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. You will need to submit it for the final project.

[56]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

# Step 1: Define the features and target
features = ['sqft_living', 'sqft_above', 'bedrooms', 'bathrooms']
X = df[features]  # Features
y = df['price']   # Target

# Step 2: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Perform second-order polynomial transformation
poly = PolynomialFeatures(degree=2)

# Transform both training and testing data
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

# Step 4: Create the Ridge regression model with regularization parameter alpha=0.1
ridge_model = Ridge(alpha=0.1)

# Step 5: Fit the Ridge regression model using the transformed training data
ridge_model.fit(X_train_poly, y_train)

# Step 6: Predict on the test data
y_pred = ridge_model.predict(X_test_poly)

# Step 7: Calculate the R^2 score
r2 = r2_score(y_test, y_pred)

# Step 8: Print the R^2 score
print("R^2 score:", r2)
```

```
R^2 score: 0.46004132432687495
```

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. Take a screenshot of your code and the value of the R^2. You will need to submit it for the final project.

[54]:
```
from sklearn.linear_model import Ridge
```

[55]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import r2_score

# Step 1: Define the features and target
features = ['sqft_living', 'sqft_above', 'bedrooms', 'bathrooms']
X = df[features]  # Features
y = df['price']   # Target

# Step 2: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Create the Ridge regression model with regularization parameter alpha=0.1
ridge_model = Ridge(alpha=0.1)

# Step 4: Fit the model on the training data
ridge_model.fit(X_train, y_train)

# Step 5: Predict on the test data
y_pred = ridge_model.predict(X_test)

# Step 6: Calculate the R^2 score
r2 = r2_score(y_test, y_pred)

# Step 7: Print the R^2 score
print("R^2 score:", r2)
```

```
R^2 score: 0.5093227739309818
```

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. Take a screenshot of your code and the value of the R^2. You will need to submit it for the final project.

[51]:
```
#from sklearn.linear_model import LinearRegression
#from sklearn.preprocessing import StandardScaler
#from sklearn.pipeline import Pipeline
#from sklearn.metrics import r2_score

# Define the list of features
features = ['sqft_living', 'sqft_above', 'bedrooms', 'bathrooms']

# Step 1: Set up the features and target
X = df[features]  # Features
y = df['price']   # Target

# Step 2: Create the pipeline
pipeline = Pipeline([
    ('scaler', StandardScaler()),  # Step 2a: Standardize the features
    ('regressor', LinearRegression())  # Step 2b: Linear regression model
])

# Step 3: Fit the pipeline
pipeline.fit(X, y)

# Step 4: Predict the prices
y_pred = pipeline.predict(X)

# Step 5: Calculate the R^2 score
r2 = r2_score(y, y_pred)

# Step 6: Print the R^2 score
print("R^2 score:", r2)
```

```
R^2 score: 0.5079932298819632
```

Question 7¶ Fit a linear regression model to predict the 'price' using the list of features

[50]:
```
features =["floors", "waterfront","lat" ,"bedrooms" ,"sqft_basement" ,"view" ,"bathrooms","sqft_living15","sqft_above","grade","sqft_living"]
# Step 1: Set up features and target
X = df[features]  # Features
y = df['price']   # Target

# Step 2: Create and fit the model
model = LinearRegression()
model.fit(X, y)

# Step 3: Predict
y_pred = model.predict(X)

# Step 4: Calculate R^2
r2 = r2_score(y, y_pred)

# Step 5: Print R^2 score
print("R^2 score:", r2)
```

```
R^2 score: 0.6576890354915759
```

Question 6 Fit a l_inear 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. You will need to submit it for the final project.

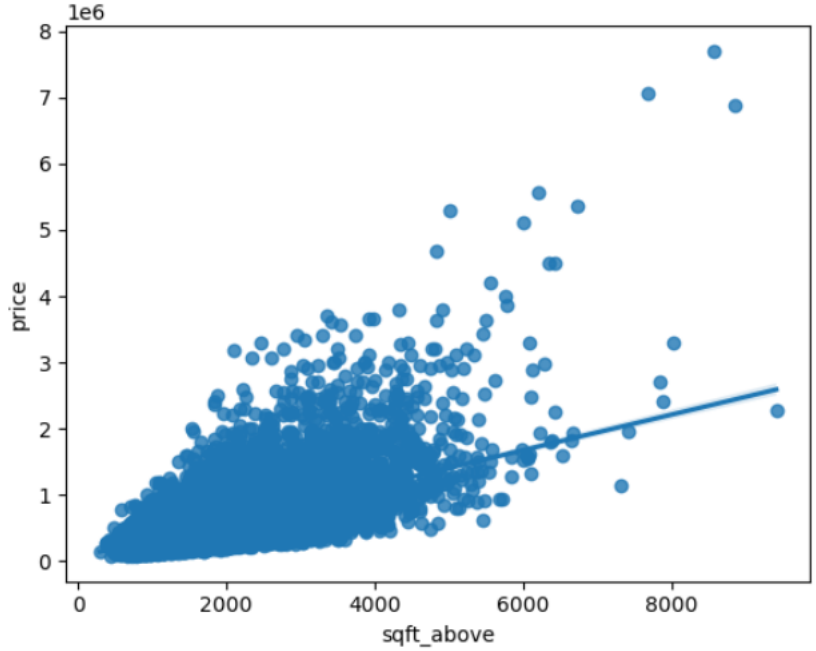
[49]:
```
from sklearn.metrics import r2_score

# Step 1: Set up the feature and target
X = df[['sqft_living']]  # Feature must be 2D
y = df['price']          # Target

# Step 2: Create and fit the model
model = LinearRegression()
model.fit(X, y)

# Step 3: Predict
y_pred = model.predict(X)

# Step 4: Calculate R^2
r2 = r2_score(y, y_pred)

# Step 5: Print R^2
print("R^2 score:", r2)
```

```
R^2 score: 0.4928532179037931
```

Question 5 Use the function regplot in the seaborn library to determine if the feature sqft_above is negatively or positively correlated with price. Take a screenshot of your code and scatterplot. You will need to submit the screenshot for the final project.

[43]:
```
sns.regplot(x='sqft_above', y='price', data=df)
```

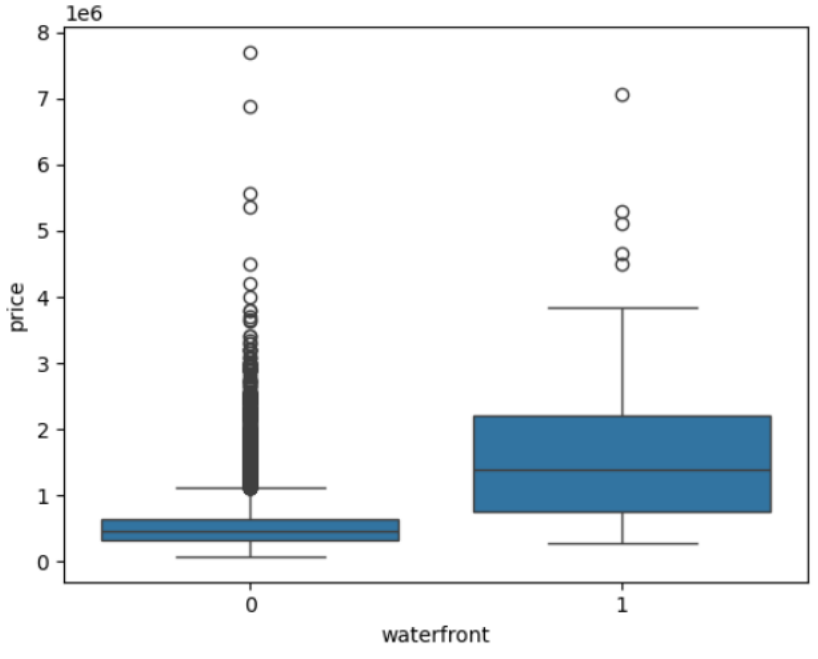
[43]:
```
<AxesSubplot:xlabel='sqft_above', ylabel='price'>
```



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. Take a screenshot of your code and boxplot. You will need to submit the screenshot for the final project.

[41]:
```
sns.boxplot(x='waterfront', y='price', data=df)
```

[41]:
```
<AxesSubplot:xlabel='waterfront', ylabel='price'>
```



Module 3: Exploratory Data Analysis

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 data frame. Take a screenshot of your code and output. You will need to submit the screenshot for the final project.

[40]:
```
# Count unique floor values and convert to a DataFrame
floor_counts = df['floors'].value_counts().to_frame()

# Optionally rename the column for clarity
floor_counts.columns = ['count']

# Display the result
floor_counts
```

[40]:

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

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. Make sure the inplace parameter is set to True. Take a screenshot of your code and output. You will need to submit the screenshot for the final project.

[28]:
```
df.drop([col for col in ['id', 'Unnamed: 0'] if col in df.columns], axis=1, inplace=True)
df.describe()
```

[28]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement
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	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	291.509045
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	442.575043
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	0.000000
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	0.000000
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	0.000000
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	560.000000
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	4820.000000

Question 1 Display the data types of each column using the function dtypes. Take a screenshot of your code and output. You will need to submit the screenshot for the final project.

[13]:
```
df.dtypes
```

[13]:
```
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
```