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.

[14]:
```
print(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
```

We use the method describe to obtain a statistical summary of the datafra

[15]:
```
df.describe()
```

[15]:

	Unnamed: 0	id	price	bedrooms	bath	iving15	sqft_lot15
count	21613.00000	2.161300e+04	2.161300e+04	21600.000000	21603.0	000000	21613.000000
mean	10806.00000	4.580302e+09	5.400881e+05	3.372870	2.1	552492	12768.455652
std	6239.28002	2.876566e+09	3.671272e+05	0.926657	0.7	28	685.391304	27304.179631
min	0.00000	1.000102e+06	7.500000e+04	1.000000	0.5	00	399.000000	651.000000
25%	5403.00000	2.123049e+09	3.219500e+05	3.000000	1.7	00	1490.000000	5100.000000
50%	10806.00000	3.904930e+09	4.500000e+05	3.000000	2.2	00	1840.000000	7620.000000
75%	16209.00000	7.308900e+09	6.450000e+05	4.000000	2.5	00	2360.000000	10083.000000
max	21612.00000	9.900000e+09	7.700000e+06	33.000000	8.0	600	-121.315000	6210.000000	871200.000000

8 rows × 21 columns

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.

[16]:

Question 3

Question 4

[21]:

Question 5

[22]:

Question 6

[23]:

Question 7

Question 8

Question 9

[32]:

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

[35]:

[37]:
```
poly = PolynomialFeatures(degree=2, include_bias=False)
x_train_poly = poly.fit_transform(x_train)
x_test_poly = poly.transform(x_test)

ridge_reg = Ridge(alpha=0.1)
ridge_reg.fit(x_train_poly, y_train)

r2_score = ridge_reg.score(x_test_poly, y_test)
print(f"R^2: {r2_score}")
```

[36]:
```
R^2: 0.7002744258083224
```

[17]:

[18]:

[19]:
```
mean=df['bathrooms'].mean()
df['bathrooms'].replace(np.nan,mean, inplace=True)
```

[20]:
```
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
```


```
pipe = Pipeline(Input)

pipe.fit(X, Y)

r2_score = pipe.score(X, Y)
print(f"R^2: {r2_score}")
```

```
R^2: 0.7512051345272872
```

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.

[16]:
```
#Enter Your Code, Execute and take the Screenshot
df.drop(["id", "Unnamed: 0"], axis=1, inplace=True)

print(df.describe())
```

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

             floors    waterfront          view     condition         grade  \
count  21613.000000  21613.000000  21613.000000  21613.000000  21613.000000
mean       1.494309      0.007542      0.234303      3.409430      7.656873
std        0.539989      0.086517      0.766318      0.650743      1.175459
min        1.000000      0.000000      0.000000      1.000000      1.000000
25%        1.000000      0.000000      0.000000      3.000000      7.000000
50%        1.500000      0.000000      0.000000      3.000000      7.000000
75%        2.000000      0.000000      0.000000      4.000000      8.000000
max        3.500000      1.000000      4.000000      5.000000     13.000000

         sqft_above  sqft_basement      yr_built  yr_renovated       zipcode  \
count  21613.000000   21613.000000  21613.000000  21613.000000  21613.000000
mean    1788.390691     291.509045   1971.005136     84.402258  98077.939805
std      828.090978     442.575043     29.373411    401.679240     53.505026
min      290.000000       0.000000   1900.000000      0.000000  98001.000000
25%     1190.000000       0.000000   1951.000000      0.000000  98033.000000
50%     1560.000000       0.000000   1975.000000      0.000000  98065.000000
75%     2210.000000     560.000000   1997.000000      0.000000  98118.000000
max     9410.000000    4820.000000   2015.000000   2015.000000  98199.000000

                lat          long  sqft_living15     sqft_lot15
count  21613.000000  21613.000000   21613.000000   21613.000000
mean      47.560053   -122.213896    1986.552492   12768.455652
std        0.138564      0.140828     685.391304   27304.179631
min       47.155900   -122.519000     399.000000     651.000000
25%       47.471000   -122.328000    1490.000000    5100.000000
50%       47.571800   -122.230000    1840.000000    7620.000000
75%       47.678000   -122.125000    2360.000000   10083.000000
max       47.777600   -121.315000    6210.000000  871200.000000
```

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

[17]:
```
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
```

We can replace the missing values of the column `'bedrooms'` with the mean of the column `'bedrooms'` using the method `replace()`. Don't forget to set the `inplace` parameter to `True`

[18]:
```
mean=df['bedrooms'].mean()
df['bedrooms'].replace(np.nan,mean, inplace=True)
```

We also replace the missing values of the column `'bathrooms'` with the mean of the column `'bathrooms'` using the method `replace()`. Don't forget to set the `inplace` parameter top `True`

[19]:
```
mean=df['bathrooms'].mean()
df['bathrooms'].replace(np.nan,mean, inplace=True)
```

[20]:
```
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
```

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.

[21]:
```
floor_counts = df['floors'].value_counts().to_frame()

floor_counts.columns = ['Number of Houses']

print(floor_counts)
```

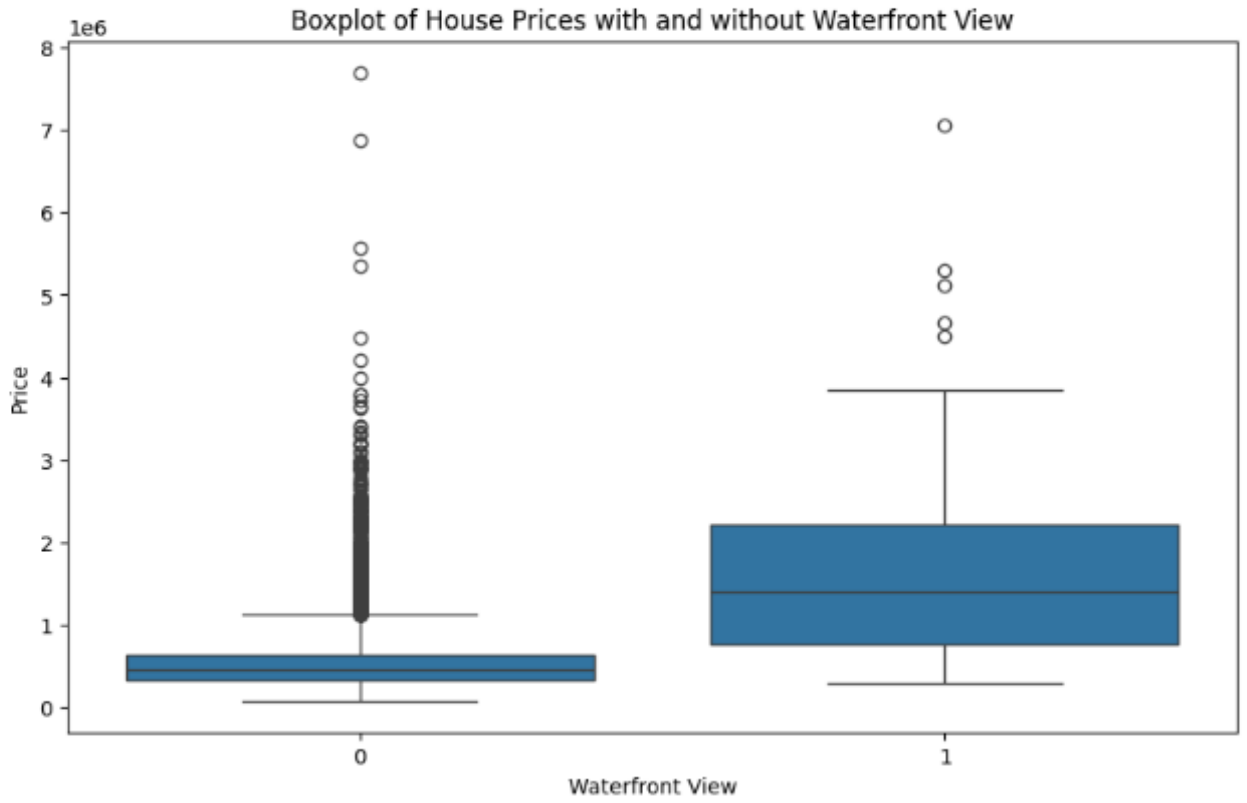
```
        Number of Houses
floors
1.0                10680
2.0                 8241
1.5                 1910
3.0                  613
2.5                  161
3.5                    8
```

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

[22]:
```
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.boxplot(x='waterfront', y='price', data=df)
plt.title('Boxplot of House Prices with and without Waterfront View')
plt.xlabel('Waterfront View')
plt.ylabel('Price')
plt.show()
```

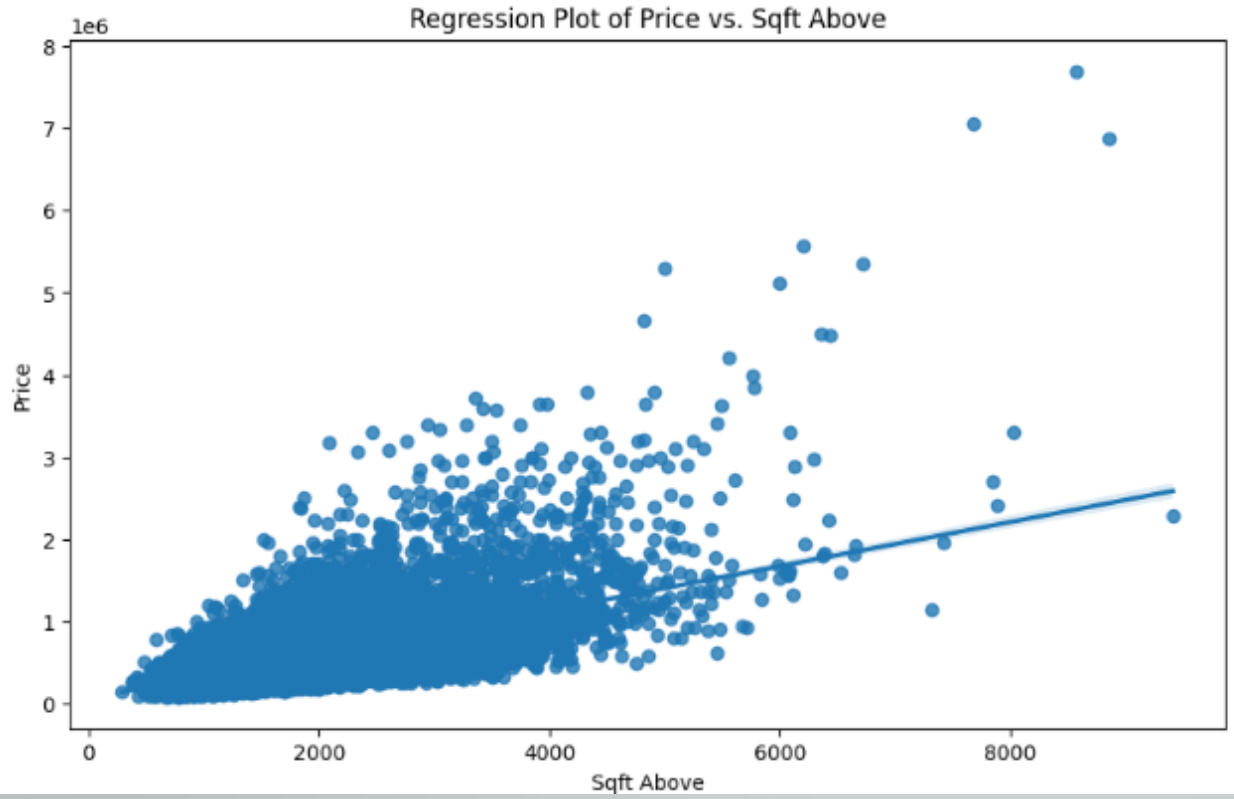


Question 5

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.

[23]:
```
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.regplot(x='sqft_above', y='price', data=df)
plt.title('Regression Plot of Price vs. Sqft Above')
plt.xlabel('Sqft Above')
plt.ylabel('Price')
plt.show()
```



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

[28]:
```
from sklearn.linear_model import LinearRegression

X = df[['sqft_living']]
Y = df['price']

lm = LinearRegression()

lm.fit(X, Y)

r2_score = lm.score(X, Y)
print(f"R^2: {r2_score}")
```

```
R^2: 0.4928532179037931
```

Question 7

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

[29]:
```
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 and the value of the R^2. You will need to submit it for the final project.

[30]:
```
X = df[features]
Y = df['price']

lm = LinearRegression()

lm.fit(X, Y)

r2_score = lm.score(X, Y)
print(f"R^2: {r2_score}")
```

```
R^2: 0.6576890354915759
```