1 A Look into King County Real Estate Market

In [1]:
```
#importing all relevant libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

pd.set_option("display.max_columns", 100)

from scipy import stats
import pandas as pd
import requests
from xml.etree import ElementTree
import numpy as np
import folium

import statsmodels.api as sm
from statsmodels.formula.api import ols
import statsmodels.formula as smf

from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import statsmodels
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from  matplotlib.ticker import FuncFormatter
```

executed in 1.64s, finished 09:21:35 2021-01-25

In [2]:
```
#importing data set into to dfs
df = pd.read_csv('data/kc_house_data.csv')
ogdf = pd.read_csv('data/kc_house_data.csv')
```

executed in 89ms, finished 09:21:35 2021-01-25

1.1 Column Names and descriptions for Kings County Data Set

- **id** - unique identified for a house

- **dateDate** - house was sold
- **pricePrice** - is prediction target
- **bedroomsNumber** - of Bedrooms/House
- **bathroomsNumber** - of bathrooms/bedrooms
- **sqft_livingsquare** - footage of the home
- **sqft_lotsquare** - footage of the lot
- **floorsTotal** - floors (levels) in house
- **waterfront** - House which has a view to a waterfront
- **view** - Has been viewed
- **condition** - How good the condition is (Overall)
- **grade** - overall grade given to the housing unit, based on King County grading system
- **sqft_above** - square footage of house apart from basement
- **sqft_basement** - square footage of the basement
- **yr_built** - Built Year
- **yr_renovated** - Year when house was renovated
- **zipcode** - zip
- **lat** - Latitude coordinate
- **long** - Longitude coordinate
- **sqft_living15** - The square footage of interior housing living space for the nearest 15 neighbors
- **sqft_lot15** - The square footage of the land lots of the nearest 15 neighbors

In [3]:
```
#testing for multicollinearity
test = df.corr().abs().stack().reset_index().sort_values(0,ascending=False)
test['pairs'] = list(zip(test.level_0,test.level_1))
test.set_index(['pairs'], inplace=True)
test.drop(columns=['level_1','level_0'], inplace=True)
test.columns = ['cc']
test.drop_duplicates(inplace=True)
```

executed in 28ms, finished 09:21:35 2021-01-25

In [4]:
```
1 test.sort_values('cc',ascending=False, inplace=True)
2 test[test.cc >.75]
```
executed in 14ms, finished 09:21:35 2021-01-25

Out[4]:

pairs	cc
(id, id)	1.000000
(sqft_living, sqft_above)	0.876448
(grade, sqft_living)	0.762779
(sqft_living, sqft_living15)	0.756402
(sqft_above, grade)	0.756073
(sqft_living, bathrooms)	0.755758

In [5]:
```
1 #dropping sqft above due to correlation with sqft living
2 df.drop(columns=["sqft_above"], inplace= True)
```
executed in 7ms, finished 09:21:36 2021-01-25

In [6]:
```
1 df.reset_index(inplace=True)
```
executed in 4ms, finished 09:21:36 2021-01-25

In [7]:
```
1 df.drop('index',axis=1,inplace=True)
```
executed in 8ms, finished 09:21:36 2021-01-25

In [8]:
```
1 df.info()
```
executed in 10ms, finished 09:21:37 2021-01-25

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21597 non-null  int64  
 1   date           21597 non-null  object 
 2   price          21597 non-null  float64
 3   bedrooms       21597 non-null  int64  
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64  
 6   sqft_lot       21597 non-null  int64  
 7   floors         21597 non-null  float64
 8   waterfront     19221 non-null  float64
 9   view           21534 non-null  float64
 10  condition      21597 non-null  int64  
 11  grade          21597 non-null  int64  
 12  sqft_basement  21597 non-null  object 
 13  yr_built       21597 non-null  int64  
```

In [9]:
```
#looking at unique values for all columns
for col in df.columns:
    print(col)
    print(df[col].value_counts(normalize = True, ascending=False).head(5))
    print("------------------------------------------------------------")
```

executed in 39ms, finished 09:21:37 2021-01-25

```
0.0    0.992404
1.0    0.007596
Name: waterfront, dtype: float64
------------------------------------------------------------
view
0.0    0.901923
2.0    0.044441
3.0    0.023591
1.0    0.015325
4.0    0.014721
Name: view, dtype: float64
------------------------------------------------------------
condition
3    0.649164
4    0.262861
5    0.078761
2    0.007871
1    0.001343
Name: condition, dtype: float64
------------------------------------------------------------
```

In [10]:
```
#replacing ? with 0 for sqft basement
df['sqft_basement'] = df.sqft_basement.replace(to_replace= '?', value = '0')
ogdf['sqft_basement'] = ogdf.sqft_basement.replace(to_replace= '?', value = '0')
```

executed in 6ms, finished 09:21:37 2021-01-25

In [11]:
```
#converting sqft basement into int
df.sqft_basement = df.sqft_basement.map(lambda x: int(x.replace('.0','')) if type(x) != 'int' else x)
ogdf.sqft_basement = ogdf.sqft_basement.map(lambda x: int(x.replace('.0','')) if type(x) != 'int' else x)
```

executed in 30ms, finished 09:21:37 2021-01-25

In [12]:
```
#making date datetime format
df['date'] = pd.to_datetime(df['date'])
```

executed in 31ms, finished 09:21:37 2021-01-25

In [13]:
```
#adding column for month
df['month'] = df.date.dt.month
```
executed in 6ms, finished 09:21:37 2021-01-25

In [14]:
```
df.isna().sum()
```
executed in 7ms, finished 09:21:37 2021-01-25

Out[14]:
```
id                0
date              0
price             0
bedrooms          0
bathrooms         0
sqft_living       0
sqft_lot          0
floors            0
waterfront     2376
view             63
condition         0
grade             0
sqft_basement     0
yr_built          0
yr_renovated   3842
zipcode           0
lat               0
long              0
sqft_living15     0
sqft_lot15        0
month             0
dtype: int64
```

In [15]:
```
#dropping na for view as only 63
df.dropna(subset=['view'],inplace=True)
ogdf.dropna(subset=['view'],inplace=True)
```
executed in 18ms, finished 09:21:38 2021-01-25

In [16]:
```
1 df.info()
```
executed in 10ms, finished 09:21:38 2021-01-25

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21534 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21534 non-null  int64
 1   date           21534 non-null  datetime64[ns]
 2   price          21534 non-null  float64
 3   bedrooms       21534 non-null  int64
 4   bathrooms      21534 non-null  float64
 5   sqft_living    21534 non-null  int64
 6   sqft_lot       21534 non-null  int64
 7   floors         21534 non-null  float64
 8   waterfront     19164 non-null  float64
 9   view           21534 non-null  float64
 10  condition      21534 non-null  int64
 11  grade          21534 non-null  int64
 12  sqft_basement  21534 non-null  int64
 13  yr_built       21534 non-null  int64
 14  yr_renovated   17704 non-null  float64
 15  zipcode        21534 non-null  int64
 16  lat            21534 non-null  float64
 17  long           21534 non-null  float64
 18  sqft_living15  21534 non-null  int64
 19  sqft_lot15     21534 non-null  int64
 20  month          21534 non-null  int64
dtypes: datetime64[ns](1), float64(8), int64(12)
memory usage: 3.6 MB
```

In [17]:
```
1 ogdf.fillna(0, inplace=True)
```
executed in 8ms, finished 09:21:38 2021-01-25

In [18]:
```
1 #getting probabilities for waterfront 1 and 0
2 total = 19164
3 no_water = len(df[df.waterfront == 0])
```
executed in 6ms, finished 09:21:38 2021-01-25

In [19]:
```
1 df.waterfront.value_counts(normalize=True)
```
executed in 6ms, finished 09:21:38 2021-01-25

Out[19]:
```
0.0    0.992434
1.0    0.007566
Name: waterfront, dtype: float64
```

In [20]:
```
1 waterfrontweights = list(df.waterfront.value_counts(normalize=True))
```
executed in 5ms, finished 09:21:38 2021-01-25

In [21]:
```
1 waterfrontweights
```
executed in 4ms, finished 09:21:38 2021-01-25

Out[21]:
```
[0.9924337299102484, 0.0075662700897516175]
```

In [22]:
```
1 a = [0,1]
```
executed in 3ms, finished 09:21:38 2021-01-25

In [23]:
```
1 #filling nan waterfront values with random probs given from our data
2 ogdf['waterfront'] = ogdf['waterfront'].apply(lambda x: np.random.choice(a,p=waterfrontweights) if np.isnan
3 df['waterfront'] = df['waterfront'].apply(lambda x: np.random.choice(a,p=waterfrontweights) if np.isnan(x)
```
executed in 96ms, finished 09:21:39 2021-01-25

In [24]:
```
1 df.info()
```
executed in 9ms, finished 09:21:39 2021-01-25

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21534 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21534 non-null  int64
 1   date           21534 non-null  datetime64[ns]
 2   price          21534 non-null  float64
 3   bedrooms       21534 non-null  int64
 4   bathrooms      21534 non-null  float64
 5   sqft_living    21534 non-null  int64
 6   sqft_lot       21534 non-null  int64
 7   floors         21534 non-null  float64
 8   waterfront     21534 non-null  float64
 9   view           21534 non-null  float64
 10  condition      21534 non-null  int64
 11  grade          21534 non-null  int64
 12  sqft_basement  21534 non-null  int64
 13  yr_built       21534 non-null  int64
 14  yr_renovated   17704 non-null  float64
 15  zipcode        21534 non-null  int64
 16  lat            21534 non-null  float64
 17  long           21534 non-null  float64
 18  sqft_living15  21534 non-null  int64
 19  sqft_lot15     21534 non-null  int64
 20  month          21534 non-null  int64
dtypes: datetime64[ns](1), float64(8), int64(12)
memory usage: 3.6 MB
```

In [25]:
```
1 #making column for age instead of yr built
2 df['age'] = 2020 - df['yr_built']
```
executed in 3ms, finished 09:21:39 2021-01-25

In [26]:
```
1 df.yr_renovated.value_counts()
```
executed in 9ms, finished 09:21:39 2021-01-25

Out[26]:
```
0.0       16961
2014.0       73
2003.0       31
2013.0       31
2007.0       30
            ...
1946.0        1
1959.0        1
1971.0        1
1951.0        1
1954.0        1
Name: yr_renovated, Length: 70, dtype: int64
```

In [27]:
```
1 #converting yr renovated into been renovated
2 df['been_renovated'] = df['yr_renovated'].apply(lambda x: 1 if x>0 else x)
```
executed in 8ms, finished 09:21:39 2021-01-25

In [28]:
```
1 #filling nan values with appropriate weights as given by the data
2 renovated_weights = list(df.been_renovated.value_counts(normalize=True))
3
4 df['been_renovated'] = df['been_renovated'].apply(lambda x: np.random.choice(a,p=renovated_weights) if np.i
```
executed in 107ms, finished 09:21:39 2021-01-25

In [29]:
```
1 #converting basement into binary variable
2 df['has_basement'] = df['sqft_basement'].map(lambda x: 1 if x >0 else x)
```
executed in 9ms, finished 09:21:39 2021-01-25

In [30]:
```
1 #creating column for bedrooms in to categorical grouping
2 #df['num_bedrooms'] = df['bedrooms'].map(lambda x: '5+' if x >= 5 else x)
```
executed in 3ms, finished 09:21:39 2021-01-25

In [31]:
```
1 yrrenovatedweights = list(df.yr_renovated.value_counts(normalize=True))
```
executed in 3ms, finished 09:21:39 2021-01-25

In [32]:
```
1 years_renindex = list(df.yr_renovated.value_counts().index)
```
executed in 5ms, finished 09:21:40 2021-01-25

In [33]:
```
#filling nan values for yr_renovated based on weights of given yrs renovated
df['yr_renovated'] = df['yr_renovated'].apply(lambda x: np.random.choice(years_renindex,p=yrrenovatedweight
```

executed in 115ms, finished 09:21:40 2021-01-25

In [34]:
```
df.info()
```

executed in 15ms, finished 09:21:40 2021-01-25

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21534 entries, 0 to 21596
Data columns (total 24 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   id              21534 non-null  int64  
 1   date            21534 non-null  datetime64[ns]
 2   price           21534 non-null  float64
 3   bedrooms        21534 non-null  int64  
 4   bathrooms       21534 non-null  float64
 5   sqft_living     21534 non-null  int64  
 6   sqft_lot        21534 non-null  int64  
 7   floors          21534 non-null  float64
 8   waterfront      21534 non-null  float64
 9   view            21534 non-null  float64
 10  condition       21534 non-null  int64  
 11  grade           21534 non-null  int64  
 12  sqft_basement   21534 non-null  int64  
 13  yr_built        21534 non-null  int64  
 14  yr_renovated    21534 non-null  float64
 15  zipcode         21534 non-null  int64  
 16  lat             21534 non-null  float64
 17  long            21534 non-null  float64
 18  sqft_living15   21534 non-null  int64  
 19  sqft_lot15      21534 non-null  int64  
 20  month           21534 non-null  int64  
 21  age             21534 non-null  int64  
 22  been_renovated  21534 non-null  float64
 23  has_basement    21534 non-null  int64  
dtypes: datetime64[ns](1), float64(9), int64(14)
memory usage: 4.1 MB
```

In [35]:
```
1 df.been_renovated.std()
```
executed in 6ms, finished 09:21:40 2021-01-25

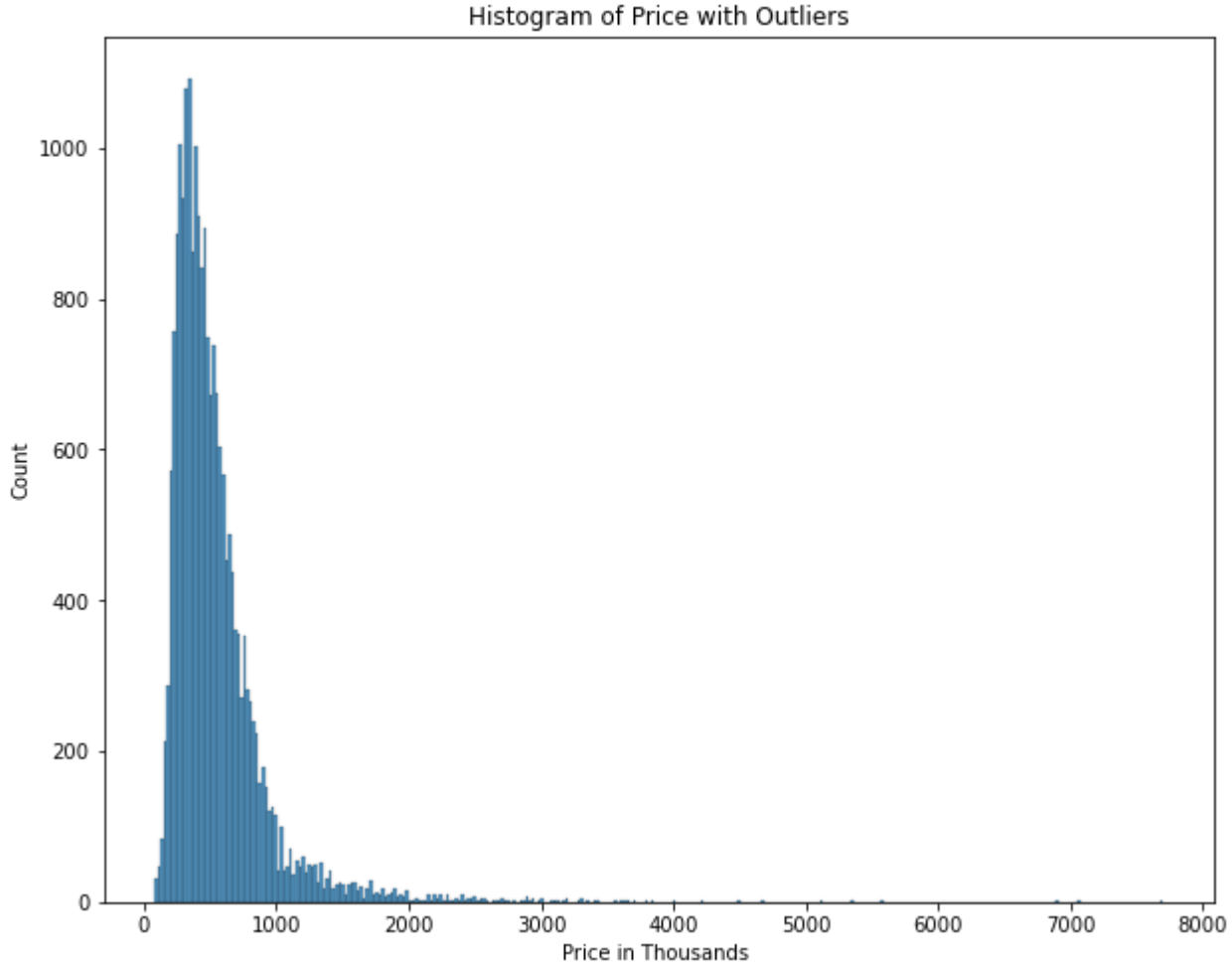
Out[35]: 0.20244679367597743

In [36]:
```
1 df.price.mean()
```
executed in 5ms, finished 09:21:40 2021-01-25

Out[36]: 540057.663833937

In [37]:
```
plt.figure(figsize=(10,8))
sns.histplot(x='price',data=df).set(xlabel='Price in Thousands ', ylabel='Count', title="Histogram of Price

plt.gca().xaxis.set_major_formatter(FuncFormatter(lambda x, _: int(round(x,0)/1000)))
```

executed in 528ms, finished 09:21:41 2021-01-25



In [38]:
```
1 df.info()
```
executed in 8ms, finished 09:21:41 2021-01-25

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21534 entries, 0 to 21596
Data columns (total 24 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   id              21534 non-null  int64
 1   date            21534 non-null  datetime64[ns]
 2   price           21534 non-null  float64
 3   bedrooms        21534 non-null  int64
 4   bathrooms       21534 non-null  float64
 5   sqft_living     21534 non-null  int64
 6   sqft_lot        21534 non-null  int64
 7   floors          21534 non-null  float64
 8   waterfront      21534 non-null  float64
 9   view            21534 non-null  float64
 10  condition       21534 non-null  int64
 11  grade           21534 non-null  int64
 12  sqft_basement   21534 non-null  int64
 13  yr_built        21534 non-null  int64
 14  yr_renovated    21534 non-null  float64
 15  zipcode         21534 non-null  int64
 16  lat             21534 non-null  float64
 17  long            21534 non-null  float64
 18  sqft_living15   21534 non-null  int64
 19  sqft_lot15      21534 non-null  int64
 20  month           21534 non-null  int64
 21  age             21534 non-null  int64
 22  been_renovated  21534 non-null  float64
 23  has_basement    21534 non-null  int64
dtypes: datetime64[ns](1), float64(9), int64(14)
memory usage: 4.1 MB
```

2 Multivariate Regression on Original Data (untransformed)

In [39]:
```
outcome = 'price'
x_cols = ogdf.drop(['price', 'id', 'date' ,], axis = 1).columns

x_cols
```

executed in 7ms, finished 09:21:41 2021-01-25

Out[39]:
```
Index(['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'long',
       'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

In [40]:
```
predictors = '+'.join(x_cols)

predictors

f = outcome + '~' + predictors

f

ogmodel = ols(formula= f, data= ogdf).fit()
ogmodel.summary()

```

executed in 78ms, finished 09:21:41 2021-01-25

Out[40]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.700
Model:	OLS	**Adj. R-squared:**	0.700
Method:	Least Squares	**F-statistic:**	2791.
Date:	Mon, 25 Jan 2021	**Prob (F-statistic):**	0.00
Time:	09:21:41	**Log-Likelihood:**	-2.9345e+05
No. Observations:	21534	**AIC:**	5.869e+05
Df Residuals:	21515	**BIC:**	5.871e+05
Df Model:	18		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	6.895e+06	2.92e+06	2.358	0.018	1.16e+06	1.26e+07
bedrooms	-3.574e+04	1896.198	-18.848	0.000	-3.95e+04	-3.2e+04
bathrooms	4.106e+04	3255.787	12.611	0.000	3.47e+04	4.74e+04
sqft_living	102.4556	18.012	5.688	0.000	67.151	137.761
sqft_lot	0.1274	0.048	2.665	0.008	0.034	0.221
floors	7068.5341	3591.513	1.968	0.049	28.903	1.41e+04

waterfront	5.955e+05	1.81e+04	32.816	0.000	5.6e+05	6.31e+05
view	5.48e+04	2118.499	25.868	0.000	5.06e+04	5.9e+04
condition	2.678e+04	2340.769	11.441	0.000	2.22e+04	3.14e+04
grade	9.683e+04	2155.795	44.914	0.000	9.26e+04	1.01e+05
sqft_above	77.9370	18.003	4.329	0.000	42.650	113.224
sqft_basement	46.4906	17.843	2.605	0.009	11.516	81.465
yr_built	-2624.5562	71.788	-36.560	0.000	-2765.266	-2483.846
yr_renovated	24.0595	3.966	6.067	0.000	16.287	31.832
zipcode	-588.0529	32.922	-17.862	0.000	-652.582	-523.524
lat	6.02e+05	1.07e+04	56.177	0.000	5.81e+05	6.23e+05
long	-2.179e+05	1.31e+04	-16.602	0.000	-2.44e+05	-1.92e+05
sqft_living15	21.3183	3.442	6.193	0.000	14.571	28.066
sqft_lot15	-0.3889	0.073	-5.318	0.000	-0.532	-0.246

Omnibus:	18187.772	**Durbin-Watson:**	1.989
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1798330.249
Skew:	3.533	**Prob(JB):**	0.00
Kurtosis:	47.208	**Cond. No.**	2.15e+08

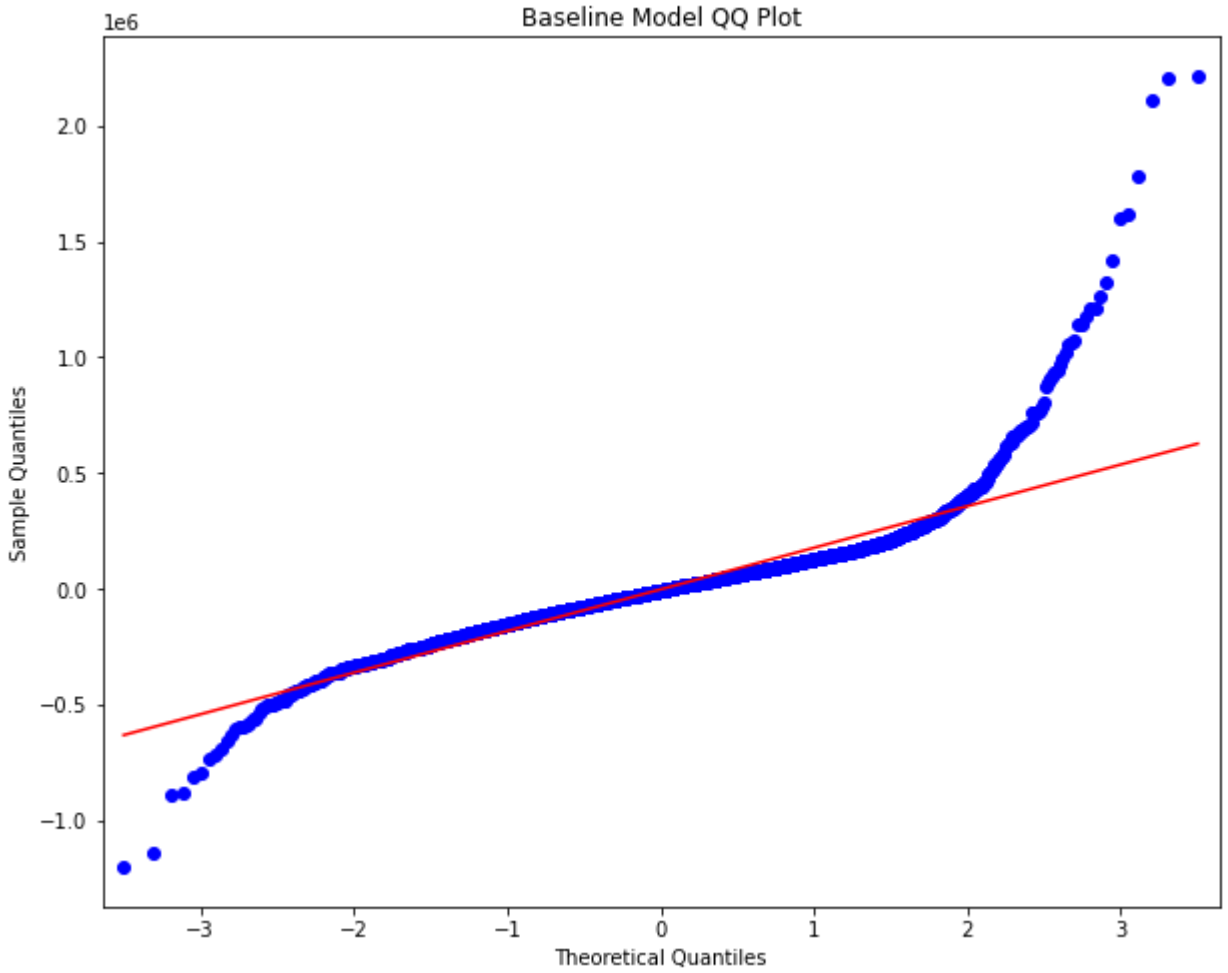
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.15e+08. This might indicate that there are
strong multicollinearity or other numerical problems.

In [41]:

```
plt.figure(figsize=(10,8))
ax = plt.subplot(111)

data = ogdf.copy()

y = data['price']
X = data.drop(['price', 'id', 'date' ,], axis = 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

len(X_test)

linreg = LinearRegression()
linreg.fit(X_train, y_train)

y_hat_train = linreg.predict(X_train)
y_hat_test = linreg.predict(X_test)


mse_train = mean_squared_error(y_train, y_hat_train)
mse_test = mean_squared_error(y_test, y_hat_test)

print('Train MSE:', mse_train)
print('Test MSE:', mse_test)

print('RMSE Train:', np.sqrt(mse_train))
print('RMSE Test:', np.sqrt(mse_test))

r2_score(y_test, y_hat_test)

residuals = (y_test - y_hat_test)

statsmodels.graphics.gofplots.qqplot(residuals, line = "r",ax=ax)
plt.title("Baseline Model QQ Plot")

plt.savefig('images/baselinemodelqqplot1')
```

executed in 305ms, finished 09:21:42 2021-01-25

```
Train MSE: 40700597544.97559
Test MSE: 38221888636.3495
RMSE Train: 201743.89097312363
RMSE Test: 195504.19084088583
```



As we can see above, in our original model our adjusted R^2 is .70 and all independent variables have statistically significant p-values at the .05 level. Our RMSE is 198k with a mean of 540k i.e. Our model explains 63% of the change in Y using RMSE as our metric.

In [42]:
```
1- (205806/ogdf.price.mean())
```
executed in 5ms, finished 09:21:42 2021-01-25

Out[42]:
```
0.6189184715221752
```

3 Final Model

In [43]:
```
#removing outliers that are 2 std away for price
df1 = df[(np.abs(stats.zscore(df['price'])) < 2)]
```
executed in 6ms, finished 09:21:42 2021-01-25

In [44]:
```
df2 = df1[(np.abs(stats.zscore(df1['bedrooms'])) < 4)]
```
executed in 9ms, finished 09:21:42 2021-01-25

In [45]:
```
df3 = df2[(np.abs(stats.zscore(df2['bathrooms'])) < 4)]
```
executed in 6ms, finished 09:21:43 2021-01-25

In [46]:
```
df4 = df3[(np.abs(stats.zscore(df3['sqft_lot15'])) < 2)]
```
executed in 6ms, finished 09:21:43 2021-01-25

In [47]:
```
fin = df4[(np.abs(stats.zscore(df4['sqft_lot'])) < 2)]
```
executed in 6ms, finished 09:21:43 2021-01-25

Above getting zscore outliers dropped for continuous variables. This resulted in 1673 rows being dropped

In [48]:
```
len(df)-len(fin)
```
executed in 4ms, finished 09:21:44 2021-01-25

Out[48]:
```
1673
```

In [49]:
```
1 fin.info()
```
executed in 10ms, finished 09:21:44 2021-01-25

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 19861 entries, 0 to 21596
Data columns (total 24 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   id              19861 non-null  int64
 1   date            19861 non-null  datetime64[ns]
 2   price           19861 non-null  float64
 3   bedrooms        19861 non-null  int64
 4   bathrooms       19861 non-null  float64
 5   sqft_living     19861 non-null  int64
 6   sqft_lot        19861 non-null  int64
 7   floors          19861 non-null  float64
 8   waterfront      19861 non-null  float64
 9   view            19861 non-null  float64
 10  condition       19861 non-null  int64
 11  grade           19861 non-null  int64
 12  sqft_basement   19861 non-null  int64
 13  yr_built        19861 non-null  int64
 14  yr_renovated    19861 non-null  float64
 15  zipcode         19861 non-null  int64
 16  lat             19861 non-null  float64
 17  long            19861 non-null  float64
 18  sqft_living15   19861 non-null  int64
 19  sqft_lot15      19861 non-null  int64
 20  month           19861 non-null  int64
 21  age             19861 non-null  int64
 22  been_renovated  19861 non-null  float64
 23  has_basement    19861 non-null  int64
dtypes: datetime64[ns](1), float64(9), int64(14)
memory usage: 3.8 MB
```

In [50]:
```
1 fin.describe()
```
executed in 64ms, finished 09:21:45 2021-01-25

Out[50]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	cor
count	1.986100e+04	1.986100e+04	19861.000000	19861.000000	19861.000000	19861.000000	19861.000000	19861.000000	19861.000000	1986
mean	4.686300e+09	4.838098e+05	3.327929	2.051269	1970.817935	9144.128946	1.479936	0.003524	0.177735	
std	2.873980e+09	2.235226e+05	0.861893	0.704757	773.099518	8008.091488	0.539514	0.059264	0.654006	
min	1.000102e+06	7.800000e+04	1.000000	0.500000	370.000000	520.000000	1.000000	0.000000	0.000000	
25%	2.215450e+09	3.150000e+05	3.000000	1.500000	1400.000000	5000.000000	1.000000	0.000000	0.000000	
50%	4.036400e+09	4.370000e+05	3.000000	2.000000	1850.000000	7374.000000	1.000000	0.000000	0.000000	
75%	7.452500e+09	6.070000e+05	4.000000	2.500000	2430.000000	9933.000000	2.000000	0.000000	0.000000	
max	9.900000e+09	1.270000e+06	6.000000	4.750000	7350.000000	57000.000000	3.500000	1.000000	4.000000	

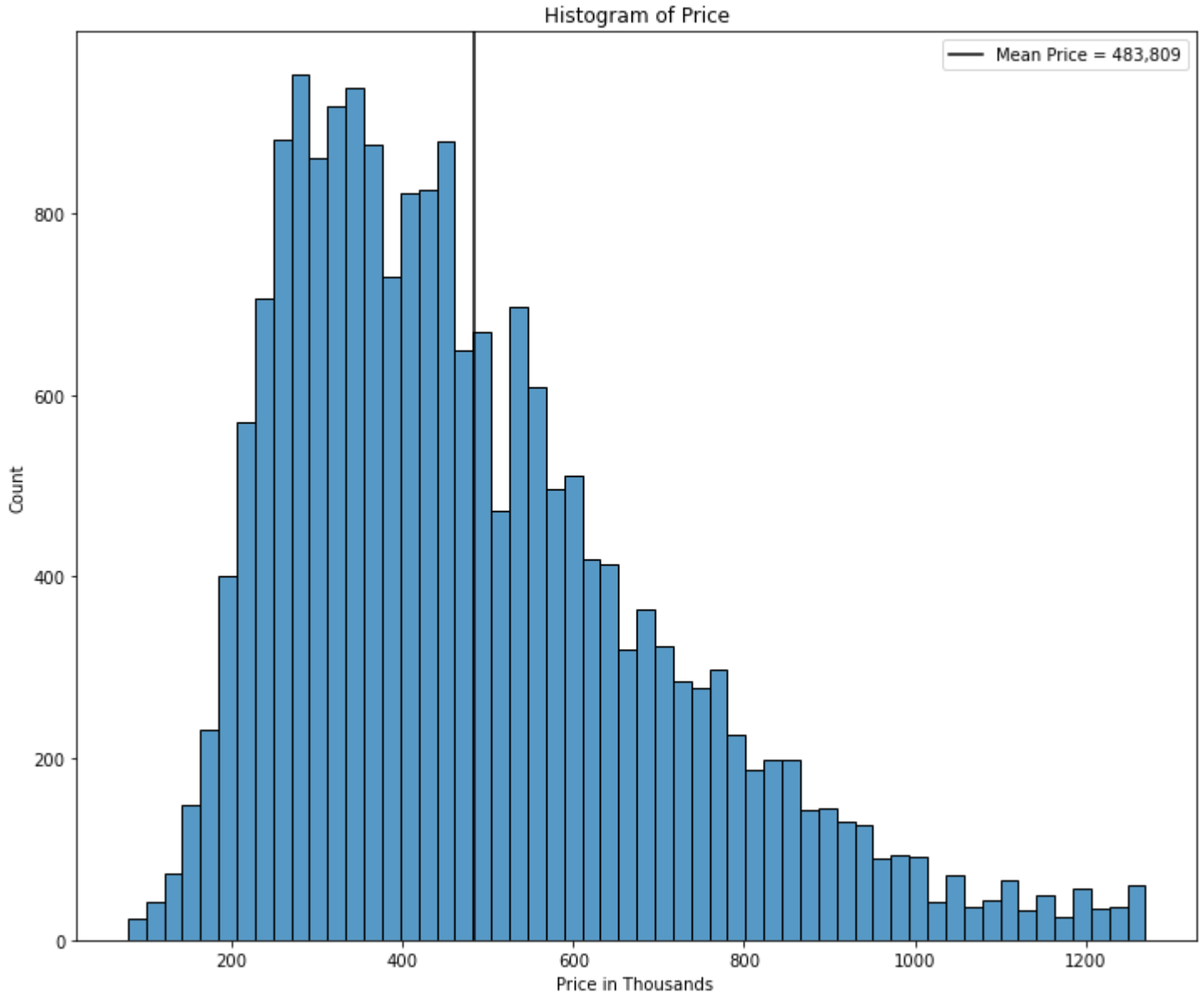
In [51]:
```
1 fin.price.mean()
```
executed in 5ms, finished 09:21:45 2021-01-25

Out[51]: 483809.7648154675

In [52]:

```
plt.figure(figsize=(12,10))
sns.histplot(x='price',data=fin,bins='auto' ).set(xlabel='Price in Thousands ', ylabel='Count', title="Hist
plt.gca().xaxis.set_major_formatter(FuncFormatter(lambda x, _: int(round(x,0)/1000)))
plt.axvline(fin.price.mean(),c='black',label='Mean Price = 483,809')
plt.legend()
plt.savefig('images/pricehistogram')
```

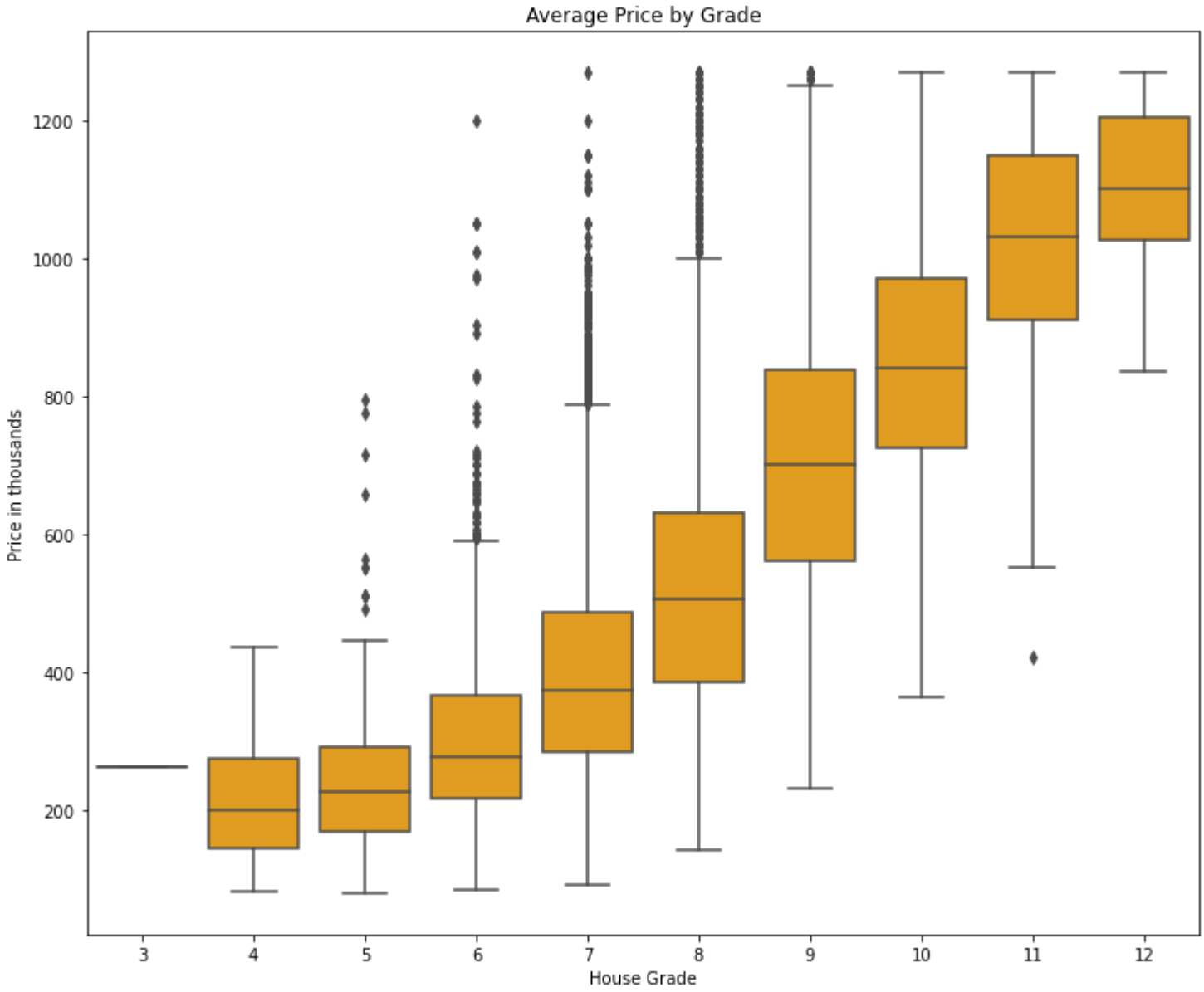
executed in 316ms, finished 09:21:47 2021-01-25


Histogram of Price
Mean Price = 483,809
Count
0
200
400
600
800
200
400
600
800
1000
1200
Price in Thousands

In [53]:

```
plt.figure(figsize=(12,10))
sns.boxplot(x='grade', y='price', data=fin,color='orange',).set(xlabel='House Grade',
                                                ylabel='Price in thousands', title="Average
plt.gca().yaxis.set_major_formatter(FuncFormatter(lambda x, _: int(round(x,0)/1000)))
plt.savefig('images/pricevgrad2')
```

executed in 290ms, finished 09:21:47 2021-01-25


Average Price by Grade
Price in thousands
1200
1000
800
600
400
200
3
4
5
6
7
8
9
10
11
12
House Grade

In [54]:

```
#convert bedrooms over 5 as 5+
fin['num_bedrooms'] = fin['bedrooms'].apply(lambda x: '5+' if x >= 5 else x)
```

executed in 7ms, finished 09:21:48 2021-01-25

```
<ipython-input-54-e6e2aeb4d901>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  fin['num_bedrooms'] = fin['bedrooms'].apply(lambda x: '5+' if x >= 5 else x)
```

In [55]:

```
fin.loc[(fin.lat > 47.5) & (fin.long > -122.05), "district"] = 'district_1'
fin.loc[(fin.lat > 47.5) & (fin.long > -122.15) & (fin.long < -122.05), "district"] = 'district_2'
fin.loc[(fin.lat > 47.5) & (fin.long > -122.259231) & (fin.long < -122.15), "district"] = 'district_3'
fin.loc[(fin.lat > 47.5) & (fin.long < -122.259231), "district"] = 'district_4'
fin.loc[(fin.lat < 47.5) & (fin.long > -122.05), "district"] = 'district_8'
fin.loc[(fin.lat < 47.5) & (fin.long > -122.15) & (fin.long < -122.05), "district"] = 'district_7'
fin.loc[(fin.lat < 47.5) & (fin.long > -122.259231) & (fin.long < -122.15), "district"] = 'district_6'
fin.loc[(fin.lat < 47.5) & (fin.long < -122.259231), "district"] = 'district_5'
```

executed in 23ms, finished 09:21:49 2021-01-25

```
C:\Users\sergi\anaconda3\envs\learn-env\lib\site-packages\pandas\core\indexing.py:1596: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy)
  self.obj[key] = _infer_fill_value(value)
C:\Users\sergi\anaconda3\envs\learn-env\lib\site-packages\pandas\core\indexing.py:1765: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy)
  isetter(loc, value)
C:\Users\sergi\anaconda3\envs\learn-env\lib\site-packages\pandas\core\indexing.py:1765: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy)
  isetter(loc, value)
C:\Users\sergi\anaconda3\envs\learn-env\lib\site-packages\pandas\core\indexing.py:1765: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
```

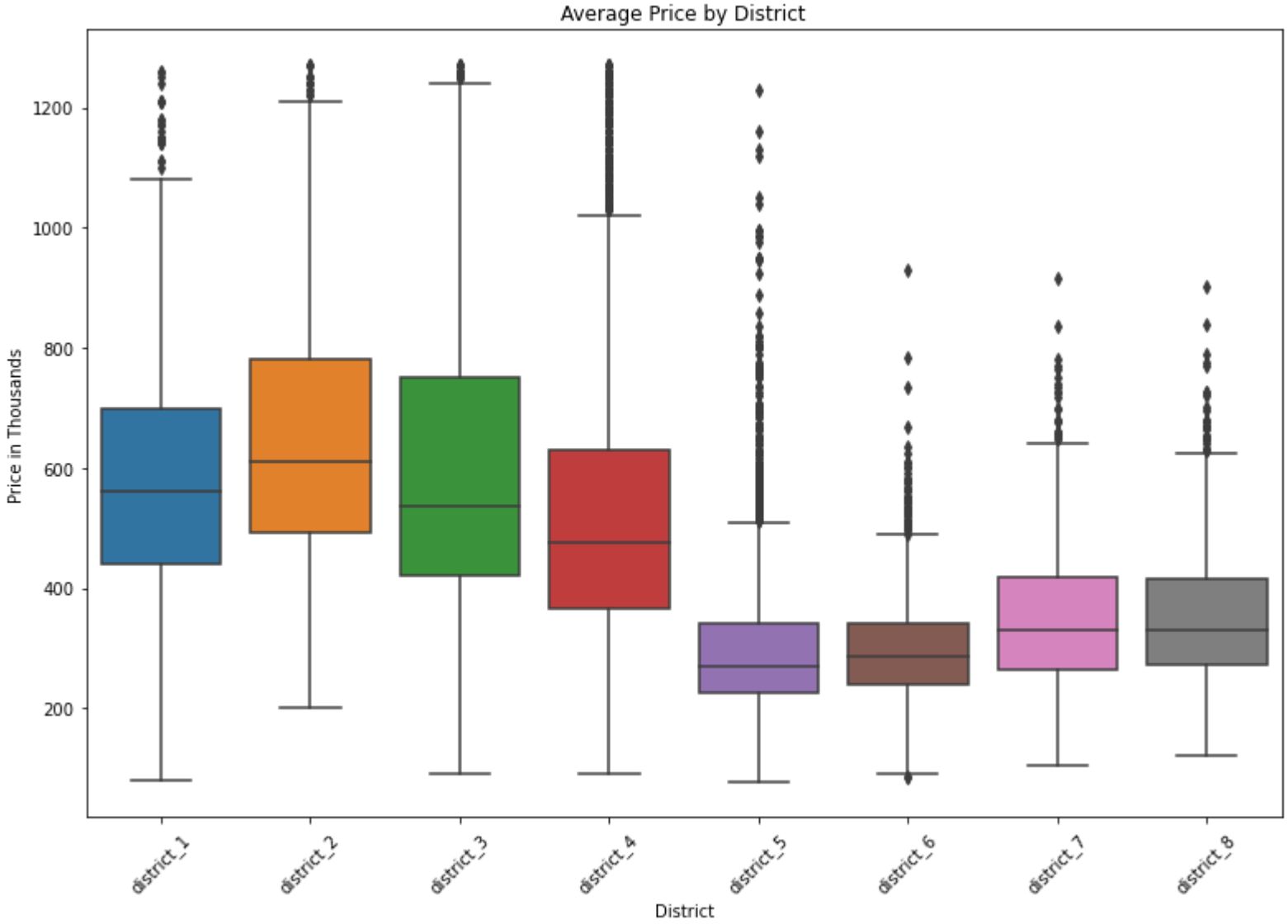
```
returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy)
  isetter(loc, value)
C:\Users\sergi\anaconda3\envs\learn-env\lib\site-packages\pandas\core\indexing.py:1765: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy)
  isetter(loc, value)
C:\Users\sergi\anaconda3\envs\learn-env\lib\site-packages\pandas\core\indexing.py:1765: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy)
  isetter(loc, value)
C:\Users\sergi\anaconda3\envs\learn-env\lib\site-packages\pandas\core\indexing.py:1765: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy)
  isetter(loc, value)
C:\Users\sergi\anaconda3\envs\learn-env\lib\site-packages\pandas\core\indexing.py:1765: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy)
  isetter(loc, value)
C:\Users\sergi\anaconda3\envs\learn-env\lib\site-packages\pandas\core\indexing.py:1765: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  isetter(loc, value)
```

In [56]:

```
plt.figure(figsize=(11,8))
sns.boxplot(x='district',y='price',data=fin.sort_values('district')).set(ylabel='Price in Thousands',
                                                                         xlabel="District", title='Average
plt.gca().yaxis.set_major_formatter(FuncFormatter(lambda x, _: int(round(x,0)/1000)))
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig('images/districtboxplot1')
```

executed in 298ms, finished 09:21:50 2021-01-25



In [57]:

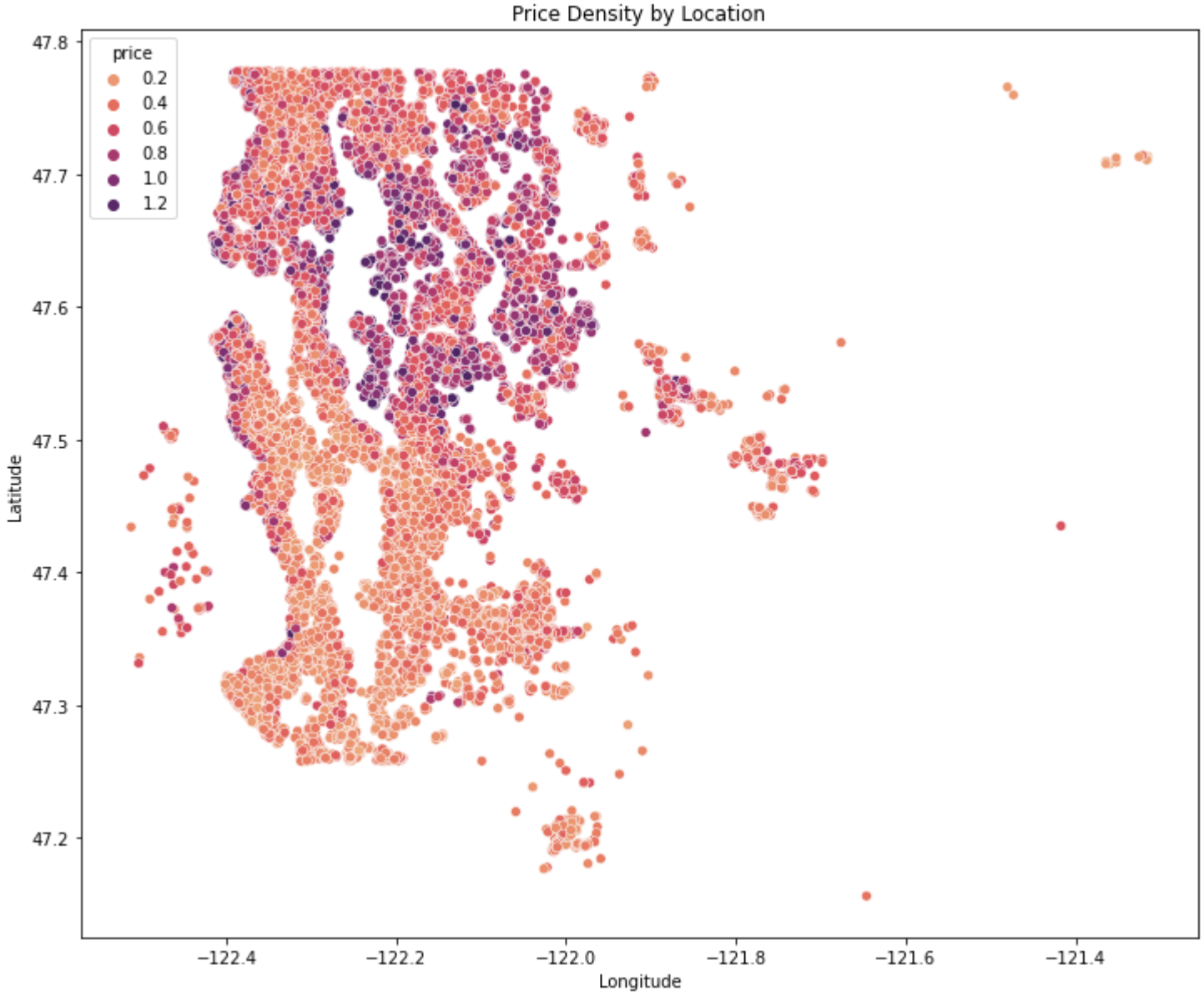
```
1 view = pd.get_dummies(fin['view'], prefix='v', drop_first=True)
2 view.columns = view.columns.map(lambda x: x.replace('.0',''))
3 condition = pd.get_dummies(fin['condition'], prefix='cond', drop_first=True)
4 district = pd.get_dummies(fin['district'],drop_first=True)
```

executed in 9ms, finished 09:21:50 2021-01-25

In [58]:
```
1 final = pd.concat([fin,view,condition,district],axis=1)
```
executed in 8ms, finished 09:21:51 2021-01-25

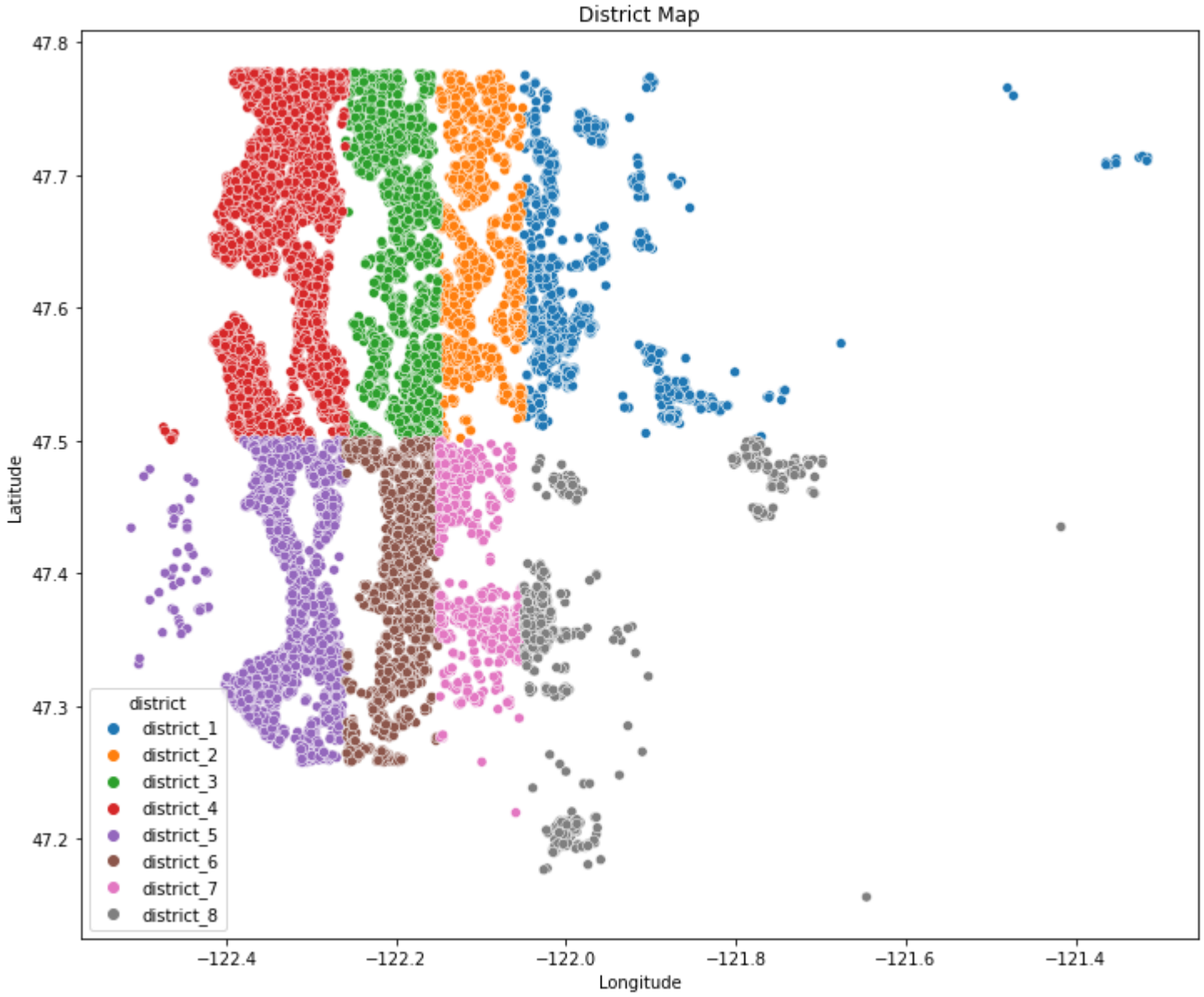
In [59]:
```
plt.figure(figsize=(12, 10))
sns.scatterplot(x = "long", y = "lat", hue = 'price', data = fin,palette='flare'
                ).set(xlabel='Longitude',ylabel='Latitude', title='Price Density by Location')
plt.savefig('images/pricedensity')
```

executed in 1.41s, finished 09:21:53 2021-01-25



In [60]:
```
plt.figure(figsize=(12, 10))
sns.scatterplot(x = "long", y = "lat", hue = 'district', data = fin.sort_values('district'),palette='tab10'
plt.savefig('images/districtmap2')
```

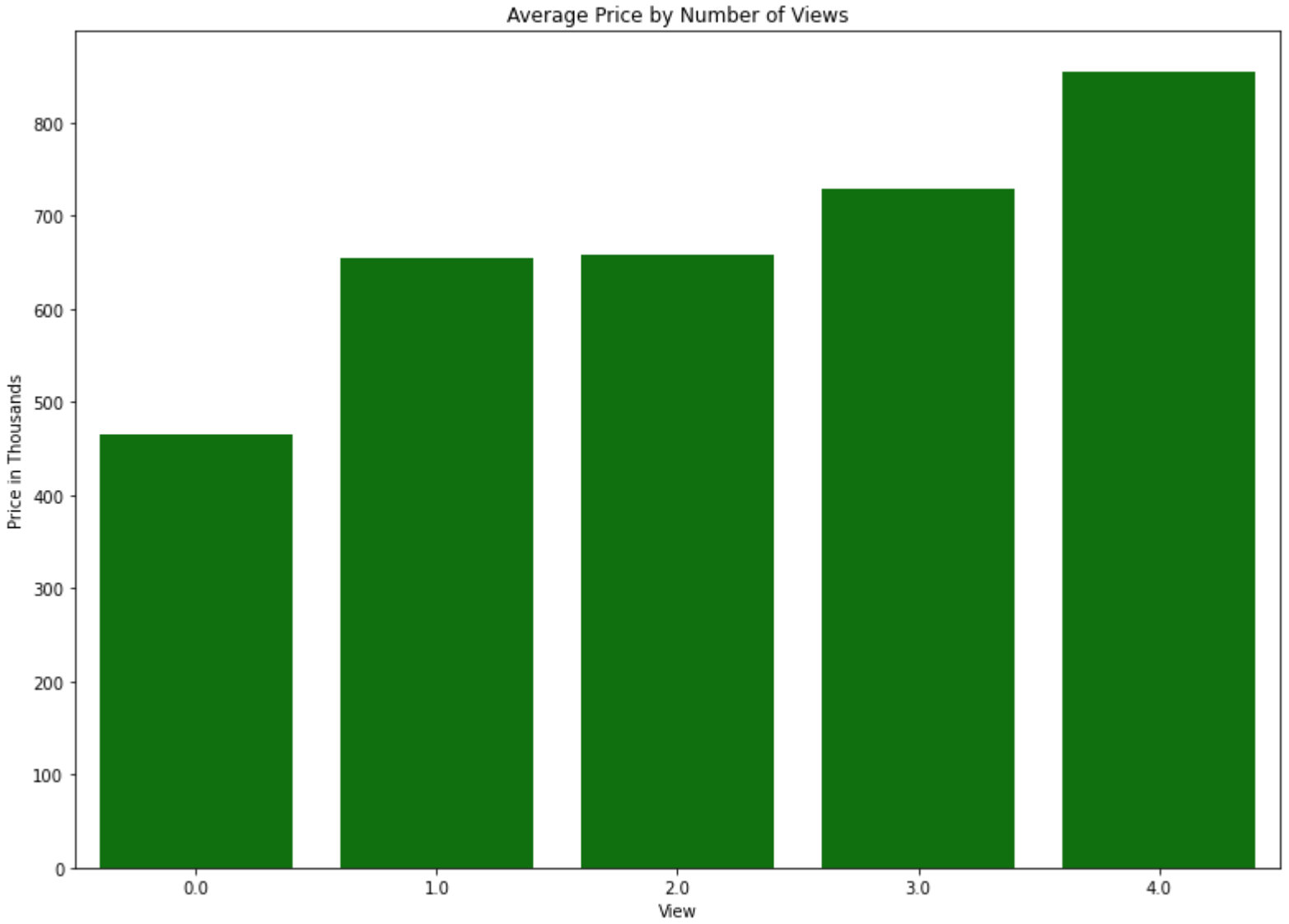
executed in 1.58s, finished 09:21:55 2021-01-25



In [61]:

```
plt.figure(figsize=(11,8))
sns.barplot(x='view',y='price',data=fin.sort_values('view'),color='green',ci=None).set(ylabel='Price in Tho
                                                                                      xlabel="View", title='Average Pric
plt.gca().yaxis.set_major_formatter(FuncFormatter(lambda x, _: int(round(x,0)/1000)))

plt.tight_layout()
plt.savefig('images/viewbarplot1')
```

executed in 198ms, finished 09:21:56 2021-01-25


Average Price by Number of Views
Price in Thousands
0
100
200
300
400
500
600
700
800
0.0
1.0
2.0
3.0
4.0
View

In [62]:

```
outcome = 'price'
x_cols = final.drop(['price', 'id', 'date','month','yr_built','has_basement',
                     'yr_renovated','view','condition','zipcode','lat','long','cond_2','district','sqft_l

x_cols

```

executed in 13ms, finished 09:21:56 2021-01-25

Out[62]:

```
Index(['bedrooms', 'bathrooms', 'sqft_living', 'floors', 'waterfront', 'grade',
       'sqft_basement', 'sqft_living15', 'sqft_lot15', 'age', 'been_renovated',
       'v_1', 'v_2', 'v_3', 'v_4', 'cond_3', 'cond_4', 'cond_5', 'district_2',
       'district_3', 'district_4', 'district_5', 'district_6', 'district_7',
       'district_8'],
      dtype='object')
```

In [63]:
```
predictors = '+'.join(x_cols)

predictors

f = outcome + '~' + predictors

f

finalmodel = ols(formula= f, data= final).fit()
finalmodel.summary()
```

executed in 85ms, finished 09:21:56 2021-01-25

Out[63]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.733
Model:	OLS	**Adj. R-squared:**	0.733
Method:	Least Squares	**F-statistic:**	2179.
Date:	Mon, 25 Jan 2021	**Prob (F-statistic):**	0.00
Time:	09:21:56	**Log-Likelihood:**	-2.5970e+05
No. Observations:	19861	**AIC:**	5.194e+05
Df Residuals:	19835	**BIC:**	5.197e+05
Df Model:	25		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-4.583e+05	1.31e+04	-35.013	0.000	-4.84e+05	-4.33e+05
bedrooms	-9910.9330	1258.220	-7.877	0.000	-1.24e+04	-7444.716
bathrooms	2.228e+04	2040.857	10.918	0.000	1.83e+04	2.63e+04
sqft_living	89.3817	2.491	35.875	0.000	84.498	94.265
floors	1.951e+04	2302.234	8.473	0.000	1.5e+04	2.4e+04
waterfront	1.348e+05	1.53e+04	8.796	0.000	1.05e+05	1.65e+05
grade	7.111e+04	1357.340	52.393	0.000	6.85e+04	7.38e+04

sqft_basement	-13.7752	2.844	-4.843	0.000	-19.350	-8.200
sqft_living15	49.7337	2.312	21.511	0.000	45.202	54.265
sqft_lot15	-0.5580	0.130	-4.290	0.000	-0.813	-0.303
age	1664.6664	45.452	36.625	0.000	1575.577	1753.756
been_renovated	2.683e+04	4406.405	6.088	0.000	1.82e+04	3.55e+04
v_1	6.25e+04	7061.223	8.851	0.000	4.87e+04	7.63e+04
v_2	5.211e+04	4339.361	12.008	0.000	4.36e+04	6.06e+04
v_3	8.2e+04	6457.872	12.698	0.000	6.93e+04	9.47e+04
v_4	1.725e+05	1.01e+04	17.050	0.000	1.53e+05	1.92e+05
cond_3	2.389e+04	8920.958	2.678	0.007	6402.741	4.14e+04
cond_4	5.147e+04	8950.539	5.750	0.000	3.39e+04	6.9e+04
cond_5	7.988e+04	9304.732	8.585	0.000	6.16e+04	9.81e+04
district_2	3.556e+04	3858.820	9.216	0.000	2.8e+04	4.31e+04
district_3	4.72e+04	3614.589	13.058	0.000	4.01e+04	5.43e+04
district_4	3.556e+04	3575.764	9.944	0.000	2.85e+04	4.26e+04
district_5	-1.624e+05	3830.851	-42.395	0.000	-1.7e+05	-1.55e+05
district_6	-1.587e+05	3845.597	-41.272	0.000	-1.66e+05	-1.51e+05
district_7	-1.443e+05	4693.350	-30.747	0.000	-1.54e+05	-1.35e+05
district_8	-1.229e+05	4764.335	-25.798	0.000	-1.32e+05	-1.14e+05

Omnibus:	2445.355	**Durbin-Watson:**	1.979
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	5841.892
Skew:	0.724	**Prob(JB):**	0.00
Kurtosis:	5.228	**Cond. No.**	2.60e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.6e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

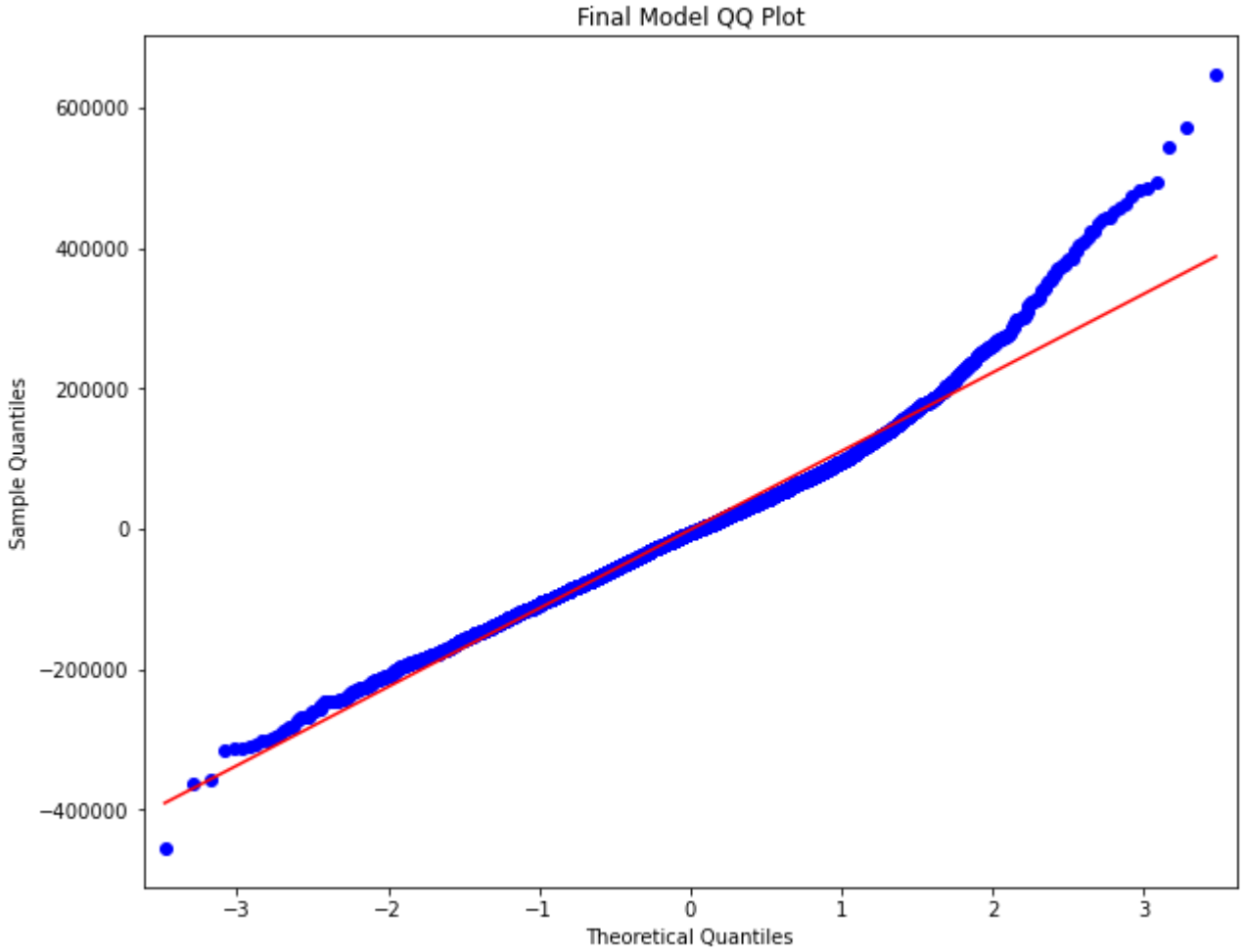
In [64]:

```
plt.figure(figsize=(10,8))
ax = plt.subplot(111)

data = final.copy()

y = data['price']
X = data.drop(['price', 'id', 'date','month','yr_built','has_basement',
                        'yr_renovated','view','condition','zipcode','lat','long','cond_2','district','sqft_l

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

len(X_test)

linreg = LinearRegression()
linreg.fit(X_train, y_train)

y_hat_train = linreg.predict(X_train)
y_hat_test = linreg.predict(X_test)


mse_train = mean_squared_error(y_train, y_hat_train)
mse_test = mean_squared_error(y_test, y_hat_test)
R2 = r2_score(y_test, y_hat_test)

print('Train MSE:', mse_train)
print('Test MSE:', mse_test)

print('RMSE Train:', np.sqrt(mse_train))
print('RMSE Test:', np.sqrt(mse_test))
print('R2 score:', round(R2,4))

residuals = (y_test - y_hat_test)

statsmodels.graphics.gofplots.qqplot(residuals, line = "r",ax=ax)
plt.title('Final Model QQ Plot')
plt.savefig('images/finalqqplot1')
```

executed in 248ms, finished 09:21:58 2021-01-25

```
Train MSE: 13476861461.167768
Test MSE: 12791964013.952364
RMSE Train: 116089.88526640797
```

```
RMSE Test: 113101.56503759073
R2 score: 0.7317
```



In [65]:
```
1 - (np.sqrt(mse_test) / final.price.mean())
```

executed in 6ms, finished 09:22:00 2021-01-25

Out[65]:
```
0.7662271965909382
```

Final result of the model has an R2 of .733 i.e. our model explains 73% of the variation in Y (prices). Given the dummy variables dropped, our reference model is a house with 0 views, rated 1 on condition (cond 2 was dropped due to P-value not being stat sig), and in district 1.

Looking at the final model summary, we were able to improve upon our original R2 of .700 to .733 and our RMSE from .62 to .76. While

this improvement is noticable, more can certainly be done to improve R2. Interestingly though, feature transformation such as log transformation and mean normalization not only did not improve our R2 but was detrimental to our overall model accuracy.

Separately, looking at the model coefficients, grade surprisingly has the largest impact on home price for any independent variable as with an average grade of 7 and a coefficient of 71k on average attributes to 537k of home price.

Sqft living peformed as expected also contributing a noticable positive influence on home price with the average home sqft being 1970 with a coefficient of 89, averaging a contribution of 175k to home price or in percentage terms 36%

As expected, waterfront properties increase the value of a home by 130k, a noticable amount, but to note only 68 homes in our data have waterfront properties so although the impact is positive more than 90% of the homes will not have a waterfront property (at least based on the data used here)

Location is by far one of the most interesting variables investigated, and could still be investigated further. It is a bit blunt to group all of the homes into 8 districts, but as we saw with zipcode dummy variables, having the 77 dummies increased our R2 from .7 to .84 more than any variable or feature transformation. This isn't all that surprising given real estate is largely determined by neighborhood or location location location as they say, but with the appropriate amount of time and data, it would be interesting to investigate further.

From a coefficient perspective, the range in difference in home price is -160k to +36k which rounding up here is a 200k difference in home price solely on location.

Our QQ plot for the final model does indicate that there are some issues with outliers on the right hand side, but relative to the baseline model there is still a noticeable improvement. I presume dropping our outliers had a large part in this effect

In [66]:
```
1 len(final[final.waterfront == 1])
```
executed in 6ms, finished 09:22:04 2021-01-25

Out[66]: 70

In [67]:
```
1 final.sqft_living.mean()
```
executed in 4ms, finished 09:22:04 2021-01-25

Out[67]: 1970.8179346457882

In [68]:
```
1 89.22*final.sqft_living.mean() / final.price.mean()
```
executed in 4ms, finished 09:22:05 2021-01-25

Out[68]:
```
0.3634411475679163
```

In [69]:
```
1 final.price.mean()
```
executed in 6ms, finished 09:22:05 2021-01-25

Out[69]:
```
483809.7648154675
```

In [70]:
```
1 ols(formula= f, data= final).fit().params
```
executed in 69ms, finished 09:22:06 2021-01-25

Out[70]:
```
Intercept        -458347.986554
bedrooms           -9910.933006
bathrooms          22282.795179
sqft_living           89.381662
floors             19506.499666
waterfront        134837.094863
grade              71114.589759
sqft_basement        -13.775201
sqft_living15         49.733678
sqft_lot15            -0.558011
age                 1664.666363
been_renovated     26827.959824
v_1                62498.117996
v_2                52106.771405
v_3                82004.315241
v_4               172473.367396
cond_3             23888.563363
cond_4             51469.370564
cond_5             79878.828746
district_2         35563.270379
district_3         47200.645674
district_4         35557.744267
district_5       -162410.024191
district_6       -158716.337979
district_7       -144306.821809
district_8       -122908.728897
dtype: float64
```

In [71]:
```
1 final.sqft_living.mean()*89
```
executed in 5ms, finished 09:22:06 2021-01-25

Out[71]: 175402.79618347515

In [72]:
```
1 final.sqft_living.mean()*89
```
executed in 6ms, finished 09:22:07 2021-01-25

Out[72]: 175402.79618347515

In [73]:
```
1 final.grade.mean()*71247
```
executed in 4ms, finished 09:22:07 2021-01-25

Out[73]: 537374.7847540407

In [74]:
```
1 len(final[final.waterfront == 1])
```
executed in 4ms, finished 09:22:07 2021-01-25

Out[74]: 70

In [75]:
```
1 len(final[(final.v_4 ==1)& (final.waterfront==1) ])
```
executed in 7ms, finished 09:22:08 2021-01-25

Out[75]: 45