### Question 1

Display the data types of each column using the function dtypes, then take a screenshot and submit it, include your code in the image.

In [7]:
```
df.dtypes
```

Out[7]:
```
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
```

We use the method describe to obtain a statistical summary of the dataframe.

In [7]:
```
df=pd.read_csv(file_name)

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

df.describe()
```

Out[7]:

| | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | condition | g |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 2.161300e+04 | 21600.000000 | 21603.000000 | 21613.000000 | 2.161300e+04 | 21613.000000 | 21613.000000 | 21613.000000 | 21613.000000 | 21613.00 |
| **mean** | 5.400881e+05 | 3.372870 | 2.115736 | 2079.899736 | 1.510697e+04 | 1.494309 | 0.007542 | 0.234303 | 3.409430 | 7.65687 |
| **std** | 3.671272e+05 | 0.926657 | 0.768996 | 918.440897 | 4.142051e+04 | 0.539989 | 0.086517 | 0.766318 | 0.650743 | 1.17545 |
| **min** | 7.500000e+04 | 1.000000 | 0.500000 | 290.000000 | 5.200000e+02 | 1.000000 | 0.000000 | 0.000000 | 1.000000 | 1.00000 |
| **25%** | 3.219500e+05 | 3.000000 | 1.750000 | 1427.000000 | 5.040000e+03 | 1.000000 | 0.000000 | 0.000000 | 3.000000 | 7.00000 |
| **50%** | 4.500000e+05 | 3.000000 | 2.250000 | 1910.000000 | 7.618000e+03 | 1.500000 | 0.000000 | 0.000000 | 3.000000 | 7.00000 |
| **75%** | 6.450000e+05 | 4.000000 | 2.500000 | 2550.000000 | 1.068800e+04 | 2.000000 | 0.000000 | 0.000000 | 4.000000 | 8.00000 |
| **max** | 7.700000e+06 | 33.000000 | 8.000000 | 13540.000000 | 1.651359e+06 | 3.500000 | 1.000000 | 4.000000 | 5.000000 | 13.0000 |

we can see we have missing values for the columns `bedrooms` and `bathrooms`

In [8]:
```
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
```

### Question 3

Use the method `value_counts` to count the number of houses with unique floor values, use the method `.to_frame()` to convert it to a dataframe.

n [27]:
```
df['floors'].value_counts().to_frame()
```

Out[27]:

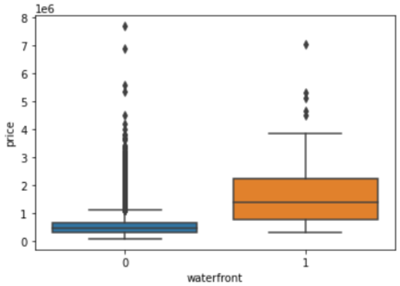
| | floors |
|---|---|
| **1.0** | 10680 |
| **2.0** | 8241 |
| **1.5** | 1910 |
| **3.0** | 613 |
| **2.5** | 161 |
| **3.5** | 8 |

### Question 4

Use the function `boxplot` in the seaborn library to determine whether houses with a waterfront view or without a waterfront view have more price outliers.

In [29]:
```
sns.boxplot(x='waterfront', y='price', data=df)
```

Out[29]: <AxesSubplot:xlabel='waterfront', ylabel='price'>

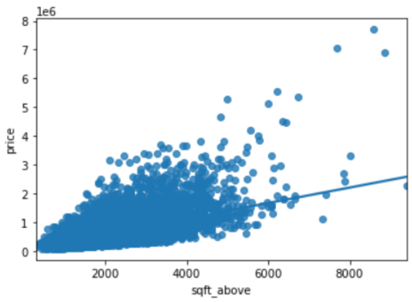



### Question 5

Use the function `regplot` in the seaborn library to determine if the feature `sqft_above` is negatively or positively correlated with price.

[30]:
```
sns.regplot(x="sqft_above", y="price", data=df, ci = None)
```

ut[30]: <AxesSubplot:xlabel='sqft_above', ylabel='price'>




We can use the Pandas method `corr()` to find the feature other than price that is most correlated with price.

[ ]:
```
df.corr()['price'].sort_values()
```

### Question 6

Fit a linear regression model to predict the `'price'` using the feature `'sqft_living'` then calculate the R^2. Take a screenshot of your code and the value of the R^2.

34]:
```
Z = df[['sqft_living']]
Y = df['price']
lm1 = LinearRegression()
lm1.fit(Z,Y)
lm1.score(Z,Y)
```

t[34]: 0.4928532179037931

### Question 7

Fit a linear regression model to predict the `'price'` using the list of features:

5]:
```
features =["floors", "waterfront","lat" ,"bedrooms" ,"sqft_basement" ,"view" ,"bathrooms","sqft_living15","sqft_above","grade","sqft_living"]
```

Then calculate the R^2. Take a screenshot of your code.

8]:
```
XX = df[features]
YY = df['price']
lm2 = LinearRegression()
lm2.fit(XX,YY)
lm2.score(XX,YY)
```

[38]: 0.6576569675583581

### Question 8

Use the list to create a pipeline object to predict the 'price', fit the object using the features in the list `features`, and calculate the R^2.

In [40]:
```
pipe=Pipeline(Input)
pipe
pipe.fit(XX,Y)
pipe.score(XX,Y)
```

Out[40]: 0.7513417707683823

### Question 9

Create and fit a Ridge regression object using the training data, set the regularization parameter to 0.1, and calculate the R^2 using the test data.

In [44]:
```
from sklearn.linear_model import Ridge
```

In [45]:
```
RM = Ridge(alpha=0.1)
RM.fit(x_train, y_train)
RM.score(x_test, y_test)
```

Out[45]: 0.6478759163939113

### Question 10

Perform a second order polynomial transform on both the training data and testing data. Create and fit a Ridge regression object using the training data, set the regularisation parameter to 0.1, and calculate the R^2 utilising the test data provided. Take a screenshot of your code and the R^2.

[46]:
```
pr=PolynomialFeatures(degree=2)
x_train_pr=pr.fit_transform(x_train[features])
x_test_pr=pr.fit_transform(x_test[features])

RM2 = Ridge(alpha=0.1)
RM2.fit(x_train_pr, y_train)
RM2.score(x_test_pr, y_test)
```

ut[46]: 0.7002744273468813