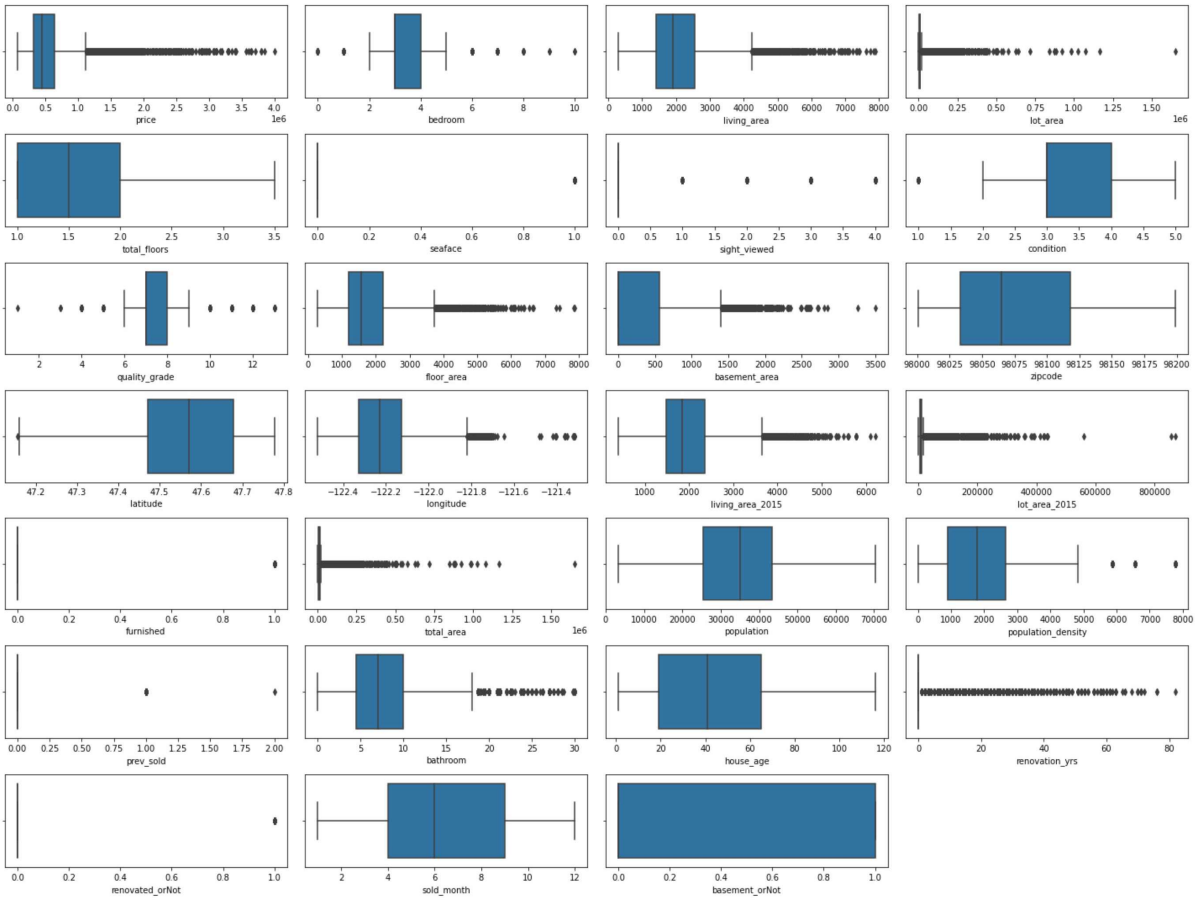
```
loss of data is 0.17300229111142373 percent
```

In [64]:
```
## After trimming lets check the boxplot

## let's boxplot all the numerical columns and see if there any outliers

data_plot=df_phase2[['price', 'bedroom', 'living_area', 'lot_area', 'total_floors',
       'seaface', 'sight_viewed', 'condition', 'quality_grade', 'floor_area',
       'basement_area', 'zipcode', 'latitude', 'longitude', 'living_area_2015',
       'lot_area_2015', 'furnished', 'total_area', 'population',
       'population_density', 'prev_sold', 'bathroom', 'house_age',
       'renovation_yrs', 'renovated_orNot', 'sold_month', 'basement_orNot']]

fig=plt.figure(figsize=(20,15));
for i in range(0,len(data_plot.columns)):
    ax=fig.add_subplot(7,4,i+1)
    sns.boxplot(data_plot[data_plot.columns[i]])
    plt.tight_layout()
```



Only the main variables from which we wish to treat outliers

Highly correlation varibales

1. total_area is totally correlated with lot_area with correlation of 1.
2. floor_area is highly correlated with living_area.
3. quality_grade is correlated with furnished.
4. renovated_orNot is correlated with renovation_yrs.

In [65]:
```
## drop total area which completely correlated variable
df_phase2.drop('total_area', axis=1, inplace=True)
```

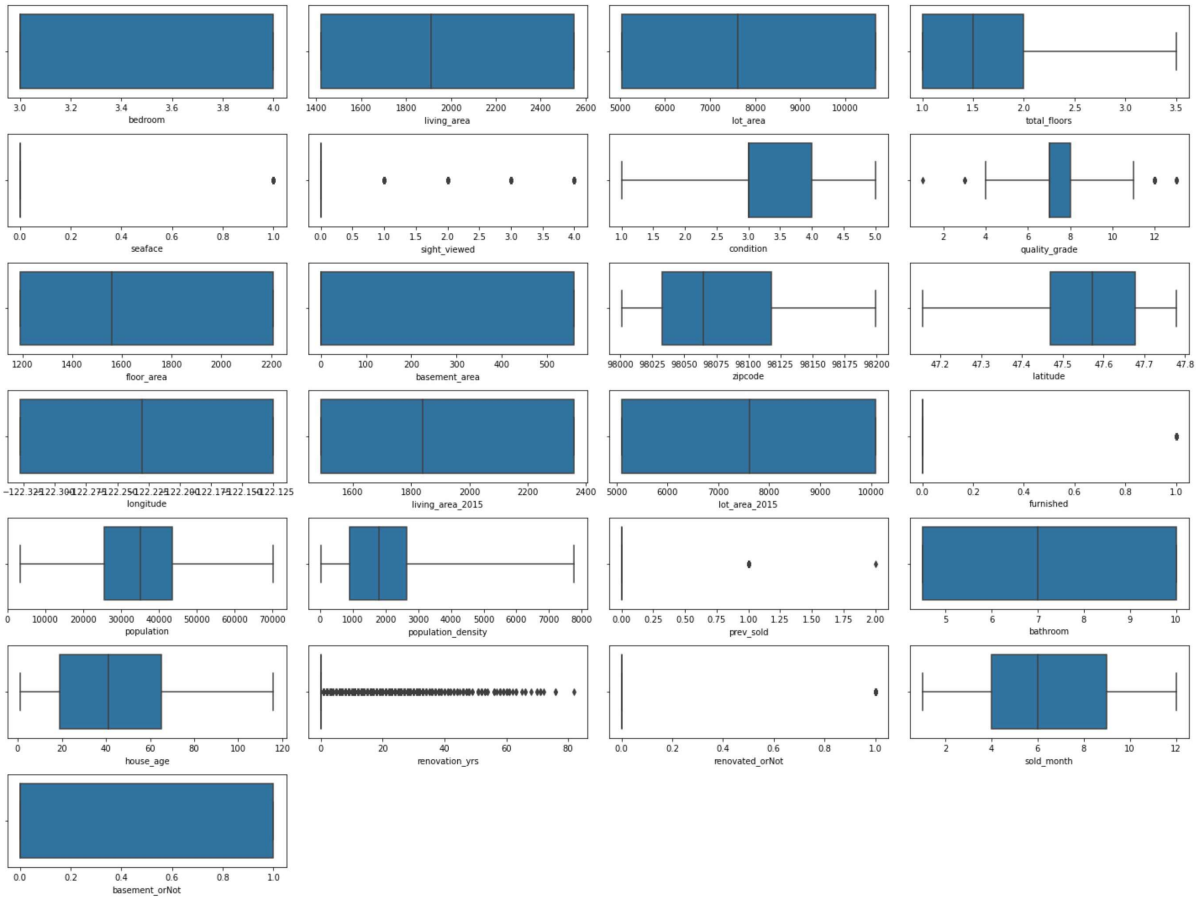
In [66]:
```
df_treated = df_phase2
```

In [67]:
```
out_data=df_treated[['bedroom','bathroom', 'living_area', 'lot_area', 'floor_area', 'basement_area', 'longitude','living_

for i in out_data:
    lr = df_treated[i].quantile(0.25)
    ur = df_treated[i].quantile(0.75)
    df_treated[i] = np.where(df_treated[i] <lr, lr,df_treated[i])
    df_treated[i] = np.where(df_treated[i] >ur, ur,df_treated[i])
```

In [68]:
```
out_plot=df_treated[['bedroom', 'living_area', 'lot_area', 'total_floors',
       'seaface', 'sight_viewed', 'condition', 'quality_grade', 'floor_area',
       'basement_area', 'zipcode', 'latitude', 'longitude', 'living_area_2015',
       'lot_area_2015', 'furnished', 'population',
       'population_density', 'prev_sold', 'bathroom', 'house_age',
       'renovation_yrs', 'renovated_orNot', 'sold_month', 'basement_orNot']]
fig=plt.figure(figsize=(20,15))
for i in range(0,len(out_plot.columns)):
    ax=fig.add_subplot(7,4,i+1)
    sns.boxplot(out_plot[out_plot.columns[i]], whis=3)
    plt.tight_layout()
print('Shape after Outliers Treatment',df_treated.shape)
```

```
Shape after Outliers Treatment (21350, 27)
```



In [69]:
```
## signs of data loss after outlier treatment
print('loss of data is', (1-(df_treated.index.size/dataset.index.size))*100, 'percent.')
```

```
loss of data is 0.17300229111142373 percent.
```

In [70]:
```
df_treated.head()
```

Out[70]:

	price	bedroom	living_area	lot_area	total_floors	seaface	sight_viewed	condition	quality_grade	floor_area	basement_area	zipcode	l
0	1400000	4.0	2550.0	5040.0	1.5	0.0	0.0	5	8.0	1910.0	560.0	98105	
1	615000	3.0	2360.0	7291.0	1.0	0.0	0.0	4	8.0	1360.0	560.0	98136	
2	400000	4.0	2550.0	10660.0	1.5	0.0	0.0	3	9.0	2210.0	560.0	98092	
3	284000	3.0	1800.0	10660.0	1.0	0.0	0.0	3	7.0	1800.0	0.0	98014	
4	335000	3.0	1420.0	5040.0	1.0	0.0	0.0	3	8.0	1350.0	0.0	98052	

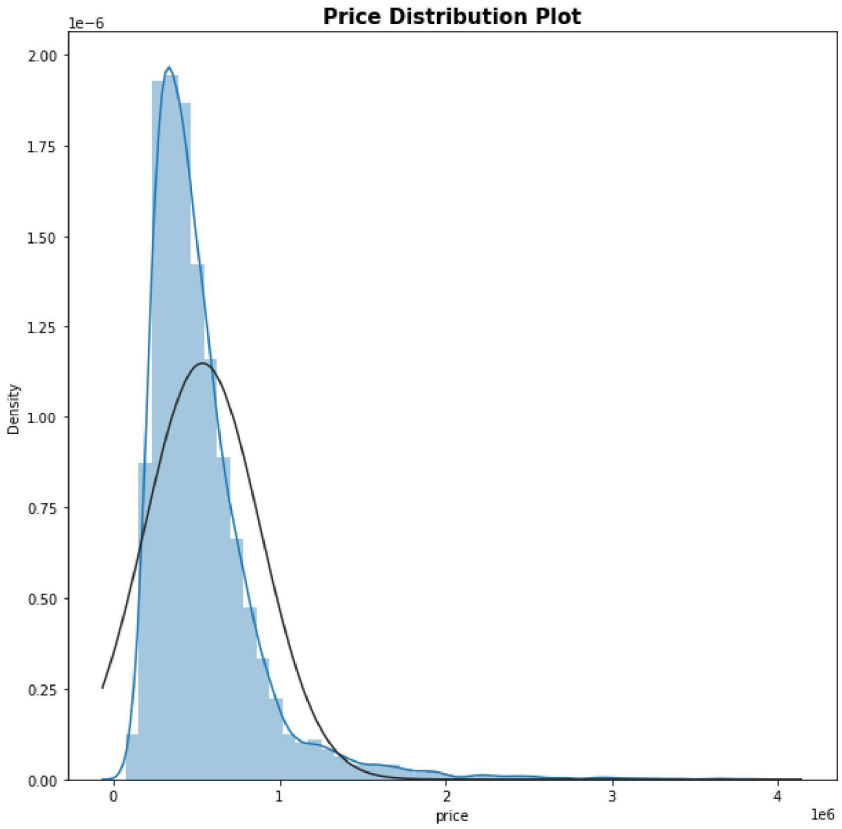
Standadization of data

Normalising the price distribution (if required)

In [71]:
```
## checking price distribution

plt.figure(figsize=(10,10))
sns.distplot(df_treated['price'], fit=norm)
plt.title("Price Distribution Plot",size=15, weight='bold')
```

Out[71]:
```
Text(0.5, 1.0, 'Price Distribution Plot')
```



In [72]:
```
## price distribution
print("Skewness", df_treated.price.skew())
df_treated.price.describe()
```

Out[72]:
```
Skewness 2.9690376616455456
count    2.135000e+04
mean     5.367483e+05
std      3.467752e+05
min      7.500000e+04
25%      3.209250e+05
50%      4.500000e+05
75%      6.430000e+05
max      4.000000e+06
Name: price, dtype: float64
```

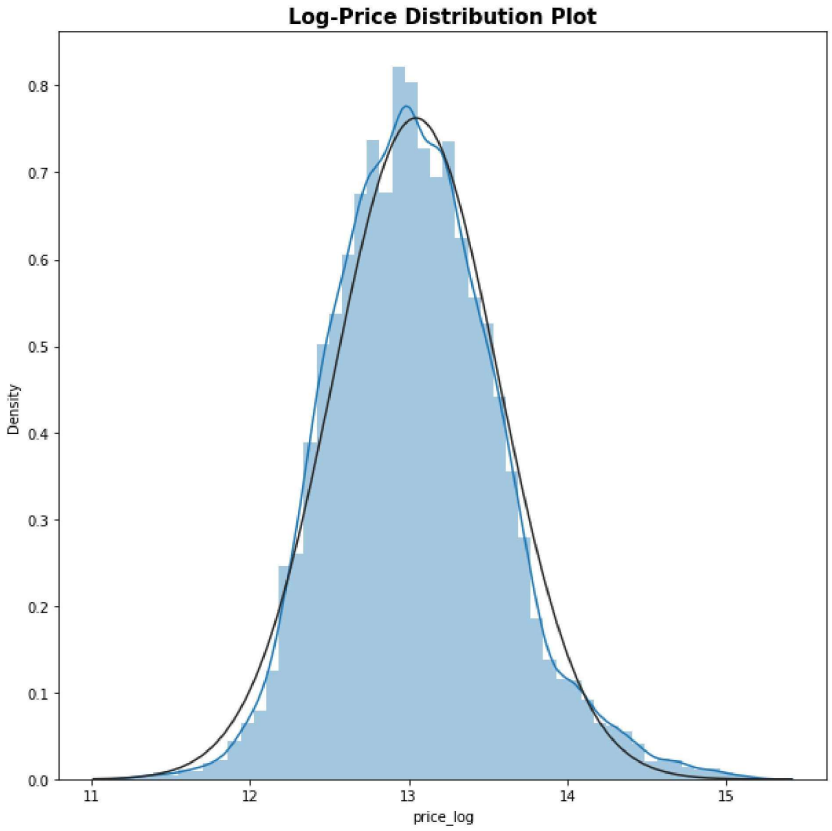
Note: The above distribution graph shows that there is a right-skewed distribution on price. This means there is a positive skewness. Log transformation will be used to make this feature less skewed. This will help to make easier interpretation and better statistical analysis

Since division by zero is a problem, log+1 transformation would be better.

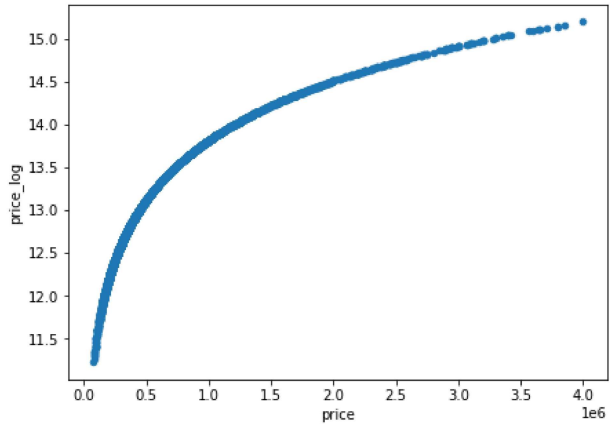
In [73]:
```
df_treated['price_log'] = np.log(df_treated.price+1)
```

In [74]:
```
plt.figure(figsize=(10,10))
sns.distplot(df_treated['price_log'], fit=norm)
plt.title("Log-Price Distribution Plot",size=15, weight='bold')
```

Out[74]:
```
Text(0.5, 1.0, 'Log-Price Distribution Plot')
```



In [75]:
```
df_treated.plot.scatter(x='price', y='price_log', figsize=(7,5));
```



Dealing with Multi-Collinearity

In [76]:
```
## features which are highly correlated with price of house (Top 15 features except price)
df_treated.corr()["price"].nlargest(17)
```

Out[76]:
```
price                1.000000
price_log            0.910557
quality_grade        0.676346
furnished            0.582748
living_area          0.537559
floor_area           0.476833
living_area_2015     0.476406
bathroom             0.417337
sight_viewed         0.397810
latitude             0.320758
bedroom              0.301891
total_floors         0.261202
seaface              0.249952
basement_area        0.214285
basement_orNot       0.180707
population_density   0.133864
lot_area             0.131412
Name: price, dtype: float64
```

These are the best features that highly correlated with the price

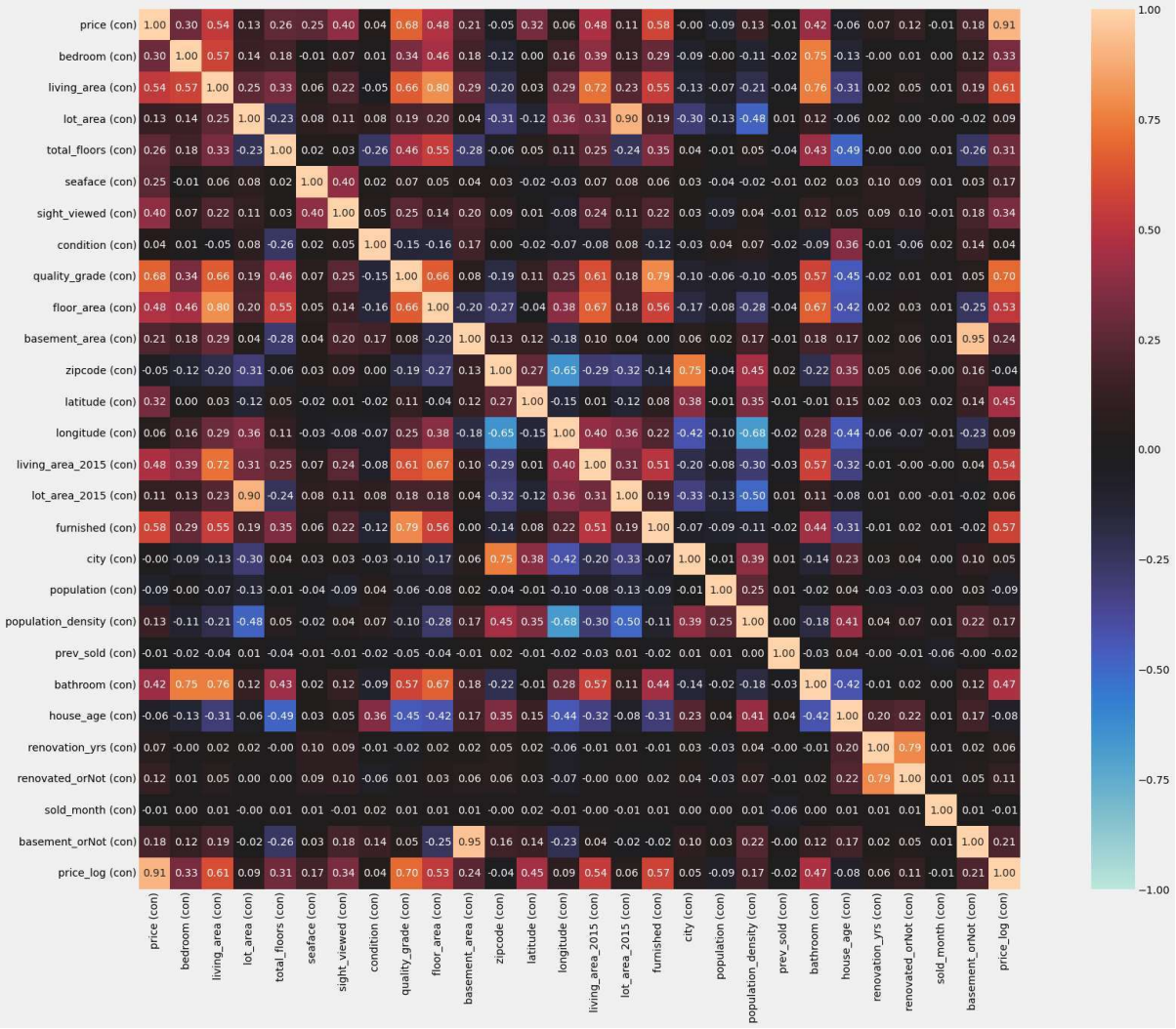
1. quality_grade
2. living_area
3. furnished
4. living_area_2015
5. floor_area

6. bathroom
7. latitude
8. sight_viewed
9. bedroom
10. total_floors
11. basement_area
12. basement_orNot
13. population_density
14. seaface
15. renovated_orNot

Checking collinearity using correlation map

In [150…

```
from dython import nominal
nominal.associations(df_treated,figsize=(30,20),mark_columns=True);
```



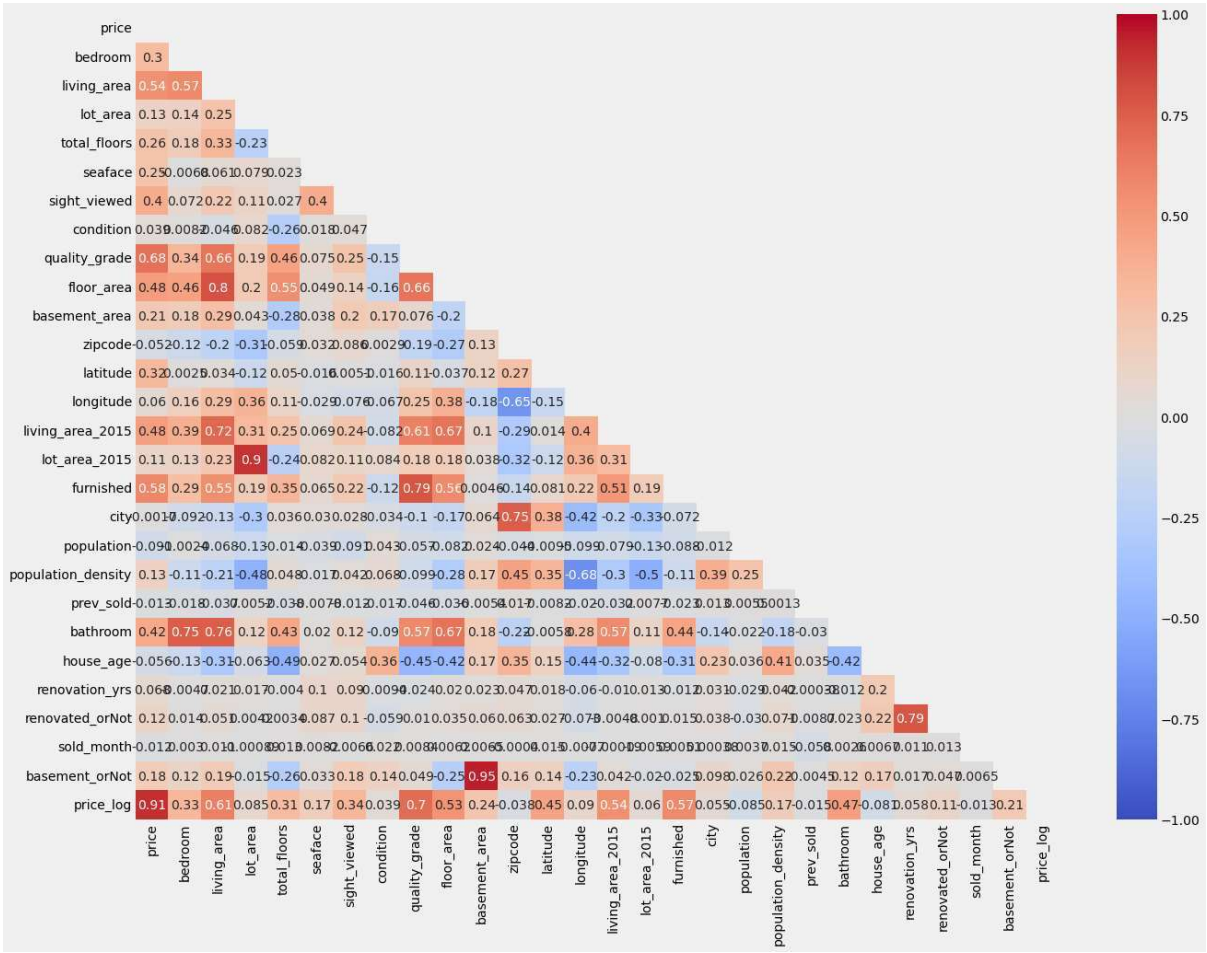
In [151…

```
#zipcode & city is changed from object datatype to int
#dataset2 = dataset2.astype({'zipcode':'int64', 'city':'int64', 'sold_month':'int64'})
```

In [152…

```
#correlation heatmap to understand the relation between the variables

plt.figure(figsize = (20,15))
sns.heatmap(df_treated.corr(), annot=True,mask=np.triu(df_treated.corr()),cmap ='coolwarm', vmin = -1, vmax= 1);
```



In [154…

```
## checking multi-collinearity using eigen values

corr=df_treated.corr(method='pearson')

#Eigen vector of a correlation matrix.
multicollinearity, V=np.linalg.eig(corr)
multicollinearity
```

Out[154…

```
array([6.63712763, 3.76549977, 2.62642779, 1.8978921 , 1.55939512,
       1.22896837, 1.20240931, 1.08166087, 1.0588104 , 0.94504591,
       0.91679422, 0.79184507, 0.65538556, 0.58507015, 0.51243886,
       0.48507807, 0.32514604, 0.29973537, 0.27313307, 0.22898667,
       0.20608395, 0.03671444, 0.05911115, 0.15603854, 0.14833898,
       0.13932046, 0.09482168, 0.08272046])
```

Multi-collinearity check using Variance Inflation Factor

There might be redundant variables in the dataset, to eliminate which we should use VIF.

Feature Selection

In [155…

```
df_treated.columns
```

Out[155…

```
Index(['price', 'bedroom', 'living_area', 'lot_area', 'total_floors',
       'seaface', 'sight_viewed', 'condition', 'quality_grade', 'floor_area',
       'basement_area', 'zipcode', 'latitude', 'longitude', 'living_area_2015',
       'lot_area_2015', 'furnished', 'city', 'population',
       'population_density', 'prev_sold', 'bathroom', 'house_age',
       'renovation_yrs', 'renovated_orNot', 'sold_month', 'basement_orNot',
       'price_log'],
      dtype='object')
```

In [156…

```
Selected_features = df_treated[['price','bedroom', 'living_area', 'lot_area', 'total_floors',
       'seaface', 'sight_viewed', 'condition', 'quality_grade', 'floor_area',
       'basement_area', 'zipcode', 'latitude', 'longitude', 'living_area_2015',
       'lot_area_2015', 'furnished', 'city', 'population',
       'population_density', 'prev_sold', 'bathroom', 'house_age',
       'renovation_yrs', 'renovated_orNot', 'sold_month', 'basement_orNot',
       ]]

from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
```

```
    return(vif)
```

In [157…

```
X = Selected_features.drop('price', axis = 1)
calc_vif(X).sort_values(by = 'VIF', ascending = False)
```

Out[157…

	variables	VIF
10	zipcode	6.239439e+06
12	longitude	5.899616e+06
11	latitude	1.518470e+05
7	quality_grade	1.906630e+02
1	living_area	1.633303e+02
0	bedroom	1.304239e+02
8	floor_area	1.299838e+02
14	lot_area_2015	8.782906e+01
13	living_area_2015	7.623409e+01
2	lot_area	7.190790e+01
20	bathroom	5.523419e+01
6	condition	3.578392e+01
9	basement_area	2.234073e+01
3	total_floors	2.127847e+01
25	basement_orNot	1.829431e+01
16	city	9.826289e+00
17	population	9.514697e+00
21	house_age	8.843930e+00
18	population_density	8.009636e+00
24	sold_month	5.475423e+00
15	furnished	3.449315e+00
23	renovated_orNot	2.903698e+00
22	renovation_yrs	2.727860e+00
5	sight_viewed	1.523747e+00
4	seaface	1.215738e+00
19	prev_sold	1.016801e+00

In [158…

```
X = X.drop('zipcode', axis = 1)
calc_vif(X).sort_values(by = 'VIF', ascending = False)
```

Out[158…

	variables	VIF
10	latitude	142358.159200
11	longitude	140273.531716
7	quality_grade	190.586210
1	living_area	163.329854
0	bedroom	130.417760
8	floor_area	129.979953
13	lot_area_2015	87.798686
12	living_area_2015	75.712303
2	lot_area	71.907889
19	bathroom	55.230506
6	condition	35.770179
9	basement_area	22.336281
3	total_floors	21.278219
24	basement_orNot	18.293290
16	population	9.513314
20	house_age	8.833099
15	city	8.004492
17	population_density	6.403477
23	sold_month	5.475227
14	furnished	3.446712

	variables	VIF
22	renovated_orNot	2.902078
21	renovation_yrs	2.726291
5	sight_viewed	1.523745
4	seaface	1.214738
18	prev_sold	1.016740

In [159…
```
X = X.drop('latitude', axis = 1)
calc_vif(X).sort_values(by = 'VIF', ascending = False)
```

Out[159…

	variables	VIF
10	longitude	263.542792
7	quality_grade	188.081762
1	living_area	163.304170
0	bedroom	130.413883
8	floor_area	129.796966
12	lot_area_2015	87.605674
11	living_area_2015	74.929624
2	lot_area	71.894438
18	bathroom	55.229575
6	condition	35.755693
9	basement_area	22.336279
3	total_floors	21.275152
23	basement_orNot	18.293288
15	population	9.482341
19	house_age	8.820164
14	city	7.486461
16	population_density	6.316335
22	sold_month	5.474642
13	furnished	3.446065
21	renovated_orNot	2.901378
20	renovation_yrs	2.726097
5	sight_viewed	1.507186
4	seaface	1.214462
17	prev_sold	1.016663

In [160…
```
X = X.drop('longitude', axis = 1)
calc_vif(X).sort_values(by = 'VIF', ascending = False)
```

Out[160…

	variables	VIF
1	living_area	163.145783
7	quality_grade	131.919514
8	floor_area	129.698516
0	bedroom	105.816267
11	lot_area_2015	85.716691
10	living_area_2015	72.335472
2	lot_area	71.860426
17	bathroom	50.776794
6	condition	32.496007
9	basement_area	22.175735
3	total_floors	20.344198
22	basement_orNot	18.185594
14	population	9.234389
18	house_age	8.646644
13	city	7.102919
15	population_density	6.316227
21	sold_month	5.401626

	variables	VIF
20	renovated_orNot	2.900058
19	renovation_yrs	2.725092
12	furnished	2.573686
5	sight_viewed	1.488391
4	seaface	1.214376
16	prev_sold	1.014654

In [161…

```
X = X.drop('living_area', axis = 1)
calc_vif(X).sort_values(by = 'VIF', ascending = False)
```

Out[161…

	variables	VIF
6	quality_grade	131.904864
0	bedroom	105.494277
10	lot_area_2015	85.675208
1	lot_area	71.772295
9	living_area_2015	68.342994
7	floor_area	66.428694
16	bathroom	48.904928
5	condition	32.495951
2	total_floors	20.230692
8	basement_area	17.749161
21	basement_orNot	17.521524
13	population	9.232913
17	house_age	8.640215
12	city	7.067305
14	population_density	6.314084
20	sold_month	5.401354
19	renovated_orNot	2.899964
18	renovation_yrs	2.725080
11	furnished	2.549076
4	sight_viewed	1.484138
3	seaface	1.214083
15	prev_sold	1.014571

In [162…

```
X = X.drop('quality_grade', axis = 1)
calc_vif(X).sort_values(by = 'VIF', ascending = False)
```

Out[162…

	variables	VIF
0	bedroom	98.111276
9	lot_area_2015	84.198911
1	lot_area	71.641443
6	floor_area	63.856879
8	living_area_2015	60.779187
15	bathroom	47.812743
5	condition	30.362854
2	total_floors	18.837925
7	basement_area	17.699049
20	basement_orNot	17.258414
12	population	9.139060
16	house_age	8.186472
11	city	6.917056
13	population_density	5.924749
19	sold_month	5.367969
18	renovated_orNot	2.884319
17	renovation_yrs	2.719012
10	furnished	1.930927

	variables	VIF
4	sight_viewed	1.483744
3	seaface	1.214029
14	prev_sold	1.014571

In [163…
```
X = X.drop('bedroom', axis = 1)
calc_vif(X).sort_values(by = 'VIF', ascending = False)
```

Out[163…

	variables	VIF
8	lot_area_2015	82.763563
0	lot_area	71.636187
5	floor_area	63.703172
7	living_area_2015	59.488151
4	condition	29.167535
14	bathroom	28.938585
1	total_floors	18.756520
6	basement_area	17.652213
19	basement_orNot	17.228787
11	population	8.970372
15	house_age	7.567898
10	city	6.730541
12	population_density	5.912252
18	sold_month	5.330900
17	renovated_orNot	2.880045
16	renovation_yrs	2.718758
9	furnished	1.893168
3	sight_viewed	1.472690
2	seaface	1.212839
13	prev_sold	1.014409

In [164…
```
X = X.drop('lot_area_2015', axis = 1)
calc_vif(X).sort_values(by = 'VIF', ascending = False)
```

Out[164…

	variables	VIF
5	floor_area	63.661065
7	living_area_2015	57.990651
13	bathroom	28.937303
4	condition	28.528076
1	total_floors	18.755578
6	basement_area	17.645153
0	lot_area	17.552453
18	basement_orNot	17.224899
10	population	8.957299
14	house_age	7.567702
9	city	6.730181
11	population_density	5.871958
17	sold_month	5.326017
16	renovated_orNot	2.878869
15	renovation_yrs	2.718471
8	furnished	1.893054
3	sight_viewed	1.471733
2	seaface	1.212198
12	prev_sold	1.014098

In [165…
```
X = X.drop('floor_area', axis = 1)
calc_vif(X).sort_values(by = 'VIF', ascending = False)
```

Out[165…

	variables	VIF

	variables	VIF
6	living_area_2015	45.133101
4	condition	28.496664
12	bathroom	23.495881
5	basement_area	17.562968
0	lot_area	17.294431
17	basement_orNot	17.125306
1	total_floors	16.507980
9	population	8.956270
13	house_age	7.509827
8	city	6.729798
10	population_density	5.831991
16	sold_month	5.324673
15	renovated_orNot	2.876498
14	renovation_yrs	2.718427
7	furnished	1.795600
3	sight_viewed	1.470563
2	seaface	1.212056
11	prev_sold	1.014037

In [166…

```
X = X.drop('living_area_2015', axis = 1)
calc_vif(X).sort_values(by = 'VIF', ascending = False)
```

Out[166…

	variables	VIF
4	condition	27.288083
11	bathroom	19.721924
5	basement_area	17.562367
16	basement_orNot	17.121749
1	total_floors	15.434232
0	lot_area	14.415916
8	population	8.763802
12	house_age	7.456787
7	city	6.670426
9	population_density	5.781721
15	sold_month	5.286449
14	renovated_orNot	2.875051
13	renovation_yrs	2.718427
6	furnished	1.730769
3	sight_viewed	1.448461
2	seaface	1.209923
10	prev_sold	1.014032

In [167…

```
X = X.drop('condition', axis = 1)
calc_vif(X).sort_values(by = 'VIF', ascending = False)
```

Out[167…

	variables	VIF
10	bathroom	18.933959
4	basement_area	17.537667
15	basement_orNot	17.121696
1	total_floors	14.688567
0	lot_area	12.219010
7	population	8.398456
6	city	6.632675
11	house_age	5.923411
8	population_density	5.781016
14	sold_month	5.198313
13	renovated_orNot	2.805810

	variables	VIF
12	renovation_yrs	2.714210
5	furnished	1.700998
3	sight_viewed	1.448374
2	seaface	1.209777
9	prev_sold	1.013689

In [168…
```
X = X.drop('bathroom', axis = 1)
calc_vif(X).sort_values(by = 'VIF', ascending = False)
```

Out[168…

	variables	VIF
14	basement_orNot	16.888556
4	basement_area	16.408270
0	lot_area	10.713033
1	total_floors	9.692652
7	population	8.174482
6	city	6.623021
10	house_age	5.904442
8	population_density	5.738273
13	sold_month	5.175228
12	renovated_orNot	2.801403
11	renovation_yrs	2.712120
5	furnished	1.638095
3	sight_viewed	1.448312
2	seaface	1.207821
9	prev_sold	1.013665

In [169…
```
X = X.drop('basement_orNot', axis = 1)
calc_vif(X).sort_values(by = 'VIF', ascending = False)
```

Out[169…

	variables	VIF
0	lot_area	10.705214
1	total_floors	9.677979
7	population	8.169797
6	city	6.597565
10	house_age	5.895856
8	population_density	5.647707
13	sold_month	5.173712
12	renovated_orNot	2.798444
11	renovation_yrs	2.711291
4	basement_area	1.854371
5	furnished	1.628678
3	sight_viewed	1.447687
2	seaface	1.207717
9	prev_sold	1.013653

In [170…
```
X = X.drop('lot_area', axis = 1)
calc_vif(X).sort_values(by = 'VIF', ascending = False)
```

Out[170…

	variables	VIF
0	total_floors	8.246759
6	population	7.079700
5	city	6.555098
12	sold_month	4.892456
9	house_age	4.687252
7	population_density	4.261547
11	renovated_orNot	2.791269
10	renovation_yrs	2.711270

	variables	VIF
3	basement_area	1.769980
4	furnished	1.569336
2	sight_viewed	1.443225
1	seaface	1.207565
8	prev_sold	1.012945

In [171…
```
X = X.drop('total_floors', axis = 1)
calc_vif(X).sort_values(by = 'VIF', ascending = False)
```

Out[171…

	variables	VIF
5	population	5.657529
4	city	5.589510
8	house_age	4.380148
11	sold_month	4.321014
6	population_density	4.171695
10	renovated_orNot	2.787050
9	renovation_yrs	2.710689
2	basement_area	1.707710
1	sight_viewed	1.439807
3	furnished	1.380814
0	seaface	1.207565
7	prev_sold	1.012898

In [172…
```
X = X.drop('population', axis = 1)
calc_vif(X).sort_values(by = 'VIF', ascending = False)
```

Out[172…

	variables	VIF
4	city	5.252973
7	house_age	4.286074
5	population_density	3.930655
10	sold_month	3.566459
9	renovated_orNot	2.783374
8	renovation_yrs	2.710683
2	basement_area	1.698079
1	sight_viewed	1.431882
3	furnished	1.358683
0	seaface	1.207564
6	prev_sold	1.011681

In [173…
```
X = X.drop('city', axis = 1)
calc_vif(X).sort_values(by = 'VIF', ascending = False)
```

Out[173…

	variables	VIF
6	house_age	3.946816
4	population_density	3.319328
9	sold_month	2.967401
8	renovated_orNot	2.780598
7	renovation_yrs	2.710621
2	basement_area	1.694152
1	sight_viewed	1.431715
3	furnished	1.327381
0	seaface	1.206317
5	prev_sold	1.010255

Using VIF we came with best factors which are not affected by multi-collinearity.

1. house_age
2. population_density

3. sold_month
4. renovated_orNot
5. renovation_yrs
6. basement_area
7. sight_viewed
8. furnished
9. seaface
10. prev_sold

Model Building

Model Building based on Feature selection using VIF

In [77]:
```
df_VIF = df_treated[['price','house_age', 'population_density', 'sold_month', 'renovated_orNot','renovation_yrs','basemer
df_VIF.head(3)
```

Out[77]:

	price	house_age	population_density	sold_month	renovated_orNot	renovation_yrs	basement_area	sight_viewed	furnished	seaface	pre
0	1400000	107	4717.7	5	0	0	560.0	0.0	0.0	0.0	
1	615000	68	2854.7	5	0	0	560.0	0.0	0.0	0.0	
2	400000	37	415.2	5	0	0	560.0	0.0	1.0	0.0	

X, y Split

Splitting the data into X and y chunks

In [78]:
```
X = df_VIF.drop("price", axis=1)
y = df_VIF["price"]
```

Standardizing the Data

In [79]:
```
scaler = StandardScaler()
X[X.columns] = scaler.fit_transform(X[X.columns])
```

In [80]:
```
## check the standarization
X.head(3)
```

Out[80]:

	house_age	population_density	sold_month	renovated_orNot	renovation_yrs	basement_area	sight_viewed	furnished	seaface	prev_sold
0	2.112390	2.008161	-0.504484	-0.20881	-0.164114	1.480299	-0.305345	-0.493991	-0.085516	-0.090915
1	0.784283	0.676721	-0.504484	-0.20881	-0.164114	1.480299	-0.305345	-0.493991	-0.085516	-0.090915
2	-0.271392	-1.066730	-0.504484	-0.20881	-0.164114	1.480299	-0.305345	2.024327	-0.085516	-0.090915

Train-Test Split

In [81]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Machine Learning Models

In [82]:
```
models_VIF = pd.DataFrame(columns=["Model","MAE","MSE","RMSE","R2 Score","RMSE (Cross-Validation)"])
```

Linear Regression

In [180…
```
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
predictions = lin_reg.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(lin_reg)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "LinearRegression","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validati
models_VIF = models_VIF.append(new_row, ignore_index=True)
```

```
MAE: 164077.42709379256
MSE: 63646042086.74129
RMSE: 252281.6721181729
R2 Score: 0.4771801896249397
```

```
------------------------------
RMSE Cross-Validation: 248840.47270037193
```

Ridge Regression

In [181…

```
ridge = Ridge()
ridge.fit(X_train, y_train)
predictions = ridge.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(ridge)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "Ridge","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)": rmse_
models_VIF = models_VIF.append(new_row, ignore_index=True)
```

```
MAE: 164076.7838119446
MSE: 63646037898.72578
RMSE: 252281.66381789575
R2 Score: 0.4771802240273535
------------------------------
RMSE Cross-Validation: 248840.402084884
```

Lasso Regression

In [182…

```
lasso = Lasso()
lasso.fit(X_train, y_train)
predictions = lasso.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(lasso)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "Lasso","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)": rmse_
models_VIF = models_VIF.append(new_row, ignore_index=True)
```

```
MAE: 164077.46036327674
MSE: 63646023336.35692
RMSE: 252281.63495656382
R2 Score: 0.47718034364979467
------------------------------
RMSE Cross-Validation: 248840.4386764097
```

Elastic Net

In [183…

```
elastic_net = ElasticNet()
elastic_net.fit(X_train, y_train)
predictions = elastic_net.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(elastic_net)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "ElasticNet","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)":
models_VIF = models_VIF.append(new_row, ignore_index=True)
```

```
MAE: 168166.95450491627
MSE: 69050081867.3627
RMSE: 262773.8226448036
R2 Score: 0.43278875599088595
------------------------------
RMSE Cross-Validation: 258668.26293427852
```

Support Vector Machines

In [184…

```
svr = SVR(C=100000)
svr.fit(X_train, y_train)
predictions = svr.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(svr)
print("RMSE Cross-Validation:", rmse_cross_val)
```

```
new_row = {"Model": "SVR","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)": rmse_cr
models_VIF = models_VIF.append(new_row, ignore_index=True)
```

```
MAE: 153611.6003180887
MSE: 63505191438.896385
RMSE: 252002.3639549764
R2 Score: 0.47833720593865037
------------------------------
RMSE Cross-Validation: 248661.92042070744
```

Random Forest Regressor

In [185…

```
random_forest = RandomForestRegressor(n_estimators=100)
random_forest.fit(X_train, y_train)
predictions = random_forest.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(random_forest)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "RandomForestRegressor","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Val
models_VIF = models_VIF.append(new_row, ignore_index=True)
```

```
MAE: 113937.40993814438
MSE: 36378119081.57222
RMSE: 190730.48807564095
R2 Score: 0.7011722850871347
------------------------------
RMSE Cross-Validation: 188586.30468290622
```

XGBoost Regressor

In [186…

```
xgb = XGBRegressor(n_estimators=1000, learning_rate=0.01)
xgb.fit(X_train, y_train)
predictions = xgb.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(xgb)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "XGBRegressor","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)"
models_VIF = models_VIF.append(new_row, ignore_index=True)
```

```
MAE: 109588.81040690867
MSE: 33885451945.153866
RMSE: 184080.01506180366
R2 Score: 0.7216482756886284
------------------------------
RMSE Cross-Validation: 177662.84213388775
```

Polynomial Regression (Degree=2)

In [187…

```
poly_reg = PolynomialFeatures(degree=2)
X_train_2d = poly_reg.fit_transform(X_train)
X_test_2d = poly_reg.transform(X_test)

lin_reg = LinearRegression()
lin_reg.fit(X_train_2d, y_train)
predictions = lin_reg.predict(X_test_2d)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(lin_reg)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "Polynomial Regression (degree=2)","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE
models_VIF = models_VIF.append(new_row, ignore_index=True)
```

```
MAE: 157616.53278688525
MSE: 60219234511.57447
RMSE: 245396.07680558888
R2 Score: 0.5053296680198227
------------------------------
RMSE Cross-Validation: 248840.47270037193
```

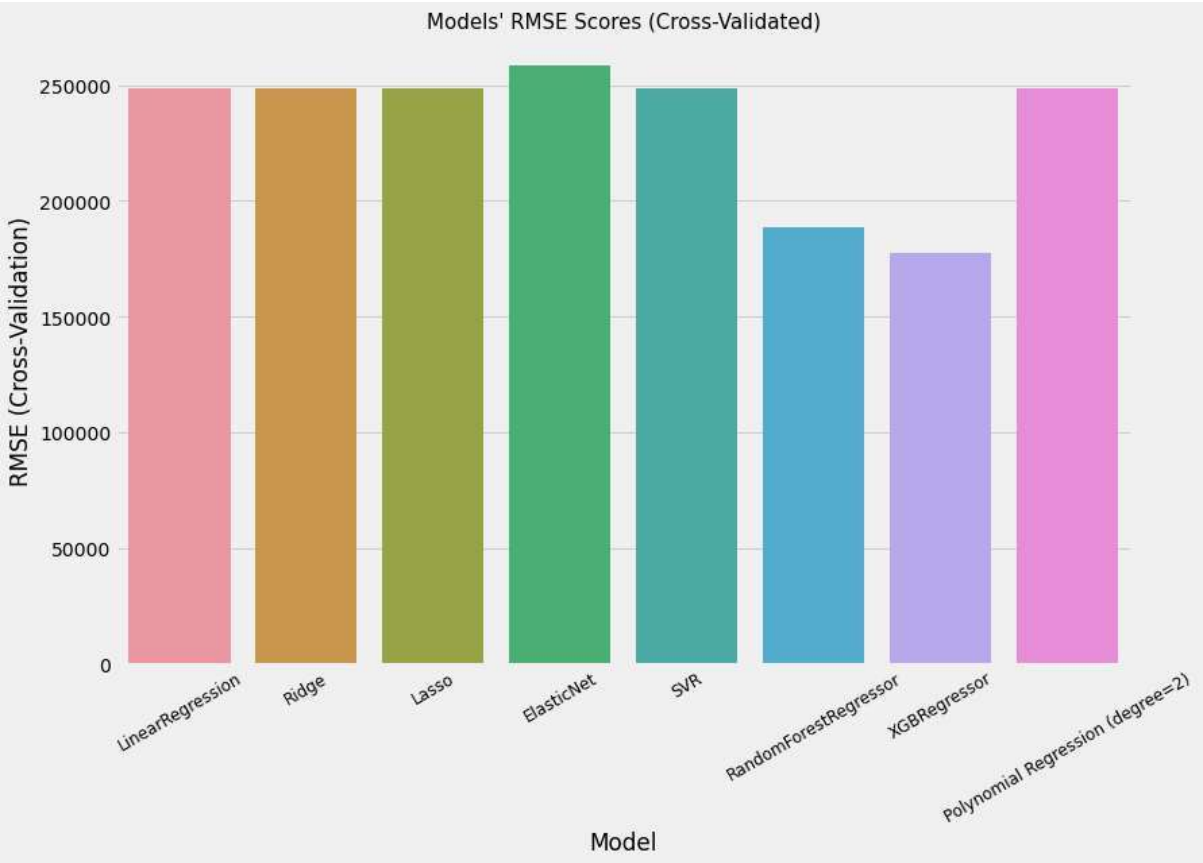
Model Comparison_VIF

In [188…]:
```
models_VIF.sort_values(by="RMSE (Cross-Validation)")
```

Out[188…]:

	Model	MAE	MSE	RMSE	R2 Score	RMSE (Cross-Validation)
6	XGBRegressor	109588.810407	3.388545e+10	184080.015062	0.721648	177662.842134
5	RandomForestRegressor	113937.409938	3.637812e+10	190730.488076	0.701172	188586.304683
4	SVR	153611.600318	6.350519e+10	252002.363955	0.478337	248661.920421
1	Ridge	164076.783812	6.364604e+10	252281.663818	0.477180	248840.402085
2	Lasso	164077.460363	6.364602e+10	252281.634957	0.477180	248840.438676
0	LinearRegression	164077.427094	6.364604e+10	252281.672118	0.477180	248840.472700
7	Polynomial Regression (degree=2)	157616.532787	6.021923e+10	245396.076806	0.505330	248840.472700
3	ElasticNet	168166.954505	6.905008e+10	262773.822645	0.432789	258668.262934

In [189…]:
```
plt.figure(figsize=(12,8))
sns.barplot(x=models_VIF["Model"], y=models_VIF["RMSE (Cross-Validation)"])
plt.title("Models' RMSE Scores (Cross-Validated)", size=15)
plt.xticks(rotation=30, size=12)
plt.show()
```



Modeling with one hot encoding

In [83]:
```
df_phase3 = df_treated
df_phase3.head()
```

Out[83]:

	price	bedroom	living_area	lot_area	total_floors	seaface	sight_viewed	condition	quality_grade	floor_area	basement_area	zipcode	l
0	1400000	4.0	2550.0	5040.0	1.5	0.0	0.0	5	8.0	1910.0	560.0	98105	.
1	615000	3.0	2360.0	7291.0	1.0	0.0	0.0	4	8.0	1360.0	560.0	98136	.
2	400000	4.0	2550.0	10660.0	1.5	0.0	0.0	3	9.0	2210.0	560.0	98092	.
3	284000	3.0	1800.0	10660.0	1.0	0.0	0.0	3	7.0	1800.0	0.0	98014	.
4	335000	3.0	1420.0	5040.0	1.0	0.0	0.0	3	8.0	1350.0	0.0	98052	.

In [84]:
```
df_phase3.columns
```

Out[84]:
```
Index(['price', 'bedroom', 'living_area', 'lot_area', 'total_floors',
       'seaface', 'sight_viewed', 'condition', 'quality_grade', 'floor_area',
       'basement_area', 'zipcode', 'latitude', 'longitude', 'living_area_2015',
       'lot_area_2015', 'furnished', 'city', 'population',
       'population_density', 'prev_sold', 'bathroom', 'house_age',
       'renovation_yrs', 'renovated_orNot', 'sold_month', 'basement_orNot',
```

```
       'price_log'],
      dtype='object')
```

In [85]:
```
df_phase3.shape
```

Out[85]:
```
(21350, 28)
```

In [86]:
```
df_p3= df_phase3.drop('price_log', axis=1)
```

In [87]:
```
# Getting dummies for columns ceil, coast, sight, condition, quality, yr_renovated, furnished
df_ohc = pd.get_dummies(df_p3, columns=['zipcode','city','sold_month'],drop_first=False)
```

In [88]:
```
df_ohc.head()
```

Out[88]:

	price	bedroom	living_area	lot_area	total_floors	seaface	sight_viewed	condition	quality_grade	floor_area	basement_area	latitude	l
0	1400000	4.0	2550.0	5040.0	1.5	0.0	0.0	5	8.0	1910.0	560.0	47.6578	
1	615000	3.0	2360.0	7291.0	1.0	0.0	0.0	4	8.0	1360.0	560.0	47.5274	
2	400000	4.0	2550.0	10660.0	1.5	0.0	0.0	3	9.0	2210.0	560.0	47.2617	
3	284000	3.0	1800.0	10660.0	1.0	0.0	0.0	3	7.0	1800.0	0.0	47.6517	
4	335000	3.0	1420.0	5040.0	1.0	0.0	0.0	3	8.0	1350.0	0.0	47.6344	

In [89]:
```
df_ohc.shape
```

Out[89]:
```
(21350, 130)
```

Model Building

In [136…
```
#Creating X, y for training and testing set
X = df_ohc.drop("price" , axis=1)
y = df_ohc["price"]
```

In [137…
```
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_state=10)
```

In [138…
```
print(X_train.shape)
print(X_val.shape)
```

```
(17080, 129)
(4270, 129)
```

In [139…
```
models_ohc = pd.DataFrame(columns=["Model","MAE","MSE","RMSE","R2 Score","RMSE (Cross-Validation)"])
```

Linear Regression

In [142…
```
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
predictions = lin_reg.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(lin_reg)
print("RMSE Cross-Validation:", rmse_cross_val)
```

```
MAE: 101175.17521127664
MSE: 28636455745.48517
RMSE: 169223.0945984772
R2 Score: 0.7729934156476528
------------------------------
RMSE Cross-Validation: 165432.176230534
```

In [143…
```
new_row = {"Model": "LinearRegression","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validati
models_ohc = models_ohc.append(new_row, ignore_index=True)
```

Ridge

In [144…
```
ridge = Ridge()
ridge.fit(X_train, y_train)
predictions = ridge.predict(X_test)
```

```
mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(ridge)
print("RMSE Cross-Validation:", rmse_cross_val)
```

```
MAE: 101181.49637062357
MSE: 28630559742.07794
RMSE: 169205.6729015843
R2 Score: 0.7730401543784056
------------------------------
RMSE Cross-Validation: 165429.112915494
```

In [145…

```
new_row = {"Model": "Ridge","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)": rmse_
models_VIF = models_VIF.append(new_row, ignore_index=True)
```

Lasso

In [146…

```
lasso = Lasso()
lasso.fit(X_train, y_train)
predictions = lasso.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(lasso)
print("RMSE Cross-Validation:", rmse_cross_val)
```

```
MAE: 101174.52064839336
MSE: 28636480529.19119
RMSE: 169223.16782636824
R2 Score: 0.7729932191825419
------------------------------
RMSE Cross-Validation: 165468.91068183884
```

In [147…

```
new_row = {"Model": "Lasso","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)": rmse_
models_ohc = models_ohc.append(new_row, ignore_index=True)
```

Elastic Net

In [148…

```
elastic_net = ElasticNet()
elastic_net.fit(X_train, y_train)
predictions = elastic_net.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(elastic_net)
print("RMSE Cross-Validation:", rmse_cross_val)
```

```
MAE: 138158.01110043743
MSE: 50883741456.46059
RMSE: 225574.24821211438
R2 Score: 0.5966349868935796
------------------------------
RMSE Cross-Validation: 218676.19054937703
```

In [149…

```
new_row = {"Model": "ElasticNet","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)":
models_ohc = models_ohc.append(new_row, ignore_index=True)
```

SVM

In [150…

```
svr = SVR(C=100000)
svr.fit(X_train, y_train)
predictions = svr.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(svr)
print("RMSE Cross-Validation:", rmse_cross_val)
```

```
MAE: 154041.22124890384
MSE: 83702624985.13995
RMSE: 289314.0594322024
R2 Score: 0.3364735088699734
```

```
----------------------------
RMSE Cross-Validation: 281061.7369464372
```

In [151…
```
new_row = {"Model": "SVR","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)": rmse_cr
models_ohc = models_ohc.append(new_row, ignore_index=True)
```

Random Forest

In [152…
```
random_forest = RandomForestRegressor(n_estimators=100)
random_forest.fit(X_train, y_train)
predictions = random_forest.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(random_forest)
print("RMSE Cross-Validation:", rmse_cross_val)
```

```
MAE: 72025.00831350507
MSE: 16640068914.330643
RMSE: 128996.39109033493
R2 Score: 0.8680910360834225
----------------------------
RMSE Cross-Validation: 126774.10747488526
```

In [153…
```
new_row = {"Model": "RandomForestRegressor","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Val
models_ohc = models_ohc.append(new_row, ignore_index=True)
```

XGBoost

In [154…
```
xgb = XGBRegressor(n_estimators=1000, learning_rate=0.01)
xgb.fit(X_train, y_train)
predictions = xgb.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(xgb)
print("RMSE Cross-Validation:", rmse_cross_val)
```

```
MAE: 69008.62580869439
MSE: 14175478416.165857
RMSE: 119060.81813999875
R2 Score: 0.887628309682787
----------------------------
RMSE Cross-Validation: 118112.88310642754
```

In [155…
```
new_row = {"Model": "XGBRegressor","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)"
models_ohc = models_ohc.append(new_row, ignore_index=True)
```

KNN Regressor

In [158…
```
from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor(n_neighbors=4,weights='distance')
knn.fit(X_train, y_train)

#predicting result over test data
predictions= knn.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(knn)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "KNN_Regressor","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)
models_ohc = models_ohc.append(new_row, ignore_index=True)
```

```
MAE: 104967.4213841638
MSE: 39134090554.58992
RMSE: 197823.3822241191
R2 Score: 0.6897766850936671
----------------------------
RMSE Cross-Validation: 196019.76684393972
```

Decision Tree Regressor

In [159…
```
from sklearn.tree import DecisionTreeRegressor
```

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

#predicting result over test data
predictions= DT.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(DT)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "DecisionTree_Regressor","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Va
models_ohc = models_ohc.append(new_row, ignore_index=True)
```

```
MAE: 105557.26510538641
MSE: 36661781514.912415
RMSE: 191472.66518987095
R2 Score: 0.7093751450270025
------------------------------
RMSE Cross-Validation: 180106.4217186778
```

Ensemble techniques

Bagging and Boosting

In [160…

```
from sklearn.ensemble import GradientBoostingRegressor, BaggingRegressor
GB = GradientBoostingRegressor(n_estimators = 200, learning_rate = 0.1, random_state=22)
GB.fit(X_train, y_train)

#predicting result over test data
predictions= GB.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(GB)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "GradientBoosting_Regressor","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cros
models_ohc = models_ohc.append(new_row, ignore_index=True)
```

```
MAE: 73105.32696490567
MSE: 15167582251.911823
RMSE: 123156.73855665317
R2 Score: 0.8797637155068454
------------------------------
RMSE Cross-Validation: 122860.53927135386
```

In [161…

```
BGG=BaggingRegressor(n_estimators=50, oob_score= True,random_state=14)
BGG.fit(X_train, y_train)

#predicting result over test data
predictions= BGG.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(BGG)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "BaggingRegressor_Regressor","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cros
models_ohc = models_ohc.append(new_row, ignore_index=True)
```

```
MAE: 73443.42651522248
MSE: 17463321019.17793
RMSE: 132148.8593184895
R2 Score: 0.8615649614168072
------------------------------
RMSE Cross-Validation: 127781.02869555743
```

In [162…

```
models_ohc.sort_values(by="RMSE (Cross-Validation)")
```

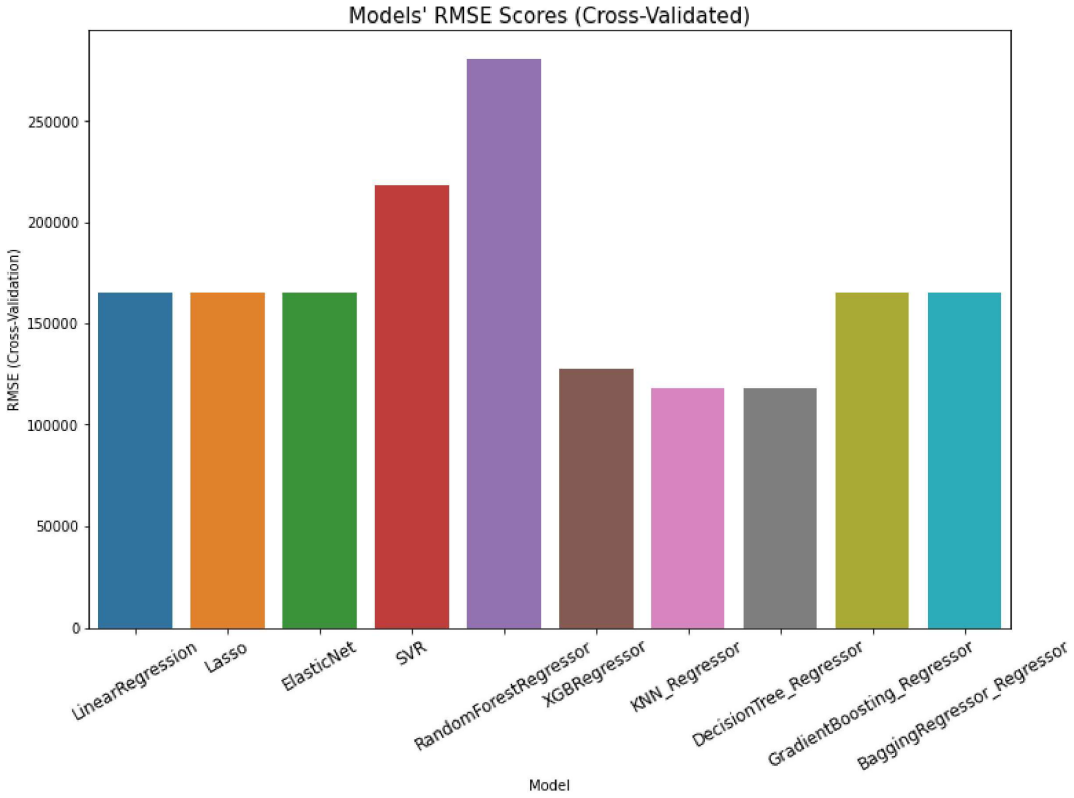
Out[162…

	Model	MAE	MSE	RMSE	R2 Score	RMSE (Cross-Validation)
5	XGBRegressor	69008.625809	1.417548e+10	119060.818140	0.887628	118112.883106
8	GradientBoosting_Regressor	73105.326965	1.516758e+10	123156.738557	0.879764	122860.539271
4	RandomForestRegressor	72025.008314	1.664007e+10	128996.391090	0.868091	126774.107475
9	BaggingRegressor_Regressor	73443.426515	1.746332e+10	132148.859318	0.861565	127781.028696
0	LinearRegression	101175.175211	2.863646e+10	169223.094598	0.772993	165432.176231

	Model	MAE	MSE	RMSE	R2 Score	RMSE (Cross-Validation)
1	Lasso	101174.520648	2.863648e+10	169223.167826	0.772993	165468.910682
7	DecisionTree_Regressor	105557.265105	3.666178e+10	191472.665190	0.709375	180106.421719
6	KNN_Regressor	104967.421384	3.913409e+10	197823.382224	0.689777	196019.766844
2	ElasticNet	138158.011100	5.088374e+10	225574.248212	0.596635	218676.190549
3	SVR	154041.221249	8.370262e+10	289314.059432	0.336474	281061.736946

In [163…

```
plt.figure(figsize=(12,8))
sns.barplot(x=models_ohc["Model"], y=models_VIF["RMSE (Cross-Validation)"])
plt.title("Models' RMSE Scores (Cross-Validated)", size=15)
plt.xticks(rotation=30, size=12)
plt.show()
```



feature importance

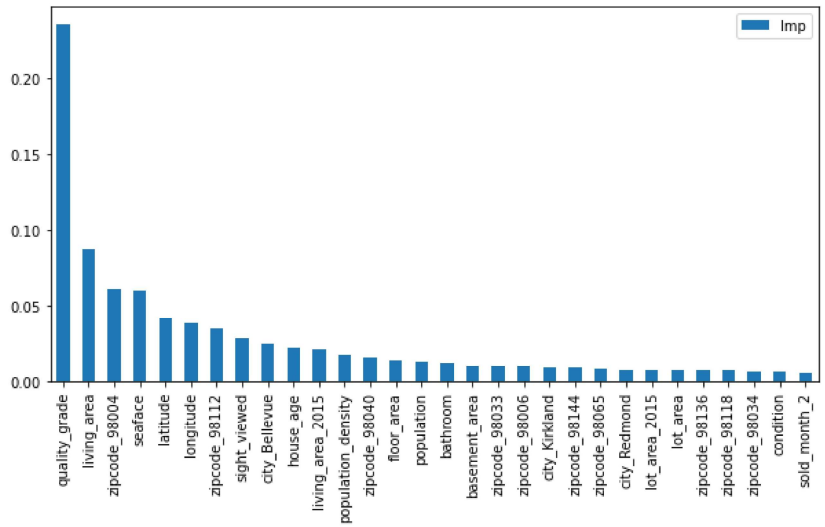
In [164…

```
rf_imp_feature_1=pd.DataFrame(xgb.feature_importances_, columns = ["Imp"], index = X_val.columns)
rf_imp_feature_1.sort_values(by="Imp",ascending=False)
rf_imp_feature_1['Imp'] = rf_imp_feature_1['Imp'].map('{0:.5f}'.format)
rf_imp_feature_1=rf_imp_feature_1.sort_values(by="Imp",ascending=False)
rf_imp_feature_1.Imp=rf_imp_feature_1.Imp.astype("float")

rf_imp_feature_1[:30].plot.bar(figsize=(10,5))

#First 20 features have an importance of 90.5% and first 30 have importance of 95.15
print("First 20 feature importance:\t",(rf_imp_feature_1[:20].sum())*100)
print("First 30 feature importance:\t",(rf_imp_feature_1[:30].sum())*100)
```

```
First 20 feature importance:     Imp    76.242
dtype: float64
First 30 feature importance:     Imp    83.347
dtype: float64
```



Above are top 30 important features that account for 95% of variation in model. We will further analyse for the hypertuning of the models for a better score.

Filtering out the important features for modeling

In [165…
```
rf_imp_feature_1[:30]
```

Out[165…

	Imp
quality_grade	0.23481
living_area	0.08709
zipcode_98004	0.06037
seaface	0.05984
latitude	0.04184
longitude	0.03856
zipcode_98112	0.03456
sight_viewed	0.02842
city_Bellevue	0.02434
house_age	0.02149
living_area_2015	0.02133
population_density	0.01705
zipcode_98040	0.01566
floor_area	0.01345
population	0.01270
bathroom	0.01194
basement_area	0.01025
zipcode_98033	0.00993
zipcode_98006	0.00985
city_Kirkland	0.00894
zipcode_98144	0.00878
zipcode_98065	0.00854
city_Redmond	0.00767
lot_area_2015	0.00725
lot_area	0.00716
zipcode_98136	0.00699
zipcode_98118	0.00689
zipcode_98034	0.00630
condition	0.00612
sold_month_2	0.00535

In [239…
```
from sklearn.pipeline import Pipeline
```

In [240…
```
def result (model,pipe_model,X_train_set,y_train_set,X_val_set,y_val_set):
    pipe_model.fit(X_train_set,y_train_set)
    #predicting result over test data
```

```
    y_train_predict= pipe_model.predict(X_train_set)
    y_val_predict= pipe_model.predict(X_val_set)

    trscore=r2_score(y_train_set,y_train_predict)
    trRMSE=np.sqrt(mean_squared_error(y_train_set,y_train_predict))
    trMSE=mean_squared_error(y_train_set,y_train_predict)
    trMAE=mean_absolute_error(y_train_set,y_train_predict)

    vlscore=r2_score(y_val,y_val_predict)
    vlRMSE=np.sqrt(mean_squared_error(y_val,y_val_predict))
    vlMSE=mean_squared_error(y_val,y_val_predict)
    vlMAE=mean_absolute_error(y_val,y_val_predict)
    result_df=pd.DataFrame({'Method':[model],'val score':vlscore,'RMSE_val':vlRMSE,'MSE_val':vlMSE,'MSE_vl': vlMSE,
                            'train Score':trscore,'RMSE_tr': trRMSE,'MSE_tr': trMSE, 'MAE_tr': trMAE})
    return result_df
```

In [241…

```
result_dff=pd.DataFrame()
pipe_LR = Pipeline([('LR', LinearRegression())])
result_dff=pd.concat([result_dff,result('LR',pipe_LR,X_train,y_train,X_val,y_val)])

pipe_knr = Pipeline([('KNNR', KNeighborsRegressor(n_neighbors=4,weights='distance'))])
result_dff=pd.concat([result_dff,result('KNNR',pipe_knr,X_train,y_train,X_val,y_val)])

pipe_DTR = Pipeline([('DTR', DecisionTreeRegressor())])
result_dff=pd.concat([result_dff,result('DTR',pipe_DTR,X_train,y_train,X_val,y_val)])

pipe_GBR = Pipeline([('GBR', GradientBoostingRegressor(n_estimators = 200, learning_rate = 0.1, random_state=22))])
result_dff=pd.concat([result_dff,result('GBR',pipe_GBR,X_train,y_train,X_val,y_val)])

pipe_BGR = Pipeline([('BGR', BaggingRegressor(n_estimators=50, oob_score= True,random_state=14))])
result_dff=pd.concat([result_dff,result('BGR',pipe_BGR,X_train,y_train,X_val,y_val)])

pipe_RFR = Pipeline([('RFR', RandomForestRegressor())])
result_dff=pd.concat([result_dff,result('RFR',pipe_RFR,X_train,y_train,X_val,y_val)])

result_dff
```

Out[241…

	Method	val score	RMSE_val	MSE_val	MSE_vl	train Score	RMSE_tr	MSE_tr	MAE_tr
0	LR	0.772993	169223.094598	2.863646e+10	2.863646e+10	0.781587	161062.731159	2.594120e+10	99077.558390
0	KNNR	0.689777	197823.382224	3.913409e+10	3.913409e+10	0.999999	262.033321	6.866146e+04	8.910246
0	DTR	0.730279	184458.005711	3.402476e+10	3.402476e+10	1.000000	81.545198	6.649619e+03	1.376698
0	GBR	0.879764	123156.738557	1.516758e+10	1.516758e+10	0.904692	106394.784858	1.131985e+10	67573.380525
0	BGR	0.861565	132148.859318	1.746332e+10	1.746332e+10	0.980068	48655.579184	2.367365e+09	27075.424393
0	RFR	0.867115	129472.744933	1.676319e+10	1.676319e+10	0.981470	46912.553621	2.200788e+09	26476.116580

Note: The best model is still from Linear Regression

Modeling Summary

- The ensemble models have performed well compared to that of linear,KNN,SVR models
- The best performance is given by Gradient boosting model with training (score-90%,RMSE- 108130), Validation (score-89.3%,RSME-109201).
- The top key features that drive the price of the property are: 'furnished','latitude', 'zipcode', 'quality_grade', living_measure','quality_8', 'HouseLandRatio', 'lot_measure15', 'quality_9', 'ceil_measure', 'total_area'. The above data is also reinforced by the analysis done during bivariate analysis. For further improvization, the datasets can be made by treating outliers in different ways and hypertuning the ensemble models.