

Model-Development.jupyterlite

June 2, 2022

1 Model Development

Estimated time needed: **30** minutes

1.1 Objectives

After completing this lab you will be able to:

- Develop prediction models

In this section, we will develop several models that will predict the price of the car using the variables or features. This is just an estimate but should give us an objective idea of how much the car should cost.

Some questions we want to ask in this module

Do I know if the dealer is offering fair value for my trade-in?

Do I know if I put a fair value on my car?

In data analytics, we often use Model Development to help us predict future observations from the data we have.

A model will help us understand the exact relationship between different variables and how these variables are used to predict the result.

Setup

Import libraries:

you are running the lab in your browser, so we will install the libraries using **piplite**

```
[1]: #you are running the lab in your browser, so we will install the libraries  
    ↳ using ``piplite``  
import piplite  
await piplite.install(['pandas'])  
await piplite.install(['matplotlib'])  
await piplite.install(['scipy'])  
await piplite.install(['seaborn'])  
await piplite.install(['scikit-learn'])
```

If you run the lab locally using Anaconda, you can load the correct library and versions by uncommenting the following:

```
[ ]: #If you run the lab locally using Anaconda, you can load the correct library
      ↪and versions by uncommenting the following:
      #install specific version of libraries used in lab
      #! mamba install pandas==1.3.3-y
      #! mamba install numpy=1.21.2-y
      #! mamba install sklearn=0.20.1-y
```

```
[2]: import pandas as pd
      import numpy as np
      import matplotlib.pyplot as plt
```

/lib/python3.9/site-packages/pandas/compat/_init_.py:124: UserWarning: Could not import the lzma module. Your installed Python is incomplete. Attempting to use lzma compression will result in a RuntimeError.

warnings.warn(msg)

This function will download the dataset into your browser

```
[3]: #This function will download the dataset into your browser

      from pyodide.http import pyfetch

      async def download(url, filename):
          response = await pyfetch(url)
          if response.status == 200:
              with open(filename, "wb") as f:
                  f.write(await response.bytes())
```

This dataset was hosted on IBM Cloud object. Click [HERE](#) for free storage.

```
[4]: path = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
      ↪IBMDeveloperSkillsNetwork-DA0101EN-SkillsNetwork/labs/Data%20files/
      ↪automobileEDA.csv'
```

you will need to download the dataset; if you are running locally, please comment out the following

```
[5]: #you will need to download the dataset; if you are running locally, please
      ↪comment out the following
      await download(path, "auto.csv")
      path="auto.csv"
```

Load the data and store it in dataframe df:

```
[6]: df = pd.read_csv(path)
      df.head()
```

```
[6]:   symboling  normalized-losses      make aspiration num-of-doors  \
0         3             122  alfa-romero      std           two
1         3             122  alfa-romero      std           two
```

2	1	122	alfa-romero	std	two
3	2	164	audi	std	four
4	2	164	audi	std	four

	body-style	drive-wheels	engine-location	wheel-base	length	...	\
0	convertible	rwd	front	88.6	0.811148	...	
1	convertible	rwd	front	88.6	0.811148	...	
2	hatchback	rwd	front	94.5	0.822681	...	
3	sedan	fwd	front	99.8	0.848630	...	
4	sedan	4wd	front	99.4	0.848630	...	

	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	price	\
0	9.0	111.0	5000.0	21	27	13495.0	
1	9.0	111.0	5000.0	21	27	16500.0	
2	9.0	154.0	5000.0	19	26	16500.0	
3	10.0	102.0	5500.0	24	30	13950.0	
4	8.0	115.0	5500.0	18	22	17450.0	

	city-L/100km	horsepower-binned	diesel	gas
0	11.190476	Medium	0	1
1	11.190476	Medium	0	1
2	12.368421	Medium	0	1
3	9.791667	Medium	0	1
4	13.055556	Medium	0	1

[5 rows x 29 columns]

1. Linear Regression and Multiple Linear Regression

Linear Regression

One example of a Data Model that we will be using is:

Simple Linear Regression

Simple Linear Regression is a method to help us understand the relationship between two variables:

The predictor/independent variable (X)

The response/dependent variable (that we want to predict)(Y)

The result of Linear Regression is a linear function that predicts the response (dependent) variable as a function of the predictor (independent) variable.

$$Y : \text{Response Variable} \quad X : \text{Predictor Variables}$$

Linear Function

$$\hat{Y} = a + bX$$

a refers to the intercept of the regression line, in other words: the value of Y when X is 0

b refers to the slope of the regression line, in other words: the value with which Y changes when X increases by 1 unit

Let's load the modules for linear regression:

```
[7]: from sklearn.linear_model import LinearRegression
```

Create the linear regression object:

```
[8]: lm = LinearRegression()  
lm
```

```
[8]: LinearRegression()
```

How could “highway-mpg” help us predict car price?

For this example, we want to look at how highway-mpg can help us predict car price. Using simple linear regression, we will create a linear function with “highway-mpg” as the predictor variable and the “price” as the response variable.

```
[9]: X = df[['highway-mpg']]  
Y = df['price']
```

Fit the linear model using highway-mpg:

```
[10]: lm.fit(X,Y)
```

```
[10]: LinearRegression()
```

We can output a prediction:

```
[11]: Yhat=lm.predict(X)  
Yhat[0:5]
```

```
[11]: array([16236.50464347, 16236.50464347, 17058.23802179, 13771.3045085 ,  
        20345.17153508])
```

What is the value of the intercept (a)?

```
[12]: lm.intercept_
```

```
[12]: 38423.30585815743
```

What is the value of the slope (b)?

```
[13]: lm.coef_
```

```
[13]: array([-821.73337832])
```

What is the final estimated linear model we get?

As we saw above, we should get a final linear model with the structure:

$$\hat{Y} = a + bX$$

Plugging in the actual values we get:

Price = 38423.31 - 821.73 x highway-mpg

Question #1 a):

Create a linear regression object called “lm1”.

```
[14]: # Write your code below and press Shift+Enter to execute
      lm1 = LinearRegression()
```

[Click here for the solution](#)

```
lm1 = LinearRegression()
lm1
```

Question #1 b):

Train the model using “engine-size” as the independent variable and “price” as the dependent variable?

```
[17]: # Write your code below and press Shift+Enter to execute
      # Need to use double brackets on engine-size portion. It doesn't work with only
      ↪ single brackets
      lm1.fit(df[['engine-size']], df['price'])
      lm1
```

```
[17]: LinearRegression()
```

[Click here for the solution](#)

```
lm1.fit(df[['engine-size']], df[['price']])
lm1
```

Question #1 c):

Find the slope and intercept of the model.

Slope

```
[18]: # Write your code below and press Shift+Enter to execute
      lm1.coef_
```

```
[18]: array([166.86001569])
```

Intercept

```
[20]: # Write your code below and press Shift+Enter to execute
      lm1.intercept_
```

[20]: -7963.338906281049

2 Slope

lm1.coef_

[Click here for the solution](#)

Slope

lm1.coef_

Intercept

lm1.intercept_

Question #1 d):

What is the equation of the predicted line? You can use x and yhat or “engine-size” or “price”.

```
[ ]: # Write your code below and press Shift+Enter to execute
      # using X and Y
      #Yhat=-7963.34 + 166.86*X

      #Price=-7963.34 + 166.86*engine-size
```

[Click here for the solution](#)

using X and Y

Yhat=-7963.34 + 166.86*X

Price=-7963.34 + 166.86*engine-size

Multiple Linear Regression

What if we want to predict car price using more than one variable?

If we want to use more variables in our model to predict car price, we can use Multiple Linear Regression. Multiple Linear Regression is very similar to Simple Linear Regression, but this method is used to explain the relationship between one continuous response (dependent) variable and two or more predictor (independent) variables. Most of the real-world regression models involve multiple predictors. We will illustrate the structure by using four predictor variables, but these results can generalize to any integer:

Y : Response Variable X_1 : Predictor Variable 1 X_2 : Predictor Variable 2 X_3 : Predictor Variable 3 X_4 :

a : intercept b_1 : coefficients of Variable 1 b_2 : coefficients of Variable 2 b_3 : coefficients of Variable 3 b_4 :

The equation is given by:

$$\hat{Y} = a + b_1X_1 + b_2X_2 + b_3X_3 + b_4X_4$$

From the previous section we know that other good predictors of price could be:

Horsepower

Curb-weight

Engine-size

Highway-mpg

Let's develop a model using these variables as the predictor variables.

```
[21]: Z = df[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']]
```

Fit the linear model using the four above-mentioned variables.

```
[22]: lm.fit(Z, df['price'])
```

```
[22]: LinearRegression()
```

What is the value of the intercept(a)?

```
[23]: lm.intercept_
```

```
[23]: -15806.62462632922
```

What are the values of the coefficients (b1, b2, b3, b4)?

```
[24]: lm.coef_
```

```
[24]: array([53.49574423,  4.70770099, 81.53026382, 36.05748882])
```

What is the final estimated linear model that we get?

As we saw above, we should get a final linear function with the structure:

$$\hat{Y} = a + b_1X_1 + b_2X_2 + b_3X_3 + b_4X_4$$

What is the linear function we get in this example?

Price = -15678.742628061467 + 52.65851272 x horsepower + 4.69878948 x curb-weight + 81.95906216 x engine-size + 33.58258185 x highway-mpg

Question #2 a):

Create and train a Multiple Linear Regression model “lm2” where the response variable is “price”, and the predictor variable is “normalized-losses” and “highway-mpg”.

```
[25]: # Write your code below and press Shift+Enter to execute
lm2 = LinearRegression()
```

```
Z1 = df[['normalized-losses', 'highway-mpg']]
y = df['price']
lm2.fit(Z1, y)
```

[25]: LinearRegression()

[Click here for the solution](#)

```
lm2 = LinearRegression()
lm2.fit(df[['normalized-losses', 'highway-mpg']], df['price'])
```

Question #2 b):

Find the coefficient of the model.

```
[26]: # Write your code below and press Shift+Enter to execute
lm2.coef_
```

[26]: array([1.49789586, -820.45434016])

[Click here for the solution](#)

```
lm2.coef_
```

2. Model Evaluation Using Visualization

Now that we've developed some models, how do we evaluate our models and choose the best one? One way to do this is by using a visualization.

Import the visualization package, seaborn:

```
[27]: # import the visualization package: seaborn
import seaborn as sns
%matplotlib inline
```

Regression Plot

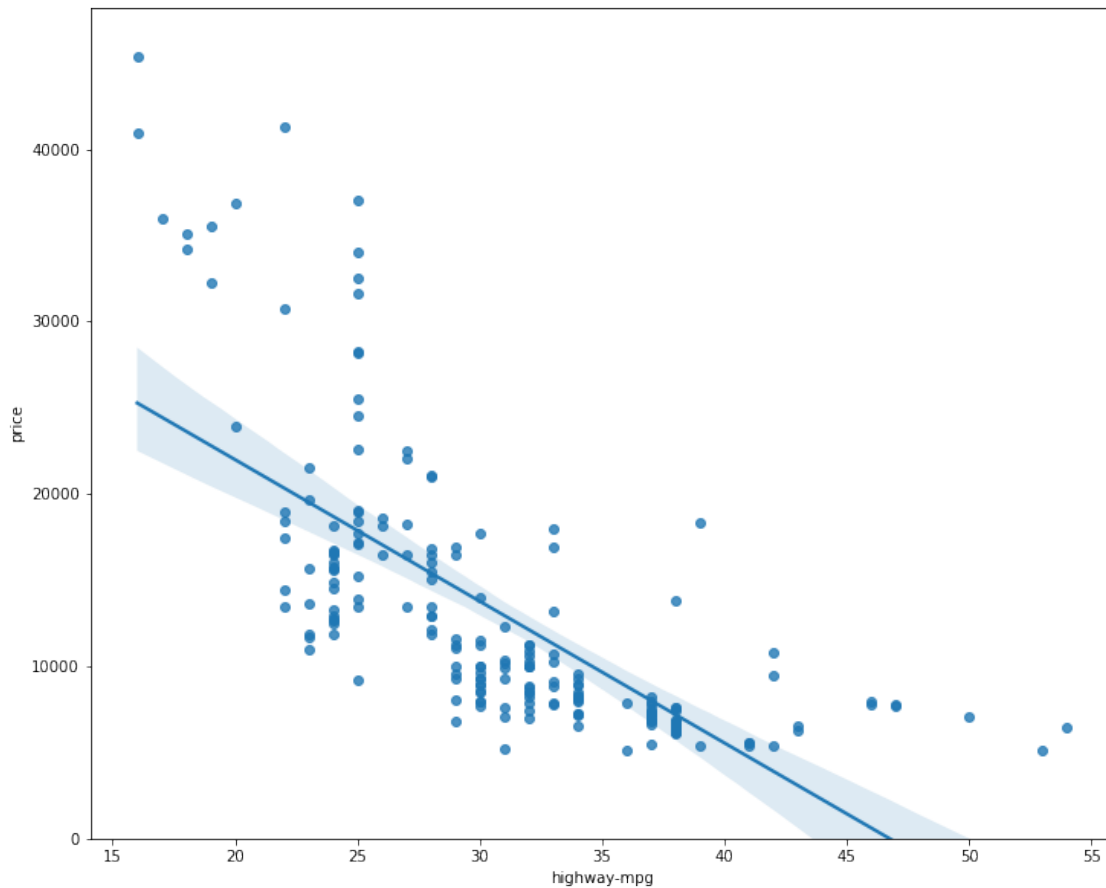
When it comes to simple linear regression, an excellent way to visualize the fit of our model is by using regression plots.

This plot will show a combination of a scattered data points (a scatterplot), as well as the fitted linear regression line going through the data. This will give us a reasonable estimate of the relationship between the two variables, the strength of the correlation, as well as the direction (positive or negative correlation).

Let's visualize **highway-mpg** as potential predictor variable of price:

```
[28]: width = 12
height = 10
plt.figure(figsize=(width, height))
sns.regplot(x="highway-mpg", y="price", data=df)
plt.ylim(0,)
```


[28]: (0.0, 48191.48713200318)



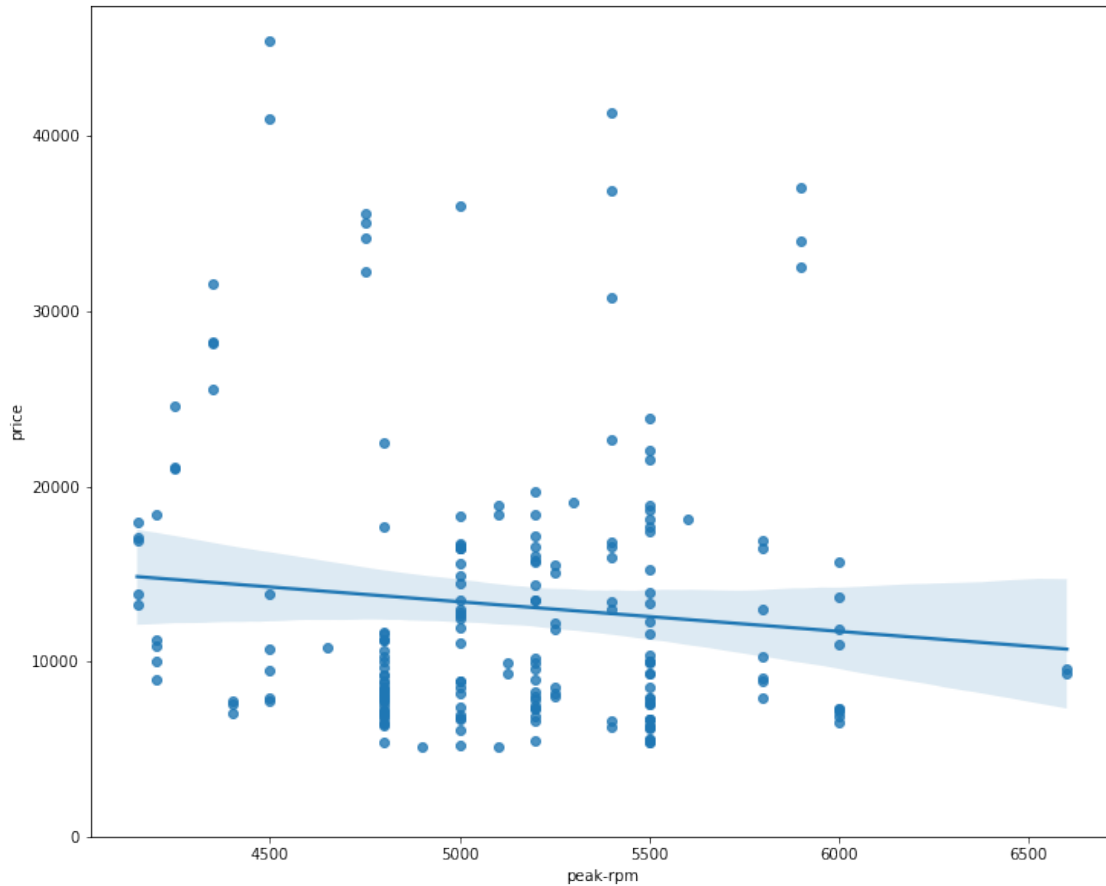
We can see from this plot that price is negatively correlated to highway-mpg since the regression slope is negative.

One thing to keep in mind when looking at a regression plot is to pay attention to how scattered the data points are around the regression line. This will give you a good indication of the variance of the data and whether a linear model would be the best fit or not. If the data is too far off from the line, this linear model might not be the best model for this data.

Let's compare this plot to the regression plot of "peak-rpm".

```
[29]: plt.figure(figsize=(width, height))
      sns.regplot(x="peak-rpm", y="price", data=df)
      plt.ylim(0,)
```

[29]: (0.0, 47414.1)



Comparing the regression plot of “peak-rpm” and “highway-mpg”, we see that the points for “highway-mpg” are much closer to the generated line and, on average, decrease. The points for “peak-rpm” have more spread around the predicted line and it is much harder to determine if the points are decreasing or increasing as the “peak-rpm” increases.

Question #3:

Given the regression plots above, is “peak-rpm” or “highway-mpg” more strongly correlated with “price”? Use the method “corr()” to verify your answer.

```
[30]: # Write your code below and press Shift+Enter to execute
df[["peak-rpm", "highway-mpg", "price"]].corr()
```

```
[30]:
```

	peak-rpm	highway-mpg	price
peak-rpm	1.000000	-0.058598	-0.101616
highway-mpg	-0.058598	1.000000	-0.704692
price	-0.101616	-0.704692	1.000000

[Click here for the solution](#)

The variable "highway-mpg" has a stronger correlation with "price", it is approximate -0.704

```
df[["peak-rpm", "highway-mpg", "price"]].corr()
```

Residual Plot

A good way to visualize the variance of the data is to use a residual plot.

What is a residual?

The difference between the observed value (y) and the predicted value (\hat{y}) is called the residual (e). When we look at a regression plot, the residual is the distance from the data point to the fitted regression line.

So what is a residual plot?

A residual plot is a graph that shows the residuals on the vertical y-axis and the independent variable on the horizontal x-axis.

What do we pay attention to when looking at a residual plot?

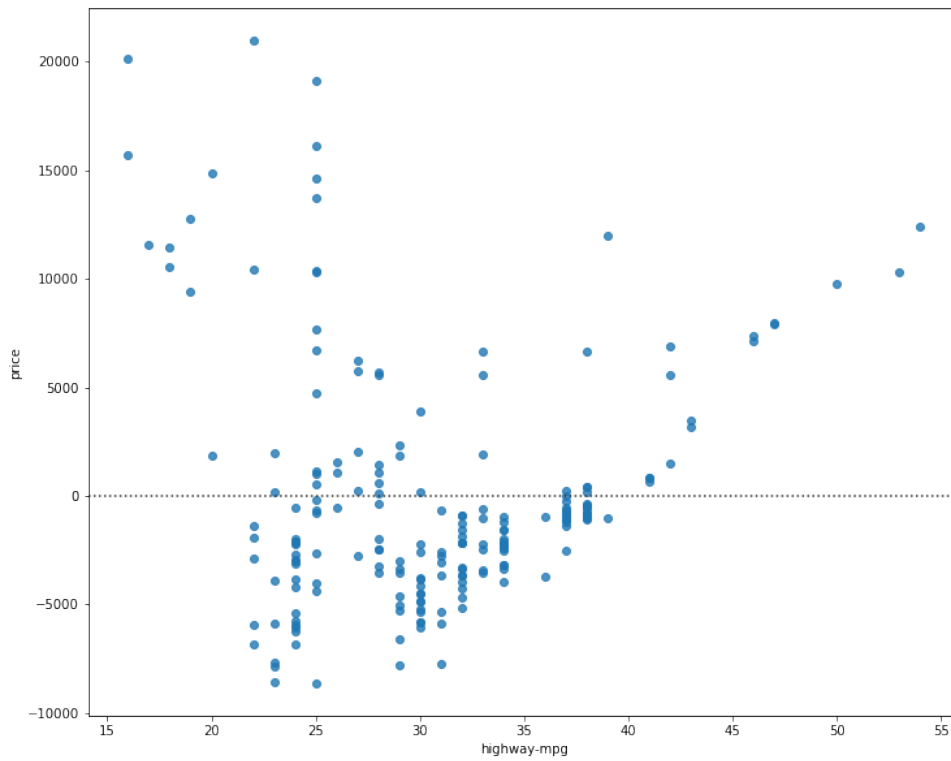
We look at the spread of the residuals:

- If the points in a residual plot are randomly spread out around the x-axis, then a linear model is appropriate for the data.

Why is that? Randomly spread out residuals means that the variance is constant, and thus the linear model is a good fit for this data.

```
[31]: width = 12
      height = 10
      plt.figure(figsize=(width, height))
      sns.residplot(df['highway-mpg'], df['price'])
      plt.show()
```

```
/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the
following variables as keyword args: x, y. From version 0.12, the only valid
positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
warnings.warn(
```



<Figure size 864x720 with 0 Axes>

What is this plot telling us?

We can see from this residual plot that the residuals are not randomly spread around the x-axis, leading us to believe that maybe a non-linear model is more appropriate for this data.

Multiple Linear Regression

How do we visualize a model for Multiple Linear Regression? This gets a bit more complicated because you can't visualize it with regression or residual plot.

One way to look at the fit of the model is by looking at the distribution plot. We can look at the distribution of the fitted values that result from the model and compare it to the distribution of the actual values.

First, let's make a prediction:

```
[32]: Y_hat = lm.predict(Z)
```

```
[33]: plt.figure(figsize=(width, height))

ax1 = sns.distplot(df['price'], hist=False, color="r", label="Actual Value")
sns.distplot(Y_hat, hist=False, color="b", label="Fitted Values" , ax=ax1)

plt.title('Actual vs Fitted Values for Price')
plt.xlabel('Price (in dollars)')
plt.ylabel('Proportion of Cars')

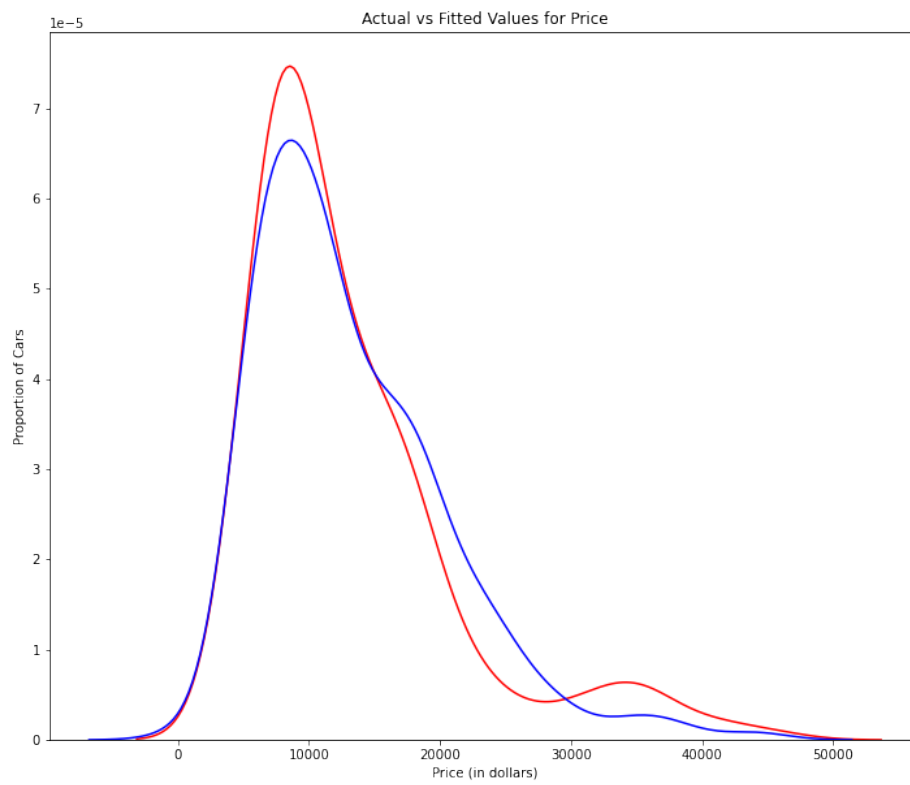
plt.show()
plt.close()
```

```
/lib/python3.9/site-packages/seaborn/distributions.py:2619: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density
plots).
```

```
warnings.warn(msg, FutureWarning)
```

```
/lib/python3.9/site-packages/seaborn/distributions.py:2619: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density
plots).
```

```
warnings.warn(msg, FutureWarning)
```



We can see that the fitted values are reasonably close to the actual values since the two distributions overlap a bit. However, there is definitely some room for improvement.

3. Polynomial Regression and Pipelines

Polynomial regression is a particular case of the general linear regression model or multiple linear regression models.

We get non-linear relationships by squaring or setting higher-order terms of the predictor variables.

There are different orders of polynomial regression:

Quadratic - 2nd Order

$$\hat{Y} = a + b_1X + b_2X^2$$

Cubic - 3rd Order

$$\hat{Y} = a + b_1X + b_2X^2 + b_3X^3$$

Higher-Order:

$$Y = a + b_1X + b_2X^2 + b_3X^3 \dots$$

We saw earlier that a linear model did not provide the best fit while using “highway-mpg” as the predictor variable. Let’s see if we can try fitting a polynomial model to the data instead.

We will use the following function to plot the data:

```
[34]: def PlotPolly(model, independent_variable, dependent_variabble, Name):
      x_new = np.linspace(15, 55, 100)
      y_new = model(x_new)

      plt.plot(independent_variable, dependent_variabble, '.', x_new, y_new, '-')
      plt.title('Polynomial Fit with Matplotlib for Price ~ Length')
      ax = plt.gca()
      ax.set_facecolor((0.898, 0.898, 0.898))
      fig = plt.gcf()
      plt.xlabel(Name)
      plt.ylabel('Price of Cars')

      plt.show()
      plt.close()
```

Let’s get the variables:

```
[35]: x = df['highway-mpg']
      y = df['price']
```

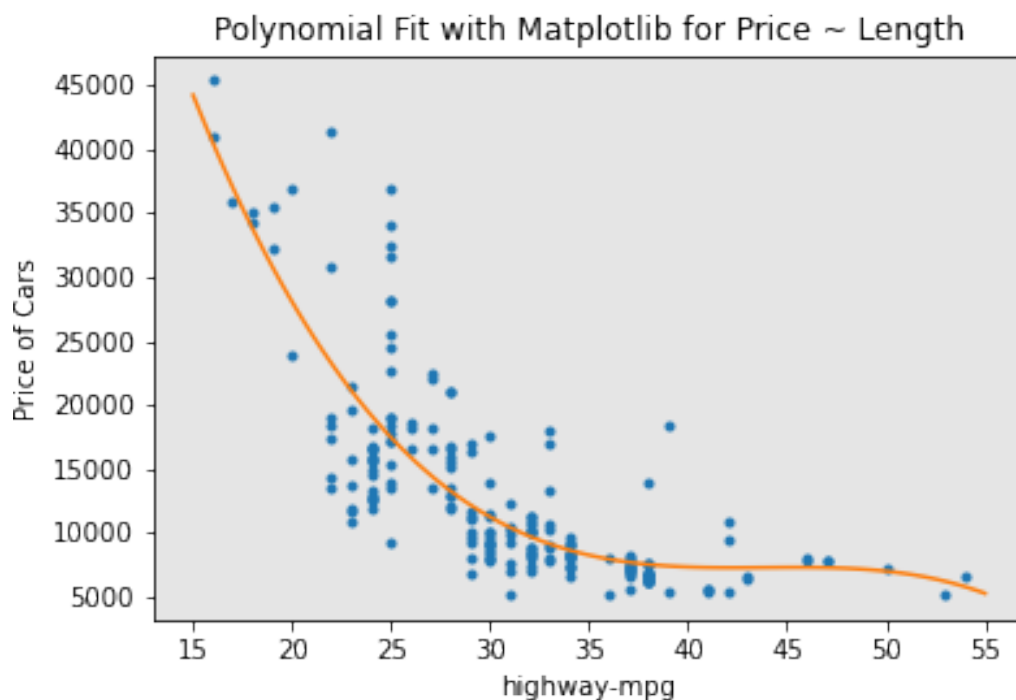
Let’s fit the polynomial using the function polyfit, then use the function poly1d to display the polynomial function.

```
[36]: # Here we use a polynomial of the 3rd order (cubic)
      f = np.polyfit(x, y, 3)
      p = np.poly1d(f)
      print(p)
```

```
          3          2
-1.557 x + 204.8 x - 8965 x + 1.379e+05
```

Let’s plot the function:

```
[37]: PlotPolly(p, x, y, 'highway-mpg')
```



```
[38]: np.polyfit(x, y, 3)
```

```
[38]: array([-1.55663829e+00,  2.04754306e+02, -8.96543312e+03,  1.37923594e+05])
```

We can already see from plotting that this polynomial model performs better than the linear model. This is because the generated polynomial function “hits” more of the data points.

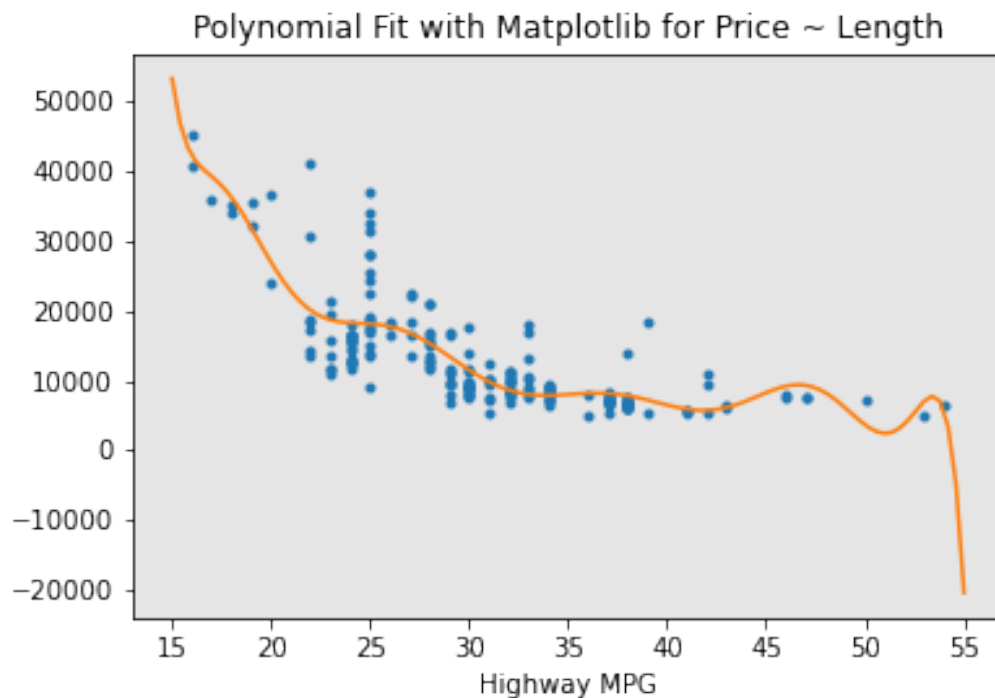
Question #4:

Create 11 order polynomial model with the variables x and y from above.

```
[39]: # Write your code below and press Shift+Enter to execute
# Here we use a polynomial of the 3rd order (cubic)
f1 = np.polyfit(x, y, 11)
p1 = np.poly1d(f1)
print(p1)
PlotPolly(p1,x,y, 'Highway MPG')
```

```

      11      10      9      8      7
-1.243e-08 x  + 4.722e-06 x  - 0.0008028 x + 0.08056 x - 5.297 x
      6      5      4      3      2
+ 239.5 x - 7588 x + 1.684e+05 x - 2.565e+06 x + 2.551e+07 x - 1.491e+08 x +
3.879e+08
```

[Click here for the solution](#)

```
# Here we use a polynomial of the 11rd order (cubic)
f1 = np.polyfit(x, y, 11)
p1 = np.poly1d(f1)
print(p1)
PlotPolly(p1,x,y, 'Highway MPG')
```

The analytical expression for Multivariate Polynomial function gets complicated. For example, the expression for a second-order (degree=2) polynomial with two variables is given by:

$$\hat{Y} = a + b_1X_1 + b_2X_2 + b_3X_1X_2 + b_4X_1^2 + b_5X_2^2$$

We can perform a polynomial transform on multiple features. First, we import the module:

```
[40]: from sklearn.preprocessing import PolynomialFeatures
```

We create a PolynomialFeatures object of degree 2:

```
[41]: pr=PolynomialFeatures(degree=2)
pr
```

```
[41]: PolynomialFeatures()
```

```
[42]: Z_pr=pr.fit_transform(Z)
```

In the original data, there are 201 samples and 4 features.

```
[43]: Z.shape
```

```
[43]: (201, 4)
```

After the transformation, there are 201 samples and 15 features.

```
[44]: Z_pr.shape
```

```
[44]: (201, 15)
```

Pipeline

Data Pipelines simplify the steps of processing the data. We use the module Pipeline to create a pipeline. We also use StandardScaler as a step in our pipeline.

```
[45]: from sklearn.pipeline import Pipeline
      from sklearn.preprocessing import StandardScaler
```

We create the pipeline by creating a list of tuples including the name of the model or estimator and its corresponding constructor.

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

We input the list as an argument to the pipeline constructor:

```
[47]: pipe=Pipeline(Input)
      pipe
```

```
[47]: Pipeline(steps=[('scale', StandardScaler()),
                      ('polynomial', PolynomialFeatures(include_bias=False)),
                      ('model', LinearRegression())])
```

First, we convert the data type Z to type float to avoid conversion warnings that may appear as a result of StandardScaler taking float inputs.

Then, we can normalize the data, perform a transform and fit the model simultaneously.

```
[48]: Z = Z.astype(float)
      pipe.fit(Z,y)
```

```
[48]: Pipeline(steps=[('scale', StandardScaler()),
                      ('polynomial', PolynomialFeatures(include_bias=False)),
                      ('model', LinearRegression())])
```

Similarly, we can normalize the data, perform a transform and produce a prediction simultaneously.

```
[49]: ypipe=pipe.predict(Z)
ypipe[0:4]
```

```
[49]: array([13102.74784201, 13102.74784201, 18225.54572197, 10390.29636555])
```

Question #5:

Create a pipeline that standardizes the data, then produce a prediction using a linear regression model using the features Z and target y.

```
[ ]: # Write your code below and press Shift+Enter to execute
```

[Click here for the solution](#)

```
Input=[('scale',StandardScaler()),('model',LinearRegression())]
```

```
pipe=Pipeline(Input)
```

```
pipe.fit(Z,y)
```

```
ypipe=pipe.predict(Z)
ypipe[0:10]
```

4. Measures for In-Sample Evaluation

When evaluating our models, not only do we want to visualize the results, but we also want a quantitative measure to determine how accurate the model is.

Two very important measures that are often used in Statistics to determine the accuracy of a model are:

R^2 / R-squared

Mean Squared Error (MSE)

R-squared

R squared, also known as the coefficient of determination, is a measure to indicate how close the data is to the fitted regression line.

The value of the R-squared is the percentage of variation of the response variable (y) that is explained by a linear model.

Mean Squared Error (MSE)

The Mean Squared Error measures the average of the squares of errors. That is, the difference between actual value (y) and the estimated value (\hat{y}).

Model 1: Simple Linear Regression

Let's calculate the R^2 :

```
[ ]: #highway_mpg_fit
lm.fit(X, Y)
# Find the R^2
```

```
print('The R-square is: ', lm.score(X, Y))
```

We can say that ~49.659% of the variation of the price is explained by this simple linear model “horsepower_fit”.

Let’s calculate the MSE:

We can predict the output i.e., “yhat” using the predict method, where X is the input variable:

```
[ ]: Yhat=lm.predict(X)
print('The output of the first four predicted value is: ', Yhat[0:4])
```

Let’s import the function mean_squared_error from the module metrics:

```
[ ]: from sklearn.metrics import mean_squared_error
```

We can compare the predicted results with the actual results:

```
[ ]: mse = mean_squared_error(df['price'], Yhat)
print('The mean square error of price and predicted value is: ', mse)
```

Model 2: Multiple Linear Regression

Let’s calculate the R^2 :

```
[ ]: # fit the model
lm.fit(Z, df['price'])
# Find the  $R^2$ 
print('The R-square is: ', lm.score(Z, df['price']))
```

We can say that ~80.896 % of the variation of price is explained by this multiple linear regression “multi_fit”.

Let’s calculate the MSE.

We produce a prediction:

```
[ ]: Y_predict_multifit = lm.predict(Z)
```

We compare the predicted results with the actual results:

```
[ ]: print('The mean square error of price and predicted value using multifit is: ',
↪ \
mean_squared_error(df['price'], Y_predict_multifit))
```

Model 3: Polynomial Fit

Let’s calculate the R^2 .

Let’s import the function r2_score from the module metrics as we are using a different function.

```
[ ]: from sklearn.metrics import r2_score
```

We apply the function to get the value of R^2 :

```
[ ]: r_squared = r2_score(y, p(x))
      print('The R-square value is: ', r_squared)
```

We can say that ~67.419 % of the variation of price is explained by this polynomial fit.

MSE

We can also calculate the MSE:

```
[ ]: mean_squared_error(df['price'], p(x))
```

5. Prediction and Decision Making

Prediction

In the previous section, we trained the model using the method fit. Now we will use the method predict to produce a prediction. Lets import pyplot for plotting; we will also be using some functions from numpy.

```
[ ]: import matplotlib.pyplot as plt
      import numpy as np

      %matplotlib inline
```

Create a new input:

```
[ ]: new_input=np.arange(1, 100, 1).reshape(-1, 1)
```

Fit the model:

```
[ ]: lm.fit(X, Y)
      lm
```

Produce a prediction:

```
[ ]: yhat=lm.predict(new_input)
      yhat[0:5]
```

We can plot the data:

```
[ ]: plt.plot(new_input, yhat)
      plt.show()
```

Decision Making: Determining a Good Model Fit

Now that we have visualized the different models, and generated the R-squared and MSE values for the fits, how do we determine a good model fit?

What is a good R-squared value?

When comparing models, the model with the higher R-squared value is a better fit for the data.

What is a good MSE?

When comparing models, the model with the smallest MSE value is a better fit for the data.

Let's take a look at the values for the different models.

Simple Linear Regression: Using Highway-mpg as a Predictor Variable of Price.

R-squared: 0.49659118843391759

MSE: 3.16×10^7

Multiple Linear Regression: Using Horsepower, Curb-weight, Engine-size, and Highway-mpg as Predictor Variables of Price.

R-squared: 0.80896354913783497

MSE: 1.2×10^7

Polynomial Fit: Using Highway-mpg as a Predictor Variable of Price.

R-squared: 0.6741946663906514

MSE: 2.05×10^7

Simple Linear Regression Model (SLR) vs Multiple Linear Regression Model (MLR)

Usually, the more variables you have, the better your model is at predicting, but this is not always true. Sometimes you may not have enough data, you may run into numerical problems, or many of the variables may not be useful and even act as noise. As a result, you should always check the MSE and R^2 .

In order to compare the results of the MLR vs SLR models, we look at a combination of both the R-squared and MSE to make the best conclusion about the fit of the model.

MSE: The MSE of SLR is 3.16×10^7 while MLR has an MSE of 1.2×10^7 . The MSE of MLR is much smaller.

R-squared: In this case, we can also see that there is a big difference between the R-squared of the SLR and the R-squared of the MLR. The R-squared for the SLR (~ 0.497) is very small compared to the R-squared for the MLR (~ 0.809).

This R-squared in combination with the MSE show that MLR seems like the better model fit in this case compared to SLR.

Simple Linear Model (SLR) vs. Polynomial Fit

MSE: We can see that Polynomial Fit brought down the MSE, since this MSE is smaller than the one from the SLR.

R-squared: The R-squared for the Polynomial Fit is larger than the R-squared for the SLR, so the Polynomial Fit also brought up the R-squared quite a bit.

Since the Polynomial Fit resulted in a lower MSE and a higher R-squared, we can conclude that this was a better fit model than the simple linear regression for predicting "price" with "highway-mpg" as a predictor variable.

Multiple Linear Regression (MLR) vs. Polynomial Fit

MSE: The MSE for the MLR is smaller than the MSE for the Polynomial Fit.

R-squared: The R-squared for the MLR is also much larger than for the Polynomial Fit.

Conclusion

Comparing these three models, we conclude that the MLR model is the best model to be able to predict price from our dataset. This result makes sense since we have 27 variables in total and we know that more than one of those variables are potential predictors of the final car price.

2.0.1 Thank you for completing this lab!

2.1 Author

Joseph Santarcangelo

2.1.1 Other Contributors

Mahdi Noorian PhD

Bahare Talayian

Eric Xiao

Steven Dong

Parizad

Hima Vasudevan

Fiorella Wenver

Yi Yao.

2.2 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-10-30	2.2	Lakshmi	Changed url of csv
2020-09-09	2.1	Lakshmi	Fixes made in Polynomial Regression Equations
2020-08-27	2.0	Lavanya	Moved lab to course repo in GitLab

##

© IBM Corporation 2020. All rights reserved.