The Garage Problem: Real Estate Recommendations for Kings County California

By Jordan Loewen-Colón October 14th 2022

The Business Problem

King's County Realtors are interested in whether or not they should renovate homes before trying to sell. Specifically, they'd like to know how much adding a garage might affect price, and if so, what size of garage.

Recommendations:

Based on our models and analysis, we recommend that if rennovations are going to occur, its best to target square footage of living space, but if renovations are going to includ the garage, it is probably worth focusing on homes that have no garage and adding a 1-car sized garage, rather than increasing the size of an existing garage.

Step 1: Data Understanding

We will analyze the 2022 data from King's Country to try and offer accurate recommendations.

We begin by importing the proper tools and then the data itself.

In [1]:
```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.formula.api as smf
import statsmodels.api as sm
from scipy import stats
from sklearn.linear_model import LinearRegression

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings("ignore")

#Import data
kcdf=pd.read_csv(r"C:\Users\legac\data\kc_house_data.csv")
```

In [2]:
```
#view data
print(kcdf.shape)
kcdf.info()
```

```
(30155, 25)
<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
```

The dataset has 30155 entries and 25 columns with a mix of string values, floats, and integers. Bathrooms as float makes sense, but "floors" as float seems odd. It is not clear what elements are contained in some of the object categories like "grade" or "nuisance." It also looks like we have some missing entries for "sewer_system" and "heat_source." Let's take a closer look at the data entries.

In [3]:
```
kcdf.head()
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	greenbelt	...	sewer_system	sqft_above	sqft_basement	sc
0	7399300360	5/24/2022	675000.0	4	1.0	1180	7140	1.0	NO	NO	...	PUBLIC	1180	0	
1	8910500230	12/13/2021	920000.0	5	2.5	2770	6703	1.0	NO	NO	...	PUBLIC	1570	1570	
2	1180000275	9/29/2021	311000.0	6	2.0	2880	6156	1.0	NO	NO	...	PUBLIC	1580	1580	
3	1604601802	12/14/2021	775000.0	3	3.0	2160	1400	2.0	NO	NO	...	PUBLIC	1090	1070	
4	8562780790	8/24/2021	592500.0	2	2.0	1120	758	2.0	NO	NO	...	PUBLIC	1120	550	

5 rows × 25 columns

It looks like there are some houses with no garages or which have never been renovated. It's also not clear how some of these columns are going to help us answer our problem, so we will probably end up focusing primarily on "yr_renovated," and the "sqft" categories. Now that we have some idea of what the values of the columns look like, let's sort by values (specifically "price") to see if we notice any outliers.

In [4]:
```
kcdf.describe().sort_values("price")
```

Out[4]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	sqft_above	sqft_basement	sqft_garage	sqft_
min	1.000055e+06	2.736000e+04	0.000000	0.000000	3.000000	4.020000e+02	1.000000	2.000000	0.000000	0.000000	0.0
count	3.015500e+04	3.015500e+04	30155.000000	30155.000000	30155.000000	3.015500e+04	30155.000000	30155.000000	30155.000000	30155.000000	30155.0
25%	2.064175e+09	6.480000e+05	3.000000	2.000000	1420.000000	4.850000e+03	1.000000	1180.000000	0.000000	0.000000	40.0
50%	3.874011e+09	8.600000e+05	3.000000	2.500000	1920.000000	7.480000e+03	1.500000	1560.000000	0.000000	400.000000	150.0
std	2.882587e+09	8.963857e+05	0.981612	0.889556	974.044318	6.038260e+04	0.567717	878.306131	579.631302	285.770536	245.3
mean	4.538104e+09	1.108536e+06	3.413530	2.334737	2112.424739	1.672360e+04	1.543492	1809.826098	476.039396	330.211142	217.4
75%	7.287100e+09	1.300000e+06	4.000000	3.000000	2619.500000	1.057900e+04	2.000000	2270.000000	940.000000	510.000000	320.0
max	9.904000e+09	3.075000e+07	13.000000	10.500000	15360.000000	3.253932e+06	4.000000	12660.000000	8020.000000	3580.000000	4370.0

Some odd things to notice here is that there seem to be houses without bedrooms or bathrooms. It also looks like there might be some outliers on the larger end as well, with houses containing 13 bedrooms and 10.5 bathrooms. Also looks like there might be a house with only 3sqft of living space, so we are going to have to fix that. We're also going to need to seperate out the houses that already have garages from those that do not. But before that, let's check for duplicates.

In [5]:
```
# Check for duplicates
kcdf.duplicated(subset = ["id", "date"]).sum() == 0
```

Out[5]:
```
False
```

So this is going to be a problem. We will need to remove them.

In [6]:
```
#Checking Outliers
print(kcdf['bedrooms'].value_counts())
print(kcdf['bathrooms'].value_counts())
```

```
3     12754
4      9597
2      3936
5      2798
6       498
1       391
7        80
0        44
8        38
9        14
10        3
13        1
11        1
Name: bedrooms, dtype: int64
2.5     8475
2.0     7349
1.0     4576
3.0     4117
3.5     2266
1.5     1808
4.0      645
4.5      533
5.0      145
5.5      104
6.0       45
0.0       31
6.5       25
7.5       12
7.0       12
0.5        5
9.5        2
8.0        2
8.5        1
10.0       1
10.5       1
Name: bathrooms, dtype: int64
```

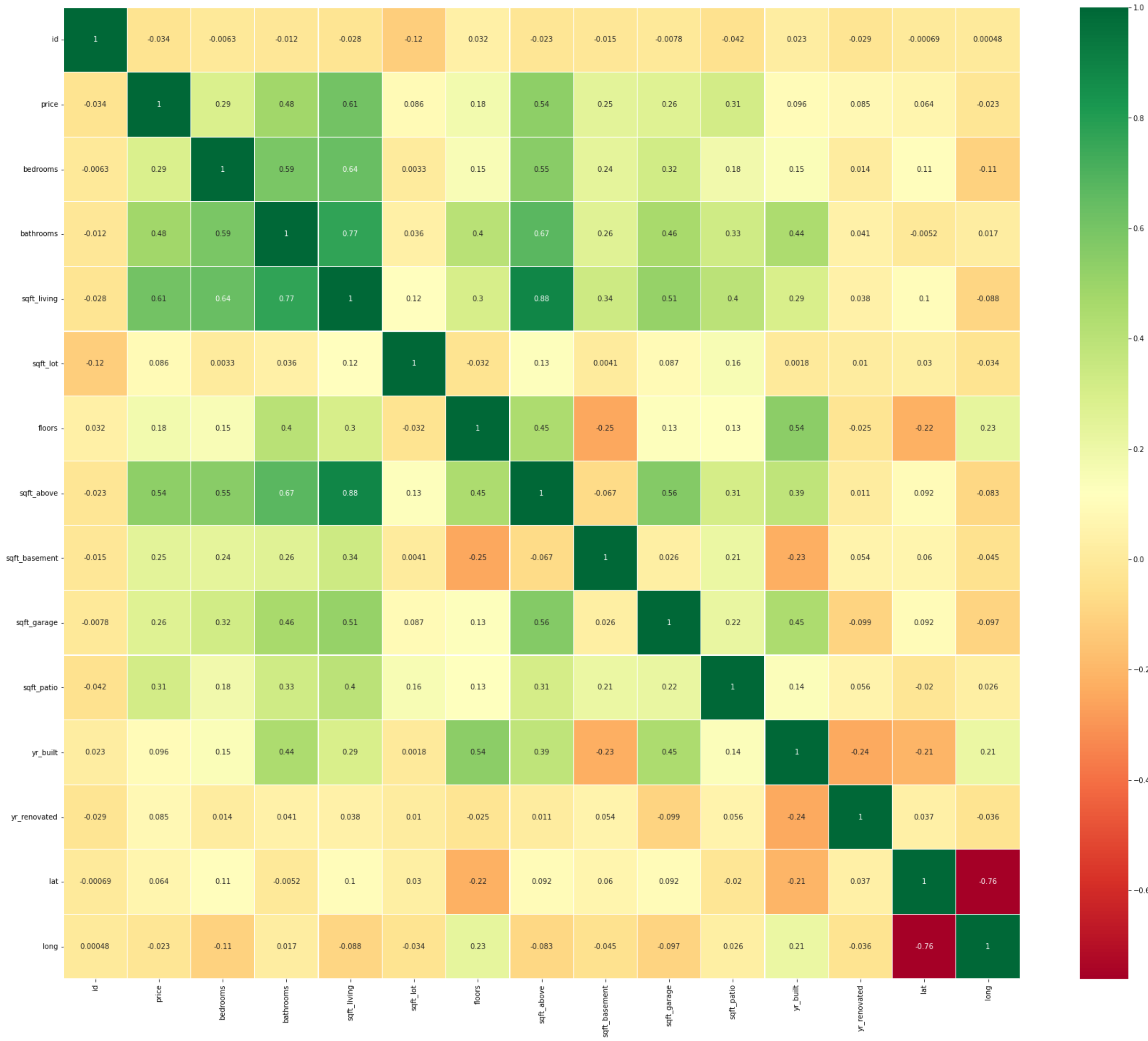
It's also still not clear how useful some of the other categories will be for getting an accurate answer to our problem. Spefically with "waterfront", "greenbelt", "nuisance", "condition", and "grade."

In [7]:
```
#Checking Values of Categories
print(kcdf['waterfront'].value_counts())
print(kcdf['greenbelt'].value_counts())
print(kcdf['nuisance'].value_counts())
print(kcdf['condition'].value_counts())
print(kcdf['grade'].value_counts())
```

```
NO     29636
YES      519
Name: waterfront, dtype: int64
NO     29382
YES      773
Name: greenbelt, dtype: int64
NO     24893
YES     5262
Name: nuisance, dtype: int64
Average      18547
Good          8054
Very Good     3259
Fair           230
Poor            65
Name: condition, dtype: int64
7 Average        11697
8 Good            9410
9 Better          3806
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              2
Name: grade, dtype: int64
```

With such small values in the "waterfront", "greenbelt", and "nuisance" categories, those are probably ripe for dropping. It looks like if we dropped some of the outliers for "grade" it might end up being useful. And "condition" could be worth keeping to see how it affects our renovation coefficients. We can double check our intuition by visualizing some of the data already.

In [8]:
```
#Checking For multicollinearity
#Pearson Corellation
sns.heatmap(kcdf.corr(),annot=True,cmap='RdYlGn',linewidths=0.2)
fig=plt.gcf()
fig.set_size_inches(30,25)
plt.show()
```



According to the our Pearson corellation calculations, our intuitions look right. We can drop quite a few of these categories that seem to have minimal relevance to our "price."

Step 2: Data Preperation

- In preparing the data, we will primarily focus on elements we think will affect our numbers involving renovations broadly, and garage additions more specifically. Since we are dealing with both continuous numbers and integers, we may end up needing some log transformations. We will test our model by looking at the correlation between price per sqft of house. According to according to the website [www.fixr.com (http://www.fixr.com)](http://www.fixr.com) the cost of building a home in California is roughly "400 and 600 per square foot." So that will be a good target for checking the accuracy of our data prep.

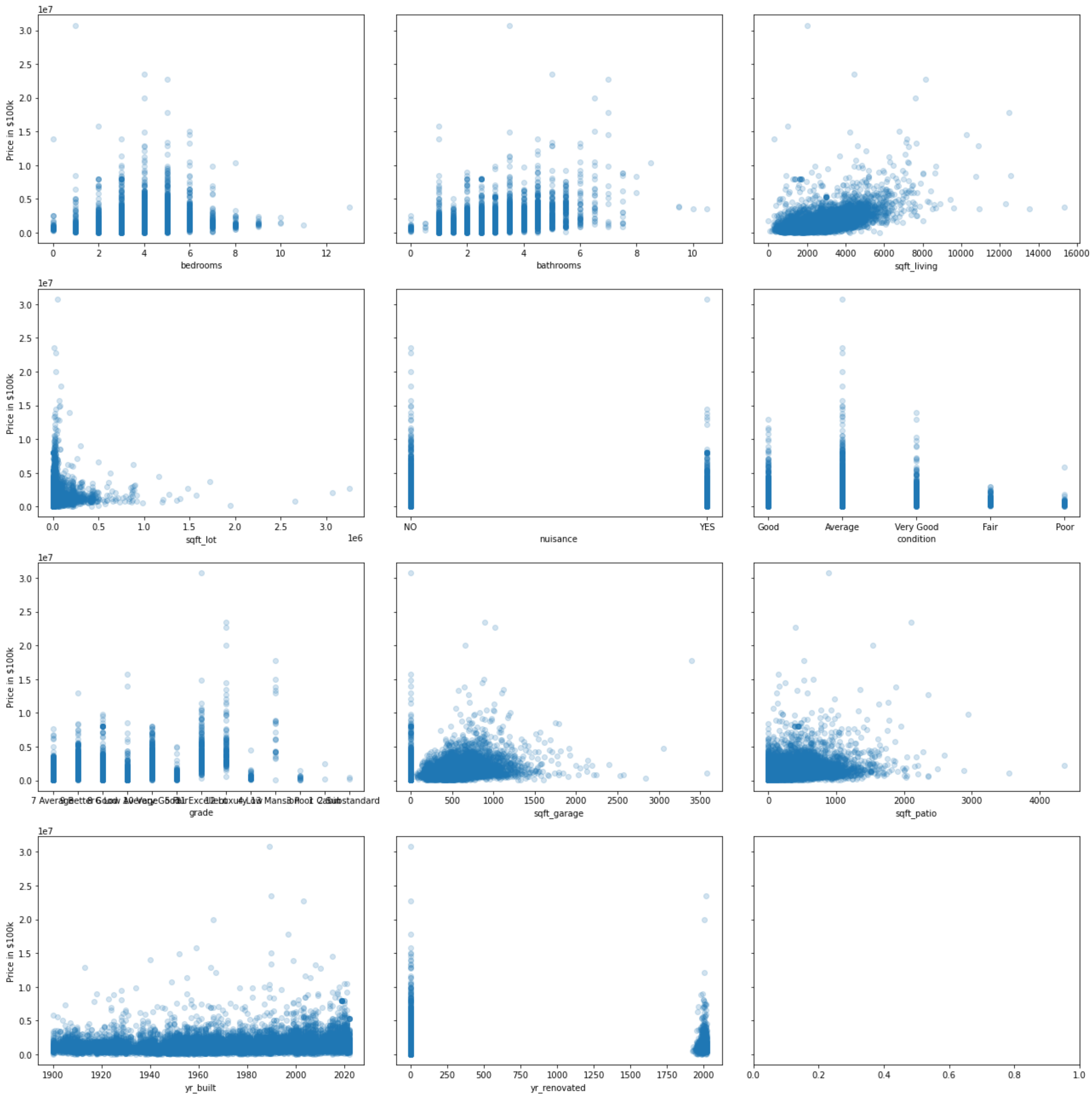
In [9]:
```
#Create a copy of DataFrame for preperation
df = kcdf.copy(deep=True)
```

In [10]:
```
#Dropping Duplicates
df.drop_duplicates(subset=['id', 'date'])

#Dropping Columns
df.drop(['id', 'date', 'view', 'lat', 'long', 'floors', 'address', 'sqft_above', 'sqft_basement', 'waterfront', 'greenbelt', 'sew

#No need to drop "NaNs" because we won't be using the sewer or heating columns.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30155 entries, 0 to 30154
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   price         30155 non-null  float64
 1   bedrooms      30155 non-null  int64  
 2   bathrooms     30155 non-null  float64
 3   sqft_living   30155 non-null  int64  
 4   sqft_lot      30155 non-null  int64  
 5   nuisance      30155 non-null  object 
 6   condition     30155 non-null  object 
 7   grade         30155 non-null  object 
 8   sqft_garage   30155 non-null  int64  
 9   sqft_patio    30155 non-null  int64  
 10  yr_built      30155 non-null  int64  
 11  yr_renovated  30155 non-null  int64  
dtypes: float64(2), int64(7), object(3)
memory usage: 2.8+ MB
```

Visualizing new dataframe for correlations

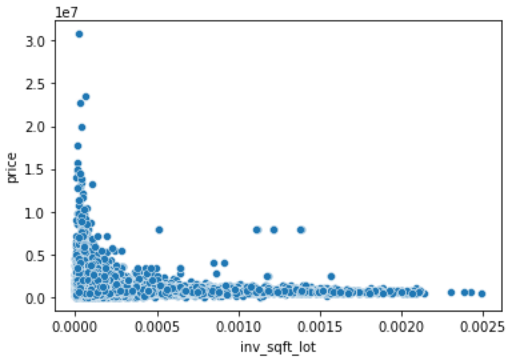
In [11]:
```
#Setting X and Y
y = df["price"]
x = df.drop("price", axis=1)

fig, axes = plt.subplots(nrows=4, ncols=3, figsize=(18,18), sharey=True)

for i, column in enumerate(x.columns):
    # Locate applicable axes
    row = i // 3
    col = i % 3
    ax = axes[row][col]

    # Plot feature vs. y and label axes
    ax.scatter(x[column], y, alpha=0.2)
    ax.set_xlabel(column)
    if col == 0:
        ax.set_ylabel("Price in $100k")

fig.tight_layout()
```



Looks like we will need to fix the outliers in bedrooms, bathrooms, and our sqft columns. The sqft_lot also seems really odd. We would expect price to go up the larger the lot size, but this plot seems to say the opposite. It also looks like we can adjust the "condition" and "grade" categories into ordinal numnbers so that our models will learn the proper relationship to the data. We will also have to adjust "yr_renovated" to make it more useful, and make a category frame for whether a home house a garage or not.

In [12]:
```
print('Shape before filtering', df.shape)

#Let's try converting sqft_lot into 1/x to see if that at least helps give us a better visual
df['inv_sqft_lot'] = df['sqft_lot'].apply(lambda x: 1/x)

sns.scatterplot(data=df, x='inv_sqft_lot', y='price');
```

```
Shape before filtering (30155, 12)
```



That definitely looks more linear! But it is no gaurantee that the fix will affect the model.

Now to fix "grade" and "condition." Turning them from strings to integers allows are model to see the relationship between the numbers more clearly and hopefully will give us more accuracy.

In [13]:
```
#Convert str to int for 'condition'
df['condition'] = df['condition'].replace(['Poor', 'Fair', 'Average', 'Good', 'Very Good'], [0, 1, 2, 3, 4])

#Convert str to int for 'grade'
df['grade'] = df['grade'].replace(['1 Cabin', '2 Substandard', '3 Poor', '4 Low', '5 Fair',
                                  '6 Low Average', '7 Average', '8 Good', '9 Better',
                                 '10 Very Good', '11 Excellent', '12 Luxury', '13 Mansion'],
                                 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
```

Outliers

In [14]:
```
#Deal with easy Outliers

print('Shape before filtering', df.shape)

#Remove "mansion" level houses and foreclosures (based on forclosure listings)
df.drop(df[(df['price']>=5000000) | (df['price']<=300000)].index, inplace = True)

#A House must have a bedroom and a bathroom. Mansions are such an outlier that our model won't make much use of them.
df.drop(df[(df['bedrooms']==0) | (df['bedrooms']>=10)].index, inplace = True)
df.drop(df[(df['bathrooms']==0) | (df['bedrooms']>=8)].index, inplace = True)

#Drop anything well below "small home" and "mansion" sized in sqft_living
df.drop(df[(df['sqft_living']<=500) | (df['sqft_living']>=7000)].index, inplace = True)

#Use Z-score to drop anything above three standard deviations for sqft_lot
df['sqft_lot'] = df['sqft_lot'].loc[(np.abs(stats.zscore(df.sqft_lot)) < 3)]
df = df.dropna(subset = ['sqft_lot'])

#Drop massive sized garages
df.drop(df[(df['sqft_garage']>=800)].index, inplace = True)

print('Shape after filtering', df.shape)
```

```
Shape before filtering (30155, 13)
Shape after filtering (27642, 13)
```

So after filtering, we lost about 2513 rows. We probably still have room to finetune the data by doing some logarithimic transformations, normalization, and removing outliers within three standard deviations. But we will save that for after we've done some baseline modeling below.

Next! Since we are specifically looking to answer the question about "rennovations" we will prepare our data by creating a column that determines whether or not a home has been renovated at all.

In [15]:
```
df['renovated']=''
def determine_reno(df):
    t=''
    x=df['yr_renovated']
    if(x==0):
        t=0
    elif(x>0):
        t=1
    return t

df['renovated']=df.apply(determine_reno,axis=1)
df.renovated.value_counts()
```

Out[15]:
```
0    26410
1     1232
Name: renovated, dtype: int64
```

Since there aren't actually that many renovated houses in our data set, we won't bother trying to see how "recency" might affect the price.

Next, we will do a similar adjustment with our garage data. First checking to see which houses actually have a garage, and then checking to see how big. According to [ShedsUnlimited (https://shedsunlimited.net/blog/how-large-is-a-one-car-garage/#:~:text=The%20minimum%20size%20of%20a,garage%20to%20fit%20inside%20comfortably.)](https://shedsunlimited.net/blog/how-large-is-a-one-car-garage/#:~:text=The%20minimum%20size%20of%20a,garage%20to%20fit%20inside%20comfortably.) the minimum size for a garage is 180ft, but really should be about 240ft in order to actually fit a car.

In [16]:
```
#Create a function to seperate homes with a grage from those without
df['garage']=''
def determine_garage(df):
    t=''
    x=df['sqft_garage']
    if(x<=239):
        t="No"
    elif(x>=240):
        t="Yes"
    return t

df['garage']=df.apply(determine_garage,axis=1)
df.garage.value_counts()
```

Out[16]:
```
Yes    16814
No     10828
Name: garage, dtype: int64
```

In [17]:
```
#Create a function to categorize by size of garage
df['garage_size']=''
def garage_sizer(df):
    t=''
    x=df['sqft_garage']
    if(x<=239):
        t=0
    elif(x>=240 and x<=359):
        t=1
    elif(x>=360 and x<=704):
        t=2
    elif(x>704):
        t=3
    return t

df['garage_size']=df.apply(garage_sizer,axis=1)
df.garage_size.value_counts()
```

Out[17]:
```
2    13172
0    10828
1     2610
3     1032
Name: garage_size, dtype: int64
```

With our garage size categories, it looks like we have a nice set of numbers to work with on our models. We may still need to adjust for some outliers, but overall, it makes sense that there seem to be more "medium" or two car garages than the others.

In [18]:
```
garage = df[df['garage'] == 1]
no_garage = df[df['garage'] == 0]

alpha = 0.05
garage_p_val = stats.ttest_ind(garage.price, no_garage.price, equal_var=False)[1]
print("Garage vs No Garage T-test P Value: ", garage_p_val)
if garage_p_val < 0.05:
    print("Having a garage vs not having a garage is statistically relevant to average property value")
else:
    print("accept null hypothesis")
```

```
Garage vs No Garage T-test P Value:  nan
accept null hypothesis
```

In [19]:
```
df.describe().sort_values("price")
```

Out[19]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	condition	grade	sqft_garage	sqft_patio	yr_built	yr_reno
count	2.764200e+04	27642.000000	27642.000000	27642.000000	27642.000000	27642.000000	27642.000000	27642.000000	27642.000000	27642.000000	27642.0
min	3.020000e+05	1.000000	0.500000	510.000000	402.000000	0.000000	1.000000	0.000000	0.000000	1900.000000	0.0
std	6.082013e+05	0.930925	0.824200	840.587017	15503.604051	0.705118	1.027392	246.656641	221.281240	32.355725	412.4
25%	6.500000e+05	3.000000	2.000000	1400.000000	4685.250000	2.000000	6.000000	0.000000	40.000000	1952.000000	0.0
50%	8.500000e+05	3.000000	2.500000	1880.000000	7236.000000	2.000000	6.000000	380.000000	140.000000	1976.000000	0.0
mean	1.041877e+06	3.391035	2.292725	2031.490449	10399.715035	2.483395	6.581253	303.278489	203.885826	1974.609363	89.0
75%	1.251750e+06	4.000000	2.500000	2520.000000	9900.000000	3.000000	7.000000	490.000000	300.000000	2003.000000	0.0
max	4.995000e+06	7.000000	9.500000	6860.000000	192212.000000	4.000000	12.000000	790.000000	2880.000000	2022.000000	2022.0

Data Modeling

Now with our data prepared and in hand, it's time to create some models. First, we will check to see the accuracy of our data preperation by checking price per sqft.

In [20]:
```
#Set "price" as y
y = df['price']
X = df['sqft_living']

model = sm.OLS(y, sm.add_constant(X))
results = model.fit()

print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.397
Model:                            OLS   Adj. R-squared:                  0.397
Method:                 Least Squares   F-statistic:                 1.818e+04
Date:                Thu, 12 Jan 2023   Prob (F-statistic):               0.00
Time:                        18:59:11   Log-Likelihood:            -4.0038e+05
No. Observations:               27642   AIC:                         8.008e+05
Df Residuals:                   27640   BIC:                         8.008e+05
Df Model:                           1
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const          1.16e+05   7431.206     15.607      0.000    1.01e+05    1.31e+05
sqft_living    455.7727      3.380    134.841      0.000     449.148     462.398
==============================================================================
Omnibus:                     9877.343   Durbin-Watson:                   1.971
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            59218.875
Skew:                           1.598   Prob(JB):                         0.00
Kurtosis:                       9.419   Cond. No.                     5.75e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.75e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

So it looks like the coefficient is within the right range, but the R-squared is very low. We will need to adjust in order to feel more confident in our model's predictions. Let's see if our log transformation gives us a better value:

In [21]:
```
# Log-transforming chosen variables
df["log_sqft_living"] = np.log(df[["sqft_living"]])
```

In [22]:
```
#Set X
X = df["log_sqft_living"]

model = sm.OLS(y, sm.add_constant(X))
results = model.fit()

print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.328
Model:                            OLS   Adj. R-squared:                  0.328
Method:                 Least Squares   F-statistic:                 1.352e+04
Date:                Thu, 12 Jan 2023   Prob (F-statistic):               0.00
Time:                        18:59:11   Log-Likelihood:            -4.0186e+05
No. Observations:               27642   AIC:                         8.037e+05
Df Residuals:                   27640   BIC:                         8.037e+05
Df Model:                           1
Covariance Type:            nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
const           -5.345e+06    5.5e+04    -97.158      0.000   -5.45e+06   -5.24e+06
log_sqft_living  8.478e+05   7291.580    116.269      0.000    8.33e+05    8.62e+05
==============================================================================
Omnibus:                    10779.110   Durbin-Watson:                   1.975
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            63271.714
Skew:                           1.778   Prob(JB):                         0.00
Kurtosis:                       9.503   Cond. No.                         141.
==============================================================================

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

Woah! Looks worse. What if we adjust using Z-score?

In [23]:
```
#First seperate continuous #s
df_cont = df[['price', 'sqft_living', 'sqft_garage', 'bedrooms', 'bathrooms', 'grade', 'condition']].copy()

#Remove outliers based on Z-score. Tinkered with the # to see what gave the best model.
df_std = df[(np.abs(stats.zscore(df_cont)) < 3).all(axis=1)]

#Check to see how many outliers were removed
print(len(df)-len(df_std))
```

```
962
```

In [24]:
```
#Set X and y
y = df_std["price"]
X = df_std["sqft_living"]

model = sm.OLS(y, sm.add_constant(X))
results = model.fit()

print(results.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                  price   R-squared:                       0.341
Model:                            OLS   Adj. R-squared:                  0.341
Method:                 Least Squares   F-statistic:                 1.378e+04
Date:                Thu, 12 Jan 2023   Prob (F-statistic):               0.00
Time:                        18:59:11   Log-Likelihood:            -3.8129e+05
No. Observations:               26680   AIC:                         7.626e+05
Df Residuals:                   26678   BIC:                         7.626e+05
Df Model:                           1                                         
Covariance Type:            nonrobust                                         
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const        2.504e+05   6654.472     37.630      0.000    2.37e+05    2.63e+05
sqft_living   370.0362      3.152    117.399      0.000     363.858     376.214
==============================================================================
Omnibus:                     3950.587   Durbin-Watson:                   2.010
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             7386.379
Skew:                           0.942   Prob(JB):                         0.00
Kurtosis:                       4.760   Cond. No.                     5.89e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.89e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

This time its better than the log transformation, but it is till worse than our original test. So that means we probably did enough cleaning during our data prep stage. Before we add the renovation data and garage data, let's see which variables give us the most accurate model.

In [25]:
```
#Set X and y
y = df["price"]
X = df[['sqft_living', 'sqft_lot', 'bedrooms', 'bathrooms', 'yr_built', "grade", 'renovated', 'condition', 'garage']]

#Dummies for Garage, Grade, and Condition
X = pd.get_dummies(X, columns=['garage', 'renovated', "grade", "condition"], drop_first=True)

model = sm.OLS(y, sm.add_constant(X))
results = model.fit()

print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.533
Model:                            OLS   Adj. R-squared:                  0.532
Method:                 Least Squares   F-statistic:                     1431.
Date:                Thu, 12 Jan 2023   Prob (F-statistic):               0.00
Time:                        18:59:11   Log-Likelihood:            -3.9685e+05
No. Observations:               27642   AIC:                         7.937e+05
Df Residuals:                   27619   BIC:                         7.939e+05
Df Model:                          22
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const         8.845e+06   4.74e+05     18.681      0.000    7.92e+06    9.77e+06
sqft_living    258.1625      5.966     43.273      0.000     246.469     269.856
sqft_lot        -0.0760      0.167     -0.454      0.650      -0.404       0.252
bedrooms     -3.947e+04   3806.885    -10.368      0.000   -4.69e+04    -3.2e+04
bathrooms     8.269e+04   5178.060     15.969      0.000    7.25e+04    9.28e+04
yr_built     -4373.9691    113.673    -38.478      0.000   -4596.774   -4151.164
garage_Yes   -3.016e+04   6045.465     -4.989      0.000    -4.2e+04   -1.83e+04
renovated_1   7.174e+04     1.3e+04      5.517      0.000    4.63e+04    9.72e+04
grade_2       5.428e+05   5.88e+05      0.922      0.356    -6.1e+05     1.7e+06
grade_3       7.339e+04   4.31e+05      0.170      0.865   -7.71e+05    9.18e+05
grade_4      -1.457e+05   4.22e+05     -0.345      0.730   -9.73e+05    6.82e+05
grade_5      -1.579e+05   4.22e+05     -0.375      0.708   -9.84e+05    6.69e+05
grade_6       -4.18e+04   4.22e+05     -0.099      0.921   -8.68e+05    7.85e+05
grade_7       1.573e+05   4.22e+05      0.373      0.709   -6.69e+05    9.84e+05
grade_8       5.112e+05   4.22e+05      1.212      0.226   -3.16e+05    1.34e+06
grade_9       1.008e+06   4.22e+05      2.388      0.017    1.81e+05    1.83e+06
grade_10      1.429e+06   4.23e+05      3.379      0.001       6e+05    2.26e+06
grade_11      1.404e+06   4.29e+05      3.269      0.001    5.62e+05    2.25e+06
grade_12      1.889e+06   5.93e+05      3.187      0.001    7.27e+05    3.05e+06
condition_1   1.211e+05   7.15e+04      1.693      0.090   -1.91e+04    2.61e+05
condition_2   1.312e+05   6.53e+04      2.007      0.045    3072.456    2.59e+05
condition_3   1.525e+05   6.53e+04      2.335      0.020    2.45e+04    2.81e+05
condition_4   1.993e+05   6.56e+04      3.039      0.002    7.07e+04    3.28e+05
==============================================================================
Omnibus:                    10623.350   Durbin-Watson:                   1.963
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            90241.675
Skew:                           1.619   Prob(JB):                         0.00
Kurtosis:                      11.238   Cond. No.                     1.10e+07
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.1e+07. This might indicate that there are
strong multicollinearity or other numerical problems.
```

With this model, it looks like houses with all other factors the same, but with a garage end up being valued at about 30k less then the same home without a garage.

However, while this model explains about 53% of our data, quite a few of our variables, specifically in the "grade" section, don't quite meet our threshold alpha = 0.05. Let's see the difference of garage size is taken into account.

In [26]:
```
#Set X and y
y = df["price"]
X = df[['sqft_living', 'bedrooms', 'bathrooms', 'yr_built', "grade", 'condition', 'renovated', 'garage_size']]

#Dummies for Renovated
X = pd.get_dummies(X, columns=["renovated"], drop_first=True)

#Dummies for grade, and drop "7 Average" as reference category
X = pd.get_dummies(X, columns=["garage_size"])
X = X.drop("garage_size_0", axis=1)

#Dummies for grade, and drop "7 Average" as reference category
X = pd.get_dummies(X, columns=["grade"])
X = X.drop("grade_7", axis=1)

#Dummies for 'condition' and drop 'Average' as reference category
X = pd.get_dummies(X, columns=["condition"])
X = X.drop("condition_2", axis=1)

model = sm.OLS(y, sm.add_constant(X))
results = model.fit()

print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.534
Model:                            OLS   Adj. R-squared:                  0.533
Method:                 Least Squares   F-statistic:                     1374.
Date:                Thu, 12 Jan 2023   Prob (F-statistic):               0.00
Time:                        18:59:11   Log-Likelihood:            -3.9682e+05
No. Observations:               27642   AIC:                         7.937e+05
Df Residuals:                   27618   BIC:                         7.939e+05
Df Model:                          23
Covariance Type:            nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
const          8.985e+06   2.27e+05     39.664      0.000    8.54e+06    9.43e+06
sqft_living     263.9585      5.930     44.511      0.000     252.335     275.582
bedrooms      -3.928e+04   3795.404    -10.350      0.000   -4.67e+04   -3.18e+04
bathrooms      8.194e+04   5161.863     15.874      0.000    7.18e+04    9.21e+04
yr_built      -4302.6286    114.421    -37.604      0.000   -4526.899   -4078.358
renovated_1    6.951e+04    1.3e+04      5.352      0.000    4.41e+04     9.5e+04
garage_size_1  3887.3469   9216.170      0.422      0.673   -1.42e+04     2.2e+04
garage_size_2  -3.84e+04   6517.193     -5.892      0.000   -5.12e+04   -2.56e+04
garage_size_3 -1.124e+05    1.49e+04     -7.547      0.000   -1.42e+05   -8.32e+04
grade_1       -1.641e+05   4.21e+05     -0.390      0.696   -9.89e+05    6.61e+05
grade_2        3.783e+05   4.21e+05      0.899      0.368   -4.46e+05     1.2e+06
grade_3       -8.124e+04   8.91e+04     -0.912      0.362   -2.56e+05    9.34e+04
grade_4       -2.992e+05   2.59e+04    -11.535      0.000    -3.5e+05   -2.48e+05
grade_5       -3.159e+05    1.1e+04    -28.616      0.000   -3.38e+05   -2.94e+05
grade_6       -2.006e+05   6724.100    -29.838      0.000   -2.14e+05   -1.87e+05
grade_8        3.565e+05   8902.655     40.046      0.000    3.39e+05    3.74e+05
grade_9        8.616e+05   1.57e+04     54.849      0.000    8.31e+05    8.92e+05
grade_10       1.293e+06   3.38e+04     38.234      0.000    1.23e+06    1.36e+06
grade_11       1.262e+06    8.1e+04     15.590      0.000     1.1e+06    1.42e+06
grade_12       1.704e+06   4.16e+05      4.093      0.000    8.88e+05    2.52e+06
condition_0   -1.319e+05   6.53e+04     -2.021      0.043    -2.6e+05   -3992.735
condition_1   -1.023e+04   3.02e+04     -0.338      0.735   -6.95e+04     4.9e+04
condition_3    2.275e+04   6245.103      3.642      0.000    1.05e+04     3.5e+04
condition_4    6.813e+04   8699.732      7.831      0.000    5.11e+04    8.52e+04
==============================================================================
Omnibus:                    10622.554   Durbin-Watson:                   1.963
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            90383.905
Skew:                           1.618   Prob(JB):                         0.00
Kurtosis:                      11.246   Cond. No.                     4.94e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.94e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [27]:
```
#Calculating absolute error
from sklearn.metrics import mean_absolute_error

y_pred = results.predict(sm.add_constant(X))
mean_absolute_error(y, y_pred)
```

Out[27]: 283292.8977197938

Data Understanding

Results of model: This model explains about 53.3% of the variance in our data This models F-statistic is statatistically significant compared to our alpha of 0.05 Most of the coefficients are statistically significant when compared to our alpha of .05

Interpretations: For a house with no garage, of average grade and condition, and with no renovations we would expect the house to about 70k less than a home that is renovated. We expect that same house to sell for about 4k more with a 1-car garage For each additional 1 square foot in living space size and all other features remaining the same, we would expect the house to gain about $263

Conclusions

According to our models, renovating a home, specifically targeting square footage of living space, will have a significant increase on the value of the home. While it looks like adding a garage can increase the value of a home in some scenarios, it is dependent on the size of garage and other factors. We would be hard pressed to recommend adding a 1-car garage to a home that does not have one, given that the price difference is only about 4k increase.

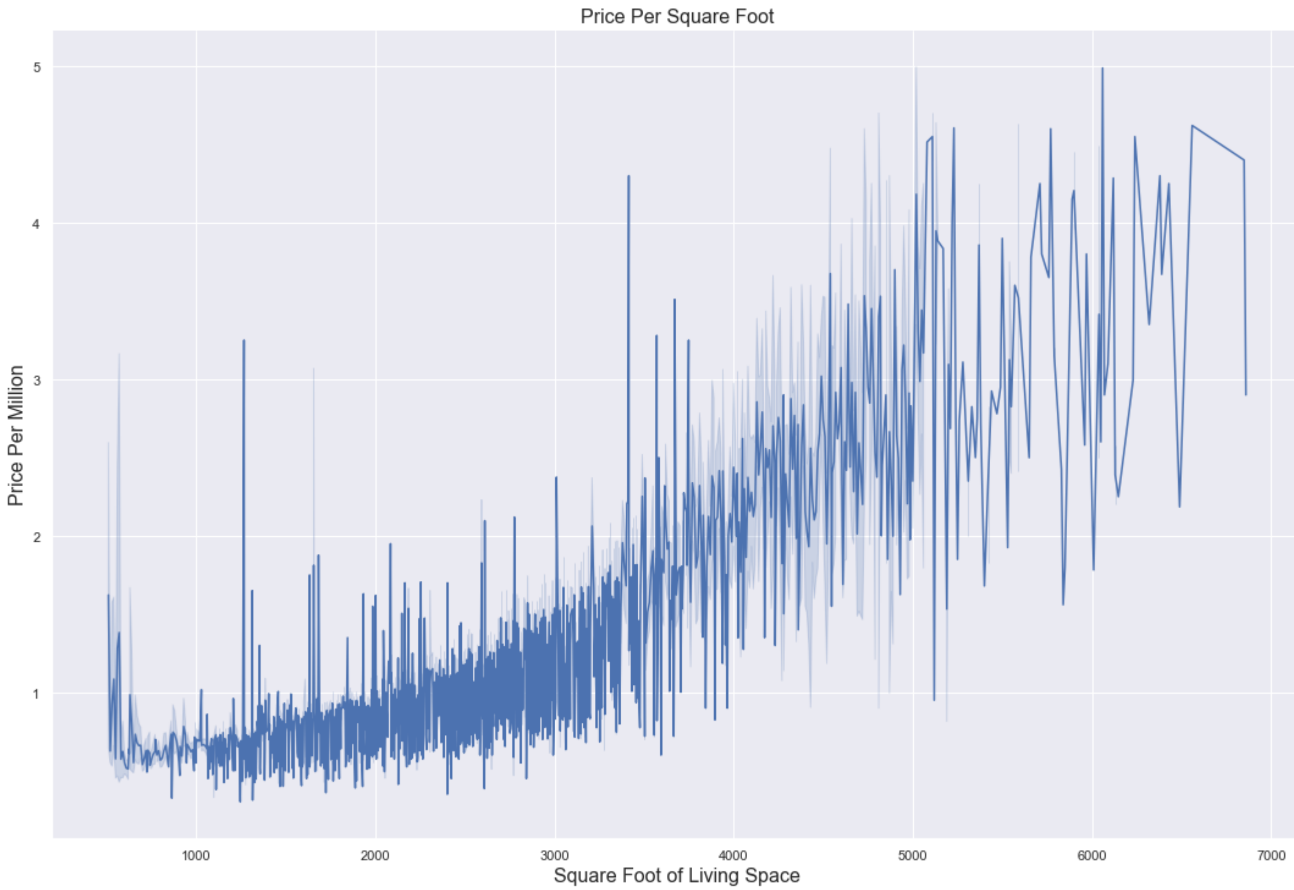
To that end, based on our models and analysis, we recommend that if rennovations are going to occur, its best to target square footage of living space, but if renovations are going to includ the garage, it is probably worth focusing on homes that have no garage and adding a 1-car sized garage, rather than increasing the size of an existing garage.

Next Steps

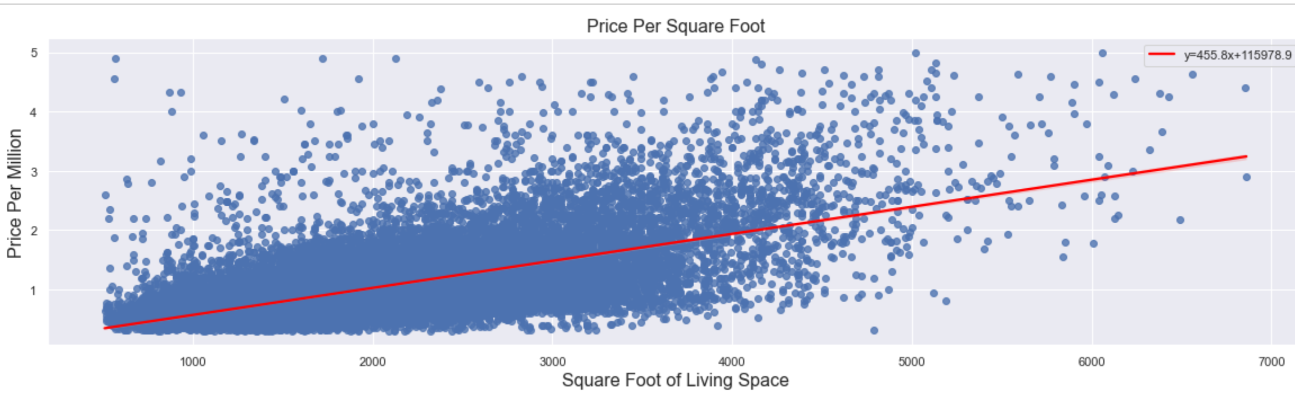
With more time, we could include other factors, like zip cope, that might increase the accuracy of our model and thus update our recommendations.

Additional Visualizations

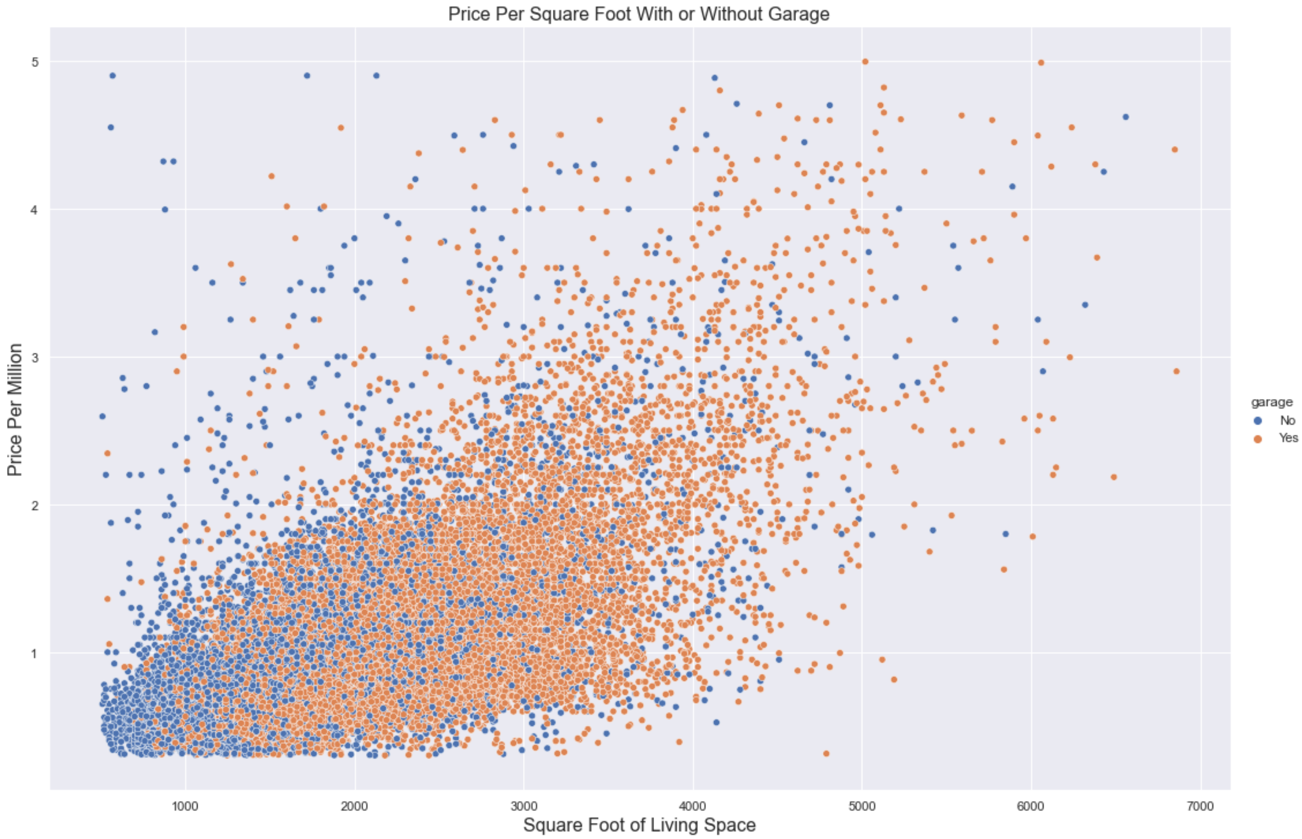
In [28]:
```
#Price per Square Foot of Living Space
price = df['price']/1000000

sns.set_theme(style="darkgrid")

sns.set(rc={"figure.figsize":(20, 5)})
sns.relplot(data=df, x="sqft_living", y=price, kind="line", height=10, aspect=1.5);
plt.title('Price Per Square Foot', fontsize=16)
plt.xlabel('Square Foot of Living Space', fontsize=16)
plt.ylabel('Price Per Million', fontsize=16)
plt.ticklabel_format(style='plain', axis='x')
plt.ticklabel_format(style='plain', axis='y');
```



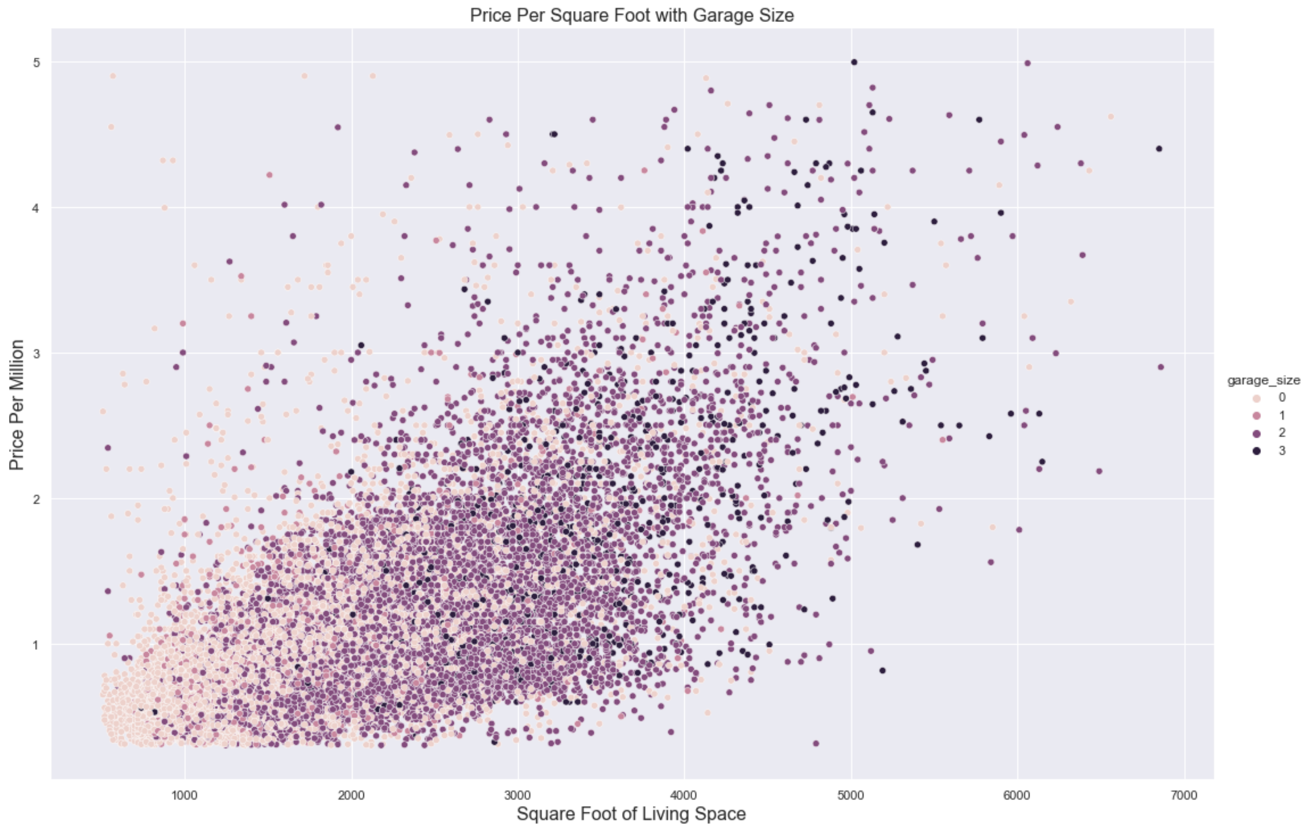
In [29]:
```
slope, intercept, r_value, p_value, std_err = stats.linregress(df['sqft_living'], y)
sns.regplot(x="sqft_living", y=price, data=df, label='',
                line_kws={'label':"y={0:.1f}x+{1:.1f}".format(slope, intercept), "color": "red"})
plt.title('Price Per Square Foot', fontsize=16)
plt.xlabel('Square Foot of Living Space', fontsize=16)
plt.ylabel('Price Per Million', fontsize=16)
plt.ticklabel_format(style='plain', axis='x')
plt.ticklabel_format(style='plain', axis='y');
plt.legend();
```



In [30]:
```
#Price With vs Without Garage
sns.relplot(data=df, x="sqft_living", y=price, hue="garage", height=10, aspect=1.5)
plt.title('Price Per Square Foot With or Without Garage', fontsize=16)
plt.xlabel('Square Foot of Living Space', fontsize=16)
plt.ylabel('Price Per Million', fontsize=16)
plt.ticklabel_format(style='plain', axis='x')
plt.ticklabel_format(style='plain', axis='y');
```



In [31]:
```
#Price vs Size of Garage
sns.relplot(data=df, x="sqft_living", y=price, hue="garage_size", height=10, aspect=1.5);
plt.title('Price Per Square Foot with Garage Size', fontsize=16)
plt.xlabel('Square Foot of Living Space', fontsize=16)
plt.ylabel('Price Per Million', fontsize=16)
plt.ticklabel_format(style='plain', axis='x')
plt.ticklabel_format(style='plain', axis='y');
```



In [ ]: