```
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
%matplotlib inline
```

```
file_name='https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DA0101EN/coursera/project/kc_house_data_NaN.csv'
df=pd.read_csv(file_name)
```

```
df.head()
```

	Unnamed: 0	id	date	price	bedrooms
0	0	7129300520	20141013T000000	221900.0	3.0
1	1	6414100192	20141209T000000	538000.0	3.0
2	2	5631500400	20150225T000000	180000.0	2.0

Question 1 Display the data types of each column using the attribute dtype, then take a screenshot and submit it, include your code in the image.

```
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
```

```
df.describe()
```

	Unnamed: 0	id	price	bedrooms
count	21613.00000	2.161300e+04	2.161300e+04	21600.000000

2.0 Data Wrangling 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

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

	price	bedrooms	bathrooms	sqft_living
count	2.161300e+04	21600.000000	21603.000000	21613.000000
mean	5.400881e+05	3.372870	2.115736	2079.899736
std	3.671272e+05	0.926657	0.768996	918.440897
min	7.500000e+04	1.000000	0.500000	290.000000
25%	3.219500e+05	3.000000	1.750000	1427.000000

```
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
```

3.0 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 dataframe.

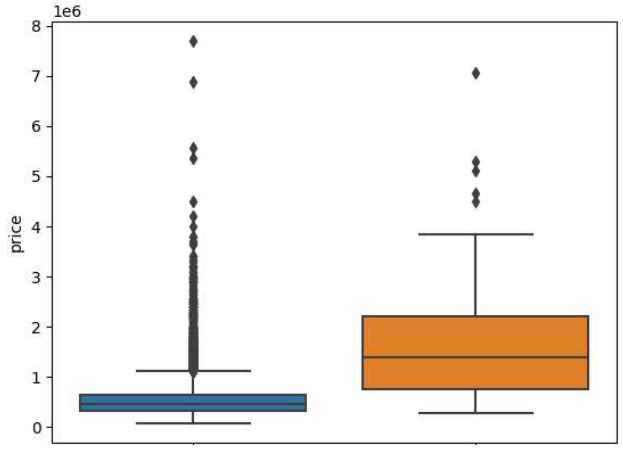
```
df['floors'].value_counts().to_frame()
```

	floors
1.0	10680
2.0	8241
1.5	1910
3.0	613
2.5	161

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

```
sns.boxplot(x='waterfront', y='price', data=df)
```

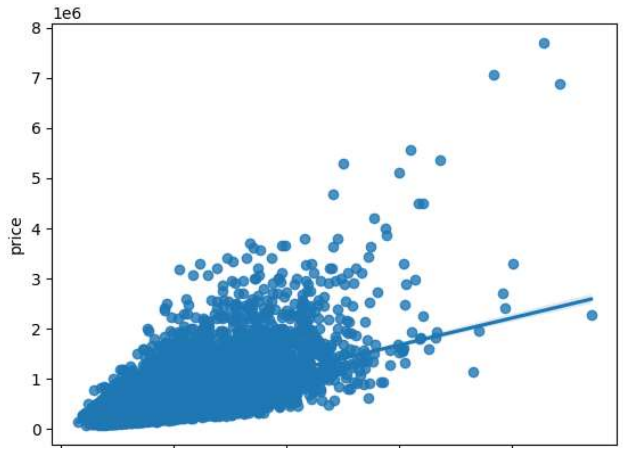
```
<Axes: xlabel='waterfront', ylabel='price'>
```



Question 5 Use the function regplot in the seaborn library to determine if the feature sqft_above is negatively or positively correlated with price.

```
sns.regplot(x='sqft_above', y='price', data=df)
```

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




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

```
<ipython-input-23-78b4f396fb2c>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future versior
  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
```

Module 4: Model Development Import libraries

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

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

```
0.00046769430149007363
```

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.

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

```
0.4928532179037931
```

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

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

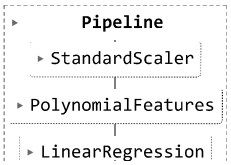
```
X = df[features]
Y= df['price']
lm = LinearRegression()
lm.fit(X, Y)
lm.score(X, Y)
```

```
0.6576722447699446
```

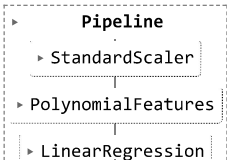
```
Input=[('scale',StandardScaler()),('polynomial', PolynomialFeatures(include_bias=False)),('model',LinearRegression())]
```

Question 8 Use the list to create a pipeline object, predict the 'price', fit the object using the features in the list features , then fit the model and calculate the R^2

```
pipe=Pipeline(Input)
pipe
```




```
pipe.fit(X,Y)
```




```
pipe.score(X,Y)
```

```
0.7513410648797747
```

Module 5: MODEL EVALUATION AND REFINEMENT import the necessary modules

```
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","sqft_living15","sqft_above","grade","sqft_living"]
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
```

Question 9 Create and fit a Ridge regression object using the training data, setting the regularization parameter to 0.1 and calculate the R^2 using the test data.

```
from sklearn.linear_model import Ridge



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

```
0.6478759163939112
```

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, setting the regularisation parameter to 0.1. Calculate the R^2 utilising the test data provided. Take a screenshot of your code and the R^2.

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
pr = PolynomialFeatures(degree=2)
x_train_pr = pr.fit_transform(x_train)
x_test_pr = pr.fit_transform(x_test)
poly = Ridge(alpha=0.1)
poly.fit(x_train_pr, y_train)
poly.score(x_test_pr, y_test)
```

```
0.700274426790608
```

✓ 0s completed at 1:02 AM