```python
import pandas as pd
import matplotlib.pylab as plt
import numpy as np
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler,PolynomialFeatures
from sklearn.pipeline import Pipeline
from tqdm import tqdm
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score

filename = "/Users/user/Documents/FPT/Spring
2023/DAP391m/dataset/lab1_kc_house_data.csv"

df = pd.read_csv(filename)
df.head(5)
```

```
           id             date     price  bedrooms  bathrooms
sqft_living  \
0  7129300520  20141013T000000  221900.0         3       1.00
1180
1  6414100192  20141209T000000  538000.0         3       2.25
2570
2  5631500400  20150225T000000  180000.0         2       1.00
770
3  2487200875  20141209T000000  604000.0         4       3.00
1960
4  1954400510  20150218T000000  510000.0         3       2.00
1680

   sqft_lot  floors  waterfront  view  ...  grade  sqft_above
sqft_basement  \
0      5650     1.0           0     0  ...      7        1180
0
1      7242     2.0           0     0  ...      7        2170
400
2     10000     1.0           0     0  ...      6         770
0
3      5000     1.0           0     0  ...      7        1050
910
4      8080     1.0           0     0  ...      8        1680
0

   yr_built  yr_renovated  zipcode      lat     long  sqft_living15  \
0      1955             0    98178  47.5112 -122.257           1340
1      1951          1991    98125  47.7210 -122.319           1690
2      1933             0    98028  47.7379 -122.233           2720
3      1965             0    98136  47.5208 -122.393           1360
4      1987             0    98074  47.6168 -122.045           1800
```

```
   sqft_lot15
0        5650
1        7639
2        8062
3        5000
4        7503

[5 rows x 21 columns]
```

Question 1: Display the data types of each column using the function dtypes

```
df.dtypes
```

```
id                 int64
date              object
price            float64
bedrooms           int64
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()
```

```
                 id         price      bedrooms     bathrooms
sqft_living  \
count  2.161300e+04  2.161300e+04  21613.000000  21613.000000
21613.000000
mean   4.580302e+09  5.400881e+05      3.370842      2.114757
2079.899736
std    2.876566e+09  3.671272e+05      0.930062      0.770163
918.440897
min    1.000102e+06  7.500000e+04      0.000000      0.000000
290.000000
25%    2.123049e+09  3.219500e+05      3.000000      1.750000
1427.000000
50%    3.904930e+09  4.500000e+05      3.000000      2.250000
```

```
1910.000000
75%    7.308900e+09  6.450000e+05      4.000000      2.500000
2550.000000
max    9.900000e+09  7.700000e+06     33.000000      8.000000
13540.000000

           sqft_lot        floors    waterfront          view
condition  \
count  2.161300e+04  21613.000000  21613.000000  21613.000000
21613.000000
mean   1.510697e+04      1.494309      0.007542      0.234303
3.409430
std    4.142051e+04      0.539989      0.086517      0.766318
0.650743
min    5.200000e+02      1.000000      0.000000      0.000000
1.000000
25%    5.040000e+03      1.000000      0.000000      0.000000
3.000000
50%    7.618000e+03      1.500000      0.000000      0.000000
3.000000
75%    1.068800e+04      2.000000      0.000000      0.000000
4.000000
max    1.651359e+06      3.500000      1.000000      4.000000
5.000000

              grade    sqft_above  sqft_basement      yr_built
yr_renovated  \
count  21613.000000  21613.000000   21613.000000  21613.000000
21613.000000
mean       7.656873   1788.390691     291.509045   1971.005136
84.402258
std        1.175459    828.090978     442.575043     29.373411
401.679240
min        1.000000    290.000000       0.000000   1900.000000
0.000000
25%        7.000000   1190.000000       0.000000   1951.000000
0.000000
50%        7.000000   1560.000000       0.000000   1975.000000
0.000000
75%        8.000000   2210.000000     560.000000   1997.000000
0.000000
max       13.000000   9410.000000    4820.000000   2015.000000
2015.000000

            zipcode           lat          long  sqft_living15
sqft_lot15
count  21613.000000  21613.000000  21613.000000   21613.000000
21613.000000
mean   98077.939805     47.560053   -122.213896    1986.552492
12768.455652
```

```
std       53.505026      0.138564      0.140828     685.391304
27304.179631
min    98001.000000     47.155900   -122.519000     399.000000
651.000000
25%    98033.000000     47.471000   -122.328000    1490.000000
5100.000000
50%    98065.000000     47.571800   -122.230000    1840.000000
7620.000000
75%    98118.000000     47.678000   -122.125000    2360.000000
10083.000000
max    98199.000000     47.777600   -121.315000    6210.000000
871200.000000
```

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

```
df.drop(["id"], axis = 1, inplace = True)
```

```
df.describe()
```

```
              price      bedrooms     bathrooms   sqft_living
sqft_lot  \
count  2.161300e+04  21613.000000  21613.000000  21613.000000
2.161300e+04
mean   5.400881e+05      3.370842      2.114757   2079.899736
1.510697e+04
std    3.671272e+05      0.930062      0.770163    918.440897
4.142051e+04
min    7.500000e+04      0.000000      0.000000    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
```

Questions 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.

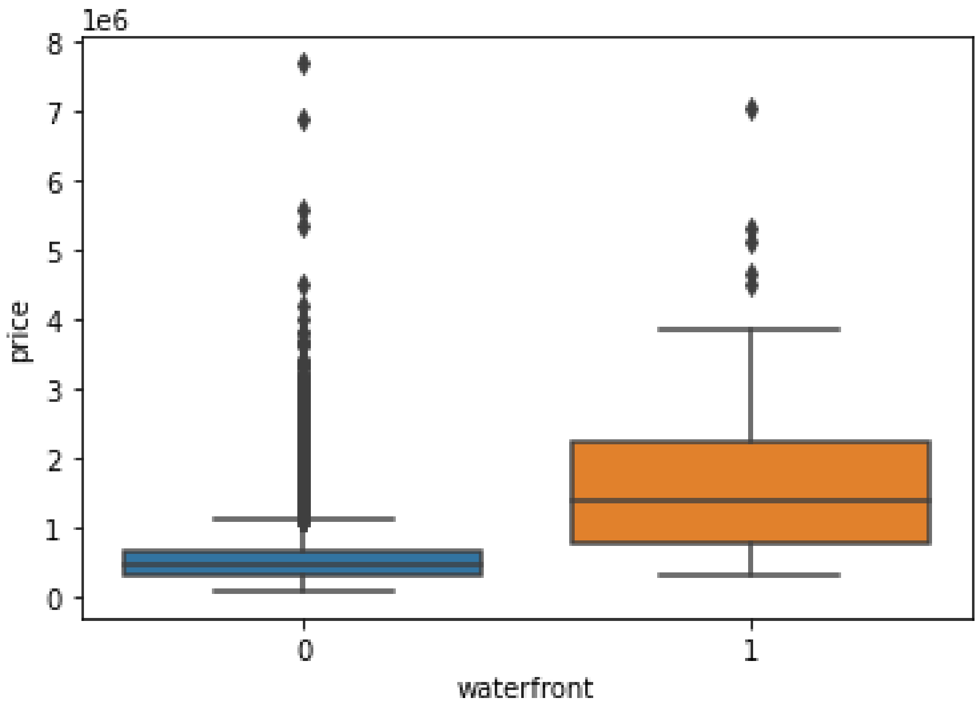
```
df['floors'].value_counts().to_frame()
```

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

Question 4: Use the function boxplot in the seaborn library to determine whether houses with a waterfront view or without the waterfront view have more price outliers.

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

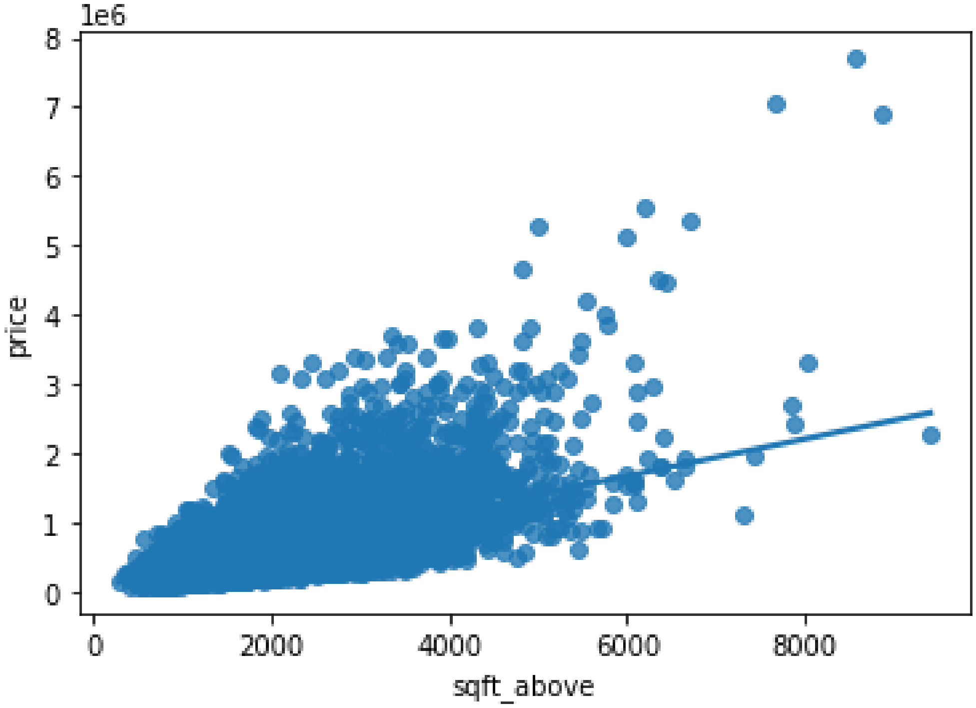
```
<AxesSubplot:xlabel='waterfront', ylabel='price'>
```



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

```
sns.regplot(data = df, x = 'sqft_above', y = 'price', ci = None)
```

```
<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 R2.

```
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"]

```
features =["floors",
"waterfront","lat" ,"bedrooms" ,"sqft_basement" ,"view" ,"bathrooms","
sqft_living15","sqft_above","grade","sqft_living"]
x = df[features]
y = df['price']

lm.fit(x,y)
lm.score(x,y)
```

0.6576821190183728

Question 8: Create a list of tuples, the first element in the tuple contains the name of the estimator:

- 'scale'
- 'polynomial'
- 'model'
- The second element in the tuple contains the model constructor
- StandardScaler()
- PolynomialFeatures(include_bias=False)
- LinearRegression()
- Input=[('scale',StandardScaler()),('polynomial', PolynomialFeatures(include_bias=False)),('model',LinearRegression())]

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

Question 9: Use the list to create a pipeline object to predict the 'price', fit the object using the features in the list feature, and calculate the R2

```
pipe = Pipeline(Input)
pipe
x = df[features]
y = df['price']
pipe.fit(x,y)
pipe.score(x,y)
```

```
0.751347545966851
```

Question 10: Create and fit a Ridge regression object using the training data, set the regularization parameter to 0.1, and calculate the R2 using the test data.

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
```

```
poly = PolynomialFeatures(degree = 2)
poly
```

```
PolynomialFeatures()
```

```
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)
```

```
x_train_poly = poly.fit_transform(x_train[["floors",
"waterfront","lat" ,"bedrooms" ,"sqft_basement" ,"view" ,"bathrooms","
```

```
sqft_living15","sqft_above","grade","sqft_living"]])
x_test_poly = poly.fit_transform(x_test[["floors",
"waterfront","lat" ,"bedrooms" ,"sqft_basement" ,"view" ,"bathrooms","
sqft_living15","sqft_above","grade","sqft_living"]])

RidgeModel = Ridge(alpha=0.1)
RidgeModel.fit(x_train_poly, y_train)
```

```
Ridge(alpha=0.1)
```

```
RidgeModel.score(x_train_poly, y_train)
```

```
0.7418393311032345
```

Question 11: 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.

```
x_train_poly = poly.fit_transform(x_train[["floors",
"waterfront","lat" ,"bedrooms" ,"sqft_basement" ,"view" ,"bathrooms","
sqft_living15","sqft_above","grade","sqft_living"]])
x_test_poly = poly.fit_transform(x_test[["floors",
"waterfront","lat" ,"bedrooms" ,"sqft_basement" ,"view" ,"bathrooms","
sqft_living15","sqft_above","grade","sqft_living"]])

RidgeModel = Ridge(alpha=0.1)
RidgeModel.fit(x_train_poly,y_train)
RidgeModel.score(x_train_poly,y_train)
```

```
0.7418393311032345
```

```
RidgeModel = Ridge(alpha=0.1)
RidgeModel.fit(x_test_poly,y_test)
RidgeModel.score(x_test_poly,y_test)
```

```
0.7655946473095798
```

Lab 1: Tính R^2

```
df['sqft_price'] = df['price'] / df['sqft_living']
z_score = (df['sqft_price']-df['sqft_price'].mean()) /
df['sqft_price'].std()
df = df[(np.abs(z_score) <= 3)]

corr_matrix = df.corr()
corr_price = corr_matrix['price']
sorted_value = corr_price.sort_values(ascending=False)
print(sorted_value)
```

```
price            1.000000
sqft_living      0.722042
```

```
grade            0.693028
sqft_above       0.619385
sqft_living15    0.600594
bathrooms        0.540960
sqft_price       0.531501
view             0.355626
lat              0.334632
bedrooms         0.329906
sqft_basement    0.327239
floors           0.279203
waterfront       0.161282
yr_renovated     0.115689
sqft_lot         0.100081
sqft_lot15       0.089622
yr_built         0.069721
long             0.036416
condition        0.033317
zipcode         -0.049983
Name: price, dtype: float64
```

```
/var/folders/_h/_mbf393j6vn3d57lvxzw185c0000gn/T/
ipykernel_34480/3888700781.py:5: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value
of numeric_only to silence this warning.
  corr_matrix = df.corr()
```

```
max_r2 = 0
for i in tqdm(range(1, len(sorted_value))):
    top_features = sorted_value[1:i+1].index.to_list()
    x = df[top_features].values
    y = df['price'].values

    x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.15, random_state=1)
    reg = RandomForestRegressor(n_estimators=100).fit(x_train,
y_train)
    y_pred = reg.predict(x_test)

    r2 = r2_score(y_test, y_pred)
    if r2 > max_r2:
        max_r2 = r2
        best_features = top_features

print(f'r2 score: {max_r2}')
```

```
100%|██████████| 19/19 [01:38<00:00,  5.20s/it]
```

```
r2 score: 0.9988156094390235
```