Data Analysis in Python ¶

House Sales in King County, USA

In [1]:

```
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
```

In [2]:

```
### Importing the dataset
df=pd.read_csv('https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/Cognitive
```

In [3]:

```
df
```

Out[3]:

	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
...	...	...	...	...	...	...	...
21608	21608	263000018	20140521T000000	360000.0	3.0	2.50	1530
21609	21609	6600060120	20150223T000000	400000.0	4.0	2.50	2310
21610	21610	1523300141	20140623T000000	402101.0	2.0	0.75	1020
21611	21611	291310100	20150116T000000	400000.0	3.0	2.50	1600
21612	21612	1523300157	20141015T000000	325000.0	2.0	0.75	1020

21613 rows × 22 columns

In [4]:

```python
df.head(5)
```

Out[4]:

	Unnamed: 0	id	date	price	bedrooms	bathrooms	sqft_living	sqft_
0	0	7129300520	20141013T000000	221900.0	3.0	1.00	1180	56
1	1	6414100192	20141209T000000	538000.0	3.0	2.25	2570	72
2	2	5631500400	20150225T000000	180000.0	2.0	1.00	770	100
3	3	2487200875	20141209T000000	604000.0	4.0	3.00	1960	50
4	4	1954400510	20150218T000000	510000.0	3.0	2.00	1680	80

5 rows × 22 columns

In [5]:

```python
df.tail(10)
```

Out[5]:

	Unnamed: 0	id	date	price	bedrooms	bathrooms	sqft_living
21603	21603	7852140040	20140825T000000	507250.0	3.0	2.50	2270
21604	21604	9834201367	20150126T000000	429000.0	3.0	2.00	1490
21605	21605	3448900210	20141014T000000	610685.0	4.0	2.50	2520
21606	21606	7936000429	20150326T000000	1007500.0	4.0	3.50	3510
21607	21607	2997800021	20150219T000000	475000.0	3.0	2.50	1310
21608	21608	263000018	20140521T000000	360000.0	3.0	2.50	1530
21609	21609	6600060120	20150223T000000	400000.0	4.0	2.50	2310
21610	21610	1523300141	20140623T000000	402101.0	2.0	0.75	1020
21611	21611	291310100	20150116T000000	400000.0	3.0	2.50	1600
21612	21612	1523300157	20141015T000000	325000.0	2.0	0.75	1020

10 rows × 22 columns

In [6]:

```
df.describe()
```

Out[6]:

	Unnamed: 0	id	price	bedrooms	bathrooms	sqft_living	
count	21613.00000	2.161300e+04	2.161300e+04	21600.000000	21603.000000	21613.000000	2.1
mean	10806.00000	4.580302e+09	5.400881e+05	3.372870	2.115736	2079.899736	1.5
std	6239.28002	2.876566e+09	3.671272e+05	0.926657	0.768996	918.440897	4.1
min	0.00000	1.000102e+06	7.500000e+04	1.000000	0.500000	290.000000	5.2
25%	5403.00000	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.0
50%	10806.00000	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.6
75%	16209.00000	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.0
max	21612.00000	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.6

8 rows × 21 columns

1.Question

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

In [7]:

```
df.dtypes
```

Out[7]:

```
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
```

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.

In [8]:

```
# drop columns "id" and "Unnamed: 0" from "df"
df.drop(["id", "Unnamed: 0"], axis = 1, inplace=True)
df.describe()
```

Out[8]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	
count	2.161300e+04	21600.000000	21603.000000	21613.000000	2.161300e+04	21613.000000	21
mean	5.400881e+05	3.372870	2.115736	2079.899736	1.510697e+04	1.494309	
std	3.671272e+05	0.926657	0.768996	918.440897	4.142051e+04	0.539989	
min	7.500000e+04	1.000000	0.500000	290.000000	5.200000e+02	1.000000	
25%	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	

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.

In [9]:

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

Out[9]:

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

Question 4

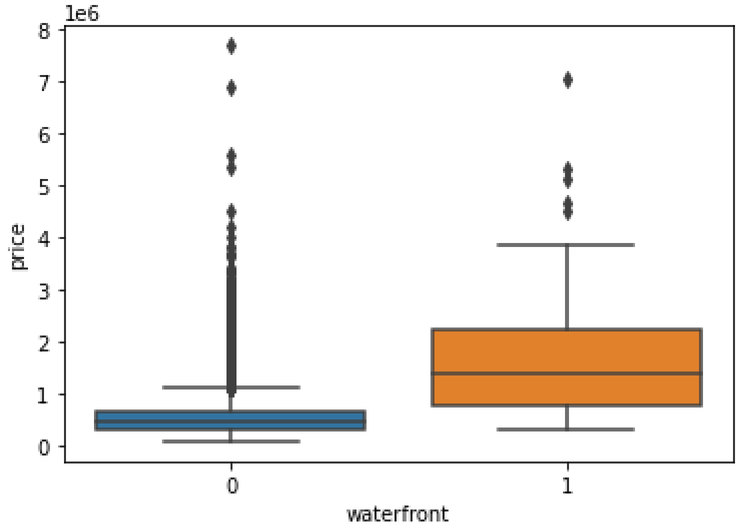
use the function boxplot in the seaborn library to produce a plot that can be used to determine whether houses with a waterfront view or without a waterfront view have more price outliers.

In [10]:

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

Out[10]:

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



Question 5

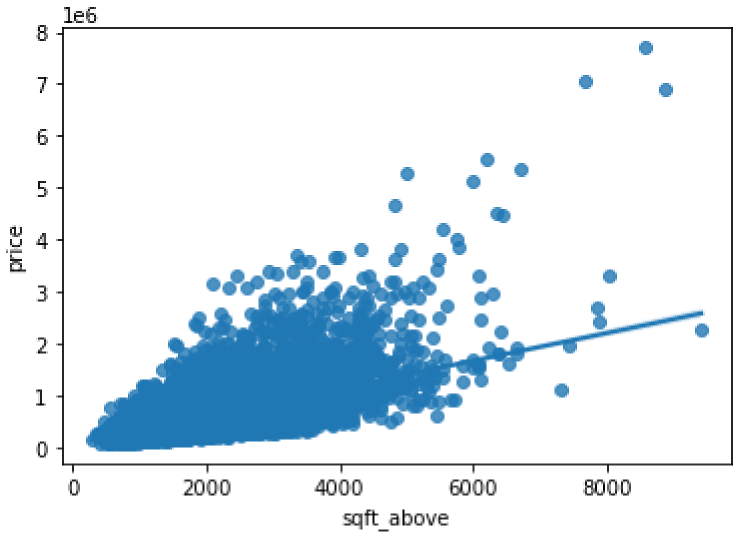
Use the function regplot in the seaborn library to determine if the feature sqft_above is negatively or positively correlated with price.

In [11]:

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

Out[11]:

```
<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 R^2.

In [12]:

```
X = df[['sqft_living']]
Y = df['price']
lm = LinearRegression()
# Fit a linear regression model using the longitude feature 'long'
lm.fit(X,Y)
# Calculate the R^2
lm.score(X, Y)
```

Out[12]:

```
0.4928532179037931
```

Question 7

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

In [13]:

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

In [14]:

```
U = df[['sqft_living']]
V = df['price']
lm.fit(U,V)
lm.score(U,V)
```

Out[14]:

```
0.4928532179037931
```

In [15]:

```
U = df[['view']]
V = df['price']
lm.fit(U,V)
lm.score(U,V)
```

Out[15]:

```
0.15784211584121532
```

Question 8

Create a pipeline object that scales the data performs a polynomial transform and fits a linear regression model. Fit the object using the features in the question above, then fit the model and calculate the R^2.

In [24]:

```
# Create a pipeline object to predict the 'price'
pipe=Pipeline(Input)
pipe
# Fit the pipeline object using a list of features
pipe.fit(U,V)
# Calculate the R^2
pipe.score(U,V)
```

Out[24]:

```
0.16178633749054383
```

In [19]:

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
print("execution done")
```

```
execution done
```

In [ ]:

```
We will split the data into training and testing sets:
```

In [20]:

```
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])
```

```
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.

In [25]:

```
from sklearn.linear_model import Ridge
```

In [26]:

```
# Create a Ridge regression object, set the regularization parameter to 0.1
RidgeModel = Ridge(alpha=0.1)
# Fit the Ridge regression object using the training data
RidgeModel.fit(x_train, y_train)
# Calculate the R^2 using the test data
RidgeModel.score(x_test, y_test)
```

Out[26]:

```
0.49100586279103864
```

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.

In [29]:

```
# Second order polynomial transform on both the training data and testing data
pr=PolynomialFeatures(degree=2,include_bias=False)
x_train_pr = pr.fit_transform(x_train)
x_test_pr = pr.fit_transform(x_test)
# Create a Ridge regression object, set the regularisation parameter to 0.1
PolyRidgeModel = Ridge(alpha=0.1)
# Fit the Ridge regression object using the training data
PolyRidgeModel.fit(x_train_pr, y_train)
# Calculate the R^2 using the test data
PolyRidgeModel.score(x_test_pr, y_test)
```

Out[29]:

```
0.4394636957768513
```

In [ ]:

```
```