King County Housing Analysis Project

- Student name: Bella Scribner
- Student pace: Flex
- Scheduled project review date/time: September 19, 2023 7:00am
- Instructor name: Morgan Jones
- Blog post URL: https://datascienceprojectsandmore.blogspot.com/ (https://datascienceprojectsandmore.blogspot.com/)

The Business + Project

A non-for-profit addiction treatment center is looking to expand their services to offer half-way houses or sober living homes to those who complete the rehabilitation in-patient treatment.

These sober living homes are fundamental in the psychology of recovery. Most patients who come out of a rehab facility have bodies and minds still in shock from detox, or do not have the skills to function in society without the crutch of their addiction. Half-way homes are meant to be a safe space for people in recovery to ease their way back into the real world, giving them the proper amount of time, support, and space to readjust. This transition phase greatly reduces the risk of relapse.

The center is inquiring about predictions on how expensive acquiring these houses would be in order to set fundraising and budgeting goals. Based on criteria from state regulators, the homes must include a functioning sewer system, kitchen, and heating system (due to the fact people in these homes may be on parole). Other than these regulations, the center is interested in predictions based on the size of homes – the larger the home the more people they can accommodate, including live-in staff.

In this project, we go through the given dataset and utilize linear regression to create a suitable model that predicts the sale price of any given home in King County using applicable variables.

The Data

The data for this project was provided and includes data on homes sold in King County, Washington.

```
In [1]:  import pandas as pd
         df = pd.read_csv('data/kc_house_data.csv')
```

```
In [2]:  df.head(3)
```

```
Out[2]:
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	gr
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 rows × 25 columns

In [3]:
```
df.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]:
```
df.describe()
```

Out[4]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	fl
count	3.015500e+04	3.015500e+04	30155.000000	30155.000000	30155.000000	3.015500e+04	30155.00
mean	4.538104e+09	1.108536e+06	3.413530	2.334737	2112.424739	1.672360e+04	1.54
std	2.882587e+09	8.963857e+05	0.981612	0.889556	974.044318	6.038260e+04	0.56
min	1.000055e+06	2.736000e+04	0.000000	0.000000	3.000000	4.020000e+02	1.00
25%	2.064175e+09	6.480000e+05	3.000000	2.000000	1420.000000	4.850000e+03	1.00
50%	3.874011e+09	8.600000e+05	3.000000	2.500000	1920.000000	7.480000e+03	1.50
75%	7.287100e+09	1.300000e+06	4.000000	3.000000	2619.500000	1.057900e+04	2.00
max	9.904000e+09	3.075000e+07	13.000000	10.500000	15360.000000	3.253932e+06	4.00

In [5]:
```
df['sqft_living'].sort_values().head()
```

Out[5]:
```
14977      3
28816    100
28815    170
25133    210
8694     260
Name: sqft_living, dtype: int64
```

According to King County Occupancy Standards, stated in Ordinance 12560, homes must have a minimum square foot living area of 120 square feet. That aside, no home will have only 3 square feet of living error, which means this entry is most likely an error. Let's go ahead and drop this entry as well as the one home that does not meet King County standards.

In [6]:
```
df = df[df['sqft_living'] >= 120]
```

Descriptions on all of the columns was provided as well. Below are the descriptions of some relevant columns:

Column	Description
`price`	Sale price of the home (target)
`sqft_living`	Square footage of living space in the home
`bathrooms`	Number of bathrooms
`greenbelt`	Whether the house is adjacent to a green belt (an area of open land around a city, on which building is restricted)
`waterfront`	Whether the house is on a waterfront (including lake, river/slough waterfronts)
`nuisaance`	Whether the house has traffic noise or other recorded nuisances
`view`	Quality of view from house
`condition`	How good the overall condition of the house is. Related to maintenance of house
`grade`	Overall grade of the house. Related to the construction and design of the house
`heat_source`	Heat source for the house
`sewer_system`	Sewer system for the house
`address`	The street address
`lat`	Latitude coordinate of the house
`long`	Longitude coordinate of the house

The business had some restrictions on heat source and sewer systems, let's investigate that to see if any houses need to be noted or removed from our modeling.

In [7]:
```
df['heat_source'].value_counts()
```

Out[7]:
```
Gas                  20582
Electricity           6465
Oil                   2899
Gas/Solar               93
Electricity/Solar       59
Other                   20
Oil/Solar                4
Name: heat_source, dtype: int64
```

In [8]:
```
df['sewer_system'].value_counts()
```

Out[8]:
```
PUBLIC                 25777
PRIVATE                 4354
PRIVATE RESTRICTED         6
PUBLIC RESTRICTED          3
Name: sewer_system, dtype: int64
```

It does not look like this will impact what homes the business can consider to purchase. These are the only two columns with missing data. However, since we are not interested in using these variables to predict house sale prices, we do not need to remove the rows with these missing data.

Let's get a quick view of some of the other columns.

In [9]:
```
df['waterfront'].value_counts()
```

Out[9]:
```
NO     29635
YES      518
Name: waterfront, dtype: int64
```

In [10]:
```
df['greenbelt'].value_counts()
```

Out[10]:
```
NO     29380
YES      773
Name: greenbelt, dtype: int64
```

In [11]:
```
df['nuisance'].value_counts()
```

Out[11]:
```
NO     24892
YES     5261
Name: nuisance, dtype: int64
```

In [12]:
```
df['view'].value_counts()
```

Out[12]:
```
NONE         26588
AVERAGE       1914
GOOD           878
EXCELLENT      553
FAIR           220
Name: view, dtype: int64
```

```
In [13]: df['condition'].value_counts()
```

```
Out[13]: Average      18546
         Good          8054
         Very Good     3259
         Fair           229
         Poor            65
         Name: condition, dtype: int64
```

```
In [14]: df['grade'].value_counts()
```

```
Out[14]: 7 Average        11697
         8 Good            9410
         9 Better          3805
         6 Low Average     2858
         10 Very Good      1371
         11 Excellent       406
         5 Fair             393
         12 Luxury          122
         4 Low               51
         13 Mansion          24
         3 Poor              13
         2 Substandard        2
         1 Cabin              1
         Name: grade, dtype: int64
```

```
In [15]: df['address'][0]
```

```
Out[15]: '2102 Southeast 21st Court, Renton, Washington 98055, United States'
```

```
In [16]: df[['lat', 'long']]
```

Out[16]:

	lat	long
0	47.461975	-122.19052
1	47.711525	-122.35591
2	47.502045	-122.22520
3	47.566110	-122.29020
4	47.532470	-122.07188
...	...	...
30150	47.664740	-122.32940
30151	47.565610	-122.38851
30152	47.610395	-122.29585
30153	47.449490	-122.18908
30154	47.435840	-122.32634

30153 rows × 2 columns

Map of the Houses

Let's get a birds-eye view of King County and the houses sold in our data set.

In [17]:
```
#pip install geopandas
```

In [18]:
```
import geopandas as gpd
from shapely.geometry import Point, Polygon
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')
```

In [19]:
```
df_lat_long = df[['price','lat', 'long']]
```

In [20]:
```
# import Kings County Map
KC_map = gpd.read_file('data/map_data/map.shp')
```

In [21]:
```
# zip lat and long
lat_long = [Point(xy) for xy in zip(df_lat_long['long'], df_lat_long['lat'])]

# create GeoPandas dataframe
geo_df = gpd.GeoDataFrame(df_lat_long, crs='EPSG:4326', geometry=lat_long)
```

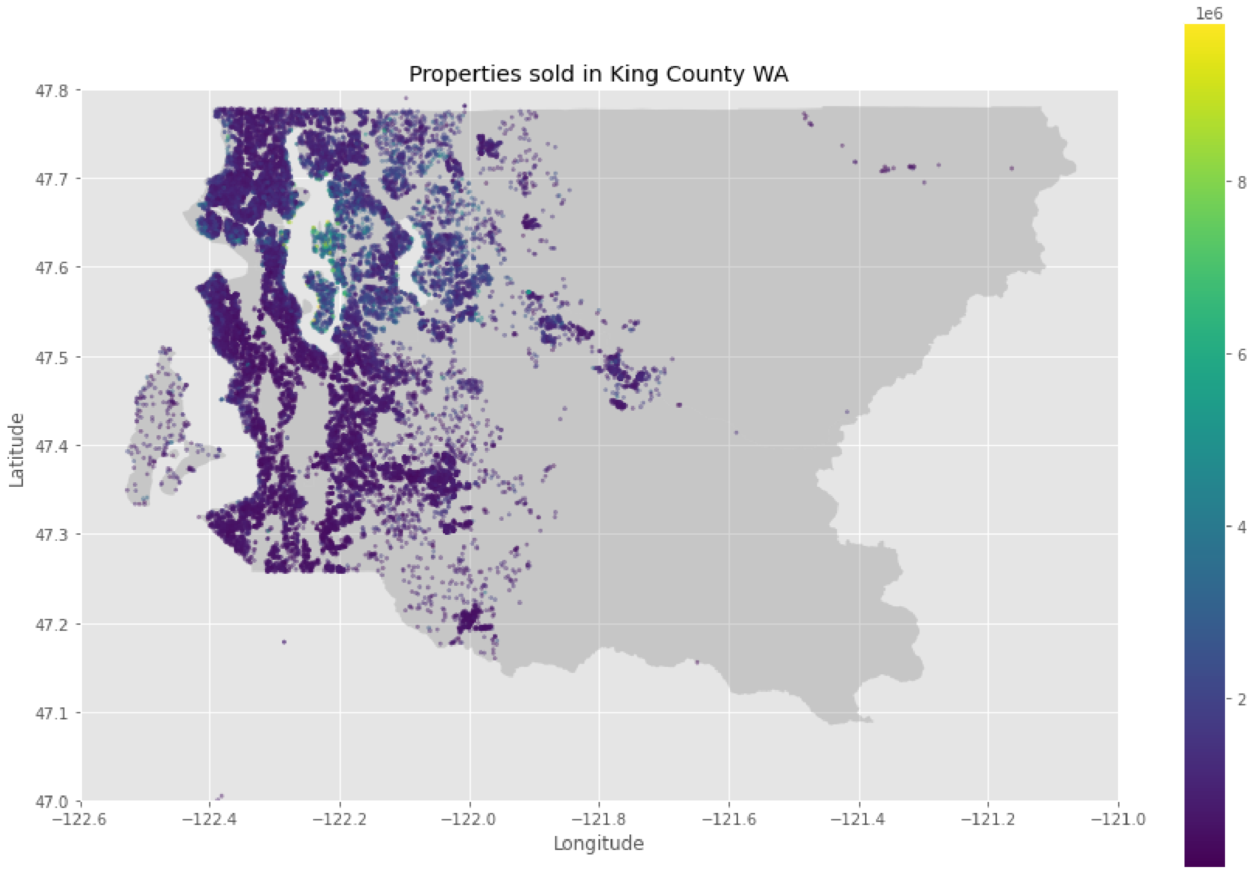
In [22]:
```
geo_df.head()
```

Out[22]:

	price	lat	long	geometry
0	675000.0	47.461975	-122.19052	POINT (-122.19052 47.46198)
1	920000.0	47.711525	-122.35591	POINT (-122.35591 47.71153)
2	311000.0	47.502045	-122.22520	POINT (-122.22520 47.50205)
3	775000.0	47.566110	-122.29020	POINT (-122.29020 47.56611)
4	592500.0	47.532470	-122.07188	POINT (-122.07188 47.53247)

In [23]:
```
geo_df['price'].sort_values(ascending=False)
geo_df = geo_df[geo_df['price'] < 10000000]
```

In [24]:
```
fig, ax = plt.subplots(figsize=(15,10))

# plot lat and long
KC_map.plot(ax=ax, color='grey', alpha=.3)
geo_df.plot(column='price', ax=ax, alpha=.4, legend=True, markersize=5)

# set lat and long boundaries for map display
plt.xlim(-122.6, -121.0)
plt.ylim(47.0, 47.8)

ax.set_ylabel('Latitude')
ax.set_xlabel('Longitude')
ax.set_title('Properties sold in King County WA');
```



Investigation of Address Information

Let's investigate the address column to ensure that all of the houses in our data set are indeed from King County, Washington.

In [25]:
```
df['address'][0]
```

Out[25]:
```
'2102 Southeast 21st Court, Renton, Washington 98055, United States'
```

```
In [26]:  df['city'] = df['address'].map(lambda x: x.split(',')[1].strip())
          df['city'].value_counts().head(3)
```

```
Out[26]:  Seattle    9368
          Renton     1946
          Kent       1583
          Name: city, dtype: int64
```

```
In [27]:  df['state'] = df['address'].map(lambda x: x.split(',')[2].strip()[:10])
          df['state'].value_counts()
```

```
Out[27]:  Washington    29239
          Nebraska 6      159
          New Jersey       76
          New York 1       66
          Minnesota        64
                         ...
          Ohio 45039        1
          New Mexico        1
          Utah 84115        1
          Texas 7620        1
          Ohio 45856        1
          Name: state, Length: 82, dtype: int64
```

Aha! Not all of the houses sold were in the state of Washington. Let's remove any data points that are not from Washington State. Furthermore, we can pull in some information from King County to cross reference that all the towns and cities are in this county, and remove any data that are for homes not in this county.

```
In [28]:  df = df[df['state'] == 'Washington']
```

```
In [29]:  cities_towns = pd.read_excel('data/KC_cities_towns.xlsx')
          cities_towns.head()
```

Out[29]:

	Name
0	Algona
1	Ames Lake
2	Auburn
3	Baring
4	Barneston

```
In [30]:  df['city'].isin(cities_towns['Name']).value_counts()
```

```
Out[30]:  True     29196
          False       43
          Name: city, dtype: int64
```

In [31]:
```
df['city_in_county'] = df['city'].isin(cities_towns['Name'])
```

In [32]:
```
df = df[df['city_in_county'] == True]
```

In [33]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29196 entries, 0 to 30154
Data columns (total 28 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   id              29196 non-null  int64  
 1   date            29196 non-null  object 
 2   price           29196 non-null  float64
 3   bedrooms        29196 non-null  int64  
 4   bathrooms       29196 non-null  float64
 5   sqft_living     29196 non-null  int64  
 6   sqft_lot        29196 non-null  int64  
 7   floors          29196 non-null  float64
 8   waterfront      29196 non-null  object 
 9   greenbelt       29196 non-null  object 
 10  nuisance        29196 non-null  object 
 11  view            29196 non-null  object 
 12  condition       29196 non-null  object 
 13  grade           29196 non-null  object 
 14  heat_source     29168 non-null  object 
 15  sewer_system    29184 non-null  object 
 16  sqft_above      29196 non-null  int64  
 17  sqft_basement   29196 non-null  int64  
 18  sqft_garage     29196 non-null  int64  
 19  sqft_patio      29196 non-null  int64  
 20  yr_built        29196 non-null  int64  
 21  yr_renovated    29196 non-null  int64  
 22  address         29196 non-null  object 
 23  lat             29196 non-null  float64
 24  long            29196 non-null  float64
 25  city            29196 non-null  object 
 26  state           29196 non-null  object 
 27  city_in_county  29196 non-null  bool   
dtypes: bool(1), float64(5), int64(10), object(12)
memory usage: 6.3+ MB
```

Wonderful, we have cleaned up our dataset to include only houses within King County, Washington.

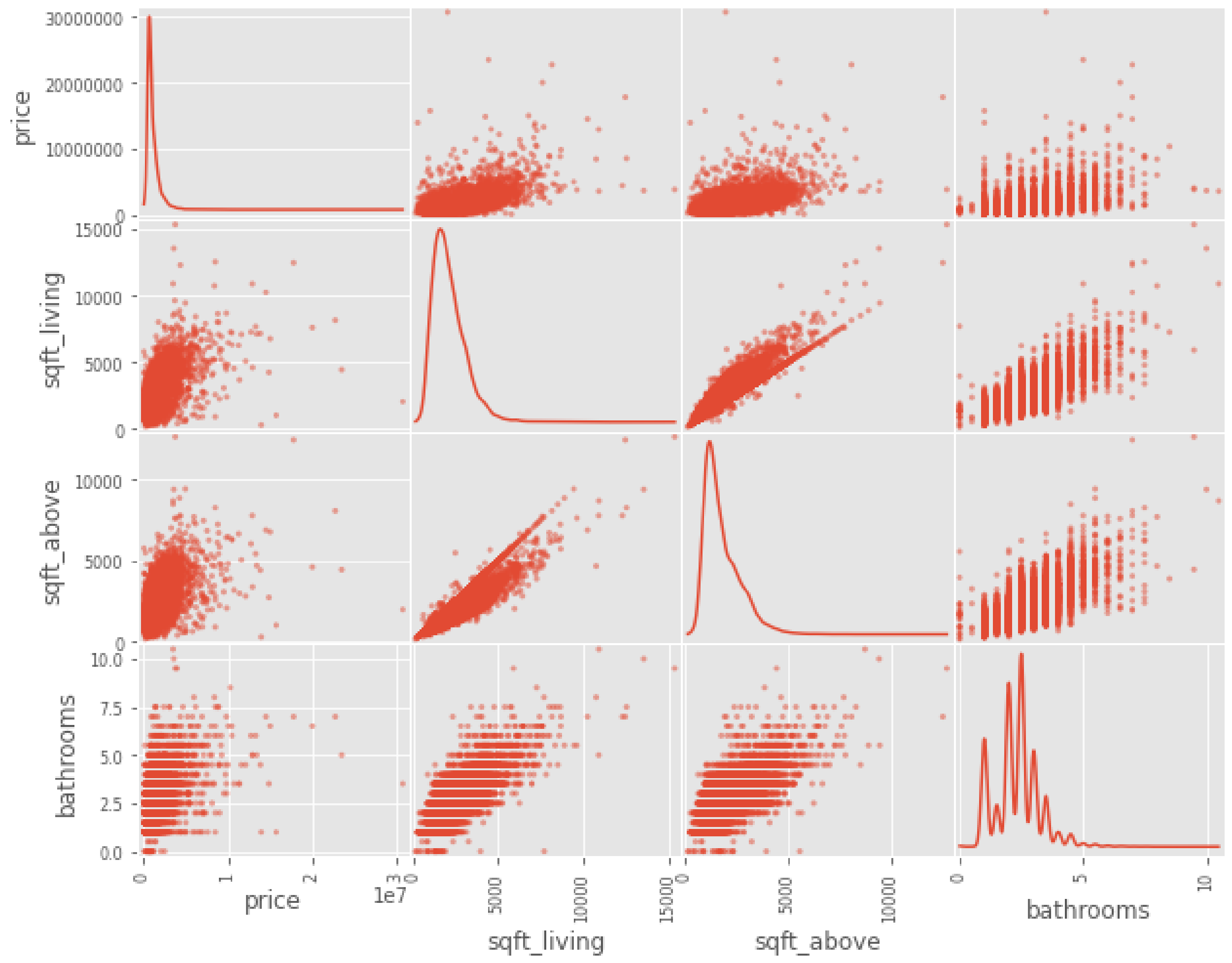
Exploration of Variables for Linear Regression

Linear regression is going to be the method of which we predict how expensive houses will be in King County. We want to include variables that will be good predictors of price.

```
In [34]: df.corr()['price'].map(abs).sort_values(ascending=False)
```

```
Out[34]: price             1.000000
         sqft_living       0.616729
         sqft_above        0.546329
         bathrooms         0.488072
         sqft_patio        0.317222
         lat               0.297789
         bedrooms          0.291110
         sqft_garage       0.267626
         sqft_basement     0.245993
         floors            0.200236
         yr_built          0.106358
         sqft_lot          0.085941
         yr_renovated      0.084953
         long              0.080313
         id                0.029471
         city_in_county         NaN
         Name: price, dtype: float64
```

```
In [35]: #fig, ax = plt.subplots(figsize=(15,10))
         pd.plotting.scatter_matrix(df[['price', 'sqft_living', 'sqft_above', 'bathrooms']],
```

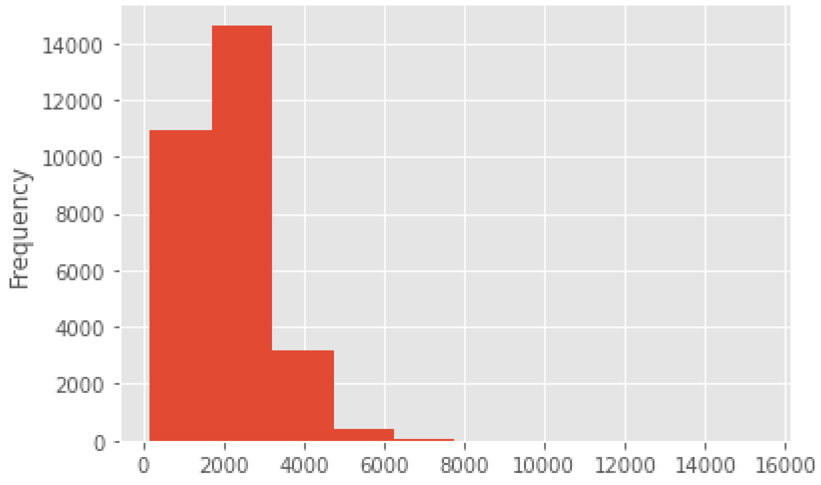


From the above, we can see that `sqft_living` , `bathrooms` , `sqft_above` , all seem promising. However, in order to avoid multicollinearity, it would be wise to not use both `sqft_living` and `sqft_above` (we can see the highly linear relationship between these two variables in the visual above).

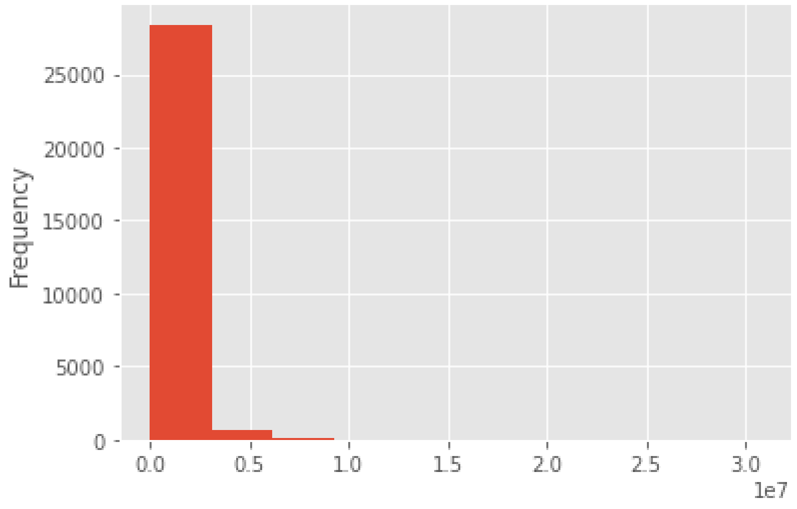
Seeing as `sqft_living` has a higher correlation, we will move forward with that variable.

For a simple linear regression model, which will our baseline model, we will use `sqft_living` as the lone independent variable. Let's take a peek at the distribution of this, the distribution of our target, `price`, and their interaction more closely.

```
In [36]:  df['sqft_living'].plot.hist();
```

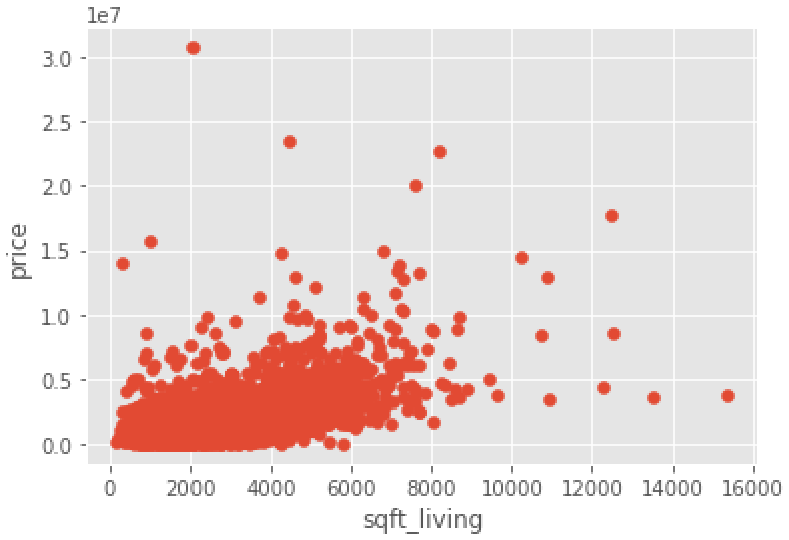


```
In [37]:  df['price'].plot.hist();
```



Both distributions look heavily right skewed. This might cause future issues with our linear regression model(s).

In [38]:
```
fig, ax = plt.subplots()
plt.scatter(df['sqft_living'], df['price'])
ax.set_xlabel('sqft_living')
ax.set_ylabel('price');
```



It appears we have a vaguely linear relationship between `sqft_living` and `price` , with a few outliers.

Model Iteration

Linear regression is a wonderful tool to create future predictions based on known, historical knowledge. Going through multiple iterations of models is an important process to ensure the best possible model is used to make these predictions.

Simple Linear Regression

We will start with a simple linear regression using `sqft_living` and `price` .

In [39]:
```
X_simple = df[['sqft_living']]
y = df['price']
```

```
In [40]:  import statsmodels.api as sm
          model_simple = sm.OLS(y, sm.add_constant(X_simple))
          results_simple = model_simple.fit()
          print(results_simple.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.380
Model:                            OLS   Adj. R-squared:                  0.380
Method:                 Least Squares   F-statistic:                 1.792e+04
Date:                Sun, 17 Sep 2023   Prob (F-statistic):               0.00
Time:                        13:52:52   Log-Likelihood:            -4.3457e+05
No. Observations:               29196   AIC:                         8.691e+05
Df Residuals:                   29194   BIC:                         8.692e+05
Df Model:                           1
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const       -9.184e+04   9894.630     -9.282      0.000   -1.11e+05   -7.24e+04
sqft_living   565.0633      4.221    133.866      0.000     556.790     573.337
==============================================================================
Omnibus:                    42272.896   Durbin-Watson:                   1.937
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         50165956.513
Skew:                           8.243   Prob(JB):                         0.00
Kurtosis:                     205.401   Cond. No.                     5.62e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 5.62e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
In [41]:  from sklearn.metrics import mean_absolute_error
          y_simple_pred = results_simple.predict(sm.add_constant(X_simple))
          mean_absolute_error(y, y_simple_pred)

Out[41]:  398617.79645136086
```

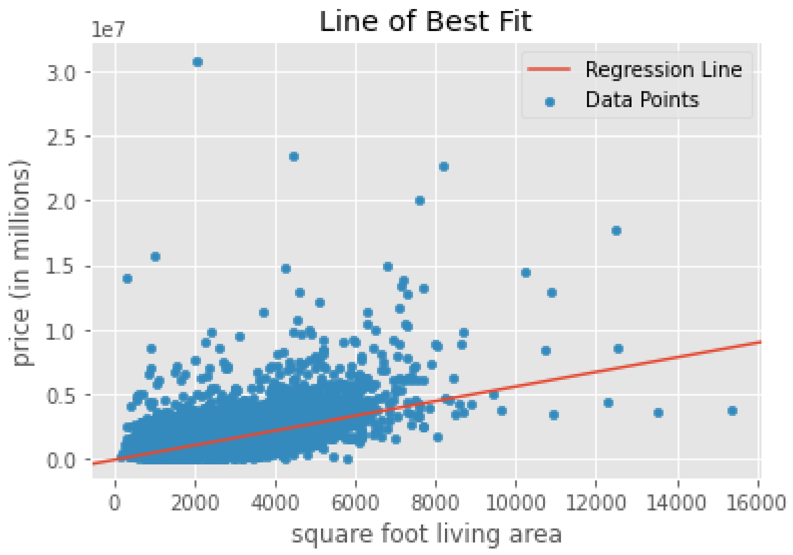
The r-squared statistic shows that this simple linear regression is explaining about 38% of the variance in sale price. The overall model as well as both coefficients are statistically significant. This model predicts that a house with zero square feet living area would sell for about negative $92k, and for each one square foot increase in living area, a house would become $565 more expensive. Lastly, the Mean Absolute Error shows that the average error is about +/- $400k.

Line of Best Fit

Let's plot the regression line against our known data.

In [42]:

```
fig, ax = plt.subplots()
df.plot.scatter(x='sqft_living', y='price', label='Data Points', ax=ax)
sm.graphics.abline_plot(model_results=results_simple, label="Regression Line", ax=a
ax.set_xlabel('square foot living area')
ax.set_ylabel('price (in millions)')
plt.title("Line of Best Fit")
ax.legend();
```

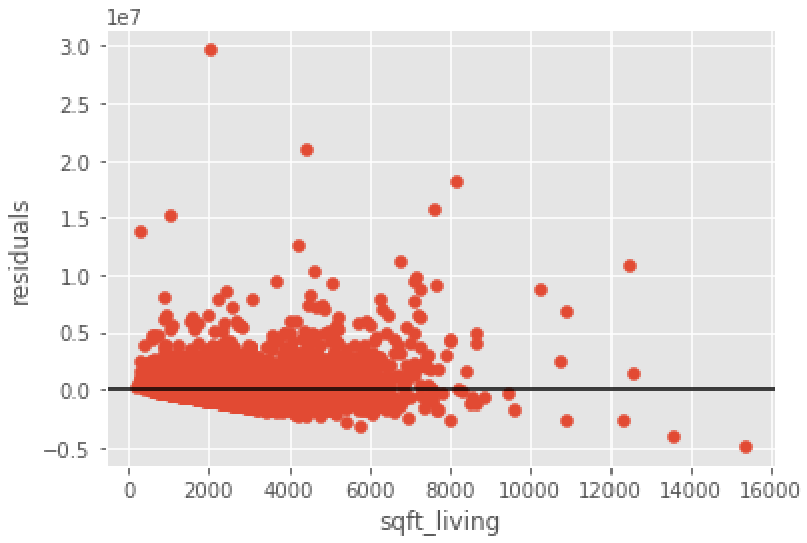


The regression line seems to follow the trend of our data fairly well. However, it can most definately be improved upon.

Residuals

The graph of residuals plots the variance not explained by our variable, we ideally want these to be spread out evenly and not have any discernible patterns.

In [43]:
```
fig, ax = plt.subplots()
ax.scatter(df['sqft_living'], results_simple.resid)
ax.axhline(y=0, color='black')
ax.set_xlabel('sqft_living')
ax.set_ylabel('residuals');
```



The graph of these residuals reinforce the idea that we are not meeting the assumptions of linear regression, as we saw from the results of the Omnibus and Jarque-Bera tests.

Linear Regression with Multiple Variables

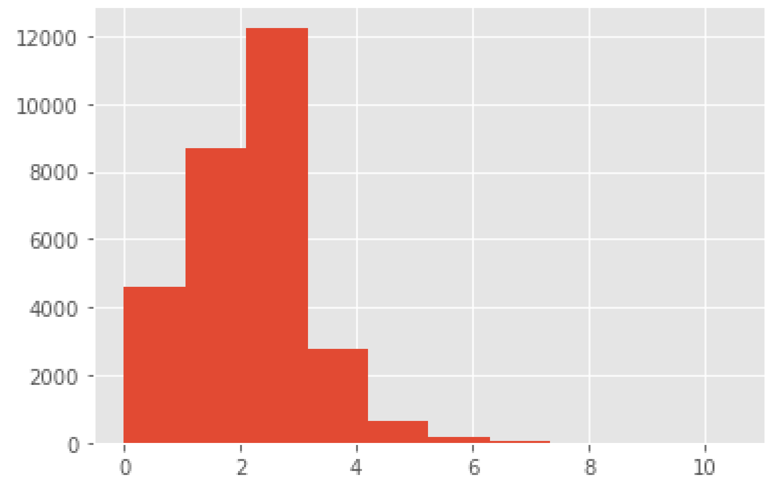
Let's see if we can improve upon our linear regression by including more variables. This will also allow us to report more usable information to the treatment center.

From a business standpoint, the variables `sqft_living` , `bathrooms` , `waterfront` , `greenbelt` , `nuisance` , `view` , `condition` , and `grade` would all be interesting. First and foremost, the larger the home, the more patients can utilize the facility at once. Additionally, the number of patients and/or staff acceptable on property will be highly dependent on the number of bathrooms in the home.

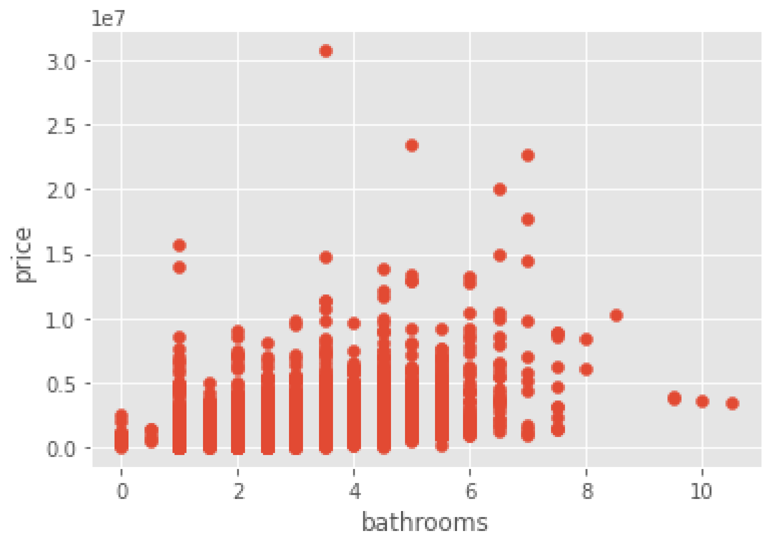
As a transition home for those newly sober, it could be beneficial to live in a home that was somehow connected to or near nature – for example having a waterfront view, or any sort of nice view, or being located near a park or some other type of greenbelt. On the other hand, if a home was located in an area with known disturbances, such as constant traffic noises, it could delay or impair patients’ success in recovering and transitioning (lack of sleep alone can cause this, thus introducing disturbances or nuisances that would increase the risk of this and other stressors would not be ideal). Lastly, the condition or grade of the home would be vital information prior to purchase. These homes need to be up to a certain standard prior to opening as half-way homes or sober living homes. If the overall state of the house is not up to par, then additional money must be set aside for renovations.

Moving forward, let's investigate these additional variables prior to creating our multi-variable linear regression.

In [44]:
```
df['bathrooms'].hist();
```



In [45]:
```
fig, ax = plt.subplots()
plt.scatter(df['bathrooms'], df['price'])
ax.set_xlabel('bathrooms')
ax.set_ylabel('price');
```



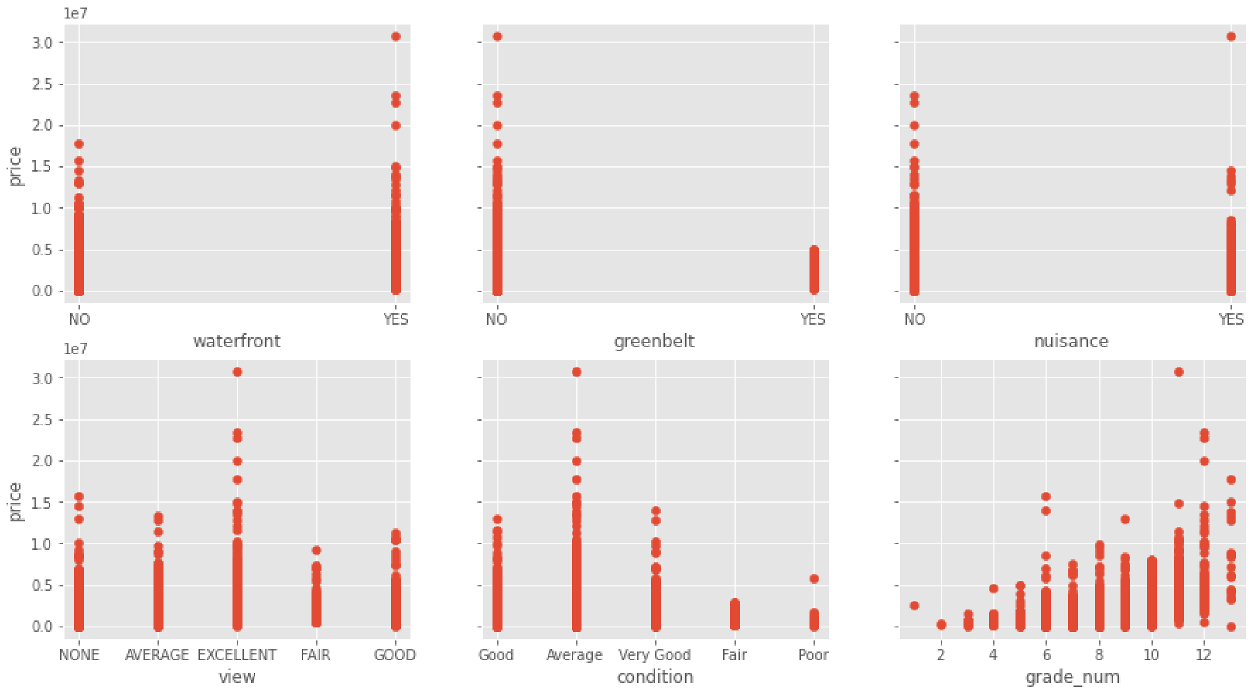
The relationship looks less linear than with our variable `sqft_living` , and again we see that the underlying data has a right skew which can potentially lead to problems with our linear regression moving forward.

The Categorical Variables

The categorical variables we would like to investigate further are `waterfront` , `greenbelt` , `nuisance` , `view` , `condition` , and `grade` .

In [46]:
```
# Grade has preset numerical data incorperated, let's create numerical data column
df['grade_num'] = df['grade'].map(lambda x: int(x[:2].strip()))
```

```
In [47]:  cat_vars = ['waterfront', 'greenbelt', 'nuisance', 'view', 'condition', 'grade_num'

          fig, axes = plt.subplots(ncols=3, nrows=2, figsize=(15,8), sharey=True)

          #Plot relationship between each predictor and target
          for i,cat in enumerate(cat_vars):
              # set proper axis
              row = i // 3
              col = i % 3
              ax = axes[row][col]

              #set x and y
              x = df[cat]
              y_graph = df['price']

              ax.scatter(x, y_graph)
              ax.set_xlabel(cat)
              if col == 0:
                  ax.set_ylabel('price')
```



Based on the visual above we see that:

- `waterfront` does not have much of a linear relationship, rather many data points are clustered between 0–1.5 million for both Yes and No
- `greenbelt` appears to have a negatively linear relationship, with prices decreasing when they are next to a greenbelt. However, it is important to not the underlying distribution -- the vast majority of homes in our dataset are not next to a greenbelt.
- `nuisance` has a very slight negative linear relationship, if the outlier in the Yes column is excluded.
- `view` is a good candidate to include in our model, fairly linear when ordering as None, Fair, Average, Good, Excellent.
- `condition` also has a quite linear relationship with price, ordering from Fair, Poor, Average, Good, Very Good.
  - With both `view` and `condition` we see spikes in Average as it is the most common rating for homes in the dataset.

- `grade` has an obviously linear relationship with price, we see more homes become pricier as the grade rating increases.

Now that we have investigated these variables a bit, let's go through some preprocessing to get them in a usable format for our regression!

In [48]:
```
df['grade'].value_counts()
```

Out[48]:
```
7 Average        11549
8 Good            8855
9 Better          3583
6 Low Average     2853
10 Very Good      1349
11 Excellent       402
5 Fair             392
12 Luxury          122
4 Low               51
13 Mansion          24
3 Poor              13
2 Substandard        2
1 Cabin              1
Name: grade, dtype: int64
```

Due to the lack of data points with lower grades, we will create a ‘Lower’ category that will encompass grades 1 through 3.

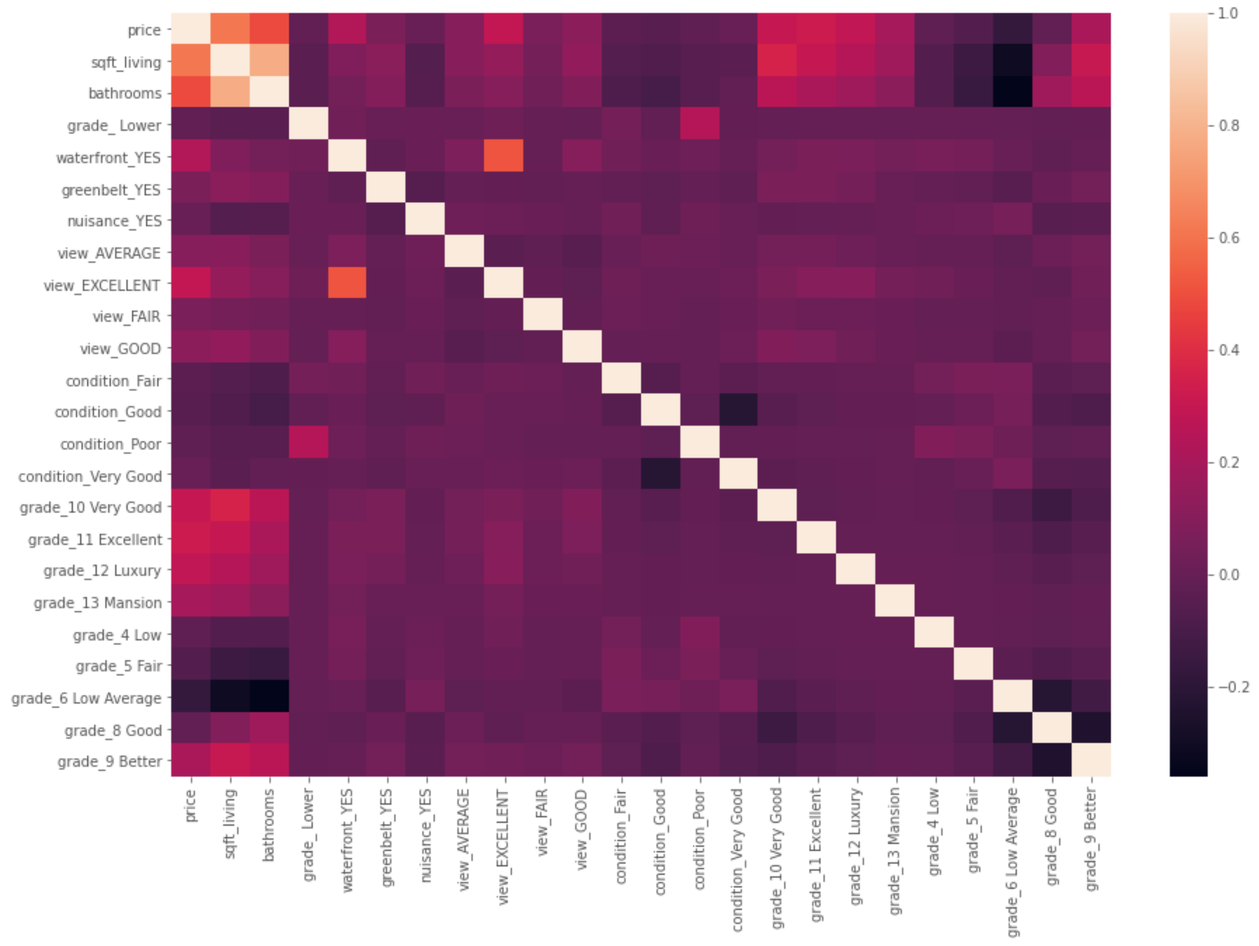
In [49]:
```
# Combine grades 1 through 3 for 'Lower' category
import numpy as np
mask = (df['grade'] == '1 Cabin') | (df['grade'] == '2 Substandard') | (df['grade']
df['grade_ Lower'] = np.where(mask, 1, 0)
df['grade_ Lower'].value_counts()
```

Out[49]:
```
0    29180
1       16
Name: grade_ Lower, dtype: int64
```

In [50]:
```
# Create DataFrame with 'dummied' categorical variables
cat_vars = ['waterfront', 'greenbelt', 'nuisance', 'view', 'condition', 'grade']
cat_vars_dummied = pd.get_dummies(df[cat_vars])

# Drop one of each column for each variable + add 'dummied' variables to dataset
df_a = cat_vars_dummied.drop(['waterfront_NO', 'view_NONE', 'greenbelt_NO', 'nuisar
                             'condition_Average', 'grade_3 Poor', 'grade_2 Substand

# create subset dataframe with only variables want to include in regression
df_b = df[['price', 'sqft_living', 'bathrooms', 'grade_ Lower']]
subset_df = pd.concat([df_b, df_a], axis=1)
```

Now that we have our encoded categorical variables and a subset of the data we are ready to work with, let's double check correlation.

In [51]:
```python
import seaborn as sns
fig = plt.figure(figsize=(15,10))
sns.heatmap(subset_df.corr(), figure=fig);
```



Based on the heatmap above, it might be a good decision to drop either `bathrooms` or `sqft_living` as they seem to have a high correlation and we would like to avoid multicollinearity. Seeing as `sqft_living` has a slightly higher correlation with our target, we will keep that variable.

In [52]:
```python
subset_df.drop('bathrooms', axis=1, inplace=True)
```

The Model

Now that we have processed our categorical variables, it is time to put it all together in our linear regression model!

In [53]:

```
X_multi = subset_df.drop('price', axis=1)
y = subset_df['price']
model_multi = sm.OLS(y, sm.add_constant(X_multi))
results_multi = model_multi.fit()
print(results_multi.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.517
Model:                            OLS   Adj. R-squared:                  0.517
Method:                 Least Squares   F-statistic:                     1419.
Date:                Sun, 17 Sep 2023   Prob (F-statistic):               0.00
Time:                        13:52:56   Log-Likelihood:            -4.3094e+05
No. Observations:               29196   AIC:                         8.619e+05
Df Residuals:                   29173   BIC:                         8.621e+05
Df Model:                          22
Covariance Type:            nonrobust
=================================================================================
======
                          coef    std err          t      P>|t|      [0.025
0.975]
---------------------------------------------------------------------------------
------
const                 2.51e+05   1.21e+04     20.829      0.000    2.27e+05    2.
75e+05
sqft_living           287.2454      5.731     50.125      0.000     276.013     2
98.478
grade_ Lower        -9.505e+04   1.61e+05     -0.589      0.556   -4.11e+05    2.
21e+05
waterfront_YES       6.964e+05    3.3e+04     21.135      0.000    6.32e+05    7.
61e+05
greenbelt_YES       -3.624e+04    2.3e+04     -1.576      0.115   -8.13e+04    88
37.464
nuisance_YES         7.477e+04   9730.883      7.683      0.000    5.57e+04    9.
38e+04
view_AVERAGE         1.189e+05   1.51e+04      7.878      0.000    8.93e+04    1.
48e+05
view_EXCELLENT       8.074e+05   3.21e+04     25.114      0.000    7.44e+05
8.7e+05
view_FAIR            3.341e+05   4.28e+04      7.814      0.000     2.5e+05    4.
18e+05
view_GOOD            1.287e+05    2.2e+04      5.848      0.000    8.56e+04    1.
72e+05
condition_Fair       -4.85e+04   4.19e+04     -1.157      0.247   -1.31e+05    3.
37e+04
condition_Good       5.682e+04   8542.617      6.651      0.000    4.01e+04    7.
36e+04
condition_Poor      -7.646e+04   8.12e+04     -0.942      0.346   -2.36e+05    8.
26e+04
condition_Very Good  1.403e+05   1.21e+04     11.624      0.000    1.17e+05    1.
64e+05
grade_10 Very Good   8.804e+05   2.15e+04     41.039      0.000    8.38e+05    9.
22e+05
grade_11 Excellent   1.723e+06   3.59e+04     47.952      0.000    1.65e+06    1.
79e+06
grade_12 Luxury      2.809e+06   6.15e+04     45.661      0.000    2.69e+06    2.
93e+06
grade_13 Mansion     4.391e+06   1.32e+05     33.138      0.000    4.13e+06    4.
65e+06
grade_4 Low         -6.176e+04   8.82e+04     -0.700      0.484   -2.35e+05    1.
11e+05
grade_5 Fair        -3.128e+04   3.25e+04     -0.962      0.336    -9.5e+04    3.
25e+04
grade_6 Low Average -3.674e+04   1.34e+04     -2.743      0.006    -6.3e+04   -1.
```

```
05e+04
grade_8 Good          1.19e+05   9417.579     12.639      0.000    1.01e+05    1.
37e+05
grade_9 Better       4.343e+05   1.39e+04     31.141      0.000    4.07e+05    4.
62e+05
==============================================================================
Omnibus:                    40303.254   Durbin-Watson:                   1.918
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         48547061.693
Skew:                           7.447   Prob(JB):                         0.00
Kurtosis:                     202.212   Cond. No.                     1.05e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.05e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [54]:
```
y_multi_pred = results_multi.predict(sm.add_constant(X_multi))
mean_absolute_error(y, y_multi_pred)
```

Out[54]:
```
351604.1585039296
```

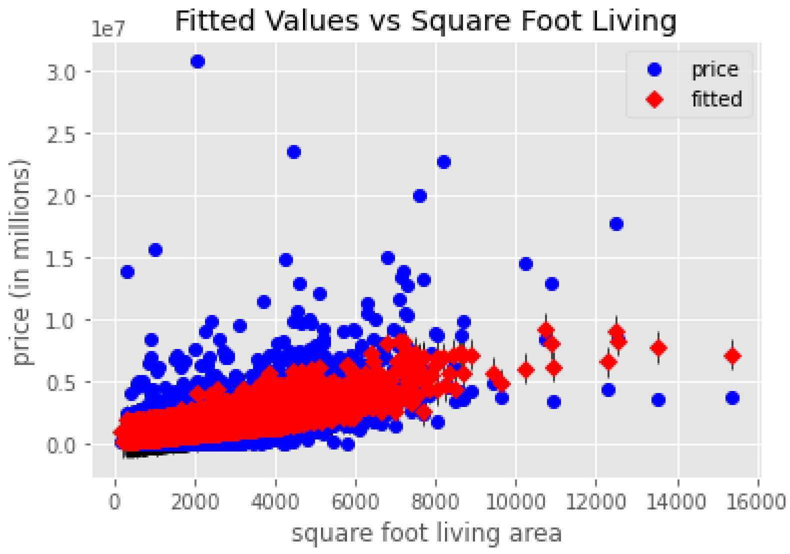
As we can see, this model is an improvement from our baseline simple linear regression. The r-squared has increased and now we are explaining about 52% of the variance in price. The overall model and most of the coefficients are statistically significant at a standard alpha of 0.05. We do see a few coefficients that are not statistically significant, including greenbelt, and the lower ratings of view, grade, and condition. Based on the Mean Absolute Error metric, we can see that the average error for the model is about +/- $352k, which is an improvement from our simple linear regression.

Lastly, from the Omnibus and Jarque-Bera tests, we can see that our model is not meeting all of the assumptions of linear regression.

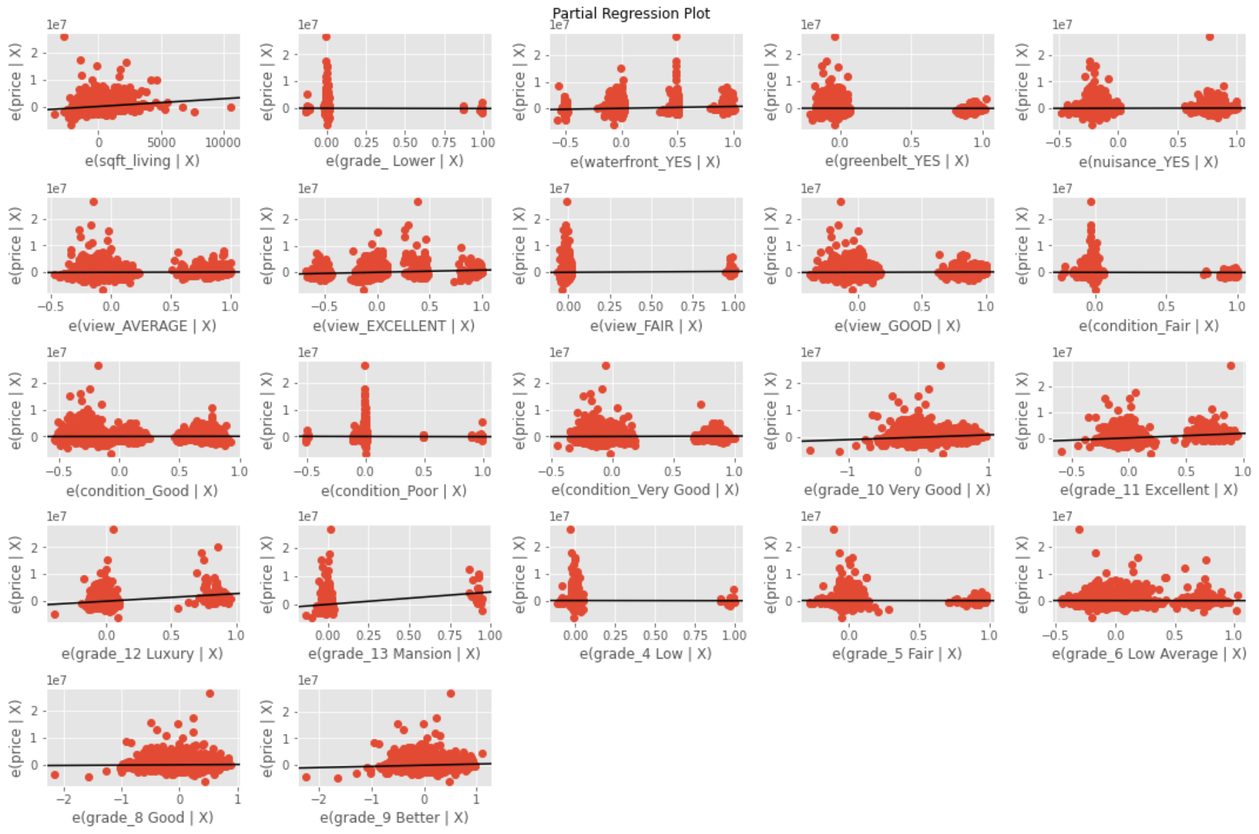
A Few Graphs of the Results

Below we see the 'best fit' line for the same variable we used in our simple linear regression baseline. The improvement is visible. Also displayed is a partial regression plot for all of the coefficients in our model.

In [55]:
```python
fig, ax = plt.subplots()
sm.graphics.plot_fit(results_multi, 'sqft_living', ax=ax)
ax.set_xlabel('square foot living area')
ax.set_ylabel('price (in millions)')
ax.set_title('Fitted Values vs Square Foot Living');
```



In [56]:
```python
fig = plt.figure(figsize=(15,10))
sm.graphics.plot_partregress_grid(results_multi, exog_idx=list(X_multi.columns.valu
                                  grid=(5,5), fig=fig)
plt.show()
```



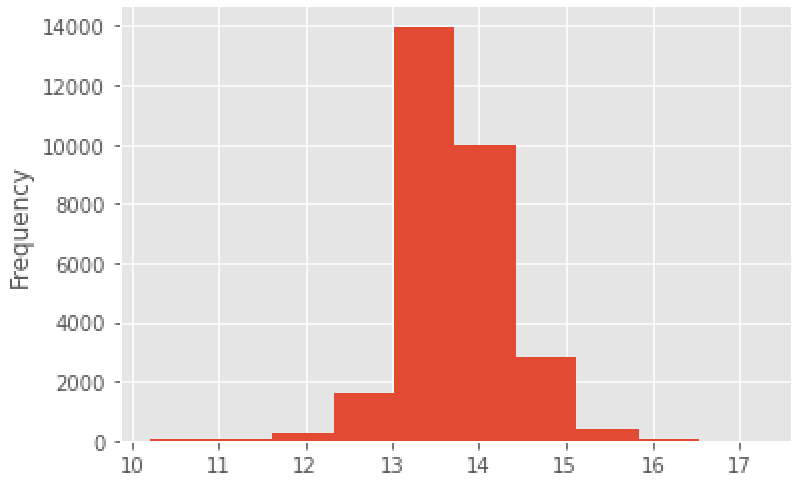
The partial regression plot shows the relationship between price and each coefficient, given the effect of the other independent variables in the model. We can see from the matrix that most of our coefficients have smaller slopes, and that the coefficients that are not statistically significant have the slopes closest to zero. Based on the slopes, `sqft_living` and the the two highest categories of grade seem to be the strongest predictors of price.

Log Transformation

From the earlier data exploration we know that thetarget variable, `price` , has a heavy right skew. Transforming this irregular distribution into more normal curve can potentially help improve the model and allow it to better meet the assumptions of linear regression. Let's try a logrithmic transformation in an attempt to create an enhanced model.

In [57]:

```
# Create Log Transformed Price + display new distribution
subset_df_1 = subset_df.copy()
subset_df_1['price_log'] = subset_df_1['price'].apply(lambda x: np.log(x))
y_log = subset_df_1['price_log']
y_log.plot.hist();
```



That curve looks much more normal, if a little skinny. Let's now create + fit a model with this transformed target.

```
In [58]:
X_1 = subset_df_1.drop(['price', 'price_log'], axis=1)

model_y_log = sm.OLS(y_log, sm.add_constant(X_1))
results_y_log = model_y_log.fit()
print(results_y_log.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.476
Model:                            OLS   Adj. R-squared:                  0.475
Method:                 Least Squares   F-statistic:                     1204.
Date:                Sun, 17 Sep 2023   Prob (F-statistic):               0.00
Time:                        13:53:02   Log-Likelihood:                -16389.
No. Observations:               29196   AIC:                         3.282e+04
Df Residuals:                   29173   BIC:                         3.301e+04
Df Model:                          22
Covariance Type:            nonrobust
=================================================================================
======
                          coef    std err          t      P>|t|      [0.025
0.975]
---------------------------------------------------------------------------------
------
const                  13.1185      0.008   1596.983      0.000      13.102
13.135
sqft_living             0.0002   3.91e-06     50.690      0.000       0.000
0.000
grade_ Lower           -0.4439      0.110     -4.037      0.000      -0.659
-0.228
waterfront_YES          0.1876      0.022      8.351      0.000       0.144
0.232
greenbelt_YES           0.0769      0.016      4.908      0.000       0.046
0.108
nuisance_YES            0.0474      0.007      7.141      0.000       0.034
0.060
view_AVERAGE            0.1252      0.010     12.171      0.000       0.105
0.145
view_EXCELLENT          0.3513      0.022     16.033      0.000       0.308
0.394
view_FAIR               0.2376      0.029      8.153      0.000       0.180
0.295
view_GOOD               0.1267      0.015      8.447      0.000       0.097
0.156
condition_Fair         -0.0265      0.029     -0.926      0.354      -0.082
0.030
condition_Good          0.0677      0.006     11.626      0.000       0.056
0.079
condition_Poor         -0.1750      0.055     -3.163      0.002      -0.283
-0.067
condition_Very Good     0.1502      0.008     18.250      0.000       0.134
0.166
grade_10 Very Good      0.6108      0.015     41.769      0.000       0.582
0.639
grade_11 Excellent      0.7728      0.024     31.556      0.000       0.725
0.821
grade_12 Luxury         0.8111      0.042     19.344      0.000       0.729
0.893
grade_13 Mansion        0.5853      0.090      6.481      0.000       0.408
0.762
grade_4 Low            -0.2421      0.060     -4.027      0.000      -0.360
-0.124
grade_5 Fair           -0.2226      0.022    -10.040      0.000      -0.266
-0.179
grade_6 Low Average    -0.1716      0.009    -18.792      0.000      -0.189
```

```
-0.154
grade_8 Good           0.1788      0.006     27.847      0.000       0.166
0.191
grade_9 Better         0.4229      0.010     44.487      0.000       0.404
0.442
==============================================================================
Omnibus:                     7103.622   Durbin-Watson:                   1.991
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            65608.924
Skew:                          -0.908   Prob(JB):                         0.00
Kurtosis:                      10.116   Cond. No.                     1.05e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.05e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [59]:
```
y_log_pred = results_y_log.predict(sm.add_constant(X_1))
mean_absolute_error(np.exp(y_log), np.exp(y_log_pred))
```

Out[59]:
```
337637.3726932627
```

From the r-squared we can see that this model is explaining about 48% of the variance in our transformed "ln(price)". While this statistic is not directly comparable to the r-squared in our first model, this is not a bad r-squared overall. Furthermore, many more of our coefficients are now statistically significant at an alpha of 0.05 (with the exception of condition_Fair, all of our coefficients are now significant). Our Mean Absolute Error also decreased by about $14k from our previous model which shows an improvement. We are seeing significant Omnibus and Jarque-Bera results indicating that we are still not meeting the assumptions of linear regression.

Drop Outliers

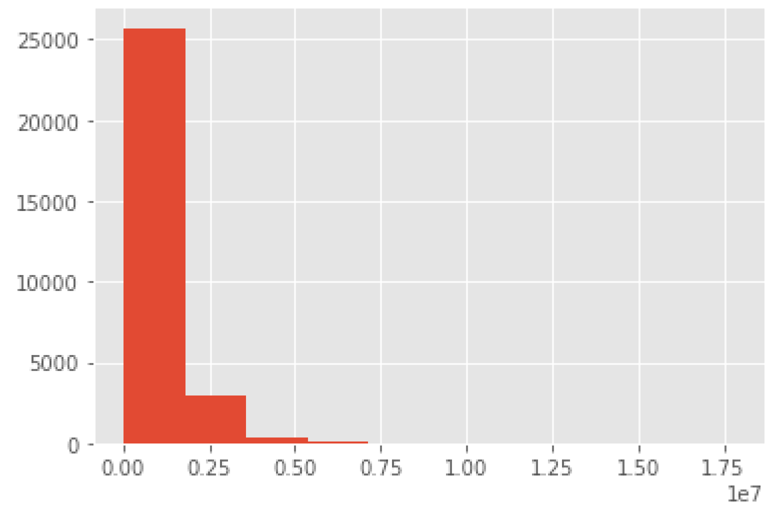
Another important note from our earlier data exploration was that there are a few outliers in our target variable `price` . Let's explore that a bit further and try a model without these outliers.

In [60]:
```
subset_df['price'].sort_values(ascending=False).head(10)
```

Out[60]:
```
23470    30750000.0
3760     23500000.0
25561    22750000.0
27175    20000000.0
18100    17800000.0
9516     15740000.0
10605    15000001.0
8054     14850000.0
22707    14500000.0
5811     13950000.0
Name: price, dtype: float64
```

```
In [61]: # create dataframe with price outliers dropped
         subset_df_2 = subset_df[subset_df['price'] < 20000000]
```

```
In [62]: subset_df_2['price'].hist();
```



While our distribution still has a significant right skew, we do not want to lose too much data. Let's try modeling with this adjustment.

In [63]:

```
y_2 = subset_df_2['price']
X_2 = subset_df_2.drop('price', axis=1)
model_2 = sm.OLS(y_2, sm.add_constant(X_2))
results_2 = model_2.fit()
print(results_2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.536
Model:                            OLS   Adj. R-squared:                  0.536
Method:                 Least Squares   F-statistic:                     1534.
Date:                Sun, 17 Sep 2023   Prob (F-statistic):               0.00
Time:                        13:53:03   Log-Likelihood:            -4.2883e+05
No. Observations:               29192   AIC:                         8.577e+05
Df Residuals:                   29169   BIC:                         8.579e+05
Df Model:                          22
Covariance Type:            nonrobust
=================================================================================
======
                          coef    std err          t      P>|t|      [0.025      
0.975]
---------------------------------------------------------------------------------
------
const                2.448e+05   1.12e+04     21.772      0.000    2.23e+05    2.
67e+05
sqft_living           290.4973      5.347     54.326      0.000     280.016     3
00.978
grade_ Lower        -6.911e+04    1.5e+05     -0.459      0.646   -3.64e+05    2.
26e+05
waterfront_YES       5.932e+05   3.08e+04     19.280      0.000    5.33e+05    6.
54e+05
greenbelt_YES       -2.694e+04   2.14e+04     -1.256      0.209    -6.9e+04    1.
51e+04
nuisance_YES         7.223e+04   9074.146      7.960      0.000    5.44e+04      
9e+04
view_AVERAGE         1.278e+05   1.41e+04      9.086      0.000       1e+05    1.
55e+05
view_EXCELLENT       7.446e+05      3e+04     24.826      0.000    6.86e+05    8.
03e+05
view_FAIR            3.404e+05   3.99e+04      8.538      0.000    2.62e+05    4.
18e+05
view_GOOD             1.45e+05   2.05e+04      7.063      0.000    1.05e+05    1.
85e+05
condition_Fair      -3.827e+04   3.91e+04     -0.979      0.328   -1.15e+05    3.
84e+04
condition_Good       6.039e+04   7965.593      7.581      0.000    4.48e+04      
7.6e+04
condition_Poor      -7.148e+04   7.57e+04     -0.945      0.345    -2.2e+05    7.
69e+04
condition_Very Good  1.434e+05   1.13e+04     12.742      0.000    1.21e+05    1.
65e+05
grade_10 Very Good   8.792e+05      2e+04     43.942      0.000     8.4e+05    9.
18e+05
grade_11 Excellent   1.661e+06   3.35e+04     49.499      0.000    1.59e+06    1.
73e+06
grade_12 Luxury      2.425e+06   5.79e+04     41.900      0.000    2.31e+06    2.
54e+06
grade_13 Mansion     4.404e+06   1.24e+05     35.648      0.000    4.16e+06    4.
65e+06
grade_4 Low          -3.22e+04   8.22e+04     -0.392      0.695   -1.93e+05    1.
29e+05
grade_5 Fair        -2.133e+04   3.03e+04     -0.703      0.482   -8.08e+04    3.
81e+04
grade_6 Low Average -3.373e+04   1.25e+04     -2.700      0.007   -5.82e+04   -92
```

```
44.847
grade_8 Good          1.18e+05   8781.912     13.434      0.000     1.01e+05    1.
35e+05
grade_9 Better       4.323e+05    1.3e+04     33.234      0.000     4.07e+05    4.
58e+05
==============================================================================
Omnibus:                    27416.399   Durbin-Watson:                   1.900
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          4128139.352
Skew:                           4.087   Prob(JB):                         0.00
Kurtosis:                      60.681   Cond. No.                     1.05e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.05e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [64]:
```
y_2_pred = results_2.predict(sm.add_constant(X_2))
mean_absolute_error(y_2, y_2_pred)
```

Out[64]:
```
347548.50127667247
```

This model is a small improvement from our first multiple variable linear regression -- the r-squared statistic increased, and the mean absolute error decreased (though not as much as with our log transformed model). We are still seeing 6 coefficients that are not statistically significant.

Dropped Outliers + Log Transformed Price

Let's try one last model where we apply both of our changes above: drop the outliers in price + transform price via a natural log function.

In [65]:

```
y_3 = np.log(y_2)
X_3 = X_2

model_3 = sm.OLS(y_3, sm.add_constant(X_3))
results_3 = model_3.fit()
print(results_3.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.475
Model:                            OLS   Adj. R-squared:                  0.474
Method:                 Least Squares   F-statistic:                     1198.
Date:                Sun, 17 Sep 2023   Prob (F-statistic):               0.00
Time:                        13:53:03   Log-Likelihood:                -16361.
No. Observations:               29192   AIC:                         3.277e+04
Df Residuals:                   29169   BIC:                         3.296e+04
Df Model:                          22
Covariance Type:            nonrobust
=================================================================================
======
                          coef    std err          t      P>|t|      [0.025
0.975]
---------------------------------------------------------------------------------
------
const                  13.1177      0.008   1597.577      0.000      13.102
13.134
sqft_living             0.0002   3.91e-06     50.811      0.000       0.000
0.000
grade_ Lower           -0.4417      0.110     -4.020      0.000      -0.657
-0.226
waterfront_YES          0.1796      0.022      7.992      0.000       0.136
0.224
greenbelt_YES           0.0776      0.016      4.955      0.000       0.047
0.108
nuisance_YES            0.0471      0.007      7.106      0.000       0.034
0.060
view_AVERAGE            0.1258      0.010     12.245      0.000       0.106
0.146
view_EXCELLENT          0.3464      0.022     15.811      0.000       0.303
0.389
view_FAIR               0.2380      0.029      8.175      0.000       0.181
0.295
view_GOOD               0.1279      0.015      8.535      0.000       0.099
0.157
condition_Fair         -0.0257      0.029     -0.899      0.369      -0.082
0.030
condition_Good          0.0680      0.006     11.681      0.000       0.057
0.079
condition_Poor         -0.1746      0.055     -3.159      0.002      -0.283
-0.066
condition_Very Good     0.1504      0.008     18.292      0.000       0.134
0.166
grade_10 Very Good      0.6103      0.015     41.763      0.000       0.582
0.639
grade_11 Excellent      0.7667      0.025     31.289      0.000       0.719
0.815
grade_12 Luxury         0.7832      0.042     18.524      0.000       0.700
0.866
grade_13 Mansion        0.5852      0.090      6.485      0.000       0.408
0.762
grade_4 Low            -0.2396      0.060     -3.989      0.000      -0.357
-0.122
grade_5 Fair           -0.2217      0.022    -10.006      0.000      -0.265
-0.178
grade_6 Low Average    -0.1712      0.009    -18.770      0.000      -0.189
```

```
-0.153
grade_8 Good            0.1786      0.006     27.840      0.000       0.166
0.191
grade_9 Better          0.4225      0.010     44.475      0.000       0.404
0.441
==============================================================================
Omnibus:                     7143.031   Durbin-Watson:                   1.990
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            65620.458
Skew:                          -0.916   Prob(JB):                         0.00
Kurtosis:                      10.113   Cond. No.                     1.05e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.05e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [66]:
```
y_3_pred = results_3.predict(sm.add_constant(X_3))
mean_absolute_error(np.exp(y_3), np.exp(y_3_pred))
```

Out[66]:
```
335177.00676580233
```

These results are very similar to the model original y_log model, with slightly decreased r-squared statistic and mean absolute error statistic.

Final Model - Which Iteration is best?

Choosing the best model is highly dependent on our specific scenario. In this case, while tempting to simply choose the model with the highest explained variance in price, it is actually more important to decrease our error and communicate to the business the most amount of usable information. With this in mind, I will be moving forward with the model that both dropped the outliers and applied and logarithmic transformation for the target `price` .

While this model did not amend the Omnibus or Jarque-Bera results in a significant manner as we were hoping, the results of this model had the smallest mean absolute error, and included the most amount of statistically significant coefficients.

Standard Scale

Let's take a look at this final model after applying a standard scaling the numerical variable. This will allow us more easily identify what the 'average' house is predicted to cost.

In [67]:
```
# Standard Scaling the numerical variable
X_stand = X_3.copy()
X_stand['sqft_living'] = (X_stand['sqft_living'] - X_stand['sqft_living'].mean()) \
                        / X_stand['sqft_living'].std()

# Create model with standard scaling
model_stand = sm.OLS(y_3, sm.add_constant(X_stand))
results_stand = model_stand.fit()

# View constant coefficient + confidence interval
print('const coeff:', round(np.exp(results_stand.params['const']),0))
print('const confidence interval: \n', np.exp(results_stand.conf_int().loc['const']
```

```
const coeff: 759490.0
const confidence interval:
 0    751819.691790
1    767238.397445
Name: const, dtype: float64
```

Prep for Analysis

Let's also pull and transform some data in preperation of analysis of this final model.

In [68]:
```
print('Mean sqft_living:', round(X_3['sqft_living'].mean(),2))
```

```
Mean sqft_living: 2130.2
```

In [69]:
```
dict_ = {'variables' : list(results_3.params.index), 'coefficients' : list(results_
df_coeff = pd.DataFrame(dict_)
df_coeff['coefficient_percentage'] = df_coeff['coefficients'].map(lambda x: round((
df_coeff.drop([0,10], inplace=True)
df_coeff
```

Out[69]:

	variables	coefficients	coefficient_percentage
1	sqft_living	0.000198	0.02
2	grade_ Lower	-0.441689	-35.71
3	waterfront_YES	0.179605	19.67
4	greenbelt_YES	0.077609	8.07
5	nuisance_YES	0.047099	4.82
6	view_AVERAGE	0.125832	13.41
7	view_EXCELLENT	0.346352	41.39
8	view_FAIR	0.238021	26.87
9	view_GOOD	0.127935	13.65
11	condition_Good	0.067958	7.03
12	condition_Poor	-0.174634	-16.02
13	condition_Very Good	0.150380	16.23
14	grade_10 Very Good	0.610291	84.10
15	grade_11 Excellent	0.766692	115.26
16	grade_12 Luxury	0.783186	118.84
17	grade_13 Mansion	0.585152	79.53
18	grade_4 Low	-0.239586	-21.30
19	grade_5 Fair	-0.221690	-19.88
20	grade_6 Low Average	-0.171236	-15.74
21	grade_8 Good	0.178569	19.55
22	grade_9 Better	0.422544	52.58

Analysis + Conclusions

Overall:

- The model is overall statistically significant and explains about 47% of the variance in our target price (specifically ln(price)).
- The final model includes the following variables: `sqft_living` , `waterfront` , `greenbelt` , `nuisance` , `view` , `condition` , and `grade` .
- All of the coefficients, aside from `condition_Fair` , are statistically significant.

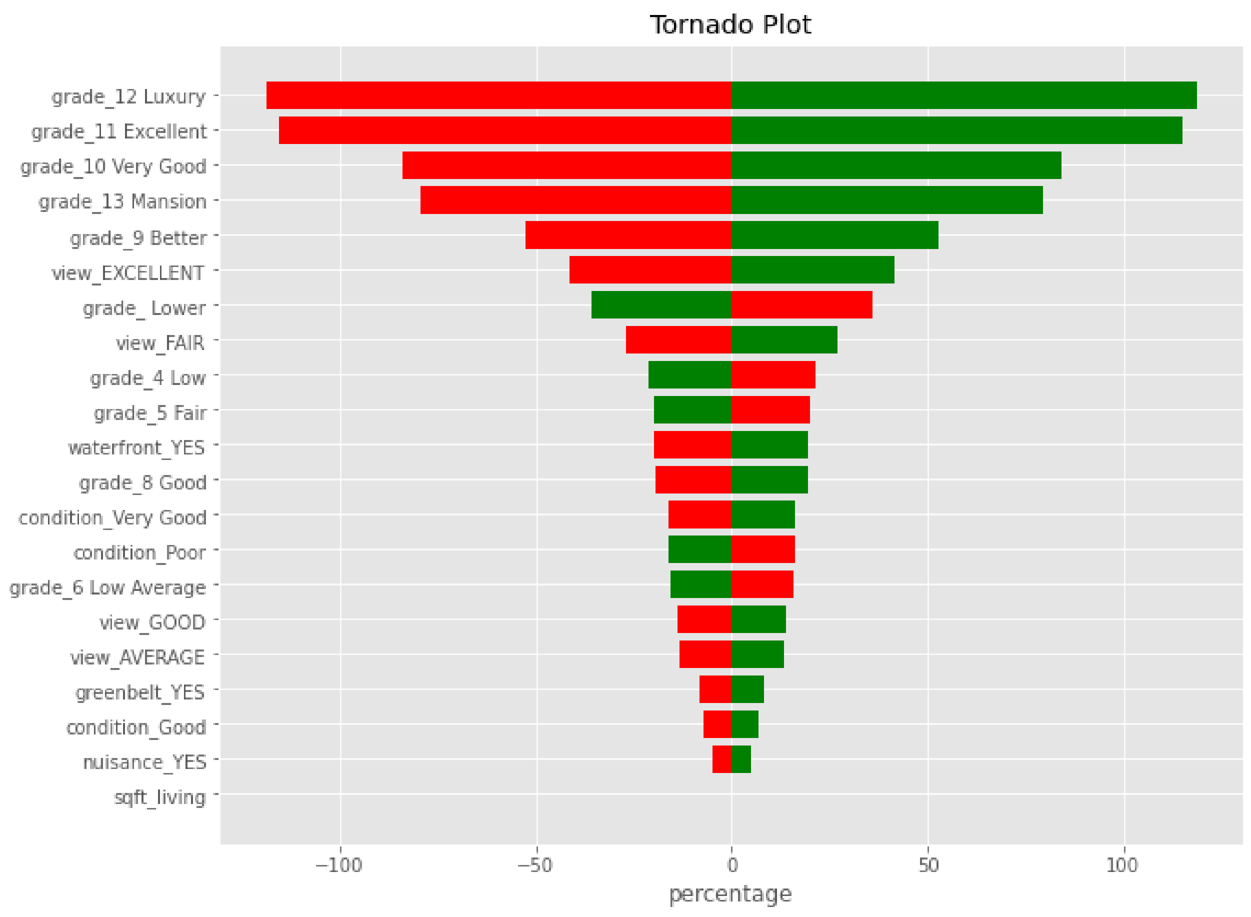
In specific:

- An average house (a house that has a living area of about 2130 square feet, is not on a waterfront, does not have any recorded nuisances, does not have any view, and has a grade and condition of average), will cost about $760k
- For every one square foot increase in living area, house price increases by about .02%
- If the house is on a waterfront, then it will be about 19.7% more expensive than if it was not on a waterfront
- If the house has a recorded nuisance next to it, the model predicts an increase in house price by about 4.8%
- Breaking down VIEW, compared to having NO view:
  - if house has a Fair view, house price increases by about 26.9%
  - if house has an Average view, house price increases by about 13.4%
  - if house has a Good view, house price increases by about 13.7%
  - if house has an Excellent view, house price increases by about 41.4%
- Breaking down CONDITION, compared to an Average condition:
  - if the condition is Poor, house price decreases by about 16.0%
  - if the condition is Good, house price increases by about 7.0%
  - if the condition is Very Good, house price increases by about 16.2%
- Breaking down GRADE, compared to an Average grade:
  - if grade is Lower, house price decreases by about 35.7%
  - if grade is Low, house price decreases by about 21.3%
  - if grade is Fair, house price decreases by about 19.9%
  - if grade is Low Average, house price decreases by about 15.7%
  - if grade is Good, house price increases by about 19.6%
  - if grade is Better, house price increases by about 52.6%
  - if grade is Very Good, house price increases by about 84.1%
  - if grade is Excellent, house price increases by about 115.3%
  - if grade is Luxury, house price increases by about 118.9%
  - if grade is Mansion, house price increases by about 79.5%

Tornado Plot

Let's graph a tornado plot of all the statistically significant coefficients. Specifically, it will be impactful to see the percent change in price for each variable. It will put a visual to the analysis above, highlighting the most impactful variables.

In [70]:
```python
def tornado_plot(categories, values):
    fig, ax = plt.subplots(figsize=(10,8))
    base = 0
    ax.barh(categories, values, left=base, color='green')
    ax.barh(categories, -values, left=base, color='red')
    ax.set_xlabel('percentage')
    ax.set_title('Tornado Plot');

df_coeff['abs_percentage'] = abs(df_coeff['coefficient_percentage'])
df_coeff = df_coeff.sort_values('abs_percentage', axis=0)
tornado_plot(df_coeff['variables'], df_coeff['coefficient_percentage'])
```



Suggestions

1. Minimum Budget

Plan to budget at least $760,000 per house plan to purchase. This is the average price of a home in King County, defined by a house with about 2130 square feet of living area, is not located on a waterfront, is not next to a greenbelt, does not have any recorded nuisances, does not have any view, and has average grade and condition ratings).

2. Condition

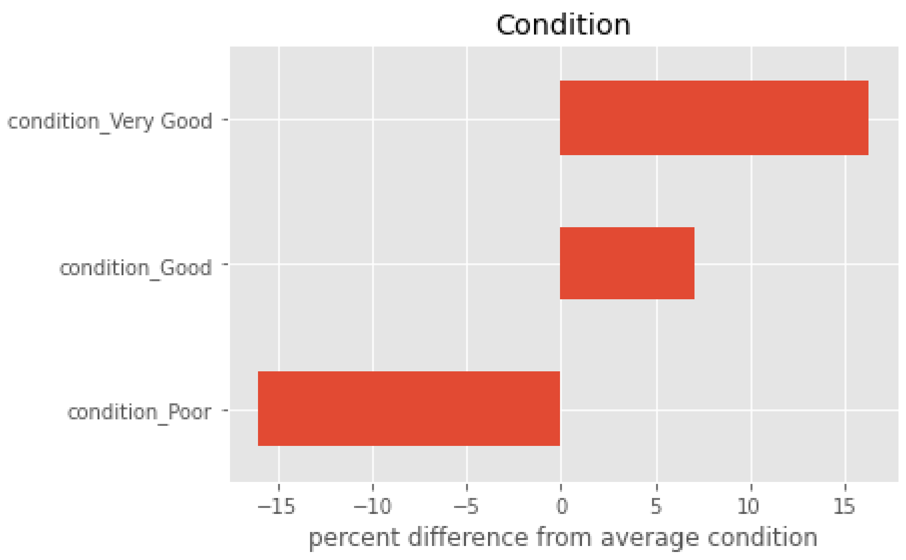
King County defines a home with an average condition to have:

> "Some evidence of deferred maintenance and normal obsolescence with age in that a few minor repairs are needed, along with some refinishing. All major components still functional and contributing toward an extended life expectancy. Effective age and utility is standard for like properties of its class and usage."

In comparison, a home with a poor condition rating is a 'worn out' home that is near the end of its lifespan. Although these homes would be about 16% cheaper, the amount of work and money needed to fix the home would most likely negate any savings from purchasing this home rather than one of an average condition. Thus, it is not recommended to purchase any homes with a poor condition rating.

Furthermore, it would be wise to set aside extra money to purchase a house with either a good or very good condition rating. All else being the same as our 'average home', this would mean setting aside an

In [71]:
```
df_coeff.set_index('variables', inplace=True)

conditions = ['condition_Poor', 'condition_Good', 'condition_Very Good']

fix, ax = plt.subplots()

df_coeff['coefficient_percentage'].loc[conditions].plot.barh(ax=ax)

ax.set_ylabel('')
ax.set_xlabel('percent difference from average condition')
ax.set_title('Condition');
```



3. Grade

King County defines a home with average grade as:

> "Average grade of construction and design. Commonly seen in plats and older sub-divisions."

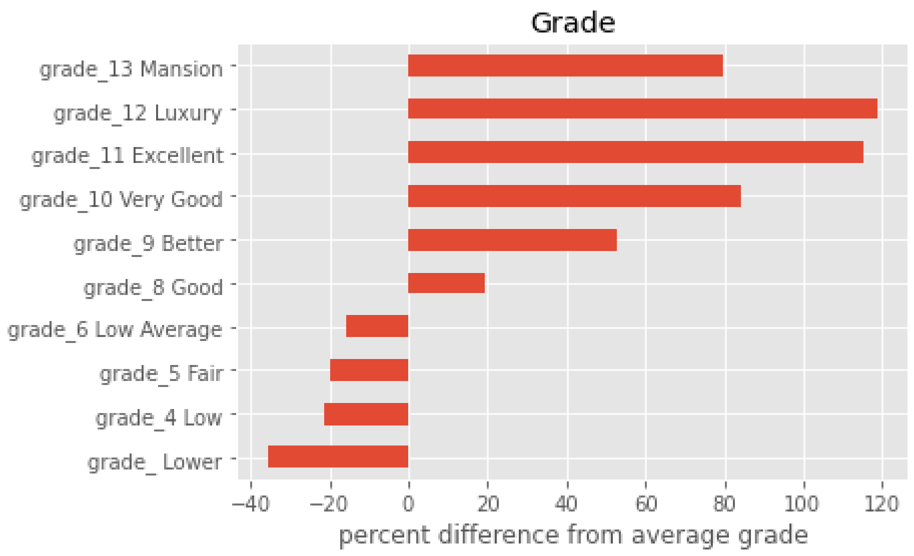
It is important to note that King County has set the rating of 6 low average as the lowest grade that meets building code standards. Grades 8 through 10 become slightly better in architectural design and construction sequentially. Grades 11 through 13 generally have custom designs and increased levels of luxurious quality.

Based on these definitions, it is recommended to only purchase homes that have a grade of 6 or higher. Similar to condition, the amount of money that would be required to bring the house up to building code standards would most likely exceed the savings from purchasing a lower grade home.

With all else about the home remaining the same as our previously defined 'average home':

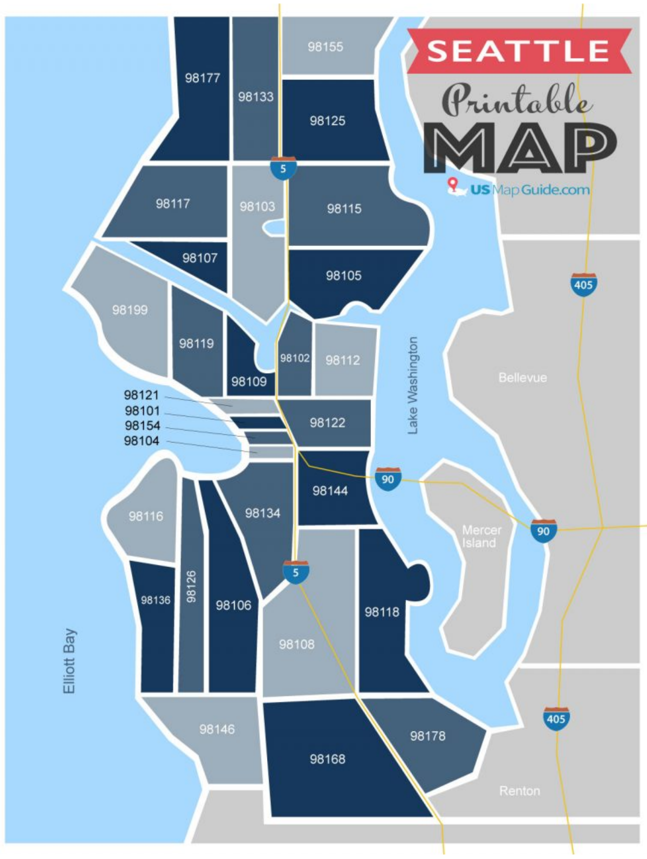
- If a home with a grade of low average is purchased, the center could see savings of about $120k.
- Furthermore, it would be wise to set aside an additional $148k to $639k for homes with grades between good to very good that are considered for purchase ($399k for a condition of better).

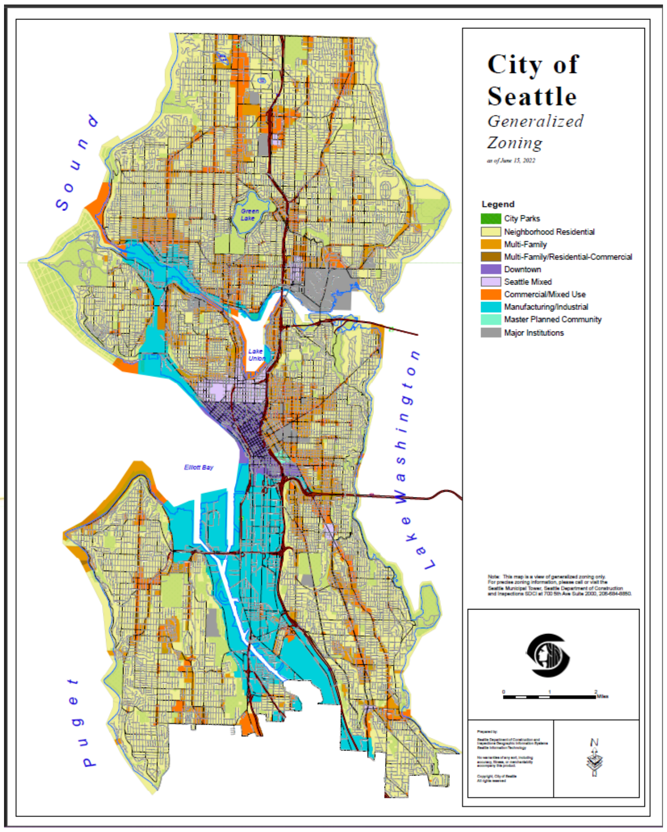
In [72]:
```
grades = ['grade_ Lower', 'grade_4 Low', 'grade_5 Fair', 'grade_6 Low Average',
          'grade_8 Good', 'grade_9 Better', 'grade_10 Very Good',
          'grade_11 Excellent', 'grade_12 Luxury', 'grade_13 Mansion']


fig, ax = plt.subplots()

df_coeff['coefficient_percentage'].loc[grades].plot.barh(ax=ax)


ax.set_xlabel('percent difference from average grade')
ax.set_ylabel('')
ax.set_title('Grade');
```



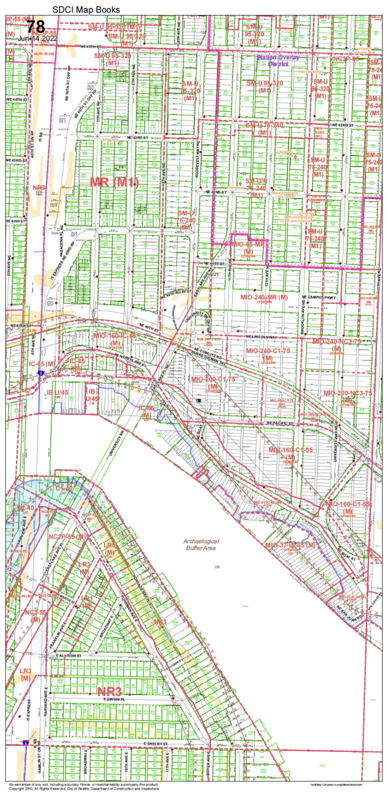
Next Steps

It would be beneficial to add information concerning the zone these houses are in. However, due to the complexity of how zones are distinguished (cities and towns decide these areas rather than the county and they are not mapped along any easily distinguishable data such as zip code), it would be more helpful to incorporate this data once a single or a few specific towns/cities have been targeted. This information would be beneficial as the residential areas tend to give pushback on companies and people who turn their homes into half-way houses or sober living homes.





As we can see in the above two images, multiple different zones can be in a single zip code. This can be seen on a more granular level from the below image - a section of Seattle that includes the most northern section of zip code area 98102, and the most southern section of sip code area 98105.


SDCI Map Books
78
June 14 2022
MR (M1)
NR3
Archaeological Buffer Area