In [4]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler,PolynomialFeatures
%matplotlib inline
```

In [5]:

```
file_name='https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/Cogn
df=pd.read_csv(file_name)
df.dtypes
```

Out[5]:

```
Unnamed: 0         int64
id                 int64
date              object
price            float64
bedrooms         float64
bathrooms        float64
sqft_living        int64
sqft_lot           int64
floors           float64
waterfront         int64
view               int64
condition          int64
grade              int64
sqft_above         int64
sqft_basement      int64
yr_built           int64
yr_renovated       int64
zipcode            int64
lat              float64
long             float64
sqft_living15      int64
sqft_lot15         int64
dtype: object
```

In [6]:

```
df.drop("id", axis = 1, inplace = True)
df.drop("Unnamed: 0", axis = 1, inplace = True)

df.describe()
```

Out[6]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wa
count	2.161300e+04	21600.000000	21603.000000	21613.000000	2.161300e+04	21613.000000	21613
mean	5.400881e+05	3.372870	2.115736	2079.899736	1.510697e+04	1.494309	0
std	3.671272e+05	0.926657	0.768996	918.440897	4.142051e+04	0.539989	0
min	7.500000e+04	1.000000	0.500000	290.000000	5.200000e+02	1.000000	0
25%	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1

In [7]:

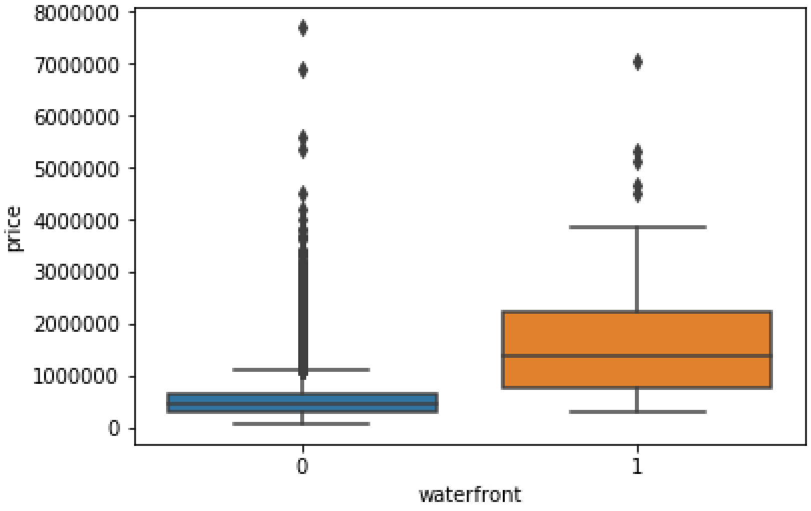
```
df['floors'].value_counts().to_frame()
```

Out[7]:

	floors
1.0	10680
2.0	8241
1.5	1910
3.0	613
2.5	161
3.5	8

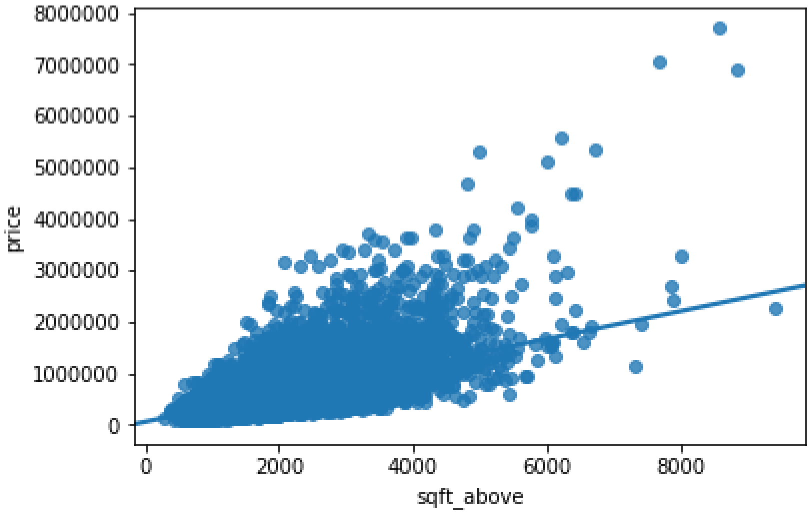
```
In [8]: sns.boxplot(x="waterfront", y="price", data=df)
```

```
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x1c231da1438>
```



```
In [9]: sns.regplot(x="sqft_above", y="price", data=df, ci = None)
```

```
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x1c231eb94a8>
```



```
In [10]: import matplotlib.pyplot as plt
         from sklearn.linear_model import LinearRegression
```

```
In [16]: X1 = df[['sqft_living']]
         Y1 = df['price']
         lm = LinearRegression()
         lm
         lm.fit(X1,Y1)
         lm.score(X1, Y1)
```

```
Out[16]: 0.49285321790379316
```

In [17]:

```
mean=df['bathrooms'].mean()
df['bathrooms'].replace(np.nan,mean, inplace=True)
mean=df['bedrooms'].mean()
df['bedrooms'].replace(np.nan,mean, inplace=True)
```

In [18]:

```
features =["floors", "waterfront","lat" ,"bedrooms" ,"sqft_basement" ,"view" ,"ba
           "sqft_living15","sqft_above","grade","sqft_living"]
X2 = df[features]
Y2 = df['price']
lm.fit(X2,Y2)
lm.score(X2,Y2)
```

Out[18]:

```
0.6576951666037494
```

In [20]:

```
Input=[('scale',StandardScaler()),('polynomial', PolynomialFeatures(include_bias
pipe=Pipeline(Input)
pipe.fit(df[features],df['price'])
pipe.score(df[features],df['price'])
```

```
C:\Users\Saurav Singla\Anaconda3\lib\site-packages\sklearn\preprocessing\data.p
y:645: DataConversionWarning: Data with input dtype int64, float64 were all con
verted to float64 by StandardScaler.
  return self.partial_fit(X, y)
C:\Users\Saurav Singla\Anaconda3\lib\site-packages\sklearn\base.py:467: DataCon
versionWarning: Data with input dtype int64, float64 were all converted to floa
t64 by StandardScaler.
  return self.fit(X, y, **fit_params).transform(X)
C:\Users\Saurav Singla\Anaconda3\lib\site-packages\sklearn\pipeline.py:511: Dat
aConversionWarning: Data with input dtype int64, float64 were all converted to
float64 by StandardScaler.
  Xt = transform.transform(Xt)
```

Out[20]:

```
0.7513404614351351
```

In [24]:

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
```

In [26]:

```
X = df[features ]
Y = df['price']
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.15, random_
RigeModel = Ridge(alpha=0.1)
RigeModel.fit(x_train, y_train)
RigeModel.score(x_test, y_test)
```

Out[26]:

```
0.6478759163939115
```

In [27]:

```
pr=PolynomialFeatures(degree=2)
x_train_pr=pr.fit_transform(x_train[features])
x_test_pr=pr.fit_transform(x_test[features])

RigeModel = Ridge(alpha=0.1)
RigeModel.fit(x_train_pr, y_train)
RigeModel.score(x_test_pr, y_test)
```

Out[27]:

```
0.7002744265869922
```

In [ ]: