Predicting King County House Prices With
Multiple Linear Regression

By Foo ZheShen

Introduction

This report discusses the steps, procedures, assumptions, and validation methods used for my analysis.

For this project, I am tasked by the Real Estate Agency to identify key features of a home when trying to predict the house prices for future use when dealing with clients looking to sell or buy houses in King County.

I will be building a Multiple Regression Model to analyze and predict house sales in a northwestern county using the King County Sales dataset that has been provided.

Exploring the Data

Before building our model, we explore the King County Sales dataset.

The objective of exploring the provided dataset is to:

1. Make assumptions.
2. Identify the dependant variables.
3. Identify the independent variables.

Column Names and descriptions for Kings County Data Set

- id - unique identified for a house
- dateDate - house was sold
- pricePrice - is prediction target
- bedroomsNumber - of Bedrooms/House
- bathroomsNumber - of bathrooms/bedrooms
- sqft_livingsquare - footage of the home
- sqft_lotsquare - footage of the lot
- floorsTotal - floors (levels) in house
- waterfront - House which has a view to a waterfront
- view - Has been viewed
- condition - How good the condition is (Overall)
- grade - overall grade given to the housing unit, based on King County grading system
- sqft_above - square footage of house apart from basement
- sqft_basement - square footage of the basement
- yr_built - Built Year
- yr_renovated - Year when house was renovated
- zipcode - zip
- lat - Latitude coordinate
- long - Longitude coordinate
- sqft_living15 - The square footage of interior housing living space for the nearest 15 neighbors
- sqft_lot15 - The square footage of the land lots of the nearest 15 neighbors

Cleaning the Data

Standard data cleaning procedures were conducted.

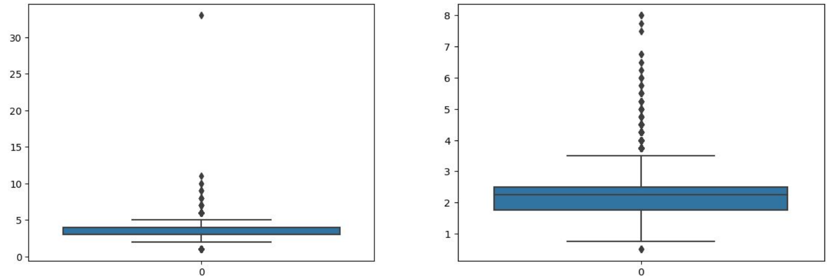
In addition, we also look for outliers in the data since linear models like linear regression and logistic regression tend to be easily influenced by outliers.

Using describe() we can see the distribution of column data.

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition
count	2.159400e+04	21594.000000	21594.000000	21594.000000	2.159400e+04	21594.000000	21594.000000	21594.000000	21594.000000
mean	5.402934e+05	3.373206	2.115808	2080.303371	1.510073e+04	1.494026	0.006761	0.233074	3.409836
std	3.673935e+05	0.926346	0.769025	918.165554	4.141535e+04	0.539687	0.081950	0.764491	0.650566
min	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	1.000000	0.000000	0.000000	1.000000
25%	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03	1.000000	0.000000	0.000000	3.000000
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.619000e+03	1.500000	0.000000	0.000000	3.000000
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068650e+04	2.000000	0.000000	0.000000	4.000000
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	5.000000

We can take a closer look at bedrooms and bathrooms.

Boxplot for Bedroom and Bathroom outliers



We can remove those outliers by identifying the upper and lower limit using 3 standards from the mean.

```
bed_std = np.std(kc_preprocess['bedrooms']) * 3
bed_mean = np.mean(kc_preprocess['bedrooms'])
bed_upperlimit = bed_mean + bed_std
bed_lowerlimit = bed_mean - bed_std

kc_data = kc_preprocess[(kc_preprocess['bedrooms'] < bed_upperlimit) & (kc_preprocess['bedrooms'] > bed_lowerlimit)]
kc_data.describe()
```

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
count	2.136000e+04	21360.000000	21360.000000	21360.000000	2.136000e+04	21360.000000
mean	5.291362e+05	3.347893	2.089291	2051.297659	1.490721e+04	1.489419
std	3.295941e+05	0.865500	0.722499	861.876026	4.082657e+04	0.538799
min	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	1.000000
25%	3.200000e+05	3.000000	1.500000	1420.000000	5.030000e+03	1.000000
50%	4.500000e+05	3.000000	2.250000	1900.000000	7.590000e+03	1.500000
75%	6.370000e+05	4.000000	2.500000	2520.000000	1.058400e+04	2.000000
max	4.490000e+06	6.000000	4.250000	7850.000000	1.651359e+06	3.500000

Removing these outliers likely also excluded rows with extreme values for other variables.

Assumptions

Once the data is clean and processed, we can make some assumptions to be included to the model

Since the project is specifically targeted toward house prices in the King County area, some columns are dropped from the dataset to reduce the complexity of the model;

1. Id - Sales id
2. Date - Date of sale
3. Lat - Latitude
4. Long - Longitude

Next, To simplify variable for renovations, I have converted data for yr_renovated into a boolean.

```
In [8]: kc_preprocess['yr_renovated'] = kc_preprocess['yr_renovated'] > 0 #converting to boolean
        kc_preprocess['yr_renovated'] = kc_preprocess['yr_renovated'].astype('int') #changing to binary format
        kc_preprocess = kc_preprocess.rename(columns = {'yr_renovated':'renovated'})
        kc_preprocess['renovated']

Out[8]: 0        0
        1        1
        2        0
        3        0
        4        0
                ..
        21592    0
        21593    0
        21594    0
        21595    0
        21596    0
        Name: renovated, Length: 21597, dtype: int32
```

Note* For future works, we should include the time between the last renovation and the time of sale.

Identifying Variables

From the figures of columns plotted against price, we can easily identify the nature of the variables we are working with.

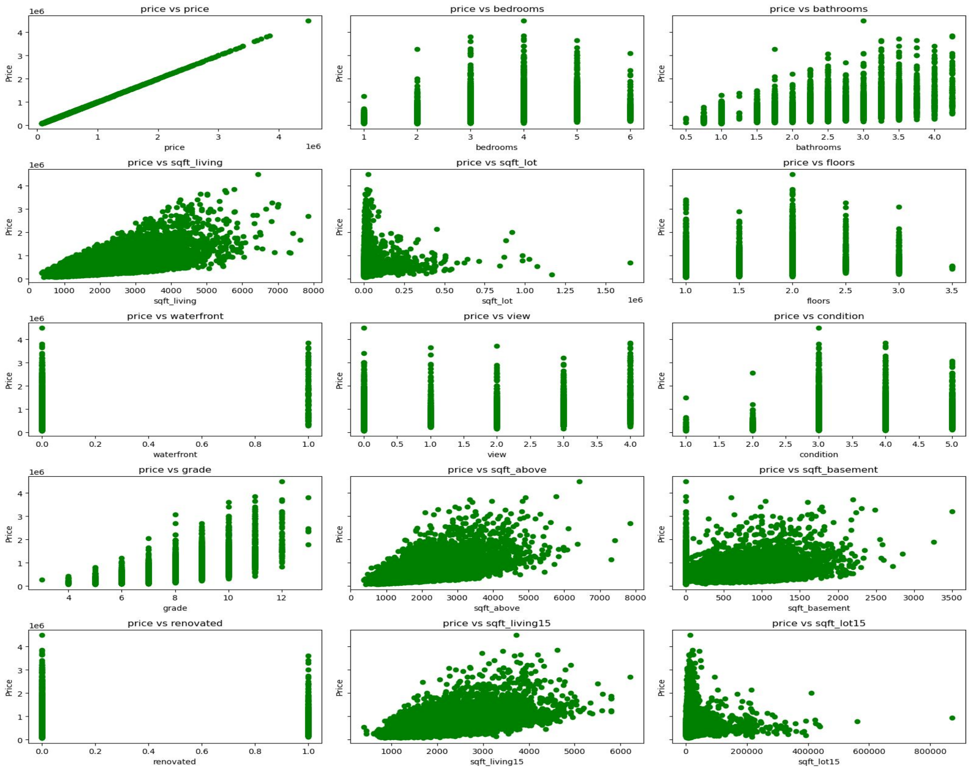
Our target: Price

Continuous Variables :

- ➔ Sqft_living
- ➔ Sqft_lot
- ➔ Sqft_above
- ➔ Sqft_basement
- ➔ Sqft_living15
- ➔ sqft_lot15

Categorical Variables:

- ➔ Bedrooms
- ➔ Bathrooms
- ➔ Floors
- ➔ Waterfront
- ➔ View
- ➔ Condition
- ➔ Grade
- ➔ Renovated



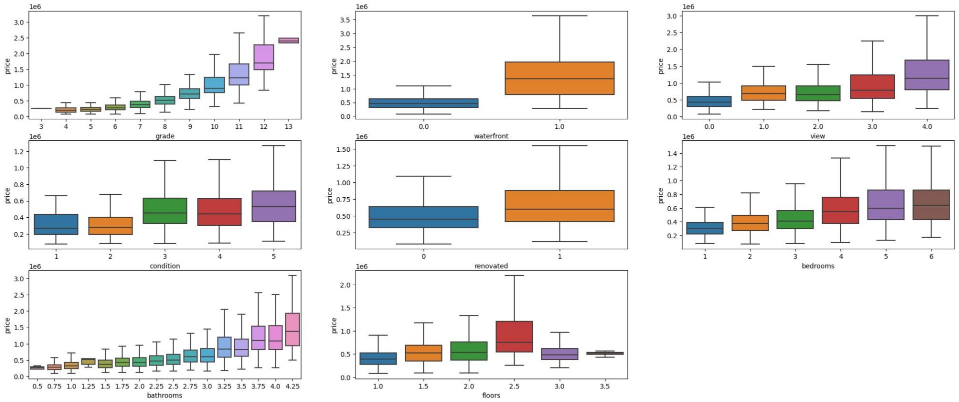
Feature transformations: Categorical Variables

1. One-hot Encoding:

One hot encoding is a technique that we use to represent categorical variables as numerical values in a machine learning model.

Even though the variables we identify as categorical are in integers (i.e. ‘bedrooms’), the number of possible values is often limited to a fixed set.

```python
bed_dummy = pd.get_dummies(kc_data['bedrooms'], prefix = 'bedrooms', drop_first=True, dtype = 'int')
#drop a column to avoid dummy variable trap
kc_data = kc_data.drop('bedrooms', axis = 1)
kc_data = kc_data.join(bed_dummy)
```



The boxplot means shows a trend, which indicate these variables may be useful as predictors of the model.

With one-hot, we convert each categorical value into a new categorical column and assign a binary value of 1 or 0 to those columns. Each integer value is represented as a binary vector.

bedrooms_2	bedrooms_3	bedrooms_4	...	waterfront1	view1	view2	view3	view4	cond2	cond3	cond4	cond5	renovated_1
0	1	0	...	0	0	0	0	0	0	1	0	0	0
0	1	0	...	0	0	0	0	0	0	1	0	0	1
1	0	0	...	0	0	0	0	0	0	1	0	0	0
0	0	1	...	0	0	0	0	0	0	0	0	1	0
0	1	0	...	0	0	0	0	0	0	1	0	0	0

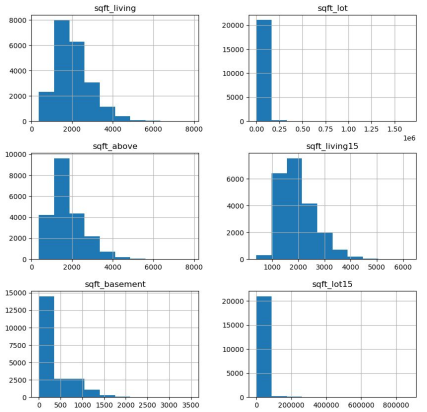
Feature transformations: Continuous Variables

When analyzing regression results, it's important to ensure that the residuals have a constant variance.

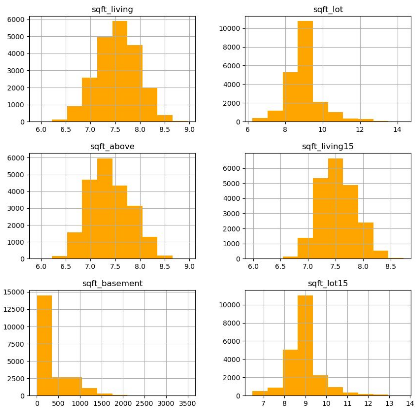
2. Log-Transform

We transform the skewed continuous data to approximately conform to normality and to counter problems in heteroskedasticity.

Before Log-Transform:



After Log-Transform:



Feature transformations: Continuous Variables

3. Feature Scaling

By reducing biases caused by value weight, we can improve our regression model by ensuring that all features are on a similar scale, preventing one feature from dominating the others in the optimization process.

A snippet of our dataset after both transformations:

```
kc_data[log_varr] = np.log(kc_data[log_varr])

kc_data.head()
```

	price	sqft_living	sqft_lot	sqft_above	sqft_basement	sqft_living15	sqft_lot15	bedrooms_2	bedrooms_3	bedrooms_4	...	waterfront1	view1	view2
0	12.309982	7.073270	8.639411	7.073270	0.0	7.200425	8.639411	0	1	0	...	0	0	0
1	13.195614	7.851661	8.887653	7.682482	400.0	7.432484	8.941022	0	1	0	...	0	0	0
2	12.100712	6.646391	9.210340	6.646391	0.0	7.908387	8.994917	1	0	0	...	0	0	0
3	13.311329	7.580700	8.517193	6.956545	910.0	7.215240	8.517193	0	0	1	...	0	0	0
4	13.142166	7.426549	8.997147	7.426549	0.0	7.495542	8.923058	0	1	0	...	0	0	0

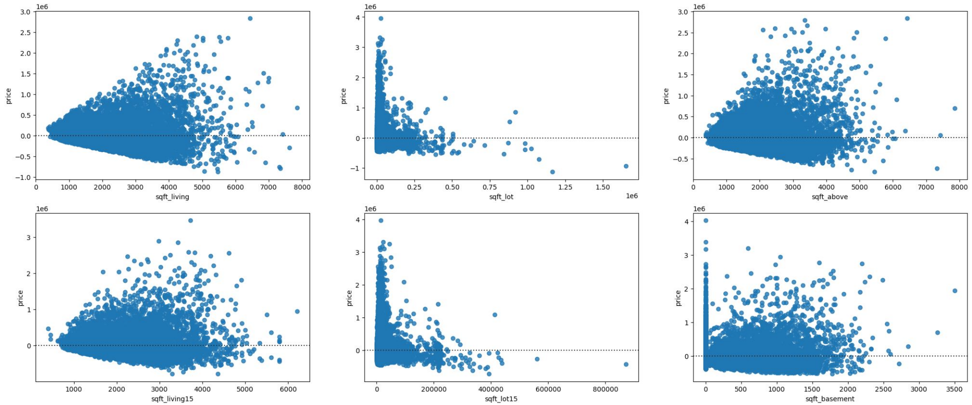
Homoscedascity Check

Homoscedasticity occurs when the variance in a dataset is constant, making it easier to estimate the standard deviation and variance of a data set.

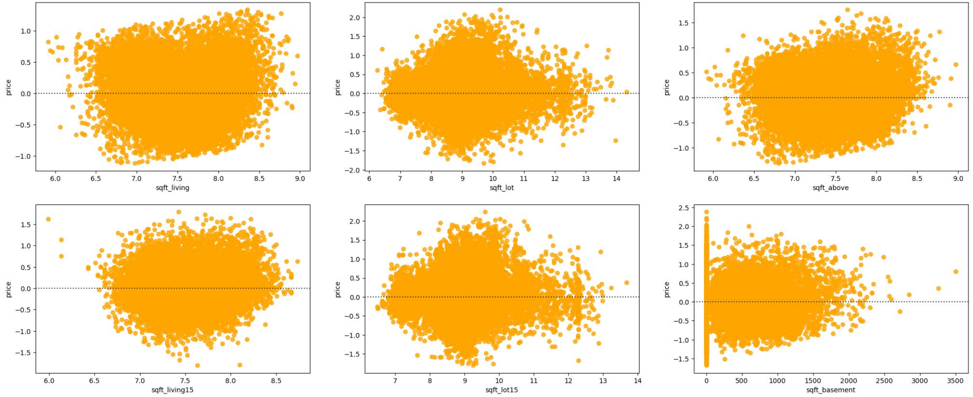
To validate the appropriateness of a linear regression analysis, homoscedasticity must not be violated outside a certain tolerance.

Hence, all continuous variables shall undergo log transformation and check for homoscedasticity.

Before Log-Transform: Mostly Heteroscedastic



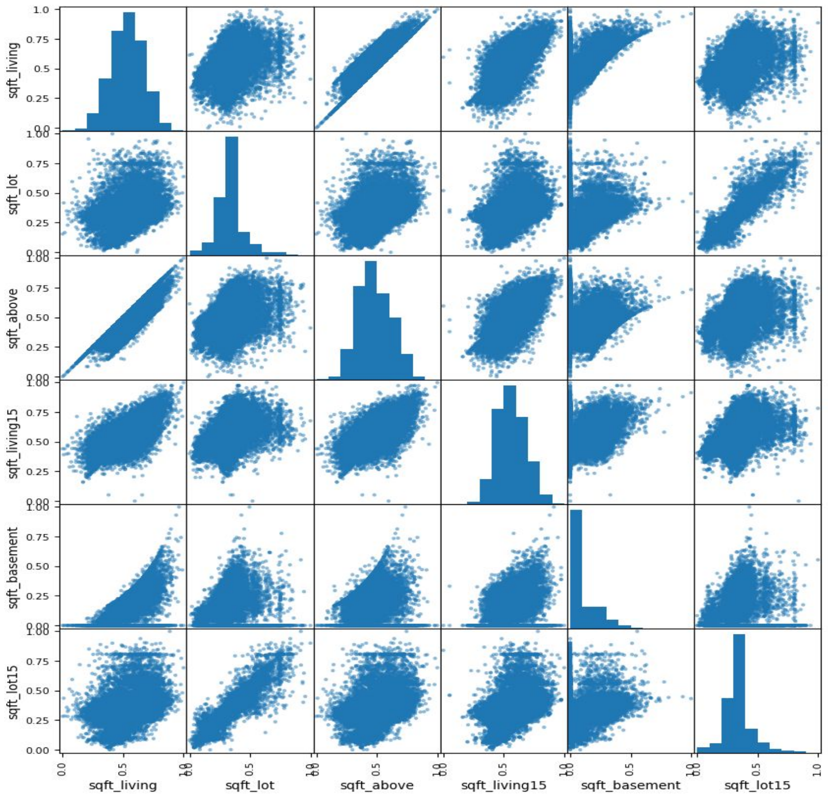
After Log-Transform: Mostly Homoscedastic



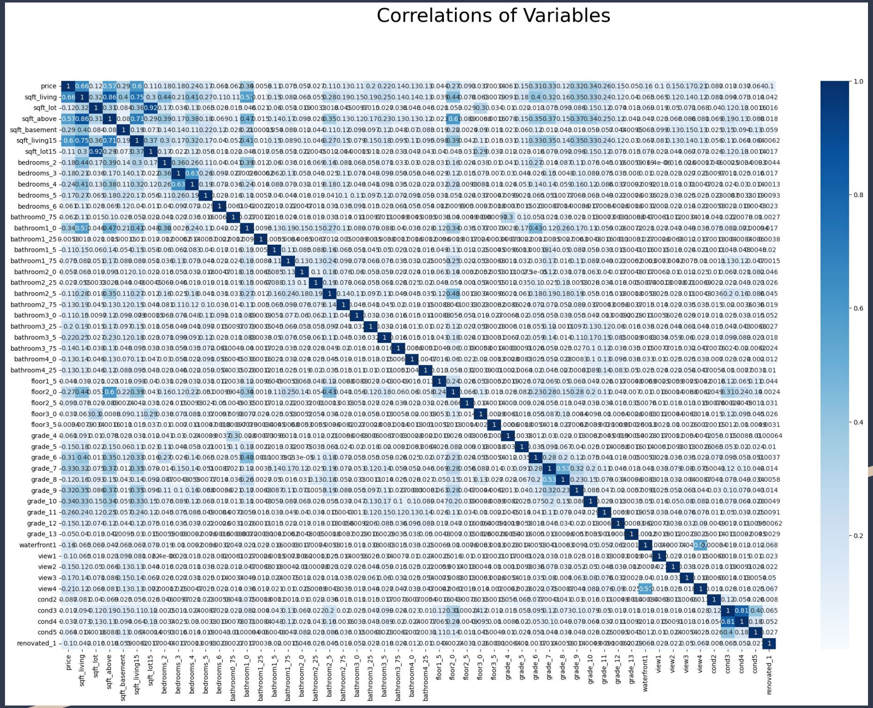
Multicollinearity

To the left shows the correlations of continuous variables from the dataset.

From the figure, we can already see there exist strong linear relationship among the predictor variables.



Looking for Multicollinearity



From the figure to the right and the snippet below,we can gather there exist high multicollinearity between multiple features.

In [36]:
```
'''
checking multicolinearity between independant variables from the heatmap
'''

features_multicor = []
correlations_multicor = []

for column in corr_var:                                      #for each variable(column) in the
    for index, correlation in corr_var[column].items():       #for index & correlation(data) in the column
        if correlation >= .70 and index != column:
            features_multicor.append([column, index])
            correlations_multicor.append(correlation)

MC_df = pd.DataFrame({'Correlations':correlations_multicor, 'Features': features_multicor}).sort_values(by=['Correlations'], asce
MC_df.drop_duplicates('Correlations')
```

Out[36]:

	Correlations	Features
2	0.919473	[sqft_lot, sqft_lot15]
0	0.859457	[sqft_living, sqft_above]
8	0.812092	[cond3, cond4]
1	0.745434	[sqft_living, sqft_living15]
4	0.710436	[sqft_above, sqft_living15]

Since it will be hard to gather information from the figure. We shall look for multicollinearity using VIF Score while building our model iterations.

```
X = x_train

# VIF dataframe
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns

# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(X.values, i)
                          for i in range(len(X.columns))]

print('Max VIF:', vif_data[vif_data['VIF'] == vif_data['VIF'].max()])
print()
print(vif_data)
```

```
Max VIF:     feature          VIF
34  grade_7  1951.171528
```

The Final Model

Finally, we arrived to Model 3.

```
features_model3 = features_model2.drop(labels = [1,2,4,5,6,8,9,10,11,13,14,16,17,19,20,27,30,47])

outcome = 'price'
model3_predictors = '+'.join(features_model3)

formula_model3 = outcome + '~' + model3_predictors
Model_3 = ols(formula=formula_model3, data = df_train).fit()
Model_3.summary()
```

We have dropped multiple features since and have able to remove all low significant (P-Value) features.

We also managed to heavily reduce the VIF score of all available features to the model which will be shown next.

The predictors of the Final Model:

```
0        sqft_living
3      sqft_basement
7        bedrooms_3
12      bathroom1_0
15     bathroom1_75
18      bathroom2_5
21     bathroom3_25
22      bathroom3_5
23     bathroom3_75
24      bathroom4_0
25     bathroom4_25
26         floor1_5
28         floor2_5
29         floor3_0
31          grade_4
32          grade_5
33          grade_6
35          grade_8
36          grade_9
37         grade_10
38         grade_11
39         grade_12
40         grade_13
41      waterfront1
42            view1
43            view2
44            view3
45            view4
46            cond2
48            cond4
49            cond5
50      renovated_1
```

The Final Model

R-squared: 0.603

All P-Value of features are under 0.05 which rejects the null hypothesis.

Almost all features in Model 3 are less that 5 indicating low correlation with each other.

There is moderate correlation on 'sqft_living', however we choose not to remove the feature as it is understandable that a feature that indicates housing size would be important factor to predicting house prices.

Note

A VIF less than 5 indicates a low correlation of that predictor with other predictors.

A value between 5 and 10 indicates a moderate correlation, while VIF values larger than 10 are a sign for high, not tolerable correlation of model predictors

VIF Score:

	feature	VIF
0	sqft_living	8.144452
1	sqft_basement	1.974745
2	bedrooms_3	1.878636
3	bathroom1_0	1.790987
4	bathroom1_75	1.419301
5	bathroom2_5	1.991328
6	bathroom3_25	1.184751
7	bathroom3_5	1.285645
8	bathroom3_75	1.075324
9	bathroom4_0	1.086666
10	bathroom4_25	1.076125
11	floor1_5	1.169135
12	floor2_5	1.025436
13	floor3_0	1.084149
14	grade_4	1.011745
15	grade_5	1.056676
16	grade_6	1.431390
17	grade_8	2.117567
18	grade_9	1.828170
19	grade_10	1.520934
20	grade_11	1.304264
21	grade_12	1.095581
22	grade_13	1.023375
23	waterfront1	1.508447
24	view1	1.036224
25	view2	1.097323
26	view3	1.088967
27	view4	1.573916
28	cond2	1.034988
29	cond4	1.549347
30	cond5	1.197511
31	renovated_1	1.063944

Dep. Variable:	price	R-squared:	0.603
Model:	OLS	Adj. R-squared:	0.603
Method:	Least Squares	F-statistic:	709.2
Date:	Wed, 17 Apr 2024	Prob (F-statistic):	0.00
Time:	18:54:41	Log-Likelihood:	-4439.7
No. Observations:	14951	AIC:	8945.
Df Residuals:	14918	BIC:	9197.
Df Model:	32		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	12.2561	0.020	614.781	0.000	12.217	12.295
sqft_living	1.0446	0.038	27.506	0.000	0.970	1.119
sqft_basement	0.2435	0.026	9.194	0.000	0.192	0.295
bedrooms_3	-0.0439	0.006	-7.765	0.000	-0.055	-0.033
bathroom1_0	0.0519	0.010	5.307	0.000	0.033	0.071
bathroom1_75	0.0184	0.009	2.146	0.032	0.002	0.035
bathroom2_5	-0.0425	0.008	-5.660	0.000	-0.057	-0.028
bathroom3_25	0.0823	0.018	4.659	0.000	0.048	0.117
bathroom3_5	0.0490	0.016	3.020	0.003	0.017	0.081
bathroom3_75	0.1566	0.032	4.892	0.000	0.094	0.219
bathroom4_0	0.1907	0.038	5.008	0.000	0.116	0.265
bathroom4_25	0.1630	0.048	3.369	0.001	0.068	0.258
floor1_5	0.1790	0.010	18.389	0.000	0.160	0.198
floor2_5	0.1816	0.032	5.635	0.000	0.118	0.245
floor3_0	0.1337	0.017	7.946	0.000	0.101	0.167
grade_4	-0.3038	0.078	-3.887	0.000	-0.457	-0.151
grade_5	-0.3881	0.027	-14.431	0.000	-0.441	-0.335
grade_6	-0.2089	0.011	-19.702	0.000	-0.230	-0.188
grade_8	0.2175	0.007	29.597	0.000	0.203	0.232
grade_9	0.4627	0.011	42.816	0.000	0.442	0.484
grade_10	0.6419	0.015	42.158	0.000	0.612	0.672
grade_11	0.8067	0.025	31.971	0.000	0.757	0.856
grade_12	0.9802	0.048	20.272	0.000	0.885	1.075
grade_13	1.2064	0.191	6.330	0.000	0.833	1.580
waterfront1	0.3487	0.042	8.399	0.000	0.267	0.430
view1	0.2159	0.022	9.846	0.000	0.173	0.259
view2	0.1391	0.013	10.511	0.000	0.113	0.165
view3	0.1802	0.018	9.920	0.000	0.145	0.216
view4	0.3058	0.028	10.809	0.000	0.250	0.361
cond2	-0.1277	0.032	-4.036	0.000	-0.190	-0.066
cond4	0.0630	0.006	9.740	0.000	0.050	0.076
cond5	0.1770	0.010	16.962	0.000	0.157	0.197
renovated_1	0.1941	0.015	12.932	0.000	0.165	0.224

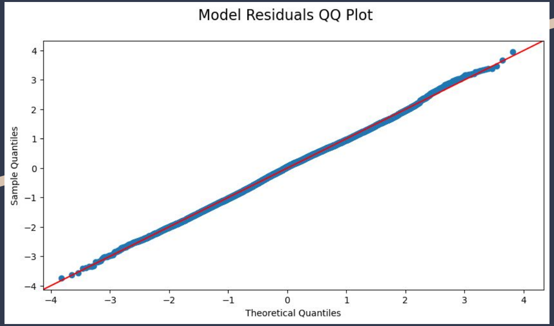
Omnibus:	2.346	Durbin-Watson:	2.022
Prob(Omnibus):	0.309	Jarque-Bera (JB):	2.337
Skew:	-0.031	Prob(JB):	0.311
Kurtosis:	3.005	Cond. No.	97.8

Model Testing & Evaluation

Residuals are the error between a predicted value and the observed actual value.

Residual Normality Check:

1. QQ Plot
2. Distribution Plot
3. Residual Plot

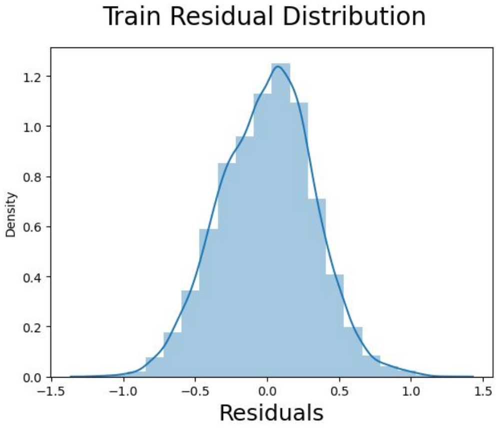


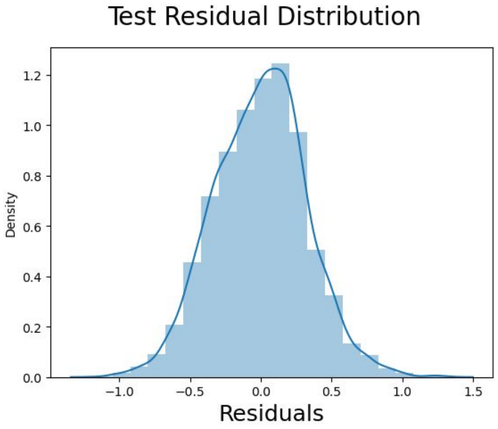
Almost all points falls along the QQ line.

This shows the residuals have little to no deviations.

Both residual from both train and test data shows a mostly normal distribution.

Hence, Model 3 residuals are normally distributed and satisfy the normality assumptions.



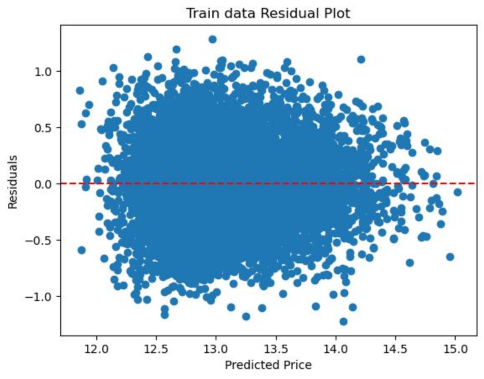


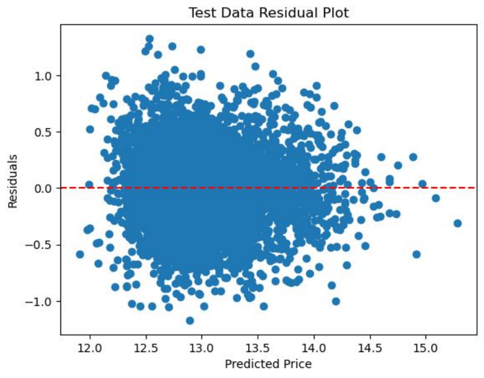
Model Testing and Validation

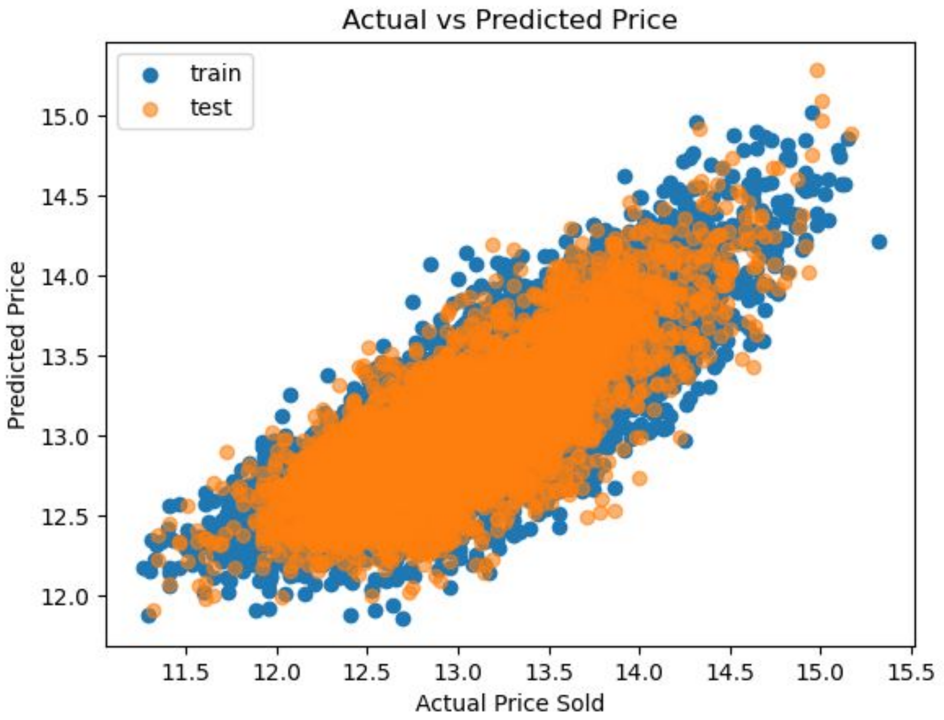
Residual Plot:

A residual plot shows the difference between the observed response and the fitted response values.

A well-performing model will have residuals scattered randomly around zero.







Results

The results show the R-squared and its adjusted value of the train and test data;

- R-squared for train: 0.6033651270434894
- R-Squared for test: 0.5898111024076923
- Adjusted R-Squared for test: 0.6025143215779707
- Adjusted R-squared for train: 0.5877521150001701
- R-squared for train and test data accounts for about 60.3% and 58.9% respectively.
- Adjusted R-squared for train and test data accounts for about 60.2% and 58.7% respectively.

Both train and test sets are very close. Hence, the final model can predict house prices with an accuracy of nearly 60% when fitted with new data.

Conclusion

From the model, we gather;

Features that impact house price prediction in King County:

1. The size of the living space and basement ('sqft_living & sqft_basement')
2. The number of bedrooms and bathrooms
3. The grade of the house.
4. House condition.
5. The view the property has.
6. If the house was renovated before.
7. If the house is close to water.

Limitations of the Model:

For this model, we did not address how much time the house has since been renovated, and thus any recent renovations that may impact the price were not considered.

Given that some of the features needed to be log-transformed to satisfy regression assumptions, new data used with the model would have to undergo similar preprocessing.

Additionally, the model was built on a dataset solely belonging to the region of King County, the model's applicability to data from other counties may be limited.

Furthermore, some outliers were removed, when fitted with new values, the model may also not accurately predict extreme values.

Future works:

Future models could include an interaction variable of date built and date renovated. Additionally, we can also explore the long and lat of houses to discover the actual distance from water and how prices may be affected.

For the code and scripts used for this project,

https://github.com/FooZheShen/KC_house_price_prediction_using_ML