Final Project Submission - Phase 2

- Student name: A. Utku Kale & Kevin Fagan
- Student pace: NYC/Full time
- Scheduled project review date/time: 2/17/23
- Instructor name: Brendan Hutchinson & Joseph Mata
- Blog post URL: https://medium.com/@utkukale (https://medium.com/@utkukale) / https://medium.com/@kevinmfagan_94380 (https://medium.com/@kevinmfagan_94380)

Business Problem

Our stakeholder Acme Construction Co. is looking for investment opportunities and houses to renovate and re-sell in King County, Washington.

Here we will conduct exploratory data analysis on housing sale data in King County Washington from June 2021 to June 2022. We will create a multiple linear regression model to predict real estate value. The goal is to identify the best areas in the county to purchase, renovate, and resell homes.

Data Understanding

The dataset sources we use for this analysis are the following.
King County Open Data: https://data.kingcounty.gov/ (https://data.kingcounty.gov/)
King County GIS Open Data: https://gis-kingcounty.opendata.arcgis.com (https://gis-kingcounty.opendata.arcgis.com)

Let's do some exploratory data analysis to have a better undertanding.

In [1]:

```
## Importing necessary libraries

# Suppress future and deprecation warnings
import warnings
warnings.filterwarnings("ignore", category = FutureWarning)
warnings.filterwarnings("ignore", category = DeprecationWarning)

# Standard Packages
import pandas as pd
import numpy as np
import datetime
pd.set_option('display.max_columns', None)

# Viz Packages
import seaborn as sns
import matplotlib.pyplot as plt

# Scipy Stats
import scipy.stats as stats

# Statsmodel Api
import statsmodels.api as sm
from statsmodels.formula.api import ols

# SKLearn Modules
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import RFE
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
import sklearn.metrics as metrics


# Location visualization
import folium
from folium.plugins import MarkerCluster
import streamlit as st
from streamlit_folium import folium_static
```

In [2]:

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

In [3]:
```
1 df.columns
```

Out[3]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'greenbelt', 'nuisance', 'view
',
       'condition', 'grade', 'heat_source', 'sewer_system', 'sqft_above',
       'sqft_basement', 'sqft_garage', 'sqft_patio', 'yr_built',
       'yr_renovated', 'address', 'lat', 'long'],
      dtype='object')
```

In [4]:
```
1 df.head()
```

Out[4]:

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

In [5]:
```
1 df.describe()
```

Out[5]:

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

In [6]:
```
1 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
```

We have latitude and longtitude data, let's create a map to visualize.

In [7]:

```
m = folium.Map(location=[df['lat'].mean(), df['long'].mean()],
zoom_start=12)
marker_cluster = MarkerCluster().add_to(m)
for index, row in df.iterrows():
    folium.Marker(location=[row['lat'], row['long']],
        color='green',
        clustered_marker=True,
        icon=folium.Icon(color='green', icon='info-sign')
    ).add_to(marker_cluster)
display(m)
```

We have data from all over the country! We need to filter it using King County zip codes.

In [8]:

```
# Pull list of zipcodes associatied with King County, dataset from
https://data.kingcounty.gov/
z =
pd.read_csv('data/Zipcodes_for_King_County_and_Surrounding_Area_(Shor
elines)___zipcode_shore_area.csv')
z = z[z['COUNTY_NAME'] == 'King County']
```

In [9]:

```
# Creating column for zipcode using 'address' column.
df['zipcode'] = 1
for i in range(len(df['address'])):
    df.loc[i, 'zipcode'] = df['address'][i][-20:-15]
df['zipcode'] = df['zipcode'].astype(int)
```

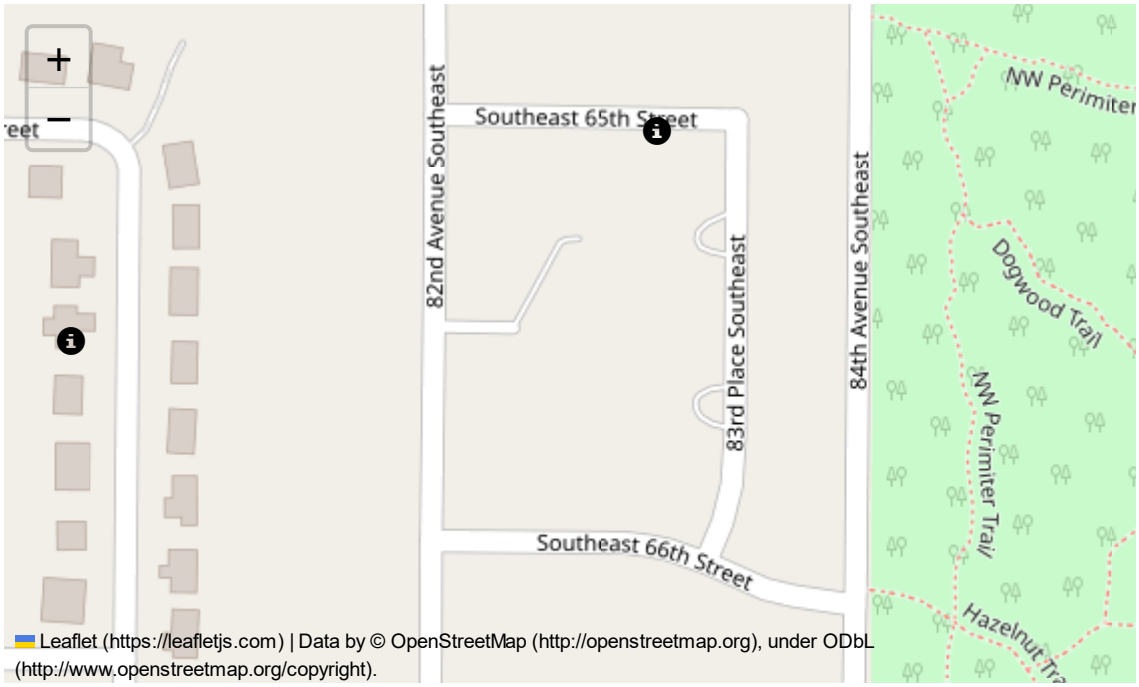
In [10]:

```
# Filtering main DataFrame with zipcodes in King County
in_king_county_mask = df['zipcode'].isin(z['ZIPCODE'])
df_king = df[in_king_county_mask]
df_king = df_king.drop_duplicates()
```

df_king is our DataFrame for King County houses! Let's create another map to make sure and do some exploratory analysis on the new dataframe.

In [11]:

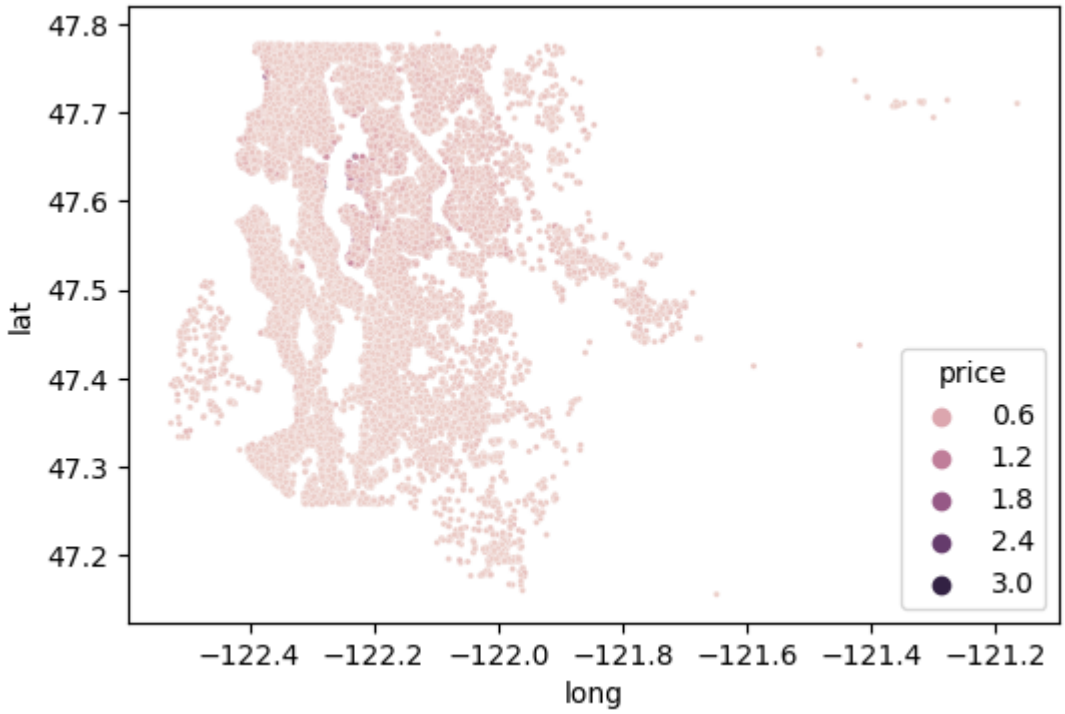
```
m = folium.Map(location=[df_king['lat'].mean(),
df_king['long'].mean()], zoom_start=12)
marker_cluster = MarkerCluster().add_to(m)
for index, row in df_king.iterrows():
    folium.Marker(location=[row['lat'], row['long']],
        color='green',
        clustered_marker=True,
        icon=folium.Icon(color='green', icon='info-sign')
    ).add_to(marker_cluster)
display(m)
```



Alright, this is King County! Let's create a scatter plot with coordinates to see what we got.

In [12]:

```
1
2 plt.figure(figsize=(6,4))
3 sns.scatterplot(x='long', y='lat', data=df_king, hue='price', s = 5);
4 plt.show()
```



In [13]:

```
1 df_king.shape
```

Out[13]: (29212, 26)

In [14]:

```
1 df_king.describe()
```

Out[14]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot
count	2.921200e+04	2.921200e+04	29212.000000	29212.000000	29212.000000	2.921200e+04
mean	4.535819e+09	1.112556e+06	3.435232	2.331867	2130.620053	1.720376e+04
std	2.881972e+09	8.955071e+05	0.978605	0.895807	977.230016	6.128768e+04
min	1.000055e+06	2.736000e+04	0.000000	0.000000	3.000000	4.020000e+02
25%	2.085201e+09	6.450000e+05	3.000000	2.000000	1440.000000	5.000000e+03
50%	3.873950e+09	8.670000e+05	3.000000	2.500000	1940.000000	7.560000e+03
75%	7.286650e+09	1.310000e+06	4.000000	3.000000	2640.000000	1.078075e+04
max	9.904000e+09	3.075000e+07	13.000000	10.500000	15360.000000	3.253932e+06

In [15]:
```
1 df_king.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29212 entries, 0 to 30154
Data columns (total 26 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             29212 non-null  int64  
 1   date           29212 non-null  object 
 2   price          29212 non-null  float64
 3   bedrooms       29212 non-null  int64  
 4   bathrooms      29212 non-null  float64
 5   sqft_living    29212 non-null  int64  
 6   sqft_lot       29212 non-null  int64  
 7   floors         29212 non-null  float64
 8   waterfront     29212 non-null  object 
 9   greenbelt      29212 non-null  object 
 10  nuisance       29212 non-null  object 
 11  view           29212 non-null  object 
 12  condition      29212 non-null  object 
 13  grade          29212 non-null  object 
 14  heat_source    29183 non-null  object 
 15  sewer_system   29199 non-null  object 
 16  sqft_above     29212 non-null  int64  
 17  sqft_basement  29212 non-null  int64  
 18  sqft_garage    29212 non-null  int64  
 19  sqft_patio     29212 non-null  int64  
 20  yr_built       29212 non-null  int64  
 21  yr_renovated   29212 non-null  int64  
 22  address        29212 non-null  object 
 23  lat            29212 non-null  float64
 24  long           29212 non-null  float64
 25  zipcode        29212 non-null  int32  
dtypes: float64(5), int32(1), int64(10), object(10)
memory usage: 5.9+ MB
```

We have non-numerical variables, we need to encode them and change into numerical variables to be able to run a multiple regression model.

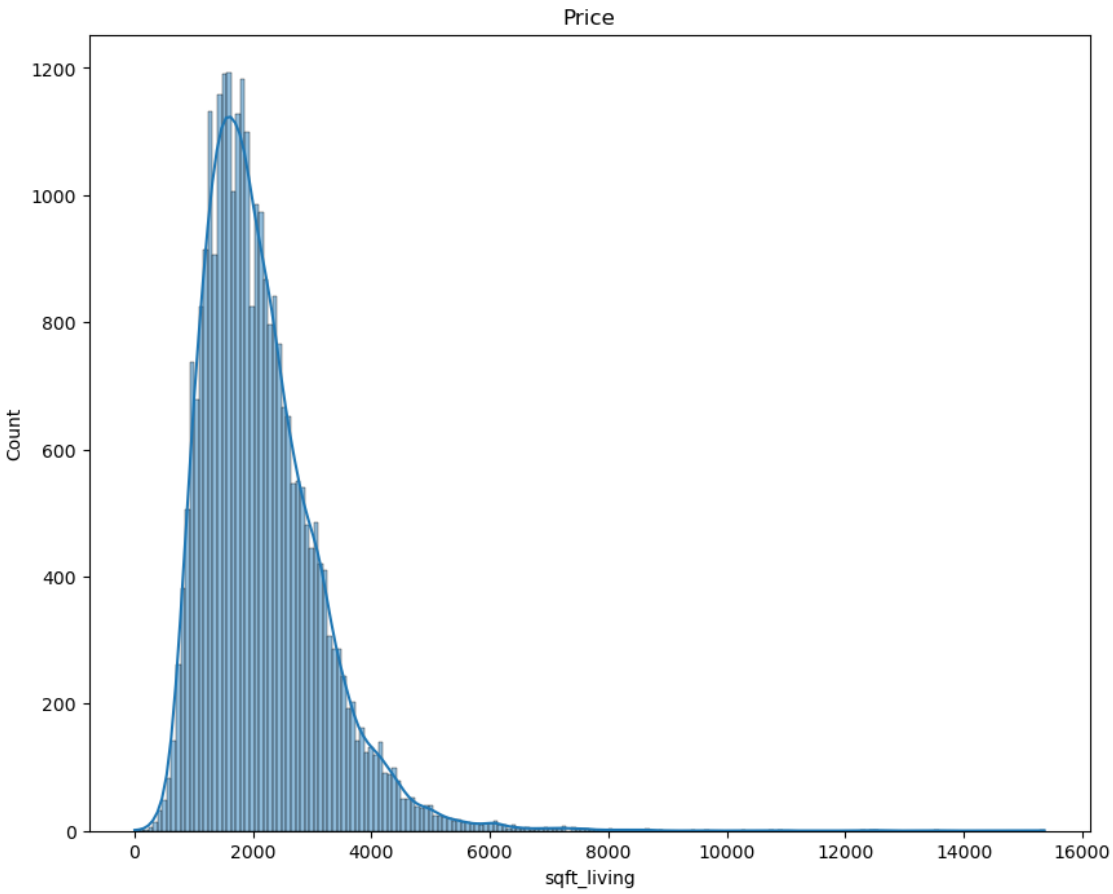
Creating a histogram to observe our target value distribution.

In [16]:

```
fig, ax = plt.subplots(figsize=(10, 8))
sns.histplot(df_king,x='sqft_living', kde=True)
ax.set_title("Price")

```

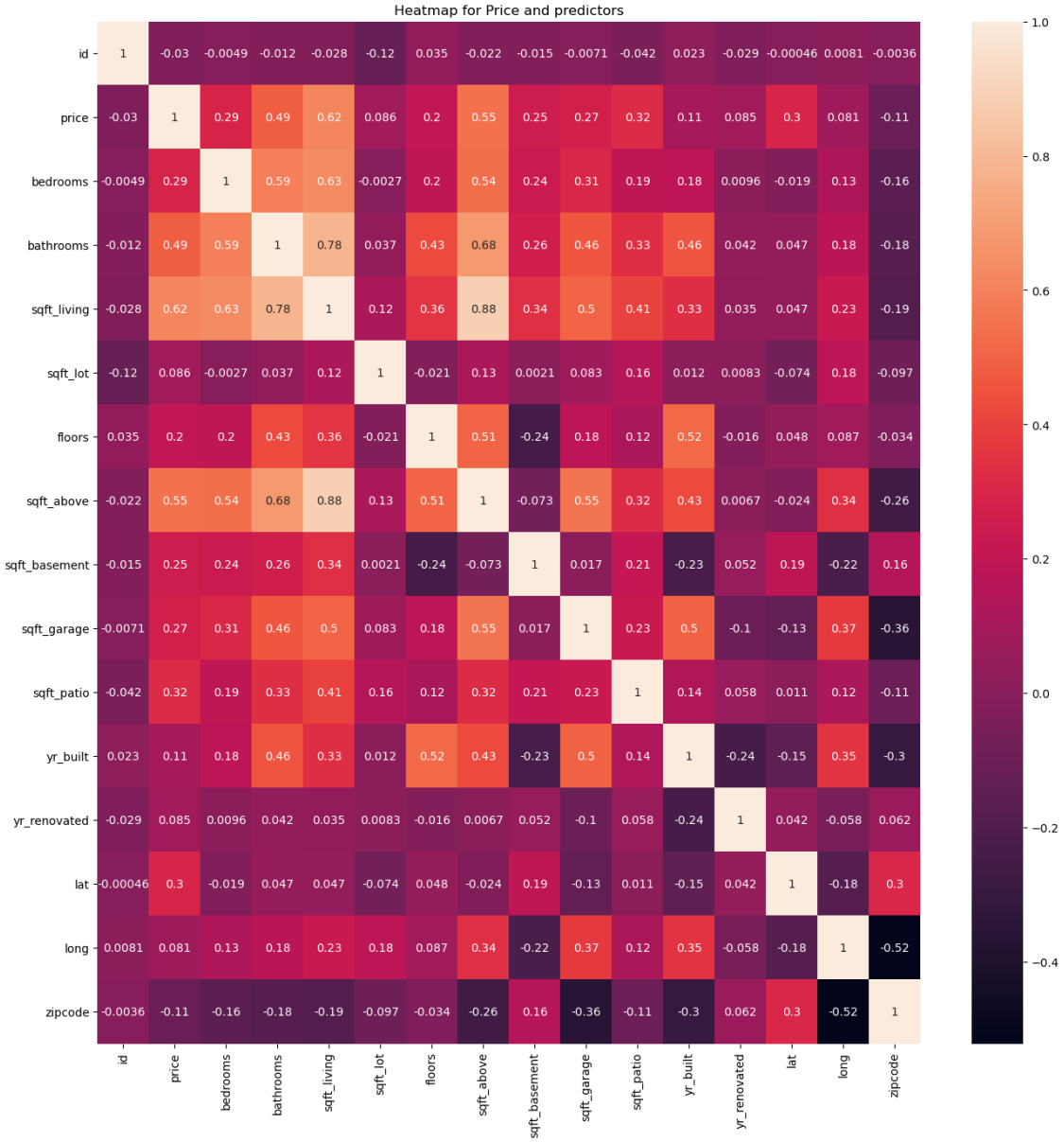
Out[16]: Text(0.5, 1.0, 'Price')



Creating a heatmap to visualize and understand the correlation between features and our target.

In [17]:

```
fig, ax = plt.subplots(figsize=(16,16))
corr = df_king.corr()
sns.heatmap(corr, annot = True)
ax.set_title("Heatmap for Price and predictors");
plt.show()
```



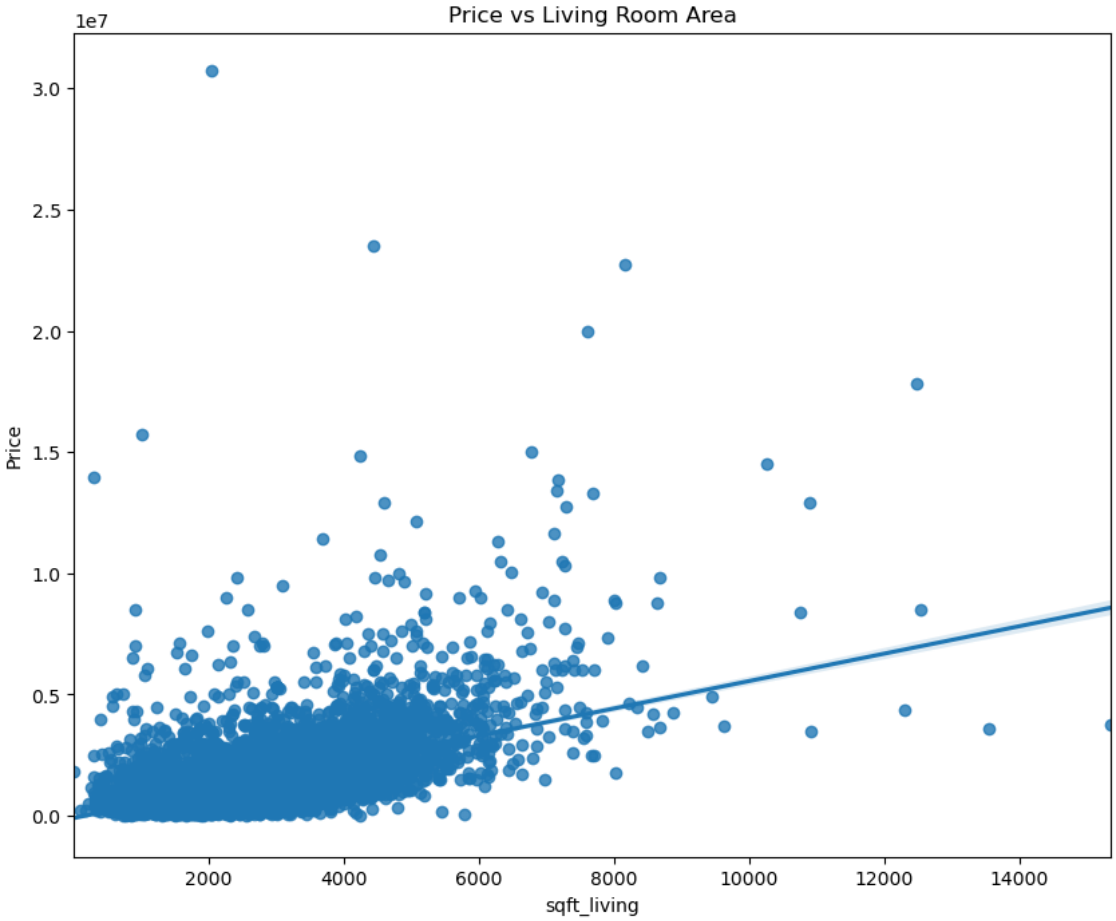
Insight 1:

Living room area(sqft_living) has the biggest correlation with price, creating a scatter plot to investigate further.

In [18]:

```
1 fig, ax = plt.subplots(figsize=(10, 8))
2 sns.regplot(data = df_king,x='sqft_living',y='price')
3 ax.set_xlabel('sqft_living')
4 ax.set_ylabel('Price')
5 ax.set_title("Price vs Living Room Area")
```

Out[18]: Text(0.5, 1.0, 'Price vs Living Room Area')



Let's create our base regression model to see the regression between our target and sqft_living.

In [19]:

```
1 y = df_king["price"]
2 X = df_king.drop("price", axis=1)
3 X_train, X_test, y_train, y_test = train_test_split(X, y,
  random_state=42)
```

In [20]:
```
model = LinearRegression()
model_OLS = sm.OLS(endog=y_train,
exog=sm.add_constant(X_train['sqft_living'])).fit()
model_OLS.summary()
```

Out[20]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.394
Model:	OLS	**Adj. R-squared:**	0.394
Method:	Least Squares	**F-statistic:**	1.426e+04
Date:	Tue, 21 Feb 2023	**Prob (F-statistic):**	0.00
Time:	10:21:29	**Log-Likelihood:**	-3.2593e+05
No. Observations:	21909	**AIC:**	6.519e+05
Df Residuals:	21907	**BIC:**	6.519e+05
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.099e+05	1.13e+04	-9.723	0.000	-1.32e+05	-8.77e+04

In [21]:
```
#Let's take a look at our error metrics.
```

In [22]:
```
y = df_king[["price"]]
X = df_king[['sqft_living']]
```

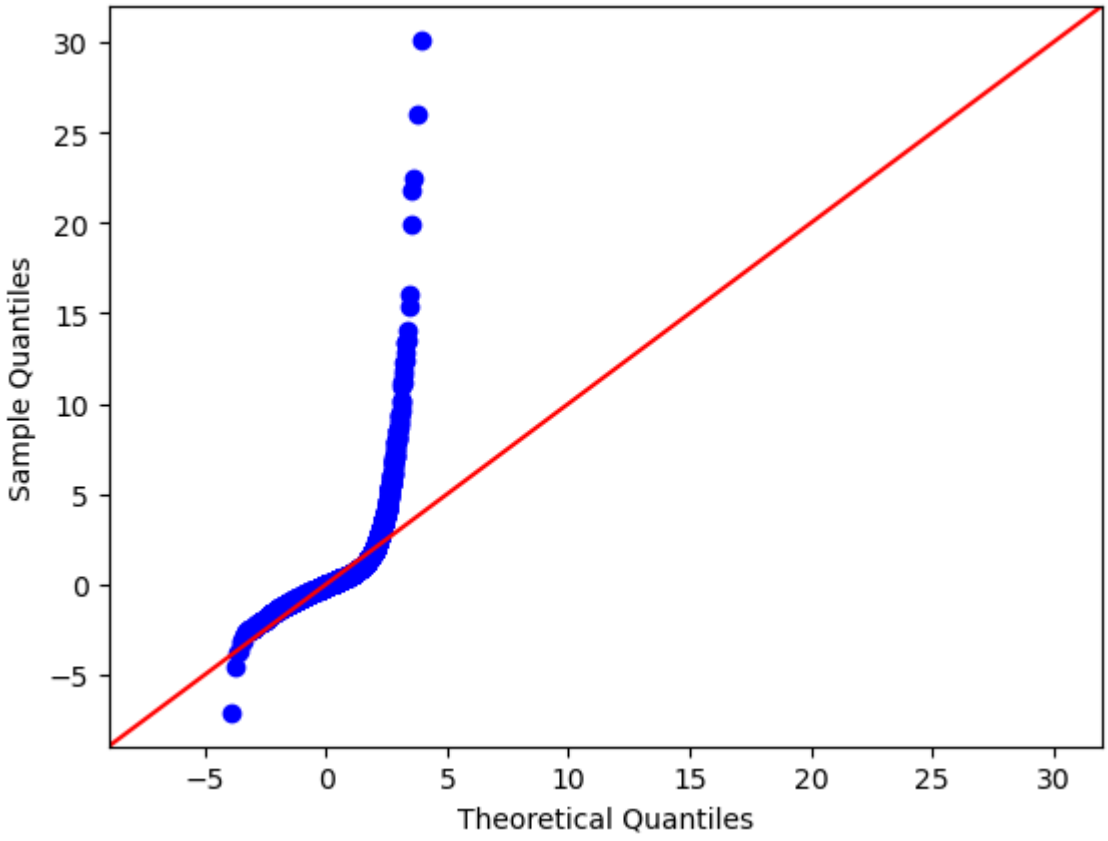
In [23]:
```
model = LinearRegression()
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.25, random_state=42)
```

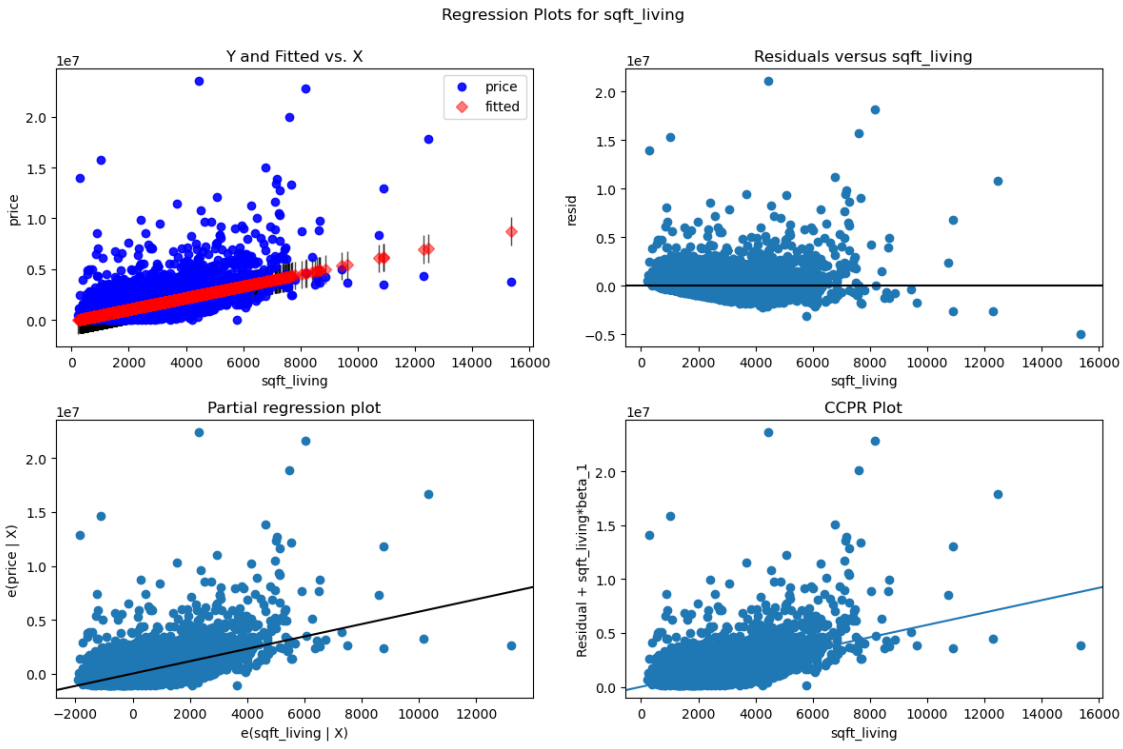
In [24]:
```
model = LinearRegression()
model.fit(X_train, y_train)
train_preds = model.predict(X_train)
score = metrics.r2_score(y_train, train_preds)
coef = (X_train.columns, model.coef_)
mae = metrics.mean_absolute_error(y_train, train_preds)
mse = metrics.mean_squared_error(y_train, train_preds)
#     return
print(f'R2 score:{score}, Mean absolute error:{mae}, Mean squared
error:{mse}'),
print(pd.DataFrame(data = model.coef_, columns = X_train.columns))
```

```
R2 score:0.39434784877633855, Mean absolute error:402057.3795837555, Mean
squared error:488565667999.3498
   sqft_living
0   574.397981
```

In [25]:

```
# We can see residuals below, the model needs improvement.
fig = sm.graphics.qqplot(model_OLS.resid, line='45',fit=True);
sm.graphics.plot_regress_exog(model_OLS,'sqft_living',
fig=plt.figure(figsize=(12,8)));
```





Feature engineering and Data cleaning

Let's use feature engineering to create a new column for overall quality of the house.

In an effort to find Zip codes where flipping a house returns the most value, we will classify houses based on 'condition' and 'grade'.

According to King County's descriptions of the column names, `condition` is how good the overall condition of the house is from a maintenance perspective. `grade` is the overall grade of the house, related to the construction and design of the house.

BUILDING CONDITION Relative to age and grade. Coded 1-5.

1 = Poor- Worn out. Repair and overhaul needed on painted surfaces, roofing, plumbing, heating and numerous functional inadequacies. Excessive deferred maintenance and abuse, limited value-in-use, approaching abandonment or major reconstruction; reuse or change in occupancy is imminent. Effective age is near the end of the scale regardless of the actual chronological age.

2 = Fair- Badly worn. Much repair needed. Many items need refinishing or overhauling, deferred maintenance obvious, inadequate building utility and systems all shortening the life expectancy and increasing the effective age.

3 = Average- 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.

4 = Good- No obvious maintenance required but neither is everything new. Appearance and utility are above the standard and the overall effective age will be lower than the typical property.

5= Very Good- All items well maintained, many having been overhauled and repaired as they have shown signs of wear, increasing the life expectancy and lowering the effective age with little deterioration or obsolescence evident with a high degree of utility.

BUILDING GRADE Represents the construction quality of improvements. Grades run from grade 1 to 13. Generally defined as:

1-3 Falls short of minimum building standards. Normally cabin or inferior structure.

4 Generally older, low quality construction. Does not meet code.

5 Low construction costs and workmanship. Small, simple design.

6 Lowest grade currently meeting building code. Low quality materials and simple designs.

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

8 Just above average in construction and design. Usually better materials in both the exterior

and interior finish work.

9 Better architectural design with extra interior and exterior design and quality.

10 Homes of this quality generally have high quality features. Finish work is better and more design quality is seen in the floor plans. Generally have a larger square footage.

11 Custom design and higher quality finish work with added amenities of solid woods, bathroom fixtures and more luxurious options.

12 Custom design and excellent builders. All materials are of the highest quality and all conveniences are present.

13 Generally custom designed and built. Mansion level. Large amount of highest quality cabinet work, wood trim, marble, entry ways etc.

Transforming categorical variables to numerical using encoding.

In [26]:
```
df_king['gradeno'] = df_king['grade'].map(lambda x: x[0:2])
df_king['gradeno'] = df_king['gradeno'].astype('int')
```

In [27]:
```
dict = {'Poor':1, 'Fair':2, 'Average':3, 'Good':4, 'Very Good':5}

df_king=df_king.replace({"condition": dict})
```

In [28]:
```
def get_class(row):
    """Classify properties into 3 groups. 0 includes all 'Poor' and 'Fair' condition rows, and well as 'Average' condition where grade is 7 or less.
    Class 2 includes all 'Very good' condition rows, as well as 'Good' condition where grade is 9 or more.
    Class 1 includes all else."""
    if row['condition'] <= 2:
        return 0
    elif row['condition'] == 3 and row['gradeno'] <= 7:
        return 0
    elif row['condition'] == 5:
        return 2
    elif row['condition'] == 4 and row['gradeno'] >= 9:
        return 2
    else:
        return 1

df_king['class'] = df_king.apply(get_class, axis=1)
```

In [29]:
```
1 df_king.head()
```

Out[29]:

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

Checking null values.

In [30]:
```
1 df_king.isnull().sum()
```

Out[30]:
```
id                0
date              0
price             0
bedrooms          0
bathrooms         0
sqft_living       0
sqft_lot          0
floors            0
waterfront        0
greenbelt         0
nuisance          0
view              0
condition         0
grade             0
heat_source      29
sewer_system     13
sqft_above        0
sqft_basement     0
sqft_garage       0
sqft_patio        0
yr_built          0
yr_renovated      0
address           0
lat               0
long              0
zipcode           0
gradeno           0
class             0
dtype: int64
```

Converting 'waterfront', 'greenbelt', 'nuisance' into numerical using label encoding.

In [31]:

```
le = preprocessing.LabelEncoder()
le_cols = ['waterfront', 'greenbelt', 'nuisance']
df_king[le_cols] = df_king[le_cols].apply(lambda x:
le.fit_transform(x))
df_king['view'] = le.fit_transform(df_king['view'])
df_king.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29212 entries, 0 to 30154
Data columns (total 28 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             29212 non-null  int64
 1   date           29212 non-null  object
 2   price          29212 non-null  float64
 3   bedrooms       29212 non-null  int64
 4   bathrooms      29212 non-null  float64
 5   sqft_living    29212 non-null  int64
 6   sqft_lot       29212 non-null  int64
 7   floors         29212 non-null  float64
 8   waterfront     29212 non-null  int32
 9   greenbelt      29212 non-null  int32
 10  nuisance       29212 non-null  int32
 11  view           29212 non-null  int32
 12  condition      29212 non-null  int64
 13  grade          29212 non-null  object
 14  heat_source    29183 non-null  object
 15  sewer_system   29199 non-null  object
 16  sqft_above     29212 non-null  int64
 17  sqft_basement  29212 non-null  int64
 18  sqft_garage    29212 non-null  int64
 19  sqft_patio     29212 non-null  int64
 20  yr_built       29212 non-null  int64
 21  yr_renovated   29212 non-null  int64
 22  address        29212 non-null  object
 23  lat            29212 non-null  float64
 24  long           29212 non-null  float64
 25  zipcode        29212 non-null  int32
 26  gradeno        29212 non-null  int32
 27  class          29212 non-null  int64
dtypes: float64(5), int32(6), int64(12), object(5)
memory usage: 5.8+ MB
```

Creating a age column using yr_built.

In [32]:

```
df_king['age'] = 2023 - df_king['yr_built']
df_king.drop(columns = 'yr_built', inplace = True)
```

In [33]:

```
df_king2 = df_king
```

In [34]:
```
1 df_king2.head()
```

Out[34]:

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

Updating zipcodes column using OneHotEncoder.

In [35]:
```
1 ohe_cols = ['zipcode']
2
3 ohe = OneHotEncoder(sparse = False, drop='first')
4 train_ohe_df = ohe.fit_transform(df_king[ohe_cols])
5 train_ohe_df = pd.DataFrame(train_ohe_df, columns =
  ohe.get_feature_names(), index = df_king.index)
6
```

In [36]:

```
df_king2 = pd.concat([df_king, train_ohe_df], 1)
df_king2
```

Out[36]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
0	7399300360	5/24/2022	675000.0	4	1.0	1180	7140	1.(
1	8910500230	12/13/2021	920000.0	5	2.5	2770	6703	1.(
2	1180000275	9/29/2021	311000.0	6	2.0	2880	6156	1.(
3	1604601802	12/14/2021	775000.0	3	3.0	2160	1400	2.(
4	8562780790	8/24/2021	592500.0	2	2.0	1120	758	2.(
...	...	...	...	...	...	...	...	..
30150	7834800180	11/30/2021	1555000.0	5	2.0	1910	4000	1.5
30151	194000695	6/16/2021	1313000.0	3	2.0	2020	5800	2.(
30152	7960100080	5/27/2022	800000.0	3	2.0	1620	3600	1.(
30153	2781280080	2/24/2022	775000.0	3	2.5	2570	2889	2.(

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
20154	0557800100	4/20/2022	500000.0	3	1.5	1200	11058	1.(

Let's remove the columns we're not going to use in our regression model.

In [37]:
```
1 df_king2.columns[:30]
```

Out[37]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'greenbelt', 'nuisance', 'view
',
       'condition', 'grade', 'heat_source', 'sewer_system', 'sqft_above',
       'sqft_basement', 'sqft_garage', 'sqft_patio', 'yr_renovated', 'add
ress',
       'lat', 'long', 'zipcode', 'gradeno', 'class', 'age', 'x0_98002',
       'x0_98003'],
      dtype='object')
```

In [38]:
```
1 df_king2.drop(columns=['id', 'date',
2         'grade', 'heat_source', 'sewer_system', 'yr_renovated',
'address','lat', 'long', 'zipcode'],inplace = True)
```

Multilinear Regression

In [39]:
```
1 y = df_king2["price"]
2 X = df_king2.drop("price", axis=1)
3 X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y,
random_state=42)
```

In [40]:

```
model = LinearRegression()
model_OLS = sm.OLS(endog=y_train2,
exog=sm.add_constant(X_train2)).fit()
model_OLS.summary()
```

Out[40]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.674
Model:	OLS	**Adj. R-squared:**	0.672
Method:	Least Squares	**F-statistic:**	479.2
Date:	Tue, 21 Feb 2023	**Prob (F-statistic):**	0.00
Time:	10:21:33	**Log-Likelihood:**	-3.1915e+05
No. Observations:	21909	**AIC:**	6.385e+05
Df Residuals:	21814	**BIC:**	6.392e+05
Df Model:	94		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.074e+06	6.72e+04	-15.968	0.000	-1.21e+06	-9.42e+05

In [41]:

```
model = LinearRegression()
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.25, random_state=42)
```

In [42]:

```
model = LinearRegression()
model.fit(X_train2, y_train2)
train_preds = model.predict(X_train2)
score = metrics.r2_score(y_train2, train_preds)
coef = (X_train2.columns, model.coef_)
mae = metrics.mean_absolute_error(y_train2, train_preds)
mse = metrics.mean_squared_error(y_train2, train_preds)
#     return
print(f'R2 score:{score}, Mean absolute error:{mae}, Mean squared
error:{mse}'),
print(pd.DataFrame(data = model.coef_, columns = X_train.columns))
```

```
R2 score:0.6737413306321627, Mean absolute error:264656.6816989318, Mean
squared error:263185368727.29117

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[42], line 10
      8 #     return
      9 print(f'R2 score:{score}, Mean absolute error:{mae}, Mean squared e
rror:{mse}'),
---> 10 print(pd.DataFrame(data = model.coef_, columns = X_train.columns))

File ~\anaconda3\envs\learn-env\lib\site-packages\pandas\core\frame.py:694
in DataFrame.__init__(self, data, index, columns, dtype, copy)
    684         mgr = dict_to_mgr(
    685             # error: Item "ndarray" of "Union[ndarray, Series, Inde
x]" has no
    686             # attribute "name"
   (...)
    691             typ=manager,
    692         )
```

In our second model, our adjested R2 is bigger so our 2nd model can explain variation better than our first one. (%67)

Let's create another model with feature engineering.

In [ ]:

```
# Find difference between class 2 and class 0 median property values
per zip, neglecting zip codes with null values
df_median_price = pd.pivot_table(df_king, values='price',
index='zipcode', columns='class', aggfunc='median')
df_median_price = df_median_price.dropna()
df_median_price = df_median_price.reset_index()
df_median_price['diff'] = df_median_price[2] - df_median_price[0]
df_median_price = df_median_price.rename(columns={'zipcode':
'ZIPCODE'})
```

In [ ]:
```
# View the top 5 in difference in value
df_median_price.sort_values('diff', ascending=False).head()
```

We will focus on the top 5 zipcodes.

98039, $2,585,000, Medina

98004, $1,558,000, Bellvue

98112, $1,325,000, Seattle

98077, $1,008,000, Woodinville

98105, $992,500, Seattle

Time for our 3rd model.

In [ ]:
```
y = df_king2["price"]
X = df_king2[['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
'floors',
       'waterfront', 'greenbelt', 'nuisance', 'view', 'condition',
       'sqft_above', 'sqft_basement', 'sqft_garage', 'sqft_patio',
'gradeno',
       'class', 'age',
'zipcode_98039','zipcode_98004','zipcode_98105','zipcode_98112','zipc
ode_98077']]
X_train3, X_test3, y_train3, y_test3 = train_test_split(X, y,
random_state=42)
```

In [ ]:
```
model = LinearRegression()
model_OLS = sm.OLS(endog=y_train3,
exog=sm.add_constant(X_train3)).fit()
model_OLS.summary()
```

In [ ]:
```
model = LinearRegression()
X_train3, X_test3, y_train3, y_test3 = train_test_split(
X, y, test_size=0.25, random_state=42)
```

In [ ]:
```
model = LinearRegression()
model.fit(X_train3, y_train3)
train_preds = model.predict(X_train3)
score = metrics.r2_score(y_train2, train_preds)
coef = (X_train3.columns, model.coef_)
mae = metrics.mean_absolute_error(y_train3, train_preds)
mse = metrics.mean_squared_error(y_train3, train_preds)
#     return
print(f'R2 score:{score}, Mean absolute error:{mae}, Mean squared error:{mse}'),
print(pd.DataFrame(data = model.coef_, columns = X_train.columns))
```

Model 4

Dropping the outliers on price and sqft_living

In [ ]:
```
df_king4 = df_king2.copy()
```

In [ ]:
```
for x in ['price','sqft_living']:
        q75,q25 = np.percentile(df_king4.loc[:,x],[75,25])
        intr_qr = q75-q25
        max = q75+(1.5*intr_qr)
        min = q25-(1.5*intr_qr)
        df_king4.loc[df_king4[x] < min,x] = np.nan
        df_king4.loc[df_king4[x] > max,x] = np.nan
df_king4 = df_king4.dropna(subset = ['price','sqft_living'] )
```

In [ ]:
```
y = df_king4["price"]
X = df_king4.drop("price", axis=1)
X_train4, X_test4, y_train4, y_test4 = train_test_split(X, y, random_state=42)
```

In [ ]:
```
model = LinearRegression()
model_OLS = sm.OLS(endog=y_train4, exog=sm.add_constant(X_train4)).fit()
model_OLS.summary()
```

In [ ]:
```
model = LinearRegression()
X_train4, X_test4, y_train4, y_test4 = train_test_split(
X, y, test_size=0.25, random_state=42)
```

In [ ]:

```
model = LinearRegression()
model.fit(X_train4, y_train4)
train_preds = model.predict(X_train4)
score = metrics.r2_score(y_train4, train_preds)
coef = (X_train4.columns, model.coef_)
mae = metrics.mean_absolute_error(y_train4, train_preds)
mse = metrics.mean_squared_error(y_train4, train_preds)
#     return
print(f'R2 score:{score}, Mean absolute error:{mae}, Mean squared
error:{mse}'),
print(pd.DataFrame(data = model.coef_, columns = X_train.columns))
```

In [ ]:

```
#Running the test date on last model, since it has highest adjusted
R2.
```

In [ ]:

```
y_pred = model_OLS.predict(exog=sm.add_constant(X_test4))
```

In [ ]:

```
metrics.r2_score(y_test4,y_pred)
```

In [ ]:

```
score = metrics.r2_score(y_test4, y_pred)
# coef = (X_test4.columns, model_OLS.coef_)
mae = metrics.mean_absolute_error(y_test4, y_pred)
mse = metrics.mean_squared_error(y_test4, y_pred)
#     return
print(f'R2 score:{score}, Mean absolute error:{mae}, Mean squared
error:{mse}'),
```

CONCLUSION

Our analysis provides valuable insights into the King County housing market and can help the contractor make informed decisions about where to invest.

MODEL 4 provides valuable insights and predictions about King County.