Phase II Final Project Submission

- Student name: Sarah Lowing
- Student pace: self paced
- Scheduled project review date/time: Friday 3/3 1:30
- Instructor name: Abhineet
- Blog post URL: https://wordpress.com/post/datamonsterdotblog.wordpress.com/47

Overview

For this analysis we'll be using the King's County Sales dataset, which can be found in kc_house_data.csv. Each record represents a house sale in the Seattle area for the year 2021-2022. We will use this data to determine mean price of a home in each neighborhood using zipcodes to group areas. After ruling out potential price boosters, such as waterfront homes, or homes with access to amenities such as parks or greenways, we'll also examine the impact of home grade on the price of a home in order to maximize our saving potential for our client, a real estate agent looking to find the elusive midrange home for sale in the Kings County metro area.

Business Understanding

Purchasing a home is one of the largest financial commitments most people will make in their lifetimes. Todays real estate markets are suffering for a lack of supply of mid-sized starter homes and our client, a real estate agent, specializes in exactly these kinds of homes. We'll find the mean price of a home for each zipcode and then look to see what kinds of features- views, access to greenways, the overall condition/grade of a home are most likely to impact price in order to identify what kinds of areas and amenities we should avoid to find a resonably priced home in the extra tight market and highly inflated real estate market of Seattle. We'll end with a list of reccomendaed neighborhoods for our agent to keep tabs on.

Data Understanding

After checking our data for missing or null values, we wil begin performing some simple statistical analysis to determine which variables have an existing and obvious correlation to our target variable, 'price'. We'll examine the format of the data (numeric vs categorical, continuous vs discrete) to determine what, if any, transformations need to take place in order to perform colinearity.. And finally we'll check for normality, and look to apply transformations (linear, logarythmic, polynomial regression or to any data that does not conform to our L.I.N.E. acronym and is useful for our model and analysis.

Preliminary Data Analysis

In [88]:
```
# import neccessary libraries
import matplotlib.pyplot as plt
import numpy as np
```

```
import pandas as pd
import scipy.stats as stats
import seaborn as sns
import statsmodels.api as sm
```

Inspect and Clean Data

In [89]:
```
## Check the columns and first few rows
kc= pd.read_csv('Data/kc_house_data.csv')
len(kc)
```

Out[89]: 30155

In [205…
```
kc.head()
```

Out[205…

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

5 rows × 25 columns

In [4]:
```
# Generate summary statistics
kc.describe()
```

Out[4]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	
count	3.015500e+04	3.015500e+04	30155.000000	30155.000000	30155.000000	3.015500e+04	3
mean	4.538104e+09	1.108536e+06	3.413530	2.334737	2112.424739	1.672360e+04	
std	2.882587e+09	8.963857e+05	0.981612	0.889556	974.044318	6.038260e+04	

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot
min	1.000055e+06	2.736000e+04	0.000000	0.000000	3.000000	4.020000e+02
25%	2.064175e+09	6.480000e+05	3.000000	2.000000	1420.000000	4.850000e+03
50%	3.874011e+09	8.600000e+05	3.000000	2.500000	1920.000000	7.480000e+03
75%	7.287100e+09	1.300000e+06	4.000000	3.000000	2619.500000	1.057900e+04
max	9.904000e+09	3.075000e+07	13.000000	10.500000	15360.000000	3.253932e+06

We can see some of our mean values here, as well as visually inspect for for any missing values or irregularities. So far it looks good!

Checking for NaN

In [4]:
```
#checking for null values
kc.isnull().sum()
```

Out[4]:
```
id                0
date              0
price             0
bedrooms          0
bathrooms         0
sqft_living       0
sqft_lot          0
floors            0
waterfront        0
greenbelt         0
nuisance          0
view              0
condition         0
grade             0
heat_source      32
sewer_system     14
sqft_above        0
sqft_basement     0
sqft_garage       0
sqft_patio        0
yr_built          0
yr_renovated      0
address           0
lat               0
long              0
dtype: int64
```

In [90]:
```
#Because of the small number of NaN values, we can drop null rows
kc.dropna(inplace=True)
```

Inspect Datatypes of Columns

In [207…
```
#Visualize datatypes for future analysis
kc.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
```

We can see that the data has several different types. Some potentially relevant data for 'price' is categorical, like 'waterfront' and 'condition'. We'll need to OHE these features if we'd like to add those variables to our model. None of our numeric variables are categorical in nature, with the exception of 'id', 'lat' and 'long'. While we won't need these for our statistical analysis, 'lat' and 'long' could come in handy later if we want to map home sales.

Inspect Distribution of Variables

We'll need to check if the distribution of our variables appears normal, which will alert us to outliers or if we will need to transform our data in order to progress with an effective model. Let's view value counts of our kc dataframe to get a more concrete idea of how our categorical data is distributed and look for outliers

In [91]:
```
# Value counts for categorical data
categoricals = kc.select_dtypes("object")
categoricals=categoricals.drop(['date', 'address'], axis=1).copy()
```

In [92]:
```
for col in categoricals:
    print(kc[col].value_counts(), "\n")
```

```
NO     29600
YES      511
Name: waterfront, dtype: int64

NO     29339
YES      772
Name: greenbelt, dtype: int64

NO     24862
YES     5249
Name: nuisance, dtype: int64

NONE         26555
AVERAGE       1910
GOOD           877
```

```
EXCELLENT      549
FAIR           220
Name: view, dtype: int64

Average      18515
Good          8052
Very Good     3258
Fair           225
Poor            61
Name: condition, dtype: int64

7 Average        11693
8 Good            9400
9 Better          3804
6 Low Average     2852
10 Very Good      1369
11 Excellent       406
5 Fair             385
12 Luxury          122
4 Low               46
13 Mansion          24
3 Poor               9
2 Substandard        1
Name: grade, dtype: int64

Gas                  20576
Electricity           6460
Oil                   2899
Gas/Solar               93
Electricity/Solar       59
Other                   20
Oil/Solar                4
Name: heat_source, dtype: int64

PUBLIC                25767
PRIVATE                4336
PRIVATE RESTRICTED        5
PUBLIC RESTRICTED         3
Name: sewer_system, dtype: int64
```

Clean "grade"

We can see the vast number of houses on the market are of average quality. There are very, very few houses being sold in the Seattle area that in what could be considered 'bad' condition. We could consider predictors 'cabin' 'substandard', 'poor', 'luxury', and 'mansion' as outliers due to the small number of houses sold in those grades. It might be worthwhile to drop those categories and clean the reamining names.

In [8]:
```
grade_dict = {'1 Cabin':0, '2 Substandard':1, '3 Poor':2,'4 Low':3,'5 Fair':4,'6
              '9 Better':8, '10 Very Good':9, '11 Excellent':10, '12 Luxury':11,
kc.grade.replace(to_replace=grade_dict,inplace=True)
```

In [9]:
```
df_kc= kc[(kc['grade']<10) & (kc['grade']> 2)]
```

In [144…
```
df_kc['grade'].value_counts()
```

Out[144…
```
6    11693
7     9400
8     3804
5     2852
```

```
9     1369
4      385
3       46
Name: grade, dtype: int64
```

Adding Zipcodes to the DataFrame

So far everything looks good. But it would be nice to have an additional column, "zip", that contains the zip code of each entry so that we can make sure our records are all actually located in King's County, as well as to group houses in neighborhoods for our model later on. Let's make that now.

In [93]:
```
#extract zip codes.  We should have 224 different zipcodes -but there's an extra
df_kc['zip'] = df_kc['address'].str.findall(r'([0-9]\d+)').apply(
    lambda x: x[-1] if len(x) >= 1 else '')
```

In [94]:
```
# examine zip codes, kc county homes ahould start with 98 but we clearly have so
zip_counts= df_kc['zip'].value_counts()
zip_counts
```

Out[94]:
```
98042    988
98038    853
98103    761
98115    757
98117    748
        ...
58490      1
02790      1
85210      1
62281      1
85296      1
Name: zip, Length: 395, dtype: int64
```

In [12]:
```
# closer look at address to double check that entries are erroneous, and not our
df_bad_zips = df_kc[~df_kc['zip'].astype(str).str.startswith('98')]
df_bad_zips['address']
```

Out[12]:
```
12       5712 A Street, Omaha, Nebraska 68106, United S...
53       1820 South State Street, Vineland, New Jersey ...
62       1804 Spruce Street, McLeansboro, Illinois 6285...
172      11th Avenue, West Babylon, New York 11704, Uni...
196      South 25th Avenue, Bellevue, Nebraska 68123, U...
                              ...
30029    Avenue Cuts, 34 Ridge Rd, North Arlington, New...
30044    36th Avenue, Kenosha, Wisconsin 53142, United ...
30116    57th Street Lane NW, Rochester, Minnesota 5590...
30129    214 B, Mount Laurel, New Jersey 08054, United ...
30144    2954 Northwest 85th Street, Miami, Florida 331...
Name: address, Length: 907, dtype: object
```

In [ ]:
```
#Does the address have Seattle, Washington in it?
df_bad_zips[df_bad_zips['address'].str.contains('Washington')]
```

In [96]:
```
# Doesn't look that way, so we'll drop the entries from the df in that are not f
df_clean = df_kc[df_kc['zip'].astype(str).str.startswith('98')]
len(df_kc)
```

Out[96]:
```
29549
```

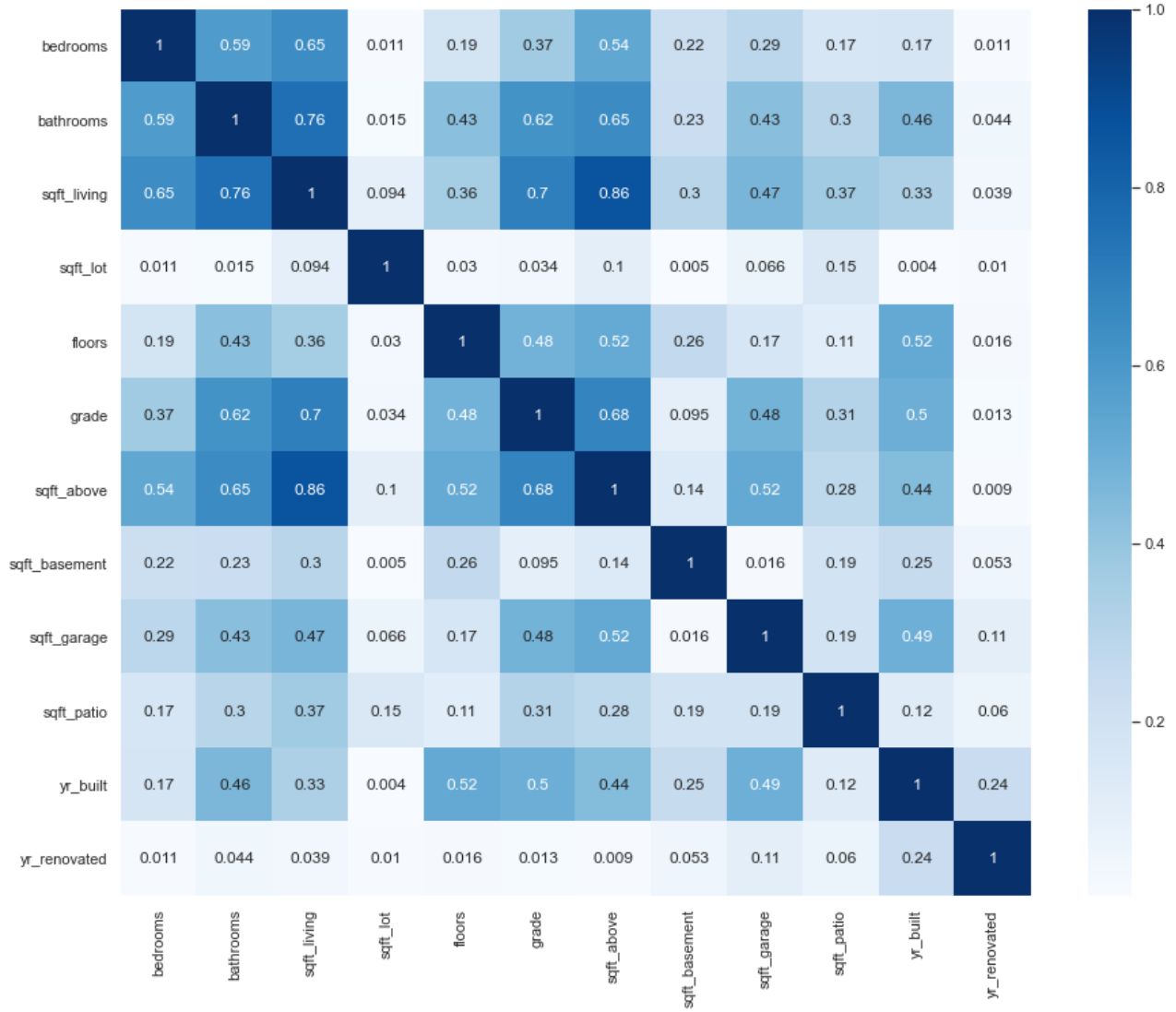
In [97]:
```
#Check if we've got the right number of records
bool(len(df_clean) + len(df_bad_zips)==len(df_kc))
```

Out[97]: True

Review of Numeric Variables

This is also a relatively long list of variables/columns. We'll want to make sure that there's no co-liniarity between the variables in our final model, so we'll take a look at a correlation heatmap of our dataframe to identify possible trouble spots for our numeric variables.

In [98]:
```
kc_colinearity = df_clean.drop(columns=['price', 'lat', 'long', 'id'])
corr = round(abs(kc_colinearity.corr()),3)
sns.set(rc = {'figure.figsize':(15,12)})
sns.heatmap(corr, cmap="Blues", annot=True);
```



We can see that there are some trouble spots in here, "sqft_living" has a collinear relationship with "sqft_above", followed by "bathrooms" and "bedrooms", if we use any of these variables it will be sqft_living. We've already cleaned/prepared the other variable of interest, 'grade' so we can we'll focus on examining and preparing "sqft_above" for use in a basic linear regression model.

Clean "sqft_living"

In [99]:
```
df_clean['sqft_living'].describe()
```

Out[99]:
```
count    28642.000000
mean      2076.709413
std        871.500140
min          3.000000
25%       1430.000000
50%       1920.000000
75%       2590.000000
max       7710.000000
Name: sqft_living, dtype: float64
```

Our std of 871 indicates that we should also be looking for outliers- when considering mean house size, that's a pretty large margin of error (almost 1/2 of the average sized house in KC) and so there must be houses that are exceptionally large that are skewing our model. We can also see from our min value of 3 that there might be some errors and outliers on the smaller side too, we'll need to remove some of these. However, we need to be very careful about how we do this, as removing data can lead to model manipulation and less acuuracy overall. We'll remove all houses in the typical oulier range of 3%, to minimize model manipulation, but gain some accuracy in our predictions. This will drop considerably fewer records than preserving the middle quartile range, so our model should still be fairly accurate.

In [100…
```
q_low = df_clean['sqft_living'].quantile(0.03)

q_high = df_clean['sqft_living'].quantile(0.97)
```

In [101…
```
kc_clean= df_clean[(df_clean['sqft_living']< q_high) & (df_clean['sqft_living']>
```

In [102…
```
len(df_clean)-len(kc_clean)
```

Out[102…
```
1767
```

In [103…
```
kc_clean['sqft_living'].describe()
```

Out[103…
```
count    26875.000000
mean      2043.032186
std        733.276050
min        850.000000
25%       1460.000000
50%       1930.000000
75%       2540.000000
max       4025.000000
Name: sqft_living, dtype: float64
```

We can see that our std has dropped to 733 from 871 for sqft_living by dropping 1767 outliers out of 28,642 records.

In [104…
```
# Build a model to fit a regression line and check the distribution of sqft_livi
yr=df_kc['price']
Xr=df_kc['sqft_living']

sgft_model_raw = sm.OLS(yr, sm.add_constant(Xr))
sqft_results_raw = sgft_model_raw.fit()
```

In [105…
```
# Build a model to fit a regression line and check the distribution of sqft_livi
y=df_clean['price']
X=df_clean['sqft_living']

sgft_model = sm.OLS(y, sm.add_constant(X))
sqft_results = sgft_model.fit()
```

In [106…
```
sqft_results.summary()
```

Out[106…

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.332
Model:	OLS	**Adj. R-squared:**	0.332
Method:	Least Squares	**F-statistic:**	1.424e+04
Date:	Thu, 02 Mar 2023	**Prob (F-statistic):**	0.00
Time:	13:33:59	**Log-Likelihood:**	-4.2006e+05
No. Observations:	28642	**AIC:**	8.401e+05
Df Residuals:	28640	**BIC:**	8.401e+05
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.053e+05	8647.412	12.176	0.000	8.83e+04	1.22e+05
sqft_living	458.2126	3.840	119.338	0.000	450.687	465.738

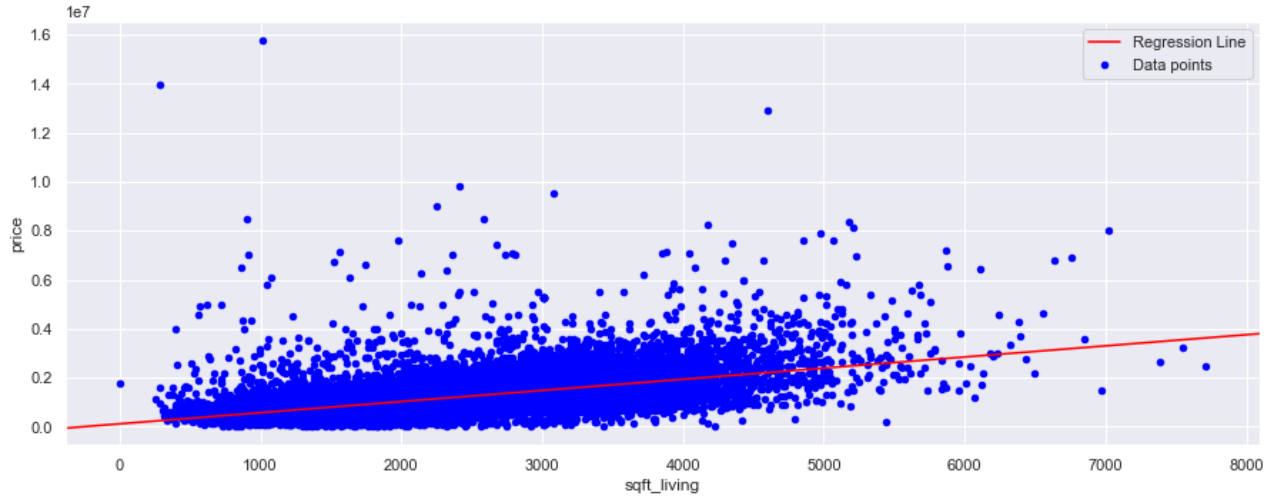
Omnibus:	27222.276	**Durbin-Watson:**	1.908
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	3861326.465
Skew:	4.196	**Prob(JB):**	0.00
Kurtosis:	59.259	**Cond. No.**	5.82e+03

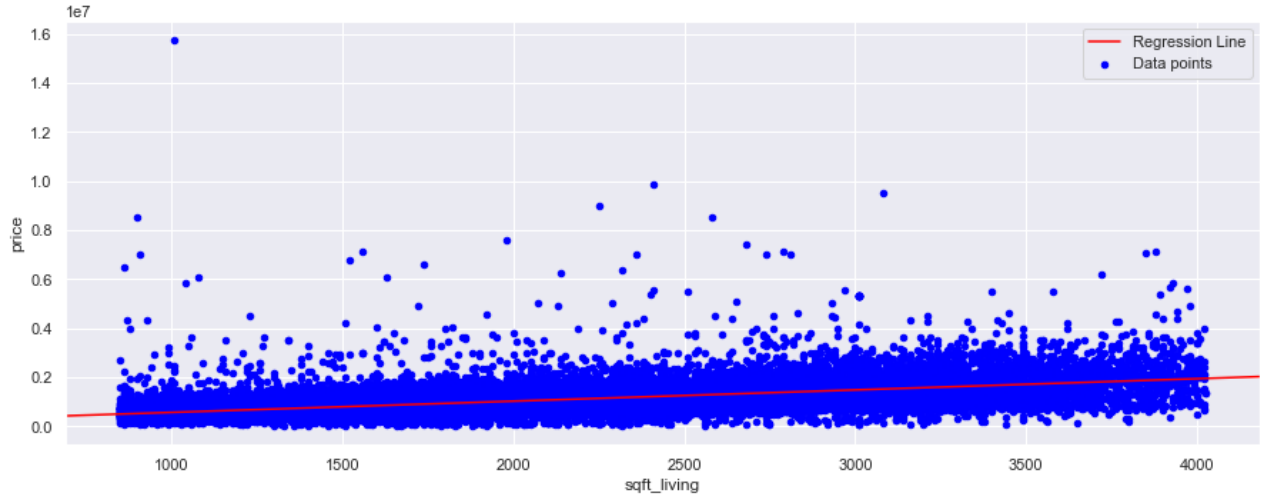
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.82e+03. This might indicate that there are strong multicollinearity or other numerical problems.

Because our model is only explaining about 36% of the variation in our model we can see what we had assumed before- this model is lacking a predictor that explains the jump in price in addition to sqft_living. Moreover the intercept of 100,530 doesn't make much sense, a house in any part of the country is bound to be more than that. We'll have to dig deeper to see what other factors influence price. In the meantime, let's take a look at the residuals to see if removing outliers had the desired effect on our data.

In [107…
```
fig, (ax1,ax2) = plt.subplots(2)
```

```
df_clean.plot.scatter(x="sqft_living", y="price", label="Data points", color='bl
sm.graphics.abline_plot(model_results=sqft_results_raw, label="Regression Line",
ax1.legend();

kc_clean.plot.scatter(x="sqft_living", y="price", label="Data points", color='bl
sm.graphics.abline_plot(model_results=sqft_results, label="Regression Line", col
ax2.legend();
```





Clean Price

We can see that minimizing outliers makes our data more normal distribution, but we're not actually seeing a sharp increase in price per sqft as expected. Instead, it seems that price increases only marginally as sqft does. We'll need to examine our other variables to see what else contributes to upticks in prices. But before we do that, let's examine our price data to check for a normal distribution and remove outliers. Then we'll look at a qq plot to examine residuals to see if a log transformation would be helpful for our final model

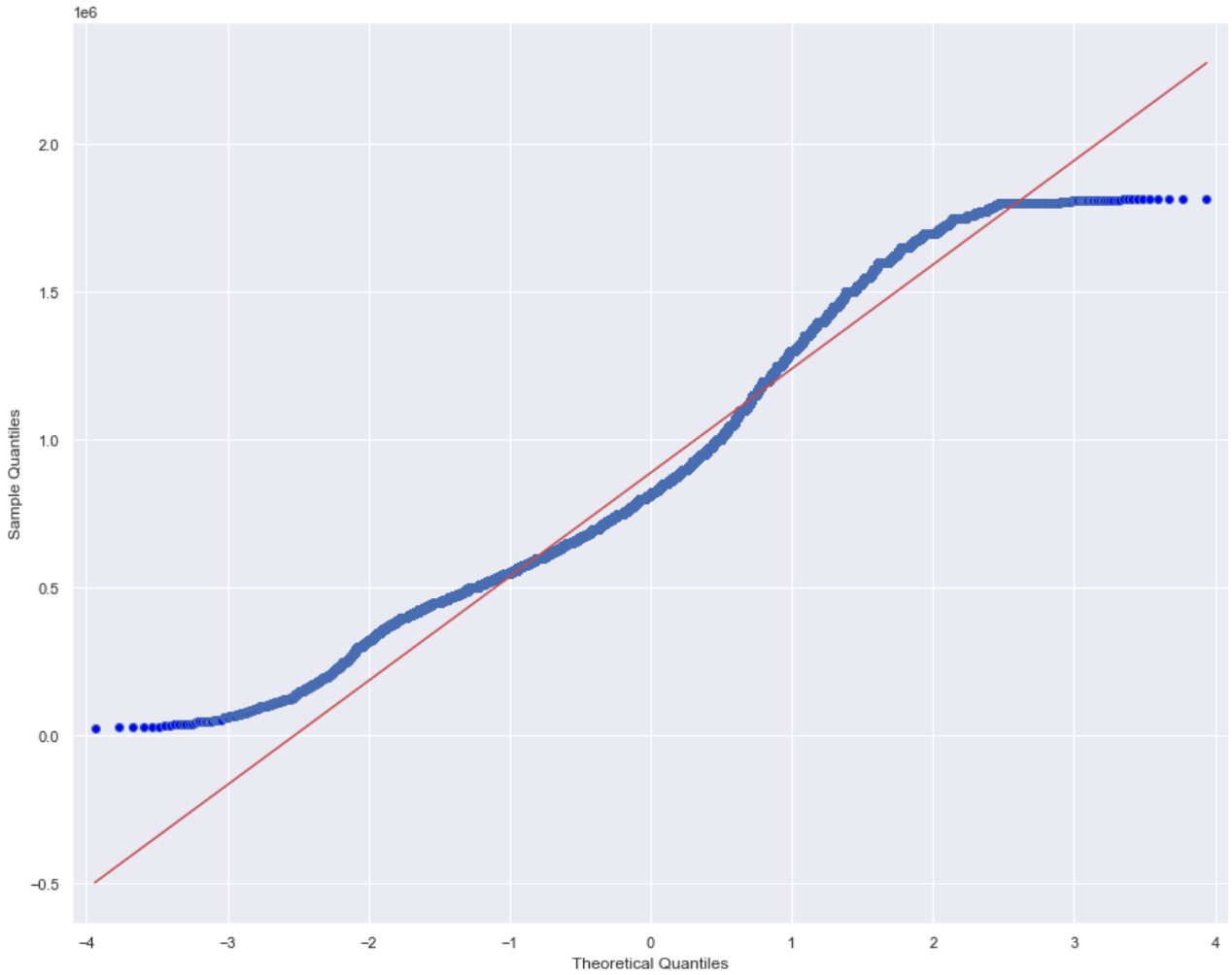
In [108…
```
kc_clean['price'].describe()
```

Out[108…
```
count    2.687500e+04
mean     1.023203e+06
std      6.059568e+05
min      2.756300e+04
25%      6.480000e+05
50%      8.550000e+05
75%      1.250000e+06
```

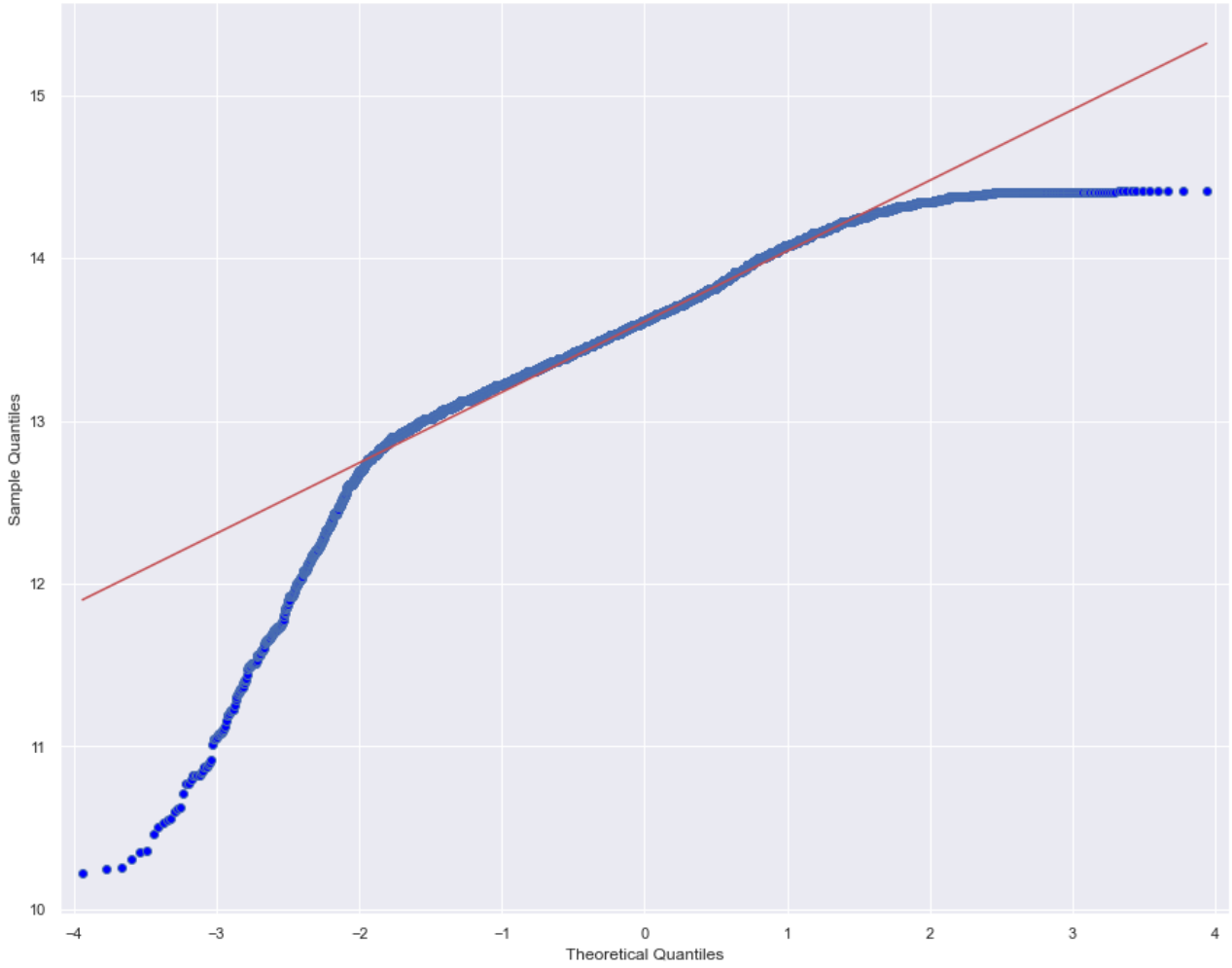
```
max      1.574000e+07
Name: price, dtype: float64
```

In [114…
```
std_thresh = kc_clean.price.std()*3
kc_clean= kc_clean[kc_clean['price'] <= std_thresh]
```

In [115…
```
sm.qqplot(kc_clean['price'], line='r');
```



In [116…
```
sm.qqplot(np.log(kc_clean['price']), line='r');
```



Looks like we'd still have a significant number of outliers on the bottom and top range of prices, but that a log transformation would be helpful in our final model

Summary: Numeric Variables

Our correlation matrix showed that sqft_living had the highest correlation to price. And we can see from the plot above that when we remove outliers there's a linear relationship between sqft_living and price. However, Our basic model does little to shed light on the predictors that might help us understand home values in KC. We'll need to examine some other variables to determine what features correlate most strongly to price.

Review of Categorical Data

Lets' begin by looking at a model of all our categorical data to see which predictors would be the best candidates for including in our final model.

In [117…
```
categoricals_clean = kc_clean.select_dtypes("object")
categoricals_clean=categoricals_clean.drop(['date', 'address', ], axis=1).copy()
```

In [118…
```
y_raw_cat = kc_clean["price"]
X_raw_cat = categoricals_clean
X_raw_cat = pd.get_dummies(X_raw_cat)

X_raw_cat.drop(['waterfront_YES', 'greenbelt_YES',
```

```
                'nuisance_YES', 'condition_Poor',
                'heat_source_Electricity', 'sewer_system_PUBLIC', 'zip_98125'],
                axis=1, inplace=True)
```

In [119…

```
cat_model= sm.OLS(y_raw_cat, sm.add_constant(X_raw_cat))
cat_results = cat_model.fit()
cat_results.summary()
```

Out[119…

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.473
Model:	OLS	**Adj. R-squared:**	0.471
Method:	Least Squares	**F-statistic:**	207.1
Date:	Thu, 02 Mar 2023	**Prob (F-statistic):**	0.00
Time:	13:38:17	**Log-Likelihood:**	-3.4106e+05
No. Observations:	24557	**AIC:**	6.823e+05
Df Residuals:	24450	**BIC:**	6.832e+05
Df Model:	106		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-8.889e+15	1.39e+16	-0.638	0.524	-3.62e+16	1.84e+16
waterfront_NO	-1.739e+04	1.89e+04	-0.919	0.358	-5.45e+04	1.97e+04
greenbelt_NO	-7.794e+04	1.18e+04	-6.623	0.000	-1.01e+05	-5.49e+04
nuisance_NO	5.07e+04	4614.683	10.987	0.000	4.17e+04	5.97e+04
view_AVERAGE	8.889e+15	1.39e+16	0.638	0.524	-1.84e+16	3.62e+16
view_EXCELLENT	8.889e+15	1.39e+16	0.638	0.524	-1.84e+16	3.62e+16
view_FAIR	8.889e+15	1.39e+16	0.638	0.524	-1.84e+16	3.62e+16
view_GOOD	8.889e+15	1.39e+16	0.638	0.524	-1.84e+16	3.62e+16
view_NONE	8.889e+15	1.39e+16	0.638	0.524	-1.84e+16	3.62e+16
condition_Average	2.144e+05	4.12e+04	5.206	0.000	1.34e+05	2.95e+05
condition_Fair	3.803e+04	4.57e+04	0.832	0.406	-5.16e+04	1.28e+05
condition_Good	2.274e+05	4.12e+04	5.514	0.000	1.47e+05	3.08e+05
condition_Very Good	2.716e+05	4.14e+04	6.557	0.000	1.9e+05	3.53e+05
heat_source_Electricity/Solar	-8.309e+04	3.8e+04	-2.185	0.029	-1.58e+05	-8537.177
heat_source_Gas	9.878e+04	4391.848	22.492	0.000	9.02e+04	1.07e+05
heat_source_Gas/Solar	2.166e+05	3.4e+04	6.377	0.000	1.5e+05	2.83e+05
heat_source_Oil	9805.5415	6452.949	1.520	0.129	-2842.632	2.25e+04
heat_source_Oil/Solar	1.166e+05	1.31e+05	0.892	0.372	-1.4e+05	3.73e+05
heat_source_Other	5.275e+04	6.79e+04	0.777	0.437	-8.03e+04	1.86e+05
sewer_system_PRIVATE	2667.3239	5810.149	0.459	0.646	-8720.923	1.41e+04

sewer_system_PRIVATE RESTRICTED	-2.216e+05	1.86e+05	-1.192	0.233	-5.86e+05	1.43e+05
sewer_system_PUBLIC RESTRICTED	2.627e+05	1.85e+05	1.423	0.155	-9.92e+04	6.25e+05
zip_98001	-3.095e+05	1.65e+04	-18.713	0.000	-3.42e+05	-2.77e+05
zip_98002	-4.284e+05	1.88e+04	-22.772	0.000	-4.65e+05	-3.91e+05
zip_98003	-3.58e+05	1.78e+04	-20.066	0.000	-3.93e+05	-3.23e+05
zip_98004	6.1e+05	3.71e+04	16.460	0.000	5.37e+05	6.83e+05
zip_98005	4.878e+05	3.29e+04	14.842	0.000	4.23e+05	5.52e+05
zip_98006	3.738e+05	1.96e+04	19.054	0.000	3.35e+05	4.12e+05
zip_98007	2.659e+05	2.64e+04	10.082	0.000	2.14e+05	3.18e+05
zip_98008	2.864e+05	1.93e+04	14.847	0.000	2.49e+05	3.24e+05
zip_98010	-1.446e+05	1.99e+04	-7.278	0.000	-1.84e+05	-1.06e+05
zip_98011	1.96e+05	2.12e+04	9.243	0.000	1.54e+05	2.38e+05
zip_98014	-2.164e+04	2.56e+04	-0.847	0.397	-7.17e+04	2.84e+04
zip_98019	3.32e+04	2.11e+04	1.570	0.116	-8249.780	7.46e+04
zip_98022	-3.348e+05	1.8e+04	-18.588	0.000	-3.7e+05	-2.99e+05
zip_98023	-3.379e+05	1.62e+04	-20.798	0.000	-3.7e+05	-3.06e+05
zip_98024	1.436e+05	3.19e+04	4.496	0.000	8.1e+04	2.06e+05
zip_98027	2.207e+05	2.01e+04	11.007	0.000	1.81e+05	2.6e+05
zip_98028	1.269e+05	1.9e+04	6.668	0.000	8.96e+04	1.64e+05
zip_98029	3.637e+05	2.08e+04	17.464	0.000	3.23e+05	4.05e+05
zip_98030	-2.926e+05	1.85e+04	-15.797	0.000	-3.29e+05	-2.56e+05
zip_98031	-2.927e+05	1.7e+04	-17.179	0.000	-3.26e+05	-2.59e+05
zip_98032	-3.424e+05	2.35e+04	-14.598	0.000	-3.88e+05	-2.96e+05
zip_98033	3.895e+05	1.99e+04	19.579	0.000	3.5e+05	4.28e+05
zip_98034	1.648e+05	1.65e+04	9.996	0.000	1.32e+05	1.97e+05
zip_98038	-1.379e+05	1.56e+04	-8.845	0.000	-1.68e+05	-1.07e+05
zip_98039	6.177e+05	2.61e+05	2.365	0.018	1.06e+05	1.13e+06
zip_98040	5.277e+05	3.04e+04	17.380	0.000	4.68e+05	5.87e+05
zip_98042	-2.508e+05	1.52e+04	-16.456	0.000	-2.81e+05	-2.21e+05
zip_98045	6.955e+04	1.79e+04	3.878	0.000	3.44e+04	1.05e+05
zip_98047	-4.018e+05	3.25e+04	-12.376	0.000	-4.65e+05	-3.38e+05
zip_98050	-1.468e+05	2.61e+05	-0.562	0.574	-6.59e+05	3.65e+05
zip_98051	-6.816e+04	3.81e+04	-1.789	0.074	-1.43e+05	6524.230
zip_98052	3.726e+05	1.8e+04	20.724	0.000	3.37e+05	4.08e+05
zip_98053	2.612e+05	2.04e+04	12.789	0.000	2.21e+05	3.01e+05

zip_98055	-2.248e+05	2.18e+04	-10.299	0.000	-2.68e+05	-1.82e+05
zip_98056	-5.29e+04	1.76e+04	-3.004	0.003	-8.74e+04	-1.84e+04
zip_98057	-2.89e+05	2.81e+04	-10.292	0.000	-3.44e+05	-2.34e+05
zip_98058	-1.81e+05	1.62e+04	-11.186	0.000	-2.13e+05	-1.49e+05
zip_98059	-6399.9099	1.73e+04	-0.369	0.712	-4.04e+04	2.76e+04
zip_98065	1.124e+05	2.09e+04	5.374	0.000	7.14e+04	1.53e+05
zip_98070	-7.95e+04	2.49e+04	-3.192	0.001	-1.28e+05	-3.07e+04
zip_98072	2.543e+05	2.07e+04	12.302	0.000	2.14e+05	2.95e+05
zip_98074	3.656e+05	2.05e+04	17.845	0.000	3.25e+05	4.06e+05
zip_98075	5.011e+05	2.29e+04	21.924	0.000	4.56e+05	5.46e+05
zip_98077	3.017e+05	2.47e+04	12.204	0.000	2.53e+05	3.5e+05
zip_98092	-2.408e+05	1.67e+04	-14.380	0.000	-2.74e+05	-2.08e+05
zip_98102	3.119e+05	2.84e+04	10.969	0.000	2.56e+05	3.68e+05
zip_98103	1.02e+05	1.6e+04	6.363	0.000	7.06e+04	1.33e+05
zip_98105	2.054e+05	2.07e+04	9.910	0.000	1.65e+05	2.46e+05
zip_98106	-1.87e+05	1.73e+04	-10.822	0.000	-2.21e+05	-1.53e+05
zip_98107	1.38e+05	1.79e+04	7.701	0.000	1.03e+05	1.73e+05
zip_98108	-1.636e+05	2.01e+04	-8.141	0.000	-2.03e+05	-1.24e+05
zip_98109	2.862e+05	3.04e+04	9.415	0.000	2.27e+05	3.46e+05
zip_98112	2.798e+05	2.28e+04	12.251	0.000	2.35e+05	3.25e+05
zip_98115	1.675e+05	1.62e+04	10.326	0.000	1.36e+05	1.99e+05
zip_98116	6.131e+04	1.91e+04	3.213	0.001	2.39e+04	9.87e+04
zip_98117	1.368e+05	1.61e+04	8.514	0.000	1.05e+05	1.68e+05
zip_98118	-1.082e+05	1.67e+04	-6.460	0.000	-1.41e+05	-7.54e+04
zip_98119	2.736e+05	2.31e+04	11.840	0.000	2.28e+05	3.19e+05
zip_98122	7.204e+04	1.83e+04	3.926	0.000	3.61e+04	1.08e+05
zip_98126	-9.968e+04	1.82e+04	-5.473	0.000	-1.35e+05	-6.4e+04
zip_98133	-8.325e+04	1.63e+04	-5.114	0.000	-1.15e+05	-5.13e+04
zip_98136	3.235e+04	2.03e+04	1.594	0.111	-7439.778	7.21e+04
zip_98144	1.527e+04	1.82e+04	0.838	0.402	-2.04e+04	5.1e+04
zip_98146	-1.776e+05	1.84e+04	-9.654	0.000	-2.14e+05	-1.42e+05
zip_98148	-2.482e+05	3.07e+04	-8.087	0.000	-3.08e+05	-1.88e+05
zip_98155	234.0021	1.73e+04	0.013	0.989	-3.38e+04	3.42e+04
zip_98166	-1.171e+05	1.96e+04	-5.959	0.000	-1.56e+05	-7.86e+04
zip_98168	-3.247e+05	1.9e+04	-17.075	0.000	-3.62e+05	-2.87e+05
zip_98177	1.104e+05	2.09e+04	5.290	0.000	6.95e+04	1.51e+05

zip_98178	-2.654e+05	1.89e+04	-14.070	0.000	-3.02e+05	-2.28e+05
zip_98188	-2.873e+05	2.29e+04	-12.526	0.000	-3.32e+05	-2.42e+05
zip_98198	-3.333e+05	1.8e+04	-18.481	0.000	-3.69e+05	-2.98e+05
zip_98199	1.993e+05	1.99e+04	10.000	0.000	1.6e+05	2.38e+05
zip_98223	5.636e+05	2.61e+05	2.158	0.031	5.18e+04	1.08e+06
zip_98224	-6.413e+05	1.85e+05	-3.459	0.001	-1e+06	-2.78e+05
zip_98251	-3.517e+05	1.51e+05	-2.326	0.020	-6.48e+05	-5.53e+04
zip_98271	-1.386e+05	1.31e+05	-1.057	0.290	-3.95e+05	1.18e+05
zip_98272	9.437e+04	1.31e+05	0.720	0.471	-1.62e+05	3.51e+05
zip_98288	-4.517e+05	7.39e+04	-6.112	0.000	-5.97e+05	-3.07e+05
zip_98296	-2.259e+05	2.61e+05	-0.865	0.387	-7.38e+05	2.86e+05
zip_98338	-1.709e+05	1.85e+05	-0.924	0.355	-5.33e+05	1.91e+05
zip_98354	-3.141e+05	5.58e+04	-5.627	0.000	-4.24e+05	-2.05e+05
zip_98372	8.184e+04	1.85e+05	0.443	0.658	-2.81e+05	4.44e+05
zip_98387	-1.471e+05	2.61e+05	-0.563	0.573	-6.59e+05	3.65e+05
zip_98422	-7.819e+05	2.61e+05	-2.990	0.003	-1.29e+06	-2.69e+05
zip_98663	-2.296e+05	1.85e+05	-1.242	0.214	-5.92e+05	1.33e+05

Omnibus:	813.863	**Durbin-Watson:**	1.976
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1741.193
Skew:	0.213	**Prob(JB):**	0.00
Kurtosis:	4.233	**Cond. No.**	4.80e+13

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 5.84e-23. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [120…
```
results_raw = pd.concat([cat_results.params,
                         cat_results.pvalues], axis=1)
results_raw.columns = ["coefficient", "p-value"]
results_raw
```

Out[120…

	coefficient	p-value
const	-8.888980e+15	5.236567e-01
waterfront_NO	-1.739380e+04	3.582115e-01
greenbelt_NO	-7.794155e+04	3.592784e-11
nuisance_NO	5.070366e+04	5.106491e-28
view_AVERAGE	8.888980e+15	5.236567e-01

	coefficient	p-value
...	...	...
zip_98354	-3.141230e+05	1.854056e-08
zip_98372	8.184236e+04	6.580517e-01
zip_98387	-1.471262e+05	5.731879e-01
zip_98422	-7.818506e+05	2.789448e-03
zip_98663	-2.296264e+05	2.142656e-01

108 rows × 2 columns

In [121…
```
results_raw = results_raw[results_raw["p-value"] < 0.05].sort_values(by="coeffic
results_raw.head(10)
```

Out[121…

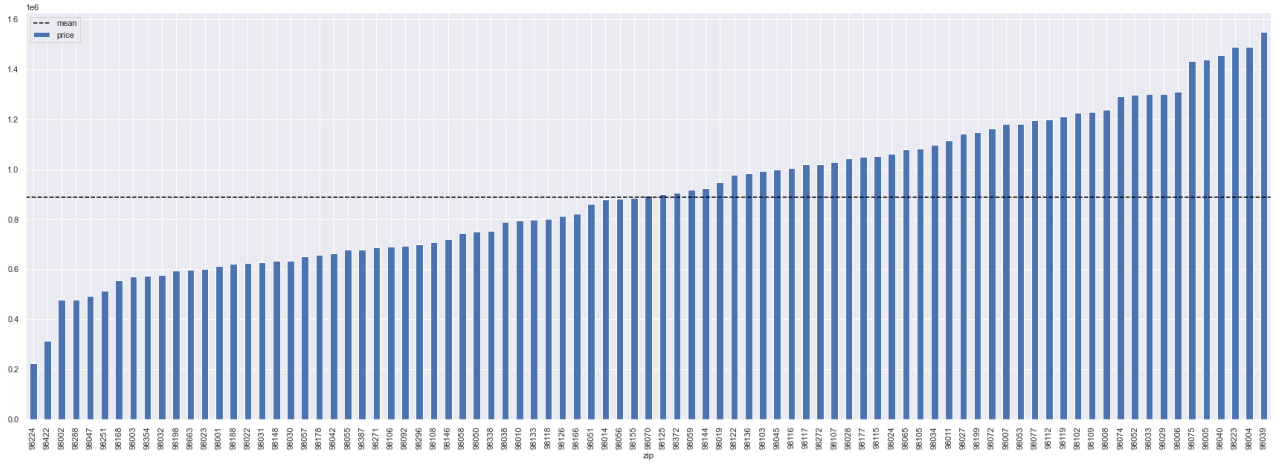
	coefficient	p-value
zip_98039	617725.861317	1.804285e-02
zip_98004	609997.803883	1.495181e-60
zip_98223	563642.588728	3.090925e-02
zip_98040	527692.536201	2.976854e-67
zip_98075	501125.182176	1.597430e-105
zip_98005	487829.756487	1.281562e-49
zip_98033	389486.294621	1.041370e-84
zip_98006	373792.953087	2.334747e-80
zip_98052	372575.242369	1.373233e-94
zip_98074	365621.129624	8.946656e-71

Our model is statistically significant. There's strong muticollinearity between our variables so moving forward we'll select our predictors more carefully. Many of our variables are statistically significant, with a significant coefficient value as well. As expected we see see variables like 'greenbelt_NO', 'waterfront_NO' make a big difference in 'price', as do many of our zips. Because we dropped the zip code in our model that was closest to the mean price, we can say that for each'zip' we could add or subtract the coefficient value from our intercept to find the gain or loss from the mean value of a house in that zipcode IF our intercept made sense. But it's a negative nuber, so at the very least we'd need to mean center this model to make sense of our predictors. Also, for such a high number of predictors we would have expected to see a larger adjusted r-squared. Our model is clearly missing at least one big component, sqft_living from the numeric predictors. Before we add that back in let's get a closer look at how our predictors interact with mean 'price', starting with our engineered column, 'zip'.

Clean 'zip'

In [35]:
```
fig, ax = plt.subplots(figsize=(30,10))
kc_clean.groupby("zip").mean().sort_values(by="price").plot.bar(y="price", ax=ax
```

```
ax.axhline(y=kc_clean["price"].mean(), label="mean", color="black", linestyle="-

ax.legend();
```



In [123…

```
#That's a bit hard to read, let's take a closer look
kc_clean.groupby("zip").mean().sort_values(by="price").head()
```

Out[123…

zip	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	flo
98224	4.017755e+09	224500.000000	2.000000	1.000000	1095.000000	62548.000000	1.250
98422	1.021039e+09	312750.000000	3.000000	2.000000	2480.000000	6615.000000	2.000
98002	4.809055e+09	476596.355301	3.286533	1.852436	1577.300860	8215.527221	1.282
98288	3.867125e+09	478176.923077	2.846154	1.461538	1589.230769	100857.230769	1.500
98047	5.137973e+09	494134.315789	3.315789	2.085526	1619.039474	10052.315789	1.500

A very clear mostly linear relationship between price and neighborhoods exists when grouped and sorted by mean price of home for "zip", moving forward it would be interesting to see this mapped out, with a mean price per neighborhood as a pop out on an interactive map. At the very least, we now have a way of listing neighborhoods where mean house prices are below the mean house price for the Seattle area, which will be helpful for our thrifty agent. We should note that the linear realtionship between zips and price only existsl when grouped by price, so it will not make for a good candidate for our final model. But we can use it to limit the kinds of houses we're interested in; Kings county covers roughly 10% of the area of Washington state, and many of these neighborhoods lay far outside the incorporated area of Seattle. Our client is only interested in the neighborhoods fully inside that boundry line, so let's drop all zipcodes from outside the Seattle city limits.

In [169…

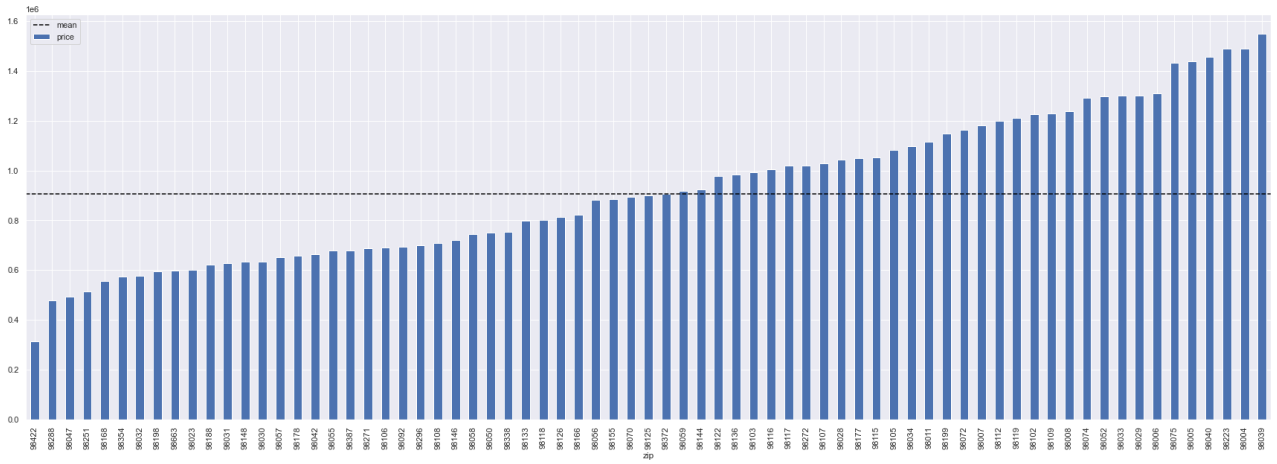
```
outer_limits= ['98077','98053', '98014', '98019', '98224', '98024', '98065',
'98027', '98045', '98038', '98022', '98051', '98010', '98288']
kc_clean = kc_clean[~df_clean['zip'].isin(outer_limits)]
```

```
<ipython-input-169-990717df7e55>:3: UserWarning: Boolean Series key will be rein
dexed to match DataFrame index.
  kc_clean = kc_clean[~df_clean['zip'].isin(outer_limits)]
```

In [162…

```
#check mean of zips again
fig, ax = plt.subplots(figsize=(30,10))
```

```
kc_clean.groupby("zip").mean().sort_values(by="price").plot.bar(y="price", ax=ax
ax.axhline(y=kc_clean["price"].mean(), label="mean", color="black", linestyle="-

ax.legend();
```

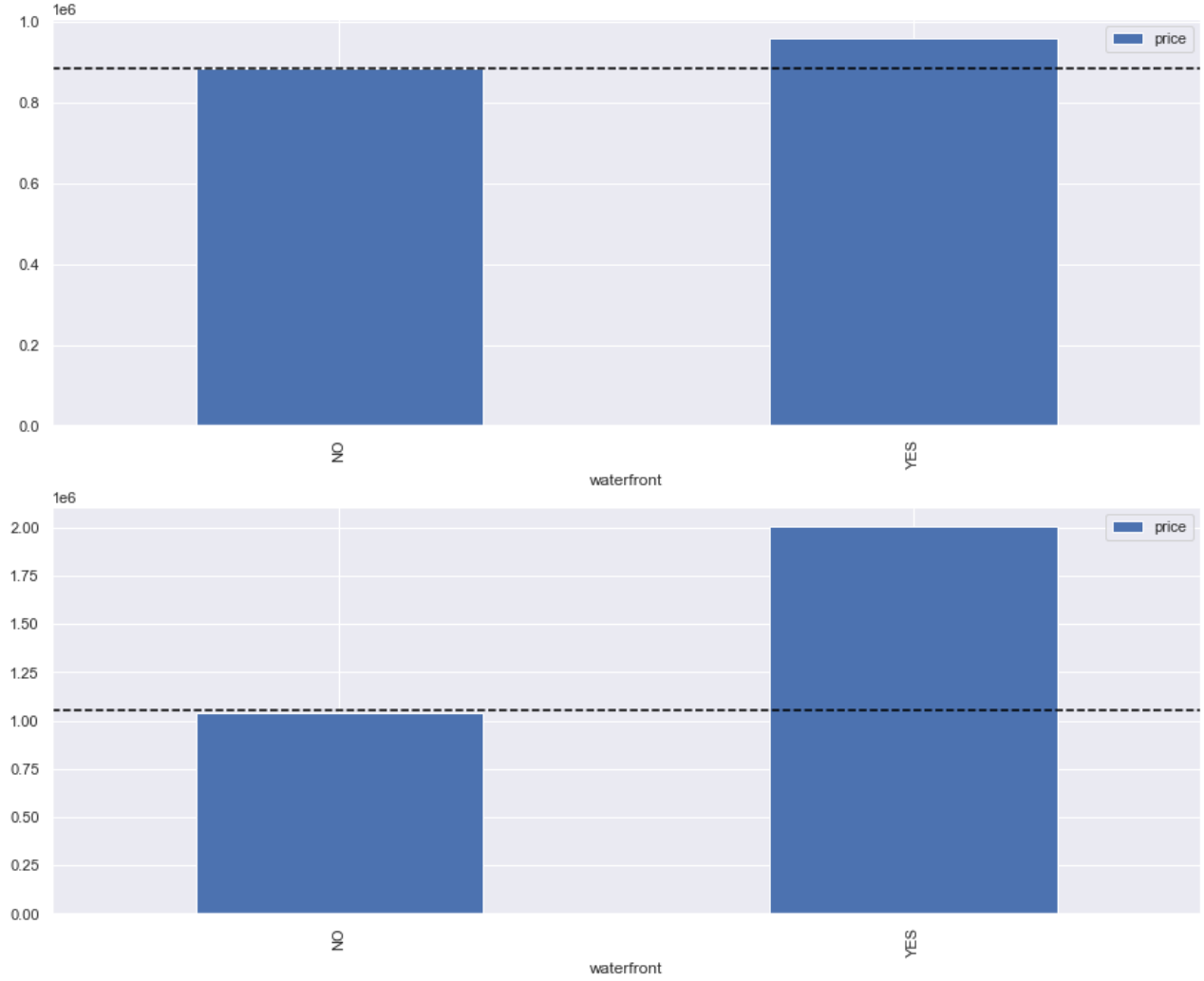


In [ ]:

Review Other Potential Predictors

In [124…

```
# Waterfront sns.barplot(data=df, x='waterfront', y='price')
fig, (ax1, ax2) = plt.subplots(2)
kc_clean.groupby("waterfront").mean().sort_values(by="price").plot.bar(y="price"
ax1.axhline(y=kc_clean["price"].mean(), label="mean", color="black", linestyle="
df_kc.groupby("waterfront").mean().sort_values(by="price").plot.bar(y="price", a
ax2.axhline(y=df_kc["price"].mean(), label="mean", color="black", linestyle="--"

ax.legend();
```



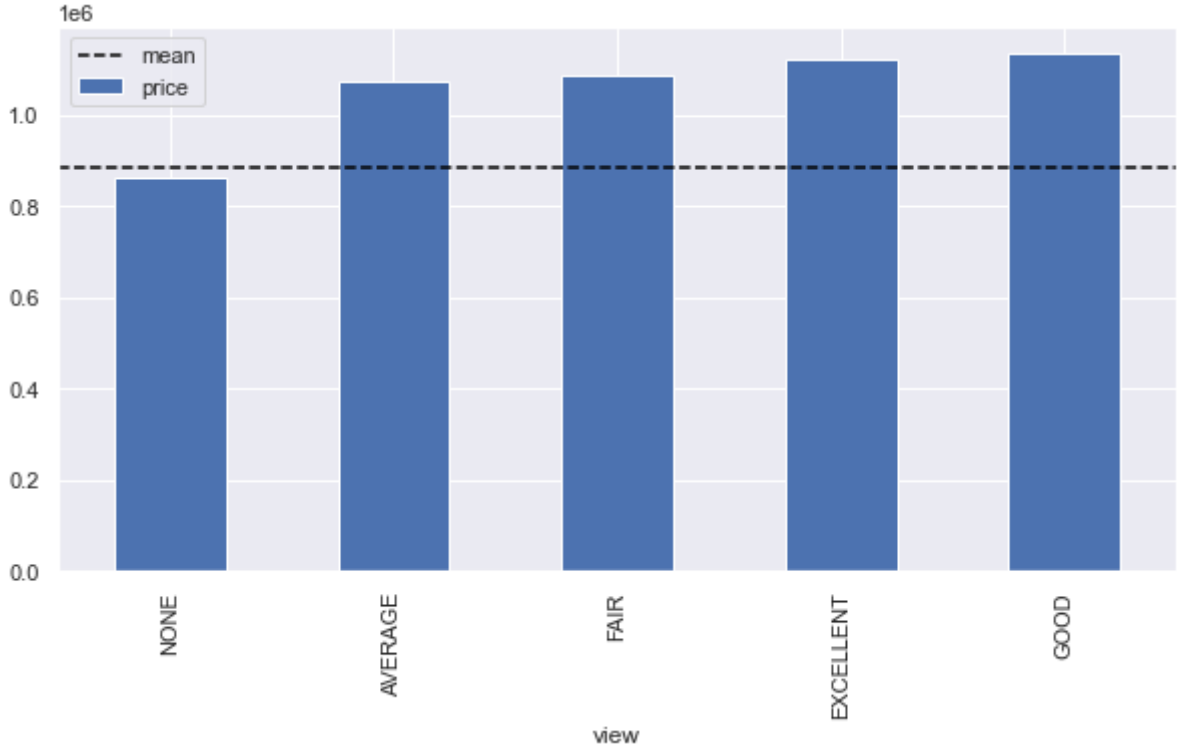
In [137…
```
print("Clean", kc_clean['waterfront'].value_counts())
print("Original", df_kc['waterfront'].value_counts())
```

```
Clean NO     21539
YES      179
Name: waterfront, dtype: int64
Original NO     29469
YES      486
Name: waterfront, dtype: int64
```

There's clearly a relationship between "price" and waterfront views in our unedited df. If we hadn't removed outliers we also would need to pay greater attention to the fact that there are very few houses that are on the waterfront, only 486 out of 29,555. Even now, with only 179 houses the mean price for a waterfront home is still above the regional average. While outlier values for things like sqft can be viewed easily with a scatterplot, it would have been easy to overlook the outlier qualities of a variable like 'waterfront' without looking more closely at the value counts. This might be a good candidate for an interaction term with sqft_living if we transform the values from 'NO' and 'YES' to 1 and 2 respectively. This way the numeric values can be multiplied by 'sqft_living' to better reflect the value of a waterfront home.

There's also likely strong colinearity between "zips" and "waterfront_YES", so as stated before, we should check for colinearity warnings if used in the same model.

In [125…
```
#view
fig, ax = plt.subplots(figsize=(10,5))
kc_clean.groupby("view").mean().sort_values(by="price").plot.bar(y="price", ax=a
ax.axhline(y=kc_clean["price"].mean(), label="mean", color="black", linestyle="-

ax.legend();
```



In [126…
```
kc_clean['view'].value_counts()
```

Out[126…
```
NONE         19153
AVERAGE       1189
GOOD           429
EXCELLENT      164
FAIR           137
Name: view, dtype: int64
```

Not looking great... "good" and excellent look about equal. We can see that houses with no views tend to have mean sales prices that fall below the average price of a house in our cleaned KC data, whereas houses with even an average view are above the mean. Looking at the value counts, there's again a disproportionate number of houses that do not have views (19661), much like our issue with waterfront. Let's revamp our "view' to boolean yes/no values

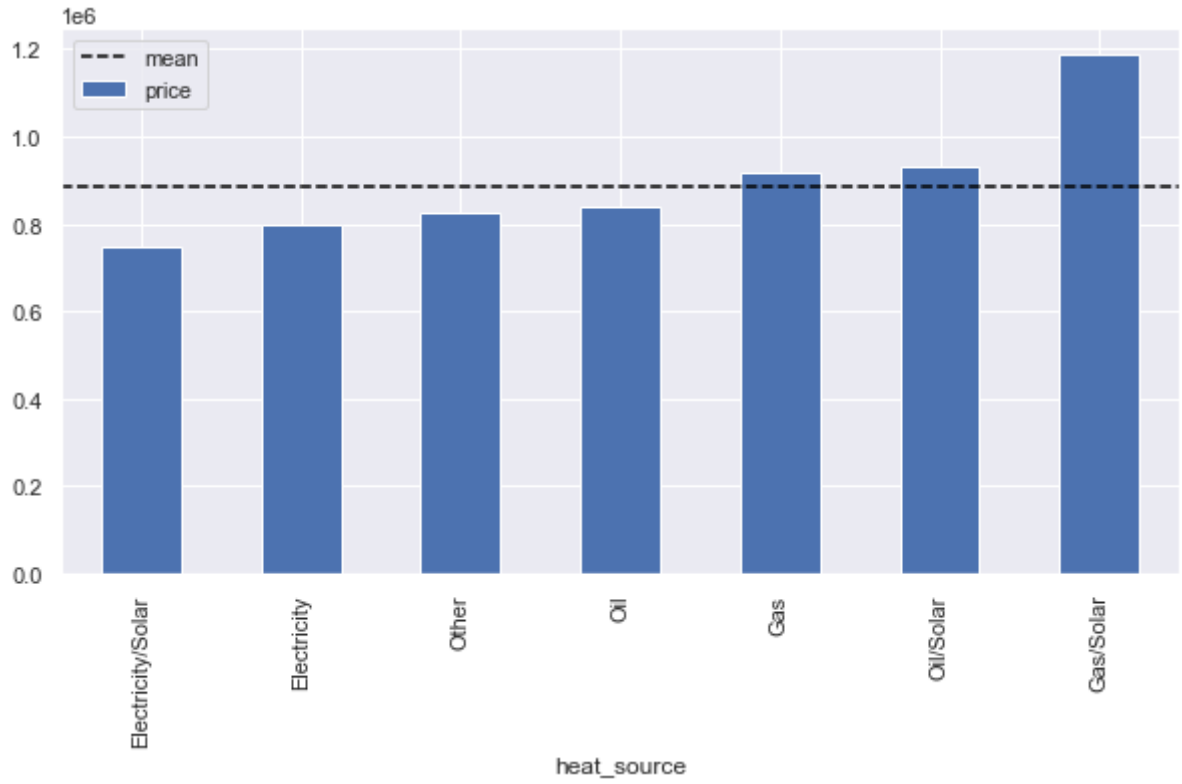
In [127…
```
view_dict = { 'NONE':'NO', "AVERAGE":"YES", "GOOD":"YES", "EXCELLENT": "YES", "F
kc_clean.view.replace(to_replace=view_dict,inplace=True)
```

In [128…
```
kc_clean['view'].value_counts()
```

Out[128…
```
NO     19153
YES     1919
Name: view, dtype: int64
```

In [129…
```
fig, ax = plt.subplots(figsize=(10,5))
kc_clean.groupby("heat_source").mean().sort_values(by="price").plot.bar(y="price
ax.axhline(y=kc_clean["price"].mean(), label="mean", color="black", linestyle="-
```

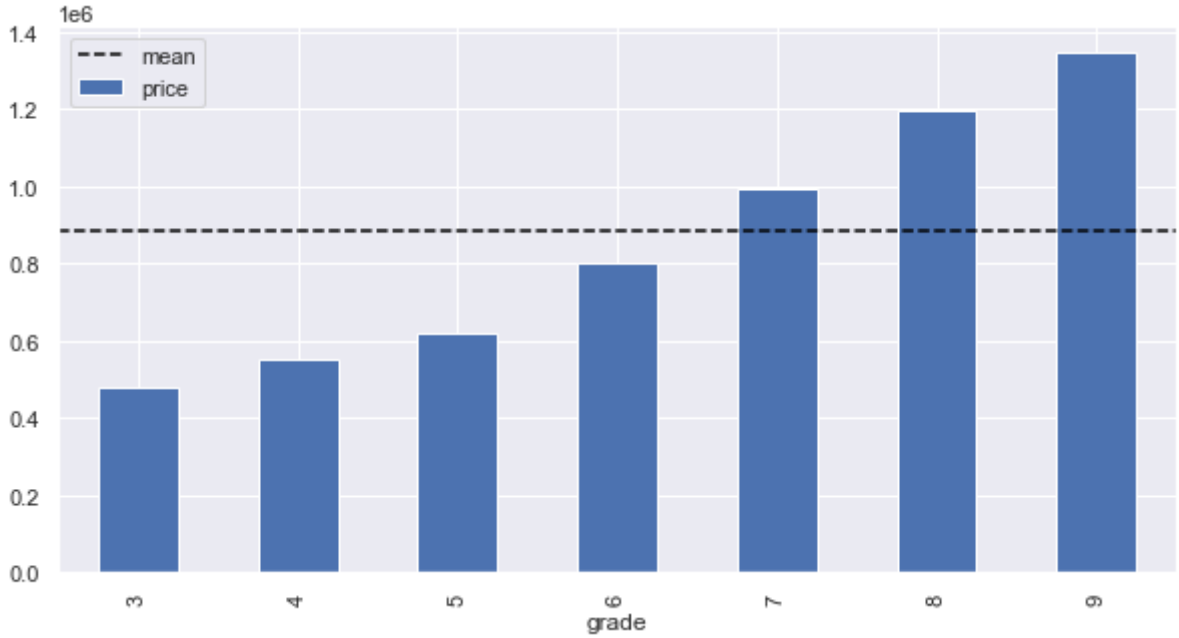
```
ax.legend();
```



I was curious if having solar power would add to the value of a house, but it looks like only houses that are equipped with gas and solar power experience an uptick in price compared to the median home price in Seattle. However, it could be that other factors are at play. Most large houses are unable to run exclusively on solar, and would need a backup for generating heat, hence the gas/solar mix. There's also a statistically significant relationship between price and sqft so eco-freindly solar only smaller homes would be more likely to have smaller square footage, depressing their sales value.

In [132…

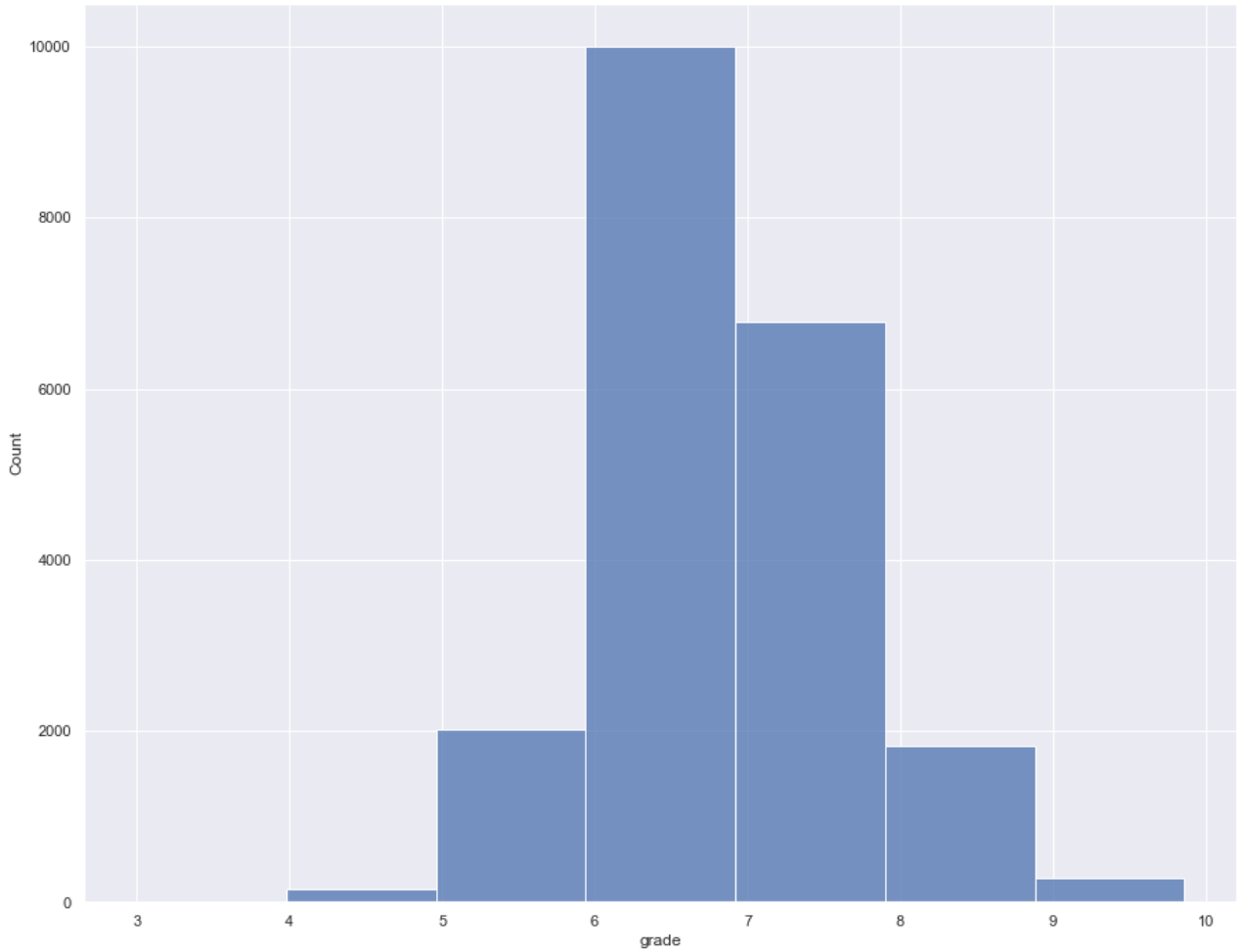
```
fig, ax = plt.subplots(figsize=(10,5))
kc_clean.groupby("grade").mean().sort_values(by="price").plot.bar(y="price", ax=
ax.axhline(y=kc_clean["price"].mean(), label="mean", color="black", linestyle="-

ax.legend();
```



In [130…
```
kc_clean['grade'].value_counts()
```

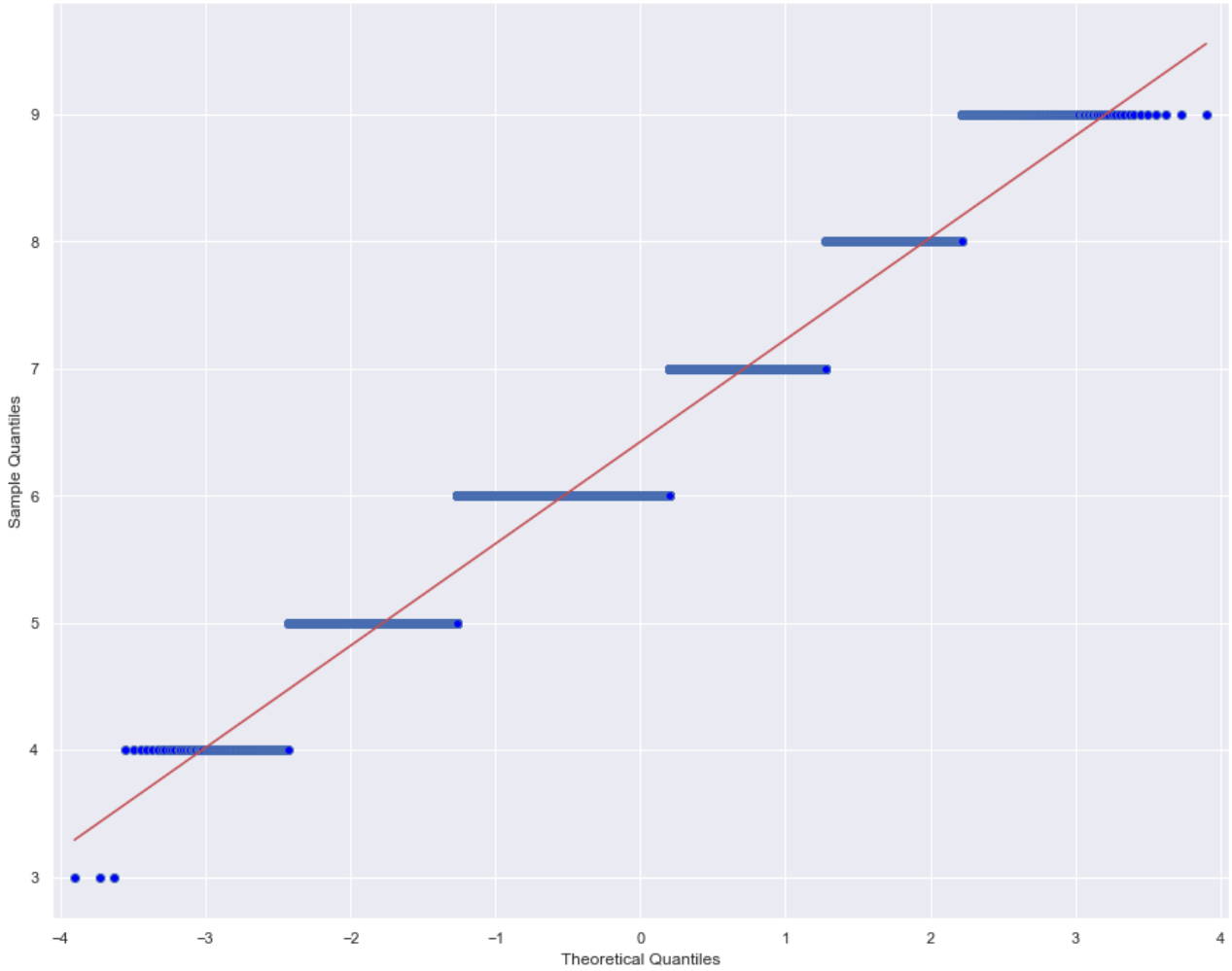
Out[130…
```
6    9999
7    6784
5    2022
8    1829
9     280
4     155
3       3
Name: grade, dtype: int64
```

In [131…
```
sns.histplot(data=kc_clean, x="grade", binwidth=.98);
```



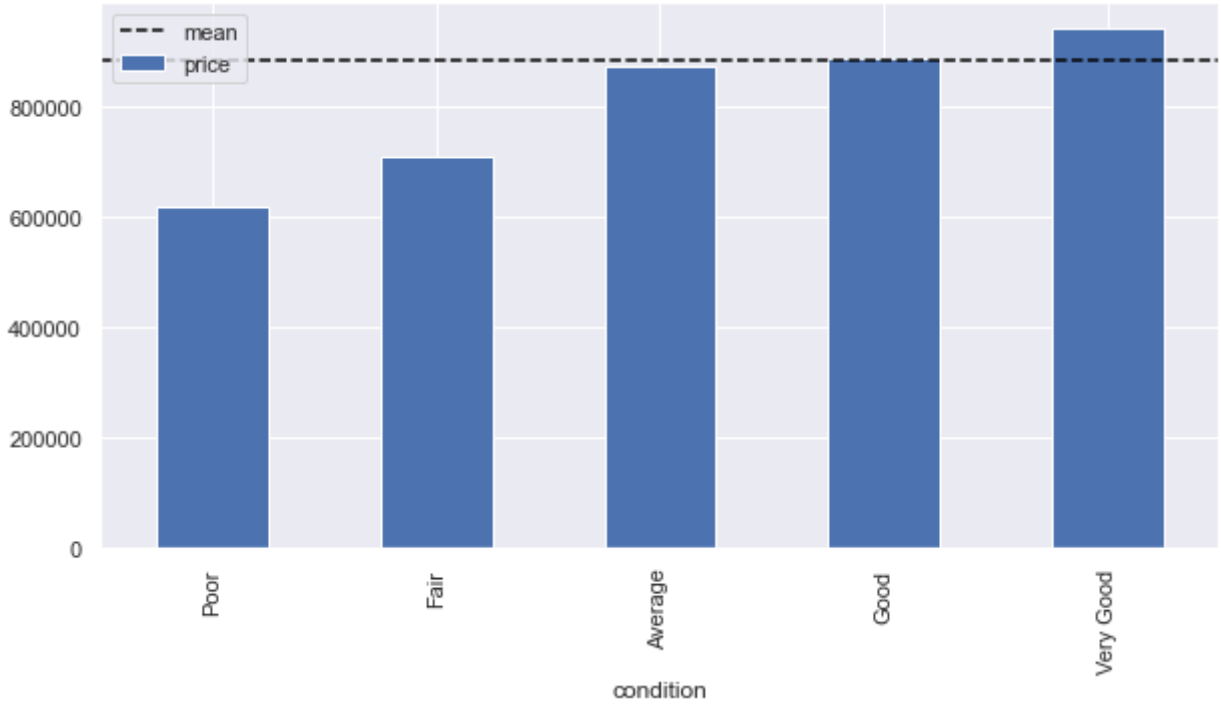
In [133…

```
sm.qqplot(kc_clean['grade'], line='r');
```



This is looking very promising. Not only does our cleaned grade have a linear relationship to price, it also has an almost normal distribution/residuals before any other kind of transformation.

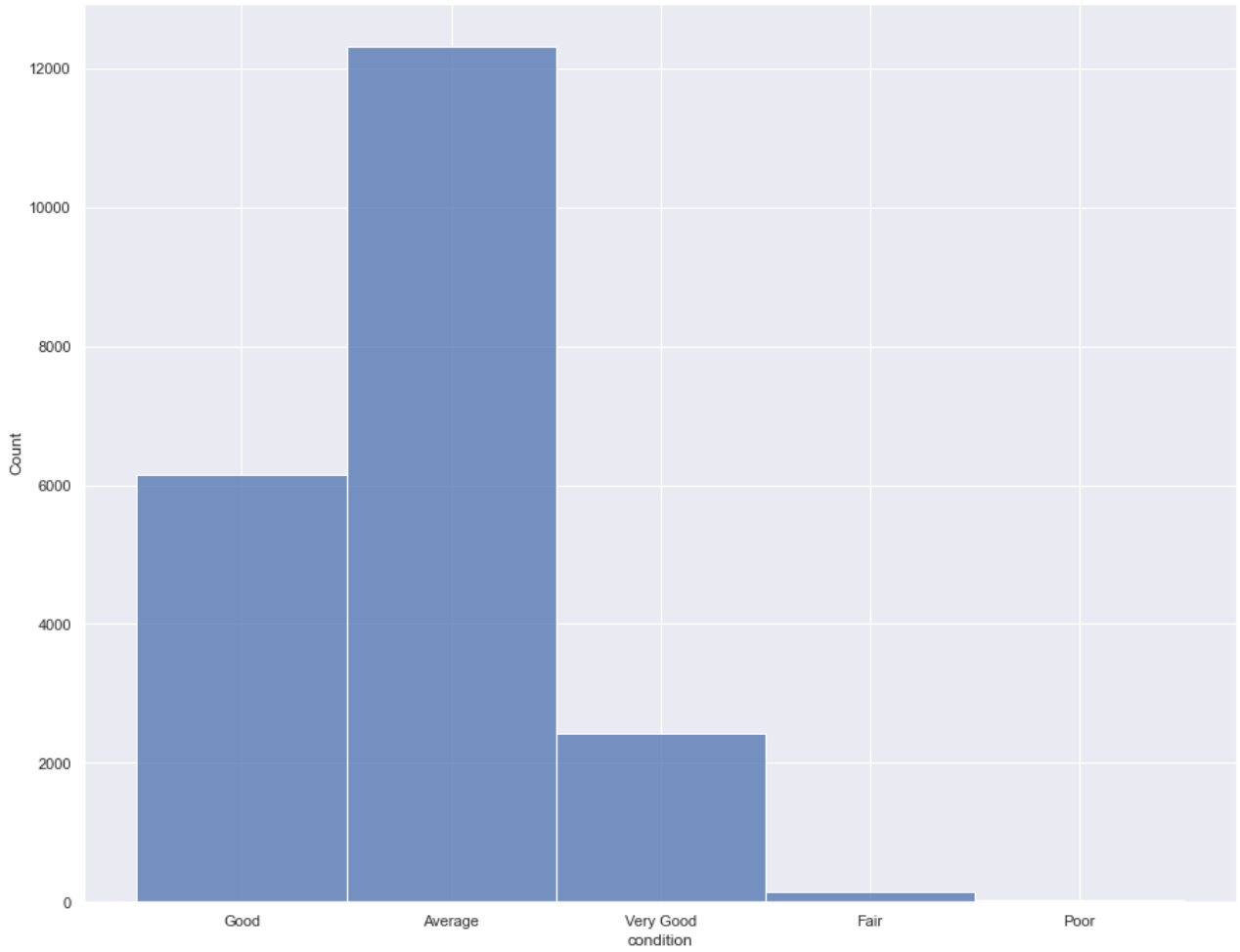
In [46]:
```
fig, ax = plt.subplots(figsize=(10,5))
kc_clean.groupby("condition").mean().sort_values(by="price").plot.bar(y="price",
ax.axhline(y=kc_clean["price"].mean(), label="mean", color="black", linestyle="-

ax.legend();
```



In [48]:
```
kc_clean['condition'].value_counts()
```

Out[48]:
```
Average      12312
Good          6152
Very Good     2423
Fair           151
Poor            34
Name: condition, dtype: int64
```

In [49]:
```
sns.histplot(data=kc_clean, x="condition", binwidth=1);
```



With the adjustments we made to grade, we would likely not need to use condition, which has an non-linear distribution anyway, making it more of a challange to use in a linear regression model.

Summary: Categorical Variables

Of the categorical variables, waterfront, zip (when grouped by mean), and grade have the most linear relationships with price. However, knowing that we'll want to exclude zip codes in at least one of our models to identify the change in our mean home price for each neighborhood, we'll use grade both to avoid potential co-linearity between 'zip' and 'waterfront', but also because we know our client isn't interested in houses with extra features or amenities that will bump up price.

Final Models

Putting it all together: Using SqFt, Grade, Zip, Waterfront, and Greenbelt to Locate Affordable Neighborhoods

Our first model will be unchanged by any kind of linear or logarithmic transformations.

If the model is correctly specified, then we should see an even distribution of the residuals in a scatter plot

In [184…
```
y_baseline = kc_clean["price"]
X_baseline = kc_clean[['grade', 'zip', 'sqft_living', 'waterfront', 'greenbelt',
X_baseline = pd.get_dummies(X_baseline, columns=['zip', 'grade', 'waterfront', '
X_baseline.drop(['zip_98039', 'grade_3', 'waterfront_NO', 'greenbelt_YES', 'view
```

In [185…
```
baseline_model = sm.OLS(y_baseline, sm.add_constant(X_baseline))
baseline_results = baseline_model.fit()
baseline_results.summary()
```

Out[185…

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.627
Model:	OLS	**Adj. R-squared:**	0.626
Method:	Least Squares	**F-statistic:**	417.1
Date:	Thu, 02 Mar 2023	**Prob (F-statistic):**	0.00
Time:	15:18:58	**Log-Likelihood:**	-2.6993e+05
No. Observations:	19675	**AIC:**	5.400e+05
Df Residuals:	19595	**BIC:**	5.406e+05
Df Model:	79		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.06e+06	2.7e+05	3.922	0.000	5.3e+05	1.59e+06
sqft_living	206.2699	3.010	68.526	0.000	200.370	212.170
zip_98004	-1.803e+05	2.22e+05	-0.812	0.417	-6.16e+05	2.55e+05
zip_98005	-3.013e+05	2.22e+05	-1.359	0.174	-7.36e+05	1.33e+05
zip_98006	-4.506e+05	2.21e+05	-2.043	0.041	-8.83e+05	-1.82e+04
zip_98007	-4.67e+05	2.21e+05	-2.112	0.035	-9e+05	-3.36e+04
zip_98008	-4.261e+05	2.21e+05	-1.932	0.053	-8.58e+05	6286.529
zip_98011	-6.165e+05	2.21e+05	-2.794	0.005	-1.05e+06	-1.84e+05
zip_98023	-1.092e+06	2.2e+05	-4.956	0.000	-1.52e+06	-6.6e+05
zip_98028	-6.917e+05	2.21e+05	-3.136	0.002	-1.12e+06	-2.59e+05
zip_98029	-4.801e+05	2.21e+05	-2.175	0.030	-9.13e+05	-4.75e+04
zip_98030	-1.048e+06	2.21e+05	-4.753	0.000	-1.48e+06	-6.16e+05
zip_98031	-1.036e+06	2.2e+05	-4.699	0.000	-1.47e+06	-6.04e+05
zip_98032	-1.046e+06	2.21e+05	-4.736	0.000	-1.48e+06	-6.13e+05
zip_98033	-3.497e+05	2.21e+05	-1.585	0.113	-7.82e+05	8.27e+04
zip_98034	-5.519e+05	2.2e+05	-2.504	0.012	-9.84e+05	-1.2e+05
zip_98040	-3.047e+05	2.21e+05	-1.376	0.169	-7.39e+05	1.29e+05
zip_98042	-1.042e+06	2.2e+05	-4.729	0.000	-1.47e+06	-6.1e+05
zip_98047	-1.065e+06	2.22e+05	-4.805	0.000	-1.5e+06	-6.31e+05

zip_98050	-7.7e+05	3.11e+05	-2.472	0.013	-1.38e+06	-1.59e+05
zip_98052	-4.098e+05	2.2e+05	-1.858	0.063	-8.42e+05	2.24e+04
zip_98055	-1.003e+06	2.21e+05	-4.545	0.000	-1.44e+06	-5.71e+05
zip_98056	-8.127e+05	2.2e+05	-3.686	0.000	-1.24e+06	-3.81e+05
zip_98057	-9.88e+05	2.21e+05	-4.465	0.000	-1.42e+06	-5.54e+05
zip_98058	-9.662e+05	2.2e+05	-4.384	0.000	-1.4e+06	-5.34e+05
zip_98059	-8.105e+05	2.2e+05	-3.677	0.000	-1.24e+06	-3.78e+05
zip_98070	-8.263e+05	2.21e+05	-3.739	0.000	-1.26e+06	-3.93e+05
zip_98072	-5.899e+05	2.21e+05	-2.673	0.008	-1.02e+06	-1.57e+05
zip_98074	-4.81e+05	2.21e+05	-2.180	0.029	-9.14e+05	-4.85e+04
zip_98075	-4.36e+05	2.21e+05	-1.975	0.048	-8.69e+05	-3187.795
zip_98092	-1.074e+06	2.2e+05	-4.874	0.000	-1.51e+06	-6.42e+05
zip_98102	-5.155e+05	2.21e+05	-2.330	0.020	-9.49e+05	-8.18e+04
zip_98103	-6.009e+05	2.2e+05	-2.727	0.006	-1.03e+06	-1.69e+05
zip_98105	-5.611e+05	2.21e+05	-2.543	0.011	-9.94e+05	-1.29e+05
zip_98106	-9.085e+05	2.2e+05	-4.121	0.000	-1.34e+06	-4.76e+05
zip_98107	-5.904e+05	2.2e+05	-2.678	0.007	-1.02e+06	-1.58e+05
zip_98108	-8.906e+05	2.21e+05	-4.037	0.000	-1.32e+06	-4.58e+05
zip_98109	-4.905e+05	2.21e+05	-2.215	0.027	-9.25e+05	-5.64e+04
zip_98112	-4.948e+05	2.21e+05	-2.241	0.025	-9.28e+05	-6.19e+04
zip_98115	-5.734e+05	2.2e+05	-2.602	0.009	-1.01e+06	-1.41e+05
zip_98116	-6.58e+05	2.21e+05	-2.983	0.003	-1.09e+06	-2.26e+05
zip_98117	-5.95e+05	2.2e+05	-2.700	0.007	-1.03e+06	-1.63e+05
zip_98118	-8.142e+05	2.2e+05	-3.694	0.000	-1.25e+06	-3.82e+05
zip_98119	-4.951e+05	2.21e+05	-2.242	0.025	-9.28e+05	-6.22e+04
zip_98122	-6.705e+05	2.21e+05	-3.040	0.002	-1.1e+06	-2.38e+05
zip_98125	-7.32e+05	2.2e+05	-3.320	0.001	-1.16e+06	-3e+05
zip_98126	-8.066e+05	2.21e+05	-3.658	0.000	-1.24e+06	-3.74e+05
zip_98133	-8.034e+05	2.2e+05	-3.645	0.000	-1.24e+06	-3.71e+05
zip_98136	-6.708e+05	2.21e+05	-3.040	0.002	-1.1e+06	-2.38e+05
zip_98144	-7.043e+05	2.21e+05	-3.194	0.001	-1.14e+06	-2.72e+05
zip_98146	-8.815e+05	2.21e+05	-3.997	0.000	-1.31e+06	-4.49e+05
zip_98148	-9.925e+05	2.21e+05	-4.481	0.000	-1.43e+06	-5.58e+05
zip_98155	-7.479e+05	2.2e+05	-3.392	0.001	-1.18e+06	-3.16e+05
zip_98166	-8.904e+05	2.21e+05	-4.036	0.000	-1.32e+06	-4.58e+05
zip_98168	-1.017e+06	2.21e+05	-4.610	0.000	-1.45e+06	-5.84e+05

zip_98177	-6.747e+05	2.21e+05	-3.057	0.002	-1.11e+06	-2.42e+05
zip_98178	-9.786e+05	2.21e+05	-4.437	0.000	-1.41e+06	-5.46e+05
zip_98188	-1.016e+06	2.21e+05	-4.601	0.000	-1.45e+06	-5.83e+05
zip_98198	-1.031e+06	2.2e+05	-4.675	0.000	-1.46e+06	-5.99e+05
zip_98199	-5.524e+05	2.21e+05	-2.504	0.012	-9.85e+05	-1.2e+05
zip_98223	-4.196e+05	3.11e+05	-1.347	0.178	-1.03e+06	1.91e+05
zip_98251	-9.518e+05	2.54e+05	-3.742	0.000	-1.45e+06	-4.53e+05
zip_98271	-8.367e+05	2.46e+05	-3.398	0.001	-1.32e+06	-3.54e+05
zip_98272	-9.661e+05	2.46e+05	-3.923	0.000	-1.45e+06	-4.83e+05
zip_98296	-1.033e+06	3.11e+05	-3.315	0.001	-1.64e+06	-4.22e+05
zip_98338	-8.814e+05	2.7e+05	-3.267	0.001	-1.41e+06	-3.53e+05
zip_98354	-1.045e+06	2.25e+05	-4.645	0.000	-1.49e+06	-6.04e+05
zip_98372	-7.294e+05	2.7e+05	-2.703	0.007	-1.26e+06	-2.01e+05
zip_98387	-7.958e+05	3.11e+05	-2.555	0.011	-1.41e+06	-1.85e+05
zip_98422	-1.598e+06	3.12e+05	-5.131	0.000	-2.21e+06	-9.88e+05
zip_98663	-8.893e+05	2.7e+05	-3.297	0.001	-1.42e+06	-3.61e+05
grade_4	1.243e+05	1.58e+05	0.789	0.430	-1.85e+05	4.33e+05
grade_5	1.509e+05	1.56e+05	0.965	0.335	-1.56e+05	4.57e+05
grade_6	1.767e+05	1.56e+05	1.130	0.258	-1.3e+05	4.83e+05
grade_7	2.352e+05	1.56e+05	1.504	0.133	-7.14e+04	5.42e+05
grade_8	3.255e+05	1.57e+05	2.080	0.038	1.87e+04	6.32e+05
grade_9	3.394e+05	1.57e+05	2.161	0.031	3.16e+04	6.47e+05
waterfront_YES	1.213e+05	1.98e+04	6.115	0.000	8.24e+04	1.6e+05
greenbelt_NO	-6834.2704	1.25e+04	-0.545	0.586	-3.14e+04	1.77e+04
view_YES	1.108e+05	5841.166	18.966	0.000	9.93e+04	1.22e+05

Omnibus:	1823.491	**Durbin-Watson:**	1.992
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	9448.999
Skew:	-0.302	**Prob(JB):**	0.00
Kurtosis:	6.341	**Cond. No.**	2.37e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.37e+06. This might indicate that there are strong multicollinearity or other numerical problems.

We can see that our model explains 62% of the variation in it by the r-squared value. The model is statistically significant overall as demonstrated by the Prob(F-statistic), as are all of our predictors. We can reject the null hypothesis that neighborhood, home size and house grade have no impact on price. We do see co-linearity warnings, so there may be colinearity between grade and sqft_living. We have what appears to be a mostly reasonable intercept of $1,060,000, although that's a large mean price it seems in line with what we could expect to pay for a home or apartment in a top tier metro area. We'll also want to check our MAE, or mean absolute error, to see by how much our model may be off. Our Durbin-Watson value is *JUST* between the desired 1-2 ANd the JB values also pretty reasonable given the scale of our y variable (homeprices). We will run a test to check for homoscedasticity when we examine our models limitations.

In [186…
```
# df with coeffiient and p-value.  Will use sorted ce's to find least expensive
results_df = pd.concat([baseline_results.params, baseline_results.pvalues], axis
results_df.columns = ["coefficient", "p-value"]
```

In [193…
```
# check to see how many predictors are statistically significant- 68 out of 80 i
results_df = results_df[results_df["p-value"] < 0.05].sort_values(by="coefficien
results_df.head(20)
```

Out[193…

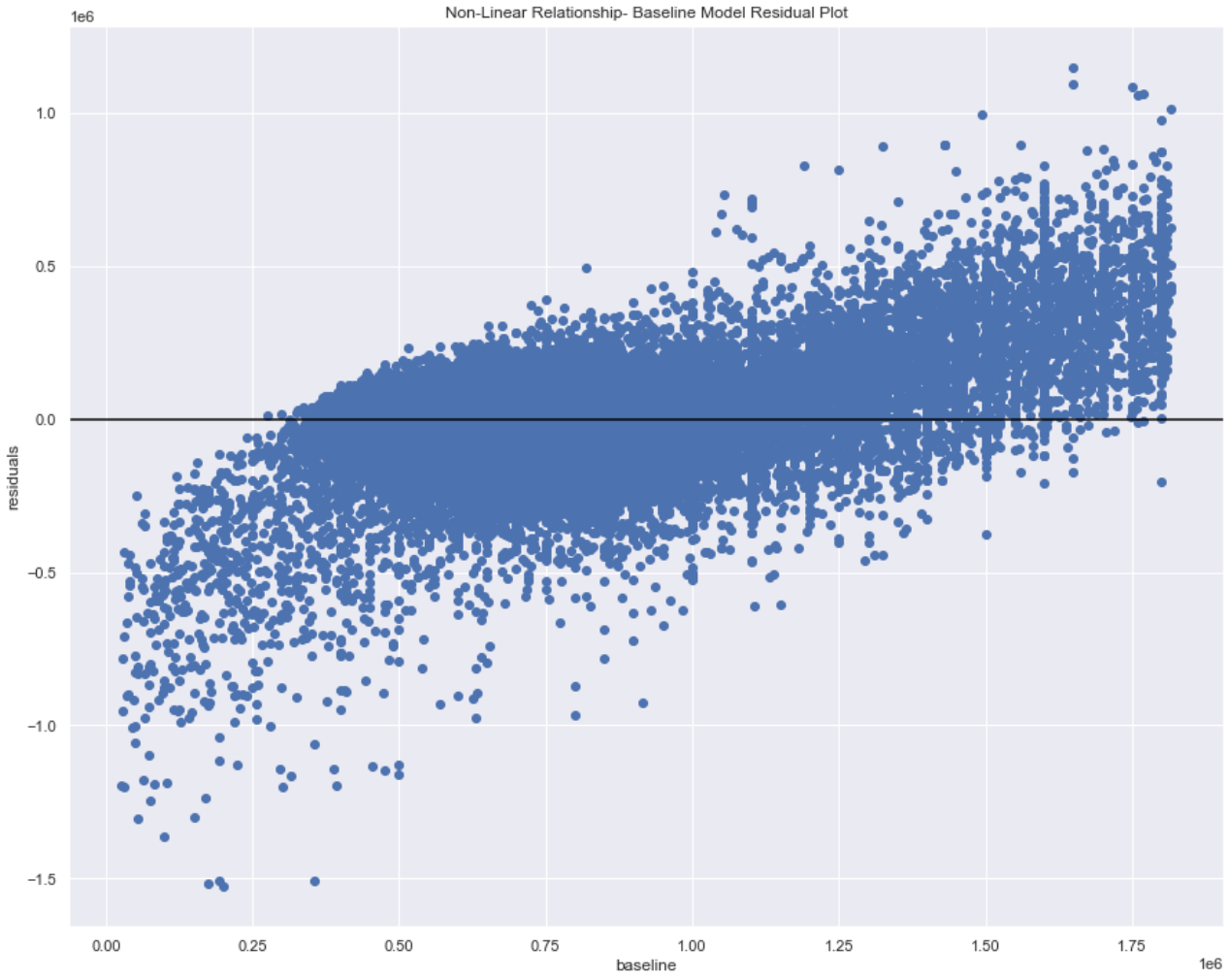
	coefficient	p-value
const	1.060390e+06	8.814231e-05
grade_9	3.394349e+05	3.068078e-02
grade_8	3.254701e+05	3.757440e-02
waterfront_YES	1.212807e+05	9.864557e-10
view_YES	1.107809e+05	1.693542e-79
sqft_living	2.062699e+02	0.000000e+00
zip_98075	-4.360423e+05	4.833663e-02
zip_98006	-4.506232e+05	4.109932e-02
zip_98007	-4.669924e+05	3.468002e-02
zip_98029	-4.800911e+05	2.960902e-02
zip_98074	-4.810025e+05	2.928127e-02
zip_98109	-4.904897e+05	2.679429e-02
zip_98112	-4.947610e+05	2.506777e-02
zip_98119	-4.950642e+05	2.499127e-02
zip_98102	-5.155091e+05	1.984005e-02
zip_98034	-5.518732e+05	1.229165e-02
zip_98199	-5.524464e+05	1.228656e-02
zip_98105	-5.610543e+05	1.101136e-02
zip_98115	-5.734142e+05	9.282111e-03
zip_98072	-5.899096e+05	7.513337e-03

In [154…
```
# what's our mean error going to be?
mae_baseline = baseline_results.resid.abs().sum() / len(y_baseline)
print(mae_baseline)
```

```
160282.30207422766
```

That's not too bad, in some of the ealier model included in the sandbox our MAE was much higher- but it's still a little high, even for real estate. Let's look at a plot of our residuals to help decide what our next steps should be (if any).

In [57]:
```
fig, ax = plt.subplots()

ax.scatter(y_baseline, baseline_results.resid)
ax.axhline(y=0, color="black")
ax.set_xlabel("baseline")
ax.set_ylabel("residuals")
ax.set_title("Non-Linear Relationship- Baseline Model Residual Plot");
```



Definitely not a perfectly normal distribution, although t- it seems like we could log transform our data to improve our residuals after we center our model. But first let's mean center our data to improve interpretability.

In [ ]:
```

```

2nd Model: Centering Data to Improve Interpretability

In [188…
```
y_centered = y_baseline.copy()
X_centered = X_baseline.copy()
```

In [189…
```
for col in X_centered.columns:
    X_centered[col] = X_centered[col] - X_centered[col].mean()

X_centered.describe()
```

Out[189…

	sqft_living	zip_98004	zip_98005	zip_98006	zip_98007	zip_98008	
count	1.967500e+04	1.967500e+04	1.967500e+04	1.967500e+04	1.967500e+04	1.967500e+04	1.9
mean	-6.286723e-14	-1.625130e-18	9.028497e-19	-5.597668e-18	-4.062824e-19	-7.945078e-18	7.
std	6.559587e+02	5.327565e-02	6.121407e-02	1.231416e-01	8.008466e-02	1.266876e-01	1.
min	-1.062141e+03	-2.846252e-03	-3.761118e-03	-1.540025e-02	-6.454892e-03	-1.631512e-02	
25%	-5.121408e+02	-2.846252e-03	-3.761118e-03	-1.540025e-02	-6.454892e-03	-1.631512e-02	
50%	-9.214079e+01	-2.846252e-03	-3.761118e-03	-1.540025e-02	-6.454892e-03	-1.631512e-02	
75%	4.178592e+02	-2.846252e-03	-3.761118e-03	-1.540025e-02	-6.454892e-03	-1.631512e-02	
max	2.112859e+03	9.971537e-01	9.962389e-01	9.845997e-01	9.935451e-01	9.836849e-01	9

8 rows × 79 columns

In [194…
```
X_centered_model = sm.OLS(y_centered, sm.add_constant(X_centered))
X_centered_results = X_centered_model.fit()
X_centered_results.summary()
```

Out[194…

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.627
Model:	OLS	**Adj. R-squared:**	0.626
Method:	Least Squares	**F-statistic:**	417.1
Date:	Thu, 02 Mar 2023	**Prob (F-statistic):**	0.00
Time:	15:36:51	**Log-Likelihood:**	-2.6993e+05
No. Observations:	19675	**AIC:**	5.400e+05
Df Residuals:	19595	**BIC:**	5.406e+05
Df Model:	79		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	9.068e+05	1570.002	577.567	0.000	9.04e+05	9.1e+05
sqft_living	206.2699	3.010	68.526	0.000	200.370	212.170

zip_98004	-1.803e+05	2.22e+05	-0.812	0.417	-6.16e+05	2.55e+05
zip_98005	-3.013e+05	2.22e+05	-1.359	0.174	-7.36e+05	1.33e+05
zip_98006	-4.506e+05	2.21e+05	-2.043	0.041	-8.83e+05	-1.82e+04
zip_98007	-4.67e+05	2.21e+05	-2.112	0.035	-9e+05	-3.36e+04
zip_98008	-4.261e+05	2.21e+05	-1.932	0.053	-8.58e+05	6286.529
zip_98011	-6.165e+05	2.21e+05	-2.794	0.005	-1.05e+06	-1.84e+05
zip_98023	-1.092e+06	2.2e+05	-4.956	0.000	-1.52e+06	-6.6e+05
zip_98028	-6.917e+05	2.21e+05	-3.136	0.002	-1.12e+06	-2.59e+05
zip_98029	-4.801e+05	2.21e+05	-2.175	0.030	-9.13e+05	-4.75e+04
zip_98030	-1.048e+06	2.21e+05	-4.753	0.000	-1.48e+06	-6.16e+05
zip_98031	-1.036e+06	2.2e+05	-4.699	0.000	-1.47e+06	-6.04e+05
zip_98032	-1.046e+06	2.21e+05	-4.736	0.000	-1.48e+06	-6.13e+05
zip_98033	-3.497e+05	2.21e+05	-1.585	0.113	-7.82e+05	8.27e+04
zip_98034	-5.519e+05	2.2e+05	-2.504	0.012	-9.84e+05	-1.2e+05
zip_98040	-3.047e+05	2.21e+05	-1.376	0.169	-7.39e+05	1.29e+05
zip_98042	-1.042e+06	2.2e+05	-4.729	0.000	-1.47e+06	-6.1e+05
zip_98047	-1.065e+06	2.22e+05	-4.805	0.000	-1.5e+06	-6.31e+05
zip_98050	-7.7e+05	3.11e+05	-2.472	0.013	-1.38e+06	-1.59e+05
zip_98052	-4.098e+05	2.2e+05	-1.858	0.063	-8.42e+05	2.24e+04
zip_98055	-1.003e+06	2.21e+05	-4.545	0.000	-1.44e+06	-5.71e+05
zip_98056	-8.127e+05	2.2e+05	-3.686	0.000	-1.24e+06	-3.81e+05
zip_98057	-9.88e+05	2.21e+05	-4.465	0.000	-1.42e+06	-5.54e+05
zip_98058	-9.662e+05	2.2e+05	-4.384	0.000	-1.4e+06	-5.34e+05
zip_98059	-8.105e+05	2.2e+05	-3.677	0.000	-1.24e+06	-3.78e+05
zip_98070	-8.263e+05	2.21e+05	-3.739	0.000	-1.26e+06	-3.93e+05
zip_98072	-5.899e+05	2.21e+05	-2.673	0.008	-1.02e+06	-1.57e+05
zip_98074	-4.81e+05	2.21e+05	-2.180	0.029	-9.14e+05	-4.85e+04
zip_98075	-4.36e+05	2.21e+05	-1.975	0.048	-8.69e+05	-3187.795
zip_98092	-1.074e+06	2.2e+05	-4.874	0.000	-1.51e+06	-6.42e+05
zip_98102	-5.155e+05	2.21e+05	-2.330	0.020	-9.49e+05	-8.18e+04
zip_98103	-6.009e+05	2.2e+05	-2.727	0.006	-1.03e+06	-1.69e+05
zip_98105	-5.611e+05	2.21e+05	-2.543	0.011	-9.94e+05	-1.29e+05
zip_98106	-9.085e+05	2.2e+05	-4.121	0.000	-1.34e+06	-4.76e+05
zip_98107	-5.904e+05	2.2e+05	-2.678	0.007	-1.02e+06	-1.58e+05
zip_98108	-8.906e+05	2.21e+05	-4.037	0.000	-1.32e+06	-4.58e+05
zip_98109	-4.905e+05	2.21e+05	-2.215	0.027	-9.25e+05	-5.64e+04

zip_98112	-4.948e+05	2.21e+05	-2.241	0.025	-9.28e+05	-6.19e+04
zip_98115	-5.734e+05	2.2e+05	-2.602	0.009	-1.01e+06	-1.41e+05
zip_98116	-6.58e+05	2.21e+05	-2.983	0.003	-1.09e+06	-2.26e+05
zip_98117	-5.95e+05	2.2e+05	-2.700	0.007	-1.03e+06	-1.63e+05
zip_98118	-8.142e+05	2.2e+05	-3.694	0.000	-1.25e+06	-3.82e+05
zip_98119	-4.951e+05	2.21e+05	-2.242	0.025	-9.28e+05	-6.22e+04
zip_98122	-6.705e+05	2.21e+05	-3.040	0.002	-1.1e+06	-2.38e+05
zip_98125	-7.32e+05	2.2e+05	-3.320	0.001	-1.16e+06	-3e+05
zip_98126	-8.066e+05	2.21e+05	-3.658	0.000	-1.24e+06	-3.74e+05
zip_98133	-8.034e+05	2.2e+05	-3.645	0.000	-1.24e+06	-3.71e+05
zip_98136	-6.708e+05	2.21e+05	-3.040	0.002	-1.1e+06	-2.38e+05
zip_98144	-7.043e+05	2.21e+05	-3.194	0.001	-1.14e+06	-2.72e+05
zip_98146	-8.815e+05	2.21e+05	-3.997	0.000	-1.31e+06	-4.49e+05
zip_98148	-9.925e+05	2.21e+05	-4.481	0.000	-1.43e+06	-5.58e+05
zip_98155	-7.479e+05	2.2e+05	-3.392	0.001	-1.18e+06	-3.16e+05
zip_98166	-8.904e+05	2.21e+05	-4.036	0.000	-1.32e+06	-4.58e+05
zip_98168	-1.017e+06	2.21e+05	-4.610	0.000	-1.45e+06	-5.84e+05
zip_98177	-6.747e+05	2.21e+05	-3.057	0.002	-1.11e+06	-2.42e+05
zip_98178	-9.786e+05	2.21e+05	-4.437	0.000	-1.41e+06	-5.46e+05
zip_98188	-1.016e+06	2.21e+05	-4.601	0.000	-1.45e+06	-5.83e+05
zip_98198	-1.031e+06	2.2e+05	-4.675	0.000	-1.46e+06	-5.99e+05
zip_98199	-5.524e+05	2.21e+05	-2.504	0.012	-9.85e+05	-1.2e+05
zip_98223	-4.196e+05	3.11e+05	-1.347	0.178	-1.03e+06	1.91e+05
zip_98251	-9.518e+05	2.54e+05	-3.742	0.000	-1.45e+06	-4.53e+05
zip_98271	-8.367e+05	2.46e+05	-3.398	0.001	-1.32e+06	-3.54e+05
zip_98272	-9.661e+05	2.46e+05	-3.923	0.000	-1.45e+06	-4.83e+05
zip_98296	-1.033e+06	3.11e+05	-3.315	0.001	-1.64e+06	-4.22e+05
zip_98338	-8.814e+05	2.7e+05	-3.267	0.001	-1.41e+06	-3.53e+05
zip_98354	-1.045e+06	2.25e+05	-4.645	0.000	-1.49e+06	-6.04e+05
zip_98372	-7.294e+05	2.7e+05	-2.703	0.007	-1.26e+06	-2.01e+05
zip_98387	-7.958e+05	3.11e+05	-2.555	0.011	-1.41e+06	-1.85e+05
zip_98422	-1.598e+06	3.12e+05	-5.131	0.000	-2.21e+06	-9.88e+05
zip_98663	-8.893e+05	2.7e+05	-3.297	0.001	-1.42e+06	-3.61e+05
grade_4	1.243e+05	1.58e+05	0.789	0.430	-1.85e+05	4.33e+05
grade_5	1.509e+05	1.56e+05	0.965	0.335	-1.56e+05	4.57e+05
grade_6	1.767e+05	1.56e+05	1.130	0.258	-1.3e+05	4.83e+05

grade_7	2.352e+05	1.56e+05	1.504	0.133	-7.14e+04	5.42e+05
grade_8	3.255e+05	1.57e+05	2.080	0.038	1.87e+04	6.32e+05
grade_9	3.394e+05	1.57e+05	2.161	0.031	3.16e+04	6.47e+05
waterfront_YES	1.213e+05	1.98e+04	6.115	0.000	8.24e+04	1.6e+05
greenbelt_NO	-6834.2704	1.25e+04	-0.545	0.586	-3.14e+04	1.77e+04
view_YES	1.108e+05	5841.166	18.966	0.000	9.93e+04	1.22e+05

Omnibus:	1823.491	**Durbin-Watson:**	1.992
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	9448.999
Skew:	-0.302	**Prob(JB):**	0.00
Kurtosis:	6.341	**Cond. No.**	7.65e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 7.65e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

Ok! This makes our coefficiants much more interpretable. Our intercept is now 884,000, or the mean price of a house in our slice of incorporated King County. We dropped our most expenzive zipcode, 'zip_98039' from the dataframe, so we can say that for any given zipcode the mean price is x percent smaller than our most expensive zip. We can also see that just by elimating waterfront houses, our client can expect to see a 189,900 drop in the mean price of a home, and that by avoiding greenbelt access they could expect to see a much smaller drop of $5656 from the mean. With that in mind, it might be worth it to target areas in more affordable neighborhoods near parks and other greenbelt amenities.

In [191…
```
results2_df = pd.concat([X_centered_results.params,
                         X_centered_results.pvalues], axis=1)
results2_df.columns = ["coefficient", "p-value"]
results2_df
```

Out[191…

	coefficient	p-value
const	906781.182770	0.000000e+00
sqft_living	206.269948	0.000000e+00
zip_98004	-180337.658486	4.170020e-01
zip_98005	-301261.242257	1.742368e-01
zip_98006	-450623.216063	4.109932e-02
...	...	...
grade_8	325470.073516	3.757440e-02
grade_9	339434.942903	3.068078e-02
waterfront_YES	121280.724585	9.864557e-10

	coefficient	p-value
greenbelt_NO	-6834.270450	5.857654e-01
view_YES	110780.929816	1.693542e-79

80 rows × 2 columns

In [197…
```
results2_df = results2_df[results2_df["p-value"] < 0.05].sort_values(by="coeffic
results2_df=results2_df.sort_values(by="p-value")

results2_df.head(20)
```

Out[197…

	coefficient	p-value
const	9.067812e+05	0.000000e+00
sqft_living	2.062699e+02	0.000000e+00
view_YES	1.107809e+05	1.693542e-79
waterfront_YES	1.212807e+05	9.864557e-10
zip_98422	-1.598316e+06	2.911759e-07
zip_98023	-1.092241e+06	7.265188e-07
zip_98092	-1.074362e+06	1.102244e-06
zip_98047	-1.065228e+06	1.555386e-06
zip_98030	-1.048233e+06	2.015259e-06
zip_98032	-1.045883e+06	2.200306e-06
zip_98042	-1.042023e+06	2.271644e-06
zip_98031	-1.035797e+06	2.634660e-06
zip_98198	-1.030732e+06	2.965219e-06
zip_98354	-1.044998e+06	3.418747e-06
zip_98168	-1.016750e+06	4.055512e-06
zip_98188	-1.016056e+06	4.225902e-06
zip_98055	-1.003319e+06	5.521760e-06
zip_98148	-9.924500e+05	7.476156e-06
zip_98057	-9.879725e+05	8.049493e-06
zip_98178	-9.785853e+05	9.173910e-06

In [316…
```
# Our mean error gshould remain the same
mae_baseline = X_centered_results.resid.abs().sum() / len(y_raw)
print(mae_baseline)
```

```
155937.0856085364
```

3rd Model: Log Transform 'price'

For this model we'll log transform sqft_living to deal with our linearity issues.

In [137…
```
X_log=X_baseline.copy()
X_log["log_sqft_living"] = np.log(X_log['sqft_living'])
X_log.drop('sqft_living', axis=1, inplace=True)
```

In [143…
```
y_log = np.log(kc_clean['price'])
X_log_model = sm.OLS(y_log, sm.add_constant(X_log))
X_log_results = X_log_model.fit()
X_log_results.summary()
```

Out[143…

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.540
Model:	OLS	**Adj. R-squared:**	0.538
Method:	Least Squares	**F-statistic:**	299.9
Date:	Thu, 02 Mar 2023	**Prob (F-statistic):**	0.00
Time:	13:48:29	**Log-Likelihood:**	-5039.5
No. Observations:	21072	**AIC:**	1.024e+04
Df Residuals:	20989	**BIC:**	1.091e+04
Df Model:	82		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	11.1387	0.361	30.892	0.000	10.432	11.845
zip_98001	-1.1067	0.308	-3.591	0.000	-1.711	-0.503
zip_98002	-1.1902	0.308	-3.859	0.000	-1.795	-0.586
zip_98003	-1.1422	0.308	-3.704	0.000	-1.746	-0.538
zip_98004	-0.2185	0.311	-0.703	0.482	-0.828	0.390
zip_98005	-0.3000	0.310	-0.968	0.333	-0.908	0.308
zip_98006	-0.4291	0.308	-1.391	0.164	-1.034	0.176
zip_98007	-0.4097	0.309	-1.325	0.185	-1.016	0.196
zip_98008	-0.3643	0.308	-1.181	0.238	-0.969	0.240
zip_98011	-0.5692	0.309	-1.844	0.065	-1.174	0.036
zip_98023	-1.1269	0.308	-3.656	0.000	-1.731	-0.523
zip_98028	-0.6488	0.308	-2.104	0.035	-1.253	-0.044
zip_98029	-0.4593	0.309	-1.488	0.137	-1.064	0.146
zip_98030	-1.0776	0.308	-3.494	0.000	-1.682	-0.473
zip_98031	-1.0701	0.308	-3.471	0.001	-1.674	-0.466
zip_98032	-1.0986	0.309	-3.557	0.000	-1.704	-0.493
zip_98033	-0.3150	0.308	-1.021	0.307	-0.920	0.290
zip_98034	-0.4907	0.308	-1.592	0.111	-1.095	0.113
zip_98040	-0.3082	0.310	-0.995	0.320	-0.915	0.299

zip_98042	-1.0537	0.308	-3.420	0.001	-1.658	-0.450
zip_98047	-1.1618	0.310	-3.748	0.000	-1.769	-0.554
zip_98050	-0.6616	0.436	-1.519	0.129	-1.515	0.192
zip_98052	-0.3764	0.308	-1.221	0.222	-0.981	0.228
zip_98055	-1.0044	0.309	-3.254	0.001	-1.609	-0.399
zip_98056	-0.8004	0.308	-2.596	0.009	-1.405	-0.196
zip_98057	-0.9573	0.309	-3.094	0.002	-1.564	-0.351
zip_98058	-0.9494	0.308	-3.080	0.002	-1.553	-0.345
zip_98059	-0.7703	0.308	-2.499	0.012	-1.375	-0.166
zip_98070	-0.7734	0.309	-2.503	0.012	-1.379	-0.168
zip_98072	-0.5631	0.309	-1.825	0.068	-1.168	0.042
zip_98074	-0.4571	0.309	-1.481	0.139	-1.062	0.148
zip_98075	-0.4227	0.309	-1.369	0.171	-1.028	0.183
zip_98092	-1.0774	0.308	-3.495	0.000	-1.682	-0.473
zip_98102	-0.4488	0.309	-1.450	0.147	-1.055	0.158
zip_98103	-0.5210	0.308	-1.691	0.091	-1.125	0.083
zip_98105	-0.4931	0.309	-1.598	0.110	-1.098	0.112
zip_98106	-0.8870	0.308	-2.877	0.004	-1.491	-0.283
zip_98107	-0.4978	0.308	-1.614	0.106	-1.102	0.107
zip_98108	-0.8826	0.309	-2.861	0.004	-1.487	-0.278
zip_98109	-0.4289	0.310	-1.385	0.166	-1.036	0.178
zip_98112	-0.4422	0.309	-1.432	0.152	-1.047	0.163
zip_98115	-0.5106	0.308	-1.657	0.098	-1.115	0.093
zip_98116	-0.5719	0.308	-1.854	0.064	-1.176	0.033
zip_98117	-0.5091	0.308	-1.652	0.099	-1.113	0.095
zip_98118	-0.7659	0.308	-2.485	0.013	-1.370	-0.162
zip_98119	-0.4309	0.309	-1.395	0.163	-1.036	0.174
zip_98122	-0.6095	0.308	-1.976	0.048	-1.214	-0.005
zip_98125	-0.6567	0.308	-2.130	0.033	-1.261	-0.052
zip_98126	-0.7253	0.308	-2.352	0.019	-1.330	-0.121
zip_98133	-0.7460	0.308	-2.420	0.016	-1.350	-0.142
zip_98136	-0.5802	0.309	-1.880	0.060	-1.185	0.025
zip_98144	-0.6334	0.308	-2.054	0.040	-1.238	-0.029
zip_98146	-0.8591	0.308	-2.786	0.005	-1.464	-0.255
zip_98148	-1.0076	0.310	-3.253	0.001	-1.615	-0.400
zip_98155	-0.6940	0.308	-2.251	0.024	-1.298	-0.090

zip_98166	-0.8608	0.308	-2.790	0.005	-1.465	-0.256
zip_98168	-1.0753	0.308	-3.486	0.000	-1.680	-0.471
zip_98177	-0.6160	0.309	-1.996	0.046	-1.221	-0.011
zip_98178	-0.9662	0.308	-3.133	0.002	-1.571	-0.362
zip_98188	-1.0530	0.309	-3.410	0.001	-1.658	-0.448
zip_98198	-1.0752	0.308	-3.487	0.000	-1.680	-0.471
zip_98199	-0.4701	0.309	-1.524	0.128	-1.075	0.135
zip_98223	-0.4418	0.436	-1.014	0.310	-1.296	0.412
zip_98251	-0.9506	0.356	-2.673	0.008	-1.648	-0.253
zip_98271	-0.7760	0.344	-2.254	0.024	-1.451	-0.101
zip_98272	-0.8601	0.344	-2.498	0.013	-1.535	-0.185
zip_98288	-1.3573	0.320	-4.246	0.000	-1.984	-0.731
zip_98296	-1.0166	0.436	-2.334	0.020	-1.870	-0.163
zip_98338	-0.8326	0.377	-2.207	0.027	-1.572	-0.093
zip_98354	-1.0967	0.315	-3.486	0.000	-1.713	-0.480
zip_98372	-0.6221	0.377	-1.649	0.099	-1.362	0.117
zip_98387	-0.7068	0.436	-1.623	0.105	-1.560	0.147
zip_98422	-1.8852	0.436	-4.328	0.000	-2.739	-1.032
zip_98663	-0.8172	0.377	-2.166	0.030	-1.557	-0.078
grade_4	-0.0324	0.180	-0.180	0.857	-0.385	0.320
grade_5	0.0383	0.178	0.215	0.830	-0.311	0.388
grade_6	0.0969	0.178	0.543	0.587	-0.253	0.446
grade_7	0.1731	0.178	0.970	0.332	-0.177	0.523
grade_8	0.2747	0.179	1.538	0.124	-0.075	0.625
grade_9	0.2739	0.179	1.526	0.127	-0.078	0.626
waterfront_NO	-0.2204	0.025	-8.944	0.000	-0.269	-0.172
greenbelt_NO	-0.0186	0.017	-1.069	0.285	-0.053	0.015
log_sqft_living	0.4434	0.008	57.647	0.000	0.428	0.458

Omnibus:	15844.251	**Durbin-Watson:**	1.992
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	493307.279
Skew:	-3.320	**Prob(JB):**	0.00
Kurtosis:	25.754	**Cond. No.**	9.65e+03

Notes:

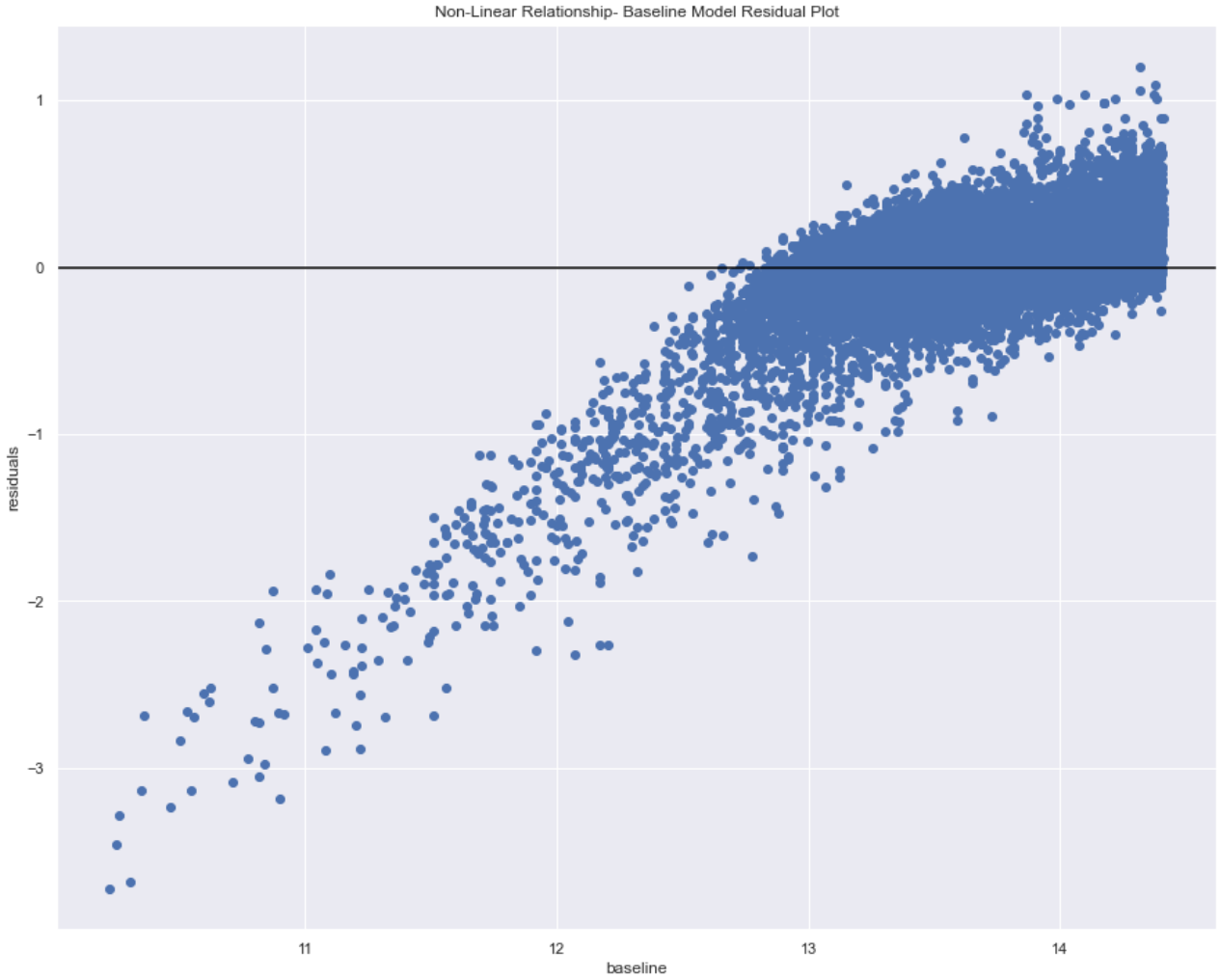
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 9.65e+03. This might indicate that there are strong multicollinearity or other numerical problems.

Now that we've log transformed sqft_living and price we'd have to desribe it's coefficients using percentages rather than dollar increase. In other words, for every 1% increase in price we'd see our predictor increase by the percentage of it's coefficient value. That's pretty hard to follow when it comes to zipcodes, although it makes slightly more sense when talking about home grade or sqft_living. Before we take time to describe the changes to our most important predictors(if any) we want to check to see if our residuals are better distributed- if not we'll scrap this model and use our Baseline or Centered models to interpret our coefficients.

In [142…

```
fig, ax = plt.subplots()

ax.scatter(y_log, X_log_results.resid)
ax.axhline(y=0, color="black")
ax.set_xlabel("baseline")
ax.set_ylabel("residuals")
ax.set_title("Non-Linear Relationship- Baseline Model Residual Plot");
```



Not only is our adjusted r-squared less, but out model residuals are concentrated in a sweep to the upper left. This model is not accurate enough for use, and our baseline and centered models would be better for describing coefficient values for our client anyway.

In [140…
```
mae_baseline = baseline_results.resid.abs().sum() / len(y_log)
print("centered MAE:", mae_baseline)
mae_log_centered = X_log_results.resid.abs().sum() / len(y_log)
print("Log MAE:", mae_log_centered)
```

```
centered MAE: 155937.08560853227
Log MAE: 0.18715337009181274
```

At first glance, it may seem as if we've radically improved our MAE, but we need to remember that a log transformation changes the units we ae using from dollars to a decimal/percentage. This makes it a little challenging to compare the two. But a cursory glance tells us that the log transformed model is roughly 20% off- much higher than the approximately 10% of our Baseline and Centered models.

Limitations

Checking L.I.N.E.

- Linear relationship between the response variable (y) and predictor (x).
- Data is independent: 1. Data avoids collinearity (features can't be used to predict each other) 2. Data is not autocorrelated (correlated with itself)
- Model residuals are normally distributed
- Homoscedasticity- data has an equal variance

Linearity

We can check our linearity of our models by using a scatterplot of of the response variable (y) and the predictor (x). We should see a constant change in y by a one unit change in x, as we did in our initial conversion of 'sqft_living' and 'grade' when we removed outliers.

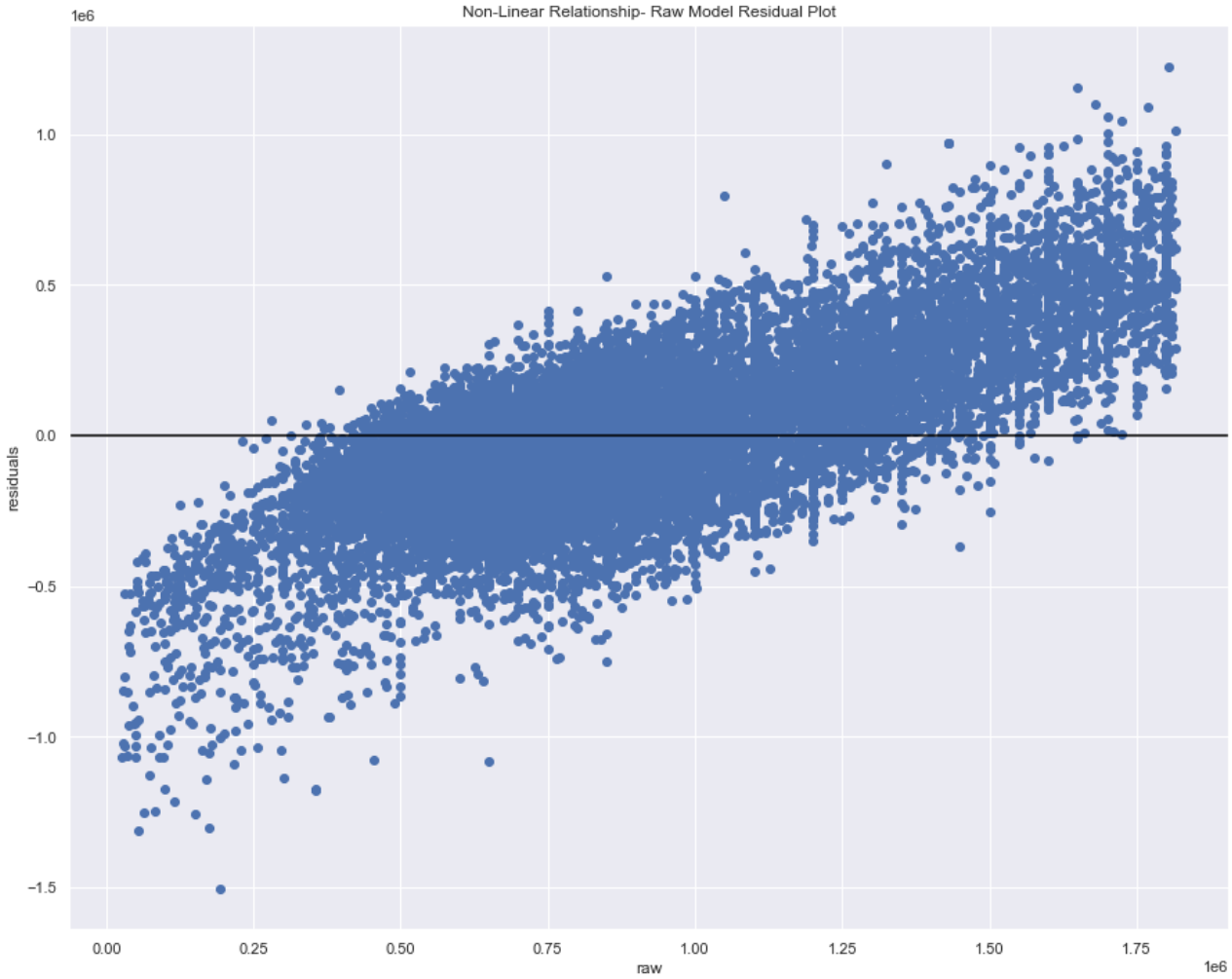
Independence of Features and Errors

If we wanted to check that assumtion we could run a pairplot to check for coliearity between our response variable and predictors. Our model warnings indicated that some colinearity existed.

Autocorrelation, where a variable becomes correlated with itself will be harder to check for

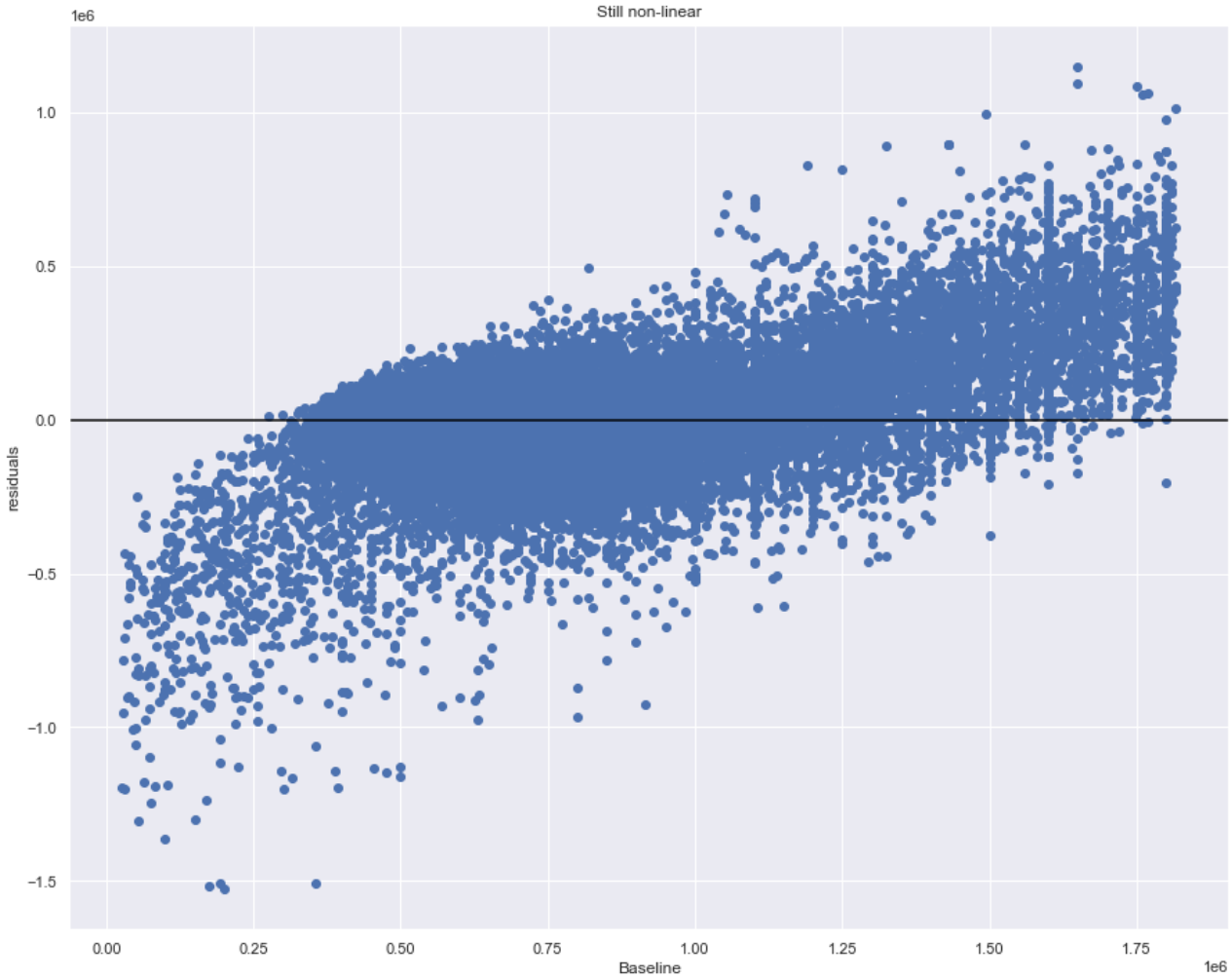
Normally Distributed Residuals

In [345…
```
# Analyze coefficiants and p values here
fig, ax = plt.subplots()

ax.scatter(y_raw_cat, cat_results.resid)
ax.axhline(y=0, color="black")
ax.set_xlabel("raw")
ax.set_ylabel("residuals")
ax.set_title("Non-Linear Relationship- Raw Model Residual Plot");
```



In [144…

```
fig, ax = plt.subplots()

ax.scatter(y_baseline, baseline_results.resid)
ax.axhline(y=0, color="black")
ax.set_xlabel("Baseline")
ax.set_ylabel("residuals")
ax.set_title("Still non-linear");
```



If our data were normally distributed we would see our residuals scattered across the plot rather than clumped together like we do here. But our final model is a marked improvement of our all categorical model.

Equal Variance/Homoscedasticity

Let's prove that there's a non-linear relationship between the variables in our model and price statistically as well by using a rainbow test. In a rainbow test, even if the true relationship is non-linear, a good linear fit can be achieved on a subsample in the “middle” of the data, in the arc of the "rainbow". The null hypothesis is the fit of the model using full sample is the same as using a central subset. The null hypothesis is rejected whenever the overall fit is significantly worse than the fit for the subsample. A rainbow test returns the f-statistic and p-value, and unlike the way we usually interpret p-values (a low score is good) in this instance, a low score would indicate a non-linear relationship.

In [351…

```
import statsmodels.api as sms


print("Baseline results:", sms.stats.diagnostic.linear_rainbow(baseline_results))
print("Log results:", sms.stats.diagnostic.linear_rainbow(X_log_results))
```

```
Baseline results: (0.9430075587483235, 0.9986754129300249)
Log results: (0.9445674716558311, 0.9982560609610807)
```

Recommendations

Neighborhoods

Using our final model we have identified which neighborhoods we would be able to locate housing that would be in the lowest mean price by zipcode. We can also see that we'll want to look for houses that are in low to average grade, as we see prices jumping over the median once we've crossed the average threshhold. Additionally, while we'll want to avoid waterfront homes because of the significant jump in price (about a million dollars), our real estate agent might want to target homes that are located near greenbelts or parks, which have a significantly lower jump in price- around 5,500.

Next Steps

Affordable housing is a challange to find in the best of times, and often neighborhoods with lower housing prices are located nearby inherant health dangers like freeways or large industrial areas, or lack the kinds of amennities that inhance quality of life, like greenspaces and grocery stores. Our next steps would be overlay a map of such amenities or dangers over the neighborhoods we've identified as being our most affordable, to analyze the additional benefits and risks of those neighborhoods and to allow our client to have a fuller understanding of its benefits or disadvantages.

In [ ]: