House Sales in King County, USA

Variable	Description
id	A notation for a house
date	Date house was sold
price	Price is prediction target
bedrooms	Number of bedrooms
bathrooms	Number of bathrooms
sqft_living	Square footage of the home
sqft_lot	Square footage of the lot
floors	Total floors (levels) in house
waterfront	House which has a view to a waterfront
view	Has been viewed
condition	How good the condition is overall
grade	overall grade given to the housing unit, based on King County grading system
sqft_above	Square footage of house apart from basement
sqft_basement	Square footage of the basement
yr_built	Built Year
yr_renovated	Year when house was renovated
zipcode	Zip code
lat	Latitude coordinate
long	Longitude coordinate
sqft_living15	Living room area in 2015(implies-- some renovations) This might or might not have affected the lotsize area
sqft_lot15	LotSize area in 2015(implies-- some renovations)

You will require the following libraries:

```
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
from sklearn.linear_model import LinearRegression
%matplotlib inline
```

Module 1: Importing Data Sets

```
file_name='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloper
df=pd.read_csv(file_name)
```

```
df.head()
```

	Unnamed: 0	id	date	price	bedrooms	bathrooms	sqft_living
0	0	7129300520	20141013T000000	221900.0	3.0	1.00	1180
1	1	6414100192	20141209T000000	538000.0	3.0	2.25	2570
2	2	5631500400	20150225T000000	180000.0	2.0	1.00	770
3	3	2487200875	20141209T000000	604000.0	4.0	3.00	1960
4	4	1954400510	20150218T000000	510000.0	3.0	2.00	1680

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.

```
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
```

discribe the data

```
df.describe()
```

	Unnamed: 0	id	price	bedrooms	bathrooms	sqft_livi
count	21613.00000	2.161300e+04	2.161300e+04	21600.000000	21603.000000	21613.0000
mean	10806.00000	4.580302e+09	5.400881e+05	3.372870	2.115736	2079.8997
std	6239.28002	2.876566e+09	3.671272e+05	0.926657	0.768996	918.4408
min	0.00000	1.000102e+06	7.500000e+04	1.000000	0.500000	290.0000
25%	5403.00000	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.0000
50%	10806.00000	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.0000
75%	16209.00000	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.0000

Double-click (or enter) to edit

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',axis='columns', inplace=True)
```

```
df.drop('Unnamed: 0',axis='columns', inplace=True)
```

```
df.describe()
```

	price	bedrooms	bathrooms	sqft_living	sqft_lot	flo
count	2.161300e+04	21600.000000	21603.000000	21613.000000	2.161300e+04	21613.000
mean	5.400881e+05	3.372870	2.115736	2079.899736	1.510697e+04	1.494
std	3.671272e+05	0.926657	0.768996	918.440897	4.142051e+04	0.539
min	7.500000e+04	1.000000	0.500000	290.000000	5.200000e+02	1.000
25%	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500

we have somme messing values

```
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`

```
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`

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

no more messing values

```
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
```

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 dataframe.

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

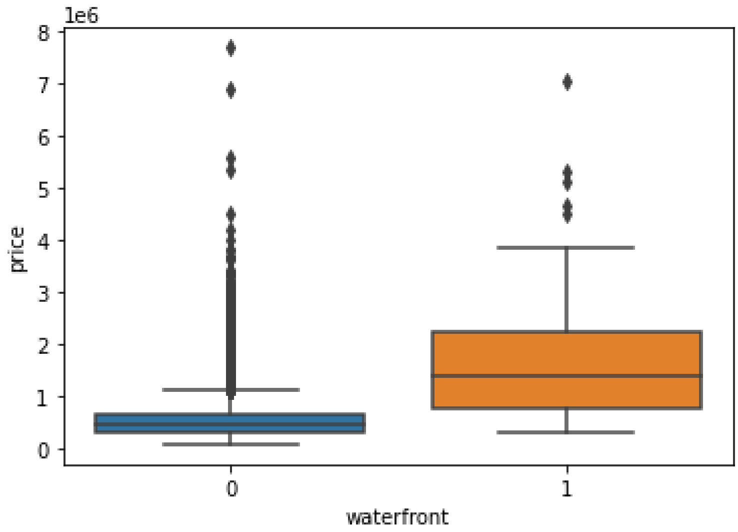
```
1.0    10680
2.0     8241
1.5     1910
3.0      613
2.5      161
3.5        8
Name: floors, dtype: int64
```

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)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f29c77113d0>
```

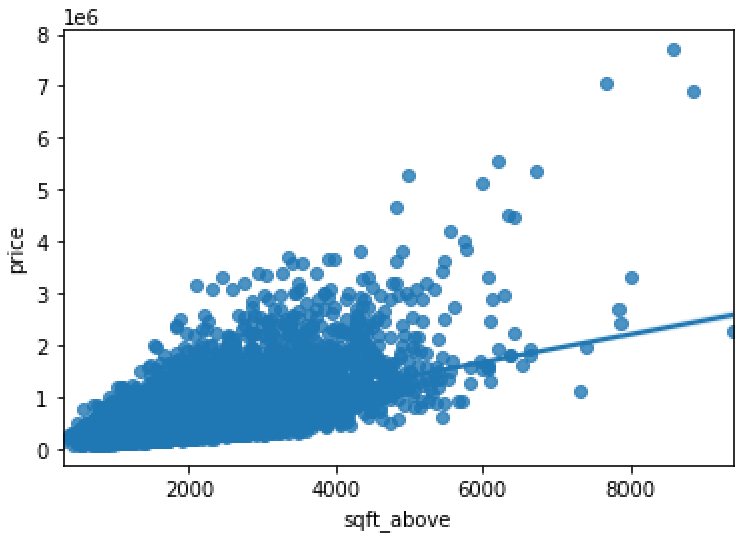


Question 5

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)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f29c76aae90>
```



We can use the Pandas method `corr()` to find the feature other than price that is most correlated with price.

```
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

We can Fit a linear regression model using the longitude feature `'long'` and caculate the R^2.

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

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
X = df[['sqft_living']]
Y = df['price']
lm=LinearRegression()
lm.fit(X,Y)
lm.score(X, Y)
```

```
0.49285321790379316
```

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",
```

```
for i in features :
  X = df[[i]]
  Y = df['price']
  lm=LinearRegression()
  lm.fit(X,Y)
  print(i,' : ',lm.score(X, Y))
```

```
    floors  :  0.06594310068341092
    waterfront  :  0.07095267538578309
    lat  :  0.09425113672917462
    bedrooms  :  0.09535546506131365
    sqft_basement  :  0.104856815269744
    view  :  0.15784211584121532
    bathrooms  :  0.27639993060314383
    sqft_living15  :  0.3426684607560172
    sqft_above  :  0.36671175283827917
    grade  :  0.4454684861092873
    sqft_living  :  0.49285321790379316
```

This will help with 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)),(
```

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

```
    Pipeline(memory=None,
             steps=[('scale',
                     StandardScaler(copy=True, with_mean=True, with_std=True)),
                    ('polynomial',
                     PolynomialFeatures(degree=2, include_bias=False,
                                        interaction_only=False, order='C')),
                    ('model',
                     LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                                      normalize=False))],
             verbose=False)
```

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

```
0.5327430940591443
```

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")
features =["floors", "waterfront","lat" ,"bedrooms" ,"sqft_basement" ,"view" ,"bathrooms",
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])
```

```
done
number of test samples: 3242
number of training samples: 18371
```

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.

```
from sklearn.linear_model import Ridge
pr=PolynomialFeatures(degree=2)
x_train_pr=pr.fit_transform(x_train[['floors', 'waterfront','lat' ,'bedrooms' ,'sqft_basem
x_test_pr=pr.fit_transform(x_test[['floors', 'waterfront','lat' ,'bedrooms' ,'sqft_basemen


RidgeModel=Ridge(alpha=0.1)

RidgeModel.fit(x_train_pr, y_train)
```

```
Ridge(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

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

```
0.7418167438697768
```

```
width = 12
height = 10
plt.figure(figsize=(width, height))
```

```
plt.plot(ALFA,Rsqu_test, label='validation data  ')
plt.plot(ALFA,Rsqu_train, 'r', label='training Data ')
plt.xlabel('alpha')
plt.ylabel('R^2')
plt.legend()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-57-9a6348e6c779> in <module>()
      3 plt.figure(figsize=(width, height))
      4
----> 5 plt.plot(ALFA,Rsqu_test, label='validation data  ')
      6 plt.plot(ALFA,Rsqu_train, 'r', label='training Data ')
      7 plt.xlabel('alpha')

NameError: name 'ALFA' is not defined
```

SEARCH STACK OVERFLOW

```
<Figure size 864x720 with 0 Axes>
```

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.

```
from sklearn.preprocessing import PolynomialFeatures
pr=PolynomialFeatures(degree=2)
pr

x_train_pr=pr.fit_transform(x_train[['floors', 'waterfront','lat' ,'bedrooms' ,'sqft_basem

x_polly=pr.fit_transform(x_train[['floors', 'waterfront','lat' ,'bedrooms' ,'sqft_basement

RidgeModel=Ridge(alpha=0.1)

RidgeModel.fit(x_train_pr, y_train)

print(RidgeModel.score(x_train_pr, y_train))

x_test_pr=pr.fit_transform(x_test[['floors', 'waterfront','lat' ,'bedrooms' ,'sqft_basemen

x_polly=pr.fit_transform(x_test[['floors', 'waterfront','lat' ,'bedrooms' ,'sqft_basement'

RidgeModel=Ridge(alpha=0.1)

RidgeModel.fit(x_test_pr, y_test)

RidgeModel.score(x_test_pr, y_test)
```

```
0.7418167438697768
0.7666545737131429
```