House Sales – Price Predictions

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/House

bathrooms: Number of bathrooms/bedrooms

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 [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
%matplotlib inline
```

1.0 Importing the Data

Load the csv:

In [2]:
```
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)
```

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

In [3]:
```
df.head()
```
Out[3]:

	Unnamed: 0	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
0	0	7129300520	20141013T000000	221900.0	3.0	1.0	1180	5650	1.0	0	...	7	1180	0	1955	0	98178	47.5112	-122.257	1340	5650
1	1	6414100192	20141209T000000	538000.0	3.0	2.25	2570	7242	2.0	0	...	7	2170	400	1951	1991	98125	47.7210	-122.319	1690	7639

	Unnamed: 0	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
2	2	5631500400	20150225T000000	180000.0	2.0	1.00	770	10000	1.0	0	...	6	770	0	1933	0	98028	47.7379	-122.233	2720	8062
3	3	2487200875	20141209T000000	604000.0	4.0	3.00	1960	5000	1.0	0	...	7	1050	910	1965	0	98136	47.5208	-122.393	1360	5000
4	4	1954400510	20150218T000000	510000.0	3.0	2.00	1680	8080	1.0	0	...	8	1680	0	1987	0	98074	47.6168	-122.045	1800	7503

5 rows × 22 columns

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 [5]:

```
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
```

We use the method describe to obtain a statistical summary of the dataframe.

In [6]:

```
df.describe()
```

Out[6]:

	Unnamed: 0	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
count	21613.000000	2.161300e+04	2.161300e+04	21600.000000	21603.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	...	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000
mean	10806.000000	4.580302e+09	5.400881e+05	3.372870	2.115736	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	...	7.656873	1788.390691	291.509045	1971.005136	84.402258	98077.939805	47.560053	-122.213896	1986.552492	12768.455652
std	6239.280002	2.876566e+09	3.671272e+05	0.926657	0.768996	918.440897	4.142051e+04	0.539989	0.086517	0.766318	...	1.175459	828.090978	442.575043	29.373411	401.679240	53.505026	0.138564	0.140828	685.391304	27304.179631
min	0.000000	1.000102e+06	7.500000e+04	1.000000	0.500000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	...	1.000000	290.000000	0.000000	1900.000000	0.000000	98001.000000	47.155900	-122.519000	399.000000	651.000000

	Unnamed: 0	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
25%	5403.00000	2.123049e+09	3.219500e+05	3.00000	1.75000	1427.000000	5.040000e+03	1.00000	0.00000	0.00000	...	7.00000	1190.000000	0.000000	1951.000000	0.00000	98033.00000	47.471000	-122.32800	1490.000000	5100.000000
50%	10806.0000	3.904930e+09	4.500000e+05	3.00000	2.25000	1910.000000	7.618000e+03	1.50000	0.00000	0.00000	...	7.00000	1560.000000	0.000000	1975.000000	0.00000	98065.00000	47.571800	-122.23000	1840.000000	7620.000000
75%	16209.0000	7.308900e+09	6.450000e+05	4.00000	2.50000	2550.000000	1.068800e+04	2.00000	0.00000	0.00000	...	8.00000	2210.000000	560.000000	1997.000000	0.00000	98118.00000	47.678000	-122.12500	2360.000000	10083.000000
max	21612.0000	9.900000e+09	7.700000e+06	33.000000	8.00000	13540.00000	1.651359e+06	3.50000	1.00000	4.00000	...	13.000000	9410.000000	4820.000000	2015.000000	2015.000000	98199.00000	47.777600	-121.31500	6210.000000	871200.000000

8 rows × 21 columns

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`

In [7]:

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

Out[7]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
count	2.161300e+04	21600.000000	21603.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000
mean	5.400881e+05	3.372870	2.115736	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	3.409430	7.656873	1788.390691	291.509045	1971.005136	84.402258	98077.939805	47.560053	-122.213896	1986.552492	12768.455652
std	3.671272e+05	0.926657	0.768996	918.440897	4.142051e+04	0.539989	0.086517	0.766318	0.650743	1.175459	828.090978	442.575043	29.373411	401.679240	53.505026	0.138564	0.140828	685.391304	27304.179631
min	7.500000e+04	1.000000	0.500000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	1.000000	1.000000	290.000000	0.000000	1900.000000	0.000000	98001.000000	47.155900	-122.519000	399.000000	651.000000
25%	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	3.000000	7.000000	1190.000000	0.000000	1951.000000	0.000000	98033.000000	47.471000	-122.328000	1490.000000	5100.000000
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	3.000000	7.000000	1560.000000	0.000000	1975.000000	0.000000	98065.000000	47.571800	-122.230000	1840.000000	7620.000000
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	4.000000	8.000000	2210.000000	560.000000	1997.000000	0.000000	98118.000000	47.678000	-122.125000	2360.000000	10083.000000
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	5.000000	13.000000	9410.000000	4820.000000	2015.000000	2015.000000	98199.000000	47.777600	-121.315000	6210.000000	871200.000000

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

In [8]:
```
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 top `True`

In [9]:
```
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 `'bedrooms' using the method replace.Don't forget to set the inplace parameter top Ture`

In [10]:
```
mean=df['bathrooms'].mean()
df['bathrooms'].replace(np.nan,mean, inplace=True)
```
In [11]:
```
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.

In [12]:
```
df['floors'].value_counts().to_frame()
```
Out[12]:

	floors
1.0	10680
2.0	8241

	floors
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 [14]:

```
sns.boxplot(x='waterfront', y='price', data=df)
/opt/conda/envs/DSX-Python35/lib/python3.5/site-packages/seaborn/categorical.py:462: FutureWarning: remove_na is deprecated and is a private function. Do not use.
  box_data = remove_na(group_data)
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f24f8d0b208>
```

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 [15]:

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

Out[15]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f24f8ca7d30>
```

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

In [16]:

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

Out[16]:

```
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

In [17]:
```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

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

In [18]:
```
X = df[['long']]
Y = df['price']
lm = LinearRegression()
lm
lm.fit(X,Y)
lm.score(X, Y)
```
Out[18]:
```
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 [20]:
```
X = df[['sqft_living']]
Y = df['price']
lm = LinearRegression()
lm.fit(X, Y)
lm.score(X, Y)
```
Out[20]:
```
0.4928532179037931
```

Question 7

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

In [21]:
```
features =["floors", "waterfront","lat" ,"bedrooms"
,"sqft_basement" ,"view"
,"bathrooms","sqft_living15","sqft_above","grade","sqft_living"]
```

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

In [22]:
```
X = df[features]
Y= df['price']
lm = LinearRegression()
lm.fit(X, Y)
lm.score(X, Y)
```
Out[22]:
```
0.65769516660374948
```

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 [23]:
```
Input=[('scale',StandardScaler()),('polynomial',
PolynomialFeatures(include_bias=False)),('model',LinearRegressio
n())]
```

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

In [24]:
```
pipe=Pipeline(Input)
pipe
```
Out[24]:
```
Pipeline(memory=None,
     steps=[('scale', StandardScaler(copy=True, with_mean=True, with_std=True)), ('polynomial', PolynomialFeatures(degree=2, include_bias=False, interaction_only=False)), ('model', LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False))])
```
In [25]:
```
pipe.fit(X,Y)
```
Out[25]:
```
Pipeline(memory=None,
     steps=[('scale', StandardScaler(copy=True, with_mean=True, with_std=True)), ('polynomial', PolynomialFeatures(degree=2, include_bias=False, interaction_only=False)), ('model', LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False))])
```
In [26]:
```
pipe.score(X,Y)
```
Out[26]:
```
0.75134126473712171
```

Module 5: MODEL EVALUATION AND REFINEMENT

import the necessary modules

In [27]:
```
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 set

In [28]:
```
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.

In [29]:

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

In [30]:

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

Out[30]:

```
0.6478759163939107
```

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.

In [34]:

```
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)
```

Out[34]:

```
0.7002744243688905
```

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, select the option all content excluding sensitive code cells.