Final Project Submission

Please fill out:

- Student name: Lucas Fishbein
- Student pace: Flex Program
- Scheduled project review date/time: 3/10/23 1:00PM
- Instructor name: Mark Barbour
- Blog post URL: https://medium.com/@fishbeinlucas/the-common-pitfalls-of-multilinear-regression-models-and-how-to-deal-with-them-353be6054b08 (https://medium.com/@fishbeinlucas/the-common-pitfalls-of-multilinear-regression-models-and-how-to-deal-with-them-353be6054b08)

Overview and Business Problem

The Stakeholders of interest will be a residental home construction team within King County, WA. Their main focus will be procuring land and then constructing residental buildings for the purpose of selling them to turn a profit. These companies will want to know what features are most imporant in maximizing home values upon their sale in order to dictate where they buy land and to what standards they should build them.

The purpose of this data analysis is to build a model that will predict the price of constructed homes for a builder located in Kings County, WA. From this model, the most vaulable features of a home will be identified and from there recomendations can be given to the building team for choosing locations and making construction decisions that will maximize the value of a home.

Photo Source: Istockphoto.com (istockphoto.com/illustrations/house-construction-frame)

Data Analysis Quick Results

Question 1: What factors coorelate most with the sale price of a home?

1. Square footage of space inside the home
2. Square footage of house apart from basement
3. Number Of Bathrooms
4. Price per Square Foot of space inside the house
5. Number of Bedrooms

Question 2: Can we build a model to predict home prices?

Yes, a model has been built in which every feature has a significant relationship with price and accounts for 55% of the variance in the sales price, the average error of this model is about half a standard deviation. This model is a moderately good predictor and can be used to understand trends as they relate to price but should not be used as an absolute predictor.

Question 3: Action steps a home builder can take on in order to best increase their home's value upon sale.

Choosing locations to Build Upon

Choose plots with a "Fair" view but no better, next to a greenbelt.

Home Construction

Choose between a "Good","Better" and "Very Good" conistruction grade.

Home Presentation and Sale

Each step up in condition will net about a $50,000 increase in value.

Aim to sell homes in the months of April or May to maximize the home sale price.

Database Understanding

A database of 30,155 home sales within King County, WA during the years 2021-2022 has been sourced from King County Assessor Data Download (https://info.kingcounty.gov/assessor/DataDownload/default.aspx) to build this predictive model, this data was choosen as it is the most recent and relevant data available within the stakeholder's area of interest.

The `address`, `lat`, and `long` fields have been retrieved using a third-party geocoding API (https://docs.mapbox.com/api/search/geocoding/). to build this predictive model, this data was choosen as it is the most recent and relevant data available within the stakeholder's county of interest.

There are 25 home features included in this data set, with "price" being treated as the target variable. The definition of each of these features can be found at below or at King County Glossary (https://info.kingcounty.gov/assessor/esales/Glossary.aspx?type=r).

Defining database columns names

- `id` - Unique identifier for a house
- `date` - Date house was sold
- `price` - Sale price (prediction target)
- `bedrooms` - Number of bedrooms
- `bathrooms` - Number of bathrooms
- `sqft_living` - Square footage of living space in the home
- `sqft_lot` - Square footage of the lot
- `floors` - Number of floors (levels) in house
- `waterfront` - Whether the house is on a waterfront
  - Includes Duwamish, Elliott Bay, Puget Sound, Lake Union, Ship Canal, Lake Washington, Lake Sammamish, other lake, and river/slough waterfronts
- `greenbelt` - Whether the house is adjacent to a green belt
- `nuisance` - Whether the house has traffic noise or other recorded nuisances
- `view` - Quality of view from house
  - Includes views of Mt. Rainier, Olympics, Cascades, Territorial, Seattle Skyline, Puget Sound, Lake Washington, Lake Sammamish, small lake / river / creek, and other
- `condition` - How good the overall condition of the house is. Related to maintenance of house.
  - See the King County Assessor Website (https://info.kingcounty.gov/assessor/esales/Glossary.aspx?type=r) for further explanation of each condition code
- `grade` - Overall grade of the house. Related to the construction and design of the house.
  - See the King County Assessor Website (https://info.kingcounty.gov/assessor/esales/Glossary.aspx?type=r) for further explanation of each building grade code
- `heat_source` - Heat source for the house
- `sewer_system` - Sewer system for the house
- `sqft_above` - Square footage of house apart from basement
- `sqft_basement` - Square footage of the basement
- `sqft_garage` - Square footage of garage space
- `sqft_patio` - Square footage of outdoor porch or deck space
- `yr_built` - Year when house was built
- `yr_renovated` - Year when house was renovated
- `address` - The street address
- `lat` - Latitude coordinate
- `long` - Longitude coordinate

Engineered Features

- `price_per_sqft` - The price per square foot of living space
- `has_patio` - A patio is present (sqft_patio > 0)
- `has_garage` - A garage is present (sqft_garage > 0)
- `month_sold` - the month of the year a house was sold.

Data Exploration and Inspection

In [1]:
```
#Importing utalized Python Packages

import pandas as pd #Allows access to csv data file as well as manipulati
import matplotlib.pyplot as plt #Creates Data visualizations
import matplotlib.cm as cm
import seaborn as sns
import numpy as np
import statsmodels.api as sm
from sklearn.preprocessing import OneHotEncoder #Used for categorical var
from statsmodels.stats.diagnostic import linear_rainbow, het_goldfeldquan
from statsmodels.stats.stattools import jarque_bera #Regression Assumptio
import scipy.stats as stats
%matplotlib inline
```

In [2]:
```
#Accessing CSV datafile stored in /data folder of repository and loading

kc_data = pd.read_csv( 'data/kc_house_data.csv')
kc_data.head()
```

Out[2]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfron
0	7399300360	5/24/2022	675000.0	4	1.0	1180	7140	1.0	NC
1	8910500230	12/13/2021	920000.0	5	2.5	2770	6703	1.0	NC
2	1180000275	9/29/2021	311000.0	6	2.0	2880	6156	1.0	NC
3	1604601802	12/14/2021	775000.0	3	3.0	2160	1400	2.0	NC
4	8562780790	8/24/2021	592500.0	2	2.0	1120	758	2.0	NC

5 rows × 25 columns

In [3]:
```
#Checking column datatypes
kc_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30155 entries, 0 to 30154
Data columns (total 25 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             30155 non-null  int64  
 1   date           30155 non-null  object 
 2   price          30155 non-null  float64
 3   bedrooms       30155 non-null  int64  
 4   bathrooms      30155 non-null  float64
 5   sqft_living    30155 non-null  int64  
 6   sqft_lot       30155 non-null  int64  
 7   floors         30155 non-null  float64
 8   waterfront     30155 non-null  object 
 9   greenbelt      30155 non-null  object 
 10  nuisance       30155 non-null  object 
 11  view           30155 non-null  object 
 12  condition      30155 non-null  object 
 13  grade          30155 non-null  object 
 14  heat_source    30123 non-null  object 
 15  sewer_system   30141 non-null  object 
 16  sqft_above     30155 non-null  int64  
 17  sqft_basement  30155 non-null  int64  
 18  sqft_garage    30155 non-null  int64  
 19  sqft_patio     30155 non-null  int64  
 20  yr_built       30155 non-null  int64  
 21  yr_renovated   30155 non-null  int64  
 22  address        30155 non-null  object 
 23  lat            30155 non-null  float64
 24  long           30155 non-null  float64
dtypes: float64(5), int64(10), object(10)
memory usage: 5.8+ MB
```

Most features are coded as integers, those coded as object will need to be converted to numeric values

In [4]:
```
#Exploring the values associated within various columns for understanding

#kc_data['bathrooms'].value_counts()
#kc_data['bedrooms'].value_counts()
#kc_data['waterfront'].value_counts()
#kc_data['greenbelt'].value_counts()
#kc_data['nuisance'].value_counts()
#kc_data['grade'].value_counts()
#kc_data['condition'].value_counts()
#kc_data['view'].value_counts()
#kc_data['heat_source'].value_counts()
#kc_data['sewer_system'].value_counts()
#kc_data['sqft_patio'].value_counts()
#kc_data['floors'].value_counts()
#kc_data['price'].sort_values()
```

Data Cleaning and Preparation

After gaining a rough understanding of the dataset in use, the first data cleaning step is to check the dataset for duplicate values and null values as well as reformating data values into numeric terms that can be loaded into a regression model.

In [5]:
```
#Checking for and then removing duplicate entries

#Checking for duplicate entries via id column
kc_dups = kc_data[kc_data['id'].isin(kc_data['id'][kc_data['id'].duplicat

#One duplicate was found and has been removed
kc_data.drop_duplicates(subset=['id'], inplace=True)

#viewing duplicat entries
kc_dups
```

Out[5]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterf
4845	1233100736	9/28/2021	2600000.0	3	4.0	3500	8455	2.0	
4846	1233100736	9/28/2021	2600000.0	3	4.0	3500	8455	2.0	

2 rows × 25 columns

Next each column is checked for the percentage of null values present

In [6]:
```
#Checking data for missing values
for col in kc_data.columns:
    percent_of_nulls = (sum(kc_data[col].isnull())/len(kc_data[col])) * 1
    print(col, percent_of_nulls)
```

```
id 0.0
date 0.0
price 0.0
bedrooms 0.0
bathrooms 0.0
sqft_living 0.0
sqft_lot 0.0
floors 0.0
waterfront 0.0
greenbelt 0.0
nuisance 0.0
view 0.0
condition 0.0
grade 0.0
heat_source 0.10612190754128806
sewer_system 0.04642833454931353
sqft_above 0.0
sqft_basement 0.0
sqft_garage 0.0
sqft_patio 0.0
yr_built 0.0
yr_renovated 0.0
address 0.0
lat 0.0
long 0.0
```

The heat_source column is missing 10.6% of its data points and the sewer_system is missing 4.6%, these are the only columns with null values.

These columns are not planned to be used in the data analysis and will likely be removed, if they are used, these null values will be dealt with properly, but for now these will be ignored

Removing Unneeded variables

Based on the question at hand it can be predetermined that some of the features of our dataset will not be utalized and therefore can be removed in order to simplify our dataframe.

We do not care about the location of these homes as the specified dataset already created our subset by location nor do we care about yr_built, yr_renovated as that will not within our stakeholders control. All of these columns will be dropped from from our working dataframe.

In [7]:
```
#Dropping features that will not be useful in this analysis

kc_data.drop(columns=['id','sqft_basement', 'address',
                      'yr_renovated','lat','long',
                      'sewer_system','heat_source', 'yr_built'], inplace='
```

Reformating existing data values

A handful of our features are stored as objects, to deal with these categorical variables, those with boolean values will be changed to 0s and 1, those that are not boolean will be converted to coded numeric values via One-Hot Encoding

In [8]:
```
#Exploring dataset column datatypes
kc_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30154 entries, 0 to 30154
Data columns (total 16 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   date         30154 non-null  object 
 1   price        30154 non-null  float64
 2   bedrooms     30154 non-null  int64  
 3   bathrooms    30154 non-null  float64
 4   sqft_living  30154 non-null  int64  
 5   sqft_lot     30154 non-null  int64  
 6   floors       30154 non-null  float64
 7   waterfront   30154 non-null  object 
 8   greenbelt    30154 non-null  object 
 9   nuisance     30154 non-null  object 
 10  view         30154 non-null  object 
 11  condition    30154 non-null  object 
 12  grade        30154 non-null  object 
 13  sqft_above   30154 non-null  int64  
 14  sqft_garage  30154 non-null  int64  
 15  sqft_patio   30154 non-null  int64  
dtypes: float64(3), int64(6), object(7)
memory usage: 3.9+ MB
```

All of the object datatypes will need to be changed to numeric values in order toe be used in our regression model down the line.

Grade

In [9]:
```
kc_data['grade'].value_counts()
```

Out[9]:
```
7 Average        11697
8 Good            9410
9 Better          3805
6 Low Average     2858
10 Very Good      1371
11 Excellent       406
5 Fair             393
12 Luxury          122
4 Low               51
13 Mansion          24
3 Poor              13
1 Cabin              2
2 Substandard        2
Name: grade, dtype: int64
```

Based on the information provided in the King County Glossary of Terms (https://info.kingcounty.gov/assessor/esales/Glossary.aspx?type=r), grades 1-4 are considered to not be up to code and therefore should not be an aim when building new homes, so these will be excluded.

Each grade already has a number value associated with it, these values will be kept and accompanying words will be removed

In [10]:
```
#Altering grade values to remove string and leave column as numeric value

kc_data['grade'] = kc_data['grade'].str.split(' ', 1, expand=True)
kc_data['grade'] = kc_data['grade'].astype(int)

#Removing homes graded 1-4
kc_data = kc_data[kc_data['grade'] > 4]
```

Condition

Conditio like grade, is coded with scaled words and will be converted into numeric values

In [11]:
```
# Replace coded words of condition variable to numeric values
kc_data['condition'].replace({'Poor': 1, 'Fair': 2, 'Average': 3, 'Good':
kc_data['condition'] = kc_data['condition'].astype(int)
```

Waterfront, Greenbelt and Nuisance

All three of these variables are boolean "YES" or "No" values, these will be converted to "1" and "0", respectively.

In [12]:
```
#Converting waterfront, greenbelt and nuisance features from 'NO' or 'YES

for col in ['waterfront', 'greenbelt', 'nuisance']:
    kc_data[col].replace({'NO': 0, 'YES': 1}, inplace=True)
```

In [13]:
```
kc_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30086 entries, 0 to 30154
Data columns (total 16 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   date         30086 non-null  object 
 1   price        30086 non-null  float64
 2   bedrooms     30086 non-null  int64  
 3   bathrooms    30086 non-null  float64
 4   sqft_living  30086 non-null  int64  
 5   sqft_lot     30086 non-null  int64  
 6   floors       30086 non-null  float64
 7   waterfront   30086 non-null  int64  
 8   greenbelt    30086 non-null  int64  
 9   nuisance     30086 non-null  int64  
 10  view         30086 non-null  object 
 11  condition    30086 non-null  int64  
 12  grade        30086 non-null  int64  
 13  sqft_above   30086 non-null  int64  
```

All remaining features have been converted into numerical values except the view and date features. View has multiple categorical values possible, this variable will be one hot encoded and date will be used to produce a month column and then removed.

Engineering Data Features

A number of features will be engineered from the existing features and then tested for their correlation with price in order to obtain the best predictor features possible for our model.

Month House was Sold

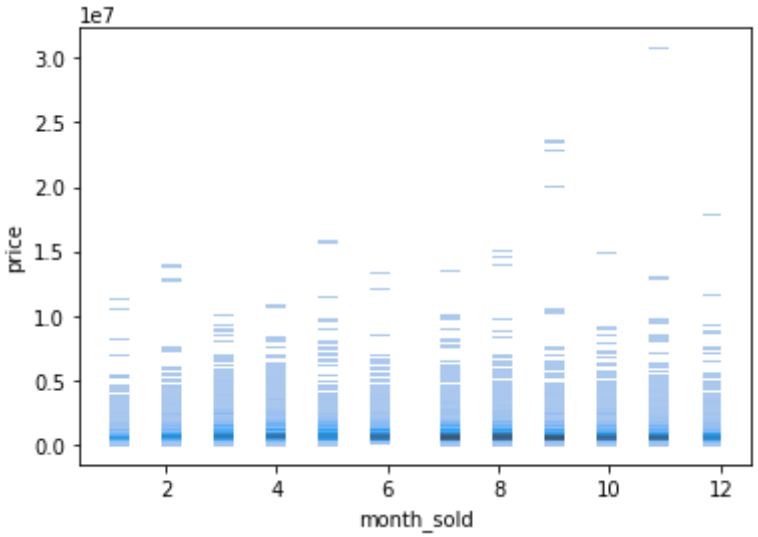
In [14]:
```
#Creating Month_sold from date sold
kc_data['date']
```

Out[14]:
```
0          5/24/2022
1         12/13/2021
2          9/29/2021
3         12/14/2021
4          8/24/2021
              ...
30150     11/30/2021
30151      6/16/2021
30152      5/27/2022
30153      2/24/2022
30154      4/29/2022
Name: date, Length: 30086, dtype: object
```

In [15]:
```
#Creating Month_sold from date sold and dropping date column

kc_data['date'] = kc_data['date'].str.split('/', 1, expand=True)
kc_data['month_sold'] = kc_data['date'].astype(int)

kc_data.drop(columns='date', inplace=True)
```

In [16]:
```
#Visualizing coorelations between month sold and price
sns.histplot(data=kc_data, x='month_sold', y='price')
```

Out[16]:
```
<AxesSubplot:xlabel='month_sold', ylabel='price'>
```



Looks like there may be a coorelation between month sold and price. Jan has the lowest average selling price, this will act as the reference variable for one hot encoding the months for the regression.

Has a Patio and Has a Garage features

While there are already sqft metrics for patios and garages in this dataset, the mere presence of one of these features may be a good predictor of price.

In [17]:
```
#Creating Columns for the presence of either a patio or a garage.

def add_has_garage_col(row):
        if row['sqft_garage'] > 0:
            return 1
        else:
            return 0

kc_data['has_garage'] = kc_data.apply(lambda row: add_has_garage_col(row)

#Adding has patio boolean column

def add_has_patio_col(row):
        if row['sqft_patio'] > 0:
            return 1
        else:
            return 0

kc_data['has_patio'] = kc_data.apply(lambda row: add_has_patio_col(row),
```

Price per sqft

The value of homes is often displayed by price per sqft, so this may be a useful metric

In [18]:
```
#Creating a Price per sqft feature based on the square footage of living

kc_data['price_per_sqft'] = kc_data['price'] / kc_data['sqft_living']
kc_data['price_per_sqft'].describe()
```

Out[18]:
```
count     30086.000000
mean        559.968986
std        3489.562236
min           6.920415
25%         360.615933
50%         491.755410
75%         643.776310
max      601000.000000
Name: price_per_sqft, dtype: float64
```

Looks like there is a potential that the price per sqft metric has some outliers when the 75th percentile is $644 dollars per sqft and the max value is $601,000 dollars per sqft

In [19]:
```
#Checking the top and bottom values of the price per sqft feature for pos
kc_data['price_per_sqft'].sort_values()
```

Out[19]:
```
6430           6.920415
25830          7.117730
9125          11.695703
21793         11.880603
4487          13.895640
                ...
6125       10000.000000
23470      15073.529412
9516       15584.158416
5811       48103.448276
14977     601000.000000
Name: price_per_sqft, Length: 30086, dtype: float64
```

Upon a closer look at the engineered price_per_sqft feature it appears very likely that there are significant outliers within this metric, the outliers in this feature as well as other features will be dealt with in prior to our regression to create a more general training set.

Removing Potential Outliers in Numeric variables

In [20]:
```
#Looking at overall dataset for possible outliers
kc_data.describe()
```

Out[20]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	w
count	3.008600e+04	30086.000000	30086.000000	30086.000000	3.008600e+04	30086.000000	3008
mean	1.109598e+06	3.418101	2.337848	2115.704913	1.657765e+04	1.544589	
std	8.965545e+05	0.977318	0.887801	972.501664	5.889275e+04	0.567818	
min	2.736000e+04	0.000000	0.000000	3.000000	4.020000e+02	1.000000	
25%	6.499500e+05	3.000000	2.000000	1420.000000	4.840250e+03	1.000000	
50%	8.600000e+05	3.000000	2.500000	1920.000000	7.474500e+03	1.500000	
75%	1.300000e+06	4.000000	3.000000	2620.000000	1.055750e+04	2.000000	
max	3.075000e+07	13.000000	10.500000	15360.000000	3.253932e+06	4.000000	

Looking at the description stats for this database a number of numeric values have a large descrepancy between the 75% percentile value and the max value, such as price and sqft_living. We will calculate acceptable outlier values from these metrics by calculating the interquartile range (IQR). Once the interquartile range is found, the outlier cutoffs will be calculated via the 25th and 75th percentile values using the following formulas:

> Lower Limit: q25 - 1.5 * IQR
> Upper Limit: q75 + 1.5 * IQR

In [21]:
```
#creating a list of variables to check for outliers
kc_outliers = ['price', 'bedrooms','bathrooms', 'sqft_living', 'sqft_lot'
                        'price_per_sqft','sqft_garage',
                       'sqft_above']
```

In [22]:
```
#Calculating outlier cutoff values using interquartile range

upper_dic={}
for column in kc_outliers:
    q100,q75, q25 = np.percentile(kc_data[column],[100,75, 25])
    iqr = q75-q25
    iqr

    lower_limit = q25 - 1.5 * iqr
    upper_limit = q75 + 1.5 * iqr

    #Storing each varuables 75th perctile as its own variable in a list
    upper_dic[str(column)] = q75 + 1.5 * iqr

    print(('The 75th percentile of', column,'is', q75, 'max value is',q10
    print('The', column,'outlier lower limit is', lower_limit)
    print('The', column,'outlier upper limit is', upper_limit)
    print('The number of homes that are above the 75th percentile are', s
    print()
```

```
('The 75th percentile of', 'price', 'is', 1300000.0, 'max value is', 30
750000.0)
The price outlier lower limit is -325125.0
The price outlier upper limit is 2275075.0
The number of homes that are above the 75th percentile are 1993

('The 75th percentile of', 'bedrooms', 'is', 4.0, 'max value is', 13.0)
The bedrooms outlier lower limit is 1.5
The bedrooms outlier upper limit is 5.5
The number of homes that are above the 75th percentile are 635

('The 75th percentile of', 'bathrooms', 'is', 3.0, 'max value is', 10.
5)
The bathrooms outlier lower limit is 0.5
The bathrooms outlier upper limit is 4.5
The number of homes that are above the 75th percentile are 883

('The 75th percentile of', 'sqft_living', 'is', 2620.0, 'max value is',
15360.0)
The sqft_living outlier lower limit is -380.0
The sqft_living outlier upper limit is 4420.0
The number of homes that are above the 75th percentile are 734

('The 75th percentile of', 'sqft_lot', 'is', 10557.5, 'max value is', 3
253932.0)
The sqft_lot outlier lower limit is -3735.625
The sqft_lot outlier upper limit is 19133.375
The number of homes that are above the 75th percentile are 3534

('The 75th percentile of', 'price_per_sqft', 'is', 643.7763096864776,
'max value is', 601000.0)
The price_per_sqft outlier lower limit is -64.12463245369713
The price_per_sqft outlier upper limit is 1068.5168749705824
The number of homes that are above the 75th percentile are 881

('The 75th percentile of', 'sqft_garage', 'is', 510.0, 'max value is',
3580.0)
The sqft_garage outlier lower limit is -765.0
The sqft_garage outlier upper limit is 1275.0
The number of homes that are above the 75th percentile are 114

('The 75th percentile of', 'sqft_above', 'is', 2270.0, 'max value is',
12660.0)
The sqft_above outlier lower limit is -455.0
The sqft_above outlier upper limit is 3905.0
The number of homes that are above the 75th percentile are 803
```

In [23]:
```
#Checking created refernce dictionary for upper outlier cut-offs

upper_keys  = list(upper_dic.keys())
upper_dic
```

Out[23]:
```
{'price': 2275075.0,
 'bedrooms': 5.5,
 'bathrooms': 4.5,
 'sqft_living': 4420.0,
 'sqft_lot': 19133.375,
 'price_per_sqft': 1068.5168749705824,
 'sqft_garage': 1275.0,
 'sqft_above': 3905.0}
```

In [24]:
```
#Applying outlier cut offs to each applicable column to create a more gen

data_no_outliers = kc_data.copy()

#Applying upper outlier cutoffs
data_no_outliers = data_no_outliers[data_no_outliers['price'] <= 2275075]
data_no_outliers = data_no_outliers[data_no_outliers['price_per_sqft'] <=
data_no_outliers = data_no_outliers[data_no_outliers['price_per_sqft'] >=
data_no_outliers = data_no_outliers[data_no_outliers['sqft_living'] <= 44

#Apply cutoffs to keep data relevant to stakehholder's goal

data_no_outliers = data_no_outliers[data_no_outliers['bedrooms'] >= 1 ] #
data_no_outliers = data_no_outliers[data_no_outliers['bathrooms'] >= 1 ]#

data_no_outliers.describe()
```

Out[24]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	w
count	2.351200e+04	23512.000000	23512.000000	23512.000000	2.351200e+04	23512.000000	2351
mean	8.892604e+05	3.444964	2.307396	2053.849481	1.525571e+04	1.547380	
std	3.945986e+05	0.916737	0.763785	757.365180	4.738053e+04	0.567551	
min	8.249400e+04	1.000000	1.000000	400.000000	4.120000e+02	1.000000	
25%	6.100000e+05	3.000000	2.000000	1460.000000	4.800000e+03	1.000000	
50%	7.950000e+05	3.000000	2.500000	1930.000000	7.390000e+03	1.500000	
75%	1.075000e+06	4.000000	2.500000	2540.000000	1.023125e+04	2.000000	
max	2.275000e+06	11.000000	7.500000	4420.000000	2.657160e+06	4.000000	

In [25]:
```
#how many rows were lost by removing potential outliers
len(kc_data) - len(data_no_outliers)
```

Out[25]:
```
6574
```

In [26]:
```
#Resetting index after rows were dropped, prior to splitting df to deal w
data_no_outliers.reset_index(drop=True, inplace=True)
data_no_outliers
```

Out[26]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	greenbelt	nuisan
0	675000.0	4	1.0	1180	7140	1.0	0	0	
1	920000.0	5	2.5	2770	6703	1.0	0	0	
2	311000.0	6	2.0	2880	6156	1.0	0	0	
3	775000.0	3	3.0	2160	1400	2.0	0	0	
4	592500.0	2	2.0	1120	758	2.0	0	0	
...	...	...	...	...	...	...	...	...	
23507	719000.0	3	2.5	1270	1141	2.0	0	0	
23508	1313000.0	3	2.0	2020	5800	2.0	0	0	
23509	800000.0	3	2.0	1620	3600	1.0	0	0	
23510	775000.0	3	2.5	2570	2889	2.0	0	0	
23511	500000.0	3	1.5	1200	11058	1.0	0	0	

23512 rows × 19 columns

Pulling out Numeric Variables

Here we are pulling out the numeric variables to later be recombined with the categorical variables after they are one hot encoded

In [27]:
```
#Getting the names of all number type columns
num_cols = data_no_outliers.select_dtypes(include=np.number).columns

#creating a numeric variable only df
kc_nums = data_no_outliers[num_cols].copy()
```

One Hot Encoding Categorical Variables

The remaining non-numeric categorical variables will be One Hot Encoded to create a column for each conditional value. To avoid colinearity issues one condition value column for each variable will be dropped and this will act the reference values when these variables are analyzed via linear regression

In [28]:
```
#One Hot Encoding all Categorical Variables

ohe = OneHotEncoder(sparse=False)

#Selecting Categorical Variables for analysis
kc_cats = data_no_outliers[['view','condition', 'month_sold','grade']].co

#Completeing OneHotEncoding in order to use categorical variables in regr
kc_cats_ohe = pd.DataFrame(
    data=ohe.fit_transform(kc_cats),
    index = data_no_outliers.index,
    columns=ohe.get_feature_names([ 'view','condition','month_sold','grad

#Dropping columns that will be used as reference columns for each OneHotE
kc_cats_ohe.drop(['view_NONE', 'condition_1', 'month_sold_1','grade_5'],
```

One Hot Encoded Dropped Columns that will act as references for regression:

view : NONE
condition: Poor
month_sold: January
grade: Fair

Recombining Numeric and One Hot Encoded categorical variables

In [29]:
```
#Concating One Hot encoded variables with Numeric Variables to create a w

kc_working_data = pd.concat([kc_nums, kc_cats_ohe], axis=1)

#Dropping origin columns for the three variables that were one hot encode
kc_working_data.drop(['condition','month_sold','grade'], axis=1, inplace=
```

In [30]:
```
#Checking to make sure concatanation was aligned properly
nan_count = kc_working_data.isna().sum()
print(nan_count)
```

```
price              0
bedrooms           0
bathrooms          0
sqft_living        0
sqft_lot           0
floors             0
waterfront         0
greenbelt          0
nuisance           0
sqft_above         0
sqft_garage        0
sqft_patio         0
has_garage         0
has_patio          0
price_per_sqft     0
view_AVERAGE       0
view_EXCELLENT     0
view_FAIR          0
view_GOOD          0
condition_2        0
condition_3        0
condition_4        0
condition_5        0
month_sold_2       0
month_sold_3       0
month_sold_4       0
month_sold_5       0
month_sold_6       0
month_sold_7       0
month_sold_8       0
month_sold_9       0
month_sold_10      0
month_sold_11      0
month_sold_12      0
grade_6            0
grade_7            0
grade_8            0
grade_9            0
grade_10           0
grade_11           0
grade_12           0
dtype: int64
```

Data Analysis

Now that the data has been cleaned and processed based on the parameters of our root questions and stakeholder relevance, we can begin analyzing the data to answer our business questions.

Question 1: Identifying Factors most Highly Correlated to Home Price

In [31]:
```
#Calculating correlation values for each feature to price

abs(kc_working_data.corr()["price"]).sort_values(ascending=False)
```

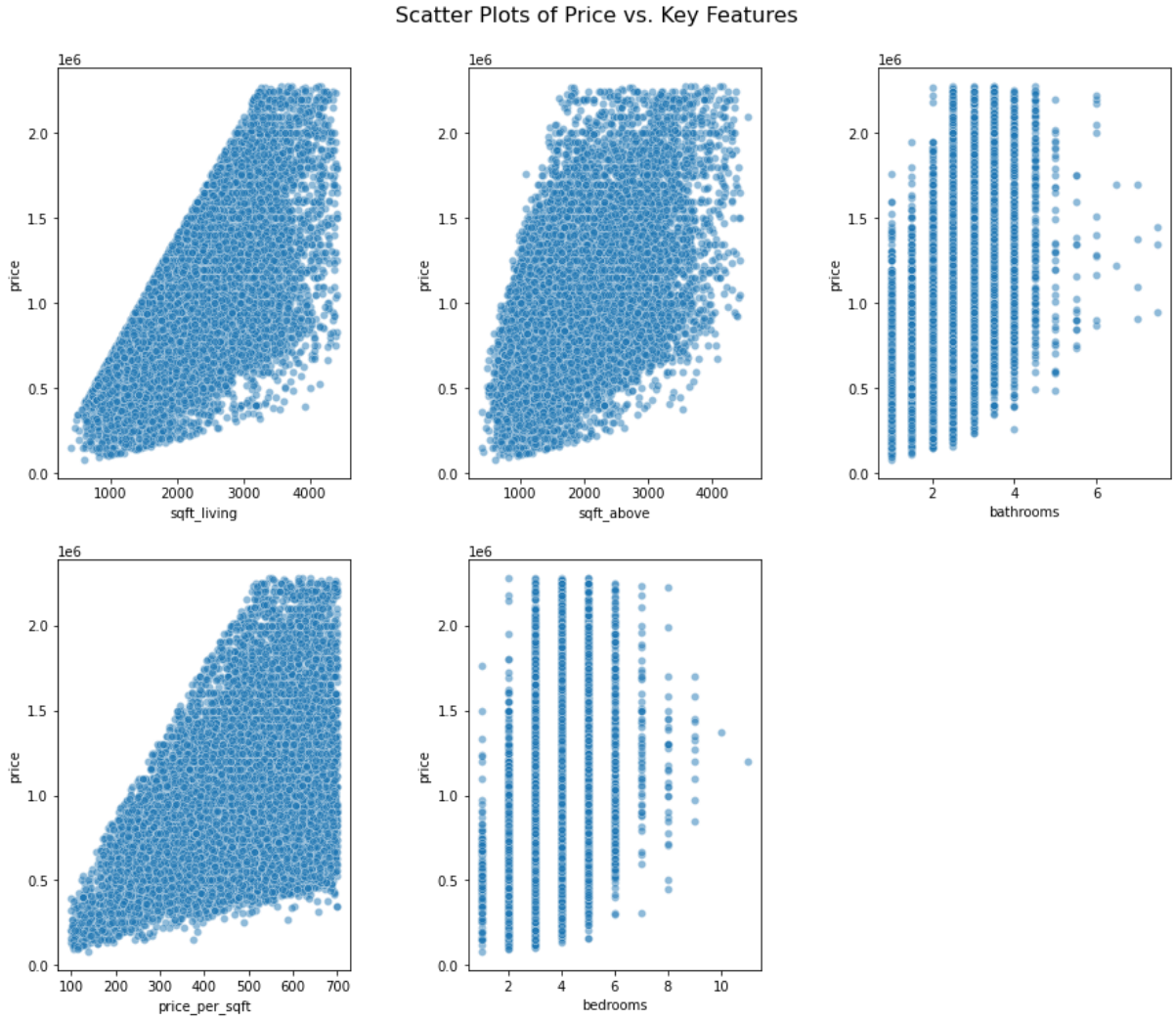
Out[31]:
```
price              1.000000
sqft_living        0.691632
sqft_above         0.567499
bathrooms          0.516993
price_per_sqft     0.498227
bedrooms           0.381597
grade_9            0.367573
grade_7            0.304110
grade_10           0.287921
sqft_garage        0.273390
grade_6            0.272866
sqft_patio         0.261282
floors             0.244365
has_patio          0.203836
has_garage         0.156665
grade_8            0.148698
grade_11           0.138032
greenbelt          0.133311
view_AVERAGE       0.121436
view_GOOD          0.110335
sqft_lot           0.086214
view_EXCELLENT     0.071682
month_sold_5       0.054487
condition_2        0.053178
view_FAIR          0.047599
month_sold_4       0.037749
condition_3        0.035861
condition_4        0.035151
grade_12           0.029965
waterfront         0.027106
month_sold_9       0.025403
month_sold_12      0.025028
month_sold_11      0.023413
month_sold_6       0.018315
month_sold_2       0.017817
month_sold_10      0.017785
nuisance           0.014810
month_sold_7       0.013496
condition_5        0.012468
month_sold_3       0.011971
month_sold_8       0.000016
Name: price, dtype: float64
```

The features with the highest correlation values will be considered our key features for this analysis, these features will be integral to building our multilinear regression model for price predition. The key features are below.

```
Features with highest Correlations to Price:
    1. sqft_living
    2. sqft_above
    3. bathrooms
    4. price_per_sqft
    5. bedrooms
```

Checking Key Features for relationships with Price

In [32]:
```
#Creating scatter plots for each key feature vs price

cols = ['sqft_living', 'sqft_above',  'bathrooms', 'price_per_sqft', 'bed

fig,ax = plt.subplots(figsize=(15,13), ncols=3, nrows=2)
fig.subplots_adjust(top=0.85)
fig.suptitle('Scatter Plots of Price vs. Key Features', y=0.9, fontsize=1
plt.subplots_adjust(wspace = 0.4, hspace = 0.2)

for i,c in enumerate(cols):
    sns.scatterplot(x=c, y='price', ci=None, data=kc_working_data, ax=ax[

fig.delaxes(ax[1][2])
```



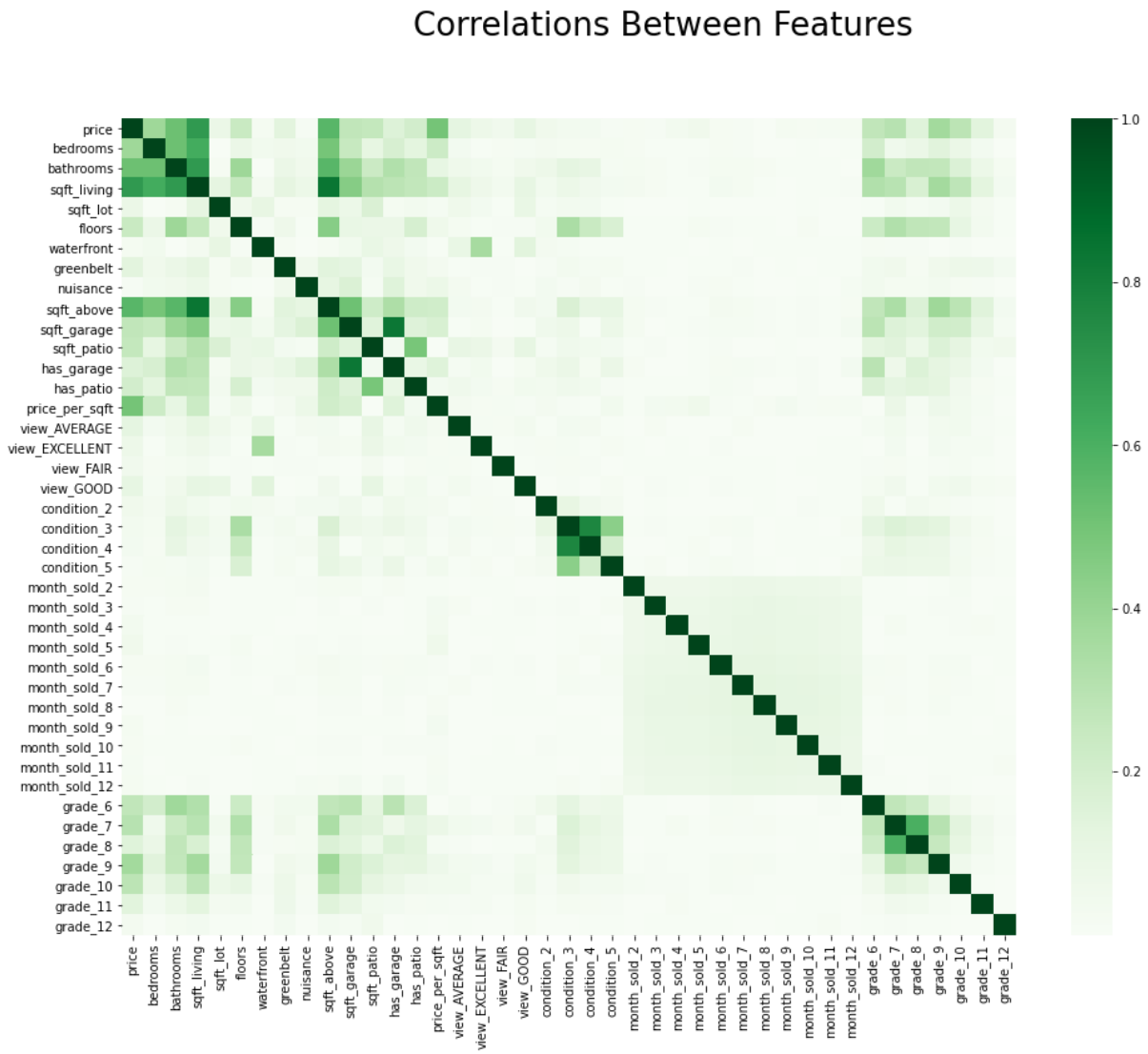
All of these features seem to have linear or somewhat linear relationships with price.

Identifying Multicollinearity Between Features

In order to satisfy the colinear assumption of linear regressions we must eliminate any highly colinear features. These will be identified and then one feature from each colinear pair will be removed prior the building the regression

In [33]:
```
#Creating heatmap of correlation values between features
fig, ax = plt.subplots(figsize=(16,12))
fig.suptitle('Correlations Between Features', fontsize=26)
sns.heatmap(abs(kc_working_data.corr().abs()), cmap='Greens')
```

Out[33]: <AxesSubplot:>



In [34]:
```
#Identifying factor pairs with high coliearity in order to remove one fro

corr_df = kc_working_data.corr().abs().stack().reset_index().sort_values(

corr_df['pairs'] = list(zip(corr_df.level_0, corr_df.level_1))

corr_df.set_index(['pairs'], inplace=True)

corr_df.drop(columns=['level_1', 'level_0'], inplace = True)

corr_df.columns = ['coor']

corr_df.drop_duplicates(inplace=True)

corr_df[(corr_df.coor>.75) & (corr_df.coor<1)]
```

Out[34]:

pairs	coor
(sqft_above, sqft_living)	0.837997
(has_garage, sqft_garage)	0.831986
(condition_4, condition_3)	0.772740

The Three pairs listed above are features that have correlation values higher than 75%. One feature from each pair must be dropped prior to our regression in order to avoid multicollinearity between features. The feature with the lower correlation to price in each pair will be dropped to assure no colinearity is present.

sqft_above and has_garage will be dropped as these have the lower coorlation with price when compared to the feature they were found to be colinear with above.

In [35]:
```
#Dropping one feature from colinear pairs
#kc_working_data.drop(columns=['sqft_above','has_garage'], inplace=True)

#Creating New dataframe for regressions
kc_regression_data = kc_working_data.copy()
kc_regression_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23512 entries, 0 to 23511
Data columns (total 41 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   price           23512 non-null  float64
 1   bedrooms        23512 non-null  int64  
 2   bathrooms       23512 non-null  float64
 3   sqft_living     23512 non-null  int64  
 4   sqft_lot        23512 non-null  int64  
 5   floors          23512 non-null  float64
 6   waterfront      23512 non-null  int64  
 7   greenbelt       23512 non-null  int64  
 8   nuisance        23512 non-null  int64  
 9   sqft_above      23512 non-null  int64  
 10  sqft_garage     23512 non-null  int64  
 11  sqft_patio      23512 non-null  int64  
 12  has_garage      23512 non-null  int64  
 13  has_patio       23512 non-null  int64  
 14  price_per_sqft  23512 non-null  float64
 15  view_AVERAGE    23512 non-null  float64
 16  view_EXCELLENT  23512 non-null  float64
 17  view_FAIR       23512 non-null  float64
 18  view_GOOD       23512 non-null  float64
 19  condition_2     23512 non-null  float64
 20  condition_3     23512 non-null  float64
 21  condition_4     23512 non-null  float64
 22  condition_5     23512 non-null  float64
 23  month_sold_2    23512 non-null  float64
 24  month_sold_3    23512 non-null  float64
 25  month_sold_4    23512 non-null  float64
 26  month_sold_5    23512 non-null  float64
 27  month_sold_6    23512 non-null  float64
 28  month_sold_7    23512 non-null  float64
 29  month_sold_8    23512 non-null  float64
 30  month_sold_9    23512 non-null  float64
 31  month_sold_10   23512 non-null  float64
 32  month_sold_11   23512 non-null  float64
 33  month_sold_12   23512 non-null  float64
 34  grade_6         23512 non-null  float64
 35  grade_7         23512 non-null  float64
 36  grade_8         23512 non-null  float64
 37  grade_9         23512 non-null  float64
 38  grade_10        23512 non-null  float64
 39  grade_11        23512 non-null  float64
 40  grade_12        23512 non-null  float64
dtypes: float64(30), int64(11)
memory usage: 7.4 MB
```

Building the Multilinear Regression Prediction Model

Now that we have our predictive features choosen and our data in order, the next step will be to run a simple baseline linear regression for our target feature, which in this case is Price. From there more features will be added into a multilinear regression, these features will be tested to see how they effect our model and will be added or removed based on their overall usefulness.

In order to run a proper linear regression, there are four assumptions about the data that must be taken into account and a violation of any of these 4 assumptions will cause the model to be inaccurate. Below are these assumptions and how they will be dealt with in this data analysis.

The 4 assumptions needed for a Linear Regression models are:

1. There must be a linear relationship between the target variable (price) and the predictor variable

   > This will be check visually via a scatter plotting the model residuals and to assure this assumption is met statistically, a Rainbow test will also be utalized

2. There must be independence of both the features and the errors in a regression

   > Feature Independence was checked earlier via the "Identifying Multicollinearity Between Features" section above and has already been accounted for within our data

3. The Residuals of the linear regression should follow a normal distribution

   > This will be checked visually via a Q-Q plot and histogram of the residuals as well as statistically via a Jarque-Bera Test

4. There must be homoscedasticity of errors

   > This will be checked statistically via a Goldfield-Quandt Test

Thankfully we have fairly straightforward ways of testing these assumptions for each model. Below a function has been built to run all fo the of the assumption tests described above so that a model can be easily tested.

Building Function to quickly test multilinear regression assumptions

In [36]:
```
'''
Tests an OLS regression model for the linearity, normality and homoscedas

The linearity is tested visually via a scatter plot of the model's residu
a rainbow test using the linear_rainbow module.

The normalilty is tested visually via a Q-Q plot of the model's residual
residuals. Normalilty is also tested statisticall via a Jarque-Bera Test

The homoscedasticity is tested via a Goldfeld-Quandt Test
'''

def model_assump_test(model_results, X, X_name, y):

    fig, ax = plt.subplots(ncols=3, figsize=(40,15))
    fig.suptitle('Linear Regression Assumption Testing Results for {x}'.f

#Linearity Testing

    #Linearity Testing via Rainbow Test
    if linear_rainbow(model_results)[1] > 0.05:
        fig.text(s=u'\u2714 Linearity Assumption passed', y=1.02, x=.4, f
    else:
        fig.text(s=u'\u2718 Linearity Assumption NOT passed', y=1.02, x=.

    #Linearity Plot
    ax[0].scatter(x=y, y=model_results.resid)
    ax[0].axhline(y=0, color="black")
    ax[0].set_xlabel("Price", fontsize=24)
    ax[0].set_ylabel("Residuals", fontsize=24)
    ax[0].set_title("Residual Linearity Plot", fontsize=30);

#Normality Testing]

    #Normality of Residuals test via Q-Q plot and Jarque-Bera Test
    if jarque_bera(model_results.resid)[1] > 0.05:
        fig.text(s=u'\u2714 Normality Assumption passed', y=.99, x=.4, fo
    else:
        fig.text(s=u'\u2718 Normality Assumption NOT passed', y=.99, x=.4

    #Q-Q Plot generation
    sm.graphics.qqplot(model_results.resid, dist=stats.norm, line='45', f
    ax[1].set_title('Q-Q Plot of Normality', fontsize=30)
    ax[1].set_xlabel("Theoretical Quantiles",fontsize=24)
    ax[1].set_ylabel("Sample Quantiles", fontsize=24)

    #Histogram of model residuals
    ax[2].set_title('Distribution of Residuals for Normality', fontsize=3
    ax[2].set_xlabel('Model Residuals', fontsize=24)
    ax[2].set_ylabel('Count', fontsize=24)
    sns.histplot(model_results.resid, ax=ax[2], kde=True)

#Homoscedasticity testing

    #Goldfeld-Quandt Test of Homoscedasticity
    if het_goldfeldquandt(y, X, alternative='two-sided')[1] > 0.05:
        fig.text(s=u'\u2714 Homoscedasticity Assumption passed', y=.96, x
```

```
    else:
        fig.text(s=u'\u2718 Homoscedasticity Assumption NOT passed', y=.9
```

Baseling Model : Simple Linear Regression with highest correlated feature

In [37]:
```
#Creating a simple linear regression of sqft_living vs price
y = kc_regression_data['price'] #Target Feature
X_baseline = kc_regression_data[['sqft_living']] #Predictor features

baseline_model = sm.OLS(y, sm.add_constant(X_baseline)) #Running OLS regr
baseline_results = baseline_model.fit()
baseline_results.summary() #Printing out summary of model results
```

Out[37]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.478
Model:	OLS	**Adj. R-squared:**	0.478
Method:	Least Squares	**F-statistic:**	2.156e+04
Date:	Fri, 10 Mar 2023	**Prob (F-statistic):**	0.00
Time:	11:02:21	**Log-Likelihood:**	-3.2868e+05
No. Observations:	23512	**AIC:**	6.574e+05
Df Residuals:	23510	**BIC:**	6.574e+05
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.492e+05	5372.352	27.763	0.000	1.39e+05	1.6e+05
sqft_living	360.3504	2.454	146.829	0.000	355.540	365.161

Omnibus:	175.692	**Durbin-Watson:**	2.021
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	186.372
Skew:	0.189	**Prob(JB):**	3.39e-41
Kurtosis:	3.219	**Cond. No.**	6.33e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6.33e+03. This might indicate that there are strong multicollinearity or other numerical problems.

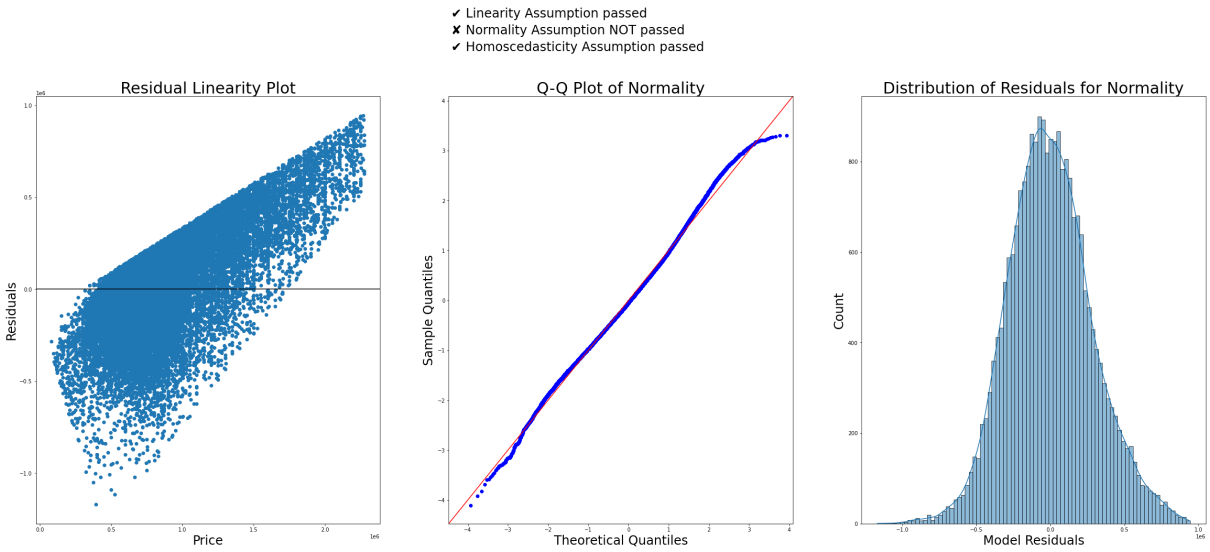
Baseline OLS Linear Regression Model Results Interpretation

- The model is statistically significant overall, with an F-statistic p-value well below 0.05
- The R-squared is .478 which means the model explains about 48% of the variance in price
- The model coefficients (`const` and `sqft_living`) are both statistically significant, with t-statistic p-values below 0.05
- If a home had 0 sqft of living space, we would expect the sale price to be about 149,200
- For each increase of 1 sqft of living space, we expect to see an increase in sale price of about $360

This is a fairly good starting point, from here additional features will be added in order to improve the model

In [38]:
```
#Testing sqft_living Model for Linear Regression Assumptions

model_assump_test(baseline_results, X_baseline, 'Baseline Model', y)
```

Linear Regression Assumption Testing Results for Baseline Model



Baseline model Assumption tests interpretation

Our baseline regression passes the linearity and homoscedasticity assumptions and is fairly normal but not completely normal as there is a slight skew in the distribution of the residuals. This seems like a good starting place and the model will be iterated further from here.

Second Model : Adding Bathrooms Feature

Bathrooms was the feature that had the second highest correlation to price.

In [39]:
```
#Adding Bathroom feature to baseline model

y = kc_regression_data['price']
X_second_model = kc_regression_data[['sqft_living','bathrooms']]

second_model = sm.OLS(y, sm.add_constant(X_second_model))
second_results = second_model.fit()
second_results.summary()
```

Out[39]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.481
Model:	OLS	**Adj. R-squared:**	0.481
Method:	Least Squares	**F-statistic:**	1.089e+04
Date:	Fri, 10 Mar 2023	**Prob (F-statistic):**	0.00
Time:	11:02:22	**Log-Likelihood:**	-3.2862e+05
No. Observations:	23512	**AIC:**	6.572e+05
Df Residuals:	23509	**BIC:**	6.573e+05
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.174e+05	6108.592	19.212	0.000	1.05e+05	1.29e+05
sqft_living	334.7203	3.402	98.375	0.000	328.051	341.389
bathrooms	3.66e+04	3373.896	10.847	0.000	3e+04	4.32e+04

Omnibus:	198.038	**Durbin-Watson:**	2.021
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	211.188
Skew:	0.200	**Prob(JB):**	1.38e-46
Kurtosis:	3.235	**Cond. No.**	7.53e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 7.53e+03. This might indicate that there are strong multicollinearity or other numerical problems.

Interpretation of Second Model

Adding the Bathrooms feature seems to have increased our R-squared value while remaining significant, this is a step in the right direction.

Third Model: Adding Price per Sqft

Adding Third Highest correlated feature

In [40]:
```
#Creating a simple linear regression of sqft_living vs price

y = kc_regression_data['price']
X_third_model = kc_regression_data[['sqft_living', 'bathrooms', 'price_pe

third_model = sm.OLS(y, sm.add_constant(X_third_model))
third_results = third_model.fit()
third_results.summary()
```

Out[40]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.937
Model:	OLS	**Adj. R-squared:**	0.937
Method:	Least Squares	**F-statistic:**	1.171e+05
Date:	Fri, 10 Mar 2023	**Prob (F-statistic):**	0.00
Time:	11:02:22	**Log-Likelihood:**	-3.0377e+05
No. Observations:	23512	**AIC:**	6.076e+05
Df Residuals:	23508	**BIC:**	6.076e+05
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-9.491e+05	3340.294	-284.143	0.000	-9.56e+05	-9.43e+05
sqft_living	443.4742	1.212	366.027	0.000	441.099	445.849
bathrooms	1658.7030	1175.807	1.411	0.158	-645.956	3963.361
price_per_sqft	2079.0839	5.027	413.595	0.000	2069.231	2088.937

Omnibus:	2868.267	**Durbin-Watson:**	1.995
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	9531.727
Skew:	-0.620	**Prob(JB):**	0.00
Kurtosis:	5.862	**Cond. No.**	1.16e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.16e+04. This might indicate that there are strong multicollinearity or other numerical problems.

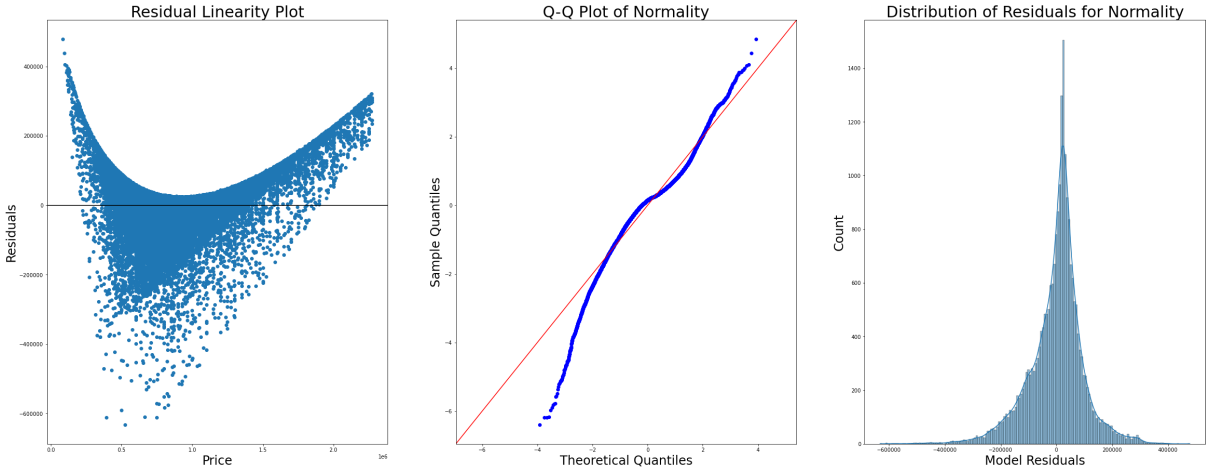
Interpretation of Third Model

Adding in the price per sqft feature has drastically pushed our R-squared value up to .937, I am fairly suspicious of this. It is very likely that this feature cannot be used in these regressions as this feature was built from the price and living_sqft features which are also present in this model.

In [41]:
```
model_assump_test(third_results, X_third_model, 'Third Model',y)
```

Linear Regression Assumption Testing Results for Third Model

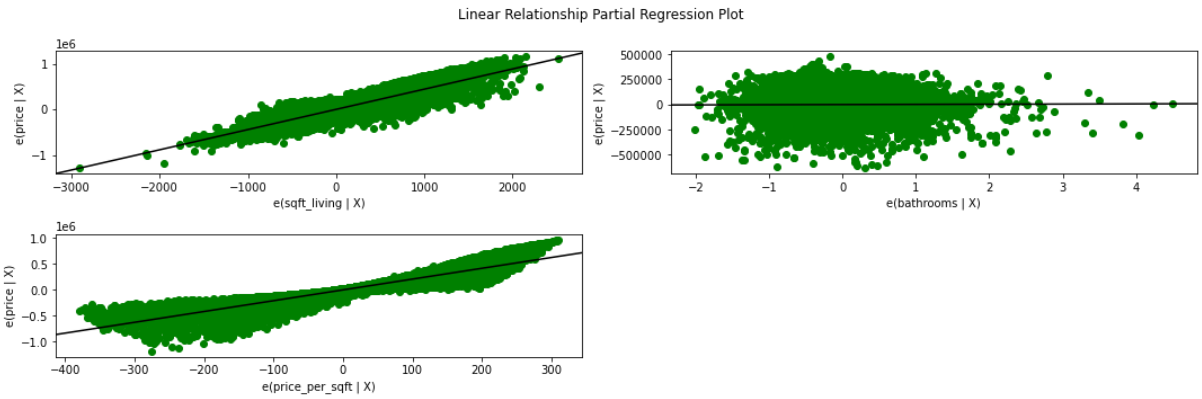
✔ Linearity Assumption passed
✘ Normality Assumption NOT passed
✔ Homoscedasticity Assumption passed



Interpretation of Third Model Assumptions

Based on the curve of the linearity graph on the left and the loss in normality, I am going to choose to exclude the price per sqft metric as it is likely it will only cause issues as it was derived from other variables present in the model

In [42]:
```
#Checking linearity of residuals of features indivdually

fig = plt.figure(figsize=(15,5))
sm.graphics.plot_partregress_grid(third_results, exog_idx=['sqft_living',

# Customizing plot appearance; note that the StatsModels code actually us
# with marker 'o', so what looks like a scatter plot is a "line" internal
# we access it using .lines rather than .collections
for ax in fig.axes:
    ax.lines[0].set_color("green")
fig.suptitle("Linear Relationship Partial Regression Plot", fontsize="lar

plt.tight_layout()
plt.show()
```



Fourth Model: Removed Price Per Sqft, added Bedrooms

Bedrooms was the forth highest correlated feature

In [43]:
```
#Creating a simple linear regression of sqft_living vs price

y = kc_regression_data['price']
X_forth_model = kc_regression_data[['sqft_living', 'bedrooms', 'bathrooms

forth_model = sm.OLS(y, sm.add_constant(X_forth_model))
forth_results = forth_model.fit()
forth_results.summary()
```

Out[43]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.486
Model:	OLS	**Adj. R-squared:**	0.486
Method:	Least Squares	**F-statistic:**	7401.
Date:	Fri, 10 Mar 2023	**Prob (F-statistic):**	0.00
Time:	11:02:23	**Log-Likelihood:**	-3.2851e+05
No. Observations:	23512	**AIC:**	6.570e+05
Df Residuals:	23508	**BIC:**	6.571e+05
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.838e+05	7560.619	24.313	0.000	1.69e+05	1.99e+05
sqft_living	358.2407	3.742	95.747	0.000	350.907	365.574
bedrooms	-3.842e+04	2597.274	-14.792	0.000	-4.35e+04	-3.33e+04
bathrooms	4.421e+04	3397.633	13.013	0.000	3.76e+04	5.09e+04

Omnibus:	202.137	**Durbin-Watson:**	2.018
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	219.402
Skew:	0.196	**Prob(JB):**	2.28e-48
Kurtosis:	3.265	**Cond. No.**	9.24e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 9.24e+03. This might indicate that there are strong multicollinearity or other numerical problems.

Interpretation of Fourth Model

Adding in Bedrooms slightly increased the R-squared value, for some reason this model claims that adding a bedroom subtracts from the value of the house, this feature will be left in for now but may be removed later on.

Now that the top correlated features have been tested, lets add in the remaining features as well as the One Hot Encoded features

Fifth Model: Adding in remaining features

In [44]:
```
y = kc_regression_data['price']
X_fifth_model = kc_regression_data.drop(['price', 'price_per_sqft'], axis


fifth_model = sm.OLS(y, sm.add_constant(X_fifth_model))
fifth_results = fifth_model.fit()
fifth_results.summary()
```

Out[44]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.560
Model:	OLS	**Adj. R-squared:**	0.559
Method:	Least Squares	**F-statistic:**	766.0
Date:	Fri, 10 Mar 2023	**Prob (F-statistic):**	0.00
Time:	11:02:23	**Log-Likelihood:**	-3.2668e+05
No. Observations:	23512	**AIC:**	6.534e+05
Df Residuals:	23472	**BIC:**	6.538e+05
Df Model:	39		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-7863.1931	4.88e+04	-0.161	0.872	-1.03e+05	8.77e+04
bedrooms	-7182.2980	2540.870	-2.827	0.005	-1.22e+04	-2202.027
bathrooms	2.235e+04	3541.219	6.312	0.000	1.54e+04	2.93e+04
sqft_living	291.3712	5.790	50.319	0.000	280.022	302.721
sqft_lot	0.0571	0.038	1.510	0.131	-0.017	0.131
floors	2714.7472	4219.753	0.643	0.520	-5556.243	1.1e+04
waterfront	-2.5e+04	1.91e+04	-1.306	0.192	-6.25e+04	1.25e+04
greenbelt	1.28e+05	1.1e+04	11.592	0.000	1.06e+05	1.5e+05
nuisance	2.9e+04	4676.357	6.202	0.000	1.98e+04	3.82e+04
sqft_above	-41.0548	5.398	-7.606	0.000	-51.635	-30.474
sqft_garage	-144.9419	13.567	-10.684	0.000	-171.533	-118.351
sqft_patio	16.3992	9.405	1.744	0.081	-2.034	34.833
has_garage	4121.6796	7102.096	0.580	0.562	-9798.891	1.8e+04
has_patio	-6858.3189	4846.903	-1.415	0.157	-1.64e+04	2641.926
view_AVERAGE	5.973e+04	7677.921	7.779	0.000	4.47e+04	7.48e+04
view_EXCELLENT	9.639e+04	2.11e+04	4.578	0.000	5.51e+04	1.38e+05
view_FAIR	9.277e+04	2.32e+04	3.995	0.000	4.73e+04	1.38e+05
view_GOOD	6.005e+04	1.19e+04	5.029	0.000	3.66e+04	8.35e+04
condition_2	6.26e+04	4.95e+04	1.265	0.206	-3.44e+04	1.6e+05
condition_3	1.124e+05	4.51e+04	2.492	0.013	2.4e+04	2.01e+05
condition_4	1.568e+05	4.51e+04	3.473	0.001	6.83e+04	2.45e+05
condition_5	2.045e+05	4.53e+04	4.510	0.000	1.16e+05	2.93e+05
month_sold_2	3.647e+04	1.16e+04	3.155	0.002	1.38e+04	5.91e+04
month_sold_3	7.534e+04	1.07e+04	7.021	0.000	5.43e+04	9.64e+04

month_sold_4	9.069e+04	1.07e+04	8.486	0.000	6.97e+04	1.12e+05
month_sold_5	1.077e+05	1.07e+04	10.103	0.000	8.68e+04	1.29e+05
month_sold_6	1.848e+04	1.03e+04	1.795	0.073	-1696.715	3.87e+04
month_sold_7	1.969e+04	1.01e+04	1.958	0.050	-17.854	3.94e+04
month_sold_8	1.518e+04	1e+04	1.516	0.129	-4442.643	3.48e+04
month_sold_9	-2420.3371	1.02e+04	-0.238	0.812	-2.24e+04	1.75e+04
month_sold_10	1.442e+04	1.02e+04	1.409	0.159	-5641.054	3.45e+04
month_sold_11	1.075e+04	1.04e+04	1.031	0.302	-9679.669	3.12e+04
month_sold_12	7927.4250	1.1e+04	0.724	0.469	-1.35e+04	2.94e+04
grade_6	4.426e+04	1.82e+04	2.438	0.015	8678.858	7.98e+04
grade_7	1.258e+05	1.78e+04	7.068	0.000	9.09e+04	1.61e+05
grade_8	2.471e+05	1.83e+04	13.495	0.000	2.11e+05	2.83e+05
grade_9	4.353e+05	1.93e+04	22.568	0.000	3.97e+05	4.73e+05
grade_10	6.163e+05	2.22e+04	27.798	0.000	5.73e+05	6.6e+05
grade_11	7.662e+05	3.62e+04	21.191	0.000	6.95e+05	8.37e+05
grade_12	5.806e+05	1.19e+05	4.873	0.000	3.47e+05	8.14e+05

Omnibus:	344.914	**Durbin-Watson:**	2.018
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	539.376
Skew:	0.149	**Prob(JB):**	7.51e-118
Kurtosis:	3.680	**Cond. No.**	3.48e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.48e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

Interpretation of Fifth Model

As expected, adding in the remaining predictors has greatly increased our adjusted R-squared value to .0558 while remaining significant overall. A number of the features were found to be non-significant, including the constant value. Those features not found to have a significant relationship with price will be removed.

Sixth Model: Removing Non-signifcant features

In [45]:
```
y = kc_regression_data['price']
X_sixth_model = kc_regression_data.drop(['price', 'price_per_sqft',
                                         'has_patio','sqft_lot','waterfro
                                         'month_sold_12','month_sold_11',
                                         'month_sold_9','month_sold_8','m
                                         'month_sold_6','condition_2','sq


sixth_model = sm.OLS(y, sm.add_constant(X_sixth_model))
sixth_results = sixth_model.fit()
sixth_results.summary()
```

Out[45]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.560
Model:	OLS	**Adj. R-squared:**	0.559
Method:	Least Squares	**F-statistic:**	1105.
Date:	Fri, 10 Mar 2023	**Prob (F-statistic):**	0.00
Time:	11:02:23	**Log-Likelihood:**	-3.2669e+05
No. Observations:	23512	**AIC:**	6.534e+05
Df Residuals:	23484	**BIC:**	6.537e+05
Df Model:	27		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	5.433e+04	2.53e+04	2.149	0.032	4786.533	1.04e+05
bedrooms	-7602.4802	2524.379	-3.012	0.003	-1.26e+04	-2654.534
bathrooms	2.205e+04	3532.776	6.240	0.000	1.51e+04	2.9e+04
sqft_living	292.4548	5.718	51.145	0.000	281.247	303.663
floors	1770.4967	4189.264	0.423	0.673	-6440.733	9981.726
greenbelt	1.271e+05	1.1e+04	11.516	0.000	1.05e+05	1.49e+05
nuisance	2.954e+04	4669.583	6.327	0.000	2.04e+04	3.87e+04
sqft_above	-40.8583	5.350	-7.638	0.000	-51.344	-30.373
sqft_garage	-141.5191	13.442	-10.528	0.000	-167.867	-115.171
has_garage	2296.0172	7016.303	0.327	0.743	-1.15e+04	1.6e+04
view_AVERAGE	5.978e+04	7602.598	7.864	0.000	4.49e+04	7.47e+04
view_EXCELLENT	8.882e+04	1.96e+04	4.535	0.000	5.04e+04	1.27e+05
view_FAIR	9.272e+04	2.32e+04	3.994	0.000	4.72e+04	1.38e+05
view_GOOD	6.063e+04	1.17e+04	5.162	0.000	3.76e+04	8.36e+04
condition_3	6.039e+04	1.91e+04	3.161	0.002	2.29e+04	9.78e+04
condition_4	1.06e+05	1.92e+04	5.522	0.000	6.84e+04	1.44e+05
condition_5	1.539e+05	1.96e+04	7.832	0.000	1.15e+05	1.92e+05
month_sold_2	2.481e+04	7905.333	3.139	0.002	9319.005	4.03e+04
month_sold_3	6.379e+04	6628.791	9.623	0.000	5.08e+04	7.68e+04
month_sold_4	7.886e+04	6547.263	12.045	0.000	6.6e+04	9.17e+04
month_sold_5	9.619e+04	6492.629	14.815	0.000	8.35e+04	1.09e+05
grade_6	4.444e+04	1.81e+04	2.450	0.014	8883.591	8e+04
grade_7	1.263e+05	1.78e+04	7.102	0.000	9.14e+04	1.61e+05
grade_8	2.481e+05	1.83e+04	13.572	0.000	2.12e+05	2.84e+05

grade_9	4.372e+05	1.92e+04	22.730	0.000	3.99e+05	4.75e+05
grade_10	6.196e+05	2.21e+04	28.014	0.000	5.76e+05	6.63e+05
grade_11	7.691e+05	3.61e+04	21.295	0.000	6.98e+05	8.4e+05
grade_12	5.921e+05	1.19e+05	4.977	0.000	3.59e+05	8.25e+05

Omnibus:	342.497	**Durbin-Watson:**	2.018
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	536.500
Skew:	0.147	**Prob(JB):**	3.17e-117
Kurtosis:	3.679	**Cond. No.**	2.03e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.03e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

Interpretation of Sixth Model

Removing the non-significant features did not change the Adjusted R-Squared value nor has it turned any other feature's p-values above 0.05.

It also changed out constant P-value to below 0.05

The issues remaining are with the bedrooms, floors and sqft_garage features, I am unable to justify why an additional bedroom or floor would reduce the value of a home, I am unsure about this predictor value and therefore will remove it. sqft_garage I have less understanding about trends and will leave it in.

Seventh Model: Removing Bedrooms and Floors

In [46]:
```
y = kc_regression_data['price']
X_seventh_model = kc_regression_data.drop(['price', 'price_per_sqft',
                                           'has_patio','sqft_lot','waterfro
                                           'month_sold_12','month_sold_11',
                                           'month_sold_9','month_sold_8','m
                                           'month_sold_6','condition_2','sq
                                            'bedrooms','floors'], axis=1)


seventh_model = sm.OLS(y, sm.add_constant(X_seventh_model))
seventh_results = seventh_model.fit()
seventh_results.summary()
```

Out[46]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.559
Model:	OLS	**Adj. R-squared:**	0.559
Method:	Least Squares	**F-statistic:**	1193.
Date:	Fri, 10 Mar 2023	**Prob (F-statistic):**	0.00
Time:	11:02:23	**Log-Likelihood:**	-3.2669e+05
No. Observations:	23512	**AIC:**	6.534e+05
Df Residuals:	23486	**BIC:**	6.536e+05
Df Model:	25		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.558e+04	2.48e+04	1.836	0.066	-3088.552	9.42e+04
bathrooms	2.062e+04	3315.655	6.219	0.000	1.41e+04	2.71e+04
sqft_living	285.7102	5.008	57.055	0.000	275.895	295.526
greenbelt	1.275e+05	1.1e+04	11.556	0.000	1.06e+05	1.49e+05
nuisance	2.961e+04	4668.746	6.342	0.000	2.05e+04	3.88e+04
sqft_above	-40.0844	4.779	-8.387	0.000	-49.452	-30.717
sqft_garage	-140.8918	13.165	-10.702	0.000	-166.697	-115.087
has_garage	1680.1031	7008.838	0.240	0.811	-1.21e+04	1.54e+04
view_AVERAGE	6.095e+04	7592.364	8.028	0.000	4.61e+04	7.58e+04
view_EXCELLENT	9.244e+04	1.95e+04	4.731	0.000	5.41e+04	1.31e+05
view_FAIR	9.304e+04	2.32e+04	4.007	0.000	4.75e+04	1.39e+05
view_GOOD	6.25e+04	1.17e+04	5.330	0.000	3.95e+04	8.55e+04
condition_3	6.135e+04	1.91e+04	3.212	0.001	2.39e+04	9.88e+04
condition_4	1.059e+05	1.92e+04	5.518	0.000	6.83e+04	1.44e+05
condition_5	1.537e+05	1.96e+04	7.825	0.000	1.15e+05	1.92e+05
month_sold_2	2.479e+04	7905.598	3.135	0.002	9290.970	4.03e+04
month_sold_3	6.385e+04	6629.729	9.630	0.000	5.09e+04	7.68e+04
month_sold_4	7.896e+04	6548.232	12.058	0.000	6.61e+04	9.18e+04
month_sold_5	9.611e+04	6492.821	14.803	0.000	8.34e+04	1.09e+05
grade_6	4.293e+04	1.81e+04	2.368	0.018	7390.929	7.85e+04
grade_7	1.252e+05	1.77e+04	7.054	0.000	9.04e+04	1.6e+05
grade_8	2.497e+05	1.81e+04	13.765	0.000	2.14e+05	2.85e+05
grade_9	4.409e+05	1.9e+04	23.185	0.000	4.04e+05	4.78e+05
grade_10	6.258e+05	2.19e+04	28.584	0.000	5.83e+05	6.69e+05

grade_11	7.773e+05	3.6e+04	21.606	0.000	7.07e+05	8.48e+05
grade_12	6.032e+05	1.19e+05	5.073	0.000	3.7e+05	8.36e+05

Omnibus:	341.102	**Durbin-Watson:**	2.018
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	532.860
Skew:	0.147	**Prob(JB):**	1.95e-116
Kurtosis:	3.676	**Cond. No.**	2.03e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.03e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

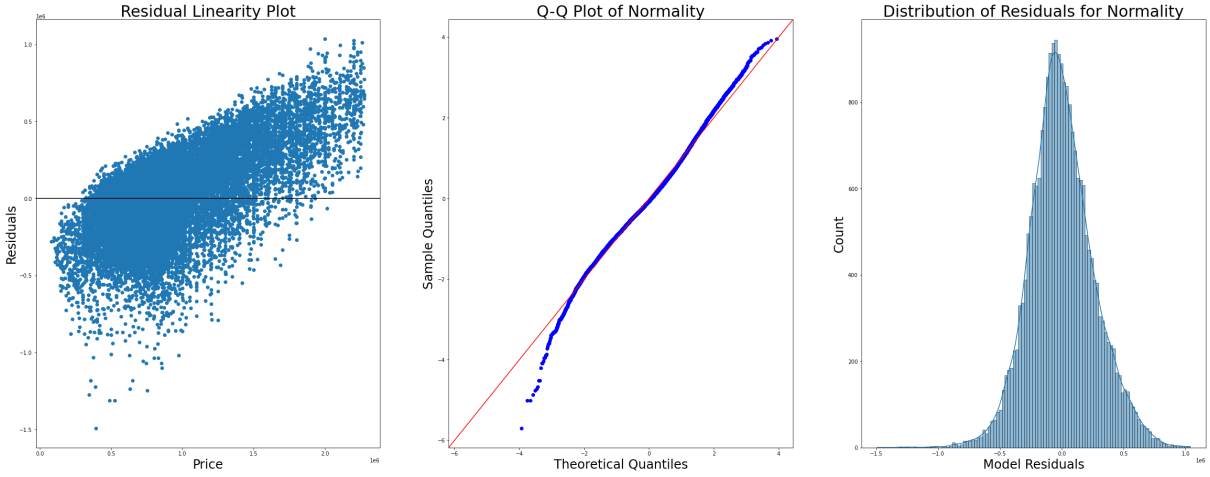
Interpretation of Seventh Model

Removing Bedrooms and floors did not effect the adjusted R-squared value, but the constant value is again insignificant. I am fairly happy with this model so far and now will check to be sure all of the multilinear regression assumptions are met, before attempt to deal with the insignificant constant value

In [47]:
```
model_assump_test(seventh_results, X_seventh_model, 'Seventh Model', y)
```

Linear Regression Assumption Testing Results for Seventh Model

✔ Linearity Assumption passed
✘ Normality Assumption NOT passed
✔ Homoscedasticity Assumption passed

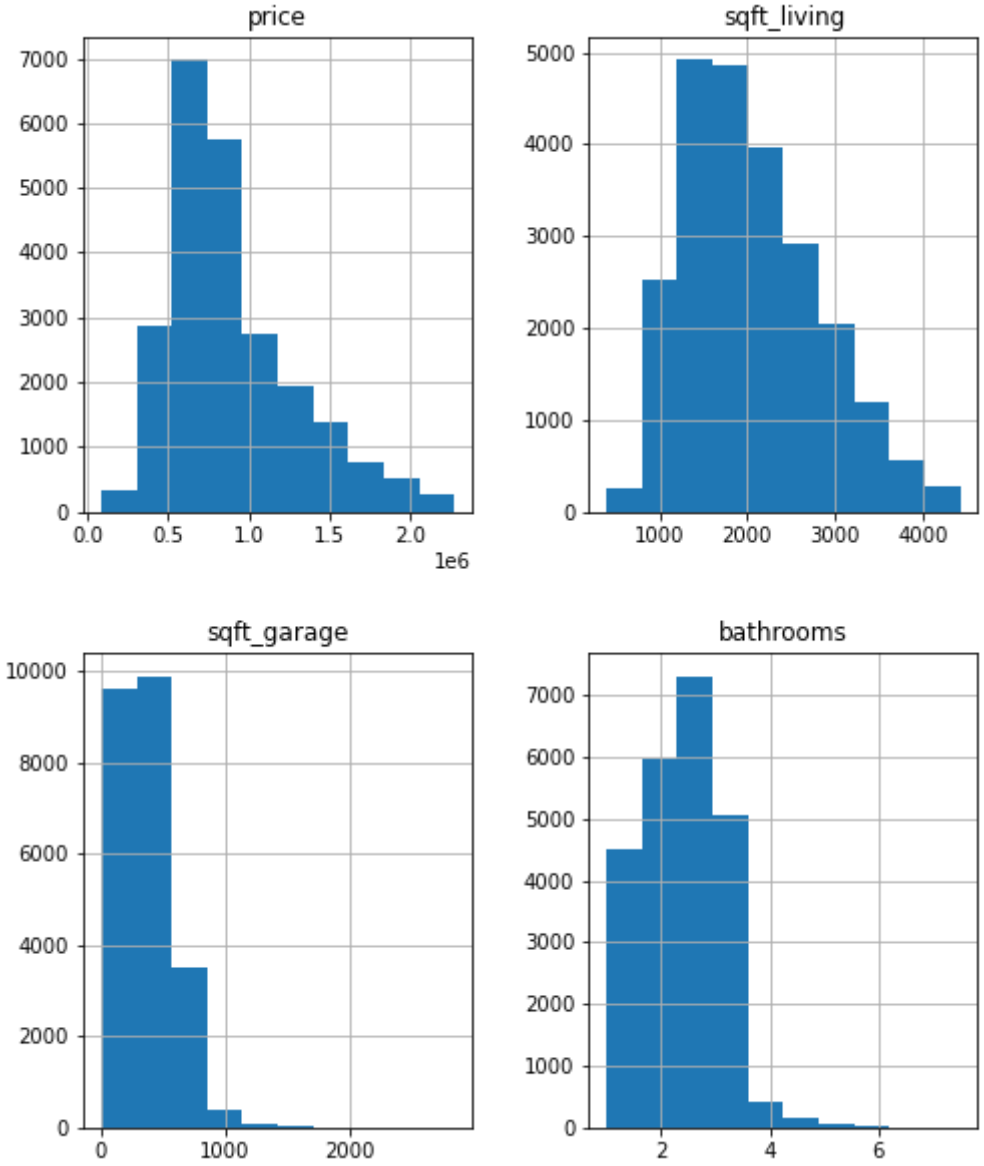


Seventh Model Assumptions

This model passes the Linearity assumption as well as the homescedasticity assumption but there are issues with the normality of the residuals. These normality issues do not seem to be huge issues as visually the Q-Q plot and histogram of model residuals both look very close to normal.

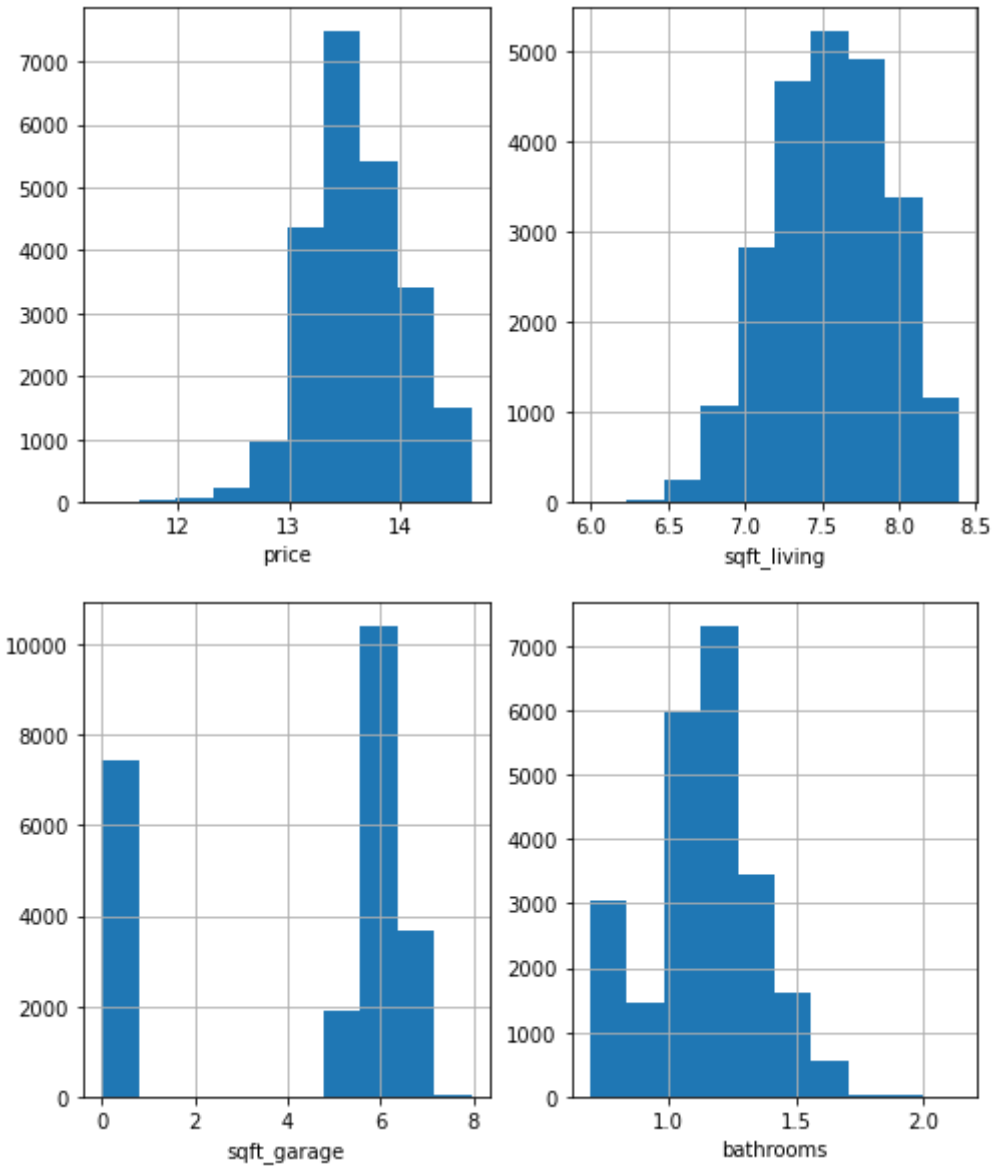
In an attempt to fix the normality issue, we will look at the distributions of all of the continuous features and log transform those that will benefit from a log transformation.

In [48]:
```
#Checking Normality of continous variables

seventh_vars = [['price', 'sqft_living', 'sqft_garage', 'bathrooms']]
for col in seventh_vars:
    kc_regression_data[col].hist(figsize=(8, 10));
```



In [49]:
```
#Log transformed continous variables

fig, ax = plt.subplots(ncols = 2, nrows = 2, figsize=(8,10))
np.log(kc_regression_data["price"]).hist(ax = ax[0][0])
ax[0][0].set_xlabel("price")

np.log(kc_regression_data["sqft_living"]).hist(ax = ax[0][1])
ax[0][1].set_xlabel("sqft_living")

np.log1p(kc_regression_data['sqft_garage']).hist(ax = ax[1][0]); #log1p u
ax[1][0].set_xlabel("sqft_garage")

np.log1p(kc_regression_data['bathrooms']).hist(ax = ax[1][1]);
ax[1][1].set_xlabel("bathrooms")
```

Out[49]:
```
Text(0.5, 0, 'bathrooms')
```



After the log transformation, sqft_living and sqft_garage became more normal in shape.

Therefore these transformations will be applied to the regression.

In [50]:
```
#Creating log transformed columns
kc_regression_data['log_sqft_garage'] = np.log1p(kc_regression_data['sqft
kc_regression_data['log_sqft_living'] = np.log(kc_regression_data['sqft_l
```

Eighth Model: Adding Log Transformed Sqft Garage and Sqft Living

In [51]:
```
y = kc_regression_data['price']
X_eighth_model = kc_regression_data.drop(['price', 'price_per_sqft',
                                          'has_patio','sqft_lot','waterfro
                                          'month_sold_12','month_sold_11',
                                          'month_sold_9','month_sold_8','m
                                          'month_sold_6','condition_2','sq
                                           'bedrooms','floors','sqft_livin


eighth_model = sm.OLS(y, sm.add_constant(X_eighth_model))
eighth_results = eighth_model.fit()
eighth_results.summary()
```

Out[51]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.552
Model:	OLS	**Adj. R-squared:**	0.552
Method:	Least Squares	**F-statistic:**	1160.
Date:	Fri, 10 Mar 2023	**Prob (F-statistic):**	0.00
Time:	11:02:24	**Log-Likelihood:**	-3.2688e+05
No. Observations:	23512	**AIC:**	6.538e+05
Df Residuals:	23486	**BIC:**	6.540e+05
Df Model:	25		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-3.196e+06	6.64e+04	-48.112	0.000	-3.33e+06	-3.07e+06
bathrooms	2.778e+04	3334.760	8.332	0.000	2.12e+04	3.43e+04
greenbelt	1.306e+05	1.11e+04	11.737	0.000	1.09e+05	1.52e+05
nuisance	3.075e+04	4711.106	6.527	0.000	2.15e+04	4e+04
sqft_above	-5.4284	4.606	-1.178	0.239	-14.457	3.600
has_garage	4.469e+05	3.82e+04	11.714	0.000	3.72e+05	5.22e+05
view_AVERAGE	6.777e+04	7642.907	8.867	0.000	5.28e+04	8.27e+04
view_EXCELLENT	1.039e+05	1.97e+04	5.280	0.000	6.53e+04	1.43e+05
view_FAIR	9.974e+04	2.34e+04	4.263	0.000	5.39e+04	1.46e+05
view_GOOD	7.82e+04	1.18e+04	6.631	0.000	5.51e+04	1.01e+05
condition_3	5.711e+04	1.92e+04	2.967	0.003	1.94e+04	9.48e+04
condition_4	1.032e+05	1.93e+04	5.332	0.000	6.52e+04	1.41e+05
condition_5	1.525e+05	1.98e+04	7.699	0.000	1.14e+05	1.91e+05
month_sold_2	2.44e+04	7967.206	3.062	0.002	8779.508	4e+04
month_sold_3	6.11e+04	6681.930	9.145	0.000	4.8e+04	7.42e+04
month_sold_4	7.785e+04	6599.846	11.795	0.000	6.49e+04	9.08e+04
month_sold_5	9.488e+04	6544.122	14.498	0.000	8.21e+04	1.08e+05
grade_6	-2630.7866	1.83e+04	-0.144	0.886	-3.85e+04	3.33e+04
grade_7	4.432e+04	1.81e+04	2.454	0.014	8927.708	7.97e+04
grade_8	1.702e+05	1.85e+04	9.220	0.000	1.34e+05	2.06e+05
grade_9	3.857e+05	1.93e+04	19.981	0.000	3.48e+05	4.24e+05
grade_10	6.053e+05	2.21e+04	27.408	0.000	5.62e+05	6.49e+05
grade_11	7.889e+05	3.62e+04	21.808	0.000	7.18e+05	8.6e+05
grade_12	5.958e+05	1.2e+05	4.973	0.000	3.61e+05	8.31e+05

log_sqft_garage	-8.567e+04	6325.729	-13.543	0.000	-9.81e+04	-7.33e+04
log_sqft_living	5.07e+05	9621.489	52.699	0.000	4.88e+05	5.26e+05

Omnibus:	421.513	**Durbin-Watson:**	2.015
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	566.074
Skew:	0.233	**Prob(JB):**	1.20e-123
Kurtosis:	3.600	**Cond. No.**	1.33e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.33e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

Eighth Model Regression Results

- The model is statistically significant overall, with an F-statistic p-value well below 0.05
- The model explains about 55% of the variance in sales price
- grade_6's p-value is not significant and leading me to believe that there is no direct relationship with price. This will be removed
- Our Constant value has become significant

This is an accepatable R-squared and at this point all of the variables that it makes sense to include in the regression have been included.

Ninth Model: Removing Grade_6

In [52]:
```
y = kc_regression_data['price']
X_ninth_model = kc_regression_data.drop(['price', 'price_per_sqft',
                                         'has_patio','sqft_lot','waterfro
                                         'month_sold_12','month_sold_11',
                                         'month_sold_9','month_sold_8','m
                                         'month_sold_6','condition_2','sq
                                          'bedrooms','floors','sqft_livin
                                         'sqft_garage','grade_6'], axis=1


ninth_model = sm.OLS(y, sm.add_constant(X_ninth_model))
ninth_results = ninth_model.fit()
ninth_results.summary()
```

Out[52]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.552
Model:	OLS	**Adj. R-squared:**	0.552
Method:	Least Squares	**F-statistic:**	1208.
Date:	Fri, 10 Mar 2023	**Prob (F-statistic):**	0.00
Time:	11:02:24	**Log-Likelihood:**	-3.2688e+05
No. Observations:	23512	**AIC:**	6.538e+05
Df Residuals:	23487	**BIC:**	6.540e+05
Df Model:	24		
Covariance Type:	nonrobust		

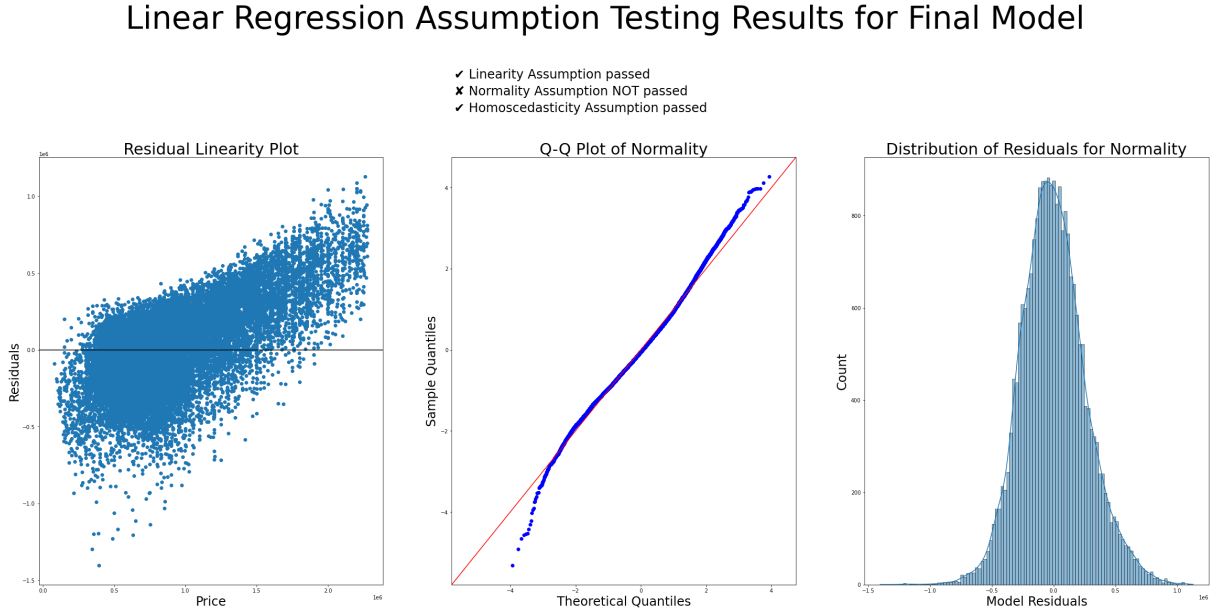
	coef	std err	t	P>|t|	[0.025	0.975]
const	-3.198e+06	6.56e+04	-48.718	0.000	-3.33e+06	-3.07e+06
bathrooms	2.779e+04	3334.064	8.336	0.000	2.13e+04	3.43e+04
greenbelt	1.306e+05	1.11e+04	11.738	0.000	1.09e+05	1.52e+05
nuisance	3.075e+04	4711.007	6.527	0.000	2.15e+04	4e+04
sqft_above	-5.4024	4.603	-1.174	0.240	-14.424	3.619
has_garage	4.467e+05	3.81e+04	11.715	0.000	3.72e+05	5.21e+05
view_AVERAGE	6.779e+04	7641.543	8.871	0.000	5.28e+04	8.28e+04
view_EXCELLENT	1.04e+05	1.97e+04	5.284	0.000	6.54e+04	1.43e+05
view_FAIR	9.974e+04	2.34e+04	4.263	0.000	5.39e+04	1.46e+05
view_GOOD	7.823e+04	1.18e+04	6.634	0.000	5.51e+04	1.01e+05
condition_3	5.695e+04	1.92e+04	2.964	0.003	1.93e+04	9.46e+04
condition_4	1.03e+05	1.93e+04	5.332	0.000	6.51e+04	1.41e+05
condition_5	1.523e+05	1.98e+04	7.704	0.000	1.14e+05	1.91e+05
month_sold_2	2.44e+04	7967.040	3.062	0.002	8779.967	4e+04
month_sold_3	6.111e+04	6681.586	9.146	0.000	4.8e+04	7.42e+04
month_sold_4	7.785e+04	6599.707	11.796	0.000	6.49e+04	9.08e+04
month_sold_5	9.488e+04	6543.971	14.499	0.000	8.21e+04	1.08e+05
grade_7	4.673e+04	6622.837	7.057	0.000	3.38e+04	5.97e+04
grade_8	1.727e+05	7518.561	22.966	0.000	1.58e+05	1.87e+05
grade_9	3.882e+05	9345.377	41.536	0.000	3.7e+05	4.06e+05
grade_10	6.077e+05	1.42e+04	42.734	0.000	5.8e+05	6.36e+05
grade_11	7.913e+05	3.2e+04	24.717	0.000	7.29e+05	8.54e+05
grade_12	5.982e+05	1.19e+05	5.043	0.000	3.66e+05	8.31e+05
log_sqft_garage	-8.564e+04	6323.141	-13.545	0.000	-9.8e+04	-7.33e+04

log_sqft_living	5.069e+05	9590.159	52.859	0.000	4.88e+05	5.26e+05

Omnibus:	421.707	**Durbin-Watson:**	2.015
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	566.321
Skew:	0.233	**Prob(JB):**	1.06e-123
Kurtosis:	3.600	**Cond. No.**	1.31e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.31e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

In [53]:
```
model_assump_test(ninth_results, X_ninth_model, 'final Model', y)
```



Evaluating Final Model based on Mean Absolute Error

In [54]:
```
#Calculating Mean Absolute Error
mae = ninth_results.resid.abs().sum() / len(y)
print('MAE: The model is off by about',mae ,'dollars on a given predictio
```

```
MAE: The model is off by about 205858.23638543105 dollars on a given pr
ediction.
```

In [55]:
```
#Calculating Number of standard deviations of error in model
mae/kc_regression_data['price'].std()
```

Out[55]:
```
0.5216902876827003
```

While an MAE of 206,000 sounds very large that value only equates to about half of a standard of price in this dataset. While ideally we would have a smaller average error, half of an STD is acceptable to allow us to gain adequate information in order to give proper recomendations to the stakeholders about pricing trends within the dataset

Interpreting Final Model

- Our model explains about 55% of the variance in home price throughout this dataset.
- The overall model is significant based on a 0.05 alpha level.
- All of the coefficeints relationship's to price are significant, in other words a relationship between the feature and price exists
- This model passes the Linearity and Homoscedasticity Assumptions
- Model has a Low condition number, meaning multicollinearity between features is very unlikely
- While the Residuals are not completely normal, they are fairly normal as can been seen in the Q-Q plot and histogram of residuals, while breaking this assumption is not ideal, it is acceptable in a model of this nature.
- The average error of this model is about half of a standard deviation in price, or about 200,000 dollars

Interpreting Model Coefficents

GreenBelt:

> Being adjacent to a greenbelt provided an increase in overall value of about $126,000.

When deciding where to build a new construction, being next to a greenbelt should be highly considered

Nuisance:

> The homes having reported traffic noise or some other nuisance was assocaited with an overall increase of $36,000

Although it may seem counterintuitive, areas with noise or other nuisances seem to increase a home's value. This could be for a variety of reasons such as the noise being related to being in a heavily populated area. While the types of nuisances are not specified in this dataset, heavy traffic noise or other small inconviences should not wholly deter construction, but I would have troubling making a solid recomendation based on this metric without further information on the types of nuisances accounted for in this dataset.

View:

> Having a view overall increased the value of a home but the quality of the view did not relate to home price increase in the most intuiative way. A fair view, the lowest quality of view, increased the value of a home by about $106,000 compared with no view, which was greater than the increase for an average view. Excellent views still added the most value by a small margine when compared with no view at a $108,000 increase.

Based on these results, I would recommend construction in areas with fair rated views, as these will likely be on cheaper land compared to areas with better views but will still create a very large home value increase.

Grade:

> The value associated with an increase in construction grade basically acted as expected with higher grades fetching higher prices. The breakdown of price increases for each grade when compared with a Fair graded house was $52,000 for an Average grade, $175,000 for Good, $381,000 for Better, $585,000 for Very Good, $565,000 for Luxury amd $757,000 for Mansion grade construction

When constructing a home choosing a high grade of construction could be very beneficial and a cost/benefit analysis should be run to decide exactly what construction grade should be choosen. This model suggests that while better materials and craftsmanship will be a larger investment, it could be worth it for a steep home value increase, except in the case of Luxury homes as these are valued less than the Very Good grade

Bathrooms:

> For each additional bathroom added to a house it will add about $28,000 in value to the home.

If the price for installing and setting up an extra bathroom is below $28,000 it will likely be a wise investment that help the property not only be more valuable but more desirable to home buyers.

Sqft of living space:

> sqft_living was log transformed and therefore the interpretation of this of metric is that a 1% increase in sqft inside the home is associated with an increase in value of 480,000/100 or about $4,800 in home value

The results of this analysis seem to suggest that you will have a higher percent increase in home value when adding square footage to smaller homes. It is important to think of construction costs when considering this and a specific cost-benefit analysis should be used before blindly increasing square footage.

Sqft of Garage Space:

> sqft_garage was log transformed and therefore the interpretation of this of metric is that a 1% increase in sqft of the garage is associated with a decrease in value of 12,000/100 or about $1,200 in home value

While this may seem counter intuitive this may be for a variety of reasons that are not represented in this data, such as a home in the city may be of more value but not have garage space. Independent of the reasoning, this model suggests that you should not waste time or resources on building an excessive garage

Condition:

> As expected as the condition of a home increases, so does the value. An increase from a condition of poor to average resulted in a $60,000, poor to good was $103,000, and poor to excellent was $155,000

Increasing the overall condition of the house either through cleaning, repainting or other types of home maintence based on this model is one of the most effective ways to increase its value as each step up in condition, not including from poor to fair, will result in an increase of around at least $40,000.

Month sold:

> When comparing to a sale in Janurary, the months that are associated with the highest increase in sales value are March, April and May with an associated increase of $61,000, $77,000 and $ 95,000 respectively

Many of the months could not be spoken on as it seems that a sale during that month did not correlate to price in a reliable way. With the data we do have a sale is recommended for April or May

Conclusions and Recommendations

Choosing locations to Build Upon

When looking for land we are looking for the cheapest property that will provide great home value, for this we recommend searching for a "Fair" view but no better, next to a greenbelt as these will raise the value of a home on average about $106,000 and $126,000 respectively.

Home Construction

Increasing the construction grade will generally increase its value significantly but will also raise investment costs, for this reason we recommend choosing between a "Good","Better" and "Very Good" grade. We suggest running a cost/benefit analysis and choosing which of these options works best for your situation.

Home Presentation and Sale

When placing the home up for sale we recommend you spend a little extra to make sure it is in good condition as each step up in condition will net you around $50,000 in value, this could be as simple as cleaning and repainting and would provided a great return on investment.

We recommend trying to sell homes in the months of April or May to maximize the home sale price, choosing these months could net you around $80,000 more dollars when compared to selling the same house in another month.

Limitations

- The overall accuracy of this database is unknown and specifically where it is sourced from is not well understood.
- While there are definitions for each categorical variable value, for example Grade 1, Grade 2 or view being average, it is hard to say if these ratings are based on opinion or a myriad of calculating factors.
- The Final Model's residuals are very close to but not completely normal, this effects the models overall prediction capability
- An Adjuested R-Squared of only .55 is not good enough to make very specific predictions and therefore this model should be used more for general trends