Final Project Submission

Please fill out:

- Student name: Dennis Mwai Kimiri
- Student pace: full time
- Scheduled project review date/time: 30/09/2022
- Instructor name:
- Blog post URL:

Introduction

A real estate agency has given us a task to analyze house prices based on various features of the house. They want to know what kind of advice to give to homeowners when they want to buy or sell homes. They also need to know what features of a house a homeowner should focus on so as to increase the selling price of their homes.The aim of this project is to analyze house sales in King County using linear regression in order to offer impactful insights on whether renovations affect house prices and by how much.

Exploration Data Analysis

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

Exploring House Data.

In [2]:
```
#Reading the dataset
df = pd.read_csv(r'C:\Users\Asus\Documents\p2project\dsc-phase-2-project-v2-3\data\k
```

In [3]:
```
#first five rows
df.head()
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NaN
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	NO
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	NO
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	NO
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	NO

5 rows × 21 columns

In [4]:
```
#last five columns
df.tail()
```

Out[4]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	water
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	3.0	
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	2.0	
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	2.0	
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	2.0	
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	1076	2.0	

5 rows × 21 columns

In [5]:
```
#summary of data to check for null values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 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  object 
 9   view           21534 non-null  object 
 10  condition      21597 non-null  object 
 11  grade          21597 non-null  object 
 12  sqft_above     21597 non-null  int64  
 13  sqft_basement  21597 non-null  object 
 14  yr_built       21597 non-null  int64  
 15  yr_renovated   17755 non-null  float64
 16  zipcode        21597 non-null  int64  
 17  lat            21597 non-null  float64
 18  long           21597 non-null  float64
 19  sqft_living15  21597 non-null  int64  
 20  sqft_lot15     21597 non-null  int64  
dtypes: float64(6), int64(9), object(6)
memory usage: 3.5+ MB
```

Columns without 21597 values have null values

In [6]:
```
df.shape
```

Out[6]:
```
(21597, 21)
```

In [7]:
```
#Description of dataset
df.describe()
```

Out[7]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	
count	2.159700e+04	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	21597.(
mean	4.580474e+09	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+04	1.4
std	2.876736e+09	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+04	0.5
min	1.000102e+06	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	1.(
25%	2.123049e+09	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03	1.(
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.5
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+04	2.(
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.5

Cleaning House Data.

In [8]:
```
#number of rows with null values
df.isnull().sum()
```

Out[8]:
```
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_above        0
sqft_basement     0
yr_built          0
yr_renovated   3842
zipcode           0
lat               0
long              0
sqft_living15     0
sqft_lot15        0
dtype: int64
```

In [9]:
```
#previewing waterfront
df['waterfront'].head()
```

Out[9]:
```
0    NaN
1     NO
2     NO
3     NO
4     NO
Name: waterfront, dtype: object
```

In [10]:
```
#percentage of missing values in waterfront column
(df['waterfront'].isna().sum()/len(df['waterfront']))*100
```

Out[10]:
```
11.00152798999861
```

In [11]:
```
#checking categories
df['waterfront'].value_counts()
```

Out[11]:
```
NO     19075
YES      146
Name: waterfront, dtype: int64
```

Having a waterfront is a very infrequent case thus we shall replace the missing values with NO

In [12]:
```
df.fillna('NO', inplace=True)
```

In [13]:
```
df['waterfront'].isna().sum()
```

Out[13]:
```
0
```

In [14]:
```
#checking categories for the views column
df['view'].value_counts()
```

Out[14]:
```
NONE         19422
AVERAGE        957
GOOD           508
FAIR           330
EXCELLENT      317
NO              63
Name: view, dtype: int64
```

Since most of the houses have no view, we shall replace the null values with NONE

In [15]:
```
df['view'].fillna('NONE', inplace=True)
```

In [16]:
```
df['view'].isna().sum()
```

Out[16]:
```
0
```

In [17]:
```
#Checking yr renovated
df['yr_renovated'].value_counts()
```

Out[17]:
```
0.0       17011
NO         3842
2014.0       73
2003.0       31
2013.0       31
          ...
1971.0        1
1944.0        1
1934.0        1
1976.0        1
1959.0        1
Name: yr_renovated, Length: 71, dtype: int64
```

The input NO shows the houses that have not been renovated. The 0 values could have been renovated but we cannot tell for sure. Since the percentage of these values is significant, the column has to be dropped.

In [18]:
```
df.drop('yr_renovated',axis=1,inplace=True)
```

In [19]:
```
#changing the sqft_basement to int
df['sqft_basement'].replace('?','0.0',inplace=True)
df['sqft_basement'] = pd.to_numeric(df['sqft_basement'], errors = 'coerce')
```

In [20]:
```
#checking that null values have been dropped
df.info()
```

```
<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     21597 non-null  object 
 9   view           21597 non-null  object 
 10  condition      21597 non-null  object 
 11  grade          21597 non-null  object 
 12  sqft_above     21597 non-null  int64  
 13  sqft_basement  21597 non-null  float64
 14  yr_built       21597 non-null  int64  
 15  zipcode        21597 non-null  int64  
 16  lat            21597 non-null  float64
 17  long           21597 non-null  float64
 18  sqft_living15  21597 non-null  int64  
 19  sqft_lot15     21597 non-null  int64  
dtypes: float64(6), int64(9), object(5)
memory usage: 3.3+ MB
```

In [21]:
```
#checking for duplicates
df.duplicated().any()
```

Out[21]: False

In [22]:
```
df.drop('id', axis=1, inplace=True)
```

There are no duplicates so our data is now ready to use

Hypotheses

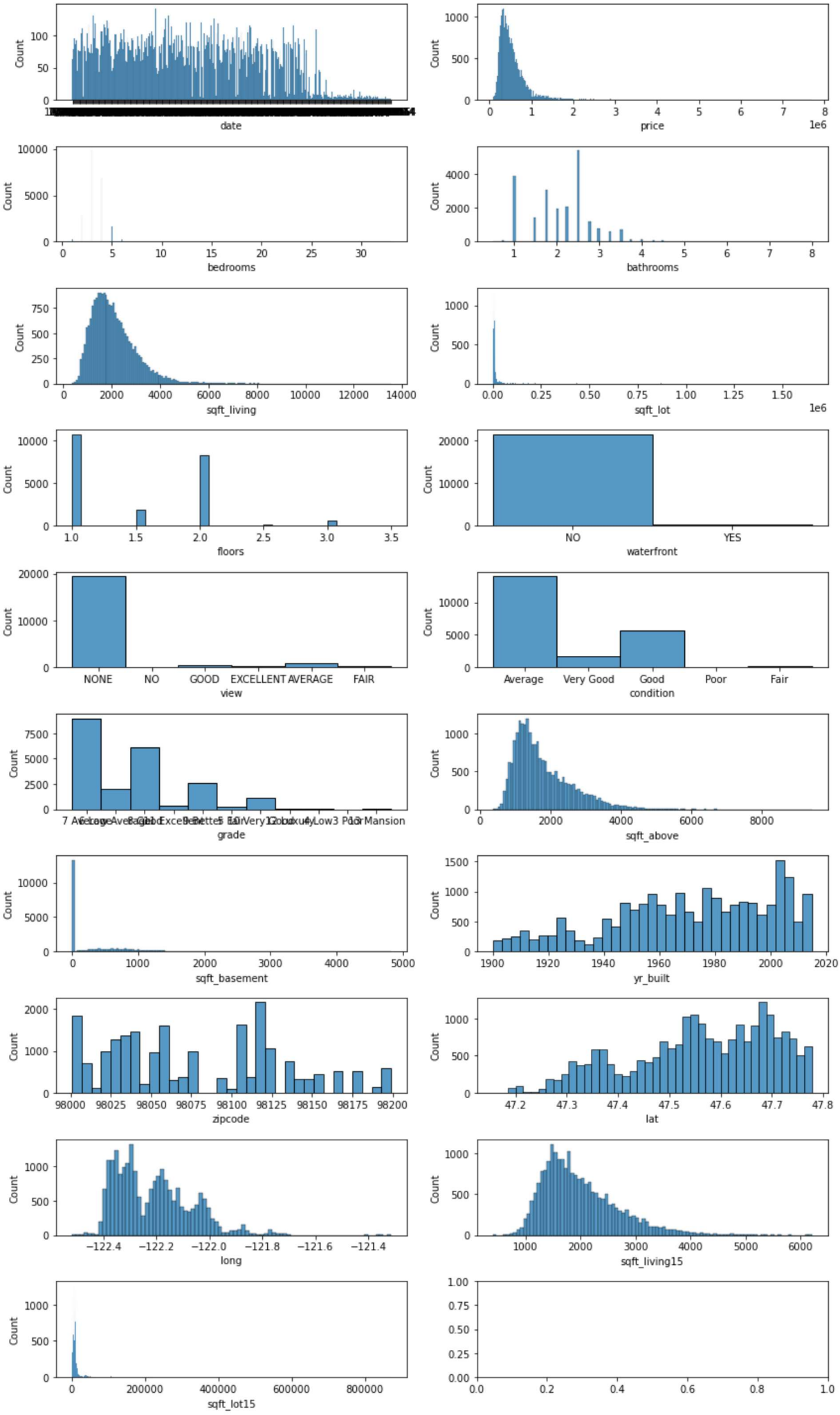
Null Hypothesis: There is no relationship between the features of a house and the price.

Alternative Hypothesis: There is a relationship between the features of a house and the price.

We will use an alpha value of 0.05 which will be used to reject or fail to reject the null hypothesis.

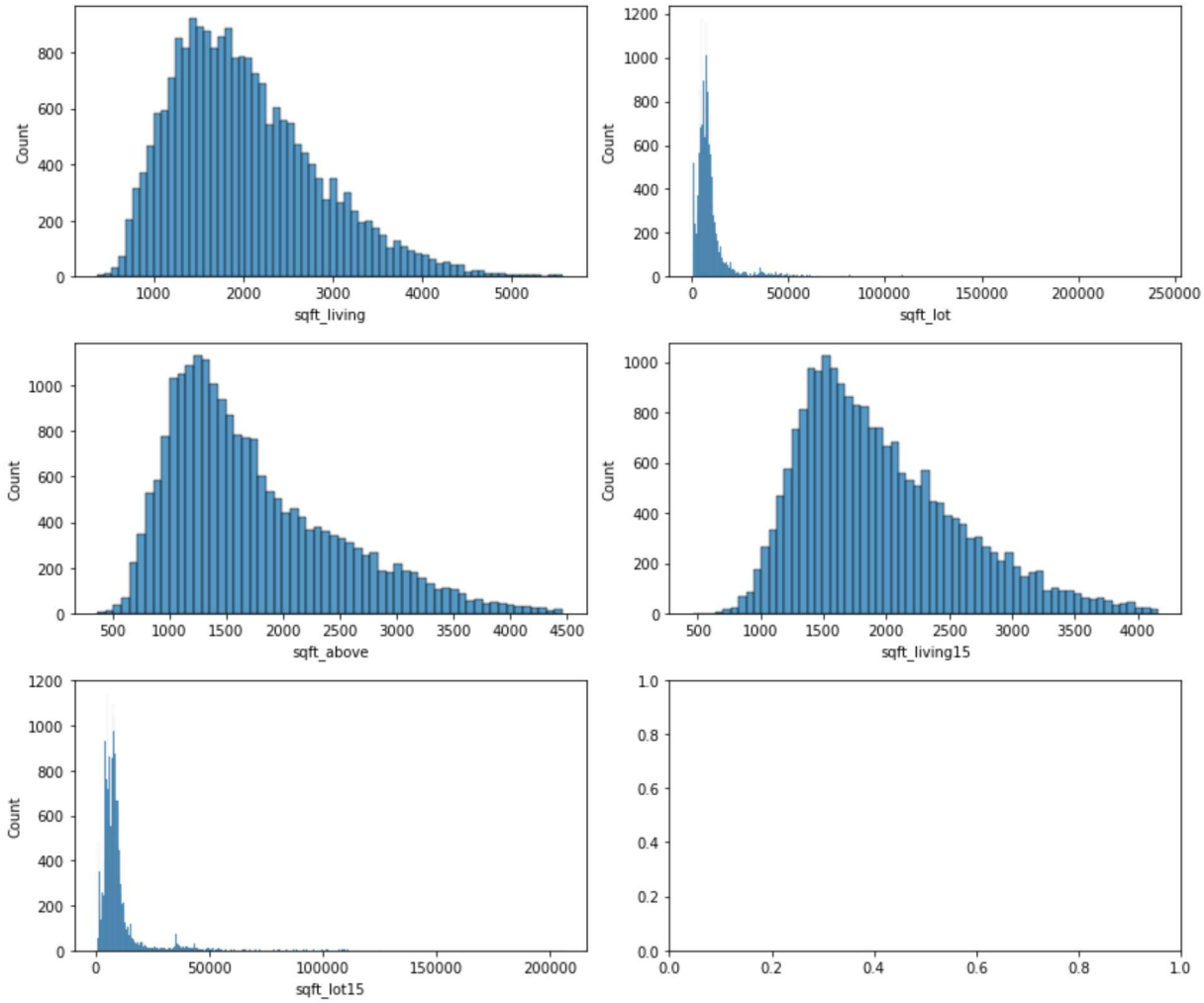
Histograms to check irregularities

In [23]:
```
fig, axs = plt.subplots(nrows=10, ncols=2, figsize=(12, 20))
for index, col in enumerate(df.columns):
    sns.histplot(data=df, x="{}".format(col),ax=axs[index//2, index%2])
plt.tight_layout()
```


Count
date
price
1e6
bedrooms
bathrooms
sqft_living
sqft_lot
1e6
floors
NO
YES
waterfront
NONE
NO
GOOD
EXCELLENT
AVERAGE
FAIR
view
Average
Very Good
Good
Poor
Fair
condition
grade
sqft_above
sqft_basement
yr_built
zipcode
lat
long
sqft_living15
sqft_lot15

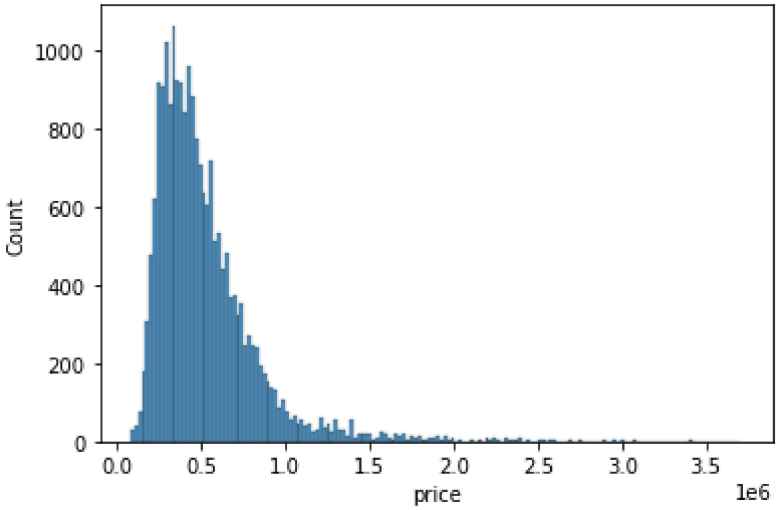
Let us remove outliers from the continuous data under the following categories:sqft_living, sqft_lot, sqft_above, sqft_living15, sqft_lot15 as they have long tails in their histograms.

In [24]:
```
# List features we want to remove outliers from
outliers_to_remove = ['sqft_living', 'sqft_lot', 'sqft_above', 'sqft_living15', 'sqf

# Remove outliers in the top 0.5%
count = 0
for col in outliers_to_remove:
    count += df.loc[df[col] >= df[col].quantile(.995)].copy().shape[0]
    df = df.loc[df[col] < df[col].quantile(.995)].copy()

# Replot histograms
fig, axs = plt.subplots(nrows=3, ncols=2, figsize=(12, 10))
for index, col in enumerate(outliers_to_remove):
    sns.histplot(data=df, x="{}".format(col),ax=axs[index//2, index%2])
plt.tight_layout()
```



Now looking at the dependent variable price in order to remove outliers as these mess up the regression(line of best fit)

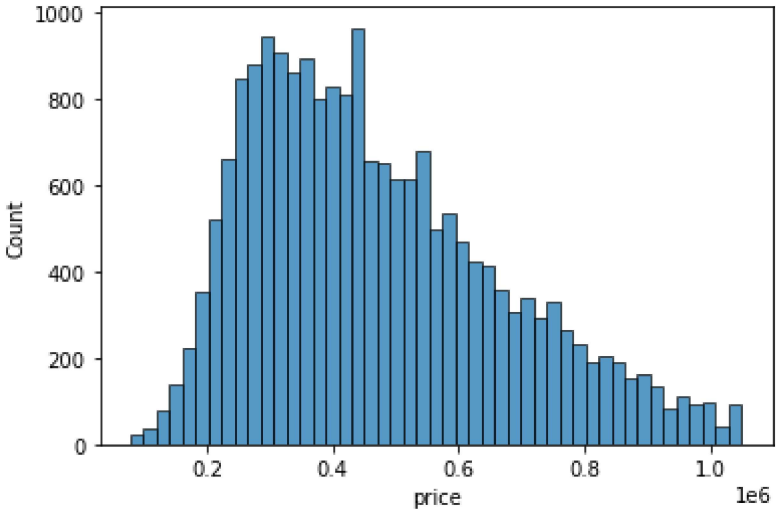
In [25]:
```
sns.histplot(data=df, x='price');
```



In [26]:
```
#removing the top 0.05%
expensive_prices_to_drop = df.loc[df.price > df.price.quantile(.95)].index
df.drop(index=expensive_prices_to_drop, inplace=True)
```

In [27]:
```
#replotting price
sns.histplot(data=df, x='price')
```

Out[27]: <AxesSubplot:xlabel='price', ylabel='Count'>



Train and Test Split

Time to separate our data into training and testing groups

In [28]:
```
from sklearn.model_selection import train_test_split
y = df['price']
x = df.drop('price',axis=1)
X_train, X_test, y_train, y_test = train_test_split(x, y, random_state=69)
```

Feature Scaling

Feature scaling allows us to bring all continous features to the same scale, which will help with comparing the effects of each feature on the model.

In [29]:
```
features_to_scale=['sqft_living','sqft_lot','sqft_above','sqft_basement','sqft_livin
'sqft_lot15']
features_not_to_scale = [ 'bedrooms', 'bathrooms','floors', 'waterfront', 'view', 'c
```

```
scale = X_train[features_to_scale]
no_scale = X_train[features_not_to_scale]
```

In [30]:
```
from sklearn.preprocessing import StandardScaler
scalar = StandardScaler()
X_train_scaled = scalar.fit_transform(scale)
scaled_X_train = pd.DataFrame(data=X_train_scaled, columns=scale.columns, index=scal
```

In [31]:
```
X_train = pd.concat([scaled_X_train, no_scale],axis=1)
```

Encoding Categorical Variables.

Waterfront

Waterfront as seen under data cleaning has two categories so we shall encode it using ordinalencoder

In [32]:
```
from sklearn.preprocessing import OrdinalEncoder
waterfront_train = X_train[['waterfront']]
encoder_waterfront = OrdinalEncoder()
encoder_waterfront.fit(waterfront_train)
encoder_waterfront.categories_[0]
waterfront_encoded_train = encoder_waterfront.transform(waterfront_train)
waterfront_encoded_train = waterfront_encoded_train.flatten()
X_train['waterfront'] = waterfront_encoded_train
```

View

In [33]:
```
X_train['view'].value_counts()
```

Out[33]:
```
NONE         13869
AVERAGE        556
GOOD           238
FAIR           208
EXCELLENT       95
NO              42
Name: view, dtype: int64
```

We shall use onehotencoder

In [34]:
```
from sklearn.preprocessing import OneHotEncoder
view_train = X_train[['view']]
ohe = OneHotEncoder(categories="auto", sparse=False, handle_unknown="ignore")
ohe.fit(view_train)
view_encoded_train = ohe.transform(view_train)
view_encoded_train = pd.DataFrame(view_encoded_train, columns=ohe.categories_[0], in
X_train.drop("view", axis=1, inplace=True)
X_train = pd.concat([X_train, view_encoded_train],axis=1)
```

Condition

In [35]:
```
X_train['condition'].value_counts()
```

Out[35]:
```
Average      9731
Good         3992
Very Good    1147
```

```
Fair          117
Poor           21
Name: condition, dtype: int64
```

In [36]:
```
#using OneHotEncoder to Encode
condition_train = X_train[['condition']]
ohe.fit(condition_train)
condition_encoded_train = ohe.transform(condition_train)
condition_encoded_train = pd.DataFrame(condition_encoded_train, columns=ohe.categori
X_train.drop('condition', axis=1, inplace=True)
X_train = pd.concat([X_train, condition_encoded_train], axis=1)
```

Grade

In [37]:
```
X_train['grade'].value_counts()
```

Out[37]:
```
7 Average        6626
8 Good           4406
9 Better         1664
6 Low Average    1523
10 Very Good      521
5 Fair            180
11 Excellent       72
4 Low              15
3 Poor              1
Name: grade, dtype: int64
```

In [38]:
```
#using onehotencoder
grade_train = X_train[['grade']]
ohe.fit(grade_train)
grade_encoded_train = ohe.transform(grade_train)
grade_encoded_train = pd.DataFrame(grade_encoded_train, columns=ohe.categories_[0],
X_train.drop("grade", axis=1, inplace=True)
X_train = pd.concat([X_train, grade_encoded_train], axis=1)
```

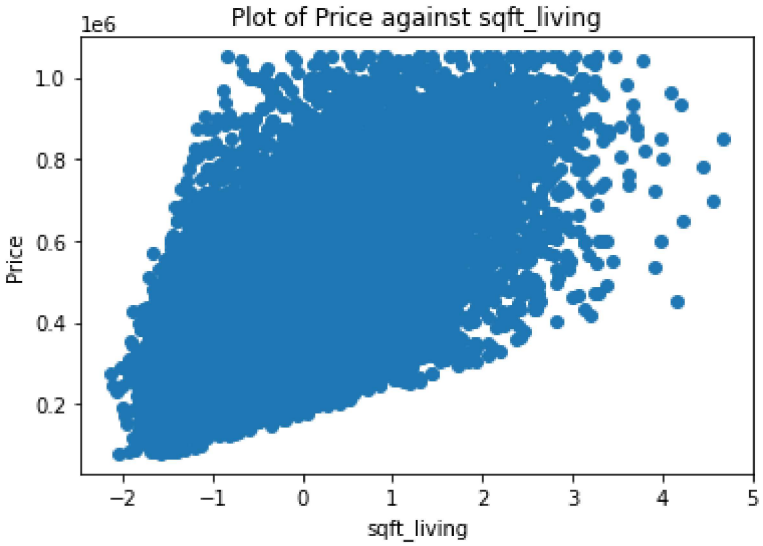
Assumptions of Linear Regression.

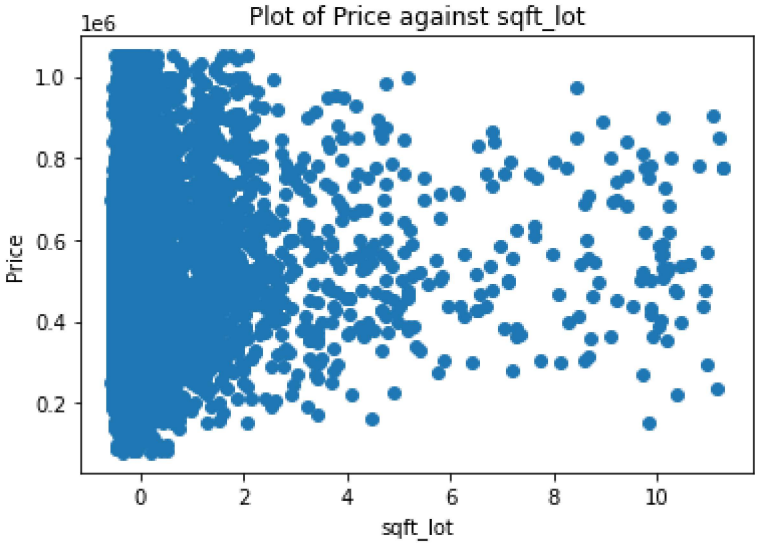
There should be a linear relationship between independent and dependent variables. This can be checked by using scatterplots and looking at correlations. The scatter plots we had already displayed initially.

Linear Relationship

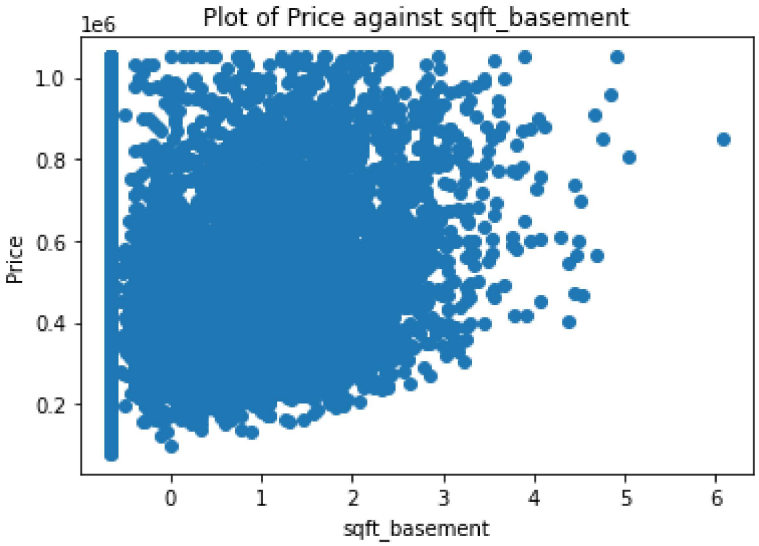
Scatterplots to show correlations betweens certain features and price

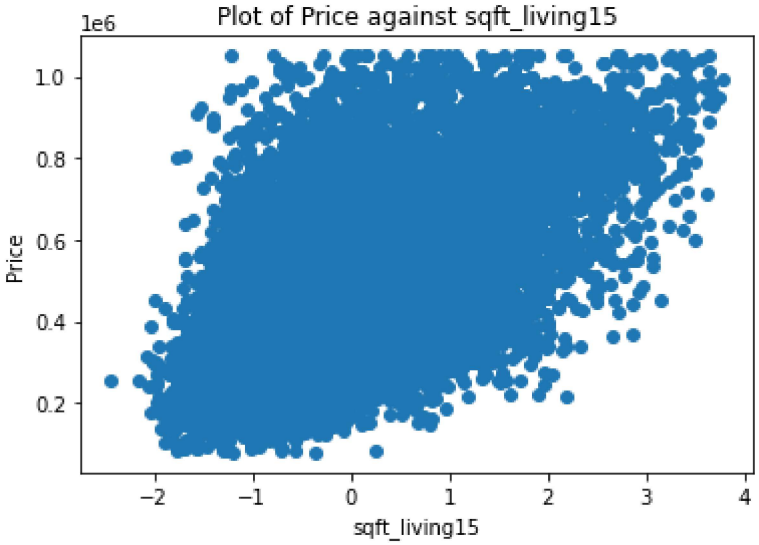
In [39]:
```
for x in X_train.columns:
    plt.scatter(X_train[x], y_train)
    plt.title(f'Plot of Price against {x}')
    plt.xlabel(x)
    plt.ylabel('Price')
    plt.show()
```

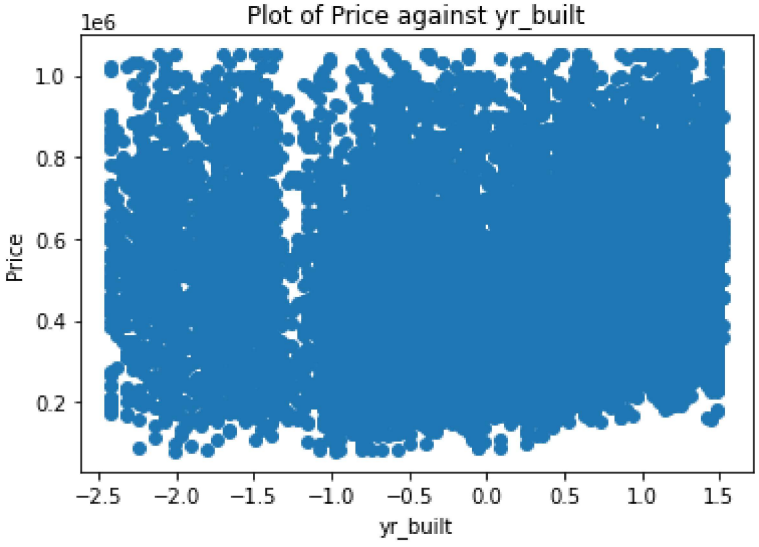

Plot of Price against sqft_living
1e6
1.0
0.8
0.6
0.4
0.2
Price
−2
−1
0
1
2
3
4
5
sqft_living

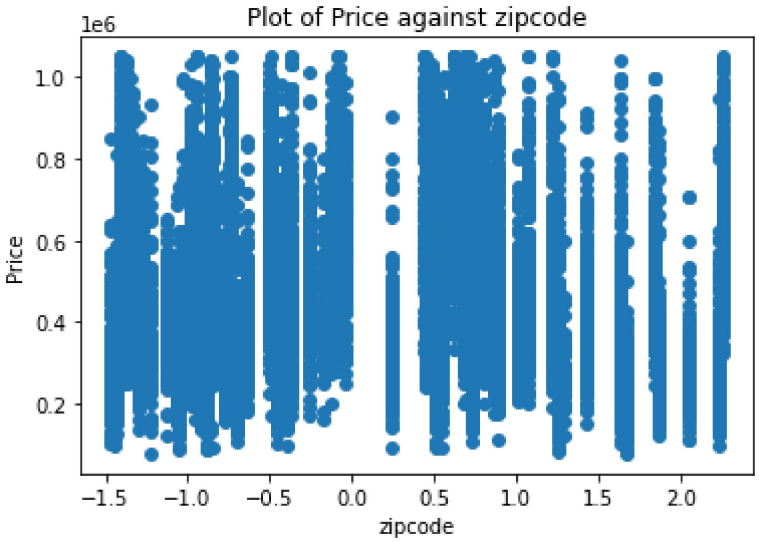

Plot of Price against sqft_lot
1e6
1.0
0.8
0.6
0.4
0.2
Price
0
2
4
6
8
10
sqft_lot

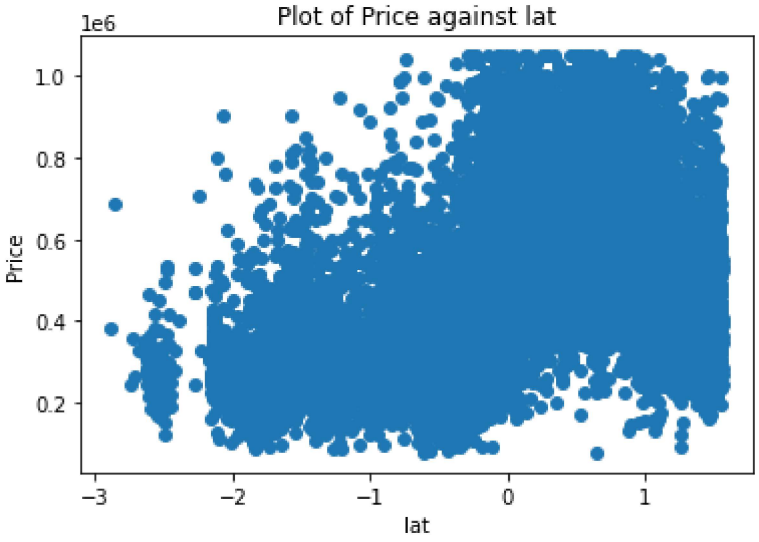

Plot of Price against sqft_above
1e6
1.0
0.8
0.6
0.4
0.2
Price
−2
−1
0
1
2
3
4
sqft_above

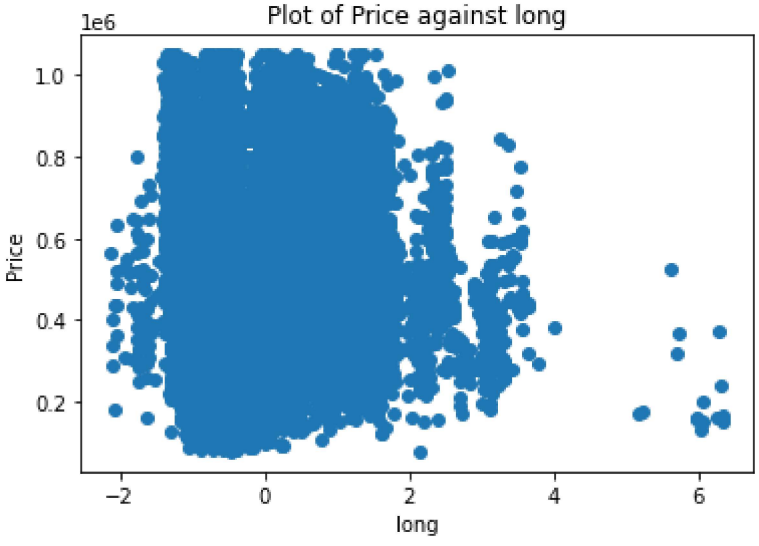

Plot of Price against sqft_basement
1e6
Price
sqft_basement

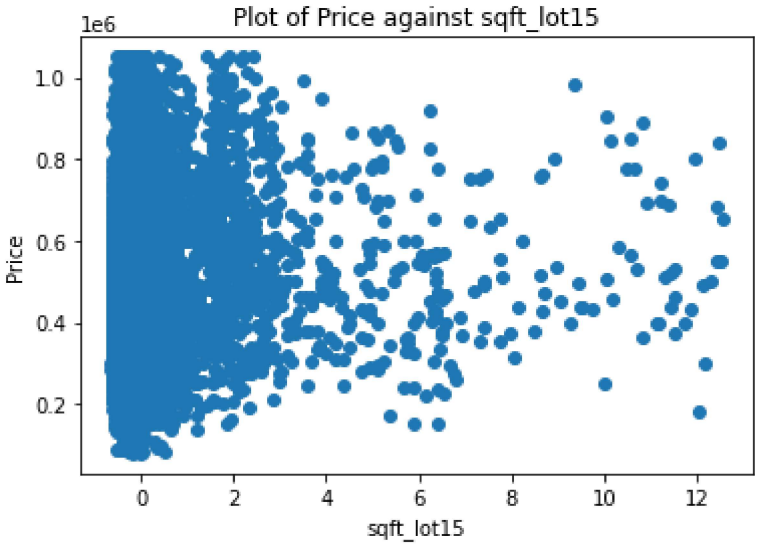

Plot of Price against sqft_living15
1e6
Price
sqft_living15

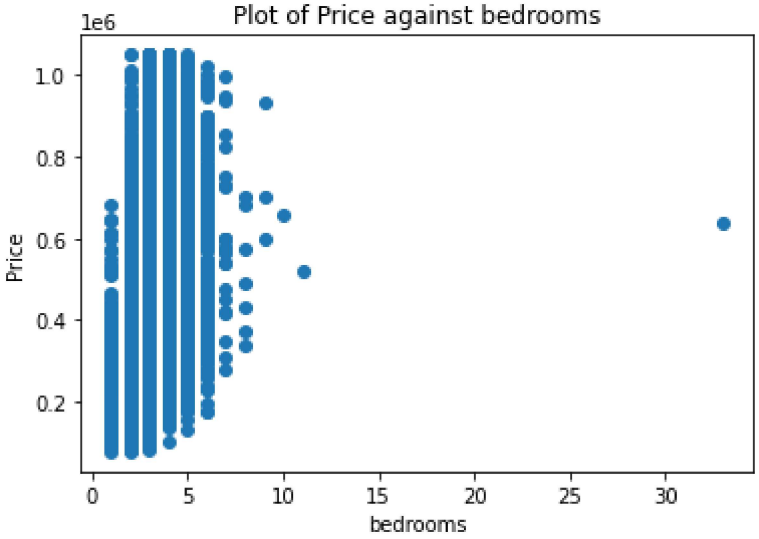

Plot of Price against yr_built
1e6
Price
yr_built

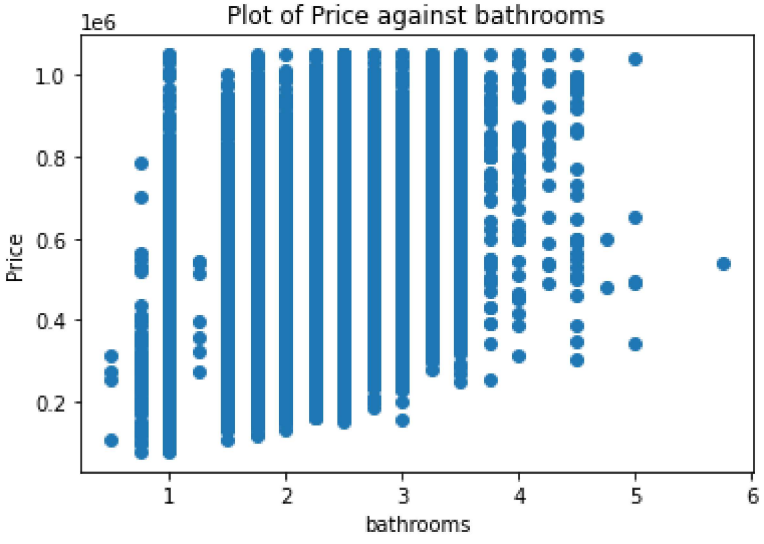

Plot of Price against zipcode
1e6
1.0
0.8
0.6
0.4
0.2
Price
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
zipcode

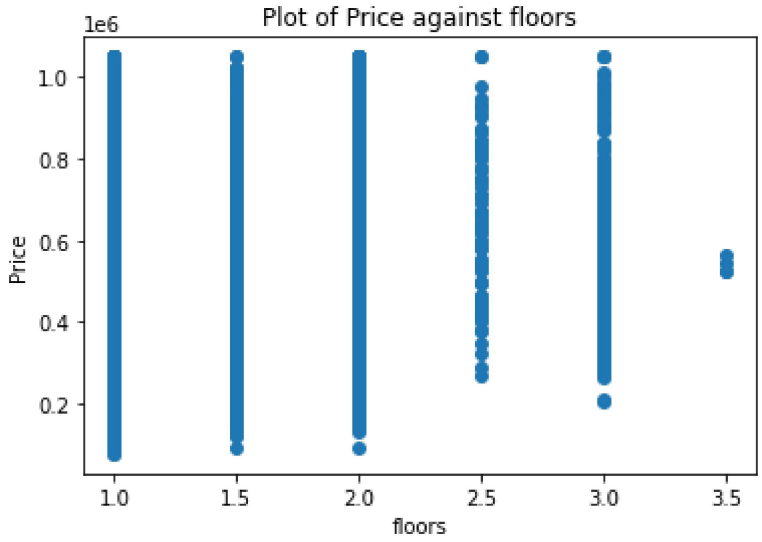

Plot of Price against lat
1e6
1.0
0.8
0.6
0.4
0.2
Price
−3
−2
−1
0
1
lat

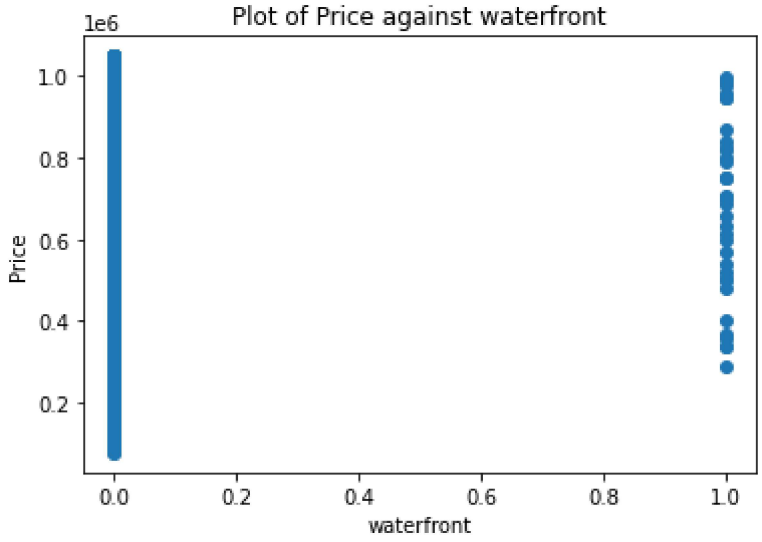

Plot of Price against long
1e6
1.0
0.8
0.6
0.4
0.2
Price
−2
0
2
4
6
long

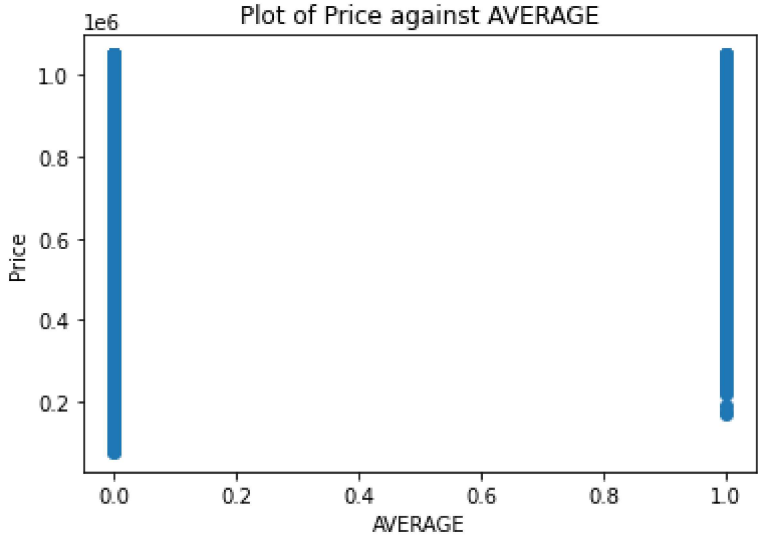

Plot of Price against sqft_lot15
1e6
Price
sqft_lot15

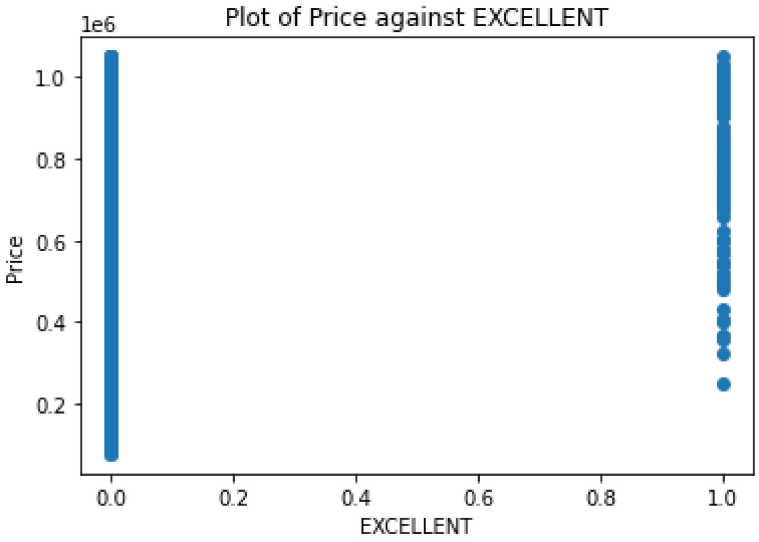

Plot of Price against bedrooms
1e6
Price
bedrooms

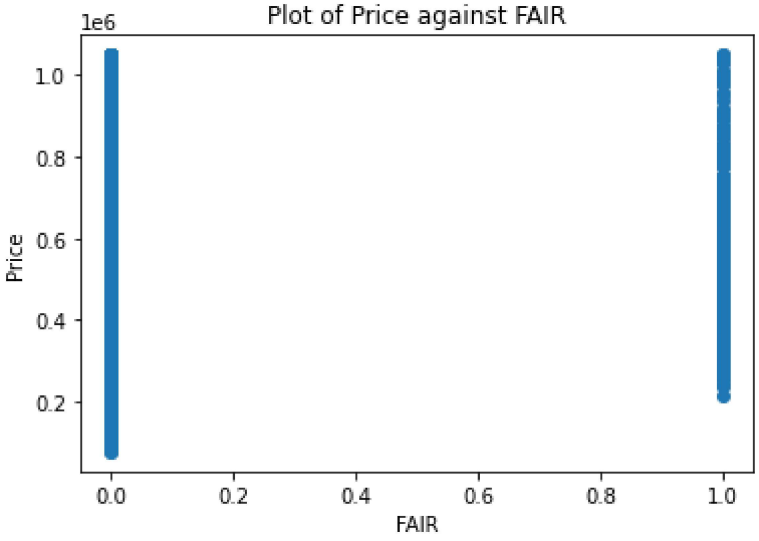

Plot of Price against bathrooms
1e6
Price
bathrooms

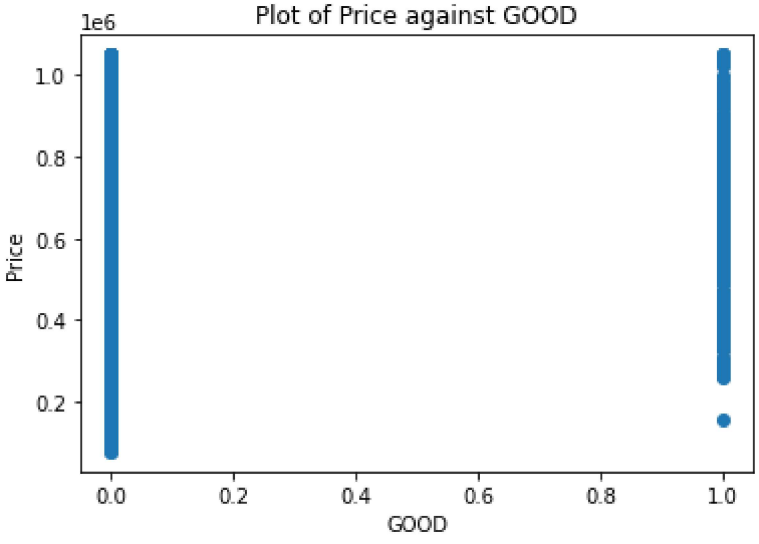

Plot of Price against floors
1e6
1.0
0.8
0.6
0.4
0.2
Price
1.0
1.5
2.0
2.5
3.0
3.5
floors

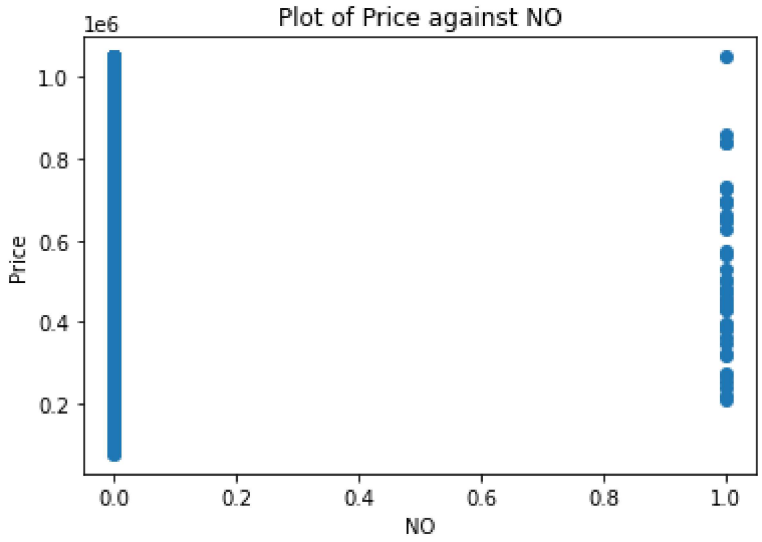

Plot of Price against waterfront
1e6
1.0
0.8
0.6
0.4
0.2
Price
0.0
0.2
0.4
0.6
0.8
1.0
waterfront

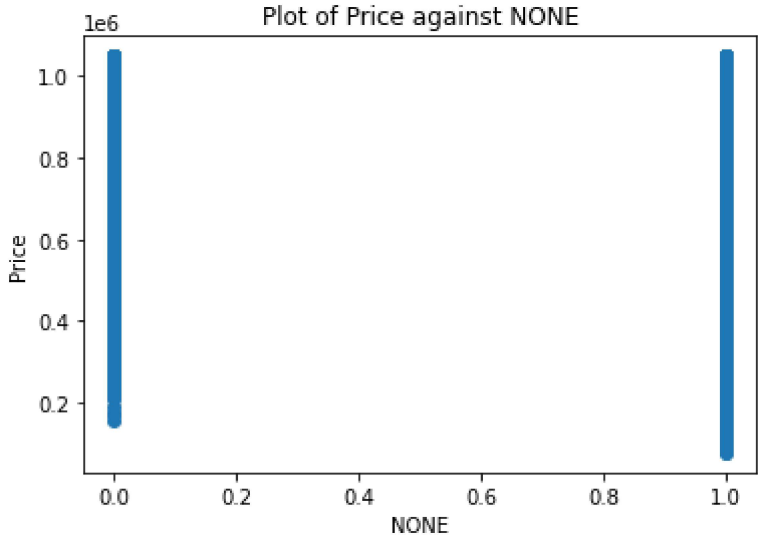

Plot of Price against AVERAGE
1e6
1.0
0.8
0.6
0.4
0.2
Price
0.0
0.2
0.4
0.6
0.8
1.0
AVERAGE

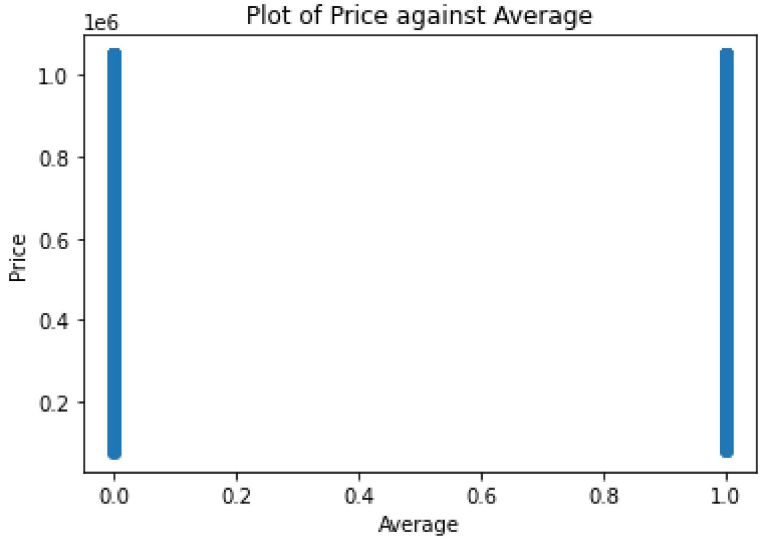

Plot of Price against EXCELLENT
1e6
1.0
0.8
0.6
0.4
0.2
Price
0.0
0.2
0.4
0.6
0.8
1.0
EXCELLENT

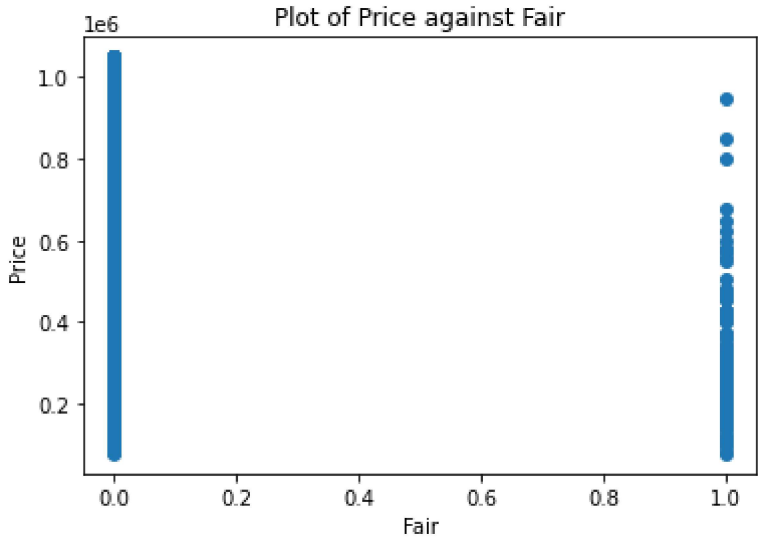

Plot of Price against FAIR
1e6
1.0
0.8
0.6
0.4
0.2
Price
0.0
0.2
0.4
0.6
0.8
1.0
FAIR

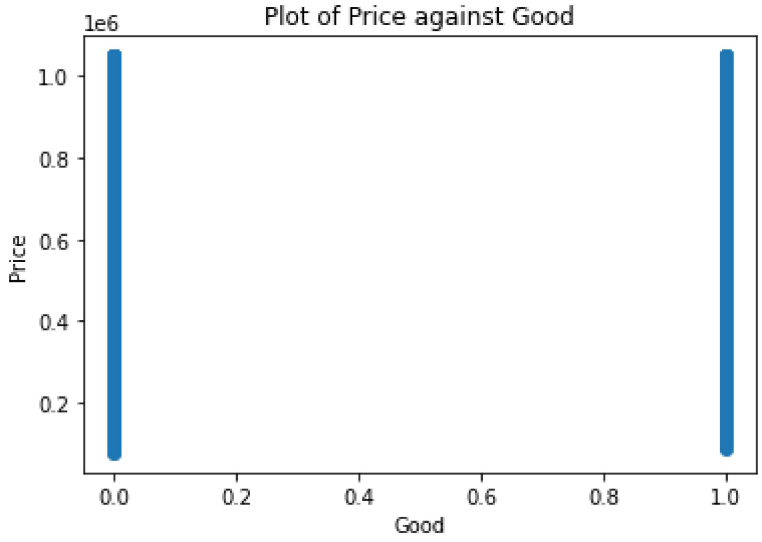

Plot of Price against GOOD
1e6
1.0
0.8
0.6
0.4
0.2
Price
0.0
0.2
0.4
0.6
0.8
1.0
GOOD

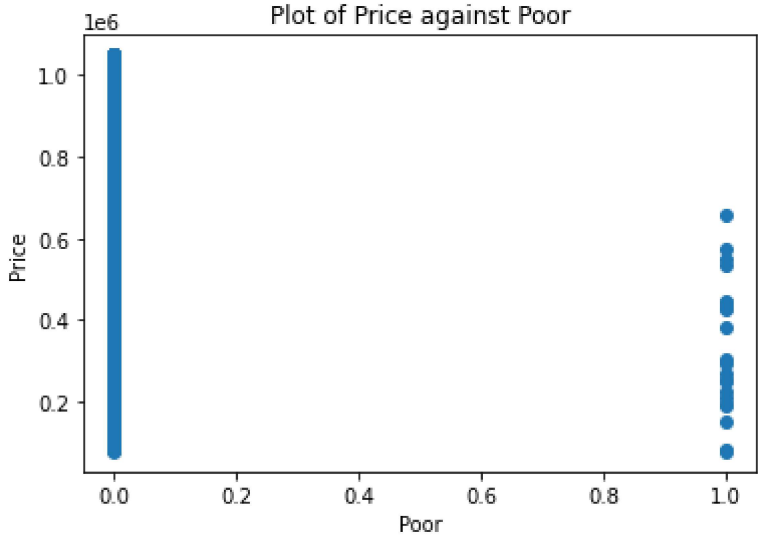

Plot of Price against NO
1e6
1.0
0.8
0.6
0.4
0.2
Price
0.0
0.2
0.4
0.6
0.8
1.0
NO

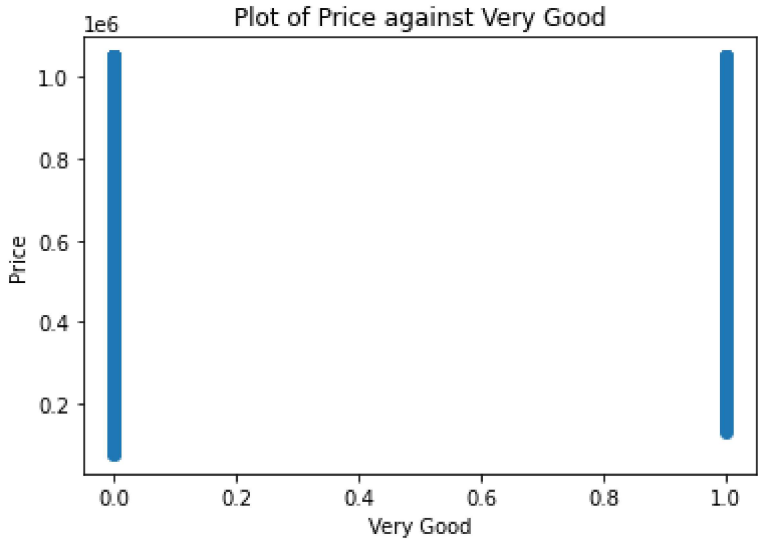

Plot of Price against NONE
1e6
1.0
0.8
0.6
0.4
0.2
Price
0.0
0.2
0.4
0.6
0.8
1.0
NONE

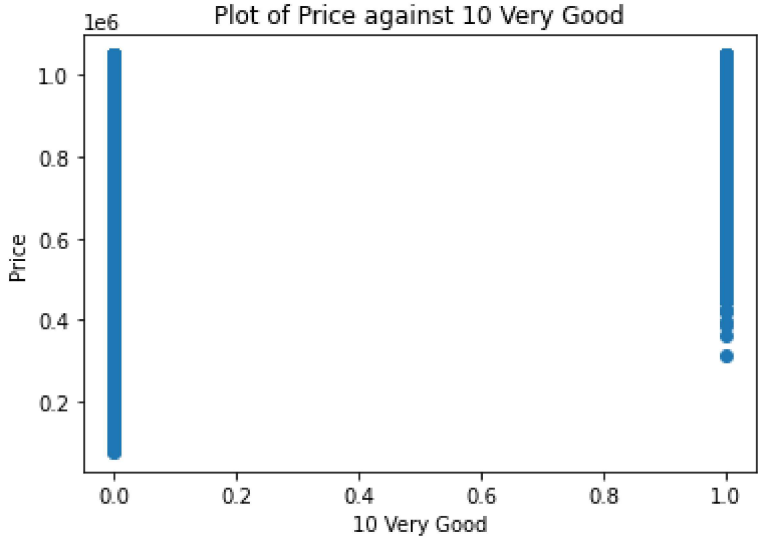

Plot of Price against Average
1e6
1.0
0.8
0.6
0.4
0.2
Price
0.0
0.2
0.4
0.6
0.8
1.0
Average

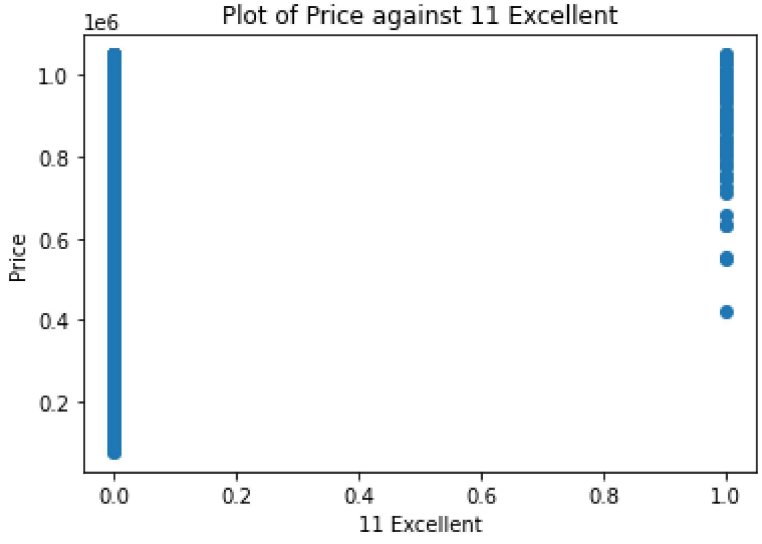

Plot of Price against Fair
1e6
1.0
0.8
0.6
0.4
0.2
Price
0.0
0.2
0.4
0.6
0.8
1.0
Fair

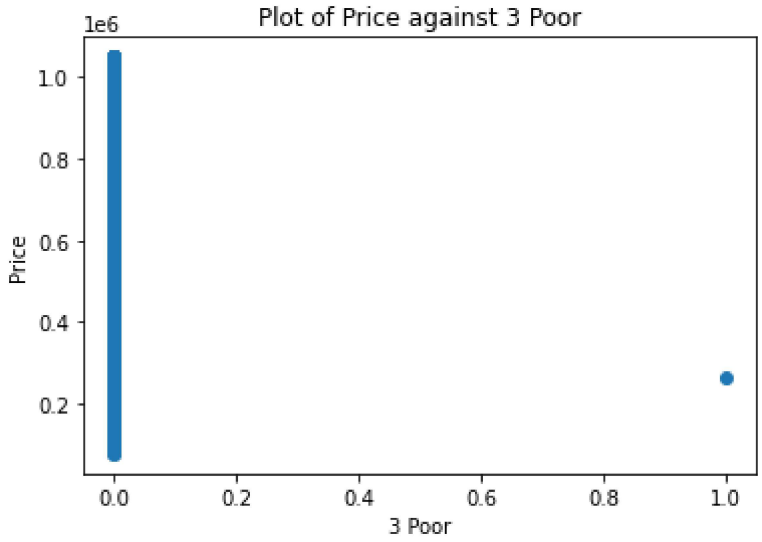

Plot of Price against Good
1e6
1.0
0.8
0.6
0.4
0.2
Price
0.0
0.2
0.4
0.6
0.8
1.0
Good

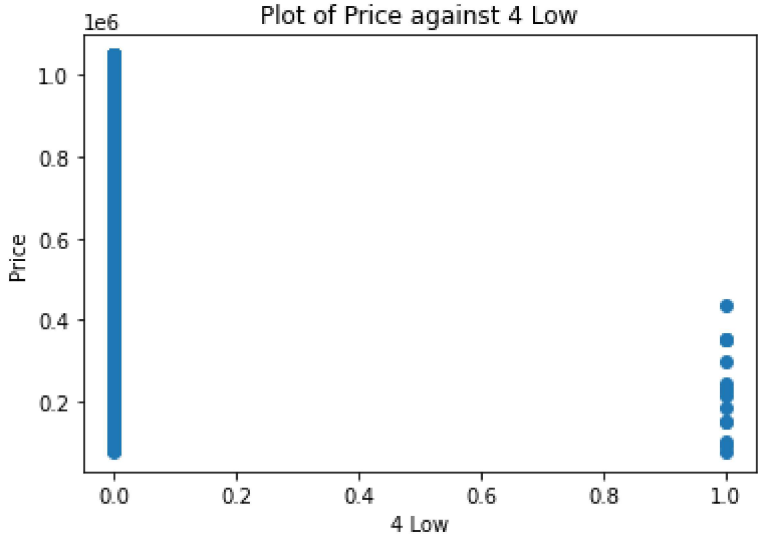

Plot of Price against Poor
1e6
1.0
0.8
0.6
0.4
0.2
Price
0.0
0.2
0.4
0.6
0.8
1.0
Poor

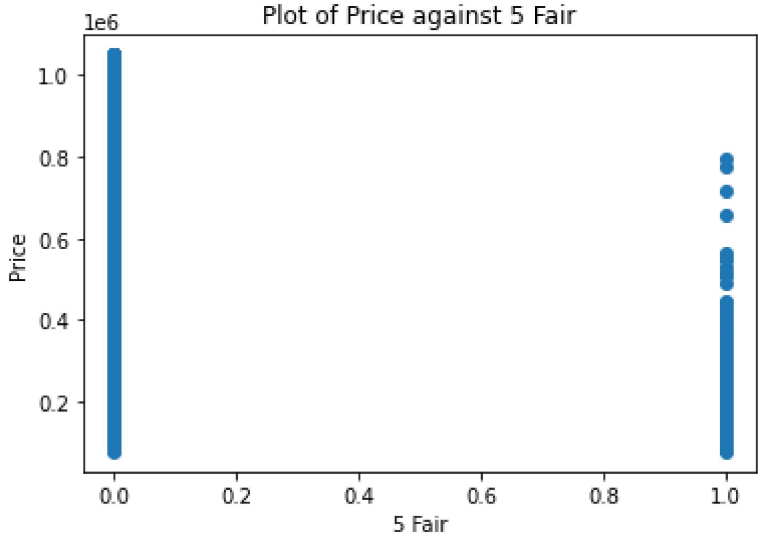

Plot of Price against Very Good
1e6
1.0
0.8
0.6
0.4
0.2
Price
0.0
0.2
0.4
0.6
0.8
1.0
Very Good

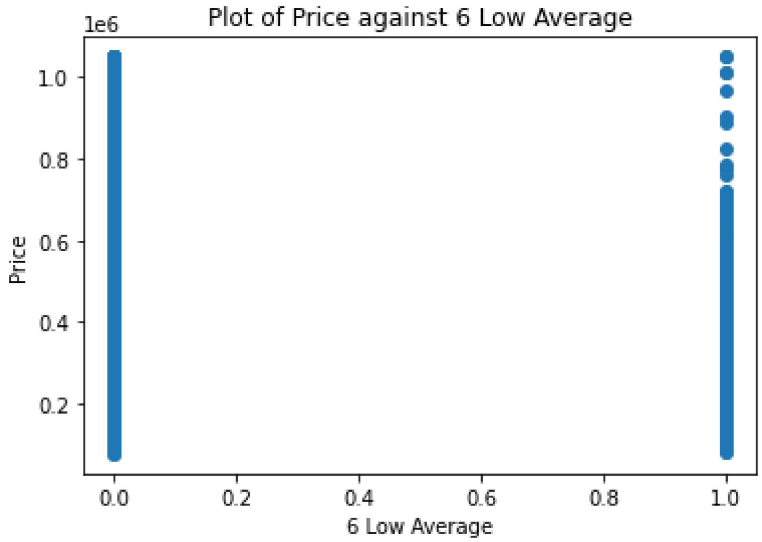

Plot of Price against 10 Very Good
1e6
1.0
0.8
0.6
0.4
0.2
Price
0.0
0.2
0.4
0.6
0.8
1.0
10 Very Good

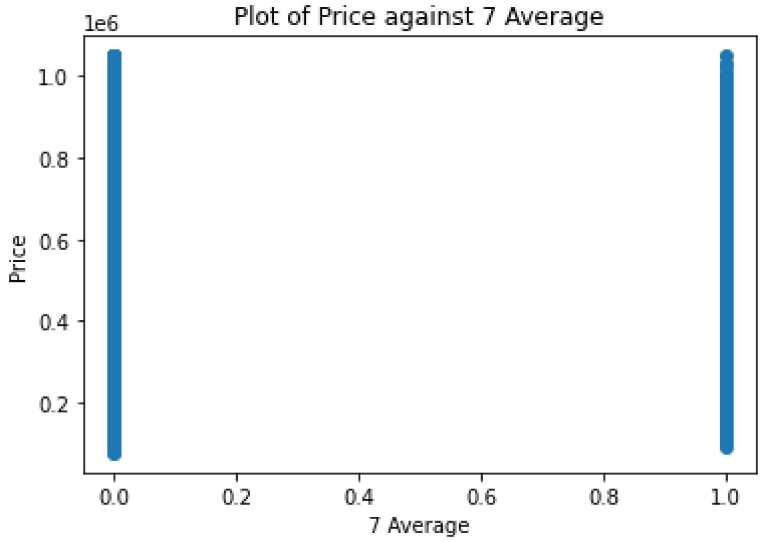

Plot of Price against 11 Excellent
1e6
1.0
0.8
0.6
0.4
0.2
Price
0.0
0.2
0.4
0.6
0.8
1.0
11 Excellent

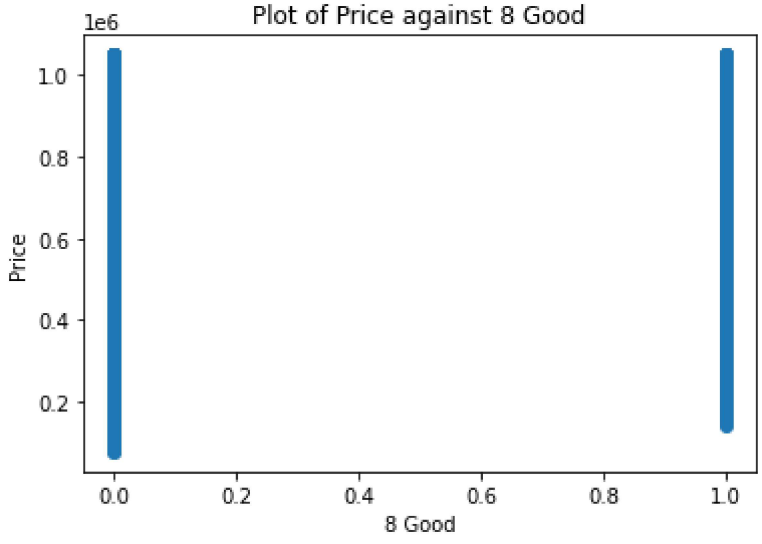

Plot of Price against 3 Poor
1e6
1.0
0.8
0.6
0.4
0.2
Price
0.0
0.2
0.4
0.6
0.8
1.0
3 Poor

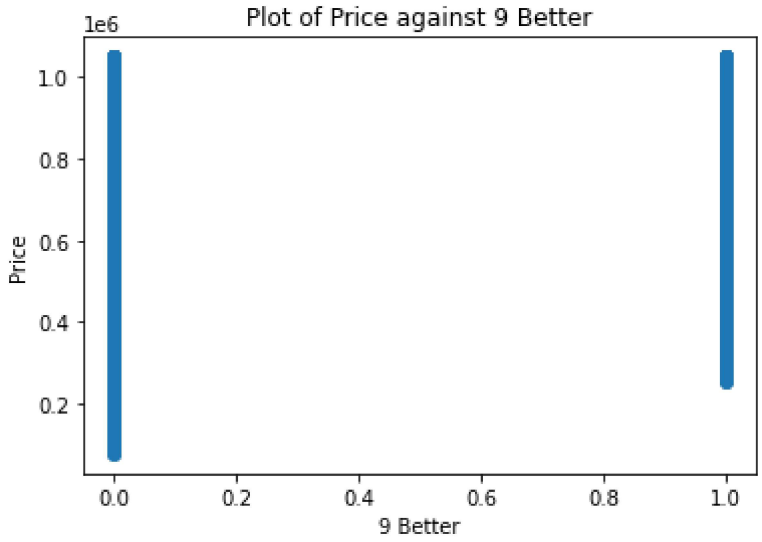

Plot of Price against 4 Low
1e6
1.0
0.8
0.6
0.4
0.2
Price
0.0
0.2
0.4
0.6
0.8
1.0
4 Low


Plot of Price against 5 Fair
1e6
1.0
0.8
0.6
0.4
0.2
Price
0.0
0.2
0.4
0.6
0.8
1.0
5 Fair


Plot of Price against 6 Low Average
1e6
1.0
0.8
0.6
0.4
0.2
Price
0.0
0.2
0.4
0.6
0.8
1.0
6 Low Average


Plot of Price against 7 Average
1e6
1.0
0.8
0.6
0.4
0.2
Price
0.0
0.2
0.4
0.6
0.8
1.0
7 Average


Plot of Price against 8 Good
1e6
1.0
0.8
0.6
0.4
0.2
Price
0.0
0.2
0.4
0.6
0.8
1.0
8 Good



It appears that sqft_living,sqft_above, sqft_living15 and bathrooms columns have a somewhat linear relationship with price.

Checking for correlations in independent variables

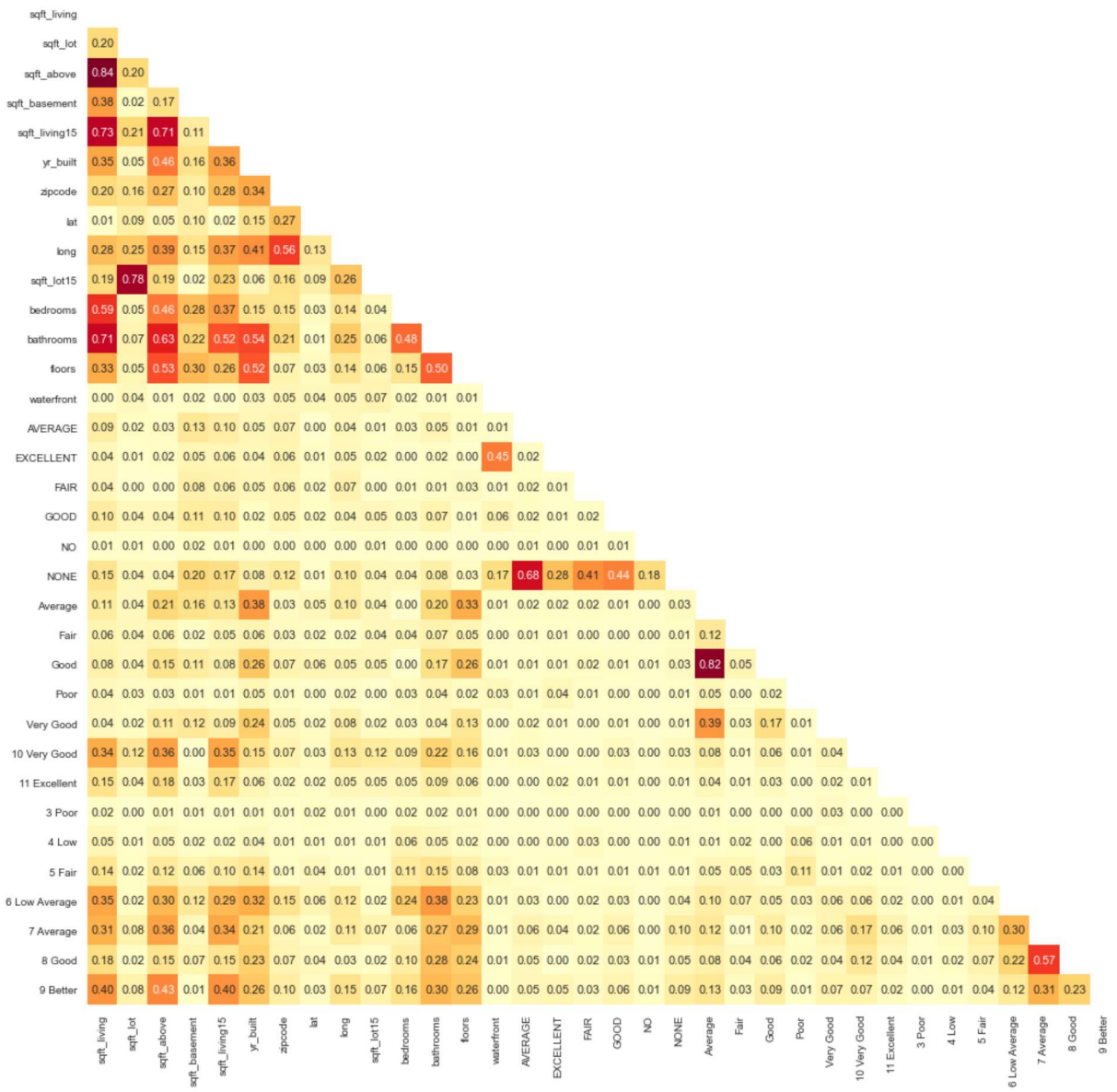
In [40]:
```
independent = pd.concat([X_train, y_train], axis=1)
independent.corr().abs()['price'].sort_values(ascending=True)
```

Out[40]:
```
3 Poor           0.008546
NO               0.010930
Average          0.011891
zipcode          0.012708
Poor             0.026312
Good             0.028222
4 Low            0.037848
yr_built         0.047166
Very Good        0.052696
waterfront       0.053402
long             0.062681
Fair             0.070767
sqft_lot15       0.079055
FAIR             0.086581
sqft_lot         0.092919
GOOD             0.110112
EXCELLENT        0.111498
5 Fair           0.117407
AVERAGE          0.136468
11 Excellent     0.140678
8 Good           0.164294
NONE             0.223066
sqft_basement    0.224108
floors           0.267107
6 Low Average    0.283934
bedrooms         0.286935
10 Very Good     0.300606
7 Average        0.311188
9 Better         0.365444
bathrooms        0.437625
lat              0.441838
sqft_above       0.513856
sqft_living15    0.553118
sqft_living      0.606072
price            1.000000
Name: price, dtype: float64
```

Low Multicollinearity.

Independent variables should not have high collinearity(state of being highly correlated). We can show a heatmap to show the correlations.

In [41]:
```
plt.figure(figsize=(17,17))
sns.set(font_scale=0.9)
mask = np.triu(np.ones_like(X_train.corr(), dtype=bool))
with sns.axes_style("white"):
    sns.heatmap(X_train.corr().abs(), annot=True, fmt='.2f', cmap="YlOrRd",mask=mask
sns.set(font_scale=1)
```



Correlations higher than 0.7 are considered to be highly correlated, so let's look at which predictor features are highly correlated with each other.

In [42]:
```
corr_mtx = abs(X_train.corr())
filtered_corr_mtx = corr_mtx.stack().drop_duplicates()
filtered_corr_mtx[(filtered_corr_mtx.values>=0.7) & (filtered_corr_mtx.values<1)]
```

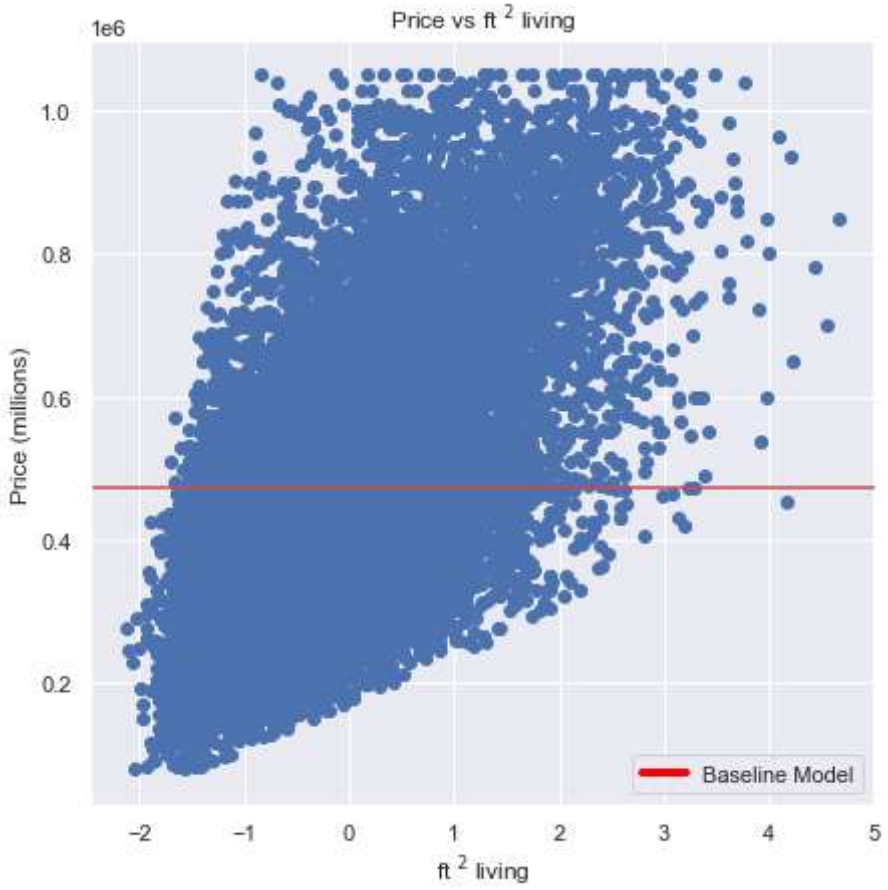
Out[42]:
```
sqft_living   sqft_above       0.844160
              sqft_living15    0.732850
              bathrooms        0.709800
sqft_lot      sqft_lot15       0.779922
sqft_above    sqft_living15    0.714629
Average       Good             0.817464
dtype: float64
```

These shall not be used in the regression as they are highly correlated with sqft_living.

Baseline Model

The baseline model in case of Logistic Regression is to predict the most frequent outcome as the outcome for all data points. (In the case of Linear regression, the baseline model predicts the average of all data points as the outcome)

In [43]:
```
from matplotlib.lines import Line2D
baseline_prediction = np.mean(y_train)
fig, ax = plt.subplots(figsize=(7,7))
ax.scatter(X_train['sqft_living'],y_train)
plt.axhline(y=baseline_prediction, color='r', linestyle='-');
ax.set_xlabel('ft $^2$ living')
ax.set_ylabel('Price (millions)')

cmap = plt.cm.coolwarm
custom_lines = [Line2D([0], [0], color='red', lw=4)]
ax.legend(custom_lines, ['Baseline Model']);
ax.set_title('Price vs ft $^2$ living');
```



In [45]:
```
#Now to run the model
import statsmodels.api as sm
independent1 = np.full(len(y_train), np.mean(y_train))
model1 = sm.OLS(y_train,independent1).fit()
model1.summary()
```

Out[45]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.000
Model:	OLS	**Adj. R-squared:**	0.000
Method:	Least Squares	**F-statistic:**	nan
Date:	Fri, 30 Sep 2022	**Prob (F-statistic):**	nan

Time:	23:26:37	**Log-Likelihood:**	-2.0454e+05
No. Observations:	15008	**AIC:**	4.091e+05
Df Residuals:	15007	**BIC:**	4.091e+05
Df Model:	0		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
x1	1.0000	0.003	288.079	0.000	0.993	1.007

Omnibus:	964.814	**Durbin-Watson:**	1.963
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1137.548
Skew:	0.665	**Prob(JB):**	9.65e-248
Kurtosis:	2.770	**Cond. No.**	1.00

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

As the R-squared value is 0, it shows that none of the dependent variables are explained by the independent variable. This is expected of the baseline model.

In [46]:
```
#Finding the mean squared error(it is used to show the error in models between the o
from sklearn.metrics import mean_squared_error
train_mse_baseline = mean_squared_error(y_train, np.full(len(y_train), np.mean(y_tra
test_mse_baseline = mean_squared_error(y_test, np.full(len(y_test), np.mean(y_train)

print('Baseline Train Mean Squared Error:', train_mse_baseline)
print('Baseline Test Mean Squared Error:', test_mse_baseline)
```

```
Baseline Train Mean Squared Error: 40323975099.31517
Baseline Test Mean Squared Error: 39178571126.21174
```

First Regression Model

In [47]:
```
#separating the variables into dependent and independent
df.head()
```

Out[47]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	cor
0	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NO	NONE	A
1	12/9/2014	538000.0	3	2.25	2570	7242	2.0	NO	NONE	A
2	2/25/2015	180000.0	2	1.00	770	10000	1.0	NO	NONE	A
3	12/9/2014	604000.0	4	3.00	1960	5000	1.0	NO	NONE	
4	2/18/2015	510000.0	3	2.00	1680	8080	1.0	NO	NONE	A

In [48]:
```
model = sm.OLS(y_train, sm.add_constant(X_train))
results = model.fit()
results.summary()
```

Out[48]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.694
Model:	OLS	**Adj. R-squared:**	0.694
Method:	Least Squares	**F-statistic:**	1063.
Date:	Fri, 30 Sep 2022	**Prob (F-statistic):**	0.00
Time:	23:26:50	**Log-Likelihood:**	-1.9565e+05
No. Observations:	15008	**AIC:**	3.914e+05
Df Residuals:	14975	**BIC:**	3.916e+05
Df Model:	32		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.997e+16	8.39e+15	2.380	0.017	3.53e+15	3.64e+16
sqft_living	5.17e+04	9898.181	5.223	0.000	3.23e+04	7.11e+04
sqft_lot	5807.2974	1466.813	3.959	0.000	2932.164	8682.431
sqft_above	470.8845	9263.272	0.051	0.959	-1.77e+04	1.86e+04
sqft_basement	9.0342	5293.308	0.002	0.999	-1.04e+04	1.04e+04
sqft_living15	2.818e+04	1531.904	18.394	0.000	2.52e+04	3.12e+04
yr_built	-5.498e+04	1398.324	-39.318	0.000	-5.77e+04	-5.22e+04
zipcode	-1.247e+04	1179.455	-10.570	0.000	-1.48e+04	-1.02e+04
lat	7.619e+04	987.499	77.155	0.000	7.43e+04	7.81e+04
long	-6384.5219	1235.067	-5.169	0.000	-8805.404	-3963.640
sqft_lot15	-5444.2161	1472.197	-3.698	0.000	-8329.902	-2558.530
bedrooms	-8553.3195	1290.122	-6.630	0.000	-1.11e+04	-6024.523
bathrooms	2.412e+04	2246.475	10.738	0.000	1.97e+04	2.85e+04
floors	3.283e+04	2474.155	13.270	0.000	2.8e+04	3.77e+04
waterfront	1.548e+05	2.07e+04	7.473	0.000	1.14e+05	1.95e+05
AVERAGE	-1.963e+16	8.25e+15	-2.380	0.017	-3.58e+16	-3.47e+15
EXCELLENT	-1.963e+16	8.25e+15	-2.380	0.017	-3.58e+16	-3.47e+15
FAIR	-1.963e+16	8.25e+15	-2.380	0.017	-3.58e+16	-3.47e+15
GOOD	-1.963e+16	8.25e+15	-2.380	0.017	-3.58e+16	-3.47e+15
NO	-1.963e+16	8.25e+15	-2.380	0.017	-3.58e+16	-3.47e+15
NONE	-1.963e+16	8.25e+15	-2.380	0.017	-3.58e+16	-3.47e+15
Average	-3.453e+15	1.45e+15	-2.380	0.017	-6.3e+15	-6.1e+14
Fair	-3.453e+15	1.45e+15	-2.380	0.017	-6.3e+15	-6.1e+14
Good	-3.453e+15	1.45e+15	-2.380	0.017	-6.3e+15	-6.1e+14

Poor	-3.453e+15	1.45e+15	-2.380	0.017	-6.3e+15	-6.1e+14
Very Good	-3.453e+15	1.45e+15	-2.380	0.017	-6.3e+15	-6.1e+14
10 Very Good	3.117e+15	1.31e+15	2.380	0.017	5.5e+14	5.68e+15
11 Excellent	3.117e+15	1.31e+15	2.380	0.017	5.5e+14	5.68e+15
3 Poor	3.117e+15	1.31e+15	2.380	0.017	5.5e+14	5.68e+15
4 Low	3.117e+15	1.31e+15	2.380	0.017	5.5e+14	5.68e+15
5 Fair	3.117e+15	1.31e+15	2.380	0.017	5.5e+14	5.68e+15
6 Low Average	3.117e+15	1.31e+15	2.380	0.017	5.5e+14	5.68e+15
7 Average	3.117e+15	1.31e+15	2.380	0.017	5.5e+14	5.68e+15
8 Good	3.117e+15	1.31e+15	2.380	0.017	5.5e+14	5.68e+15
9 Better	3.117e+15	1.31e+15	2.380	0.017	5.5e+14	5.68e+15

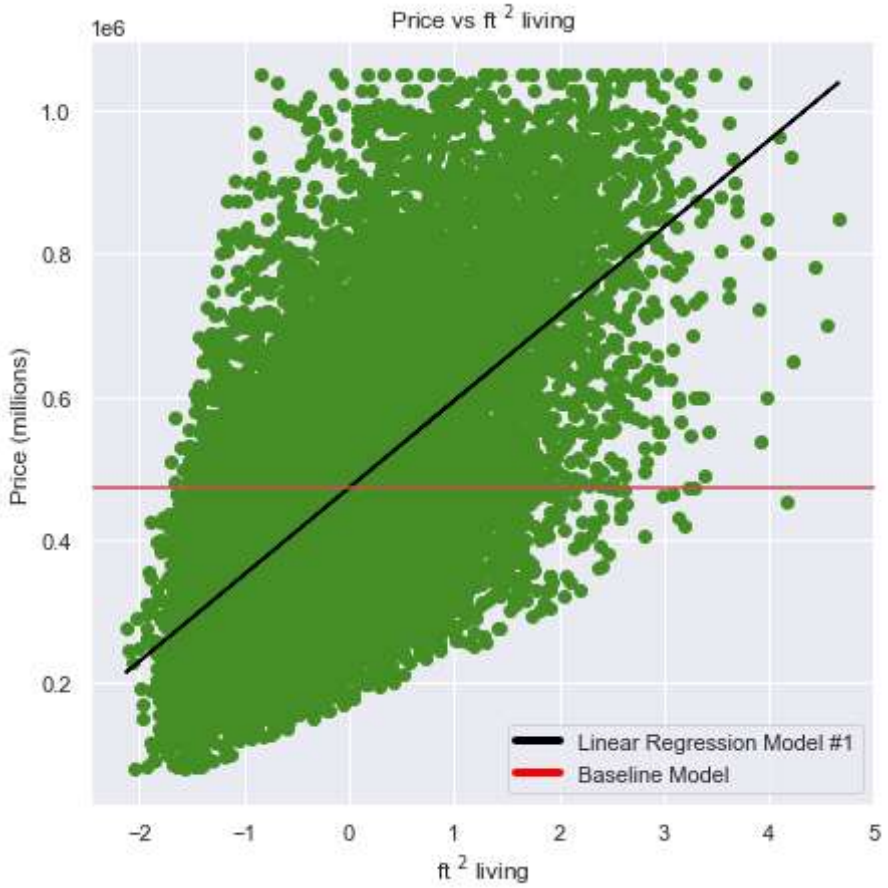
Omnibus:	1033.490	**Durbin-Watson:**	2.000
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1758.453
Skew:	0.528	**Prob(JB):**	0.00
Kurtosis:	4.303	**Cond. No.**	9.32e+15

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.64e-27. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

Only 69.4% of the dependent variable is explained by the independent variable in this model.
We will use sklearn to extract the intercept and coefficient.

In [50]:
```
from sklearn.linear_model import LinearRegression
linreg_modelone = LinearRegression()
linreg_modelone.fit(X_train[['sqft_living']], y_train)
coef_m1 = linreg_modelone.coef_
intercept_m1 = linreg_modelone.intercept_
```

In [51]:
```
#Comparing to the baseline model
fig, ax = plt.subplots(figsize=(7,7))
ax.scatter(X_train[['sqft_living']], y_train, color='#458D25')
ax.plot(X_train[['sqft_living']], intercept_m1 + coef_m1 * X_train[['sqft_living']],
plt.axhline(y=baseline_prediction,color='r', linestyle='-');
ax.set_xlabel('ft $^2$ living')
ax.set_ylabel('Price (millions)');

cmap = plt.cm.coolwarm
from matplotlib.lines import Line2D
custom_lines = [Line2D([0], [0], color='black', lw=4),
                Line2D([0], [0], color='red', lw=4)]

ax.legend(custom_lines, ['Linear Regression Model #1', 'Baseline Model']);
ax.set_title('Price vs ft $^2$ living');
```



In [52]:
```
#Now to calculate the mean squared error and compare the two regressions
y_pred = results.predict(sm.add_constant(X_train))
first_mse_model=mean_squared_error(y_train, y_pred)
first_mse_model
```

Out[52]: 12328219001.840534

In [53]:
```
#Comparing with our baseline model
first_mse_model<train_mse_baseline
```

Out[53]: True

The model is thus better as it has a lesser mean squared error.

Model Interpretation

1.The adjusted R and the R.squared value are close to each other meaning our model reflects relevant features that influence the price.

2.The F-statistic is large and the p-value is 0. We can therefore reject the null hypothesis. There is evidence of a linear relation between the variables.

3.The p-values for sqft_basement and sqft_above are greater than 0.05. Since we carried out a test for a 95% confidence interval, we fail to reject the null hypothesis for these variables. On the contrary, since the p-values for all the other variables are less than 0.05, we reject the null hypothesis and accept the alternate hypothesis

Conclusions from the model

1.The higher the sqft_living which stands for the number of square feet for the house, the higher the price of the house.An increase of $51780 is seen

2.When all features are constant, there is an increase of $30910.

3.Grades above 8 record an increase in price. The higher the grade the higher the increase.

4.The better the view the higher the increase in price

I decided to focus on view, grade as they have a smaller collinearity with sqft_living as per the heatmap