Final Project Submission

Please fill out:

- Student name: Leshmi Jayakumar
- Student pace: Part time
- Scheduled project review date/time: 12/02/2023
- Instructor name: Hardik Idnani

KING COUNTRY HOUSE DATA ANALYSIS

****Column Names and descriptions for Kings County Data Set**

- **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

Goal

The goal of this project is to create a relatively accurate prediction model for the prices that future houses sell for. We will explore how different factors affect the pricing the homes, given the data, and compile the most important feature.

In [1]:

```
# First, let's import the libraries we are going to use
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# Ignores warnings
import warnings
warnings.filterwarnings('ignore')

# Used for working with the z-score
from scipy import stats
from scipy import interpolate
from itertools import combinations
#For OLS model
import statsmodels.api as sm
import statsmodels.formula.api as smf
import scipy.stats as stats
import statsmodels.stats.api as sms
from statsmodels.formula.api import ols
import pylab

# Used for Linear Regression model and Cross Validation model
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_error, make_scorer
from sklearn.model_selection import cross_val_score
from sklearn import metrics

# Used when showing normalization of data
from statsmodels.stats.diagnostic import normal_ad

# Used when creating bar plots and using median instead of mean
from numpy import median
```

In [2]:

```
#Let's import our King's County housing data and take a look
data_k=pd.read_csv('data/kc_house_data.csv')
```

In [3]:

```
data_k.head()
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterf
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	

5 rows × 21 columns

In [4]:

```
#Data information
data_k.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  float64
 9   view           21534 non-null  float64
 10  condition      21597 non-null  int64  
 11  grade          21597 non-null  int64  
 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(8), int64(11), object(2)
memory usage: 3.5+ MB
```

In [5]:

```
#Convert date to date time
data_k['date'] =  pd.to_datetime(data_k['date'], format='%m/%d/%Y')
#Convert sqft_basement to float
data_k['sqft_basement'] = pd.to_numeric(data_k['sqft_basement'], errors="coerce")
#using errors='coerce' because sqft_basement contains '?' string values. These  values will
```

In [6]:

```
#after converting the dtype of sqft_basement and Date
data_k.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  datetime64[ns]
 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_above     21597 non-null  int64
 13  sqft_basement  21143 non-null  float64
 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: datetime64[ns](1), float64(9), int64(11)
memory usage: 3.5 MB
```

In [7]:

```
# Handle the missing data.

print(data_k.isnull().sum())
print(data_k.shape)
```

```
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     454
yr_built            0
yr_renovated     3842
zipcode             0
lat                 0
long                0
sqft_living15       0
sqft_lot15          0
dtype: int64
(21597, 21)
```

In [8]:

```
#For sqft_basement, waterfront and view, replace missing values with 0
data_k['sqft_basement'].fillna(0, inplace=True)
data_k['waterfront'].fillna(0, inplace=True)
data_k['view'].fillna(0, inplace=True)
#For yr_renovated, set yr_renovated to yr_built
data_k['yr_renovated'].fillna(data_k[data_k['yr_renovated'].isna()]['yr_built'], inplace=Tr
```

In [9]:

```python
print(data_k.isnull().sum())
```

```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

Questions

Will the price of the house positvely correlated with :

```
* a)No:of bedrooms
* b)Sqft_living
* c)Condition
* d)Grade
* e)Zipcode
```

Baseline

In [10]:

```
#Let's describe our data now that all our values have been cleaned up.
data_k.describe()
```

Out[10]:

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

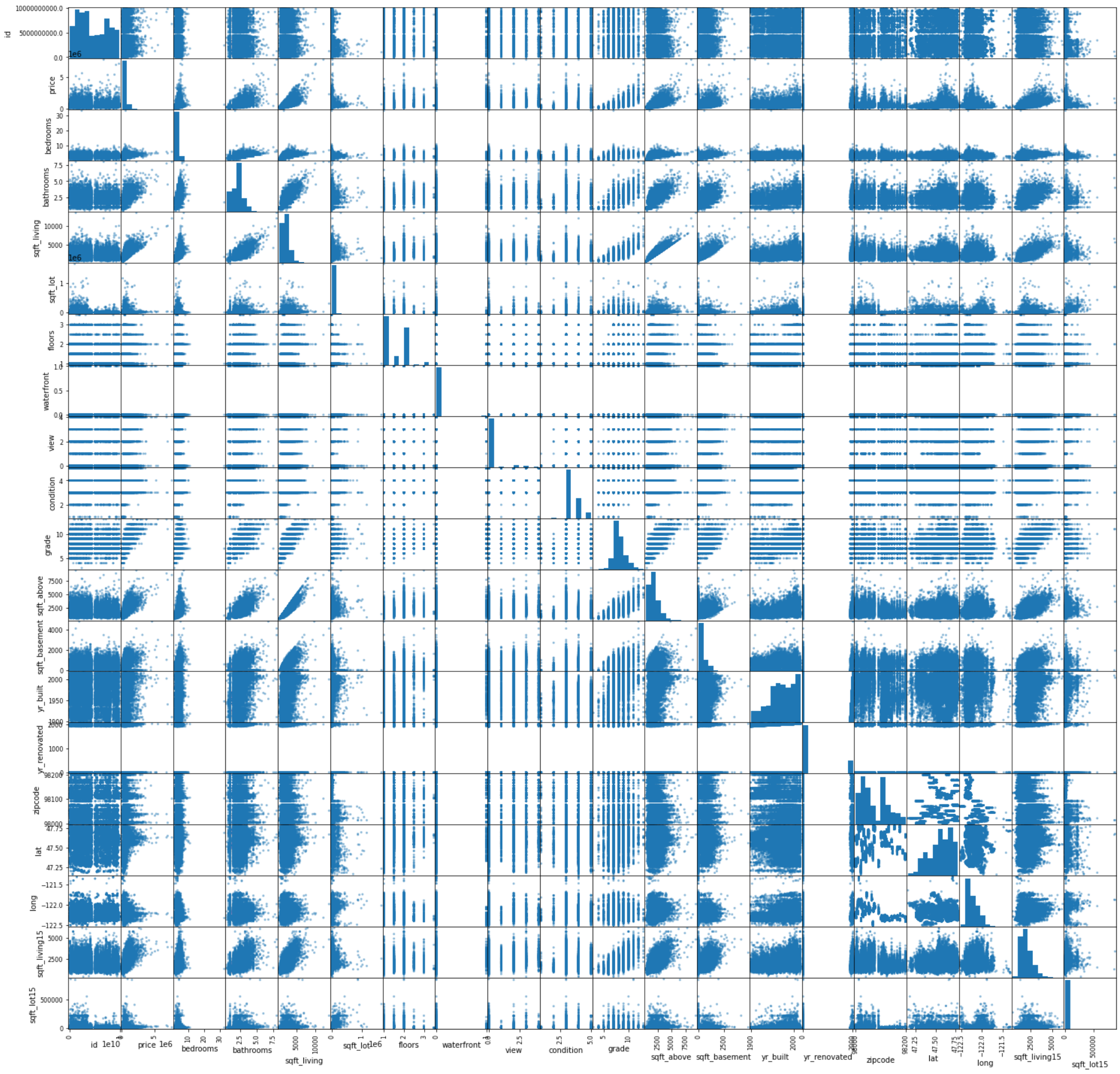
Looking at the information above we can see the following:

1) Our target variable, Price, has a mean value of roughly 540K. The lowest price is 78K and the highest price is over 7 million. We can also see that 50% of our data falls below the 500K price. 2) We can also see some outliers such as a home with 33 bedrooms and 8 bathrooms. 3) We also notice that some of our data is categorical, such as zipcode and waterfront.

Now, let's run some graphics in order to better look at our data

In [11]:

```
#Scatter plot of the data
pd.plotting.scatter_matrix(data_k, figsize=(25,25));
```



Looking at our scatter matrix above, we can see interesting relationships in our data, but because the data is difficult to actually look at.

In [12]:

```
#checking the mean and standard deviation
price_mean = data_k['price'].mean()
price_std = data_k['price'].std()
price_mean, price_std
```

Out[12]:

```
(540296.5735055795, 367368.1401013945)
```

In [13]:

```
#checking the average price of the house according to the year built
avg_price_yr=data_k.groupby(['yr_built']).price.mean()
avg_price_yr.sort_index()
```

Out[13]:

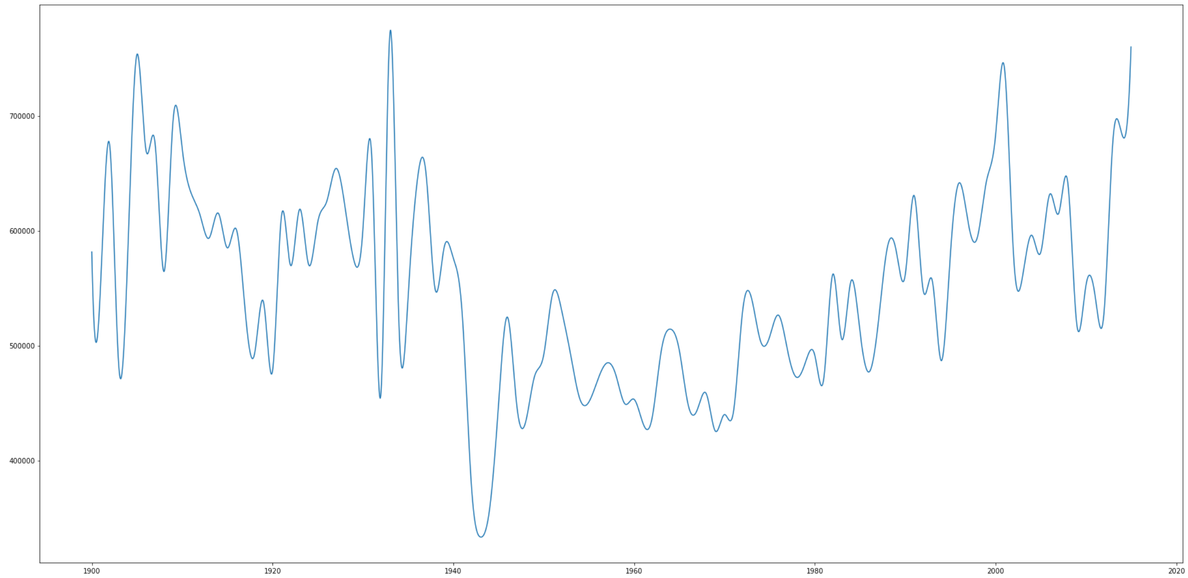
```
yr_built
1900    581536.632184
1901    557108.344828
1902    673192.592593
1903    480958.195652
1904    583867.755556
            ...
2011    544648.384615
2012    527436.982353
2013    678599.582090
2014    683792.685152
2015    759970.947368
Name: price, Length: 116, dtype: float64
```

In [14]:

```
#Data visulisation according to the price and year built
T=avg_price_yr.index
X=np.linspace(T.min(),T.max(),10000)
Y=avg_price_yr.values
spl=interpolate.make_interp_spline(T,Y,3)
test=spl(X)

fig,ax=plt.subplots(figsize=(30,15))
ax.plot(X,test)
# Interpolate to make more smooth(Interpolation in Python is a technique used to estimate u
```

Out[14]:

```
[<matplotlib.lines.Line2D at 0x2a49bb75880>]
```



In [299]:

```
Answer:This figure shows that the year built doesnot affect the price and it also showsther
```

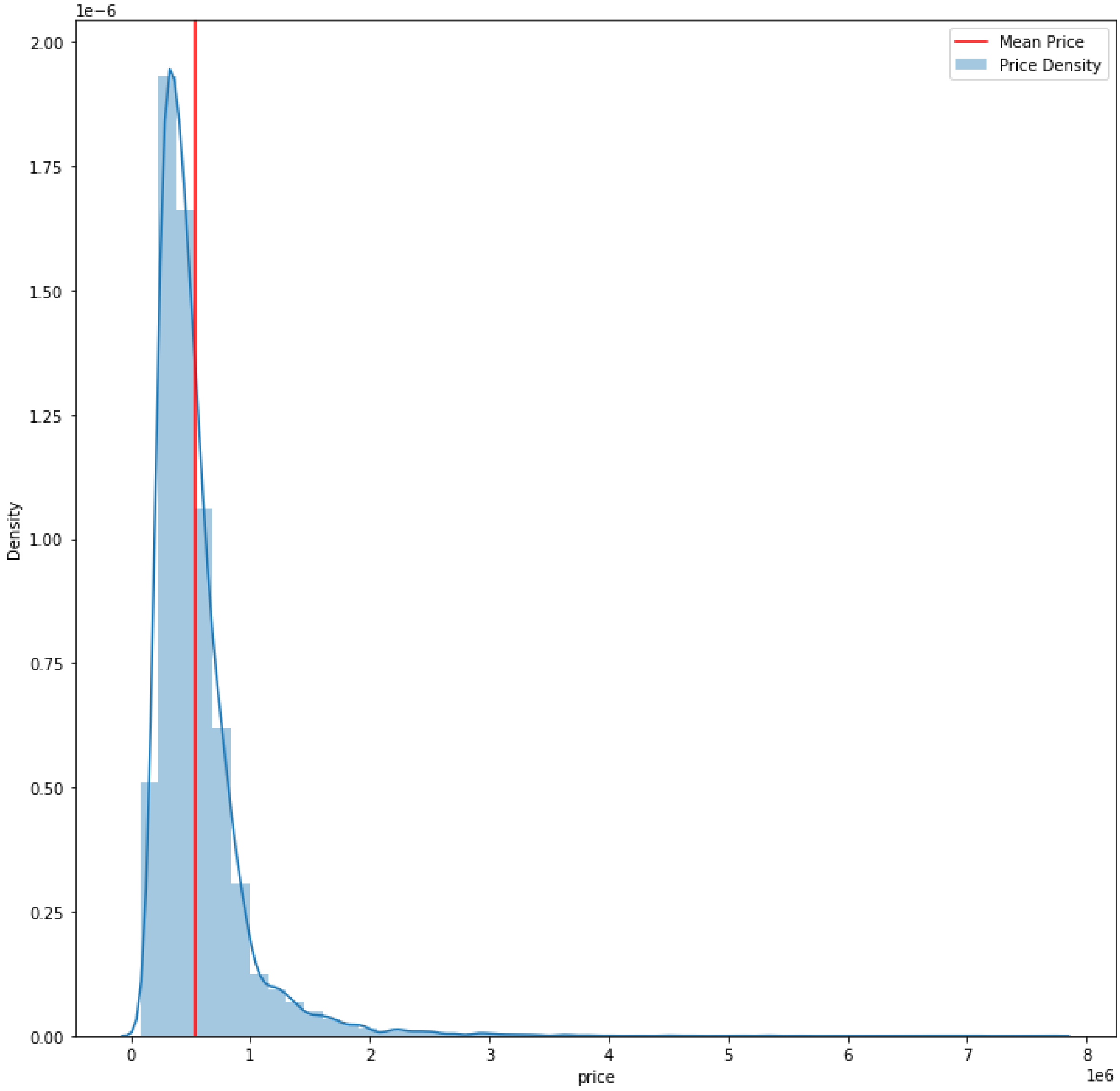
```
  File "<ipython-input-299-622f73eb687e>", line 1
    Answer:This figure shows that the year built doesnot affect the price
                 ^
SyntaxError: invalid syntax
```

In [15]:

```
fig, ax = plt.subplots(figsize=(12,12))
ax = sns.distplot(data_k['price'], label='Price Density')
plt.axvline(price_mean, color='red', label='Mean Price')
plt.legend()
```

Out[15]:

```
<matplotlib.legend.Legend at 0x2a4a589c580>
```



In [300]:

```
data_k.corr()
```

Out[300]:

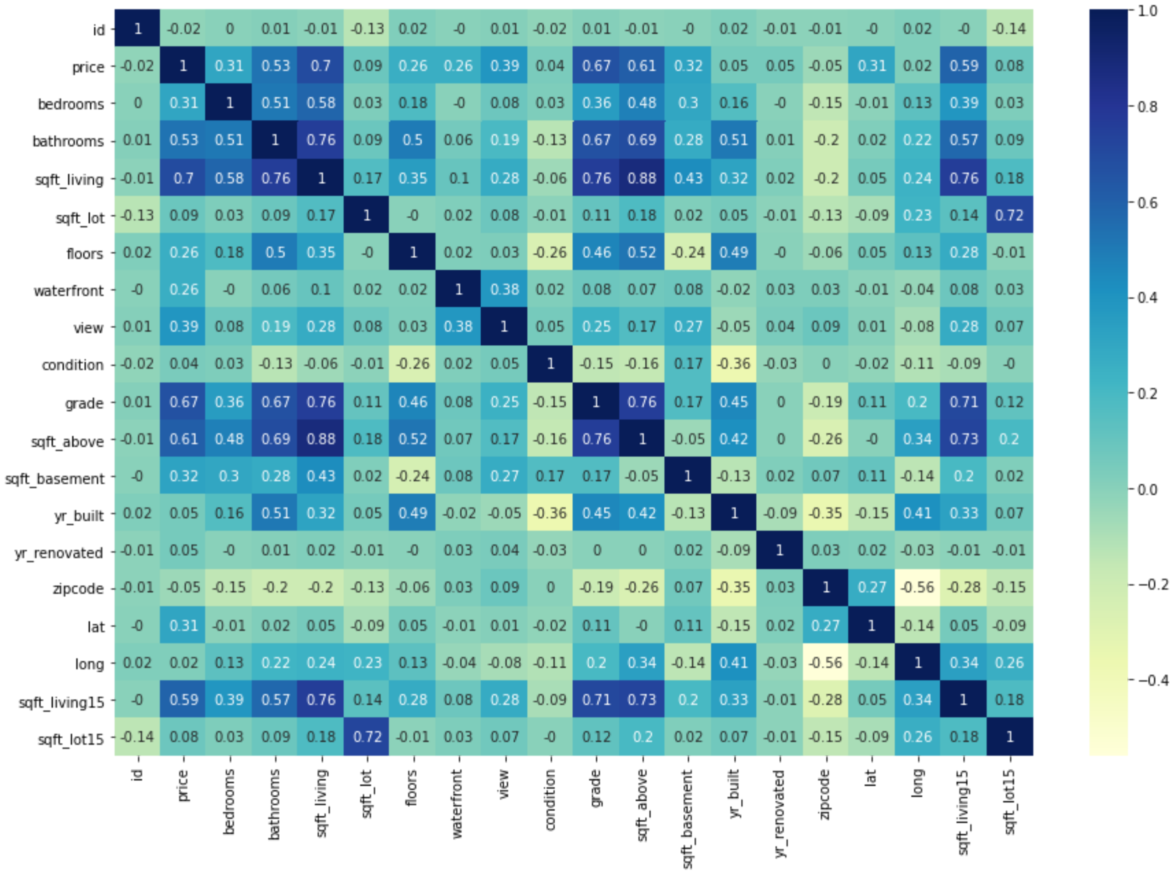
id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	vie
1.000000	-0.016772	0.001150	0.005162	-0.012241	-0.131911	0.018608	-0.003599	0.01177
-0.016772	1.000000	0.308787	0.525906	0.701917	0.089876	0.256804	0.264306	0.39349
0.001150	0.308787	1.000000	0.514508	0.578212	0.032471	0.177944	-0.002127	0.07835
0.005162	0.525906	0.514508	1.000000	0.755758	0.088373	0.502582	0.063629	0.18601
-0.012241	0.701917	0.578212	0.755758	1.000000	0.173453	0.353953	0.104637	0.28171
-0.131911	0.089876	0.032471	0.088373	0.173453	1.000000	-0.004814	0.021459	0.07505
0.018608	0.256804	0.177944	0.502582	0.353953	-0.004814	1.000000	0.020797	0.02841
-0.003599	0.264306	-0.002127	0.063629	0.104637	0.021459	0.020797	1.000000	0.38054
0.011772	0.393497	0.078354	0.186016	0.281715	0.075054	0.028414	0.380543	1.00000
-0.023803	0.036056	0.026496	-0.126479	-0.059445	-0.008830	-0.264075	0.016648	0.04562
0.008188	0.667951	0.356563	0.665838	0.762779	0.114731	0.458794	0.082818	0.24908
-0.010799	0.605368	0.479386	0.686668	0.876448	0.184139	0.523989	0.071778	0.16601
-0.004359	0.321108	0.297229	0.278485	0.428660	0.015031	-0.241866	0.083050	0.27062
0.021617	0.053953	0.155670	0.507173	0.318152	0.052946	0.489193	-0.024487	-0.05445
-0.007119	0.051588	-0.002292	0.012400	0.016479	-0.006170	-0.001467	0.031756	0.04483
-0.008211	-0.053402	-0.154092	-0.204786	-0.199802	-0.129586	-0.059541	0.028923	0.08505
-0.001798	0.306692	-0.009951	0.024280	0.052155	-0.085514	0.049239	-0.012157	0.00632
0.020672	0.022036	0.132054	0.224903	0.241214	0.230227	0.125943	-0.037628	-0.07770
-0.002701	0.585241	0.393406	0.569884	0.756402	0.144763	0.280102	0.083823	0.27892
-0.138557	0.082845	0.030690	0.088303	0.184342	0.718204	-0.010722	0.030658	0.07308

In [16]:

```
#checking the correlation using data visualization
plt.figure(figsize=(15,10))
sns.heatmap(data_k.corr().round(2),annot=True, cmap=('YlGnBu'))
```

Out[16]:

<AxesSubplot:>



But, even our heatmap is difficult to read. We are going to use a pairwise correlation in order to determine which values are highly correlated with our target (price) by using any value greater than 0.70.

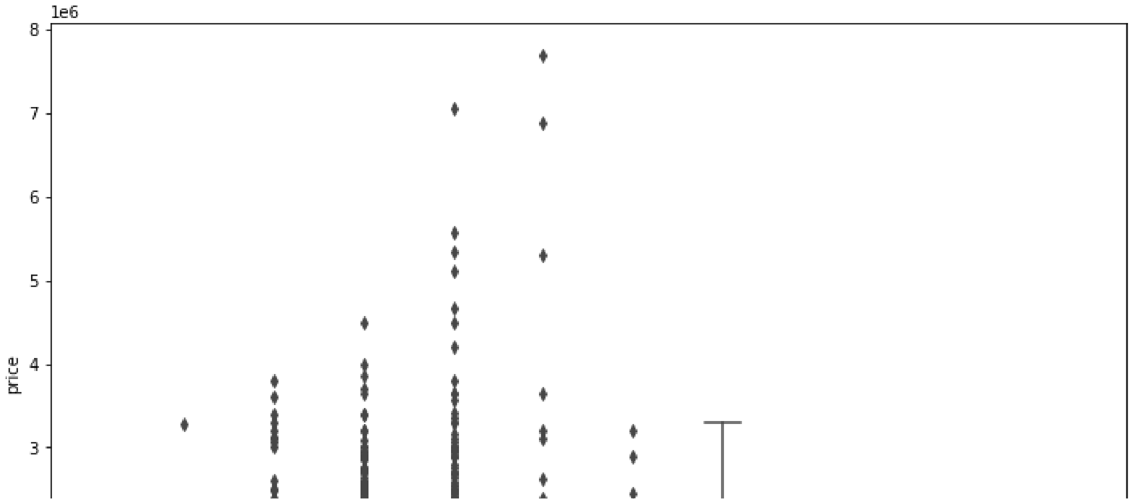
Checking outliers

In [17]:

```
plt.figure(figsize=(12,8))
sns.boxplot(x='bedrooms',y='price',data=data_k)
```

Out[17]:

<AxesSubplot:xlabel='bedrooms', ylabel='price'>

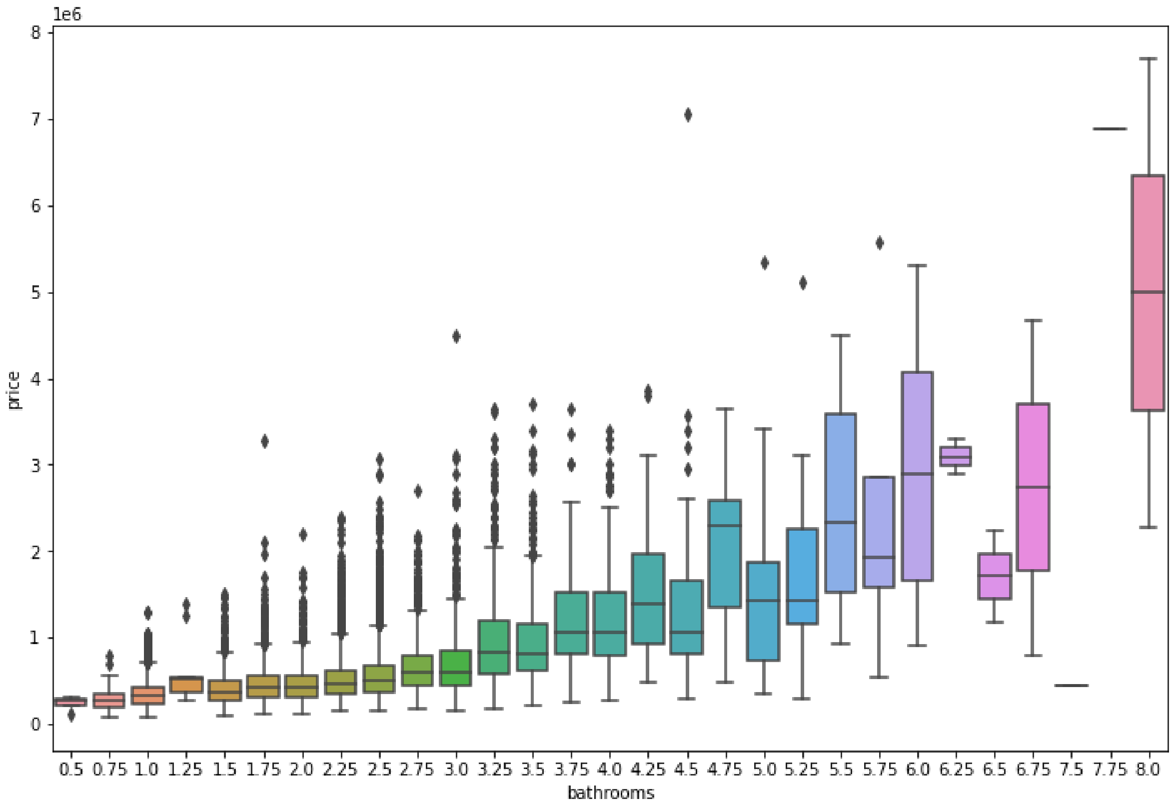


In [18]:

```
plt.figure(figsize=(12,8))
sns.boxplot(x='bathrooms',y='price',data=data_k)
```

Out[18]:

```
<AxesSubplot:xlabel='bathrooms', ylabel='price'>
```

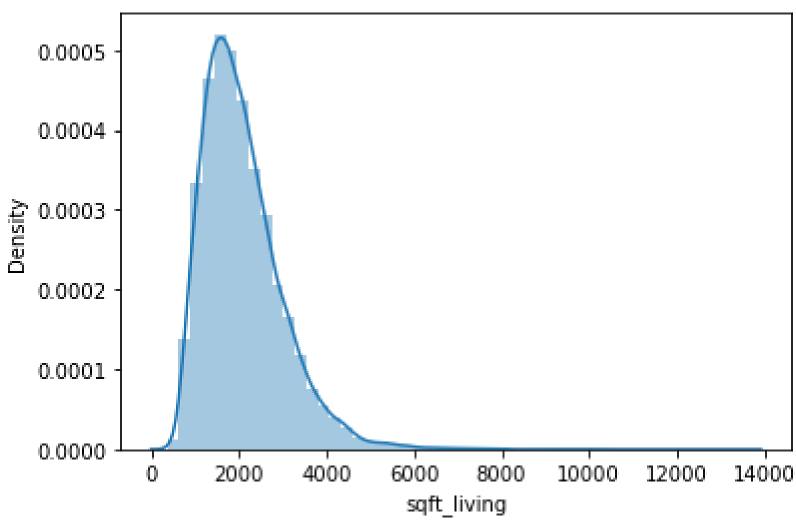


In [19]:

```
sns.distplot(data_k.sqft_living)
```

Out[19]:

```
<AxesSubplot:xlabel='sqft_living', ylabel='Density'>
```

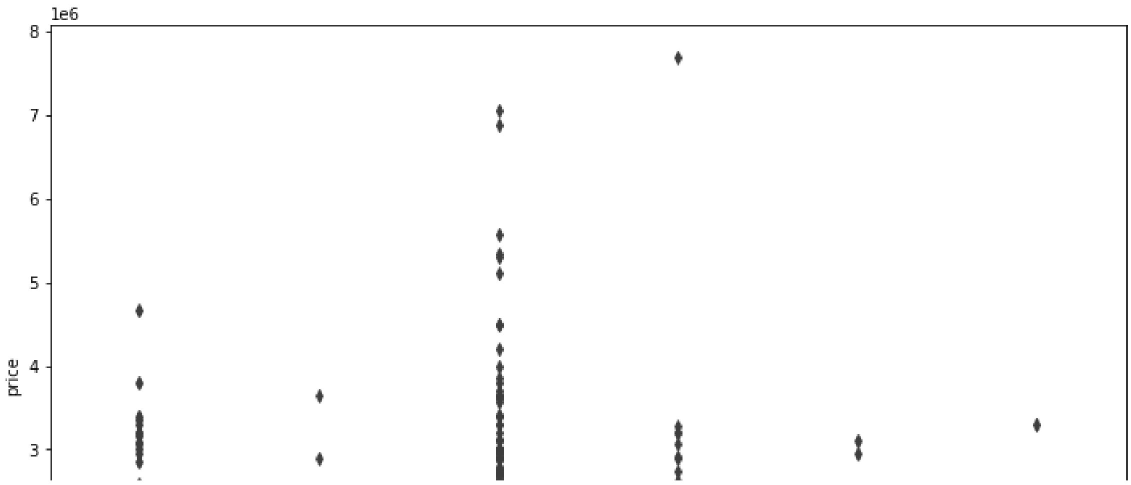


In [20]:

```
plt.figure(figsize=(12,8))
sns.boxplot(x='floors',y='price',data=data_k)
```

Out[20]:

<AxesSubplot:xlabel='floors', ylabel='price'>



In [21]:

```
plt.figure(figsize=(10,7))
sns.scatterplot(data_k['condition'], data_k['price'])
plt.title('House Condition and price', fontsize=15, fontname='silom')
```

Out[21]:

Text(0.5, 1.0, 'House Condition and price')

findfont: Font family ['silom'] not found. Falling back to DejaVu Sans.

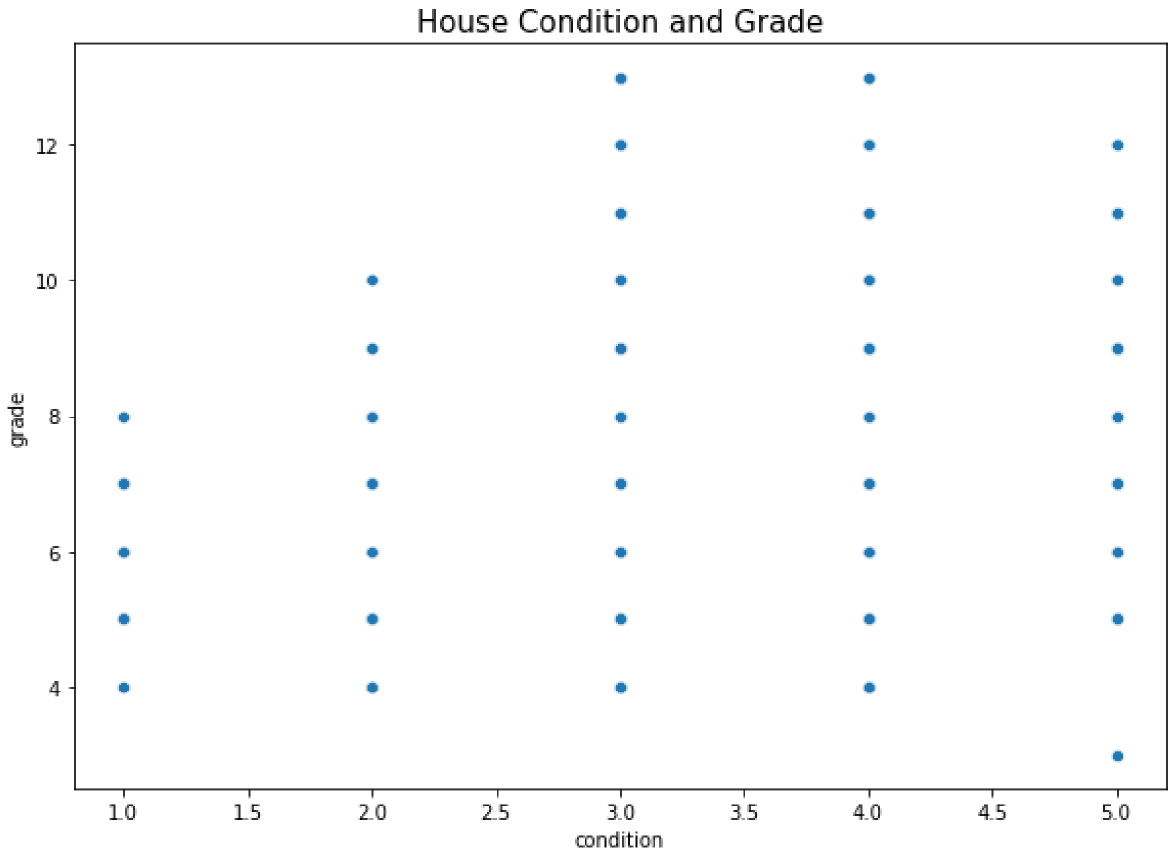
The house condition is not affecting the price according to this figure

In [22]:

```
plt.figure(figsize=(10,7))
sns.scatterplot(data_k['condition'], data_k['grade'])
plt.title('House Condition and Grade', fontsize=15, fontname='silom')
```

Out[22]:

```
Text(0.5, 1.0, 'House Condition and Grade')
```



In [ ]:

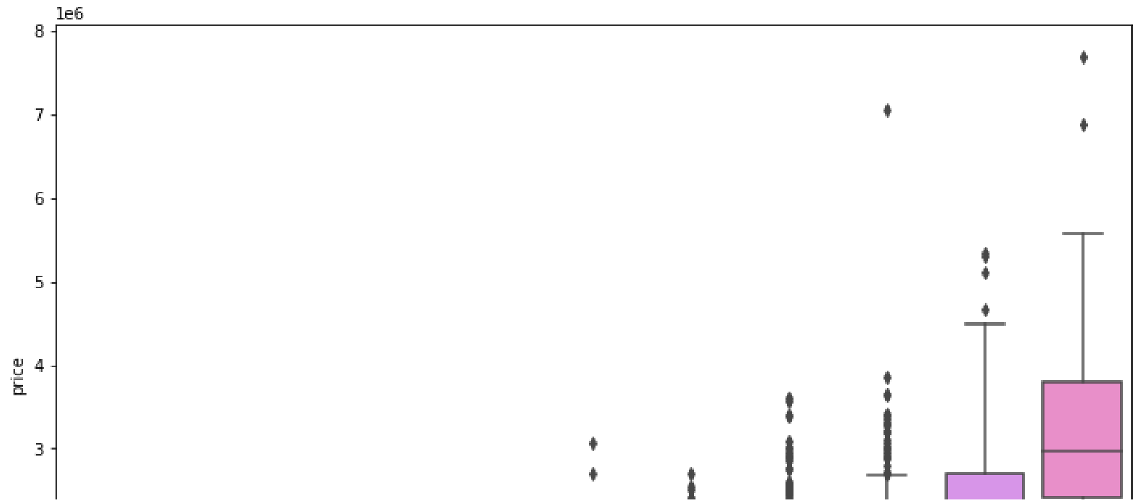
```
In this figure the condition of the house doenot affect the grade
```

In [287]:

```
plt.figure(figsize=(12,8))
sns.boxplot(x='grade',y='price',data=data_k)
```

Out[287]:

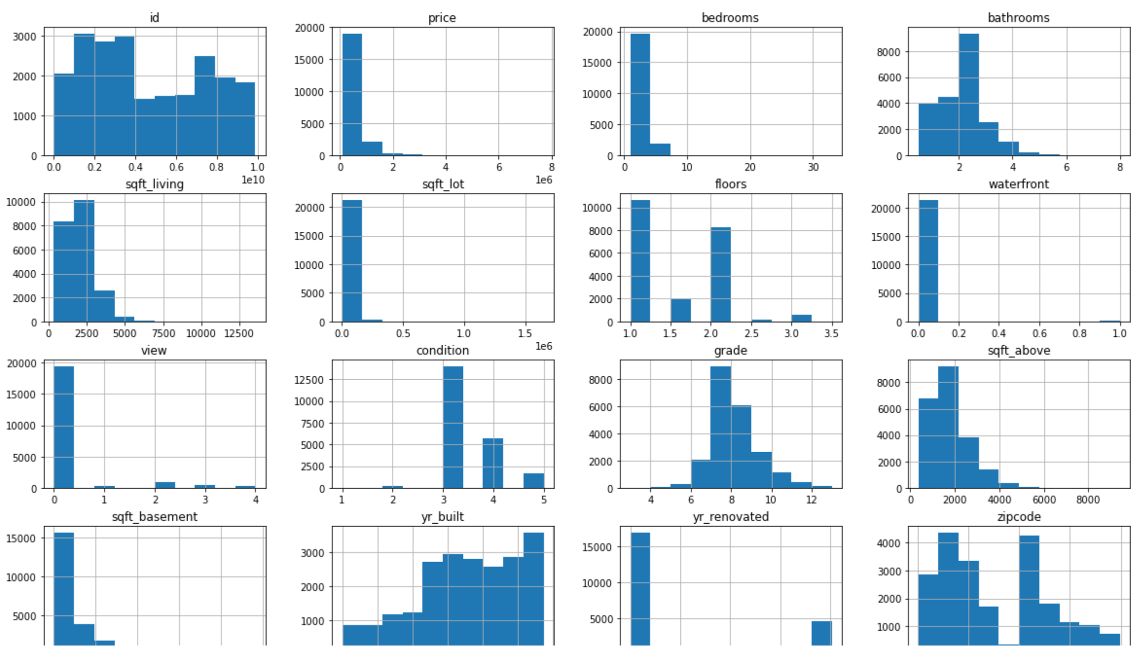
```
<AxesSubplot:xlabel='grade', ylabel='price'>
```



In this figure the house price increase with increase in grade

In [23]:

```
fig = plt.figure(figsize = (20,15))
ax = fig.gca()
data_k.hist(ax = ax);
```



In [ ]:

```
We can see from the box plots and histograms above that many of our continuos data have out
```

In [303]:

```
outcome = 'price'
x_cols = data_k.drop('price',axis=1)
predictors = '+'.join(x_cols)
formula = outcome + '~' + predictors
model = ols(formula=formula, data=data_k).fit()
model.summary()
```

Out[303]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.707
Model:	OLS	**Adj. R-squared:**	0.702
Method:	Least Squares	**F-statistic:**	131.8
Date:	Sun, 12 Feb 2023	**Prob (F-statistic):**	0.00
Time:	22:07:35	**Log-Likelihood:**	-2.9412e+05
No. Observations:	21597	**AIC:**	5.890e+05
Df Residuals:	21207	**BIC:**	5.921e+05
Df Model:	389		
Covariance Type:	nonrobust		

In [307]:

```
new_data= data_k.drop(['id','view','date',], axis=1)
new_data
```

Out[307]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition	grade
0	221900.0	3	1.00	1180	5650	1.0	0.0	3	7
1	538000.0	3	2.25	2570	7242	2.0	0.0	3	7
2	180000.0	2	1.00	770	10000	1.0	0.0	3	6
3	604000.0	4	3.00	1960	5000	1.0	0.0	5	7
4	510000.0	3	2.00	1680	8080	1.0	0.0	3	8
...	...	...	...	...	...	...	...	...	...
1592	360000.0	3	2.50	1530	1131	3.0	0.0	3	8
1593	400000.0	4	2.50	2310	5813	2.0	0.0	3	8
1594	402101.0	2	0.75	1020	1350	2.0	0.0	3	7
1595	400000.0	3	2.50	1600	2388	2.0	0.0	3	8
1596	325000.0	2	0.75	1020	1076	2.0	0.0	3	7

597 rows × 18 columns

In [309]:

```
new_data['waterfront'].fillna(0, inplace=True)
```

In [310]:

```
#Now, let's place our continuous and categorical data into a separate features
cont_data = new_data[['price', 'sqft_living', 'sqft_lot', 'sqft_basement', 'yr_built', 'lat
]]
cat_data = new_data[['bedrooms','bathrooms','floors','waterfront', 'condition', 'grade', 'z
```

Remove outliners

In [311]:

```
count = 0
bath_outliers = []
mean = np.mean(cat_data['bathrooms'])
max_distance = np.std(cat_data['bathrooms']) * 3

for idx, row in cat_data['bathrooms'].T.iteritems():
    if abs(row-mean) >= max_distance:
        count += 1
        cat_data.drop(idx, inplace=True)
count
```

Out[311]:

187

In [312]:

```
count = 0
bed_outliers = []
mean = np.mean(cat_data['bedrooms'])
max_distance = np.std(cat_data['bedrooms']) * 3

for idx, row in cat_data['bedrooms'].T.iteritems():
    if abs(row-mean) >= max_distance:
        count += 1
        cat_data.drop(idx,inplace=True)
count
```

Out[312]:

47

In [313]:

```
cont_data.shape
```

Out[313]:

(21597, 11)

In [314]:

```
z_cont=np.abs(stats.zscore(cont_data))
```

In [315]:

```
cont_data1 = cont_data[(z_cont < 3).all(axis=1)]
```

In [316]:

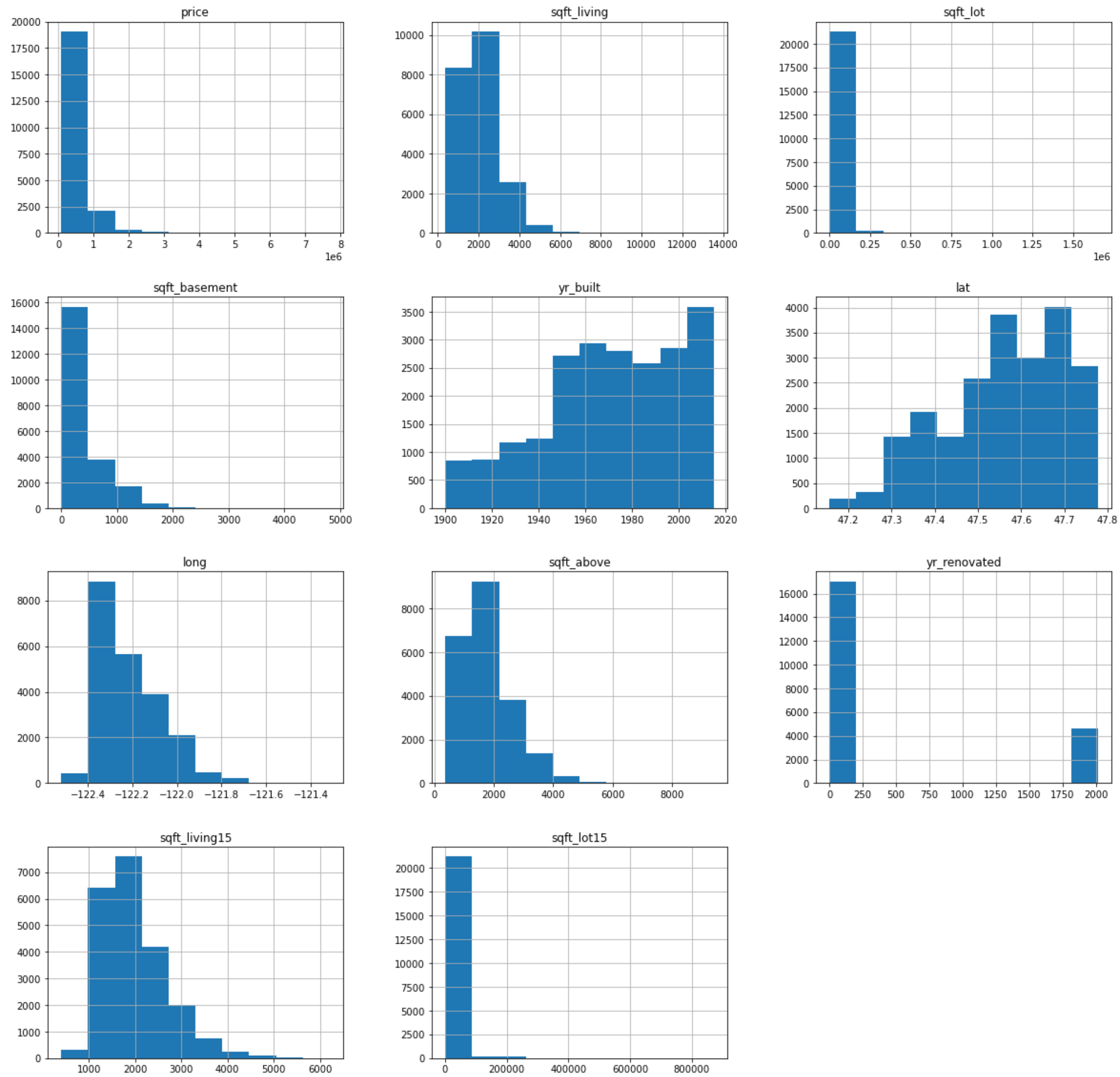
```
cont_data1.shape
```

Out[316]:

```
(20135, 11)
```

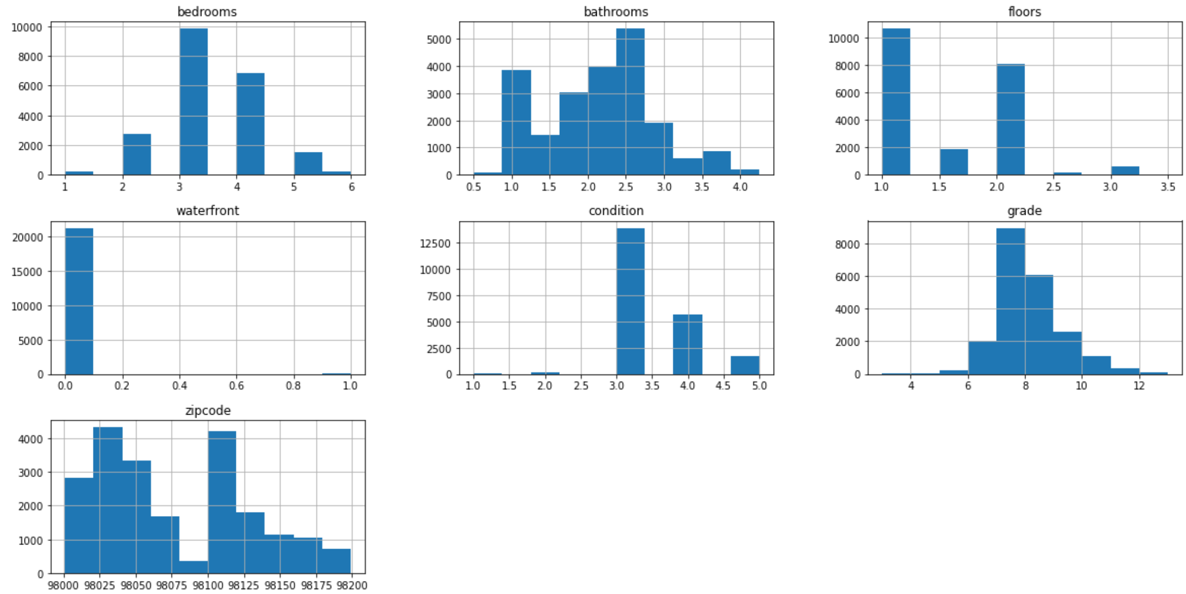
In [317]:

```
# Analysising data with histogram
cont_data.hist(figsize=(20,20));
```



In [107]:

```python
# Analysising data with histogram
cat_data.hist(figsize=(20,10));
```



In [289]:

```python
plt.figure(figsize=(12,8))
sns.boxplot(x='sqft_living',y='price',data=cont_data)
```

Out[289]:

```
<AxesSubplot:xlabel='sqft_living', ylabel='price'>
```

In [318]:

```python
#Let's make a continuous data feature and a feature for our target variable
target = cont_data['price']
cont_feat = cont_data.drop('price', axis=1)
```

In [319]:

```
cont_feat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   sqft_living    21597 non-null  int64  
 1   sqft_lot       21597 non-null  int64  
 2   sqft_basement  21597 non-null  float64
 3   yr_built       21597 non-null  int64  
 4   lat            21597 non-null  float64
 5   long           21597 non-null  float64
 6   sqft_above     21597 non-null  int64  
 7   yr_renovated   21597 non-null  float64
 8   sqft_living15  21597 non-null  int64  
 9   sqft_lot15     21597 non-null  int64  
dtypes: float64(4), int64(6)
memory usage: 1.6 MB
```

In [320]:

```
target
```

Out[320]:

```
0        221900.0
1        538000.0
2        180000.0
3        604000.0
4        510000.0
           ...   
21592    360000.0
21593    400000.0
21594    402101.0
21595    400000.0
21596    325000.0
Name: price, Length: 21597, dtype: float64
```

In [321]:

```
dummy_1= pd.get_dummies(data=cat_data, columns=['bedrooms'],prefix='bed', drop_first=True)
dummy_2= pd.get_dummies(data=cat_data, columns=['condition'],prefix='condi', drop_first=Tru
dummy_3= pd.get_dummies(data=cat_data, columns=['grade'],prefix='grd', drop_first=True)
dummy_4= pd.get_dummies(data=cat_data, columns=['zipcode'],prefix='zip_c', drop_first=True)
#dummy_5= pd.get_dummies(data=cat_data, columns=['waterfront'],prefix='wf', drop_first=True
#dummy_6= pd.get_dummies(data=cat_data, columns=['floors'],prefix='flr', drop_first=True)
#dummy_7= pd.get_dummies(data=cat_data, columns=['bathrooms'],prefix='bath', drop_first=Tru
```

In [322]:

```
#Let's create one feature containing all of our dummies
cat_dummies = pd.concat([dummy_1,dummy_2,dummy_3,dummy_4],axis=1)
cat_dummies
```

	bathrooms	floors	waterfront	condition	grade	zipcode	bed_2	bed_3	bed_4	bed_5	...	zip_c_981
0	1.00	1.0	0.0	3	7	98178	0	1	0	0	...	
1	2.25	2.0	0.0	3	7	98125	0	1	0	0	...	
2	1.00	1.0	0.0	3	6	98028	1	0	0	0	...	
3	3.00	1.0	0.0	5	7	98136	0	0	1	0	...	
4	2.00	1.0	0.0	3	8	98074	0	1	0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
21592	2.50	3.0	0.0	3	8	98103	0	1	0	0	...	
21593	2.50	2.0	0.0	3	8	98146	0	0	1	0	...	
21594	0.75	2.0	0.0	3	7	98144	1	0	0	0	...	
21595	2.50	2.0	0.0	3	8	98027	0	1	0	0	...	
21596	0.75	2.0	0.0	3	7	98144	1	0	0	0	...	

In [323]:

```
d1= pd.concat([target,cont_feat,cat_dummies], axis=1)
d1.dropna(how='any',inplace=True)
```

In [324]:

```
outcome = 'price'
x_cols = d1.drop('price',axis=1)
predictors = '+'.join(x_cols)
formula = outcome + '~' + predictors
model = ols(formula=formula, data=d1).fit()
model.summary()
```

Out[324]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.822
Model:	OLS	**Adj. R-squared:**	0.821
Method:	Least Squares	**F-statistic:**	969.6
Date:	Sun, 12 Feb 2023	**Prob (F-statistic):**	0.00
Time:	22:17:30	**Log-Likelihood:**	-2.8333e+05
No. Observations:	21363	**AIC:**	5.669e+05
Df Residuals:	21261	**BIC:**	5.677e+05
Df Model:	101		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	-1.11e+04	827.588	-13.414	0.000	-1.27e+04	-9478.875
sqft_living	92.5485	13.153	7.036	0.000	66.768	118.329
sqft_lot	0.2927	0.034	8.699	0.000	0.227	0.359
sqft_basement	24.3504	13.028	1.869	0.062	-1.185	49.886
yr_built	-645.8462	55.246	-11.690	0.000	-754.132	-537.560
lat	1.414e+05	5.54e+04	2.553	0.011	3.28e+04	2.5e+05
long	-1.861e+05	3.98e+04	-4.673	0.000	-2.64e+05	-1.08e+05
sqft_above	53.1626	13.180	4.034	0.000	27.329	78.996
yr_renovated	6.9696	1.200	5.809	0.000	4.618	9.321
sqft_living15	38.7259	2.565	15.098	0.000	33.699	43.753
sqft_lot15	-0.0845	0.054	-1.568	0.117	-0.190	0.021
bathrooms[0]	6819.1897	597.974	11.404	0.000	5647.115	7991.265
bathrooms[1]	6819.1897	597.974	11.404	0.000	5647.115	7991.265
bathrooms[2]	6819.1897	597.974	11.404	0.000	5647.115	7991.265
bathrooms[3]	6819.1897	597.974	11.404	0.000	5647.115	7991.265
floors[0]	-5276.1961	703.622	-7.499	0.000	-6655.348	-3897.044
floors[1]	-5276.1961	703.622	-7.499	0.000	-6655.348	-3897.044
floors[2]	-5276.1961	703.622	-7.499	0.000	-6655.348	-3897.044
floors[3]	-5276.1961	703.622	-7.499	0.000	-6655.348	-3897.044
waterfront[0]	1.824e+05	3149.220	57.905	0.000	1.76e+05	1.89e+05

waterfront[1]	1.824e+05	3149.220	57.905	0.000	1.76e+05	1.89e+05
waterfront[2]	1.824e+05	3149.220	57.905	0.000	1.76e+05	1.89e+05
waterfront[3]	1.824e+05	3149.220	57.905	0.000	1.76e+05	1.89e+05
condition[0]	1.374e+04	2884.312	4.762	0.000	8082.786	1.94e+04
condition[1]	1.374e+04	2884.312	4.762	0.000	8082.786	1.94e+04
condition[2]	1.374e+04	2884.312	4.762	0.000	8082.786	1.94e+04
grade[0]	1.729e+04	6704.427	2.579	0.010	4146.463	3.04e+04
grade[1]	1.729e+04	6704.427	2.579	0.010	4146.463	3.04e+04
grade[2]	1.729e+04	6704.427	2.579	0.010	4146.463	3.04e+04
zipcode[0]	-96.6712	18.290	-5.286	0.000	-132.520	-60.822
zipcode[1]	-96.6712	18.290	-5.286	0.000	-132.520	-60.822
zipcode[2]	-96.6712	18.290	-5.286	0.000	-132.520	-60.822
bed_2	1.102e+04	7955.804	1.385	0.166	-4573.723	2.66e+04
bed_3	1.912e+04	5269.881	3.628	0.000	8791.674	2.95e+04
bed_4	6408.2021	3608.259	1.776	0.076	-664.258	1.35e+04
bed_5	866.7682	4297.550	0.202	0.840	-7556.754	9290.290
bed_6	-1.265e+04	6417.575	-1.972	0.049	-2.52e+04	-74.548
bedrooms[0]	-2414.4030	1015.512	-2.378	0.017	-4404.884	-423.922
bedrooms[1]	-2414.4030	1015.512	-2.378	0.017	-4404.884	-423.922
bedrooms[2]	-2414.4030	1015.512	-2.378	0.017	-4404.884	-423.922
condi_2	3.659e+04	2.03e+04	1.800	0.072	-3248.890	7.64e+04
condi_3	1.269e+04	9132.466	1.390	0.165	-5206.924	3.06e+04
condi_4	-5780.6609	2222.139	-2.601	0.009	-1.01e+04	-1425.101
condi_5	-4948.6116	8659.385	-0.571	0.568	-2.19e+04	1.2e+04
grd_4	-1.475e+05	1.23e+05	-1.199	0.231	-3.89e+05	9.36e+04
grd_5	-2.439e+05	1.01e+05	-2.425	0.015	-4.41e+05	-4.68e+04
grd_6	-2.99e+05	8.04e+04	-3.718	0.000	-4.57e+05	-1.41e+05
grd_7	-3.424e+05	6.06e+04	-5.649	0.000	-4.61e+05	-2.24e+05
grd_8	-3.592e+05	4.11e+04	-8.735	0.000	-4.4e+05	-2.79e+05
grd_9	-3.234e+05	2.27e+04	-14.231	0.000	-3.68e+05	-2.79e+05
grd_10	-2.499e+05	1.24e+04	-20.138	0.000	-2.74e+05	-2.26e+05
grd_11	-9.688e+04	2.48e+04	-3.901	0.000	-1.46e+05	-4.82e+04
grd_12	1.869e+05	4.48e+04	4.174	0.000	9.91e+04	2.75e+05
grd_13	7.532e+05	7.59e+04	9.923	0.000	6.04e+05	9.02e+05
zip_c_98002	2.355e+04	1.26e+04	1.862	0.063	-1242.622	4.83e+04
zip_c_98003	-9671.8789	1.12e+04	-0.860	0.390	-3.17e+04	1.24e+04
zip_c_98004	7.013e+05	2.05e+04	34.175	0.000	6.61e+05	7.42e+05
zip_c_98005	2.666e+05	2.19e+04	12.181	0.000	2.24e+05	3.09e+05
zip_c_98006	2.429e+05	1.79e+04	13.538	0.000	2.08e+05	2.78e+05
zip_c_98007	2.203e+05	2.26e+04	9.731	0.000	1.76e+05	2.65e+05

zip_c_98008	2.5e+05	2.15e+04	11.646	0.000	2.08e+05	2.92e+05
zip_c_98010	1.126e+05	1.95e+04	5.780	0.000	7.44e+04	1.51e+05
zip_c_98011	7.668e+04	2.78e+04	2.758	0.006	2.22e+04	1.31e+05
zip_c_98014	1.226e+05	3.09e+04	3.966	0.000	6.2e+04	1.83e+05
zip_c_98019	9.223e+04	3.03e+04	3.042	0.002	3.28e+04	1.52e+05
zip_c_98022	8.253e+04	1.75e+04	4.707	0.000	4.82e+04	1.17e+05
zip_c_98023	-4.345e+04	1e+04	-4.323	0.000	-6.31e+04	-2.37e+04
zip_c_98024	1.843e+05	2.75e+04	6.703	0.000	1.3e+05	2.38e+05
zip_c_98027	1.758e+05	1.87e+04	9.409	0.000	1.39e+05	2.12e+05
zip_c_98028	7.822e+04	2.67e+04	2.934	0.003	2.6e+04	1.3e+05
zip_c_98029	2.362e+05	2.13e+04	11.069	0.000	1.94e+05	2.78e+05
zip_c_98030	1.705e+04	1.27e+04	1.347	0.178	-7759.708	4.19e+04
zip_c_98031	2.181e+04	1.31e+04	1.665	0.096	-3871.639	4.75e+04
zip_c_98032	638.5251	1.48e+04	0.043	0.966	-2.85e+04	2.97e+04
zip_c_98033	3.278e+05	2.29e+04	14.292	0.000	2.83e+05	3.73e+05
zip_c_98034	1.651e+05	2.45e+04	6.739	0.000	1.17e+05	2.13e+05
zip_c_98038	7.731e+04	1.5e+04	5.144	0.000	4.79e+04	1.07e+05
zip_c_98039	1.061e+06	2.9e+04	36.561	0.000	1e+06	1.12e+06
zip_c_98040	4.739e+05	1.78e+04	26.551	0.000	4.39e+05	5.09e+05
zip_c_98042	3.576e+04	1.28e+04	2.793	0.005	1.07e+04	6.09e+04
zip_c_98045	1.887e+05	2.71e+04	6.953	0.000	1.35e+05	2.42e+05
zip_c_98052	2.176e+05	2.35e+04	9.245	0.000	1.72e+05	2.64e+05
zip_c_98053	2.131e+05	2.56e+04	8.312	0.000	1.63e+05	2.63e+05
zip_c_98055	4.886e+04	1.44e+04	3.382	0.001	2.05e+04	7.72e+04
zip_c_98056	9.577e+04	1.56e+04	6.144	0.000	6.52e+04	1.26e+05
zip_c_98058	4.578e+04	1.42e+04	3.227	0.001	1.8e+04	7.36e+04
zip_c_98059	9.06e+04	1.57e+04	5.765	0.000	5.98e+04	1.21e+05
zip_c_98065	1.62e+05	2.53e+04	6.408	0.000	1.12e+05	2.12e+05
zip_c_98070	8584.9938	1.64e+04	0.523	0.601	-2.36e+04	4.08e+04
zip_c_98072	1.294e+05	2.73e+04	4.741	0.000	7.59e+04	1.83e+05
zip_c_98074	1.815e+05	2.3e+04	7.903	0.000	1.36e+05	2.26e+05
zip_c_98075	1.905e+05	2.24e+04	8.495	0.000	1.47e+05	2.34e+05
zip_c_98077	9.471e+04	2.88e+04	3.286	0.001	3.82e+04	1.51e+05
zip_c_98092	1.762e+04	1.37e+04	1.283	0.200	-9302.858	4.46e+04
zip_c_98102	4.238e+05	2.26e+04	18.767	0.000	3.8e+05	4.68e+05
zip_c_98103	3.053e+05	2.04e+04	14.986	0.000	2.65e+05	3.45e+05
zip_c_98105	4.448e+05	2.16e+04	20.587	0.000	4.02e+05	4.87e+05
zip_c_98106	1.132e+05	1.47e+04	7.692	0.000	8.44e+04	1.42e+05
zip_c_98107	3.039e+05	2.08e+04	14.589	0.000	2.63e+05	3.45e+05
zip_c_98108	1.075e+05	1.71e+04	6.284	0.000	7.4e+04	1.41e+05

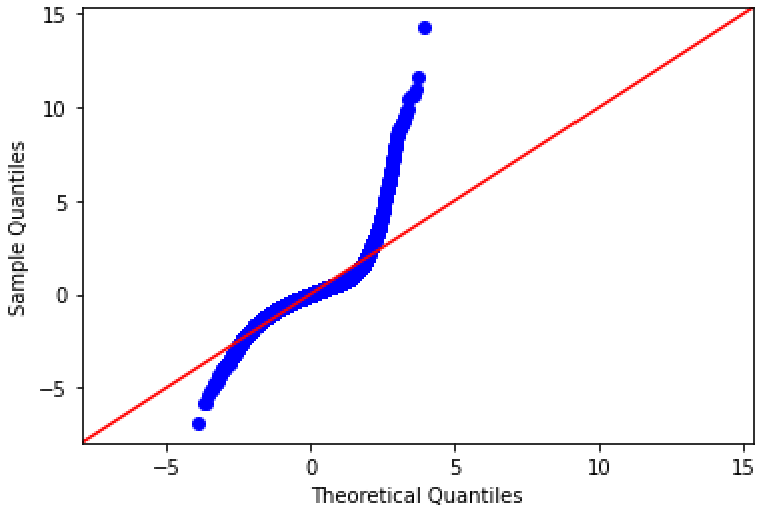
zip_c_98109	4.667e+05	2.2e+04	21.177	0.000	4.24e+05	5.1e+05
zip_c_98112	5.745e+05	1.97e+04	29.138	0.000	5.36e+05	6.13e+05
zip_c_98115	3.118e+05	2.1e+04	14.851	0.000	2.71e+05	3.53e+05
zip_c_98116	2.898e+05	1.6e+04	18.125	0.000	2.58e+05	3.21e+05
zip_c_98117	2.811e+05	2.05e+04	13.732	0.000	2.41e+05	3.21e+05
zip_c_98118	1.695e+05	1.53e+04	11.068	0.000	1.39e+05	1.99e+05
zip_c_98119	4.552e+05	2.03e+04	22.399	0.000	4.15e+05	4.95e+05
zip_c_98122	3.344e+05	1.86e+04	17.986	0.000	2.98e+05	3.71e+05
zip_c_98125	1.836e+05	2.27e+04	8.097	0.000	1.39e+05	2.28e+05
zip_c_98126	1.899e+05	1.47e+04	12.956	0.000	1.61e+05	2.19e+05
zip_c_98133	1.273e+05	2.29e+04	5.563	0.000	8.24e+04	1.72e+05
zip_c_98136	2.596e+05	1.49e+04	17.391	0.000	2.3e+05	2.89e+05
zip_c_98144	2.884e+05	1.72e+04	16.734	0.000	2.55e+05	3.22e+05
zip_c_98146	1.176e+05	1.37e+04	8.580	0.000	9.08e+04	1.45e+05
zip_c_98148	8.107e+04	2.09e+04	3.886	0.000	4.02e+04	1.22e+05
zip_c_98155	1.215e+05	2.41e+04	5.037	0.000	7.42e+04	1.69e+05
zip_c_98166	9.899e+04	1.33e+04	7.469	0.000	7.3e+04	1.25e+05
zip_c_98168	7.003e+04	1.46e+04	4.806	0.000	4.15e+04	9.86e+04
zip_c_98177	2.2e+05	2.33e+04	9.446	0.000	1.74e+05	2.66e+05
zip_c_98178	7.61e+04	1.64e+04	4.644	0.000	4.4e+04	1.08e+05
zip_c_98188	7.111e+04	1.7e+04	4.173	0.000	3.77e+04	1.05e+05
zip_c_98198	6.279e+04	1.38e+04	4.566	0.000	3.58e+04	8.98e+04
zip_c_98199	3.825e+05	1.83e+04	20.856	0.000	3.47e+05	4.18e+05

Omnibus:	12606.338	**Durbin-Watson:**	1.978
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	398788.018
Skew:	2.296	**Prob(JB):**	0.00
Kurtosis:	23.662	**Cond. No.**	7.20e+18

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.21e-23. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [325]:

```
# Q-Q plot and Regression plot
model=smf.ols(formula,data=d1).fit()
fig = sm.graphics.qqplot(model.resid, line='45',fit=True);
#fig.savefig('Basic_model_qqplot')
sm.graphics.plot_regress_exog(model, 'sqft_living', fig=plt.figure(figsize=(12,8)));
```



Conclusion:

The base model shows that sqft_living(independent variable) has high correlation with price(dependent variable). In Basic linear model high R-squared value (0.82) and low P-value(0) shows there is a relationship between variables. Q-Q data plot is not normally distributed and scatter plots are very homoscedastic.

Iteration1

Log transformation

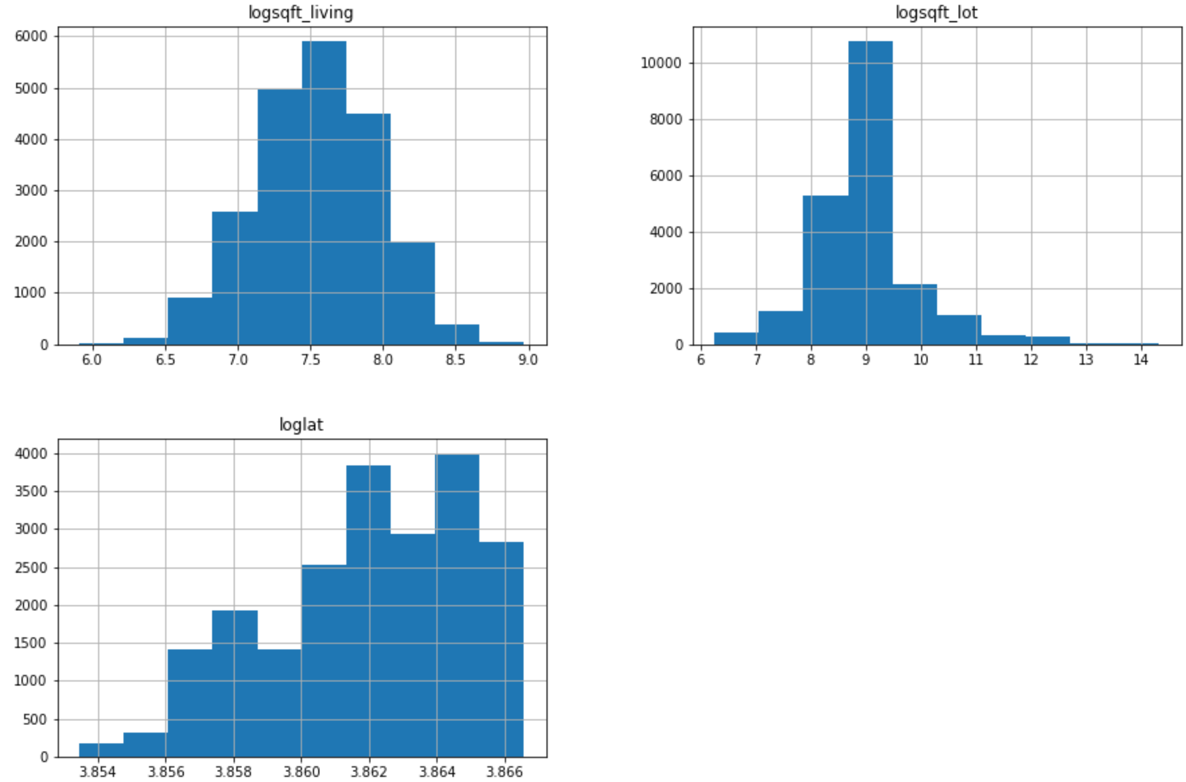
In [326]:

```
n.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21597 entries, 0 to 21596
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   bedrooms     21363 non-null  float64
 1   bathrooms    21363 non-null  float64
 2   floors       21363 non-null  float64
 3   waterfront   21363 non-null  float64
 4   condition    21363 non-null  float64
 5   grade        21363 non-null  float64
 6   zipcode      21363 non-null  float64
 7   price        21597 non-null  float64
 8   sqft_living  21597 non-null  int64  
 9   sqft_lot     21597 non-null  int64  
 10  yr_built     21597 non-null  int64  
 11  lat          21597 non-null  float64
 12  long         21597 non-null  float64
dtypes: float64(10), int64(3)
memory usage: 2.9 MB
```

In [346]:

```
data_log = pd.DataFrame([])
data_log['logsqft_living'] = np.log(d1['sqft_living'])
data_log['logsqft_lot'] = np.log(d1['sqft_lot'])
#data_log['logsqft_basement'] = np.log(d1['sqft_basement'])
data_log['loglat'] = np.log(d1['lat'])
#data_log['loglong'] = np.log(d1['long'])
data_log.hist(figsize  = [15,10 ]);
```



In [347]:

```
pp = d1['price']
logliving = np.log(d1['sqft_living'])
loglot = np.log(d1['sqft_lot'])
loglat = np.log(d1['lat'])
#loglong = np.log(d1['long'])

scaled_pp = (pp-min(pp))/(max(pp)-min(pp))
scaled_lot = (loglot-np.mean(loglot))/np.sqrt(np.var(loglot))
scaled_lat = (loglat-np.mean(loglat))/(max(loglat)-min(loglat))
scaled_living=(logliving-np.mean(logliving))/np.sqrt(np.var(logliving))

data_fin = pd.DataFrame([])
data_fin['pp'] = scaled_pp
data_fin['lot'] = scaled_lot
data_fin['lat'] = scaled_lat
data_fin['living'] = scaled_living
```

In [348]:

```
data_fin
```

Out[348]:

	pp	lot	lat	living
0	0.032616	-0.382321	-0.077514	-1.126726
1	0.104261	-0.105936	0.258885	0.746830
2	0.023119	0.253334	0.285919	-2.154206
3	0.119220	-0.518395	-0.062088	0.094637
4	0.097915	0.015972	0.091993	-0.276397
...	...	...	...	...
21592	0.063917	-2.173235	0.224160	-0.501510
21593	0.072983	-0.350656	-0.078317	0.490108
21594	0.073459	-1.976166	0.056069	-1.477448
21595	0.072983	-1.341154	-0.040081	-0.393833
21596	0.055984	-2.228739	0.055588	-1.477448

21363 rows × 4 columns

In [349]:

```
outcome = 'pp'
x_cols = data_fin.drop('pp',axis=1)
predictors = '+'.join(x_cols)
formula = outcome + '~' + predictors
model = ols(formula=formula, data=data_fin).fit()
model.summary()
```

Out[349]:

OLS Regression Results

Dep. Variable:	pp	**R-squared:**	0.469
Model:	OLS	**Adj. R-squared:**	0.469
Method:	Least Squares	**F-statistic:**	6292.
Date:	Sun, 12 Feb 2023	**Prob (F-statistic):**	0.00
Time:	22:23:01	**Log-Likelihood:**	31874.
No. Observations:	21363	**AIC:**	-6.374e+04
Df Residuals:	21359	**BIC:**	-6.371e+04
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.1023	0.000	274.582	0.000	0.102	0.103
lot	0.0003	0.000	0.872	0.383	-0.000	0.001
lat	0.1044	0.002	61.560	0.000	0.101	0.108
living	0.0447	0.000	113.442	0.000	0.044	0.045

Omnibus:	15259.012	**Durbin-Watson:**	1.985
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	396659.314
Skew:	3.137	**Prob(JB):**	0.00
Kurtosis:	23.156	**Cond. No.**	5.23

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

In [350]:

```
r_lot = data_fin.drop('lot',axis=1)
r_lot
```

Out[350]:

	pp	lat	living
0	0.032616	-0.077514	-1.126726
1	0.104261	0.258885	0.746830
2	0.023119	0.285919	-2.154206
3	0.119220	-0.062088	0.094637
4	0.097915	0.091993	-0.276397
...	...	...	...
21592	0.063917	0.224160	-0.501510
21593	0.072983	-0.078317	0.490108
21594	0.073459	0.056069	-1.477448
21595	0.072983	-0.040081	-0.393833
21596	0.055984	0.055588	-1.477448

21363 rows × 3 columns

In [351]:

```
outcome = 'pp'
x_cols = r_lot.drop('pp',axis=1)
predictors = '+'.join(x_cols)
formula = outcome + '~' + predictors
model = ols(formula=formula, data=r_lot).fit()
model.summary()
```

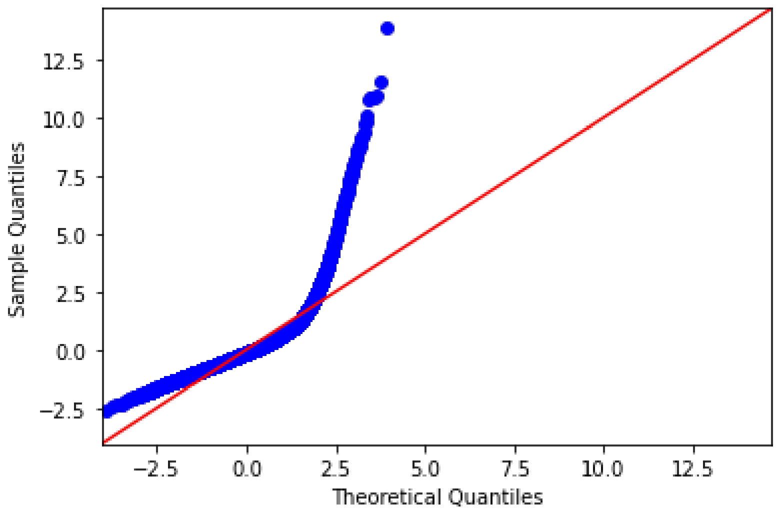
Out[351]:

OLS Regression Results

Dep. Variable:	pp	**R-squared:**	0.469
Model:	OLS	**Adj. R-squared:**	0.469
Method:	Least Squares	**F-statistic:**	9437.
Date:	Sun, 12 Feb 2023	**Prob (F-statistic):**	0.00
Time:	22:26:31	**Log-Likelihood:**	31873.
No. Observations:	21363	**AIC:**	-6.374e+04
Df Residuals:	21360	**BIC:**	-6.372e+04
Df Model:	2		
Covariance Type:	nonrobust		

In [353]:

```
# Q-Q plot and Regression plot
model=smf.ols(formula,data=r_lot).fit()
fig = sm.graphics.qqplot(model.resid, line='45',fit=True);
#fig.savefig('Basic_model_qqplot')
sm.graphics.plot_regress_exog(model, 'living', fig=plt.figure(figsize=(12,8)));
```



Conclusion

The Iteration 1 shows that sqft_living(independent variable) has high correlation with price(dependent variable). In this model high R-squared value (0.40) and low P-value(0) shows there is a relationship between variables. Q-Q data plot is not normally distributed and scatter plots are very heteroscedastic.

In [354]:

```
data_pred = data_fin.iloc[:,0:15]
data_pred
```

Out[354]:

	pp	lot	lat	living
0	0.032616	-0.382321	-0.077514	-1.126726
1	0.104261	-0.105936	0.258885	0.746830
2	0.023119	0.253334	0.285919	-2.154206
3	0.119220	-0.518395	-0.062088	0.094637
4	0.097915	0.015972	0.091993	-0.276397
...	...	...	...	...
21592	0.063917	-2.173235	0.224160	-0.501510
21593	0.072983	-0.350656	-0.078317	0.490108
21594	0.073459	-1.976166	0.056069	-1.477448
21595	0.072983	-1.341154	-0.040081	-0.393833

In [355]:

```
data_pred.corr()
```

Out[355]:

	pp	lot	lat	living
pp	1.000000	0.146261	0.331163	0.610355
lot	0.146261	1.000000	-0.152340	0.316015
lat	0.331163	-0.152340	1.000000	0.033681
living	0.610355	0.316015	0.033681	1.000000

In [356]:

```
abs(data_pred.corr()) > 0.5
```

Out[356]:

	pp	lot	lat	living
pp	True	False	False	True
lot	False	True	False	False
lat	False	False	True	False
living	True	False	False	True

In [357]:

```
plt.figure(figsize=(15,10))
sns.heatmap(data_fin.corr().round(2),annot=True)
```

Out[357]:

```
<AxesSubplot:>
```

In [ ]:

```
Conclusion:

The price and sqft living is 61% correlated and showing normal distribution.
the r2 and adjust r2 is 45%
```

Iteration 2-Model evaluation

Train-test split

In [335]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import FunctionTransformer, OneHotEncoder
from sklearn.linear_model import LinearRegression
```

In [336]:

```
X = cont_data.drop("price", axis=1)
y = cont_data["price"]
```

In [337]:

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=101)
```

In [338]:

```
lm=LinearRegression()
lm.fit(X_train,y_train)
```

Out[338]:

```
LinearRegression()
```

In [339]:

```
rootMeanSquareError = np.sqrt(meanSquareError).round(3)
rootMeanSquareError
```

Out[339]:

```
228036.996
```

In [340]:

```
y_predict= lm.predict(X_train)
```

In [341]:

```
y_predict
```

Out[341]:

```
array([765179.34565939, 288045.7561236 , 694204.53102167, ...,
       412681.85528234, 357586.28744134, 294053.1575178 ])
```

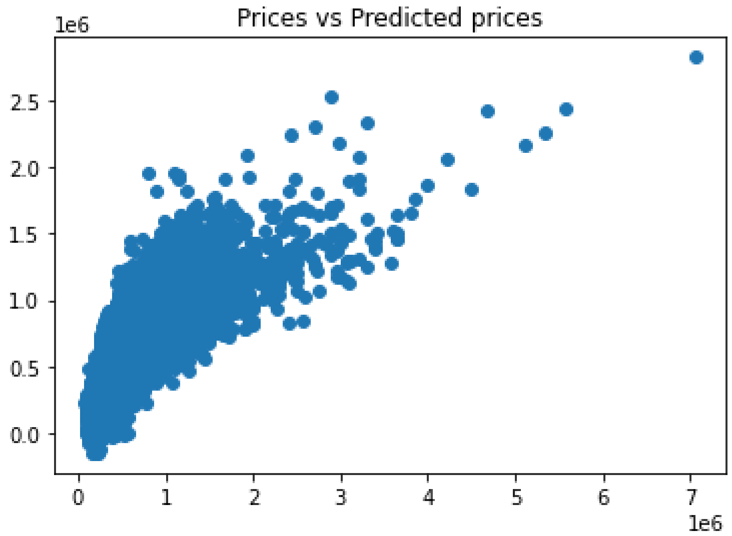
In [342]:

```
#model evaluation
print('R^2:',metrics.r2_score(y_train, y_predict))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_train, y_predict))*(len(y_train)-1)/(len(y_
print('MAE:',metrics.mean_absolute_error(y_train, y_predict))
print('MSE:',metrics.mean_squared_error(y_train, y_predict))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_predict)))
```

```
R^2: 0.6003850414249247
Adjusted R^2: 0.6001205008724455
MAE: 148100.68136135113
MSE: 54323037409.31655
RMSE: 233073.030205806
```

In [343]:

```
import matplotlib.pyplot as plt

plt.scatter(y_train,y_predict)
plt.title("Prices vs Predicted prices")
plt.show()
```



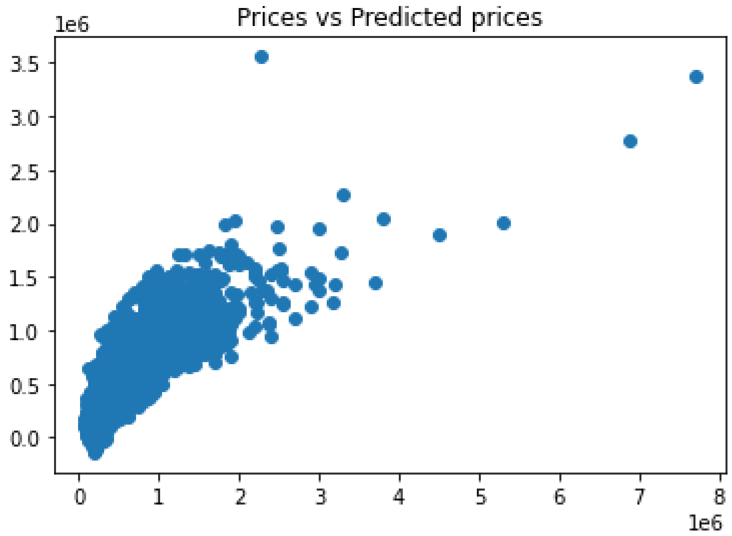
In [344]:

```
#linear regression for ols

y_test_pred=lm.predict(X_test)

acc_linreg=metrics.r2_score(y_test,y_test_pred)
print('R^2:',acc_linreg)
print('Adjusted R^2:', 1- (1-metrics.r2_score(y_test,y_test_pred))*(len(y_test)-1)/(len(y_t
print('MAE:',metrics.mean_absolute_error(y_test, y_test_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_test_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
R^2: 0.6166708534334627
Adjusted R^2: 0.6160782902141606
MAE: 143292.99656191687
MSE: 50835736790.27583
RMSE: 225467.81763763056
```

In [345]:

```
import matplotlib.pyplot as plt

plt.scatter(y_test,y_test_pred)
plt.title("Prices vs Predicted prices")
plt.show()
```



Conclusion:

This model shows that MSE value is higher for x-test than for x-train. R-squared value (0.60) for X-train and R-squared value (0.61) for X-test. The scatter plot is showing non linear effect.

In [ ]: