House Sales in King County, USA

March 22, 2020

[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

[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)
```

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

[3]:
```
   Unnamed: 0          id             date     price  bedrooms  bathrooms  \
0           0  7129300520  20141013T000000  221900.0       3.0       1.00
1           1  6414100192  20141209T000000  538000.0       3.0       2.25
2           2  5631500400  20150225T000000  180000.0       2.0       1.00
3           3  2487200875  20141209T000000  604000.0       4.0       3.00
4           4  1954400510  20150218T000000  510000.0       3.0       2.00

   sqft_living  sqft_lot  floors  waterfront  ...  grade  sqft_above  \
0         1180      5650     1.0           0  ...      7        1180
1         2570      7242     2.0           0  ...      7        2170
2          770     10000     1.0           0  ...      6         770
3         1960      5000     1.0           0  ...      7        1050
4         1680      8080     1.0           0  ...      8        1680

   sqft_basement  yr_built  yr_renovated  zipcode      lat     long  \
0              0      1955             0    98178  47.5112 -122.257
1            400      1951          1991    98125  47.7210 -122.319
2              0      1933             0    98028  47.7379 -122.233
3            910      1965             0    98136  47.5208 -122.393
4              0      1987             0    98074  47.6168 -122.045

   sqft_living15  sqft_lot15
0           1340        5650
```

```
1           1690        7639
2           2720        8062
3           1360        5000
4           1800        7503

[5 rows x 22 columns]
```

[4]:
```
df.dtypes
```

[4]:
```
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
```

[5]:
```
df.describe()
```

[5]:
```
        Unnamed: 0            id         price      bedrooms     bathrooms  \
count  21613.00000  2.161300e+04  2.161300e+04  21600.000000  21603.000000
mean   10806.00000  4.580302e+09  5.400881e+05      3.372870      2.115736
std     6239.28002  2.876566e+09  3.671272e+05      0.926657      0.768996
min        0.00000  1.000102e+06  7.500000e+04      1.000000      0.500000
25%     5403.00000  2.123049e+09  3.219500e+05      3.000000      1.750000
50%    10806.00000  3.904930e+09  4.500000e+05      3.000000      2.250000
75%    16209.00000  7.308900e+09  6.450000e+05      4.000000      2.500000
max    21612.00000  9.900000e+09  7.700000e+06     33.000000      8.000000

        sqft_living      sqft_lot        floors    waterfront          view  \
count  21613.000000  2.161300e+04  21613.000000  21613.000000  21613.000000
mean    2079.899736  1.510697e+04      1.494309      0.007542      0.234303
std      918.440897  4.142051e+04      0.539989      0.086517      0.766318
```

```
min      290.000000  5.200000e+02     1.000000     0.000000     0.000000
25%     1427.000000  5.040000e+03     1.000000     0.000000     0.000000
50%     1910.000000  7.618000e+03     1.500000     0.000000     0.000000
75%     2550.000000  1.068800e+04     2.000000     0.000000     0.000000
max    13540.000000  1.651359e+06     3.500000     1.000000     4.000000

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

       yr_renovated       zipcode           lat          long  sqft_living15  \
count  21613.000000  21613.000000  21613.000000  21613.000000   21613.000000
mean      84.402258  98077.939805     47.560053   -122.213896    1986.552492
std      401.679240     53.505026      0.138564      0.140828     685.391304
min        0.000000  98001.000000     47.155900   -122.519000     399.000000
25%        0.000000  98033.000000     47.471000   -122.328000    1490.000000
50%        0.000000  98065.000000     47.571800   -122.230000    1840.000000
75%        0.000000  98118.000000     47.678000   -122.125000    2360.000000
max     2015.000000  98199.000000     47.777600   -121.315000    6210.000000

          sqft_lot15
count   21613.000000
mean    12768.455652
std     27304.179631
min       651.000000
25%      5100.000000
50%      7620.000000
75%     10083.000000
max    871200.000000

[8 rows x 21 columns]
```

2.0 Data Wrangling

[6]:
```python
df.drop(['id','Unnamed: 0'], axis = 1, inplace = True)
df.describe()
```

[6]:
```
              price      bedrooms     bathrooms   sqft_living      sqft_lot  \
count  2.161300e+04  21600.000000  21603.000000  21613.000000  2.161300e+04
mean   5.400881e+05      3.372870      2.115736   2079.899736  1.510697e+04
std    3.671272e+05      0.926657      0.768996    918.440897  4.142051e+04
min    7.500000e+04      1.000000      0.500000    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
```

[7]:
```python
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
```

[8]:
```python
#replace the missing values of the column 'bedrooms' with the mean of the␣
 ↪column 'bedrooms' using the method replace.
mean=df['bedrooms'].mean()
df['bedrooms'].replace(np.nan,mean, inplace=True)
```

[9]:
```
mean=df['bathrooms'].mean()
df['bathrooms'].replace(np.nan,mean, inplace=True)
```

[10]:
```
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

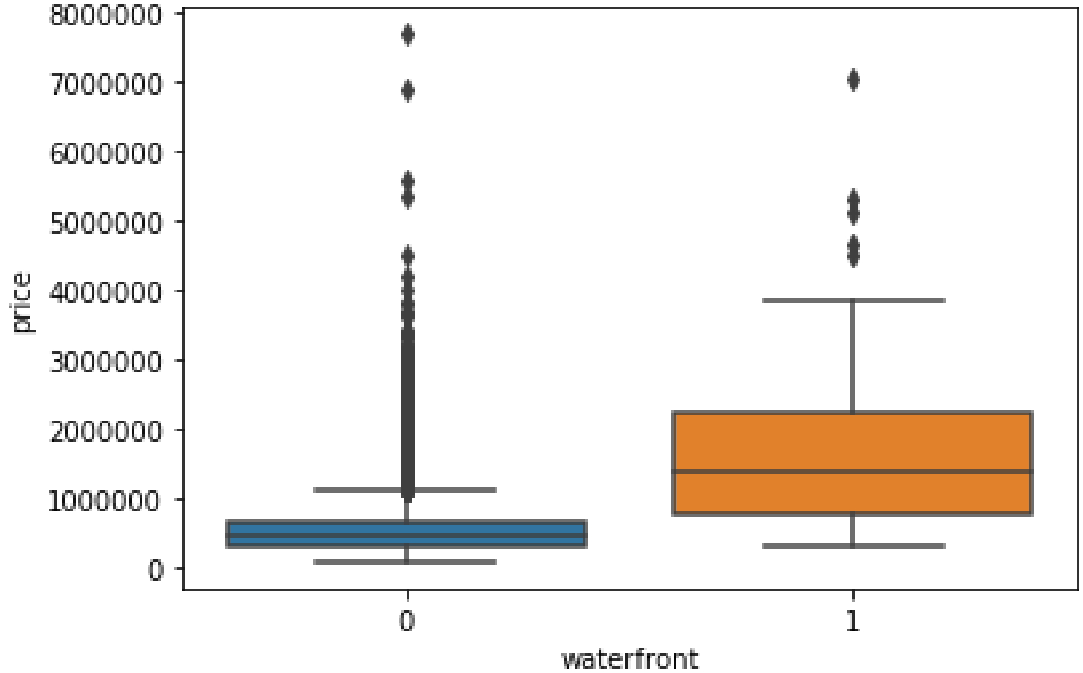
[11]:
```
#Use the method value_counts to count the number of houses with unique floor␣
↪values.
floor_values = df['floors'].value_counts()
floor_values.to_frame()
```

[11]:

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

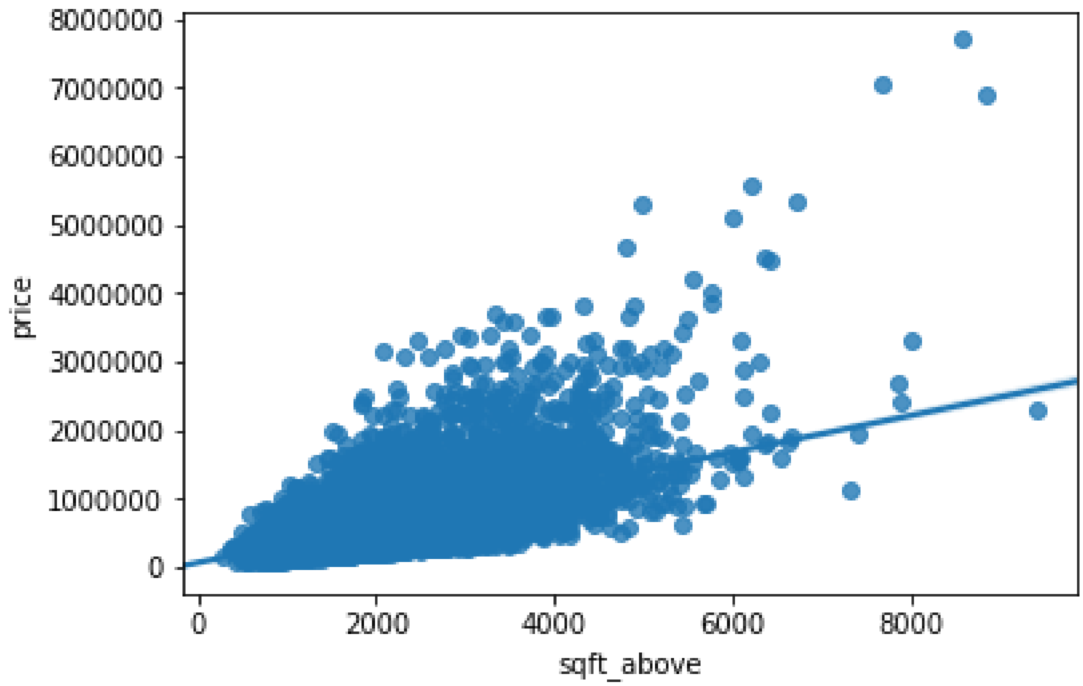
[12]:
```
#Use the function boxplotto determine whether houses with a waterfront view or␣
↪without have more price outliers.
sns.boxplot(x=df['waterfront'], y = df['price'])
# So houses with a waterfront has more price outliers
```

[12]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1e381d30>



[13]:
```
#Use the function  regplot library to determine if the feature sqft_above is␣
↪negatively or positively correlated with price.
sns.regplot(x=df['sqft_above'], y=df['price'])
# they are positively correlated with price
```

[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1f40e438>



[14]:
```
df.corr()['price'].sort_values()
```

[14]:
```
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

[ ]:
```
#Linear Regression
```

[15]:
```python
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

[20]:
```python
features =["floors", "waterfront","lat" ,"bedrooms" ,"sqft_basement" ,"view"␣
↪,"bathrooms","sqft_living15","sqft_above","grade","sqft_living"]
```

[21]:
```python
features = df[features]
y1 = df['price']
```

[22]:
```python
lm = LinearRegression()
lm.fit(features, y1)
lm.score(features, y1)
```

[22]:
```
0.657679183672129
```

[23]:
```python
#create a pipeline to build polynomial regression
```

[24]:
```python
Input=[('scale',StandardScaler()),('polynomial',␣
↪PolynomialFeatures(include_bias=False)),('model',LinearRegression())]
```

[25]:
```python
pipe=Pipeline(Input)
pipe
```

[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, order='C')),
                ('model',
                 LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                                  normalize=False))],
         verbose=False)
```

[26]:
```python
pipe.fit(features,y1)
```

[26]:
```
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)
```

[27]:
```python
pipe.score(features, y1)
```

[27]:
```
0.7513408553309376
```

[28]:
```python
# We can see polynomial regression has better performance than linear␣
↪regression
```

[29]:
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
```

[32]:
```
#split data to training and testing
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
```

[33]:
```
#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.
from sklearn.linear_model import Ridge
Ridgemodel = Ridge(alpha=0.1)
Ridgemodel.fit(x_train, y_train)
Ridgemodel.score(x_test, y_test)
```

[33]: 0.6478759163939121

[34]:
```
#Perform a second order polynomial transform
pr = PolynomialFeatures(degree = 2)
x_train_pr = pr.fit_transform(x_train)
x_test_pr = pr.fit_transform(x_test)
Ridgemodel_pr = Ridge(alpha=0.1)
Ridgemodel_pr.fit(x_train_pr, y_train)
Ridgemodel_pr.score(x_test_pr, y_test)
```

[34]: 0.7002744279699229

[35]:
```
#To choose the best alpha
from sklearn.model_selection import GridSearchCV
```

[36]:
```
parameter2 = [{'alpha':[0.01,0.1,1,10,100],'normalize':[True,False]}]
RR = Ridge()
Grid1 = GridSearchCV(RR, parameter2, cv=4)
Grid1.fit(x_train_pr, y_train)
Grid1.best_estimator_
```

```
//anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:147:
LinAlgWarning: Ill-conditioned matrix (rcond=1.85665e-17): result may not be
```

```
accurate.
  overwrite_a=True).T
//anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:147:
LinAlgWarning: Ill-conditioned matrix (rcond=2.27951e-17): result may not be
accurate.
  overwrite_a=True).T
//anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:147:
LinAlgWarning: Ill-conditioned matrix (rcond=1.99368e-17): result may not be
accurate.
  overwrite_a=True).T
//anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:147:
LinAlgWarning: Ill-conditioned matrix (rcond=1.48596e-17): result may not be
accurate.
  overwrite_a=True).T
```

[36]:
```
Ridge(alpha=0.01, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

[37]:
```python
# we use the best alpha '0.01' to build another model
pr = PolynomialFeatures(degree = 2)
x_train_pr = pr.fit_transform(x_train)
x_test_pr = pr.fit_transform(x_test)
Ridgemodel_pr = Ridge(alpha=0.01)
Ridgemodel_pr.fit(x_train_pr, y_train)
Ridgemodel_pr.score(x_test_pr, y_test)
```

[37]:
```
0.7017148826621857
```

[ ]:
```python
# We get the largest R^2 here.
# Here 70.17% data can be explained with this new model.
```