King County Housing Data Project

Overview

A company that buys houses to flip and resell is interested in finding out what pre-existine features of houses are likely to lead to a higher sale price. Since they plan on "flipping" the house, or adding their own renovations, they aren't as interested in details such as the overall condition of the house and are more interested in things such as location, how big of a lot the house is built on, etc.

Business Understanding

The features of the data from a housing dataset that I will be looking at, and comparing to the sale price of the houses, include number of bedrooms, number of bathrooms, square footage of the living area, square footage of the lot, number of floors, whether the house is on a waterfront, whether the house is adjacent to a green belt, whether the house has traffic noise or other nuisances, and the quality of the view of the house. After performing exploratory data analysis and determining which of these factors seem to relate to sale price, I will narrow down my efforts to determine which of those factors are the best predictors of sale price.

Data Understanding

I begin by importing the necessary modules and the dataset I will be using, which includes housing data for King County.

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import statsmodels.api as sm
import numpy as np
import math
import matplotlib.cm as cm
%matplotlib nbagg
import seaborn as sns
from sklearn.linear_model import LinearRegression
```

In [2]:
```
1 data = pd.read_csv('Data/kc_house_data.csv')
2 data.head()
```

Out[2]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
0	7399300360	5/24/2022	675000.0	4	1.0	1180	7140	1.0	NO
1	8910500230	12/13/2021	920000.0	5	2.5	2770	6703	1.0	NO
2	1180000275	9/29/2021	311000.0	6	2.0	2880	6156	1.0	NO
3	1604601802	12/14/2021	775000.0	3	3.0	2160	1400	2.0	NO
4	8562780790	8/24/2021	592500.0	2	2.0	1120	758	2.0	NO

5 rows × 25 columns

Next, I try to find out more about the data and narrow down the dataframe I will be using to only include the necessary columns.

In [3]:
```
1 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
```

In [4]:
```
1 data.describe()
```

Out[4]:

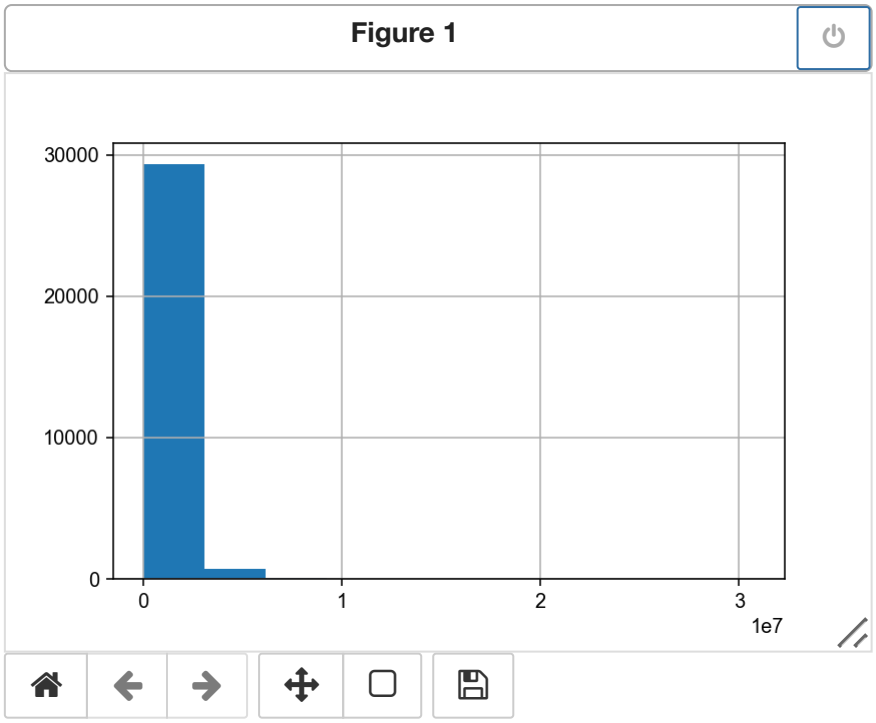
	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	
count	3.015500e+04	3.015500e+04	30155.000000	30155.000000	30155.000000	3.015500e+04	30155.(
mean	4.538104e+09	1.108536e+06	3.413530	2.334737	2112.424739	1.672360e+04	1.5
std	2.882587e+09	8.963857e+05	0.981612	0.889556	974.044318	6.038260e+04	0.5
min	1.000055e+06	2.736000e+04	0.000000	0.000000	3.000000	4.020000e+02	1.(
25%	2.064175e+09	6.480000e+05	3.000000	2.000000	1420.000000	4.850000e+03	1.(
50%	3.874011e+09	8.600000e+05	3.000000	2.500000	1920.000000	7.480000e+03	1.5
75%	7.287100e+09	1.300000e+06	4.000000	3.000000	2619.500000	1.057900e+04	2.(
max	9.904000e+09	3.075000e+07	13.000000	10.500000	15360.000000	3.253932e+06	4.(

Data Preparation

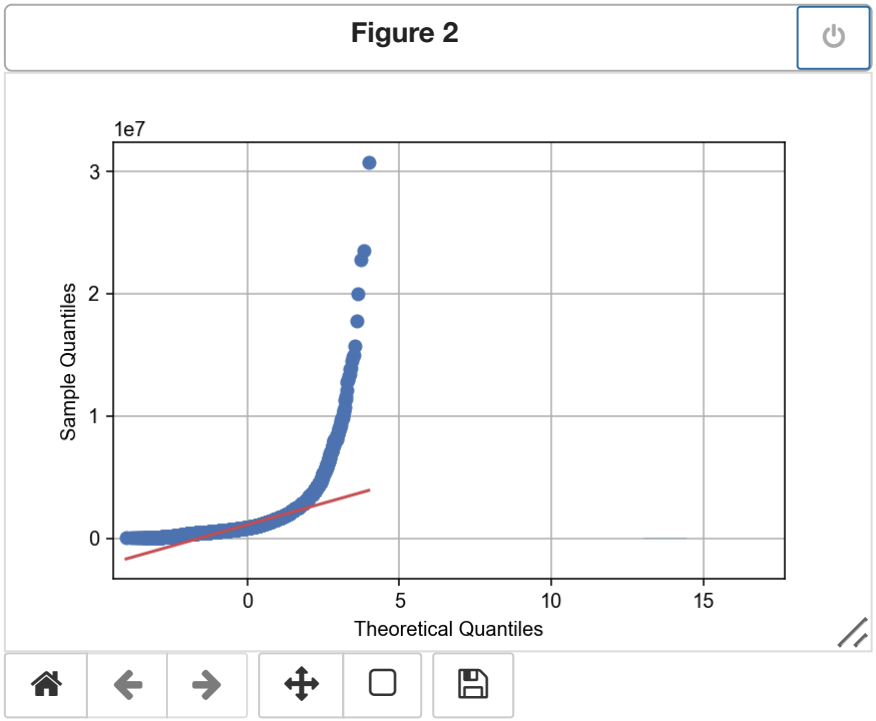
Before I look at the predictors, I want to investigate the target variable ('price'):

I start by plotting the target variable's distribution with a histogram and its residuals with a QQ plot.

In [5]:
```
1 data['price'].hist();
```



In [6]:
```
price_qq = sm.qqplot(data['price'], line='r');
```



The histogram does not seem to show a perfectly normal looking distribution, and the QQ plot shows the residuals getting further and further away from the theoretical fit line.

In [7]:
```
lower_lim = np.mean(data['price']) - (2*np.std(data['price']))
upper_lim = np.mean(data['price']) + (2*np.std(data['price']))
```

In [8]:
```
lower_lim
```

Out[8]:
```
-684205.7543299033
```

In [9]:
```
upper_lim
```

Out[9]:
```
2901277.4300719034
```

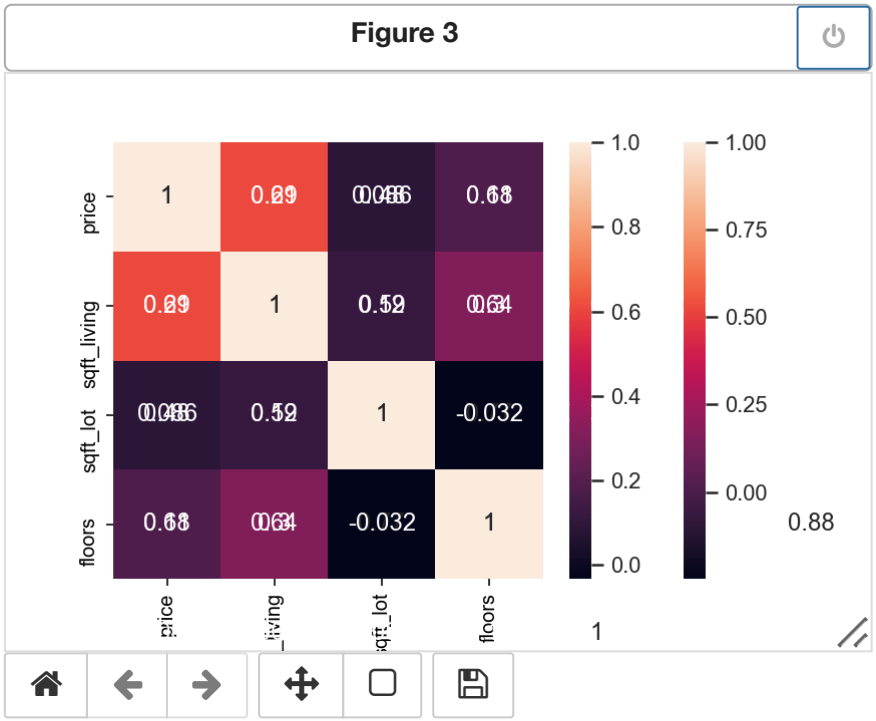
This indicates that the data becomes unreliable outside of plus or minus 2 standard deviations from the mean, or outside of the range where price equals between 684,205.75 and 2,901,227.43 dollars.

I then try to log transform the target variable to see if that improves its distribution or the distribution of its residuals.

In [10]:
```
np.log(data['price']).hist();
```

The log-transformed histogram of price looks more normal than the original variable's distribution.

In [11]:
```
1 sm.qqplot(np.log(data['price']), line='r');
```



This QQ plot is still not perfectly linear but is it better than before we log-transformed the target. The data is still only reliable within the same same, so it likely is not worthwhile to use the log-transformed target feature moving forward as it will be harder to interpret.

Choosing a baseline model feature:

Next, I want to know which variables are most correlated with price, so that I can choose a feature for the baseline model. However, first I want to make sure that if any features are causing multicollinearity, that they are removed so that they will not later affect my results.

In [12]:
```
1 data_all = data.copy()
```

In [13]:
```
1 data_pred = data_all.iloc[:,2:21]
2 data_pred.head()
```

Out[13]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	greenbelt	nuisance	
0	675000.0	4	1.0	1180	7140	1.0	NO	NO	NO	
1	920000.0	5	2.5	2770	6703	1.0	NO	NO	YES	AVE
2	311000.0	6	2.0	2880	6156	1.0	NO	NO	NO	AVE
3	775000.0	3	3.0	2160	1400	2.0	NO	NO	NO	AVE
4	592500.0	2	2.0	1120	758	2.0	NO	NO	YES	

In [14]:
```
1 data_pred.corr()
```

Out[14]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	sqft_above	sqft_b
price	1.000000	0.289204	0.480401	0.608521	0.085730	0.180576	0.538651	
bedrooms	0.289204	1.000000	0.589273	0.637874	0.003306	0.147592	0.547164	
bathrooms	0.480401	0.589273	1.000000	0.772677	0.035886	0.404412	0.674924	
sqft_living	0.608521	0.637874	0.772677	1.000000	0.119563	0.304240	0.883984	
sqft_lot	0.085730	0.003306	0.035886	0.119563	1.000000	-0.032097	0.129231	
floors	0.180576	0.147592	0.404412	0.304240	-0.032097	1.000000	0.448281	-(
sqft_above	0.538651	0.547164	0.674924	0.883984	0.129231	0.448281	1.000000	-(
sqft_basement	0.245058	0.238502	0.260902	0.338460	0.004111	-0.248093	-0.066801	
sqft_garage	0.264169	0.319441	0.457022	0.511740	0.087169	0.132656	0.560551	
sqft_patio	0.313409	0.183439	0.327551	0.396030	0.155250	0.125183	0.312117	
yr_built	0.096013	0.146191	0.443648	0.291694	0.001750	0.544646	0.387448	-(

In [15]:
```
1
2 sns.set(rc={'figure.figsize':(12,8)})
3 sns.heatmap(data_pred.corr(), annot=True)
```

Out[15]: <Axes: >

In [16]:
```
1  df=data_pred.corr().abs().stack().reset_index().sort_values(0, ascendin
2
3  # zip the variable name columns (Which were only named level_0 and leve
4  df['pairs'] = list(zip(df.level_0, df.level_1))
5
6  # set index to pairs
7  df.set_index(['pairs'], inplace = True)
8
9  #d rop level columns
10 df.drop(columns=['level_1', 'level_0'], inplace = True)
11
12 # rename correlation column as cc rather than 0
13 df.columns = ['cc']
14
15 # drop duplicates. This could be dangerous if you have variables perfec
16 # for the sake of exercise, kept it in.
17 df.drop_duplicates(inplace=True)
```

In [17]:
```
1 abs(df) > 0.75
```

Out[17]:

pairs	cc
(price, price)	True
(sqft_living, sqft_above)	True
(sqft_living, bathrooms)	True
(bathrooms, sqft_above)	False
(bedrooms, sqft_living)	False
(price, sqft_living)	False
(bedrooms, bathrooms)	False
(sqft_garage, sqft_above)	False
(bedrooms, sqft_above)	False
(yr_built, floors)	False
(price, sqft_above)	False
(sqft_garage, sqft_living)	False
(bathrooms, price)	False
(bathrooms, sqft_garage)	False
(floors, sqft_above)	False
(yr_built, sqft_garage)	False
(yr_built, bathrooms)	False
(bathrooms, floors)	False
(sqft_patio, sqft_living)	False
(yr_built, sqft_above)	False
(sqft_basement, sqft_living)	False
(sqft_patio, bathrooms)	False
(bedrooms, sqft_garage)	False
(sqft_patio, price)	False
(sqft_patio, sqft_above)	False
(sqft_living, floors)	False
(yr_built, sqft_living)	False
(price, bedrooms)	False
(sqft_garage, price)	False
(sqft_basement, bathrooms)	False
(sqft_basement, floors)	False
(price, sqft_basement)	False
(bedrooms, sqft_basement)	False

pairs	cc
(sqft_basement, yr_built)	False
(sqft_patio, sqft_garage)	False
(sqft_basement, sqft_patio)	False
(bedrooms, sqft_patio)	False
(price, floors)	False
(sqft_patio, sqft_lot)	False
(floors, bedrooms)	False
(yr_built, bedrooms)	False
(yr_built, sqft_patio)	False
(floors, sqft_garage)	False
(sqft_lot, sqft_above)	False
(sqft_patio, floors)	False
(sqft_living, sqft_lot)	False
(yr_built, price)	False
(sqft_lot, sqft_garage)	False
(sqft_lot, price)	False
(sqft_above, sqft_basement)	False
(sqft_lot, bathrooms)	False
(sqft_lot, floors)	False
(sqft_garage, sqft_basement)	False
(sqft_lot, sqft_basement)	False
(bedrooms, sqft_lot)	False
(yr_built, sqft_lot)	False

In [18]:
```
1 df[(df.cc>.75) & (df.cc <1)]
```

Out[18]:

pairs	cc
(sqft_living, sqft_above)	0.883984
(sqft_living, bathrooms)	0.772677

The only pairs of features that have correlations higher than 0.75 are sqft_living and sqft_above, and sqft_living and bathrooms. This makes sense, because the square feet of the living area is likely a large portion of the square feet above ground for a house.

Also, it would make sense that the larger amount of square footage of living space, the higher number of bathrooms a house would have. Following this same logic, the 'bedrooms' feature would also likely cause multicollinearity.

Looking at the correlations, I see that the bedrooms and bathrooms features are more correlated with the feature 'sqft_living' than they are with price, so this indicates multicollinearity. Because sqft_living is the most correlated with the target variable out of these, I am going to exclude the

Now, to look at the rest of the features:

Paying more attention to which variables are correlated with price, as opposed to with each other, I compare the rest of the relevant numeric features.

In [19]:
```
data1 = data_all.copy()
data1 = data1[['price', 'sqft_living', 'sqft_lot', 'floors', 'waterfron
data1.head()
```

Out[19]:

	price	sqft_living	sqft_lot	floors	waterfront	greenbelt	nuisance	view	heat_source	sev
0	675000.0	1180	7140	1.0	NO	NO	NO	NONE	Gas	
1	920000.0	2770	6703	1.0	NO	NO	YES	AVERAGE	Oil	
2	311000.0	2880	6156	1.0	NO	NO	NO	AVERAGE	Gas	
3	775000.0	2160	1400	2.0	NO	NO	NO	AVERAGE	Gas	
4	592500.0	1120	758	2.0	NO	NO	YES	NONE	Electricity	

In [20]:
```
data1.corr()["price"]
```

Out[20]:
```
price          1.000000
sqft_living    0.608521
sqft_lot       0.085730
floors         0.180576
Name: price, dtype: float64
```

In [21]:
```

sns.set(rc={'figure.figsize':(12,8)})
sns.heatmap(data1.corr(), annot=True)
```

Out[21]:
```
<Axes: >
```

In [22]:
```
data_num = data1.copy().select_dtypes("number")
data_num.dropna(inplace=True)
data_num
```

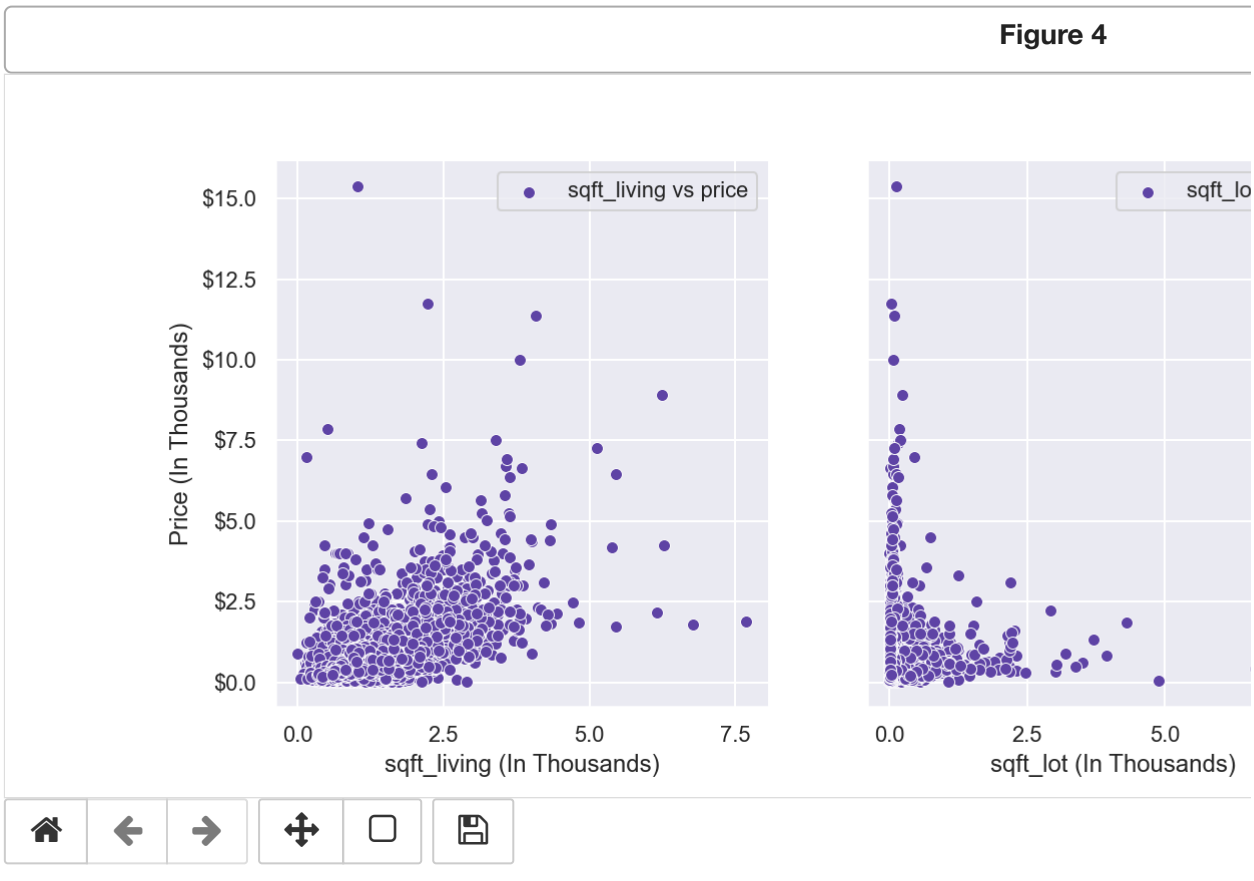
Out[22]:

	price	sqft_living	sqft_lot	floors
0	675000.0	1180	7140	1.0
1	920000.0	2770	6703	1.0
2	311000.0	2880	6156	1.0
3	775000.0	2160	1400	2.0
4	592500.0	1120	758	2.0
...	...	...	...	...
30150	1555000.0	1910	4000	1.5
30151	1313000.0	2020	5800	2.0
30152	800000.0	1620	3600	1.0
30153	775000.0	2570	2889	2.0
30154	500000.0	1200	11058	1.0

In [23]:

```
import matplotlib.pyplot as plt
from matplotlib import ticker

sns.set_palette("twilight", 2)
# Plot scatterplots for multiple columns
fig, (ax1, ax2, ax3) = plt.subplots(figsize=(15,5), sharey=True, nrows=
ax1 = sns.scatterplot(data = data_pred, x='sqft_living', y='price', lab
ax2 = sns.scatterplot(data = data_pred, x='sqft_lot', y='price', label=
ax3 = sns.scatterplot(data = data_pred, x='floors', y='price', label='f

# A function can also be used directly as a formatter. The function mus
# two arguments: ``x`` for the tick value and ``pos`` for the tick posi
# and must return a ``str``. This creates a FuncFormatter automatically
#setup(ax1, title="lambda x, pos: str(x-5)")
#ax1.yaxis.set_major_formatter(ticker.FormatStrFormatter("$%.2d")*1e-8)
ylabels = ['${:,.1f}'.format(x) for x in ax1.get_xticks()/1000]
ax1.set_yticklabels(ylabels)
ax1.set_ylabel("Price (In Thousands)")

xlabels = ['{:,.1f}'.format(x) for x in ax1.get_xticks()/1000]
ax1.set_xticklabels(xlabels)
ax2.set_xticklabels(xlabels)
ax1.set_xlabel("sqft_living (In Thousands)")
ax2.set_xlabel("sqft_lot (In Thousands)")

sns.set_style("whitegrid")
sns.despine()
plt.legend()
plt.show();
```

Figure 4



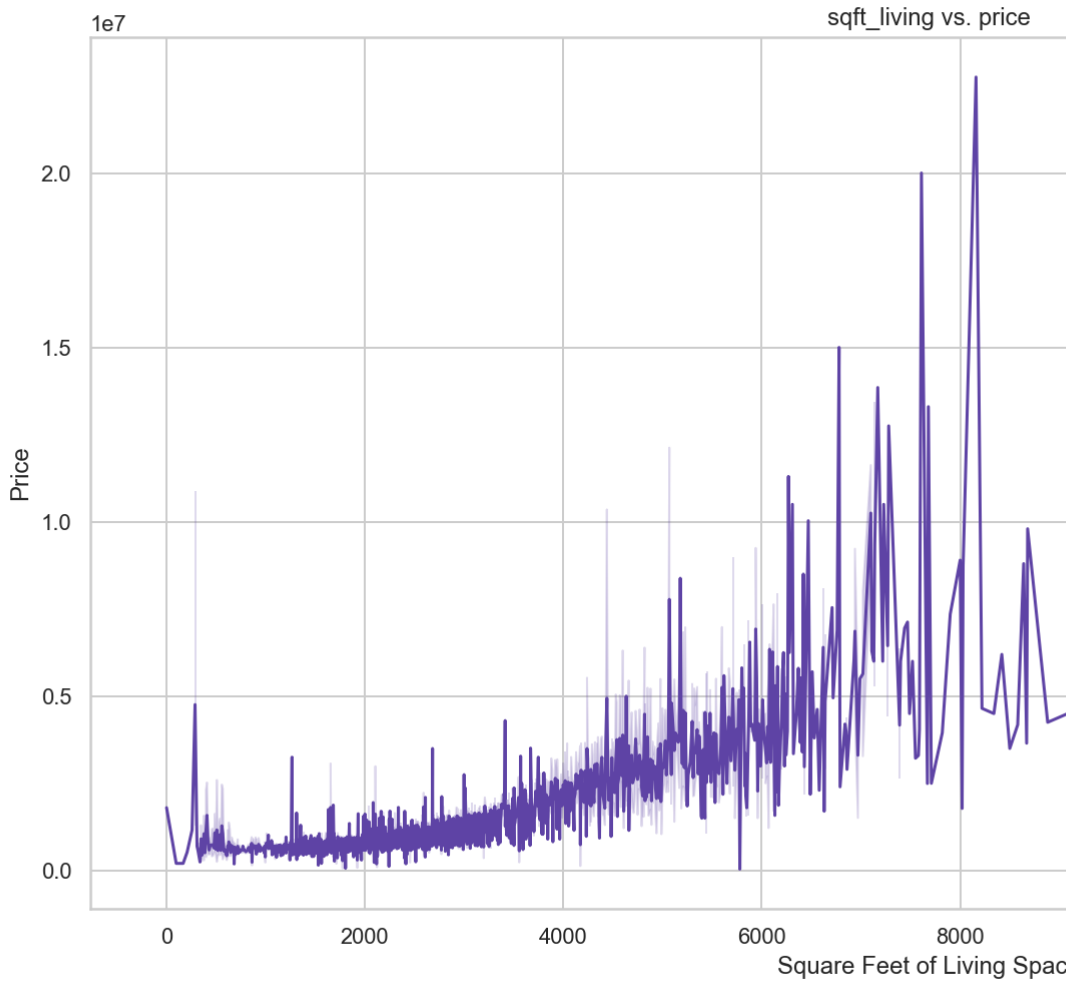
```
<ipython-input-23-811fe0fb7bb7>:17: UserWarning: FixedFormatter should on
ly be used together with FixedLocator
  ax1.set_yticklabels(ylabels)
<ipython-input-23-811fe0fb7bb7>:21: UserWarning: FixedFormatter should on
ly be used together with FixedLocator
  ax1.set_xticklabels(xlabels)
<ipython-input-23-811fe0fb7bb7>:22: UserWarning: FixedFormatter should on
ly be used together with FixedLocator
  ax2.set_xticklabels(xlabels)
```

Of the numeric variables in the data, aside from those already removed, it looks like the feature most correlated with price and having the strongest linear relationship with price is sqft_living. Therefore, this seems like a good feature to use for the baseline model.

In [24]:
```
price_usable = data1[(data1['price']>= 684205.75)&(data1['price']<= 290

```

In [25]:
```
fig, ax = plt.subplots(figsize = (15,8))
sns.set_style("darkgrid", {"axes.facecolor": ".9"})
sns.lineplot(data=data1, x='sqft_living', y='price').set(xlabel="Square
```

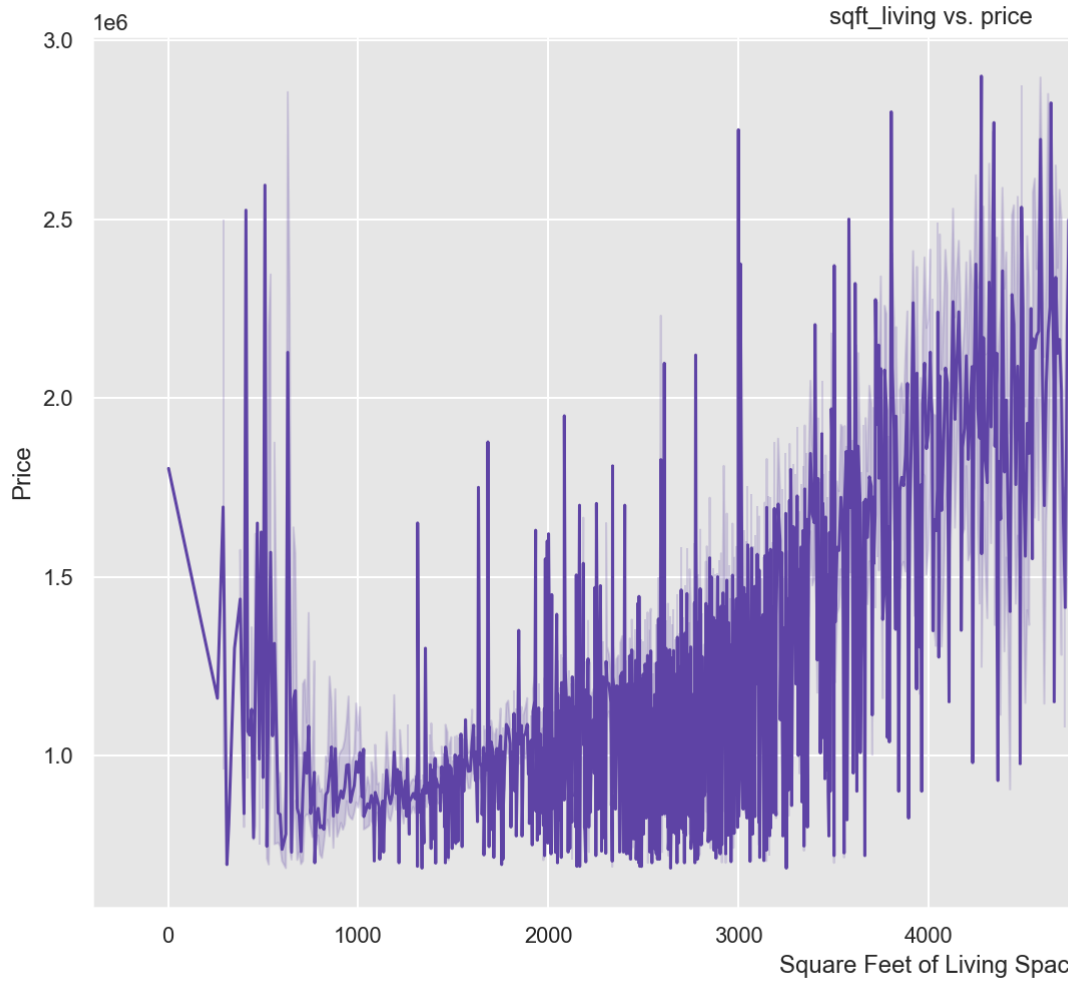
Figure 5



Out[25]:
```
[Text(0.5, 0, 'Square Feet of Living Space'),
 Text(0, 0.5, 'Price'),
 Text(0.5, 1.0, 'sqft_living vs. price')]
```

In [26]:
```
fig, ax = plt.subplots(figsize = (15,8))
sns.set_style("darkgrid", {"axes.facecolor": ".9"})
sns.set_palette("cool", 2)
sns.lineplot(data=price_usable, x='sqft_living', y='price').set(xlabel=
```

Figure 6

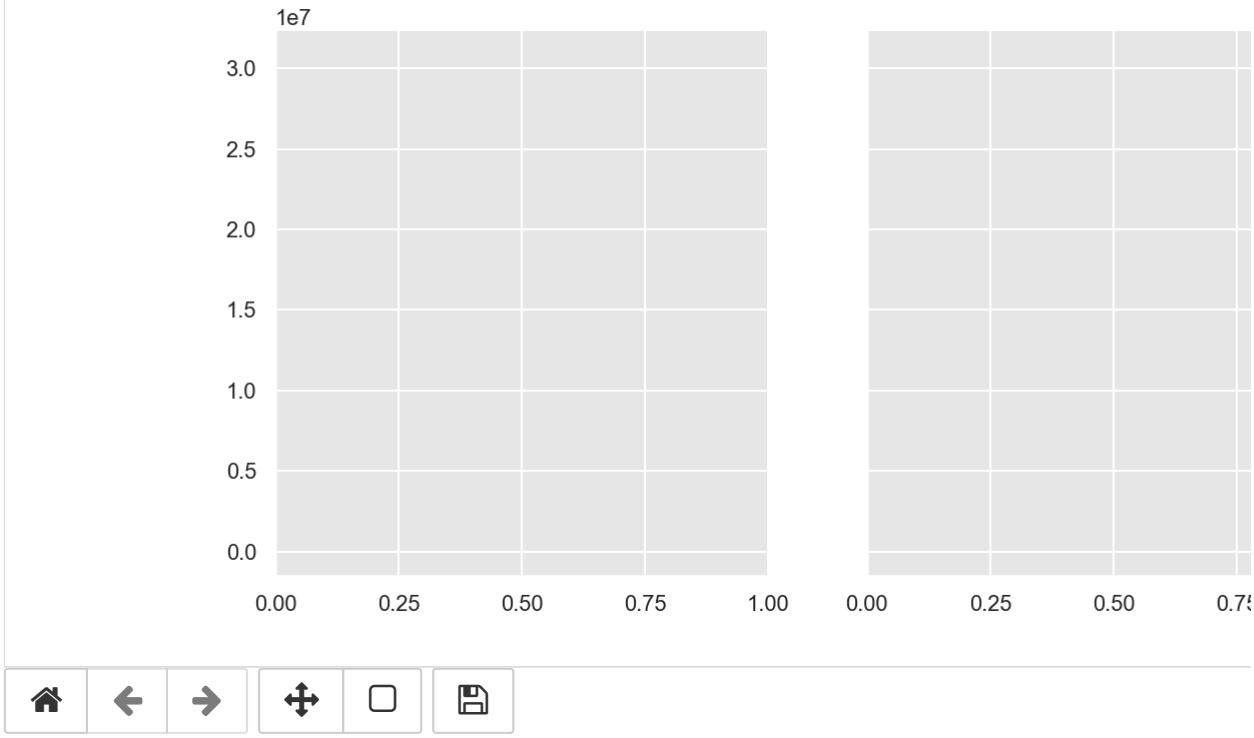


Out[26]:
```
[Text(0.5, 0, 'Square Feet of Living Space'),
 Text(0, 0.5, 'Price'),
 Text(0.5, 1.0, 'sqft_living vs. price')]
```

In [27]:

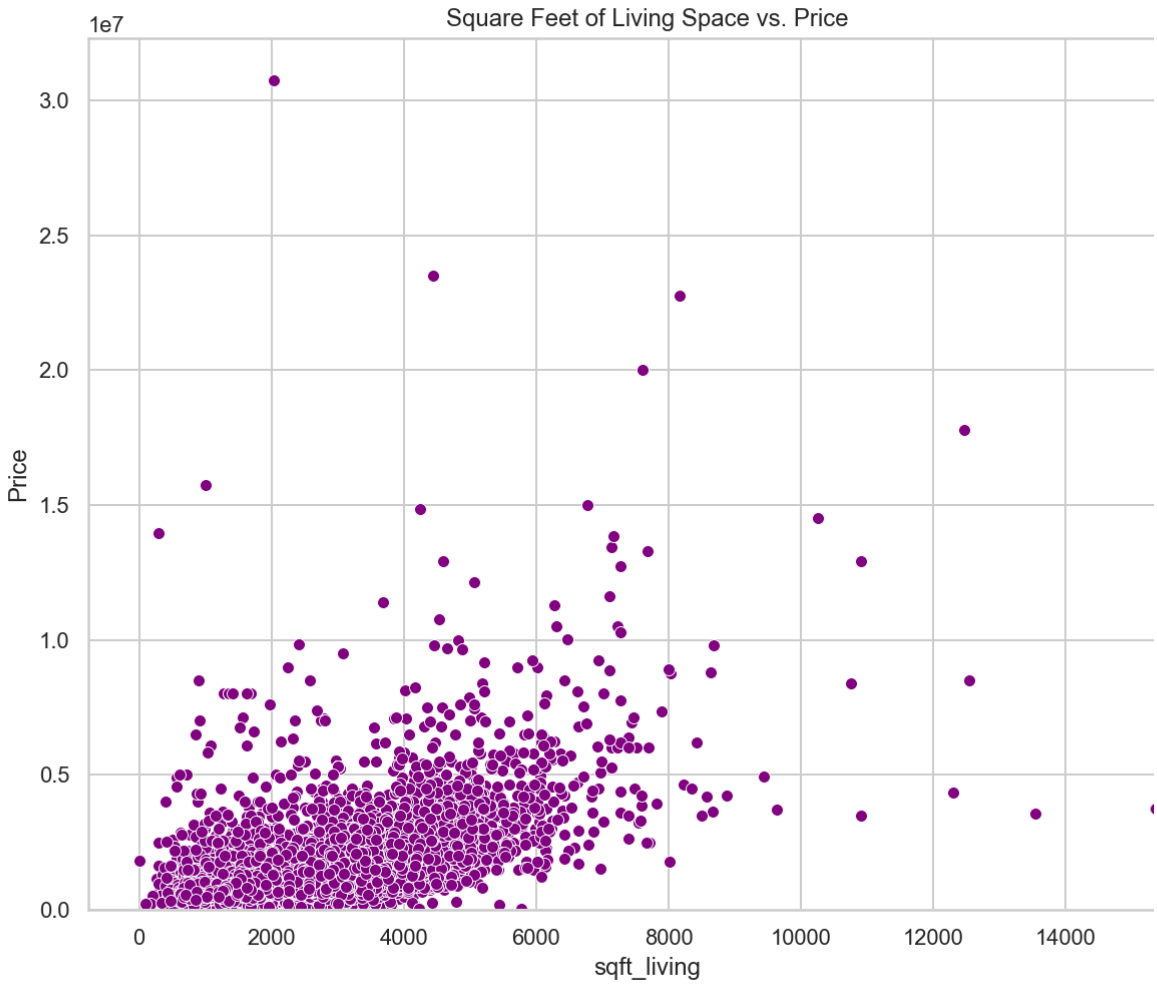
```
sns.set_palette("twilight", 2)
# Plot scatterplots for multiple columns
fig, ax = plt.subplots(figsize=(15,5), sharey=True, nrows=1, ncols=3)
sns.lineplot(data = data1, x='sqft_living', y='price', label='sqft_livi
sns.lineplot(data = data1, x='sqft_lot', y='price', label='sqft_lot vs
sns.lineplot(data = data1, x='floors', y='price', label='floors vs pric
# Display the plot
sns.set_style("whitegrid")

sns.despine()
plt.legend()
plt.show();
```

Figure 7



In [28]:
```
sns.set_style()
fig, ax = plt.subplots(figsize=(10,8))
g = sns.scatterplot(data=data_num, x='sqft_living', y='price', color="p
g.set_title("Square Feet of Living Space vs. Price")
g.set_ylabel("Price")
g.set_xlabel("sqft_living")
```

Figure 8



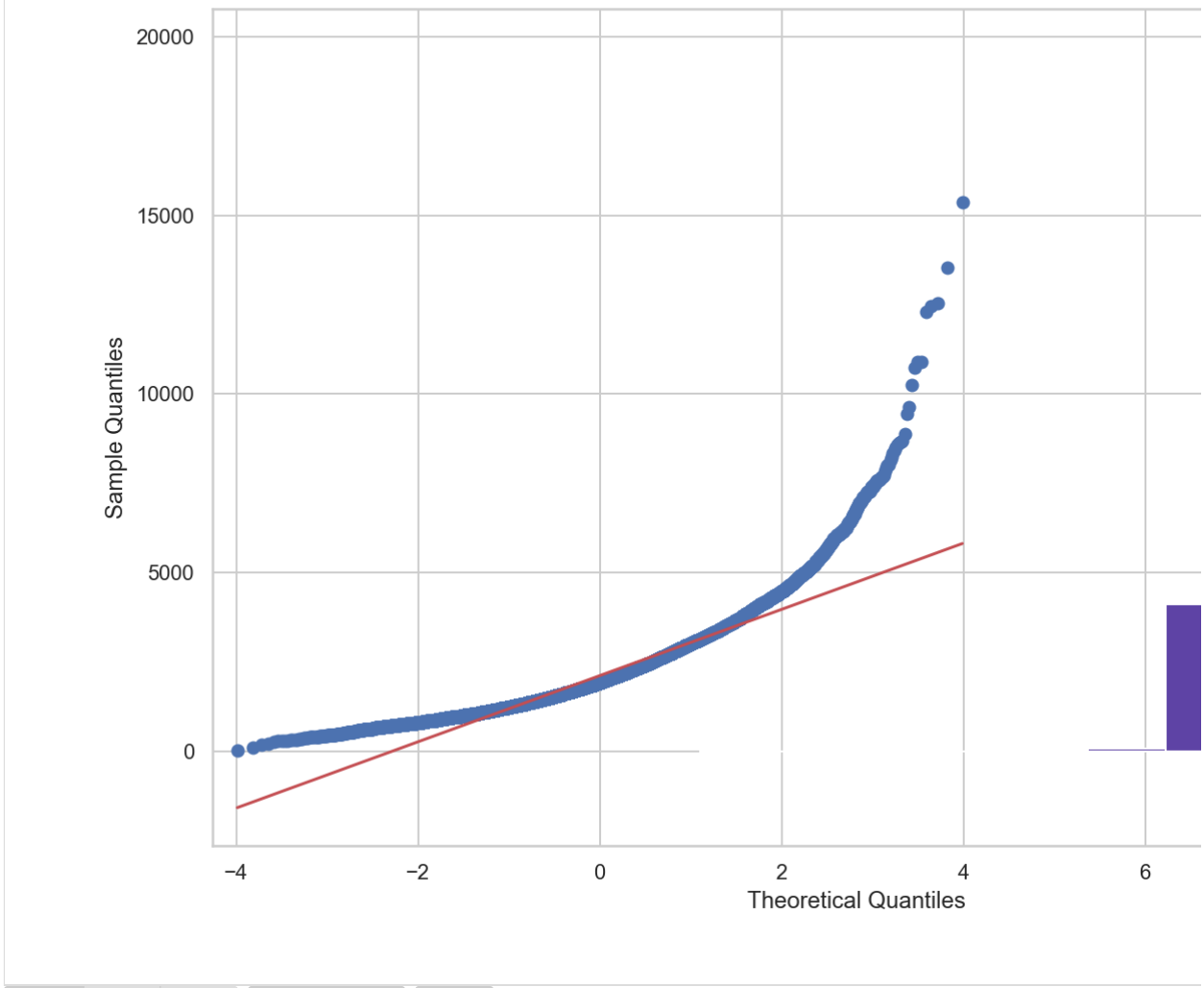
Out[28]: Text(0.5, 0, 'sqft_living')

Now, I want to investigate the distribution of sqft_living to see if it is normal and if its residuals follow the theoretical model.

In [29]:
```
sqftliving_hist = data_pred['sqft_living'].hist()
plt.show();
```

In [30]:
```
sqftliving_qq = sm.qqplot(data_pred['sqft_living'], line='r')
plt.show();
```

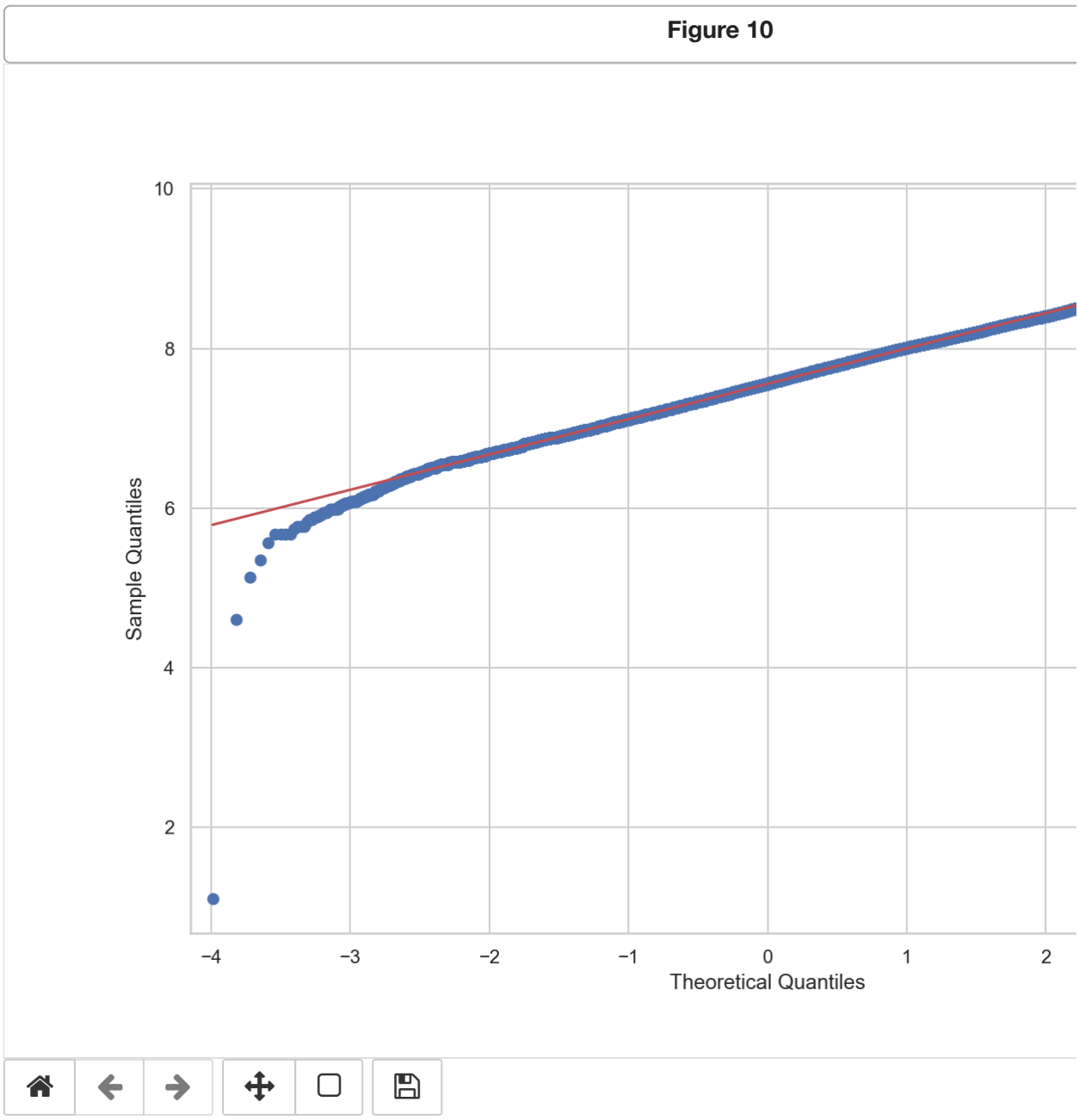
Figure 9



The histogram of sqft_living does not look very normal, and much like the QQ plot of price, the residuals also tend to get further away from the theoretical line. To see if it improves the distribution, I am going to try log transforming this variable.

In [31]:
```
sqftliving_log_hist = np.log(data_pred['sqft_living']).hist()
```

In [32]:
```
1
2 sqftliving_log_qq = sm.qqplot(np.log(data_pred['sqft_living']), line='r
```



The transformed feature's distribution looks much more normal than the untransformed variable, and the residuals follow the theoretical line better as well.

Modeling

First, a baseline model is created using the variable sqft_living, as it is the most correlated with sale price, to compare all other models to.

In [33]:
```
y = data1[['price']]
X_baseline = data1[['sqft_living']]
```

Baseline results:

In [34]:
```
baseline_model = sm.OLS(y, sm.add_constant(X_baseline))
baseline_results = baseline_model.fit()

print(baseline_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.370
Model:                            OLS   Adj. R-squared:                  0.370
Method:                 Least Squares   F-statistic:                 1.773e+04
Date:                Fri, 02 Jun 2023   Prob (F-statistic):               0.00
Time:                        13:50:22   Log-Likelihood:            -4.491e+05
No. Observations:               30155   AIC:                         8.982e+05
Df Residuals:                   30153   BIC:                         8.983e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -7.443e+04   9782.728     -7.609      0.000   -9.36e+04   -5.53e+04
sqft_living   560.0050      4.206    133.160      0.000     551.762     568.248
==============================================================================
Omnibus:                    43429.367   Durbin-Watson:                   1.862
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          4715918.471
Skew:                           8.188   Prob(JB):                         0.00
Kurtosis:                     196.042   Cond. No.                     5.56e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.56e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

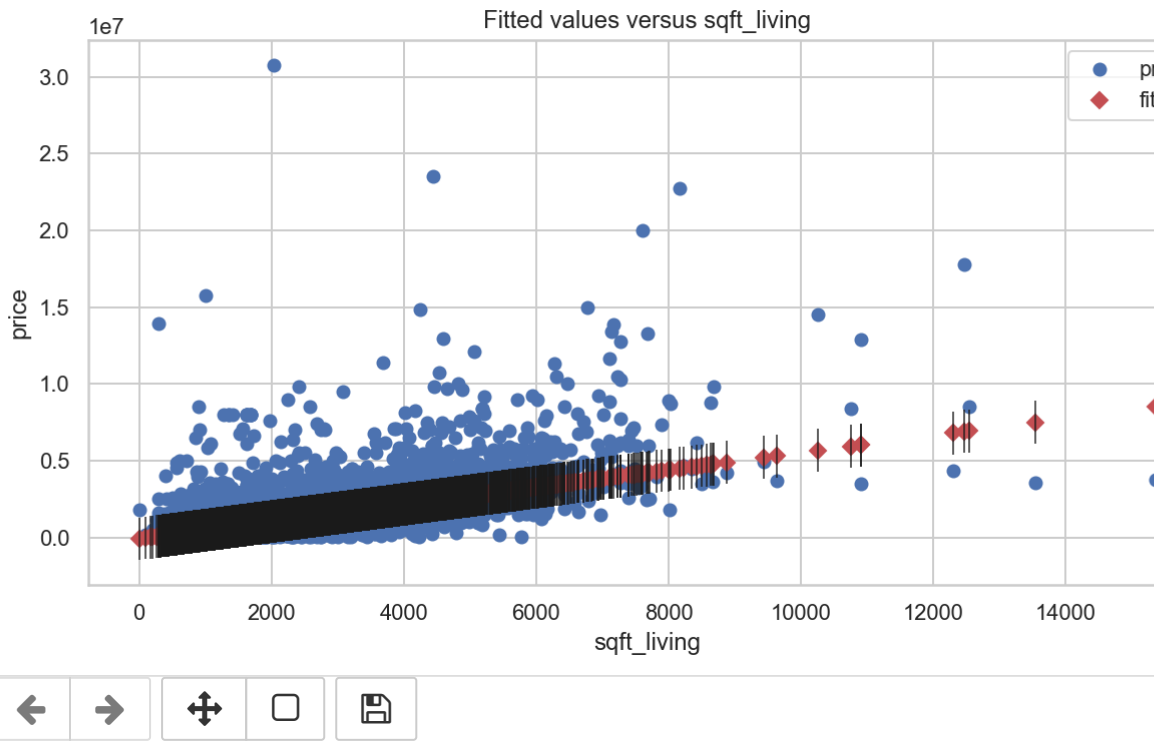
In [35]:
```
1 baseline_results.pvalues
```

Out[35]:
```
const          2.852024e-14
sqft_living    0.000000e+00
dtype: float64
```

Plotting the actual vs. predicted values of this model:

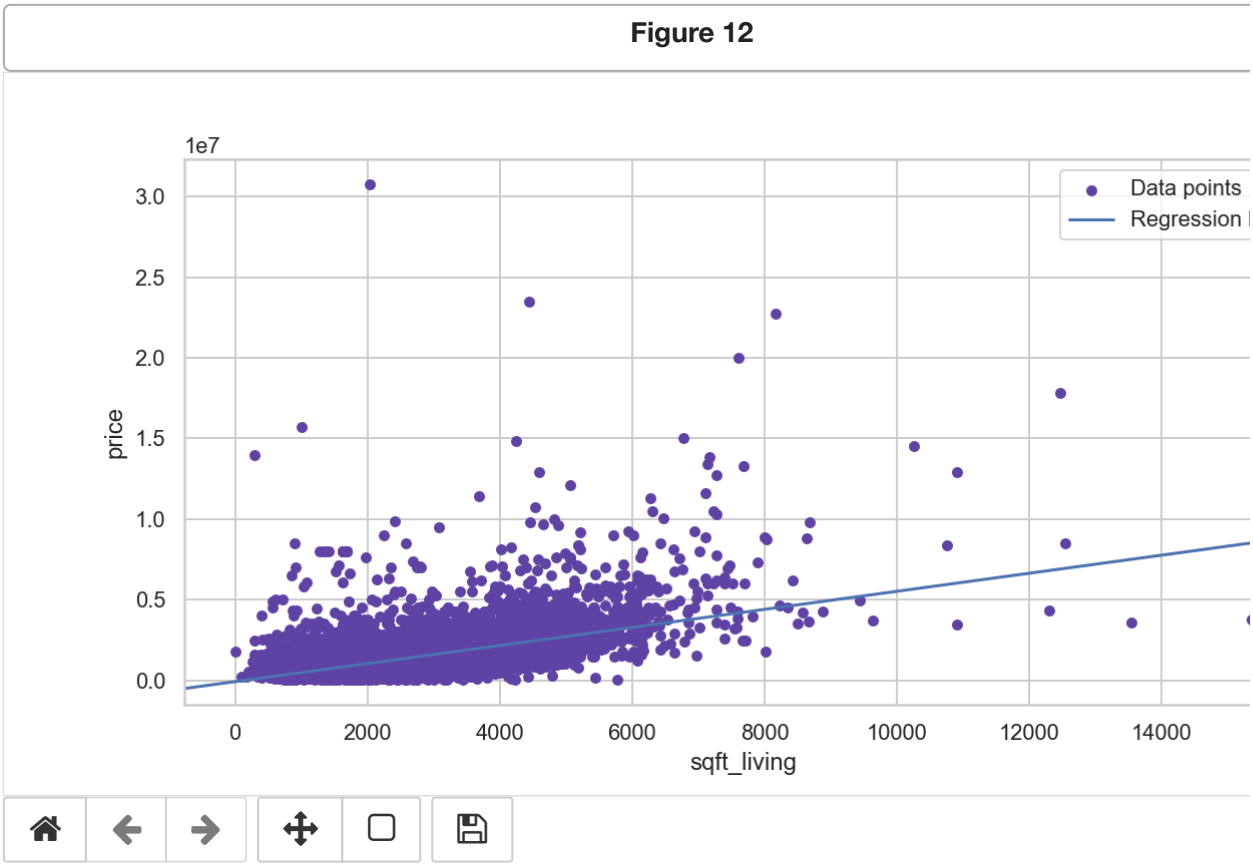
In [36]:
```
1 fig, ax = plt.subplots(figsize=(10,5))
2 sm.graphics.plot_fit(baseline_results, "sqft_living", ax=ax)
3 plt.show()
```

Figure 11



Plotting the regression line:

In [37]:
```
fig, ax = plt.subplots(figsize=(10,5))
data1.plot.scatter(x="sqft_living", y="price", label="Data points", ax=
sm.graphics.abline_plot(model_results=baseline_results, label="Regressi
ax.legend();
```

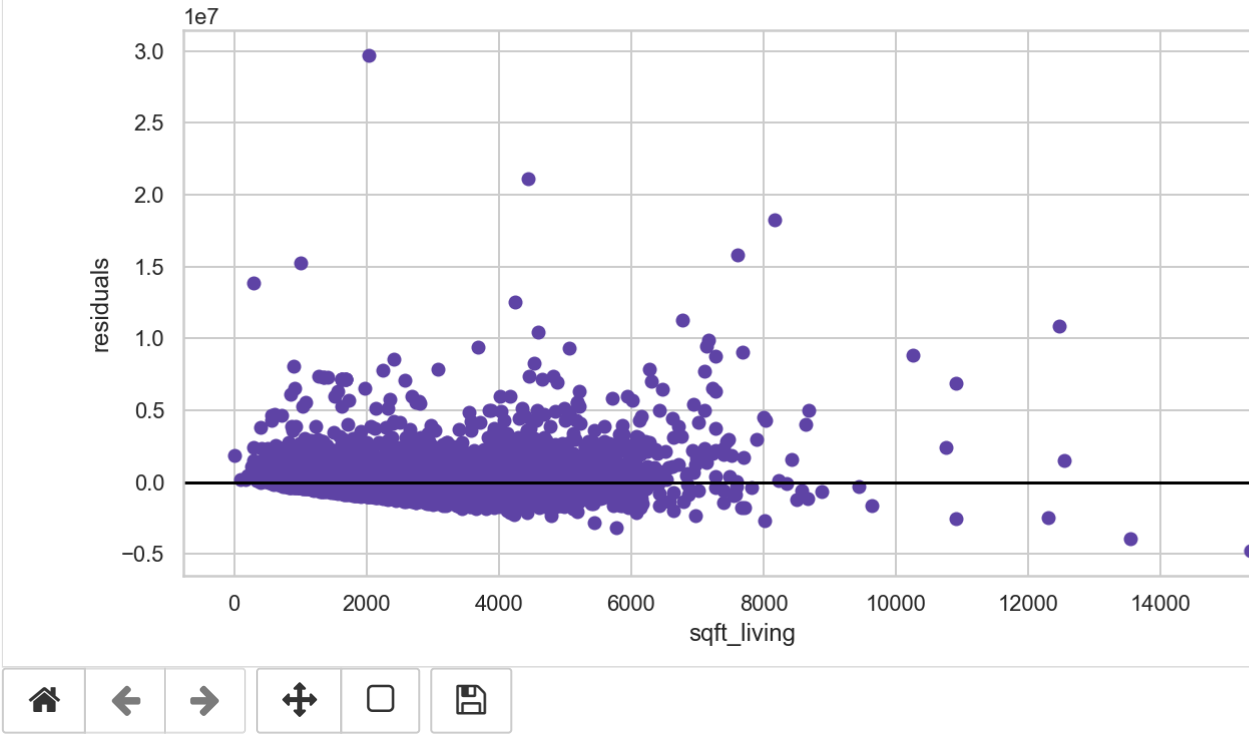



```
/Users/nicolemichaud/opt/anaconda3/envs/learn-env/lib/python3.8/site-pack
ages/pandas/plotting/_matplotlib/core.py:1010: UserWarning: *c* argument
looks like a single numeric RGB or RGBA sequence, which should be avoided
as value-mapping will have precedence in case its length matches with *x*
& *y*.  Please use the *color* keyword-argument or provide a 2D array wit
h a single row if you intend to specify the same RGB or RGBA value for al
l points.
  scatter = ax.scatter(
/Users/nicolemichaud/opt/anaconda3/envs/learn-env/lib/python3.8/site-pack
ages/pandas/plotting/_matplotlib/core.py:1010: UserWarning: No data for c
olormapping provided via 'c'. Parameters 'cmap' will be ignored
  scatter = ax.scatter(
```

Plotting the residuals:

In [38]:
```
fig, ax = plt.subplots(figsize=(10,5))

ax.scatter(data1["sqft_living"], baseline_results.resid)
ax.axhline(y=0, color="black")
ax.set_xlabel("sqft_living")
ax.set_ylabel("residuals");
```

Figure 13



Numeric data:

Now, to add the other numeric features to a multiple linear regression to see if it improves our model:

In [39]:
```
X_all = data1.drop(['price'], axis=1).select_dtypes("number")
X_all
```

Out[39]:

	sqft_living	sqft_lot	floors
0	1180	7140	1.0
1	2770	6703	1.0
2	2880	6156	1.0
3	2160	1400	2.0
4	1120	758	2.0
...	...	...	...
30150	1910	4000	1.5
30151	2020	5800	2.0
30152	1620	3600	1.0
30153	2570	2889	2.0
30154	1200	11058	1.0

Results for all numeric variables:

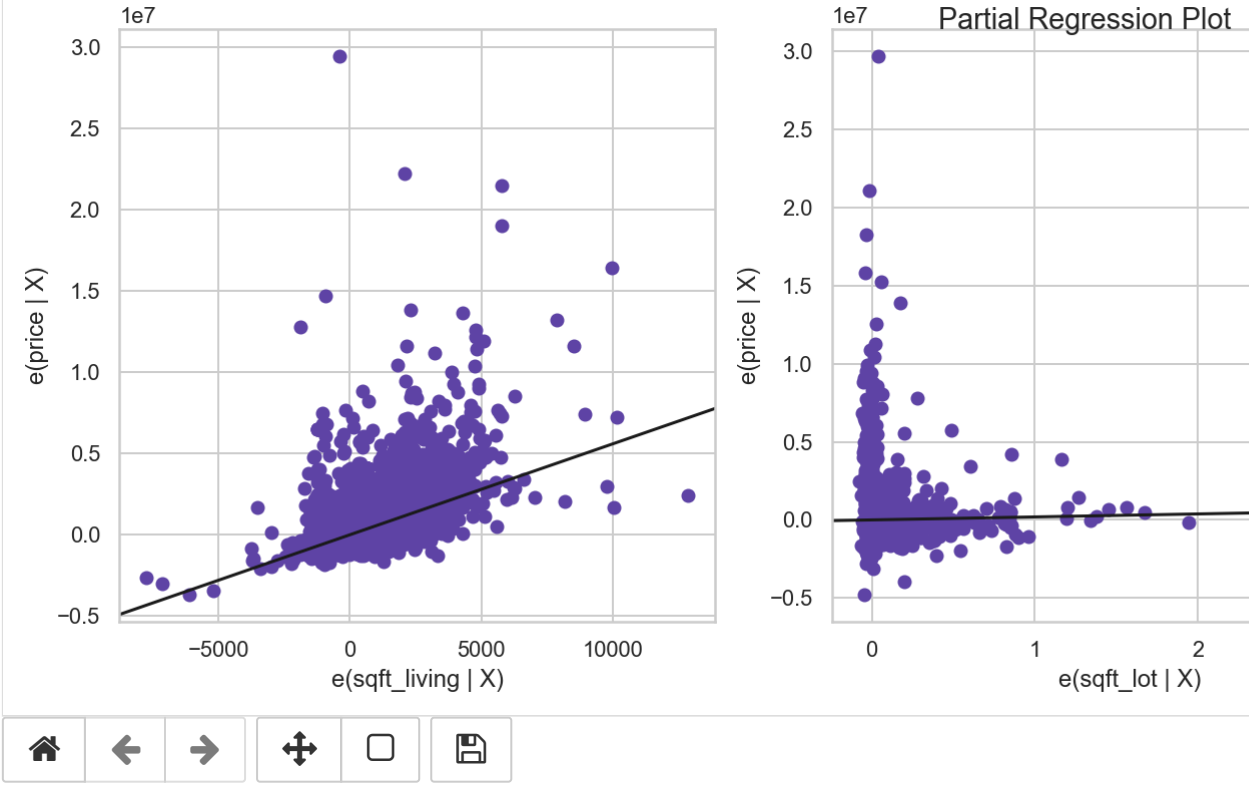
In [40]:

```
model = sm.OLS(y, sm.add_constant(X_all))
results_allnum = model.fit()

print(results_allnum.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.370
Model:                            OLS   Adj. R-squared:                  0.370
Method:                 Least Squares   F-statistic:                     5915.
Date:                Fri, 02 Jun 2023   Prob (F-statistic):               0.00
Time:                        13:50:23   Log-Likelihood:            -4.491e+05
No. Observations:               30155   AIC:                         8.982e+05
Df Residuals:                   30151   BIC:                         8.983e+05
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -6.716e+04   1.32e+04     -5.083      0.000   -9.31e+04   -4.13e+04
sqft_living   559.7226      4.456    125.623      0.000     550.989     568.456
sqft_lot        0.1912      0.069      2.791      0.005       0.057       0.325
floors      -6399.5386   7593.611     -0.843      0.399   -2.13e+04    8484.263
==============================================================================
Omnibus:                    43413.876   Durbin-Watson:                   1.862
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          4713866.786
Skew:                           8.182   Prob(JB):                         0.00
Kurtosis:                     196.001   Cond. No.                     2.19e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.19e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [41]:
```
fig = plt.figure(figsize=(15,5))
sm.graphics.plot_partregress_grid(
    results_allnum,
    exog_idx=list(X_all.columns.values),
    grid=(1,3),
    fig=fig)
plt.show();
```

Figure 14



These models look worse, so likely we included too many features. Since the numeric features of sqft_lot, and floors do not appear to have a positive linear relationship with price, we will remove those features.

Categorical data:

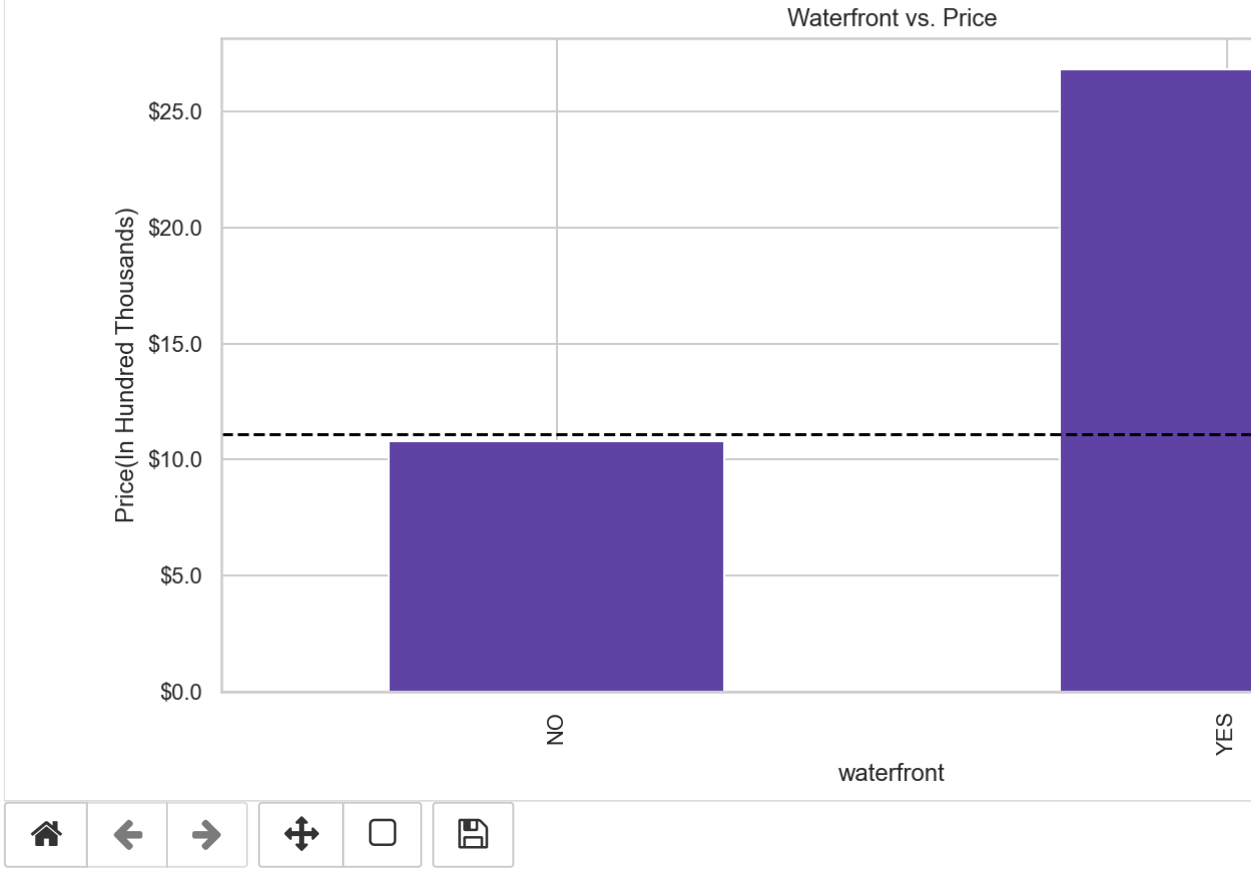
Now, before the categorical variables can be modeled, they will need to be transformed using one-hot encoding.

In [42]:
```
X_wf = data1[["waterfront"]]
X_gb = data1[["greenbelt"]]
X_nu = data1[["nuisance"]]
X_vw = data1[["view"]]
```

'waterfront' feature:

In [43]:
```
fig, ax = plt.subplots(figsize=(12,6))
data1.groupby("waterfront").mean().plot.bar(y="price", ax=ax).set(ylabe
ax.axhline(y=data1["price"].mean(), label="mean", color="black", linest
ylabels = ['${:,.1f}'.format(x) for x in ax.get_yticks()/100000]
ax.set_yticklabels(ylabels);

```

Figure 15




```
<ipython-input-43-f7f5419c8f52>:5: UserWarning: FixedFormatter should onl
y be used together with FixedLocator
  ax.set_yticklabels(ylabels);
```

In [44]:
```
waterfront_X = pd.get_dummies(X_wf, columns=["waterfront"], drop_first=
waterfront_X
```

Out[44]:

	waterfront_YES
0	0
1	0
2	0
3	0
4	0
...	...
30150	0
30151	0
30152	0
30153	0
30154	0

30155 rows × 1 columns

In [45]:
```
waterfront_model = sm.OLS(y, sm.add_constant(waterfront_X))
wf_results = waterfront_model.fit()

print(wf_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.054
Model:                            OLS   Adj. R-squared:                  0.054
Method:                 Least Squares   F-statistic:                     1719.
Date:                Fri, 02 Jun 2023   Prob (F-statistic):               0.00
Time:                        13:50:24   Log-Likelihood:            -4.552e+05
No. Observations:               30155   AIC:                         9.105e+05
Df Residuals:                   30153   BIC:                         9.105e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.081e+06   5064.675    213.436      0.000    1.07e+06    1.09e+06
waterfront_YES  1.601e+06   3.86e+04     41.463      0.000    1.53e+06    1.68e+06
==============================================================================
Omnibus:                    35538.027   Durbin-Watson:                   1.907
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1094446.954
Skew:                           5.884   Prob(JB):                         0.00
Kurtosis:                      95.585   Cond. No.                         7.69
==============================================================================

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

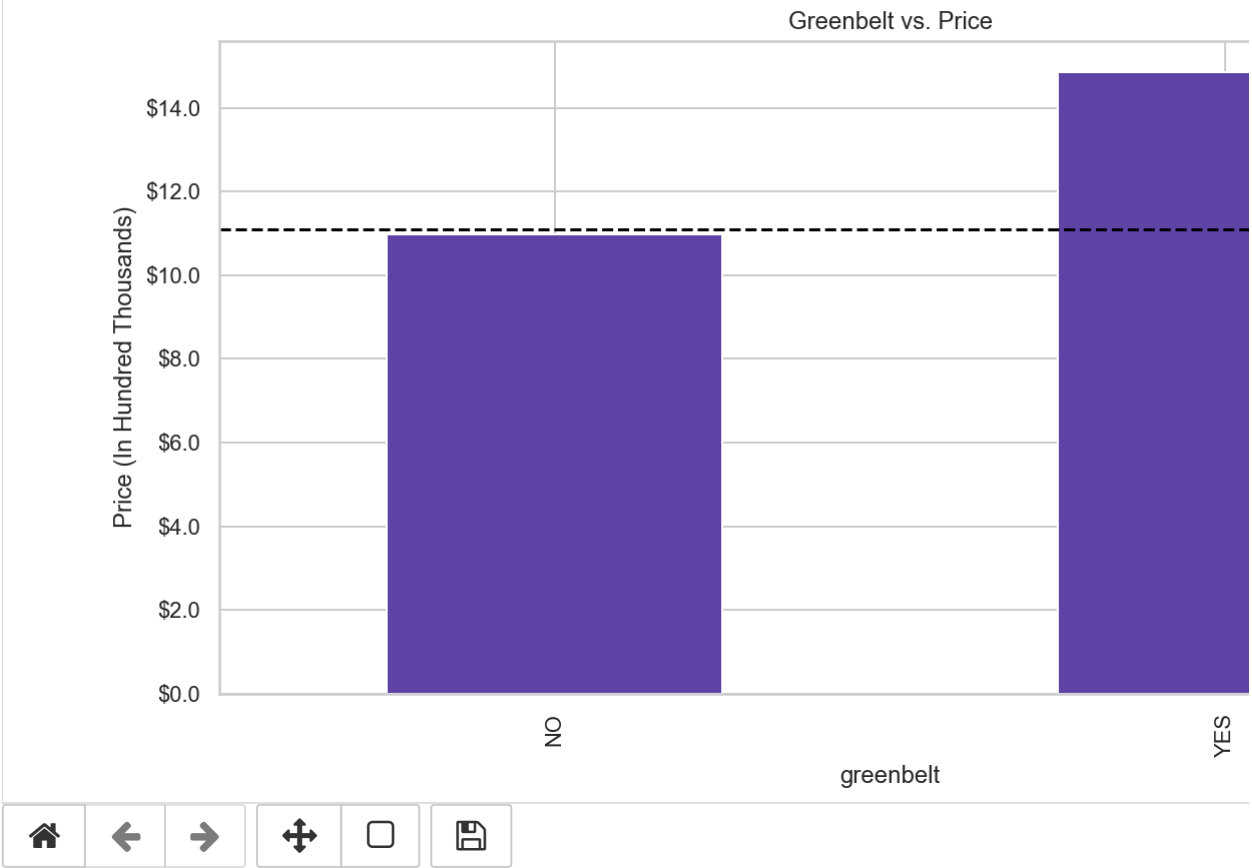
Interpretation:

This model is statistically significant and explains about 5.4% of the variance in price. Compared to

'greenbelt' feature:

In [46]:
```
fig, ax = plt.subplots(figsize=(12,6))
data1.groupby("greenbelt").mean().plot.bar(y="price", ax=ax).set(ylabel
ax.axhline(y=data1["price"].mean(), label="mean", color="black", linest
ylabels = ['${:,.1f}'.format(x) for x in ax.get_yticks()/100000]
ax.set_yticklabels(ylabels);
```

Figure 16




```
<ipython-input-46-3abed14de063>:5: UserWarning: FixedFormatter should onl
y be used together with FixedLocator
  ax.set_yticklabels(ylabels);
```

In [47]:
```
greenbelt_X = pd.get_dummies(X_gb, columns=["greenbelt"], drop_first=Tr
greenbelt_X
```

Out[47]:

	greenbelt_YES
0	0
1	0
2	0
3	0
4	0
...	...
30150	0
30151	0
30152	0
30153	0
30154	0

30155 rows × 1 columns

In [48]:
```
greenbelt_model = sm.OLS(y, sm.add_constant(greenbelt_X))
gb_results = greenbelt_model.fit()

print(gb_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.005
Model:                            OLS   Adj. R-squared:                  0.005
Method:                 Least Squares   F-statistic:                     141.1
Date:                Fri, 02 Jun 2023   Prob (F-statistic):           1.77e-32
Time:                        13:50:24   Log-Likelihood:            -4.5603e+05
No. Observations:               30155   AIC:                             9.121e+05
Df Residuals:                   30153   BIC:                             9.121e+05
Df Model:                           1
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const          1.099e+06   5217.319    210.570      0.000    1.09e+06    1.11e+06
greenbelt_YES  3.871e+05   3.26e+04     11.880      0.000    3.23e+05    4.51e+05
==============================================================================
Omnibus:                    38131.263   Durbin-Watson:                   1.913
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          14716007.254
Skew:                           6.655   Prob(JB):                         0.00
Kurtosis:                     110.402   Cond. No.                         6.33
==============================================================================

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

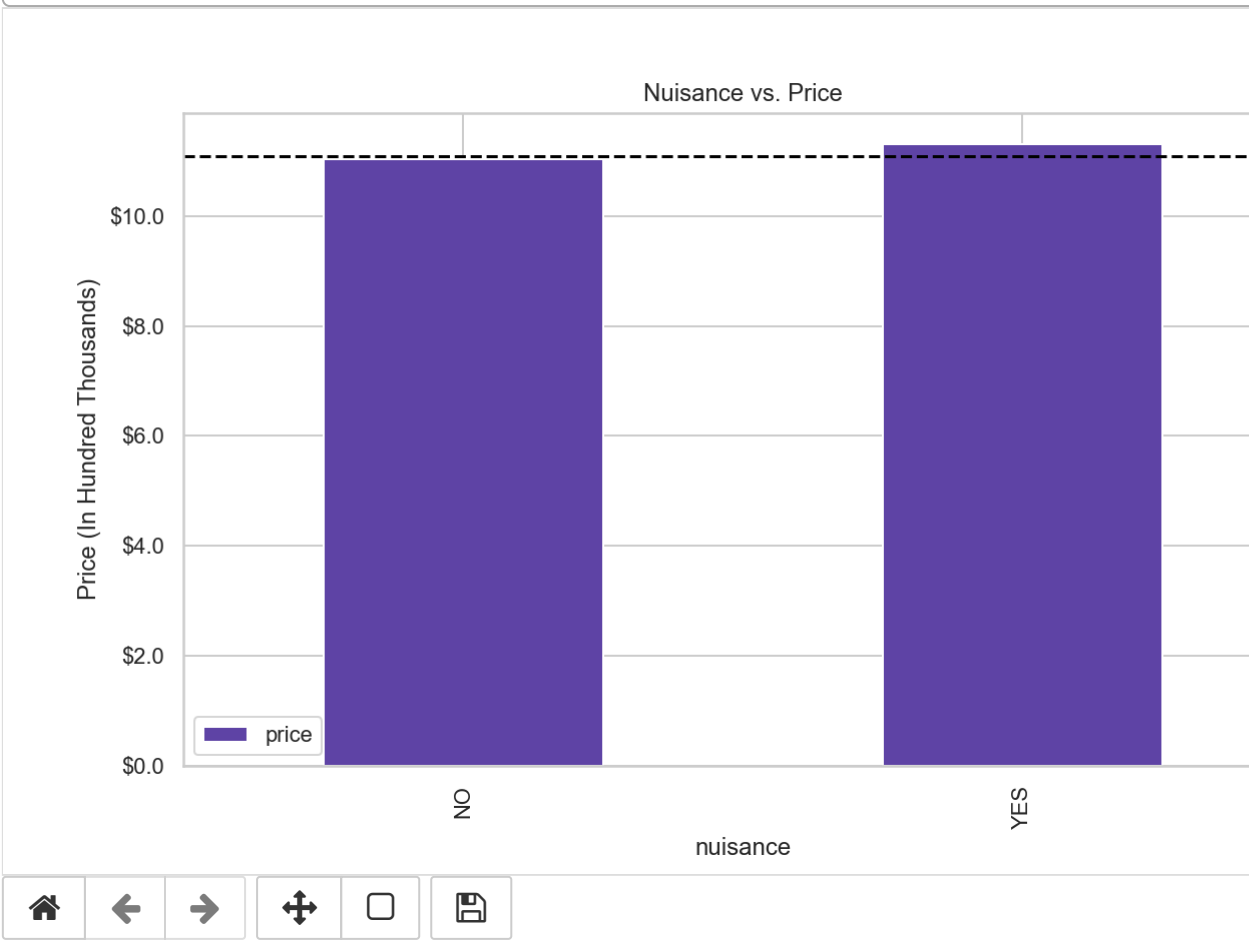
Interpretation:

This model is statistically significant and explains about 0.5% of the variance in price Compared to

'nuisance' feature:

In [49]:
```
fig, ax = plt.subplots(figsize=(10,6))
data1.groupby("nuisance").mean().plot.bar(y="price", ax=ax).set(ylabel=
ax.axhline(y=data1["price"].mean(), label="mean", color="black", linest
ylabels = ['${:,.1f}'.format(x) for x in ax.get_yticks()/100000]
ax.set_yticklabels(ylabels);
```

Figure 17




```
<ipython-input-49-572d1115eff5>:5: UserWarning: FixedFormatter should onl
y be used together with FixedLocator
  ax.set_yticklabels(ylabels);
```

In [50]:
```
nuisance_X = pd.get_dummies(X_nu, columns=["nuisance"], drop_first=True
nuisance_X
```

Out[50]:

	nuisance_YES
0	0
1	1
2	0
3	0
4	1
...	...
30150	0
30151	0
30152	1
30153	0
30154	0

30155 rows × 1 columns

In [51]:
```
nuisance_model = sm.OLS(y, sm.add_constant(nuisance_X))
nu_results = nuisance_model.fit()

print(nu_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.000
Model:                            OLS   Adj. R-squared:                  0.000
Method:                 Least Squares   F-statistic:                     4.021
Date:                Fri, 02 Jun 2023   Prob (F-statistic):             0.0449
Time:                        13:50:24   Log-Likelihood:            -4.560e+05
No. Observations:               30155   AIC:                         9.122e+05
Df Residuals:                   30153   BIC:                         9.122e+05
Df Model:                           1
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const         1.104e+06   5681.127    194.288      0.000    1.09e+06    1.11e+06
nuisance_YES  2.727e+04   1.36e+04      2.005      0.045     614.521    5.39e+04
==============================================================================
Omnibus:                    37941.688   Durbin-Watson:                   1.914
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          14348330.387
Skew:                           6.598   Prob(JB):                         0.00
Kurtosis:                     109.045   Cond. No.                         2.73
==============================================================================

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

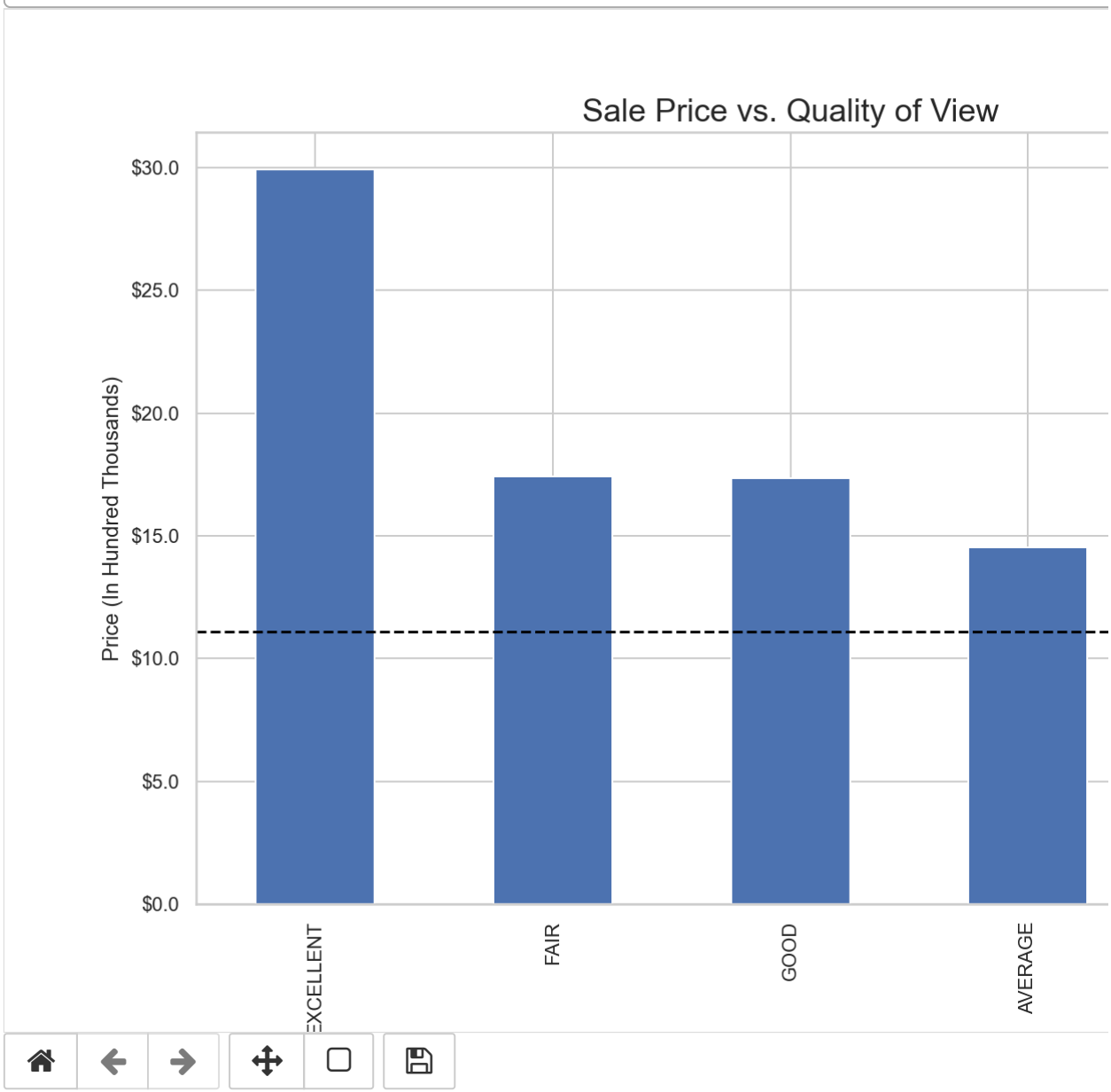
Interpretation:

This model is not statistically significant and it explains 0% of the variance in price. This indicates

'view' feature:

In [52]:
```
fig, ax = plt.subplots(figsize=(12,8))

sns.set(font_scale=1.5)
#sns.barplot(data=data1, x='view', y='price')
data1.groupby("view").mean().sort_values("price",ascending=False).plot.
ax.axhline(y=data1["price"].mean(), label="mean", color="black", linest
ylabels = ['${:,.1f}'.format(x) for x in ax.get_yticks()/100000]
ax.set_yticklabels(ylabels);
```

Figure 18




```
<ipython-input-52-94cbc99bedda>:8: UserWarning: FixedFormatter should onl
y be used together with FixedLocator
  ax.set_yticklabels(ylabels);
```

In [53]:
```
view_X = pd.get_dummies(X_vw, columns=["view"], drop_first=True)
view_X
```

Out[53]:

	view_EXCELLENT	view_FAIR	view_GOOD	view_NONE
0	0	0	0	1
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	1
...	...	...	...	...
30150	0	0	0	1
30151	0	1	0	0
30152	0	0	0	1
30153	0	0	0	1
30154	0	0	0	1

In [54]:
```
view_model = sm.OLS(y, sm.add_constant(view_X))
vw_results = view_model.fit()

print(vw_results.summary())
```

```
                            OLS Regression Results
==============================================================================
=====
Dep. Variable:                  price   R-squared:
0.117
Model:                            OLS   Adj. R-squared:
0.117
Method:                 Least Squares   F-statistic:
1001.
Date:                Fri, 02 Jun 2023   Prob (F-statistic):
0.00
Time:                        13:50:25   Log-Likelihood:             -4.542
1e+05
No. Observations:               30155   AIC:                          9.08
4e+05
Df Residuals:                   30150   BIC:                          9.08
5e+05
Df Model:                           4
Covariance Type:            nonrobust
```

Interpretation:

Of the categorical variables, the one that appears to be the best predictors of price is **view**.

Regression Results

Creating a multiple regression model with sqft_living and view:

Since 'view' is the categorical variable that seems to be the best predictor of price, I am going to add it to a model with sqft_living to see if it is an improvement to the baseline.

In [55]:
```
multi_x = data1[['sqft_living', 'view']]
```

In [56]:
```
multi_x_forvis = pd.get_dummies(multi_x, columns=["view"]).drop('sqft_l
multi_x_forvis
```

Out[56]:

	view_AVERAGE	view_EXCELLENT	view_FAIR	view_GOOD	view_NONE
0	0	0	0	0	1
1	1	0	0	0	0
2	1	0	0	0	0
3	1	0	0	0	0
4	0	0	0	0	1
...	...	...	...	...	...
30150	0	0	0	0	1
30151	0	0	1	0	0
30152	0	0	0	0	1
30153	0	0	0	0	1
30154	0	0	0	0	1

30155 rows × 5 columns

In [57]:
```
multi_x = pd.get_dummies(multi_x, columns=["view"], drop_first=True)
multi_x
```

Out[57]:

	sqft_living	view_EXCELLENT	view_FAIR	view_GOOD	view_NONE
0	1180	0	0	0	1
1	2770	0	0	0	0
2	2880	0	0	0	0
3	2160	0	0	0	0
4	1120	0	0	0	1
...	...	...	...	...	...
30150	1910	0	0	0	1
30151	2020	0	1	0	0
30152	1620	0	0	0	1
30153	2570	0	0	0	1
30154	1200	0	0	0	1

30155 rows × 5 columns

Multiple linear regression results:

In [58]:

```
multi_model = sm.OLS(y, sm.add_constant(multi_x))
multi_results = multi_model.fit()

print(multi_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.416
Model:                            OLS   Adj. R-squared:                  0.416
Method:                 Least Squares   F-statistic:                     4289.
Date:                Fri, 02 Jun 2023   Prob (F-statistic):               0.00
Time:                        13:50:25   Log-Likelihood:            -4.480e+05
No. Observations:               30155   AIC:                         8.960e+05
Df Residuals:                   30149   BIC:                         8.961e+05
Df Model:                           5
Covariance Type:            nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
const          1.48e+05   1.89e+04      7.850      0.000    1.11e+05    1.85e+05
sqft_living     518.4362      4.179    124.065      0.000     510.246     526.627
view_EXCELLENT 1.196e+06   3.32e+04     36.039      0.000    1.13e+06    1.26e+06
view_FAIR      2.259e+05   4.88e+04      4.631      0.000     1.3e+05    3.22e+05
view_GOOD      8.875e+04    2.8e+04      3.173      0.002    3.39e+04    1.44e+05
view_NONE     -1.824e+05   1.63e+04    -11.163      0.000   -2.14e+05    -1.5e+05
==============================================================================
Omnibus:                    41848.759   Durbin-Watson:                   1.850
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          4184506.297
Skew:                           7.606   Prob(JB):                         0.00
Kurtosis:                     184.859   Cond. No.                     3.0e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large,  3e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [59]:
```
1 multi_results.pvalues
```

Out[59]:
```
const              4.313516e-15
sqft_living        0.000000e+00
view_EXCELLENT    1.714013e-278
view_FAIR          3.662044e-06
view_GOOD          1.512205e-03
view_NONE          7.013998e-29
dtype: float64
```

Interpretation:

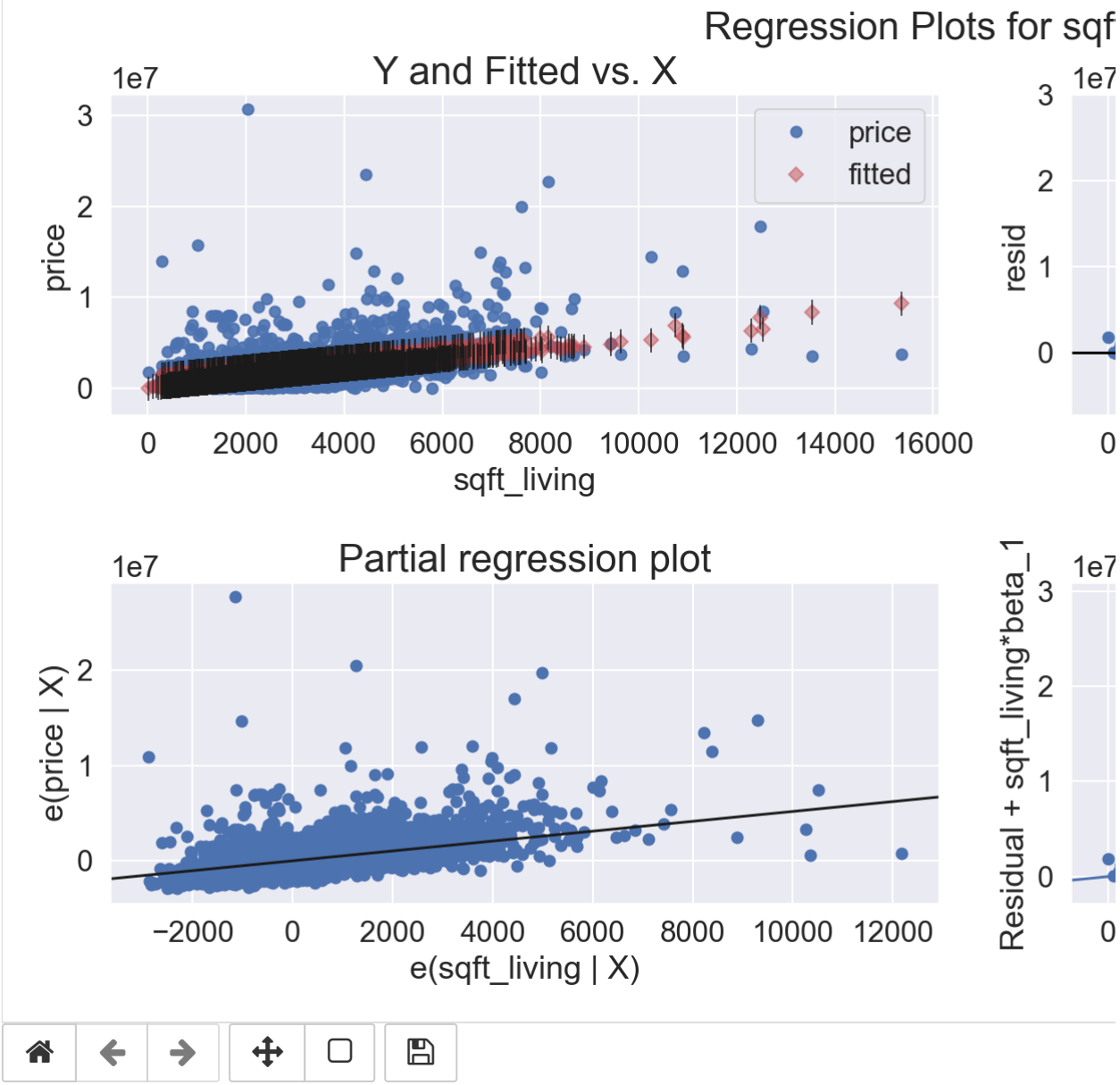
This model is statistically significant and explains about 41.6% of the variance in price, which is an improvement from the baseline model that only explained about 37%.

For each increase by 1 square foot of living space, we expect to see an increase in price of about 521 dollars. Compared to a house with an average view, for a house with an excellent view we see an associated increase in price of about of about 1,196,00 dollars. For a house with a good view, we see an associated increase in price of aboutof about 88,750 dollars. For a house with a fair view, we see an associated increase in price of about 225,900 dollars. For a house with no view, we see an associated decrease in price of about 182,400 dollars.

Multiple Linear Regression Visualization:

In [60]:
```
fig = plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(multi_results, "sqft_living", fig=fig)
plt.show()
```

Figure 19



Since waterfront appeared to be the second most predictive categorical feature of price, I am interested to see if it improves this model as well.

In [61]:
```
multi_x2 = data1[['sqft_living', 'view', 'waterfront']]
multi_x2 = pd.get_dummies(multi_x2, columns=["waterfront", "view"], dro

multi_x2
```

Out[61]:

	sqft_living	waterfront_YES	view_EXCELLENT	view_FAIR	view_GOOD	view_NONE
0	1180	0	0	0	0	1
1	2770	0	0	0	0	0
2	2880	0	0	0	0	0
3	2160	0	0	0	0	0
4	1120	0	0	0	0	1
...	...	...	...	...	...	...
30150	1910	0	0	0	0	1
30151	2020	0	0	1	0	0
30152	1620	0	0	0	0	1
30153	2570	0	0	0	0	1
30154	1200	0	0	0	0	1

30155 rows × 6 columns

In [62]:

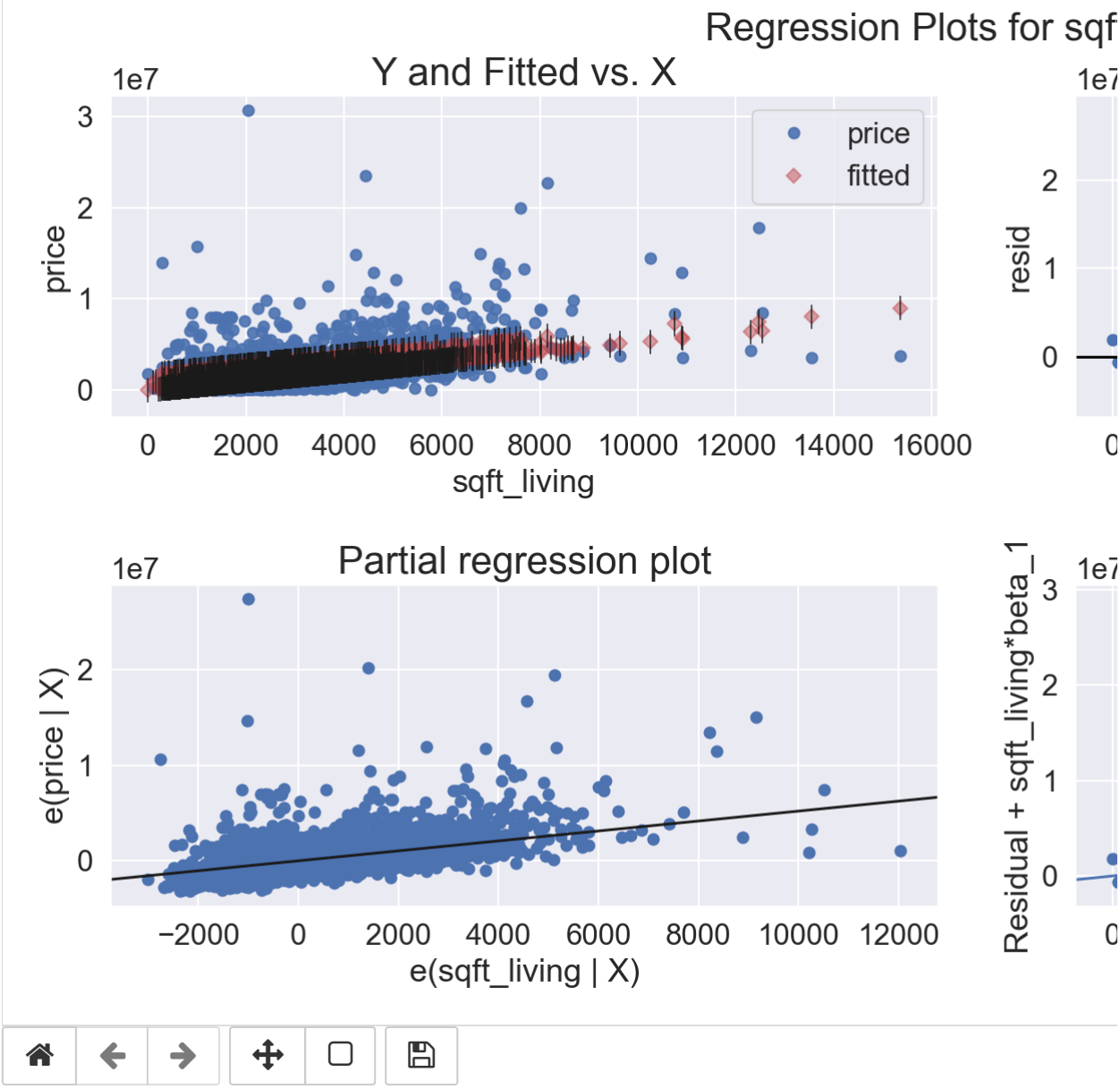
```
model_multi2 = sm.OLS(y, sm.add_constant(multi_x2))
multi2_results = model_multi2.fit()

print(multi2_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.423
Model:                            OLS   Adj. R-squared:                  0.423
Method:                 Least Squares   F-statistic:                     3686.
Date:                Fri, 02 Jun 2023   Prob (F-statistic):               0.00
Time:                        13:50:26   Log-Likelihood:            -4.478e+05
No. Observations:               30155   AIC:                         8.956e+05
Df Residuals:                   30148   BIC:                         8.957e+05
Df Model:                           6
Covariance Type:            nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
const          1.029e+05   1.89e+04      5.452      0.000    6.59e+04     1.4e+05
sqft_living      521.1201      4.154    125.447      0.000     512.978     529.262
waterfront_YES  7.067e+05   3.57e+04     19.807      0.000    6.37e+05    7.77e+05
view_EXCELLENT  8.765e+05   3.67e+04     23.865      0.000    8.05e+05    9.49e+05
view_FAIR       2.511e+05   4.85e+04      5.179      0.000    1.56e+05    3.46e+05
view_GOOD       5.932e+04   2.78e+04      2.131      0.033    4757.237    1.14e+05
view_NONE       -1.44e+05   1.64e+04     -8.809      0.000   -1.76e+05   -1.12e+05
==============================================================================
Omnibus:                    41151.910   Durbin-Watson:                   1.847
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         39162635.482
Skew:                           7.365   Prob(JB):                         0.00
Kurtosis:                     178.932   Cond. No.                     3.0e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large,  3e+04. This might indicate that there
```

```
are
strong multicollinearity or other numerical problems.
```

In [63]:
```
fig = plt.figure(figsize=(15,8))
sm.graphics.plot_regress_exog(multi2_results, "sqft_living", fig=fig)
plt.show()
```

Figure 20



These plots look about the same and since this model explains 42.3% of the variance, this model may be an improvement to the baseline and to the first multiple linear regressions model.

MAE for interpretability of models:

MAE for baseline:

In [64]:
```
mae = baseline_results.resid.abs().sum() / len(y)
mae
```

Out[64]:
```
396335.99168420106
```

Our baseline model is off by about $396,336 in a given prediction. This is pretty high for housing prices.

MAE for multiple linear regression models:

In [65]:
```
mae = multi_results.resid.abs().sum() / len(y)
mae
```

Out[65]:
```
388646.59853549825
```

The first multiple linear regression model is off by about $388,646.60 in a given prediction.

In [66]:
```
mae = multi2_results.resid.abs().sum() / len(y)
mae
```

Out[66]:
```
389498.98261728266
```

The second multiple linear regression model is off by about $389,498.98 in a given prediction. This is more than the first mutiple linear regression model so this model is probably not the best model afterall.

The model with the most improvement overall to the baseline model is the multiple linear regression model that included both sqft_living and view. This model had a higher R-squared (adjusted) value than the baseline and it had a lower MAE (mean absolute error) value.

Conclusion

The models created show that the features that are most predictive of house sale price are the square feet of living space and the quality of the view. If the stakeholder is to only go off of this information, those are the two factors that they should look for when deciding which houses to buy in order to flip and make the highest possible profits.

Limitations:

As can be seen in the QQ plot of the target variable 'price', the model becomes unreliable more than plus or minus 2 standard deviations away from the mean, or outside of the price range of 684,205.75 and 2,901,227.43 dollars. The best model that was found still only explained 41.6% of the variance in price and was still estimated to be off by about 388,646.60 dollars on a given predicition, which is a pretty high number. Additionally, this dataset does not include information on certain housing features that would likely be even more related to sale price, such as the subdivision that the house is in.

Next Steps:

In order to get a better idea of the true best predictors of housing sale prices, further analysis should be conducted on more datasets that include more features, such as housing subdivision.