```
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
from sklearn.linear_model import LinearRegression
%matplotlib inline
```

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

```
df.head()
```

	Unnamed: 0	id	date	price	bedrooms	bathrooms	sqft_living
0	0	7129300520	20141013T000000	221900.0	3.0	1.00	1180
1	1	6414100192	20141209T000000	538000.0	3.0	2.25	2570
2	2	5631500400	20150225T000000	180000.0	2.0	1.00	770
3	3	2487200875	20141209T000000	604000.0	4.0	3.00	1960
4	4	1954400510	20150218T000000	510000.0	3.0	2.00	1680

5 rows × 22 columns

```
df.dtypes
```

```
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
```

```
df.describe()
```

	Unnamed: 0	id	price	bedrooms	bathrooms	sqft_livin
count	21613.00000	2.161300e+04	2.161300e+04	21600.000000	21603.000000	21613.00000
mean	10806.00000	4.580302e+09	5.400881e+05	3.372870	2.115736	2079.89973
std	6239.28002	2.876566e+09	3.671272e+05	0.926657	0.768996	918.44089
min	0.00000	1.000102e+06	7.500000e+04	1.000000	0.500000	290.00000
25%	5403.00000	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.00000
50%	10806.00000	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.00000
75%	16209.00000	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.00000
max	21612.00000	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.00000

8 rows × 21 columns

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

df.describe()
```

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floo
count	2.161300e+04	21600.000000	21603.000000	21613.000000	2.161300e+04	21613.0000
mean	5.400881e+05	3.372870	2.115736	2079.899736	1.510697e+04	1.4943
std	3.671272e+05	0.926657	0.768996	918.440897	4.142051e+04	0.5399
min	7.500000e+04	1.000000	0.500000	290.000000	5.200000e+02	1.0000
25%	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.0000
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.5000
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.0000
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.5000

```
print("number of NaN values for the column bedrooms :", df['bedrooms'].isnull().sum())
print("number of NaN values for the column bathrooms :", df['bathrooms'].isnull().sum())
```

```
number of NaN values for the column bedrooms : 13
number of NaN values for the column bathrooms : 10
```

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


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


print("number of NaN values for the column bedrooms :", df['bedrooms'].isnull().sum())
print("number of NaN values for the column bathrooms :", df['bathrooms'].isnull().sum())
```

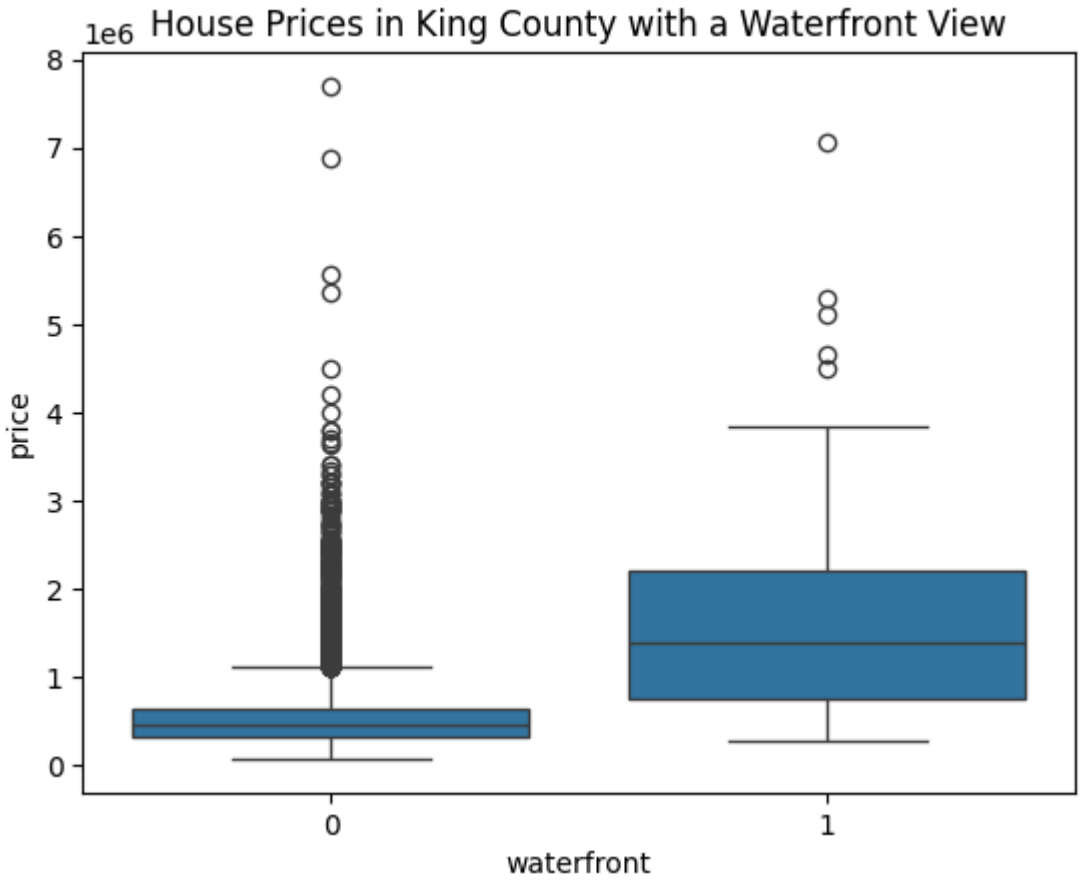
```
number of NaN values for the column bedrooms : 0
number of NaN values for the column bathrooms : 0
```

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

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

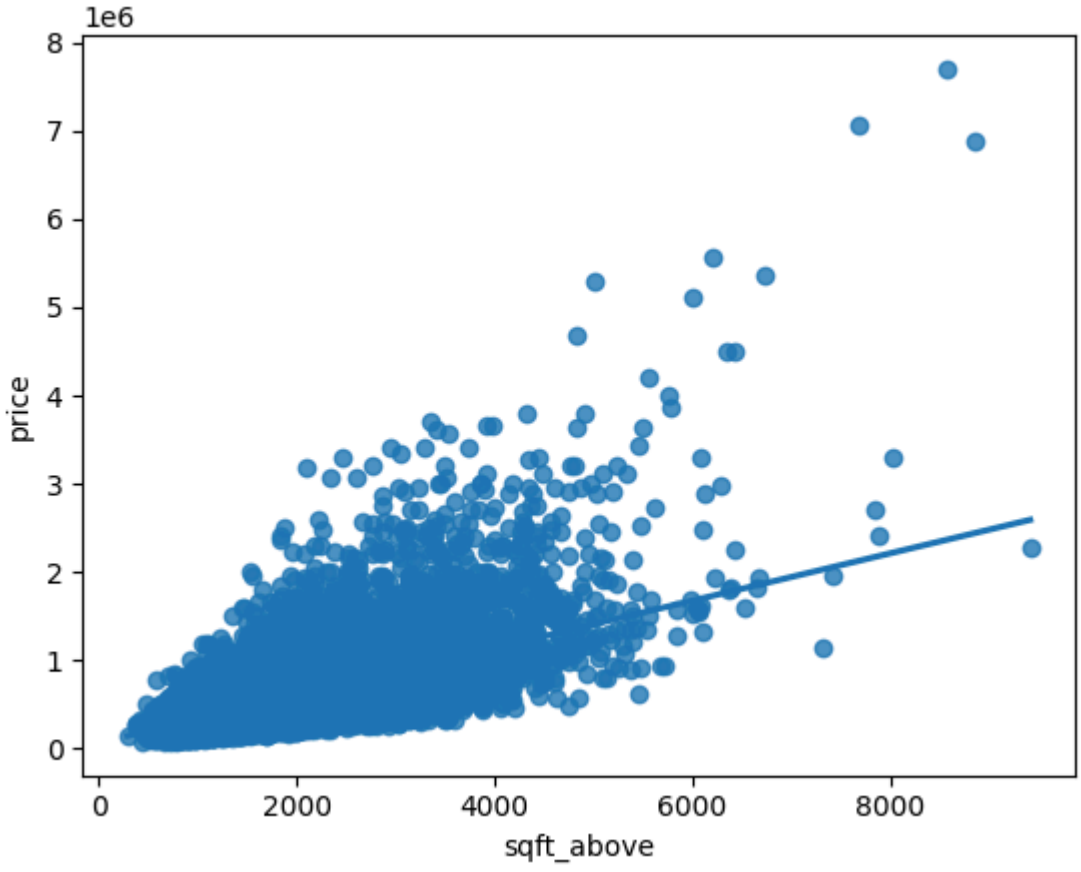
```
sns.boxplot(x="waterfront", y="price", data=df).set_title('House Prices in King County wi
```

```
Text(0.5, 1.0, 'House Prices in King County with a Waterfront View')
```




```
sns.regplot(x="sqft_above", y="price", data=df, ci = None)
```

```
<Axes: xlabel='sqft_above', ylabel='price'>
```



```
df = df.drop('date', axis=1)
df.corr()['price'].sort_values()
```

```
zipcode         -0.053203
long             0.021626
condition        0.036362
yr_built         0.054012
sqft_lot15       0.082447
sqft_lot         0.089661
yr_renovated     0.126434
floors           0.256794
waterfront       0.266369
lat              0.307003
bedrooms         0.308797
sqft_basement    0.323816
view             0.397293
bathrooms        0.525738
sqft_living15    0.585379
sqft_above       0.605567
grade            0.667434
sqft_living      0.702035
price            1.000000
Name: price, dtype: float64
```

```
X = df[['long']]
Y = df['price']
lm = LinearRegression()
lm.fit(X,Y)
lm.score(X, Y)
```

```
0.00046769430149007363
```

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

```
0.4928532179037931
```

```
features =["floors", "waterfront","lat" ,"bedrooms" ,"sqft_basement" ,"view" ,
           "bathrooms","sqft_living15","sqft_above","grade","sqft_living"]
```

```
X2 = df[features]
Y2 = df['price']
lm.fit(X2,Y2)
lm.score(X2,Y2)
```

```
0.6576722447699446
```

```
Input=[('scale',StandardScaler()),('polynomial', PolynomialFeatures(include_bias=False)),
```

```
pipe=Pipeline(Input)
pipe
pipe.fit(df[features],df['price'])
pipe.score(df[features],df['price'])
```

```
0.7513410648797747
```

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
print("done")
```

```
done
```

```
features =["floors", "waterfront","lat" ,"bedrooms" ,"sqft_basement" ,"view" ,"bathrooms"
X = df[features]
Y = df['price']

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.15, random_state=1)


print("number of test samples:", x_test.shape[0])
print("number of training samples:",x_train.shape[0])
```

```
number of test samples: 3242
number of training samples: 18371
```

```
from sklearn.linear_model import Ridge


RigeModel = Ridge(alpha=0.1)
RigeModel.fit(x_train, y_train)
RigeModel.score(x_test, y_test)
```

```
0.6478759163939112
```

```
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)
```