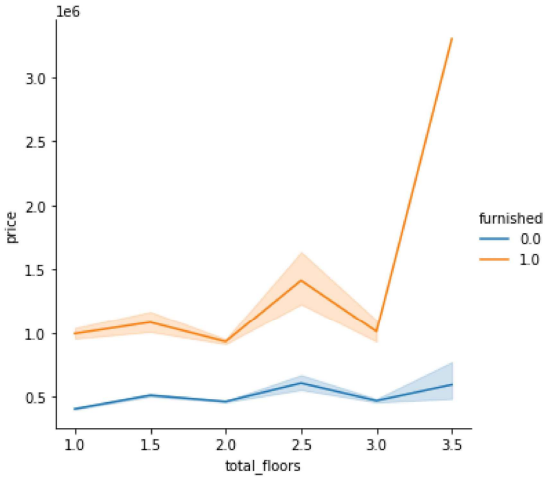


Note: This shows that the furnished house has higher cost and as the number of floors increases beyond 3 then the cost of the house shoots up for furnished house.

In [74]:
```
import missingno as msn
import folium
from folium import plugins
import branca.colormap as cm


m = folium.Map([47 ,-122], zoom_start=5,width="%100",height="%100")
locations = list(zip(dataset.latitude, dataset.longitude))
cluster = plugins.MarkerCluster(locations=locations,popups=dataset["price"].tolist())
m.add_child(cluster)
m
```

Out[74]: Make this Notebook Trusted to load map: File -> Trust Notebook

3. Data Processing

Data processing before Modeling

Imputing the missing values

In [50]:
```
dataset.isnull().sum()
```

Out[50]:
```
price                0
bedroom              0
living_area          0
lot_area             0
total_floors        30
seaface             30
sight_viewed         0
condition            0
quality_grade        0
floor_area           0
basement_area        0
zipcode              0
```

```
latitude              0
longitude            34
living_area_2015      0
lot_area_2015         0
furnished             0
total_area           39
city                  0
population            0
population_density    0
prev_sold             0
bathroom              0
house_age             0
renovation_yrs        0
renovated_orNot       0
sold_month            0
basement_orNot        0
dtype: int64
```

In [51]:
```
dataset.isnull().sum().sum()
```

Out[51]:
```
133
```

In [52]:
```
missing_col = ['total_floors', 'seaface', 'longitude', 'total_area']

# treating misiing values using median to impute the missing values
for i in missing_col:
    dataset.loc[dataset.loc[:,i].isnull(),i]=dataset.loc[:,i].median()

print("count of NULL values after imputation\n")
dataset.isnull().sum()
```

```
count of NULL values after imputation
```

Out[52]:
```
price                0
bedroom              0
living_area          0
lot_area             0
total_floors         0
seaface              0
sight_viewed         0
condition            0
quality_grade        0
floor_area           0
basement_area        0
zipcode              0
latitude             0
longitude            0
living_area_2015     0
lot_area_2015        0
furnished            0
total_area           0
city                 0
population           0
population_density   0
prev_sold            0
bathroom             0
house_age            0
renovation_yrs       0
renovated_orNot      0
sold_month           0
basement_orNot       0
dtype: int64
```

Basic Linear Regression Modeling

In [75]:
```
df=dataset
df.head(3)
```

Out[75]:

	price	bedroom	living_area	lot_area	total_floors	seaface	sight_viewed	condition	quality_grade	floor_area	basement_area	zipcode	l
0	1400000	4.0	2920.0	4000.0	1.5	0.0	0.0	5	8.0	1910.0	1010.0	98105	-
1	615000	3.0	2360.0	7291.0	1.0	0.0	0.0	4	8.0	1360.0	1000.0	98136	-
2	400000	4.0	3630.0	42884.0	1.5	0.0	0.0	3	9.0	2300.0	1330.0	98092	-

In [76]:
```
#for basic modeling we are dropping city
df=df.drop('city', axis=1)

# changing data type of zipcode from int to object
df=df.astype({'zipcode':'object'})
df.dtypes
```

Out[76]:
```
price                int64
bedroom            float64
living_area        float64
lot_area           float64
total_floors       float64
seaface            float64
```

```
sight_viewed          float64
condition               int64
quality_grade         float64
floor_area            float64
basement_area         float64
zipcode                object
latitude              float64
longitude             float64
living_area_2015      float64
lot_area_2015         float64
furnished             float64
total_area            float64
population            float64
population_density    float64
prev_sold               int64
bathroom              float64
house_age               int64
renovation_yrs          int64
renovated_orNot         int32
sold_month              int64
basement_orNot          int64
dtype: object
```

In [77]:
```
df.head(2)
```

Out[77]:

	price	bedroom	living_area	lot_area	total_floors	seaface	sight_viewed	condition	quality_grade	floor_area	basement_area	zipcode	l
0	1400000	4.0	2920.0	4000.0	1.5	0.0	0.0	5	8.0	1910.0	1010.0	98105	
1	615000	3.0	2360.0	7291.0	1.0	0.0	0.0	4	8.0	1360.0	1000.0	98136	

In [78]:
```
## outlier treatment

def remove_outlier(col):
    sorted(col)
    Q1,Q3=np.percentile(col,[25,75])
    IQR=Q3-Q1
    lower_range= Q1-(1.5 * IQR)
    upper_range= Q3+(1.5 * IQR)
    return lower_range, upper_range
```

In [79]:
```
out_df=df.drop('zipcode', axis=1).columns

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

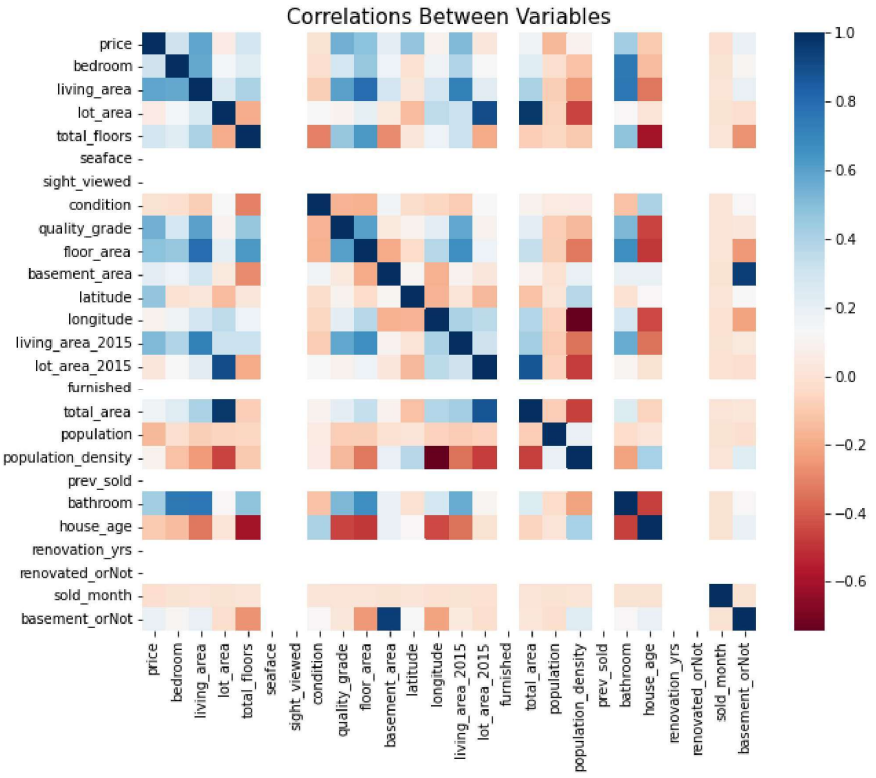
Feature Selection

In [80]:
```
df.head(3)
```

Out[80]:

	price	bedroom	living_area	lot_area	total_floors	seaface	sight_viewed	condition	quality_grade	floor_area	basement_area	zipcode
0	645000.0	4.0	2550.0	5040.0	1.5	0.0	0.0	4.0	8.0	1910.0	560.0	98105
1	615000.0	3.0	2360.0	7291.0	1.0	0.0	0.0	4.0	8.0	1360.0	560.0	98136
2	400000.0	4.0	2550.0	10687.5	1.5	0.0	0.0	3.0	8.0	2210.0	560.0	98092

In [81]:
```
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), cmap="RdBu")
plt.title("Correlations Between Variables", size=15)
plt.show()
```



In [82]:
```
important_num_cols = list(df.corr()["price"][(df.corr()["price"]>0.1) | (df.corr()["price"]<-0.1)].index)
cat_cols = ['zipcode']
important_cols = important_num_cols + cat_cols

df2 = df[important_cols]
df2.head(3)
```

Out[82]:

	price	bedroom	living_area	total_floors	quality_grade	floor_area	basement_area	latitude	living_area_2015	total_area	population	po
0	645000.0	4.0	2550.0	1.5	8.0	1910.0	560.0	47.65780	2360.0	7040.0	43471.0	
1	615000.0	3.0	2360.0	1.0	8.0	1360.0	560.0	47.52740	1860.0	9651.0	25474.0	
2	400000.0	4.0	2550.0	1.5	8.0	2210.0	560.0	47.47065	2360.0	12977.5	43471.0	

In [83]:
```
df.shape
```

Out[83]:
```
(21387, 27)
```

In [84]:
```
df2.shape
```

Out[84]:
```
(21387, 15)
```

In [85]:
```
# changing data type of zipcode from obj to int64
df2=df2.astype({'zipcode':'int64'})
df2.head(2)
```

Out[85]:

	price	bedroom	living_area	total_floors	quality_grade	floor_area	basement_area	latitude	living_area_2015	total_area	population	po
0	645000.0	4.0	2550.0	1.5	8.0	1910.0	560.0	47.6578	2360.0	7040.0	43471.0	
1	615000.0	3.0	2360.0	1.0	8.0	1360.0	560.0	47.5274	1860.0	9651.0	25474.0	

In [86]:
```
#categorical coding of zipcode

#df['zipcode'] = df_phase1['zipcode'].astype("category").cat.codes
#df.head(2)
```

In [87]:
```
## normalising the price distribution (log price)
#df2['price_log'] = np.log(df2.price+1)
#plt.figure(figsize=(7,5))
#sns.distplot(df2['price_log'], fit=norm)
#plt.title("Log-Price Distribution Plot",size=15, weight='bold')
```

In [88]:
```
#df3=df2.drop('price', axis=1)
```

X, y Split

In [89]:
```
#X = df3.drop("price_log", axis=1)
#y = df3["price_log"]
```

In [90]:
```
X = df2.drop("price", axis=1)
y = df2["price"]
```

In [91]:
```
## standarizing the data
X.columns
```

Out[91]:
```
Index(['bedroom', 'living_area', 'total_floors', 'quality_grade', 'floor_area',
       'basement_area', 'latitude', 'living_area_2015', 'total_area',
       'population', 'population_density', 'bathroom', 'basement_orNot',
       'zipcode'],
      dtype='object')
```

In [92]:
```
## scaling the data
#important_num_cols=['bedroom', 'living_area', 'total_floors', 'quality_grade', 'floor_area',
       'basement_area', 'latitude', 'living_area_2015', 'total_area', 'city',
       'population', 'population_density', 'bathroom', 'basement_orNot','zipcode']

#scaler = StandardScaler()
#X[important_num_cols] = scaler.fit_transform(X[important_num_cols])
```

```
  File "<ipython-input-92-efca2e903cc6>", line 3
    'basement_area', 'latitude', 'living_area_2015', 'total_area', 'city',
    ^
IndentationError: unexpected indent
```

In [93]:
```
X.head(2)
```

Out[93]:

	bedroom	living_area	total_floors	quality_grade	floor_area	basement_area	latitude	living_area_2015	total_area	population	population_d
0	4.0	2550.0	1.5	8.0	1910.0	560.0	47.6578	2360.0	7040.0	43471.0	
1	3.0	2360.0	1.0	8.0	1360.0	560.0	47.5274	1860.0	9651.0	25474.0	

In [94]:
```
## Train-Test Split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Linear Regression using statsmodel

In [95]:
```
# concatenate X and y into a single dataframe
data_train = pd.concat([X_train, y_train], axis=1)
data_test=pd.concat([X_test,y_test],axis=1)
data_train.head(3)
```

Out[95]:

	bedroom	living_area	total_floors	quality_grade	floor_area	basement_area	latitude	living_area_2015	total_area	population	populat
4774	3.0	1520.0	1.0	7.0	1190.0	440.0	47.47065	1720.0	10240.0	43471.0	
5749	4.0	1640.0	1.5	7.0	1190.0	560.0	47.53340	2360.0	10939.0	35722.0	
10091	3.0	1660.0	2.0	7.0	1660.0	0.0	47.47065	1810.0	7040.0	35904.0	

In [96]:
```
data_train.columns
```

Out[96]:
```
Index(['bedroom', 'living_area', 'total_floors', 'quality_grade', 'floor_area',
       'basement_area', 'latitude', 'living_area_2015', 'total_area',
       'population', 'population_density', 'bathroom', 'basement_orNot',
       'zipcode', 'price'],
      dtype='object')
```

In [97]:
```
expr= 'price ~ bedroom + living_area + total_floors + quality_grade + floor_area + basement_area + latitude + living_area
```

In [98]:
```
import statsmodels.formula.api as smf
lm1 = smf.ols(formula= expr, data = data_train).fit()
lm1.params
```

Out[98]:
```
Intercept            -1.303518e+07
bedroom               3.758522e+03
living_area           7.482800e+01
total_floors         -1.484735e+03
quality_grade         5.128605e+04
floor_area            4.430427e+01
basement_area         3.945192e+00
latitude              5.922479e+05
living_area_2015      6.112509e+01
total_area            7.466665e-01
population           -2.662449e+00
population_density    3.717019e+01
```

```
bathroom             -2.866845e+03
basement_orNot        1.917576e+04
zipcode              -1.566977e+02
dtype: float64
```

In [99]:
```
print(lm1.summary())
```

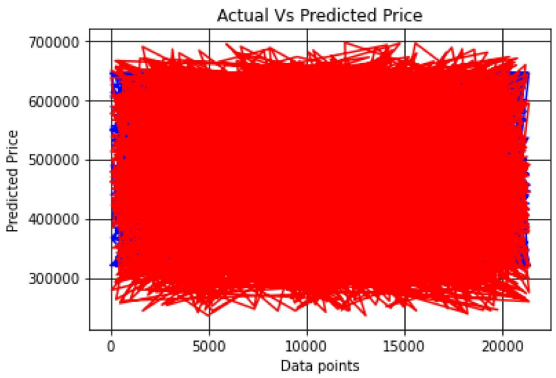
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.635
Model:                            OLS   Adj. R-squared:                  0.635
Method:                 Least Squares   F-statistic:                     2124.
Date:                Mon, 17 Jan 2022   Prob (F-statistic):               0.00
Time:                        10:54:34   Log-Likelihood:            -2.1730e+05
No. Observations:               17109   AIC:                         4.346e+05
Df Residuals:                   17094   BIC:                         4.348e+05
Df Model:                          14
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      -1.304e+07   1.41e+06     -9.272      0.000   -1.58e+07   -1.03e+07
bedroom         3758.5219   1960.847      1.917      0.055     -84.939    7601.983
living_area       74.8280      3.841     19.482      0.000      67.300      82.356
total_floors   -1484.7348   1945.650     -0.763      0.445   -5298.409    2328.939
quality_grade   5.129e+04   1701.727     30.138      0.000     4.8e+04    5.46e+04
floor_area         44.3043      4.039     10.968      0.000      36.387      52.222
basement_area       3.9452      9.041      0.436      0.663     -13.776      21.666
latitude        5.922e+05   7956.080     74.440      0.000    5.77e+05    6.08e+05
living_area_2015   61.1251      2.706     22.590      0.000      55.821      66.429
total_area          0.7467      0.334      2.233      0.026       0.091       1.402
population         -2.6624      0.088    -30.417      0.000      -2.834      -2.491
population_density  37.1702      1.259     29.517      0.000      34.702      39.639
bathroom        -2866.8453    575.095     -4.985      0.000   -3994.091   -1739.599
basement_orNot   1.918e+04   4136.395      4.636      0.000    1.11e+04    2.73e+04
zipcode          -156.6977     13.829    -11.331      0.000    -183.803    -129.592
==============================================================================
Omnibus:                       13.504   Durbin-Watson:                   2.001
Prob(Omnibus):                  0.001   Jarque-Bera (JB):               14.555
Skew:                           0.035   Prob(JB):                     0.000691
Kurtosis:                       3.124   Cond. No.                     2.42e+08
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.42e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [100…
```
# Calculate MSE
mse = np.mean((lm1.predict(data_train.drop('price',axis=1))-data_train['price'])**2)

#Root Mean Squared Error - RMSE
np.sqrt(mse)
```

Out[100…
```
79389.34449820884
```

In [101…
```
# Prediction on Test data
y_pred = lm1.predict(data_test)
```

In [102…
```
#plt.scatter(y_test['price'], y_pred)

plt.figure(figsize=(6,4))
plt.plot(y_test,"blue")
plt.plot(y_pred,"red")
plt.title("Actual Vs Predicted Price")
plt.xlabel ("Data points")
plt.ylabel ("Predicted Price");
plt.grid(True, color ="k")
plt.style.use("fivethirtyeight")
```



In [103…
```
for i,j in np.array(lm1.params.reset_index()):
    print('({}) * {} +'.format(round(j,2),i),end=' ')
```

```
(-13035181.5) * Intercept + (3758.52) * bedroom + (74.83) * living_area + (-1484.73) * total_floors + (51286.05) * qualit
y_grade + (44.3) * floor_area + (3.95) * basement_area + (592247.9) * latitude + (61.13) * living_area_2015 + (0.75) * to
tal_area + (-2.66) * population + (37.17) * population_density + (-2866.85) * bathroom + (19175.76) * basement_orNot + (-
156.7) * zipcode +
```

In [104…
```
pred_df=df
```

In [105…
```
pred_df['predicted_price']= (-13035181.5) + (3758.52) * df.bedroom + (74.83) * df.living_area + (-1484.73) * df.total_flo
```

In [106…
```
pred_df['%diff']=(pred_df.predicted_price - pred_df.price)*100/pred_df.price
```

In [107…
```
pred_df[['price','predicted_price', '%diff']]
```

Out[107…

	price	predicted_price	%diff
0	645000.0	640633	-0.677015
1	615000.0	549830	-10.5967
2	400000.0	484270	21.0676
3	321000.0	462050	43.9409
4	335000.0	434620	29.7373
...	...	...	...
21382	456500.0	490730	7.4984
21383	335000.0	348167	3.93049
21384	450000.0	500371	11.1935
21385	445500.0	444371	-0.253327
21386	645000.0	654158	1.41977

21387 rows × 3 columns

--------- lets move on to advance modeling

Advanced Modeling

Creating datasets for two phase modelling

(Phase.1 without any treatment to outliers and multi-collinarity and Phase.2 with all th treatment to outliers and multicollinearity)

In [108…
```
df_phase1=dataset ## (modeling without dealing with multicollinearity or outliers )
```

In [53]:
```
df_phase2=dataset ## (modeling with less features without signs of multi-collinearity)
```

Phase.1 Modeling without treatment of outliers or multicollinearity

Categorical encoding

In [110…
```
#categorical coding of city
df_phase1['city'] = df_phase1['city'].astype("category").cat.codes
```

In [111…
```
df_phase1.head()
```

Out[111…

	price	bedroom	living_area	lot_area	total_floors	seaface	sight_viewed	condition	quality_grade	floor_area	basement_area	zipcode	l
0	1400000	4.0	2920.0	4000.0	1.5	0.0	0.0	5	8.0	1910.0	1010.0	98105	-
1	615000	3.0	2360.0	7291.0	1.0	0.0	0.0	4	8.0	1360.0	1000.0	98136	-
2	400000	4.0	3630.0	42884.0	1.5	0.0	0.0	3	9.0	2300.0	1330.0	98092	-
3	284000	3.0	1800.0	23103.0	1.0	0.0	0.0	3	7.0	1800.0	0.0	98014	-
4	335000	2.0	1350.0	2560.0	1.0	0.0	0.0	3	8.0	1350.0	0.0	98052	-

X, y Split

Splitting the data into X and y chunks

In [112…
```
X = df_phase1.drop("price", axis=1)
y = df_phase1["price"]
```

Train-Test Split

Splitting the data into Train and Test set

In [113…

```
## splitting the data into 80:20
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Machine Learning Models

In [101…

```
## Defining several evaluation functions for convenience

def rmse_cv(model):
    rmse = np.sqrt(-cross_val_score(model, X, y, scoring="neg_mean_squared_error", cv=5)).mean()
    return rmse


def evaluation(y, predictions):
    mae = mean_absolute_error(y, predictions)
    mse = mean_squared_error(y, predictions)
    rmse = np.sqrt(mean_squared_error(y, predictions))
    r_squared = r2_score(y, predictions)
    return mae, mse, rmse, r_squared
```

In [115…

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

Model.1 Linear Regression

In [116…

```
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 = models.append(new_row, ignore_index=True)
```

```
MAE: 121010.03171941447
MSE: 38516365675.19301
RMSE: 196255.86787455046
R2 Score: 0.7278342169949374
------------------------------
RMSE Cross-Validation: 195130.60601704326
```

Model.2 Ridge Regression

In [117…

```
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 = models.append(new_row, ignore_index=True)
```

```
MAE: 121001.7036217499
MSE: 38524928693.68964
RMSE: 196277.68261748363
R2 Score: 0.7277737086735223
------------------------------
RMSE Cross-Validation: 195129.58694770918
```

Model.3 Lasso Regression

In [118…

```
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 = models.append(new_row, ignore_index=True)
```

```
MAE: 120999.41753113015
MSE: 38514812290.572495
RMSE: 196251.9102851549
R2 Score: 0.7278451935794137
------------------------------
RMSE Cross-Validation: 195105.91007165756
```

Model.3 Elastic Regression

In [119…

```
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 = models.append(new_row, ignore_index=True)
```

```
MAE: 134349.7921715217
MSE: 48100756051.6868
RMSE: 219318.84563732045
R2 Score: 0.6601086394198786
------------------------------
RMSE Cross-Validation: 216292.8395180516
```

Model.4 Support Vector Machines

In [120…

```
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 = models.append(new_row, ignore_index=True)
```

```
MAE: 159320.25083425205
MSE: 88393760518.94342
RMSE: 297310.881938323
R2 Score: 0.3753887049656156
------------------------------
RMSE Cross-Validation: 288353.4531623061
```

Model.5 Random Forest Regressor

In [121…

```
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 = models.append(new_row, ignore_index=True)
```

```
MAE: 69847.44695059996
MSE: 18053246322.23379
RMSE: 134362.3694426151
R2 Score: 0.8724314759468956
------------------------------
RMSE Cross-Validation: 130698.27352513644
```

Mode.6 XGBoost Regressor

In [122…

```
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 = models.append(new_row, ignore_index=True)
```

```
MAE: 66778.23343633123
MSE: 16323711974.466715
RMSE: 127764.2828589693
R2 Score: 0.8846527762108857
------------------------------
RMSE Cross-Validation: 123340.5247554143
```

Model.7 Polynomial Regression (Degree=2)

In [123…

```
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 = models.append(new_row, ignore_index=True)
```

```
MAE: 95224.24547185627
MSE: 24444305909.86882
RMSE: 156346.74895842516
R2 Score: 0.8272707317694925
------------------------------
RMSE Cross-Validation: 195130.60601704326
```

Model Comparison

1. Mean Absolute Error (MAE) shows the difference between predictions and actual values.
2. Mean squared error (MSE) tells you how close a regression line is to a set of points
3. Root Mean Square Error (RMSE) shows how accurately the model predicts the response.
4. R^2 will be calculated to find the goodness of fit measure.
5. RMSE cross validation: The less the Root Mean Squared Error (RMSE), The better the model is.

In [124…

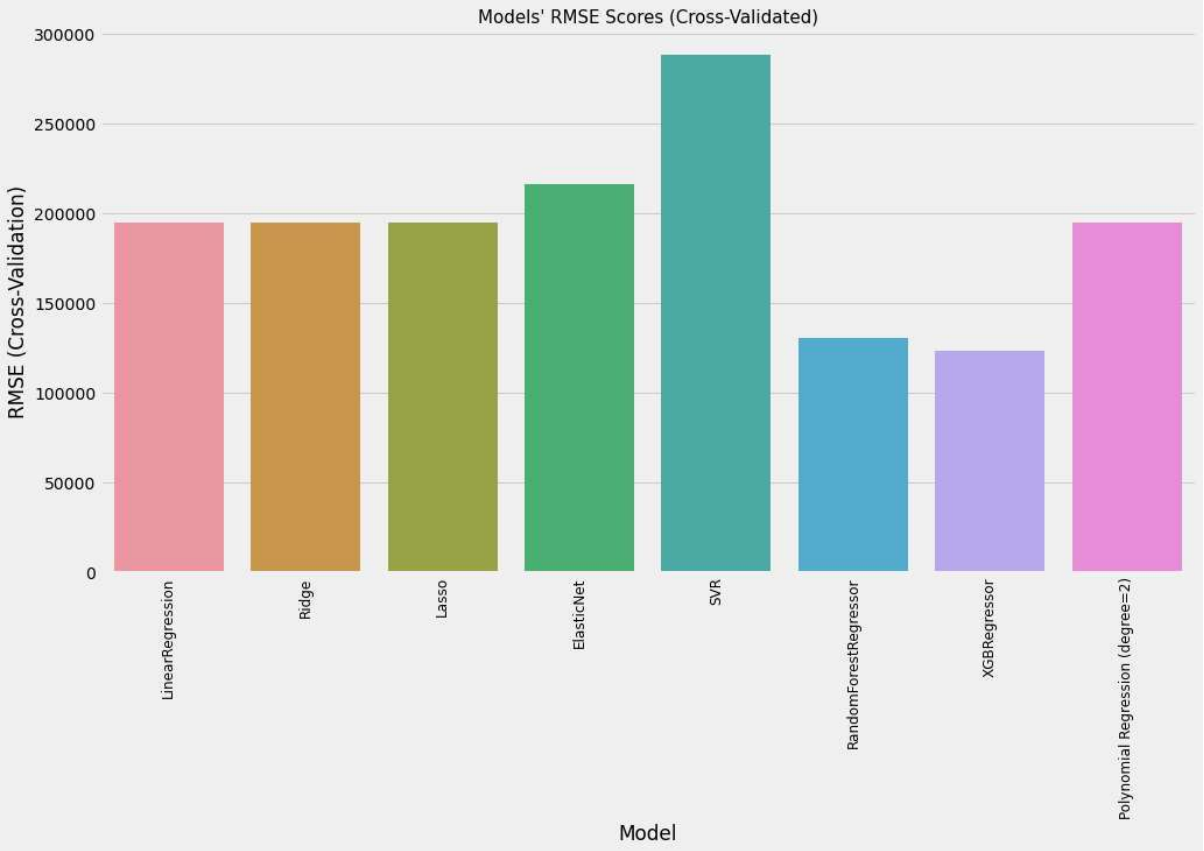
```
models.sort_values(by="RMSE (Cross-Validation)")
```

Out[124…

	Model	MAE	MSE	RMSE	R2 Score	RMSE (Cross-Validation)
6	XGBRegressor	66778.233436	1.632371e+10	127764.282859	0.884653	123340.524755
5	RandomForestRegressor	69847.446951	1.805325e+10	134362.369443	0.872431	130698.273525
2	Lasso	120999.417531	3.851481e+10	196251.910285	0.727845	195105.910072
1	Ridge	121001.703622	3.852493e+10	196277.682617	0.727774	195129.586948
0	LinearRegression	121010.031719	3.851637e+10	196255.867875	0.727834	195130.606017
7	Polynomial Regression (degree=2)	95224.245472	2.444431e+10	156346.748958	0.827271	195130.606017
3	ElasticNet	134349.792172	4.810076e+10	219318.845637	0.660109	216292.839518
4	SVR	159320.250834	8.839376e+10	297310.881938	0.375389	288353.453162

In [125…

```
plt.figure(figsize=(15,8))
sns.barplot(x=models["Model"], y=models["RMSE (Cross-Validation)"])
plt.title("Models' RMSE Scores (Cross-Validated)", size=15)
plt.xticks(rotation=90, size=12)
plt.show()
```



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()
```