Data Analysis with Python

House Sales in King County, USA

This dataset contains house sale prices for King County, which includes Seattle. It includes homes sold between May 2014 and May 2015.

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:

In [4]:

```
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

Load the csv:

In [5]:

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

We use the method `head` to display the first 5 columns of the dataframe.

In [28]:

```
df.head()
```

Out[28]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfror

0	20141013T000000	221900.0	3.0	1.00	1180	5650	1.0
1	20141209T000000	538000.0	3.0	2.25	2570	7242	2.0
2	20150225T000000	180000.0	2.0	1.00	770	10000	1.0
3	20141209T000000	604000.0	4.0	3.00	1960	5000	1.0
4	20150218T000000	510000.0	3.0	2.00	1680	8080	1.0

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.

In [11]:

```
df.dtypes
```

Out[11]:

```
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 dataframe.

In [12]:

```
df.describe()
```

Out[12]:

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

8 rows × 21 columns

Module 2: 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`

In [18]:

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

Out[18]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfror
0	20141013T000000	221900.0	3.0	1.00	1180	5650	1.0	
1	20141209T000000	538000.0	3.0	2.25	2570	7242	2.0	

2	20150225T000000	180000.0	2.0	1.00	770	10000	1.0
3	20141209T000000	604000.0	4.0	3.00	1960	5000	1.0
4	20150218T000000	510000.0	3.0	2.00	1680	8080	1.0

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

In [19]:

```
print("number of NaN values for the column bedrooms :", df['bedrooms'].isnull
print("number of NaN values for the column bathrooms :", df['bathrooms'].isnu
```

```
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`

In [20]:

```
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`

In [21]:

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

In [22]:

```
print("number of NaN values for the column bedrooms :", df['bedrooms'].isnull
print("number of NaN values for the column bathrooms :", df['bathrooms'].isnu
```

```
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.

In [26]:

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

Out[26]:

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

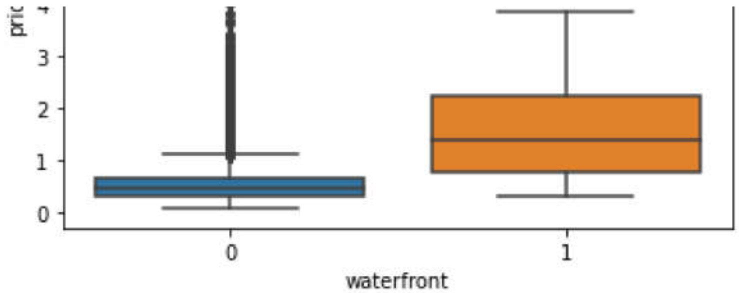
Question 4

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.

In [31]:

```
# import visualization packages "Matplotlib" and "seaborn"
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# relationship between "houses with or without waterfront view" and "price"
sns.boxplot(x='waterfront', y='price', data=df)
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x7f271180d2d0>



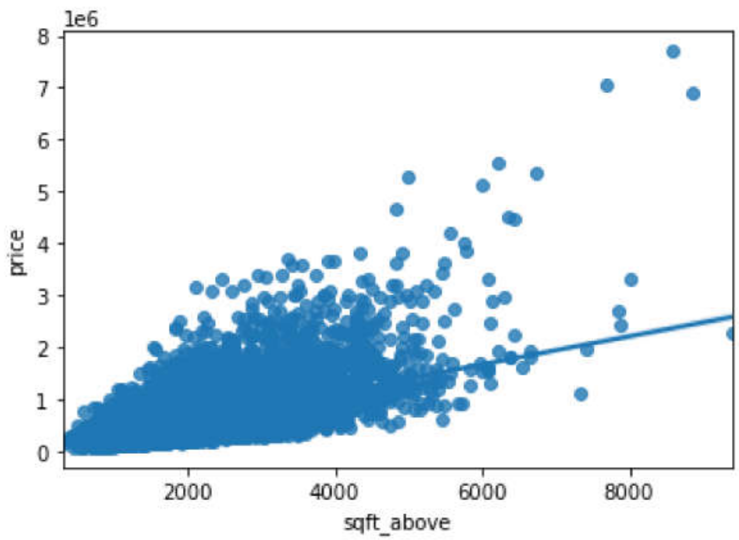
Question 5

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

In [33]:

```
# determining correlation of "sqft_above" with "price"
sns.regplot(x='sqft_above', y='price', data=df)
```

Out[33]: `<matplotlib.axes._subplots.AxesSubplot at 0x7f271174ab10>`



We can see that there is a strong correlation between `sqft_above` and `price`. The higher the number of `sqft_above`, the higher the `price` of houses sold.

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

In [34]:

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

Out[34]:

```
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 calculate the R^2.

In [36]:

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

Out[36]:

```
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.

In [40]:
```
# How can "sqft_living" help us predict house "price"
X = df[['sqft_living']]
Y = df[['price']]
lm = LinearRegression()
lm.fit(X,Y)
print('The R-square value is: ', lm.score(X, Y))
```

```
The R-square value is:  0.4928532179037931
```

Question 7

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

In [45]:
```
features = df[["floors", "waterfront", "lat", "bedrooms", "sqft_basement", "v

# linear regression model to predict "price" using "list of features"
lm.fit(features, df[['price']])
```

Out[45]:
```
LinearRegression()
```

Then calculate the R^2. Take a screenshot of your code.

In [46]:
```
# the R-square value
print('The R-square value is: ', lm.score(features, df[['price']]))
```

```
The R-square value is:  0.657679183672129
```

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()
```

In [48]:

```
# create data pipelines
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

Input=[('scale',StandardScaler()), ('polynomial', PolynomialFeatures(include_
```

Question 8

Use the list to create a pipeline object to predict the 'price', fit the object using the features in the list `features`, and calculate the R^2.

In [54]:

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

# the R-square value
print('The R-square value is: ', pipe.score(features, df[['price']]))
```

```
The R-square value is:  0.7513408553309376
```

Module 5: Model Evaluation and Refinement

Import the necessary modules:

In [55]:

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

```
done
```

We will split the data into training and testing sets:

In [56]:

```
features =["floors", "waterfront","lat" ,"bedrooms" ,"sqft_basement" ,"view"
X = df[features]
Y = df['price']

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.15, ran


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, set the regularization parameter to 0.1, and calculate the R^2 using the test data.

In [59]:

```
# import Ridge from the module linear_model
from sklearn.linear_model import Ridge
```

In [63]:

```
# create Ridge regression object and set regularization parameter to 0.1
RigeModel = Ridge(alpha=0.1)

# fit Ridge regression object using training data
RigeModel.fit(x_train, y_train)

# calculated R^2 using test data
RigeModel.score(x_test, y_test)
```

Out[63]:

```
0.6478759163939122
```

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.

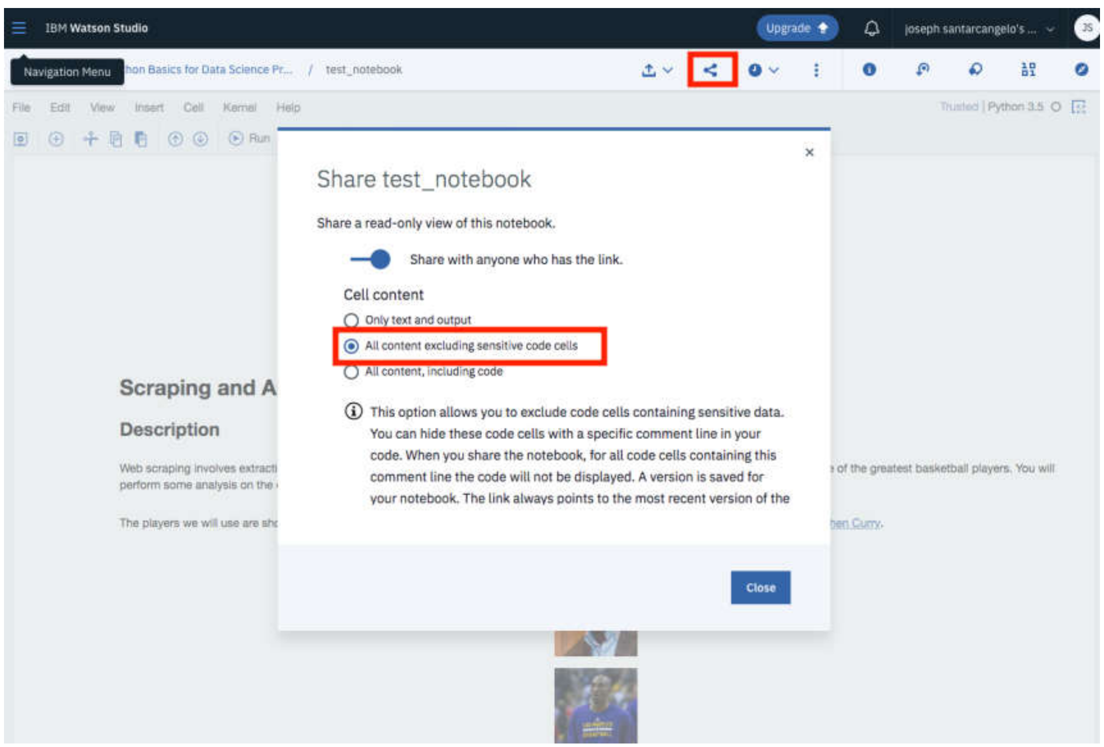
In [62]:

```
# perform a second order polynomial transform on both the training data and t
pr=PolynomialFeatures(degree=2)
x_train_pr=pr.fit_transform(x_train[["floors", "waterfront","lat" ,"bedrooms"
x_test_pr=pr.fit_transform(x_test[["floors", "waterfront","lat" ,"bedrooms" ,

# create Ridge regression object and set regularization parameter to 0.1
RigeModel = Ridge(alpha=0.1)

# fit Ridge regression object using training data
RigeModel.fit(x_train_pr, y_train)

# calculated R^2 using test data
RigeModel.score(x_test_pr, y_test)
```

Out[62]:

```
0.7002744279896707
```

Once you complete your notebook you will have to share it. Select the icon on the top right a marked in red in the image below, a dialogue box should open, and select the option all content excluding sensitive code cells.



You can then share the notebook via a URL by scrolling down as shown in the following image: