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.

[9]:
```
df.dtypes
```

[9]:
```
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
```

Question 2

Drop the columns `"id"` and `"Unnamed: 0"` from axis 1 using the method `drop()`, then use the method `describe()` to obtain a statistical summary of the data. Take a screenshot and submit it, make sure the `inplace` parameter is set to `True`

[11]:
```
df.drop(['Unnamed: 0','id'], axis=1,inplace = True)
df.describe()
```

[11]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	s
count	2.161300e+04	21600.000000	21603.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	
mean	5.400881e+05	3.372870	2.115736	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	3.409430	7.656873	1788.390691	
std	3.671272e+05	0.926657	0.768996	918.440897	4.142051e+04	0.539989	0.086517	0.766318	0.650743	1.175459	828.090978	
min	7.500000e+04	1.000000	0.500000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	1.000000	1.000000	290.000000	
25%	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	3.000000	7.000000	1190.000000	
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	3.000000	7.000000	1560.000000	
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	4.000000	8.000000	2210.000000	
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	5.000000	13.000000	9410.000000	

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.

[17]:
```
df['floors'].value_counts().to_frame()
```

[17]:

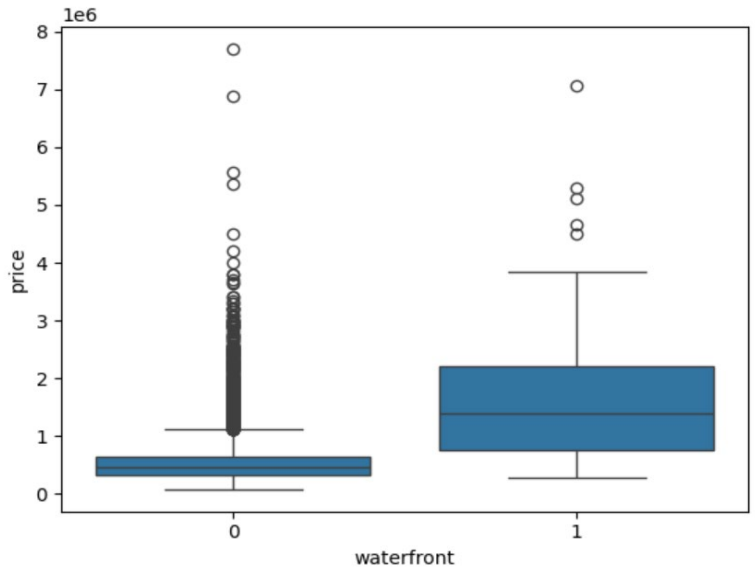
	floors
1.0	10680
2.0	8241
1.5	1910
3.0	613
2.5	161
3.5	8

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.

[18]:
```
sns.boxplot(x='waterfront',y='price',data=df)
```

[18]:
```
<AxesSubplot:xlabel='waterfront', ylabel='price'>
```

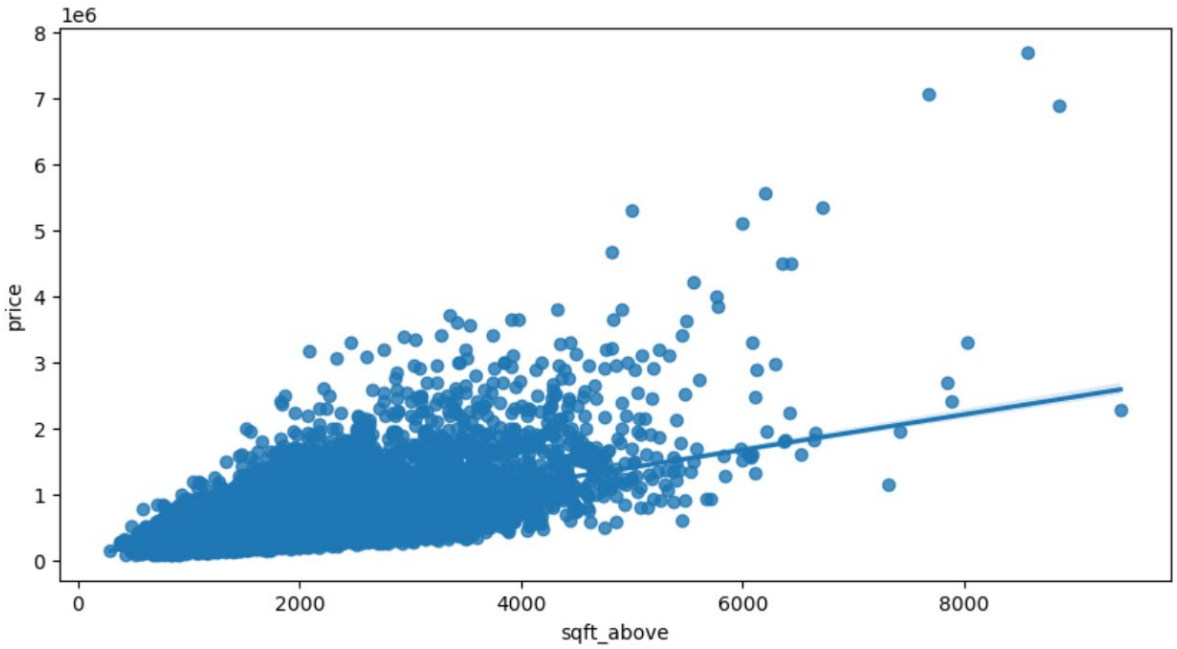


Question 5

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

[19]:
```
plt.figure(figsize=(10,5))
sns.regplot(x='sqft_above',y='price',data=df)
```

[19]:
```
<AxesSubplot:xlabel='sqft_above', ylabel='price'>
```



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.

[22]:
```
X = df[['sqft_living']]
Y = df[['price']]
lm = LinearRegression()
lm.fit(X,Y)
lm.score(X, Y)
```

[22]:
```
0.4928532179037931
```

Question 7

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

[23]:
```
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.

[24]:
```
X = df[features]
Y = df['price']
lm = LinearRegression()
lm.fit(X,Y)
lm.score(X,Y)
```

[24]:
```
0.6576950629068081
```

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.

[26]:
```
Z = df[features]
Y = df['price']
pipe = Pipeline(Input)
pipe.fit(Z,Y)
yhat=pipe.predict(Z)
pipe.score(Z,Y)
```

[26]:
```
0.7512786321941719
```

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.

[29]:
```
from sklearn.linear_model import Ridge
```

[30]:
```
RigeModel=Ridge(alpha=0.1)
RigeModel.fit(x_train, y_train)
RigeModel.score(x_test, y_test)
```

[30]:
```
0.647875916393906
```

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.

[31]:
```
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)
```

[31]:
```
0.7002744263350642
```