Module5_Linear_Regression_Interactions_and_Transformations

August 11, 2024

0.1 Dempsey Wade

1 Module 5: Linear Regression - Interactions and Transformations

Author:Favio Vázquez and Jessica Cervi

In this assignment, we will perform some feature transformation on a database describing airplane accidents and next, we will study a complete example of linear regression.

Index:

- Question 1
- Question 2
- Question 3
- Question 4
- Question 5
- Question 6
- Question 7
- Question 8
- Question 9
- Question 10
- Question 11
- Question 12

[1]:
```
!pip install yellowbrick
```

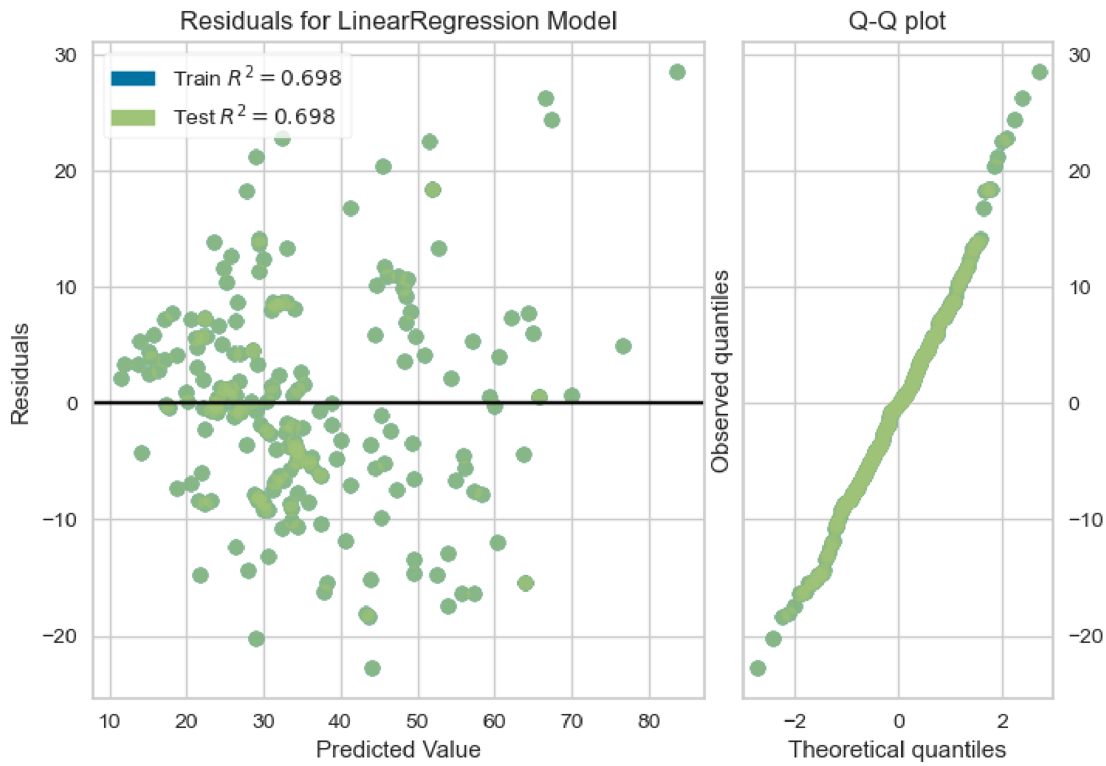
```
Requirement already satisfied: yellowbrick in
/Users/dempseywade/opt/anaconda3/lib/python3.9/site-packages (1.5)
Requirement already satisfied: scipy>=1.0.0 in
/Users/dempseywade/opt/anaconda3/lib/python3.9/site-packages (from yellowbrick)
(1.9.1)
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.2 in
/Users/dempseywade/opt/anaconda3/lib/python3.9/site-packages (from yellowbrick)
(3.5.2)
Requirement already satisfied: cycler>=0.10.0 in
/Users/dempseywade/opt/anaconda3/lib/python3.9/site-packages (from yellowbrick)
(0.11.0)
Requirement already satisfied: scikit-learn>=1.0.0 in
/Users/dempseywade/opt/anaconda3/lib/python3.9/site-packages (from yellowbrick)
(1.0.2)
```

```
Requirement already satisfied: numpy>=1.16.0 in
/Users/dempseywade/opt/anaconda3/lib/python3.9/site-packages (from yellowbrick)
(1.21.5)
Requirement already satisfied: kiwisolver>=1.0.1 in
/Users/dempseywade/opt/anaconda3/lib/python3.9/site-packages (from
matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.4.2)
Requirement already satisfied: pyparsing>=2.2.1 in
/Users/dempseywade/opt/anaconda3/lib/python3.9/site-packages (from
matplotlib!=3.0.0,>=2.0.2->yellowbrick) (3.0.9)
Requirement already satisfied: packaging>=20.0 in
/Users/dempseywade/opt/anaconda3/lib/python3.9/site-packages (from
matplotlib!=3.0.0,>=2.0.2->yellowbrick) (21.3)
Requirement already satisfied: fonttools>=4.22.0 in
/Users/dempseywade/opt/anaconda3/lib/python3.9/site-packages (from
matplotlib!=3.0.0,>=2.0.2->yellowbrick) (4.25.0)
Requirement already satisfied: pillow>=6.2.0 in
/Users/dempseywade/opt/anaconda3/lib/python3.9/site-packages (from
matplotlib!=3.0.0,>=2.0.2->yellowbrick) (9.2.0)
Requirement already satisfied: python-dateutil>=2.7 in
/Users/dempseywade/opt/anaconda3/lib/python3.9/site-packages (from
matplotlib!=3.0.0,>=2.0.2->yellowbrick) (2.8.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/Users/dempseywade/opt/anaconda3/lib/python3.9/site-packages (from scikit-
learn>=1.0.0->yellowbrick) (2.2.0)
Requirement already satisfied: joblib>=0.11 in
/Users/dempseywade/opt/anaconda3/lib/python3.9/site-packages (from scikit-
learn>=1.0.0->yellowbrick) (1.1.0)
Requirement already satisfied: six>=1.5 in
/Users/dempseywade/opt/anaconda3/lib/python3.9/site-packages (from python-
dateutil>=2.7->matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.16.0)
```

[2]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

from yellowbrick.datasets import load_concrete
from yellowbrick.regressor import ResidualsPlot
```

[3]:
```
X, y = load_concrete()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

[4]:
```
lr = LinearRegression()
visualizer = ResidualsPlot(lr, hist=False, qqplot=True)
visualizer.fit(X_test, y_test)
visualizer.score(X_test, y_test)
visualizer.show() #Green test, blue train
```



[4]: <AxesSubplot:title={'center':'Residuals for LinearRegression Model'}, xlabel='Predicted Value', ylabel='Residuals'>

1.1 Import the necessary libraries

[5]:
```
## DON'T CHANGE THIS CODE
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Return to top

1.2 Question 1

Read the CSV file named "airplane_crash.csv" in the `data` folder and assign it to a dataframe called `accident`. Next, drop the column `Summary` using the `pandas` command `drop` .

[6]:
```
### GRADED

### YOUR SOLUTION HERE
accident= pd.read_csv('/Users/dempseywade/Desktop/gitRepo/
↪DartmouthCodingAssignments/data/Mod5_airplane_crash.csv')
accident.head(5)
accident = accident.drop('Summary', axis=1)

###
### YOUR CODE HERE
###

### Answer check
accident.columns
```

[6]:
```
Index(['Date', 'Time', 'Location', 'Operator', 'Flight #', 'Route', 'Type',
       'Registration', 'cn/In', 'Aboard', 'Fatalities', 'Ground'],
      dtype='object')
```

[7]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

Now we extract the info and visuaize the first 10 rows of our dataframe

[8]:
```
## DON'T CHANGE THIS CODE
accident.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5268 entries, 0 to 5267
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Date          5268 non-null   object
 1   Time          3049 non-null   object
 2   Location      5248 non-null   object
 3   Operator      5250 non-null   object
 4   Flight #      1069 non-null   object
 5   Route         3562 non-null   object
 6   Type          5241 non-null   object
 7   Registration  4933 non-null   object
 8   cn/In         4040 non-null   object
 9   Aboard        5161 non-null   float64
 10  Fatalities    5256 non-null   float64
 11  Ground        5246 non-null   float64
dtypes: float64(3), object(9)
memory usage: 494.0+ KB
```

[9]:
```
## DON'T CHANGE THIS CODE
accident.head()
```

[9]:
```
         Date   Time                            Location  \
0  09/17/1908  17:18                 Fort Myer, Virginia
1  07/12/1912  06:30             AtlantiCity, New Jersey
2  08/06/1913    NaN  Victoria, British Columbia, Canada
3  09/09/1913  18:30                  Over the North Sea
4  10/17/1913  10:30          Near Johannisthal, Germany

                 Operator Flight #          Route                    Type  \
0    Military - U.S. Army      NaN  Demonstration        Wright Flyer III
1    Military - U.S. Navy      NaN    Test flight               Dirigible
2                 Private        -            NaN        Curtiss seaplane
3  Military - German Navy      NaN            NaN  Zeppelin L-1 (airship)
4  Military - German Navy      NaN            NaN  Zeppelin L-2 (airship)

  Registration cn/In  Aboard  Fatalities  Ground
0          NaN     1     2.0         1.0     0.0
1          NaN   NaN     5.0         5.0     0.0
2          NaN   NaN     1.0         1.0     0.0
3          NaN   NaN    20.0        14.0     0.0
4          NaN   NaN    30.0        30.0     0.0
```

Return to top

1.3 Question 2

This dataset does not have duplicate rows, however it is always good practice to verify that you aren't aggregating duplicate rows.

Double up the `accident` dataframe by appending it to itself. Assign the resulting dataframe to a new dataframe called `temp_accident`. Finally, drop the `last` of the duplicate rows and assign the result to a dataframe called `orig_accident`.

[10]:
```
### GRADED

### YOUR SOLUTION HERE
temp_accident = accident.append(accident)
orig_accident = accident

# keep=last

###
### YOUR CODE HERE
###

### Answer check
```

```
print("Rows in duplicate dataframe: {}".format(temp_accident.shape[0]))
print("Rows in duplicate-free dataframe: {}".format(orig_accident.shape[0]))
```

```
Rows in duplicate dataframe: 10536
Rows in duplicate-free dataframe: 5268
```

[11]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

Return to top

1.4 Question 3

Imputation is a feature engineering technique used to keep valuable data that have null values by replacing the missing values with an estimate.

In our dataframe `orig_accident`, the column `Aboard` has some missing values. Follow these steps to impute the missing values: - Extract this column and as a `pandas.Series` and assign it to a variable called `aboard_missing`. - Compute the mean of `aboard_missing` and store the result in `aboard_average`. - Finally, create a new variable `aboard_people` where the missing values in `aboard_missing` have been imputed with the value in `aboard_average`.

Hint: you can use the methods `.fillna()` or `.isnull()`.

[12]:
```
### GRADED

### YOUR SOLUTION HERE
aboard_missing = accident['Aboard']
print(type(aboard_missing))
aboard_average = aboard_missing.mean()
print(aboard_average)
aboard_people = aboard_missing.fillna(aboard_average)

###
### YOUR CODE HERE
###

### Answer check
print("Average aboard people: {}".format(aboard_average))
```

```
<class 'pandas.core.series.Series'>
27.595427242782407
Average aboard people: 27.595427242782407
```

[13]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

1.5 Regression Evaluation Metrics

In general, we can use three different error evaluation metrics:

Mean Absolute Error (MAE): the mean of the absolute value of the errors

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Mean Squared Error (MSE): the mean of the squared errors

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE): the square root of the mean of the squared errors

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

These evaluation metrics compare to each other in the following way:

- The **MAE** is the easiest to understand because it's just the average error.
- The **MSE** is more popular than MAE because MSE "punishes" larger errors. For this reason MSE tends to be more useful in real world problems.
- The **RMSE** is even more popular than MSE because is interpretable in the "y" units.

Because our goal is to minimize the error, we can also refer to these metrics as **loss functions**.

Return to top

1.6 Question 4

Next, we will find the error between the aboard people and fatalities in our dataset. - Fill the missing values in the column `Fatalities` from `orig_accident` with the average value. Store the result as a `Pandas.Series` to `fatal_count`. - Compute the MAE, MSE and RMSE between `fatal_count` and `aboard_people`. Save the result of each metric comparison into variables called `crash_mae`, `crash_mse`, and `crash_rmse`, respectively.

[14]:
```
from sklearn import metrics
import math

### GRADED

fatalities_missing = orig_accident['Fatalities']
fatalities_average = fatalities_missing.mean()
orig_accident['Fatalities'] = fatalities_missing.fillna(fatalities_average)

### YOUR SOLUTION HERE
fatal_count = orig_accident['Fatalities']
```

```
crash_mae = metrics.mean_absolute_error(fatal_count, aboard_people)
crash_mse = metrics.mean_squared_error(fatal_count, aboard_people)
crash_rmse = math.sqrt(crash_mse)

###
### YOUR CODE HERE
###

### Answer check
print("Missing values in fatal_count: {}".format(fatal_count.isnull().sum()))
print("MAE: {}".format(crash_mae))
print("MSE: {}".format(crash_mse))
print("RMSE: {}".format(crash_rmse))
```

```
Missing values in fatal_count: 0
MAE: 7.852438601750174
MSE: 908.9028198094563
RMSE: 30.14801518855688
```

[15]:

```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

Return to top

1.7 Question 5

Sometimes it is useful to extract rows or columns from a dataframe by setting a condition based on some specific feature we are interested in. For example, we can extract only the entries from the dataframe that have a desired value. Alternatively, we can select entries by applying a boolean condition to the DataFrame.

From the dataframe `orig_accident`, extract only the rows that have the value `"Zeppelin L-1 (airship)"` in the column `Type`. Assign the resulting array to the variable `zeppelin_flights`.

[16]:

```
### GRADED

### YOUR SOLUTION HERE
#zeppelin_flights = orig_accident['Type' == 'Zeppelin L-1 (airship)']
zeppelin_flights = orig_accident[(orig_accident['Type'] == "Zeppelin L-1␣
 ↪(airship)")]
###
### YOUR CODE HERE
###

### Answer check
zeppelin_flights
```

```
[16]:          Date   Time              Location                Operator Flight #  \
      3  09/09/1913  18:30  Over the North Sea  Military - German Navy       NaN

         Route                     Type Registration cn/In  Aboard  Fatalities  Ground
      3   NaN  Zeppelin L-1 (airship)          NaN   NaN    20.0        14.0     0.0
```

```
[17]: ###
      ### AUTOGRADER TEST - DO NOT REMOVE
      ###
```

1.8 Training a model and the Shapiro-Wilk test

In the second part of the assignment, we will work using the linear regression and the *Shapiro-Wilk* test on a dataframe containing information about houses in different regions of the United States.

Imagine your friend is a real estate agent and wants some help predicting housing prices for different regions in the USA. It would be helpful if you could somehow create a model that takes a few features of a house and returns the estimate of what the house would sell for.

He has asked you if you could help him out with your new data science skills. You say yes and decide that Linear Regression might be a good path to solve this problem!

The dataset with historical real state information is stored in the file `USA_Housing.csv` and contains the following columns:

- 'Avg. Area Income': Avg. Income of residents of the city the house is located in.
- 'Avg. Area House Age': Avg Age of Houses in the same city.
- 'Avg. Area Number of Rooms': Avg Number of Rooms for Houses in the same city.
- 'Avg. Area Number of Bedrooms': Avg Number of Bedrooms for Houses in the same city.
- 'Area Population': Population of the city the house is located in.
- 'Price': Final Sale Price for the house.
- 'Address': Address for the house.

1.9 Read and extract information about the data

1.10 DON'T CHANGE THIS CODE

USAhousing = pd.read_csv('data/USA_Housing.csv')

```
[18]: # Import Housing data
      USAhousing = pd.read_csv('/Users/dempseywade/Desktop/gitRepo/
       ↪DartmouthCodingAssignments/data/Mod5_USA_Housing.csv')
```

```
[19]: USAhousing.head()
```

```
[19]:    Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
      0      79545.458574             5.682861                   7.009188
      1      79248.642455             6.002900                   6.730821
      2      61287.067179             5.865890                   8.512727
      3      63345.240046             7.188236                   5.586729
```

```
4      59982.197226            5.040555                  7.839388

   Avg. Area Number of Bedrooms  Area Population         Price  \
0                          4.09     23086.800503  1.059034e+06
1                          3.09     40173.072174  1.505891e+06
2                          5.13     36882.159400  1.058988e+06
3                          3.26     34310.242831  1.260617e+06
4                          4.23     26354.109472  6.309435e+05

                                             Address
0  208 Michael Ferry Apt. 674\nLaurabury, NE 3701…
1  188 Johnson Views Suite 079\nLake Kathleen, CA…
2  9127 Elizabeth Stravenue\nDanieltown, WI 06482…
3                        USS Barnett\nFPO AP 44820
4                       USNS Raymond\nFPO AE 09386
```

[20]:
```
USAhousing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5   Price                         5000 non-null   float64
 6   Address                       5000 non-null   object
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

[21]:
```
USAhousing.describe()
```

[21]:
```
       Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
count       5000.000000          5000.000000                5000.000000
mean       68583.108984             5.977222                   6.987792
std        10657.991214             0.991456                   1.005833
min        17796.631190             2.644304                   3.236194
25%        61480.562388             5.322283                   6.299250
50%        68804.286404             5.970429                   7.002902
75%        75783.338666             6.650808                   7.665871
max       107701.748378             9.519088                  10.759588

       Avg. Area Number of Bedrooms  Area Population         Price
count                   5000.000000      5000.000000  5.000000e+03
```

```
mean                          3.981330     36163.516039  1.232073e+06
std                           1.234137      9925.650114  3.531176e+05
min                           2.000000       172.610686  1.593866e+04
25%                           3.140000     29403.928702  9.975771e+05
50%                           4.050000     36199.406689  1.232669e+06
75%                           4.490000     42861.290769  1.471210e+06
max                           6.500000     69621.713378  2.469066e+06
```

Return to top

1.11 Question 6

Extract the first 10 rows of the the `USAhousing` dataset and store them in a dataframe called `df`:

[22]:
```
### GRADED

### YOUR SOLUTION HERE
df = USAhousing.head(10)

###
### YOUR CODE HERE
###

### Answer check
df
```

[22]:
```
   Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
0      79545.458574             5.682861                   7.009188
1      79248.642455             6.002900                   6.730821
2      61287.067179             5.865890                   8.512727
3      63345.240046             7.188236                   5.586729
4      59982.197226             5.040555                   7.839388
5      80175.754159             4.988408                   6.104512
6      64698.463428             6.025336                   8.147760
7      78394.339278             6.989780                   6.620478
8      59927.660813             5.362126                   6.393121
9      81885.927184             4.423672                   8.167688

   Avg. Area Number of Bedrooms  Area Population         Price  \
0                          4.09     23086.800503  1.059034e+06
1                          3.09     40173.072174  1.505891e+06
2                          5.13     36882.159400  1.058988e+06
3                          3.26     34310.242831  1.260617e+06
4                          4.23     26354.109472  6.309435e+05
5                          4.04     26748.428425  1.068138e+06
6                          3.41     60828.249085  1.502056e+06
7                          2.42     36516.358972  1.573937e+06
8                          2.30     29387.396003  7.988695e+05
```

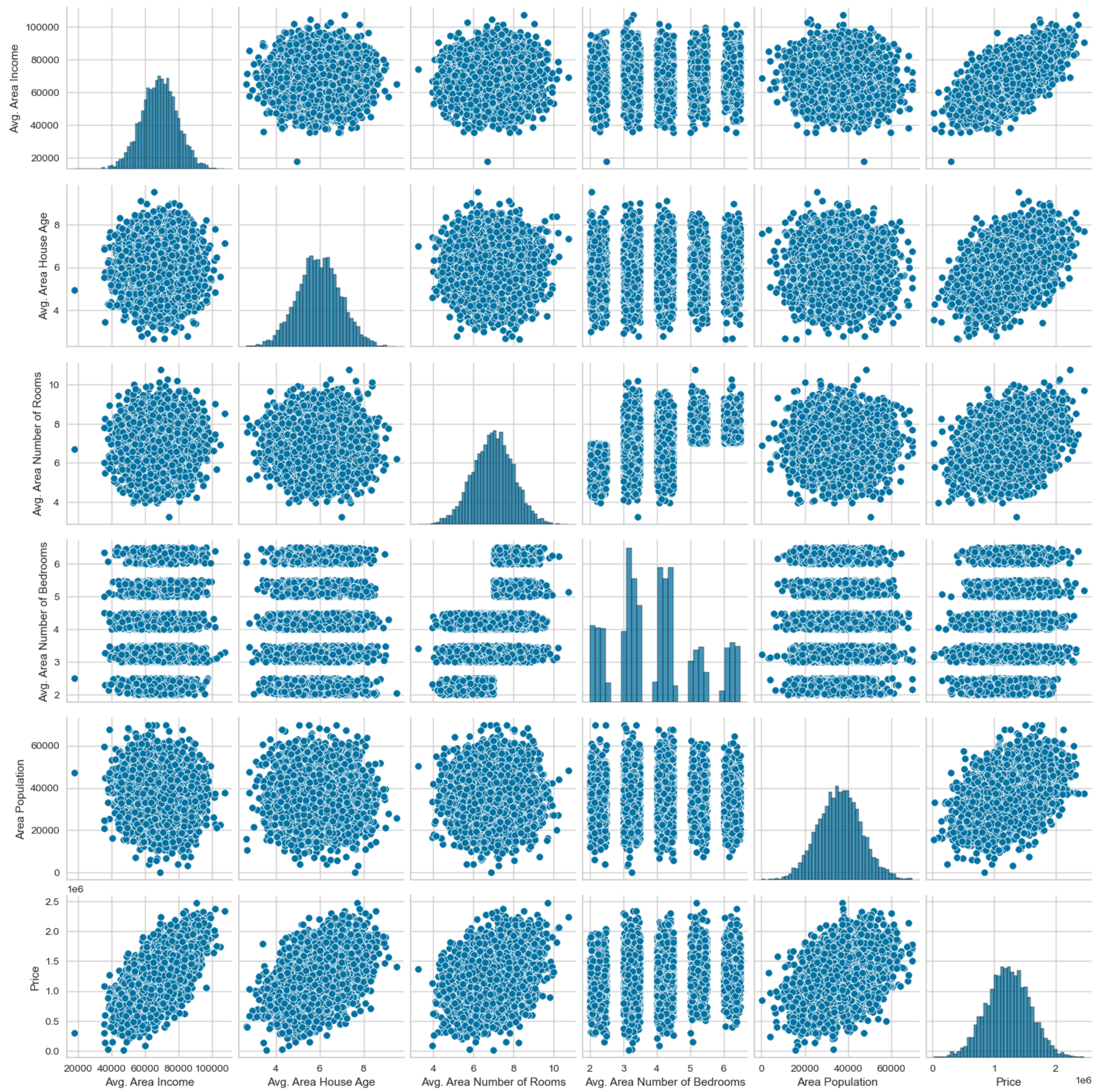
```
9                         6.10     40149.965749  1.545155e+06

                                             Address
0  208 Michael Ferry Apt. 674\nLaurabury, NE 3701…
1  188 Johnson Views Suite 079\nLake Kathleen, CA…
2  9127 Elizabeth Stravenue\nDanieltown, WI 06482…
3                         USS Barnett\nFPO AP 44820
4                        USNS Raymond\nFPO AE 09386
5  06039 Jennifer Islands Apt. 443\nTracyport, KS…
6  4759 Daniel Shoals Suite 442\nNguyenburgh, CO …
7     972 Joyce Viaduct\nLake William, TN 17778-6483
8                         USS Gilbert\nFPO AA 20957
9                  Unit 9446 Box 0958\nDPO AE 97025
```

[23]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

1.12 Exploratory data analysis (EDA)

We will start our EDA by creating simple plots to visualize scatterplots of all numeric features against each other.

[24]:
```
tf=sns.pairplot(USAhousing)
```



Return to top

1.13 Question 7

Check whether the `Price` quantity is following a normal distribution or not by using the Shapiro-Wilk test. To do so, create a function called `normal_test` that takes as first argumant a `pandas.Series` and as second argument a threshold p-value. The function must return `True` if the whole Series follows a normal distribution.

Hints: - Check the documentation of `scipy.stats.shapiro` - A Series is considered to follow a Normal Distribution when the p-value returned by the *Shapiro-Wilk test* is larger than the threshold p-value.

Test the function with the column `Price` and a threshold p-value of `0.05`. Store the result in a (boolean) variable called `ans1`.

[25]:
```python
from scipy.stats import shapiro


### GRADED

### YOUR SOLUTION HERE
def normal_test(column, p_thr):
    start, p = shapiro(column)

    if (p_thr < p):
        return True
    else:
        return False

ans1 = normal_test(USAhousing['Price'], 0.05)

###
### YOUR CODE HERE
###

### Answer check
print("USA Housing price follows a Normal distribution: {}".format(ans1))
```

```
USA Housing price follows a Normal distribution: True
```

[26]:
```python
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

Return to top

1.14 Question 8

Create a dataframe containing the correlation of all the variables against each other. Save the final dataframe in a variable called `corr`.

[27]:
```python
USAhousing.head(5)
```

[27]:
```
   Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
0      79545.458574             5.682861                   7.009188
1      79248.642455             6.002900                   6.730821
2      61287.067179             5.865890                   8.512727
3      63345.240046             7.188236                   5.586729
4      59982.197226             5.040555                   7.839388

   Avg. Area Number of Bedrooms  Area Population         Price  \
0                          4.09     23086.800503  1.059034e+06
1                          3.09     40173.072174  1.505891e+06
```

```
2                           5.13     36882.159400  1.058988e+06
3                           3.26     34310.242831  1.260617e+06
4                           4.23     26354.109472  6.309435e+05

                                             Address
0  208 Michael Ferry Apt. 674\nLaurabury, NE 3701…
1  188 Johnson Views Suite 079\nLake Kathleen, CA…
2  9127 Elizabeth Stravenue\nDanieltown, WI 06482…
3                         USS Barnett\nFPO AP 44820
4                        USNS Raymond\nFPO AE 09386
```

[28]:

```
### GRADED

### YOUR SOLUTION HERE
corr = USAhousing.corr()

###
### YOUR CODE HERE
###

### Answer check
corr
```

[28]:

```
                              Avg. Area Income  Avg. Area House Age  \
Avg. Area Income                      1.000000            -0.002007
Avg. Area House Age                  -0.002007             1.000000
Avg. Area Number of Rooms            -0.011032            -0.009428
Avg. Area Number of Bedrooms          0.019788             0.006149
Area Population                      -0.016234            -0.018743
Price                                 0.639734             0.452543

                              Avg. Area Number of Rooms  \
Avg. Area Income                              -0.011032
Avg. Area House Age                           -0.009428
Avg. Area Number of Rooms                      1.000000
Avg. Area Number of Bedrooms                   0.462695
Area Population                                0.002040
Price                                          0.335664

                              Avg. Area Number of Bedrooms  Area Population  \
Avg. Area Income                                  0.019788        -0.016234
Avg. Area House Age                               0.006149        -0.018743
Avg. Area Number of Rooms                         0.462695         0.002040
Avg. Area Number of Bedrooms                      1.000000        -0.022168
Area Population                                  -0.022168         1.000000
Price                                             0.171071         0.408556
```

```
                                   Price
Avg. Area Income                0.639734
Avg. Area House Age             0.452543
Avg. Area Number of Rooms       0.335664
Avg. Area Number of Bedrooms    0.171071
Area Population                 0.408556
Price                           1.000000
```

[29]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

1.15 Training a Linear Regression Model

Let's now begin to train our regression model! We will first need to split up our data into an `X` dataframe that contains the features we want to train on, and a `y` a series with the desired target variable (in this case `Price`).

For this part, we will ignore the `Address` column because it contains strings which the linear regression model can't use.

1.15.1 Defining the X and y arrays

[30]:
```
X = USAhousing[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of␣
↪Rooms',
               'Avg. Area Number of Bedrooms', 'Area Population']]
y = USAhousing['Price']
```

Return to top

1.16 Question 9

Split both the features `X` and the target `y` into **training** sets `X_train`, and `y_train`, respectively, and and into **testing** sets, `X_test` and `y_test`. This is done so that we can train our model on the training sets `X_train`, and `y_train`, and then test it on the testing sets `X_test` and `y_test`. Split both `X` and `y` so that 30% of the data is used for testing, and the remaining 70% for training.

Important: Use the function `train_test_split` and set the `random_state` equal to 101.

[31]:
```
from sklearn.model_selection import train_test_split

### GRADED

### YOUR SOLUTION HERE
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=.3,␣
↪random_state=101)

###
### YOUR CODE HERE
```

```
###

### Answer check
print("Train features shape: {}".format(X_train.shape))
print("Train labels shape: {}".format(y_train.shape))
print("Test features shape: {}".format(X_test.shape))
print("Test features shape: {}".format(y_test.shape))
```

```
Train features shape: (3500, 5)
Train labels shape: (3500,)
Test features shape: (1500, 5)
Test features shape: (1500,)
```

[32]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

1.17 Creating and Training the Model

Return to top

1.18 Question 10

Create and fit a Linear Regression model using `X_train` and `y_train`. Save the fitted model in a variable called `model`.

Make sure to import the module `LinearRegression` from `sklearn`. Use `LinearRegression` using the default parameters.

[33]:
```
### GRADED
from sklearn.linear_model import LinearRegression

### YOUR SOLUTION HERE

model = LinearRegression().fit(X_train, y_train)

#model = Instance()
#then fit !

###
### YOUR CODE HERE
###

### Answer check
print("Model intercept: {}".format(model.intercept_))
print("Model coefficients: {}".format(model.coef_))
```

```
Model intercept: -2641372.6673013847
Model coefficients: [2.16176350e+01 1.65221120e+05 1.21405377e+05 1.31871878e+03
```

```
1.52251955e+01]
```

[34]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

1.19 Model Evaluation

Let's check out the coefficients and the intercept of our model so we can interpret them.

[35]:
```
# print the intercept
print(model.intercept_)
```

```
-2641372.6673013847
```

[36]:
```
pd.DataFrame(model.coef_,X.columns,columns=['Coefficient'])
```

[36]:

	Coefficient
Avg. Area Income	21.617635
Avg. Area House Age	165221.119872
Avg. Area Number of Rooms	121405.376596
Avg. Area Number of Bedrooms	1318.718783
Area Population	15.225196

Interpreting the coefficients:

- Holding all other features fixed, a 1 unit increase in **Avg. Area Income** is associated with an **increase in price of about $20**.
- Holding all other features fixed, a 1 unit increase in **Avg. Area House Age** is associated with an **increase in price of about $165k**.
- Holding all other features fixed, a 1 unit increase in **Avg. Area Number of Rooms** is associated with an **increase in price of about $120k**.
- Holding all other features fixed, a 1 unit increase in **Avg. Area Number of Bedrooms** is associated with an **increase in price of about $1k**.
- Holding all other features fixed, a 1 unit increase in **Area Population** is associated with an **increase of in price of about $15**.

1.20 Predictions from our Model

Let's now consider the testing sets and see how well it did!

Return to top

1.21 Question 11

Use the model to create predictions for `X_test` and store them in the variable `predictions`.

[37]:
```
### GRADED

### YOUR SOLUTION HERE
```

```
predictions = model.predict(X_test)

###
### YOUR CODE HERE
###

### Answer check
print("First 10 predictions:\n{}\n".format(pd.Series(predictions[:10])))
print("First 10 prices:\n{}".format(y_test[:10].reset_index().
 ↪drop('index',axis=1)))
```

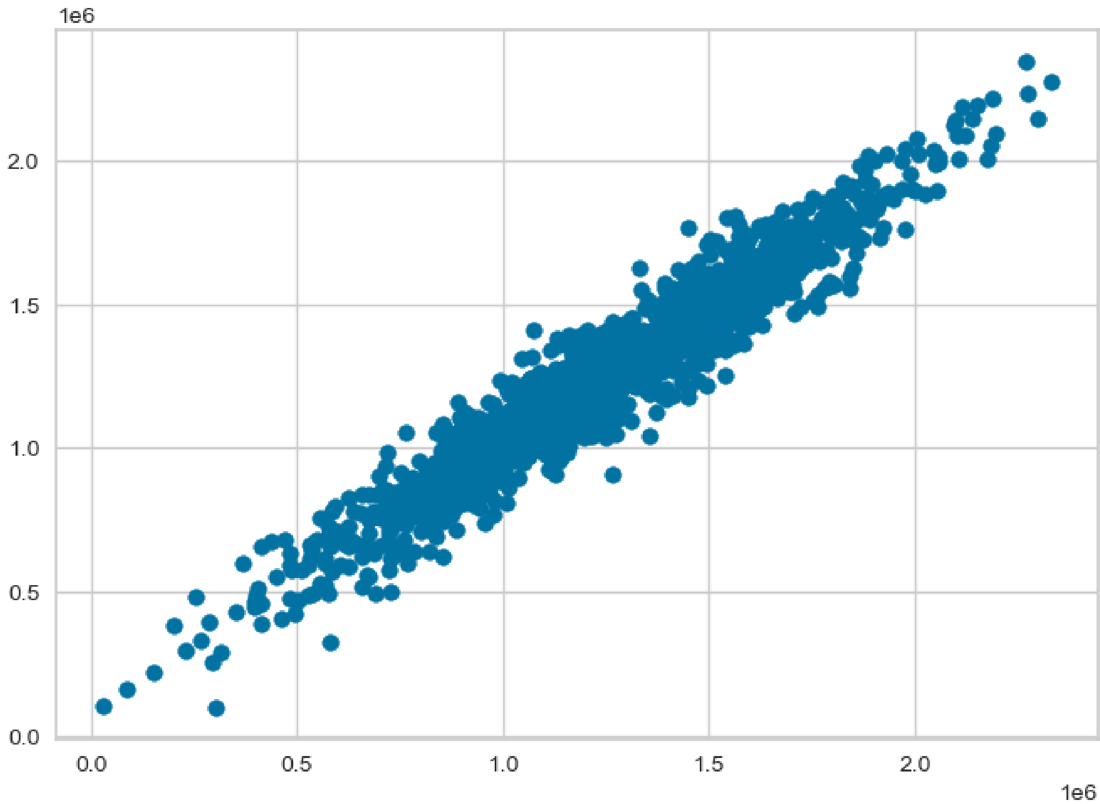
```
First 10 predictions:
0    1.258935e+06
1    8.226946e+05
2    1.742214e+06
3    9.729370e+05
4    9.945460e+05
5    6.444863e+05
6    1.078071e+06
7    8.547560e+05
8    1.445901e+06
9    1.203355e+06
dtype: float64

First 10 prices:
          Price
0  1.251689e+06
1  8.730483e+05
2  1.696978e+06
3  1.063964e+06
4  9.487883e+05
5  7.300436e+05
6  1.166925e+06
7  7.054441e+05
8  1.499989e+06
9  1.288199e+06
```

[38]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

Take a look at your predictions compared to the true values:

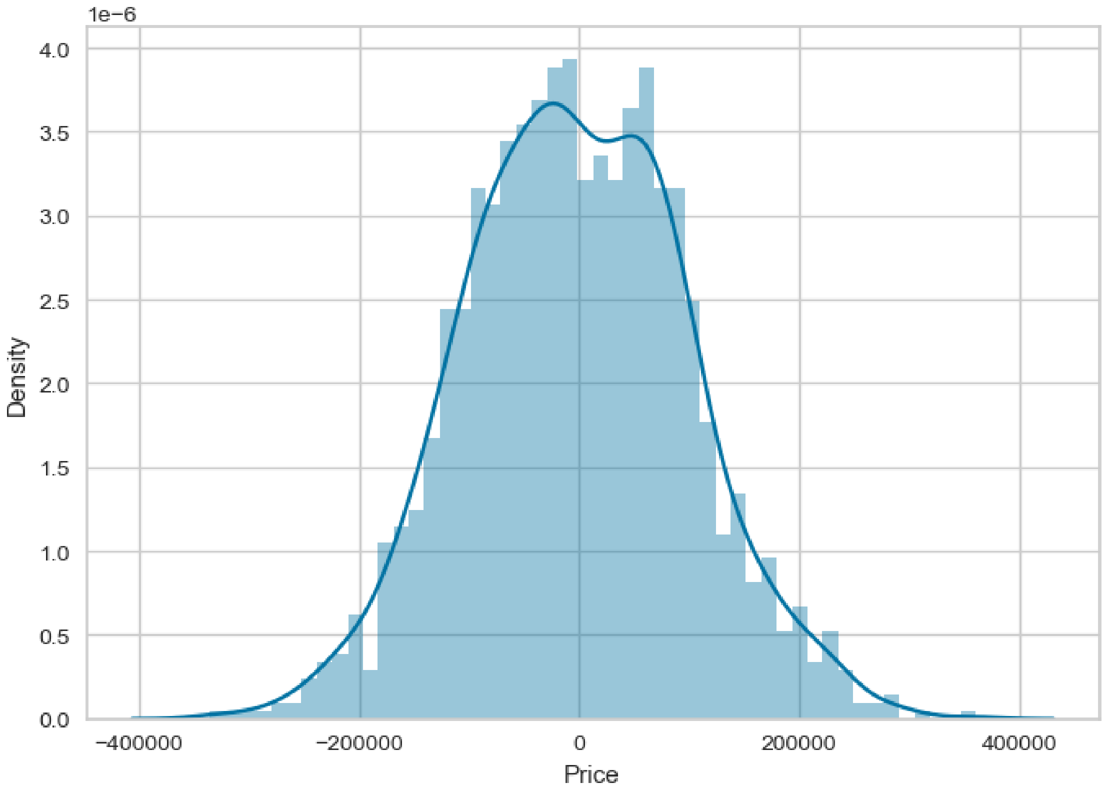
[39]:
```
plt.scatter(y_test,predictions)
```

[39]:
```
<matplotlib.collections.PathCollection at 0x7fda20f1a2b0>
```



Now, take a look at your Residual Histogram. The Residual Histogram shows the distribution of the error:

[40]:
```
sns.distplot((y_test-predictions),bins=50);
```



Return to top

1.22 Question 12

Find the MAE, MSE and RMSE of your model using the true values `y_test` and `predictions`. Save them in variables called `mae`, `mse` and `rmse` respectively.

[41]:
```
from sklearn import metrics
### GRADED

### YOUR SOLUTION HERE
mae = metrics.mean_absolute_error(y_test, predictions)
mse = metrics.mean_squared_error(y_test, predictions)
rmse = math.sqrt(mse)

###
### YOUR CODE HERE
###

### Answer check
print("MAE: {}".format(mae))
print("MSE: {}".format(mse))
```

```
print("RMSE: {}".format(rmse))
```

```
MAE: 81257.5579585593
MSE: 10169125565.89757
RMSE: 100842.0823163503
```

[42]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```