House_Sales_in_King_Count_USA

June 17, 2022

Data Analysis with Python

1 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.

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:

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler,PolynomialFeatures
from sklearn.linear_model import LinearRegression
%matplotlib inline
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/utils/validation.py:37: DeprecationWarning: distutils Version
classes are deprecated. Use packaging.version instead.
  LARGE_SPARSE_SUPPORTED = LooseVersion(scipy_version) >= '0.14.0'
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/linear_model/least_angle.py:35: DeprecationWarning: `np.float`
is a deprecated alias for the builtin `float`. To silence this warning, use
`float` by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  eps=np.finfo(np.float).eps,
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/linear_model/least_angle.py:597: DeprecationWarning: `np.float`
is a deprecated alias for the builtin `float`. To silence this warning, use
`float` by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  eps=np.finfo(np.float).eps, copy_X=True, fit_path=True,
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/linear_model/least_angle.py:836: DeprecationWarning: `np.float`
is a deprecated alias for the builtin `float`. To silence this warning, use
`float` by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  eps=np.finfo(np.float).eps, copy_X=True, fit_path=True,
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/linear_model/least_angle.py:862: DeprecationWarning: `np.float`
is a deprecated alias for the builtin `float`. To silence this warning, use
`float` by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  eps=np.finfo(np.float).eps, positive=False):
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/linear_model/least_angle.py:1097: DeprecationWarning:
`np.float` is a deprecated alias for the builtin `float`. To silence this
warning, use `float` by itself. Doing this will not modify any behavior and is
safe. If you specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
```

```
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  max_n_alphas=1000, n_jobs=None, eps=np.finfo(np.float).eps,
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/linear_model/least_angle.py:1344: DeprecationWarning:
`np.float` is a deprecated alias for the builtin `float`. To silence this
warning, use `float` by itself. Doing this will not modify any behavior and is
safe. If you specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  max_n_alphas=1000, n_jobs=None, eps=np.finfo(np.float).eps,
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/linear_model/least_angle.py:1480: DeprecationWarning:
`np.float` is a deprecated alias for the builtin `float`. To silence this
warning, use `float` by itself. Doing this will not modify any behavior and is
safe. If you specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  eps=np.finfo(np.float).eps, copy_X=True, positive=False):
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/linear_model/randomized_l1.py:152: DeprecationWarning:
`np.float` is a deprecated alias for the builtin `float`. To silence this
warning, use `float` by itself. Doing this will not modify any behavior and is
safe. If you specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  precompute=False, eps=np.finfo(np.float).eps,
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/linear_model/randomized_l1.py:320: DeprecationWarning:
`np.float` is a deprecated alias for the builtin `float`. To silence this
warning, use `float` by itself. Doing this will not modify any behavior and is
safe. If you specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  eps=np.finfo(np.float).eps, random_state=None,
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/linear_model/randomized_l1.py:580: DeprecationWarning:
`np.float` is a deprecated alias for the builtin `float`. To silence this
warning, use `float` by itself. Doing this will not modify any behavior and is
safe. If you specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  eps=4 * np.finfo(np.float).eps, n_jobs=None,
```

2 Module 1: Importing Data Sets

Load the csv:

[2]:
```
file_name='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
↪IBMDeveloperSkillsNetwork-DA0101EN-SkillsNetwork/labs/FinalModule_Coursera/
↪data/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.

[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]
```

2.0.1 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.

[4]:
```
datatype = df.dtypes
print(datatype)
```

```
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.

[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]
```

3 Module 2: Data Wrangling

3.0.1 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

[6]:
```
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
```

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

[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 : 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

[7]:
```
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

[8]:
```
mean=df['bathrooms'].mean()
df['bathrooms'].replace(np.nan,mean, inplace=True)
```

[9]:
```
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
```

4 Module 3: Exploratory Data Analysis

4.0.1 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.

[10]:
```
df['floors'].value_counts().to_frame()
```

[10]:

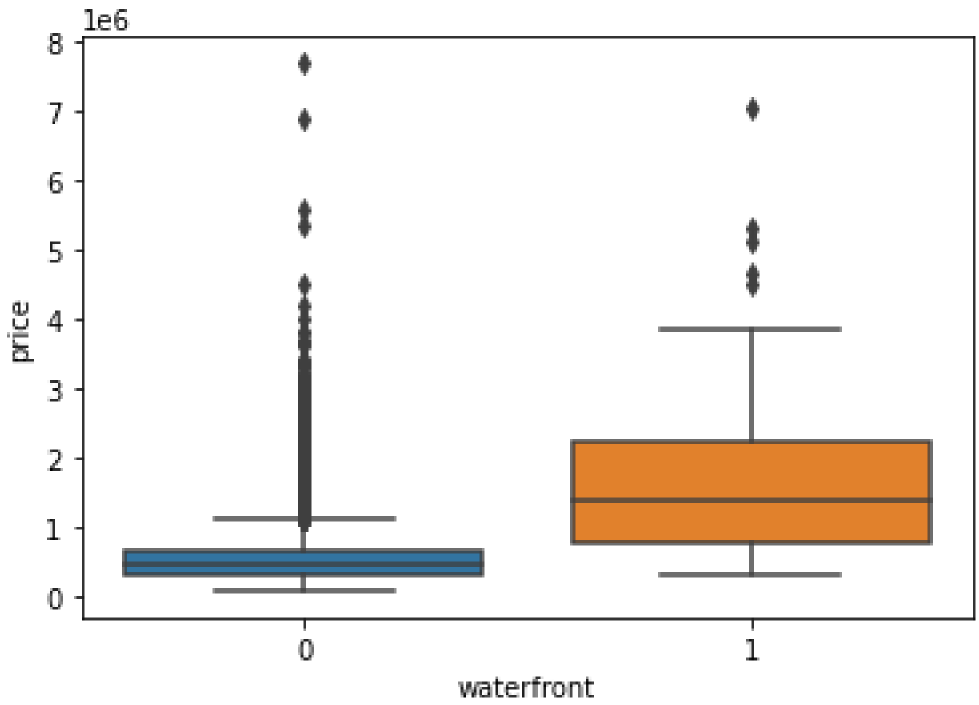
	floors
1.0	10680
2.0	8241
1.5	1910
3.0	613
2.5	161
3.5	8

4.0.2 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.

[11]:
```
sns.boxplot(x='waterfront', y='price', data=df)
```

[11]:
```
<AxesSubplot:xlabel='waterfront', ylabel='price'>
```

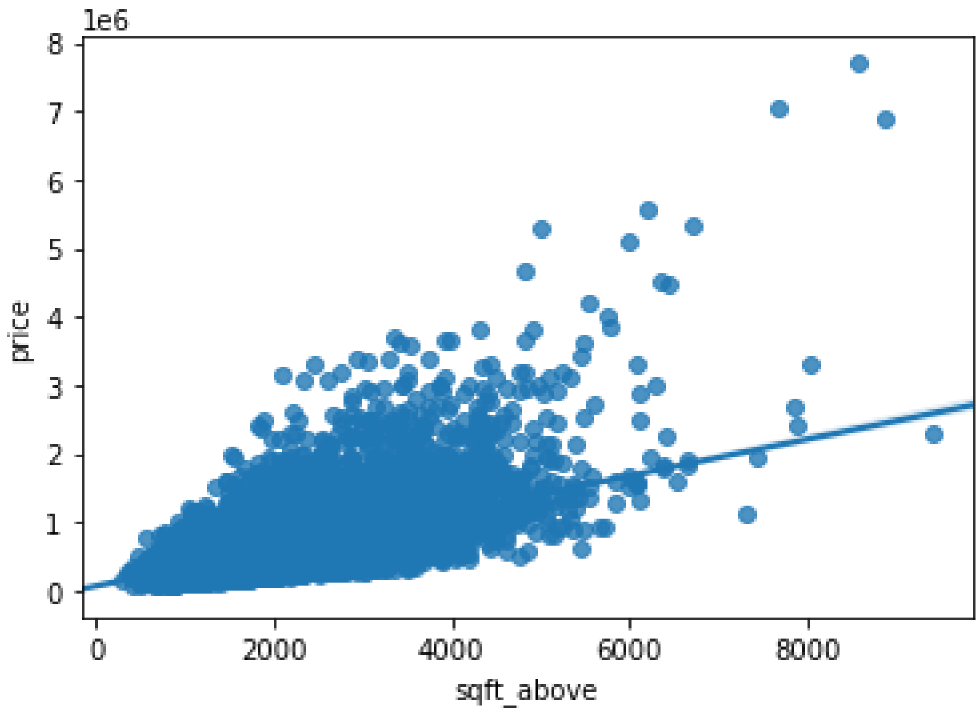


4.0.3 Question 5

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

[12]:
```
sns.regplot(x = "sqft_above",
            y = "price",
            data = df)
```

[12]: <AxesSubplot:xlabel='sqft_above', ylabel='price'>



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

[13]:
```
df.corr()['price'].sort_values()
```

[13]:
```
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
```

5 Module 4: Model Development

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

[14]:
```
X = df[['long']]
Y = df['price']
lm = LinearRegression()
lm.fit(X,Y)
lm.score(X, Y)
```

[14]:
```
0.00046769430149029567
```

5.0.1 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.

[15]:
```
X = df[['sqft_living']]
Y = df['price']
lm = LinearRegression()
lm.fit(X,Y)
lm.score(X, Y)
```

[15]:
```
0.49285321790379316
```

5.0.2 Question 7

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

[16]:
```
features =["floors", "waterfront","lat" ,"bedrooms" ,"sqft_basement" ,"view"␣
 ↪,"bathrooms","sqft_living15","sqft_above","grade","sqft_living"]
```

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

[17]:
```
#df['bedrooms'].describe()

#df

#df.fillna(df.mean())

X = df[["floors", "waterfront","lat" ,"bedrooms" ,"sqft_basement" ,"view"␣
 ↪,"bathrooms","sqft_living15","sqft_above","grade","sqft_living"]]
Y = df['price']
lm.fit (X, df['price'])
lm.score(X, Y)
```

[17]: 0.6576951666037498

5.0.3 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()

[18]:
```
Input=[('scale',StandardScaler()),('polynomial',␣
↪PolynomialFeatures(include_bias=False)),('model',LinearRegression())]
```

5.0.4 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.

[19]:
```
X = df[["floors", "waterfront","lat" ,"bedrooms" ,"sqft_basement" ,"view"␣
↪,"bathrooms","sqft_living15","sqft_above","grade","sqft_living"]]
Y = df['price']

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

pipe=Pipeline(Input)
pipe.fit(X,Y)
pipe.score(X,Y)
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/utils/validation.py:209: DeprecationWarning: distutils Version
classes are deprecated. Use packaging.version instead.
  if LooseVersion(joblib_version) < '0.12':
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/preprocessing/data.py:625: DataConversionWarning: Data with
input dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/base.py:465: DataConversionWarning: Data with input dtype
int64, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, y, **fit_params).transform(X)
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/pipeline.py:511: DataConversionWarning: Data with input dtype
int64, float64 were all converted to float64 by StandardScaler.
  Xt = transform.transform(Xt)
```

[19]:
```
0.751339641572321
```

6 Module 5: Model Evaluation and Refinement

Import the necessary modules:

[20]:
```python
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:

[21]:
```python
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
```

6.0.1 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.

[22]:
```python
from sklearn.linear_model import Ridge
```

[23]:
```python
RidgeModel=Ridge(alpha=0.1)
RidgeModel.fit(x_train, y_train)
RidgeModel.score(x_test, y_test)
```

[23]:
```
0.647875916393911
```

6.0.2 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.

[24]:
```
pf = PolynomialFeatures(degree=2)
x_train_pf = pf.fit_transform(x_train)
x_test_pf = pf.fit_transform(x_test)

RigeModel=Ridge(alpha=0.1)
RigeModel.fit(x_train_pf, y_train)
RigeModel.score(x_test_pf, y_test)
```

[24]:
```
0.7002744261580325
```

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.

```
<p><img width="600" src="https://cf-courses-data.s3.us.cloud-object-storage.appdomain.clou
<p></p>
<p>You can then share the notebook  via a  URL by scrolling down as shown in the
<p style="text-align: center;"><img width="600"  src="https://cf-courses-data.s3.us.cloud-
<p> </p>
```

About the Authors:

Joseph Santarcangelo has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Other contributors: Michelle Carey, Mavis Zhou

6.1 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-12-01	2.2	Aije Egwaikhide	Coverted Data describtion from text to table
2020-10-06	2.1	Lakshmi Holla	Changed markdown instruction of Question1
2020-08-27	2.0	Malika Singla	Added lab to GitLab

##

[ ]: