Model Development

Estimated time needed: **30** minutes

Objectives

After completing this lab you will be able to:

- Develop prediction models

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.

Import libraries:

```
#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

import piplite
await piplite.install('seaborn')

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Load the data and store it in dataframe `df`:

This dataset was hosted on IBM Cloud object. Click HERE for free storage. Download it by running the cell below.

```
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())

file_path= "https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DA0101EN-
SkillsNetwork/labs/Data%20files/automobileEDA.csv"

await download(file_path, "usedcars.csv")
file_name="usedcars.csv"
```

```
df = pd.read_csv(file_name)
df.head()
```

```
   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]
```

Note: This version of the lab is working on JupyterLite, which requires the dataset to be downloaded to the interface.While working on the downloaded version of this notebook on their local machines(Jupyter Anaconda), the learners can simply skip the steps above, and simply use the URL directly in the pandas.read_csv() function. You can uncomment and run the statements in the cell below.

```
#filepath = "https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DA0101EN-
SkillsNetwork/labs/Data%20files/automobileEDA.csv"
#df = pd.read_csv(filepath, header=None)
```

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)

$$ Y: Response \ Variable\\\\\\ X: Predictor \ Variables $$

Linear Function

$$Yhat = a + bX$$

```
from sklearn.linear_model import LinearRegression

lm = LinearRegression()
lm

LinearRegression()
```

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.

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

Fit the linear model using highway-mpg:

```
lm.fit(X,Y)

LinearRegression()
```

We can output a prediction:

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

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

```
lm.intercept_
```

```
38423.30585815743
```

```
lm.coef_
```

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

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

$$Yhat = a + bX$$

Plugging in the actual values we get:

Price = 38423.31 - 821.73 x highway-mpg

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

```
LinearRegression()
```

```
# Write your code below and press Shift+Enter to execute
lm1.fit(df[['engine-size']], df[['price']])
lm1
```

```
LinearRegression()
```

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

```
array([[166.86001569]])
```

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

```
array([-7963.33890628])
```

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

Price=-7963.34 + 166.86*df['engine-size']
```

$$ Y: Response \ Variable\\\\\\ X_1 :Predictor\ Variable \ 1\\\\ X_2: Predictor\ Variable \ 2\\\\ X_3: Predictor\ Variable \ 3\\\\ X_4: Predictor\ Variable \ 4\\\\ $$

$$ a: intercept\\\\\\ b_1 :coefficients \ of\ Variable \ 1\\\\ b_2: coefficients \ of\ Variable \ 2\\\\ b_3: coefficients \ of\ Variable \ 3\\\\ b_4: coefficients \ of\ Variable \ 4\\\\ $$

The equation is given by:

$$Yhat = a + b_1 X_1 + b_2 X_2 + b_3 X_3 + b_4 X_4$$

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

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

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

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

```
LinearRegression()
```

What is the value of the intercept(a)?

```
lm.intercept_
```

```
-15806.62462632922
```

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

```
lm.coef_
```

```
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:

$$Yhat = a + b_1 X_1 + b_2 X_2 + b_3 X_3 + b_4 X_4$$

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

```
# Write your code below and press Shift+Enter to execute
lm2 = LinearRegression()
lm2.fit(df[['normalized-losses' , 'highway-mpg']],df['price'])
```

```
LinearRegression()
```

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

```
array([   1.49789586, -820.45434016])
```

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.

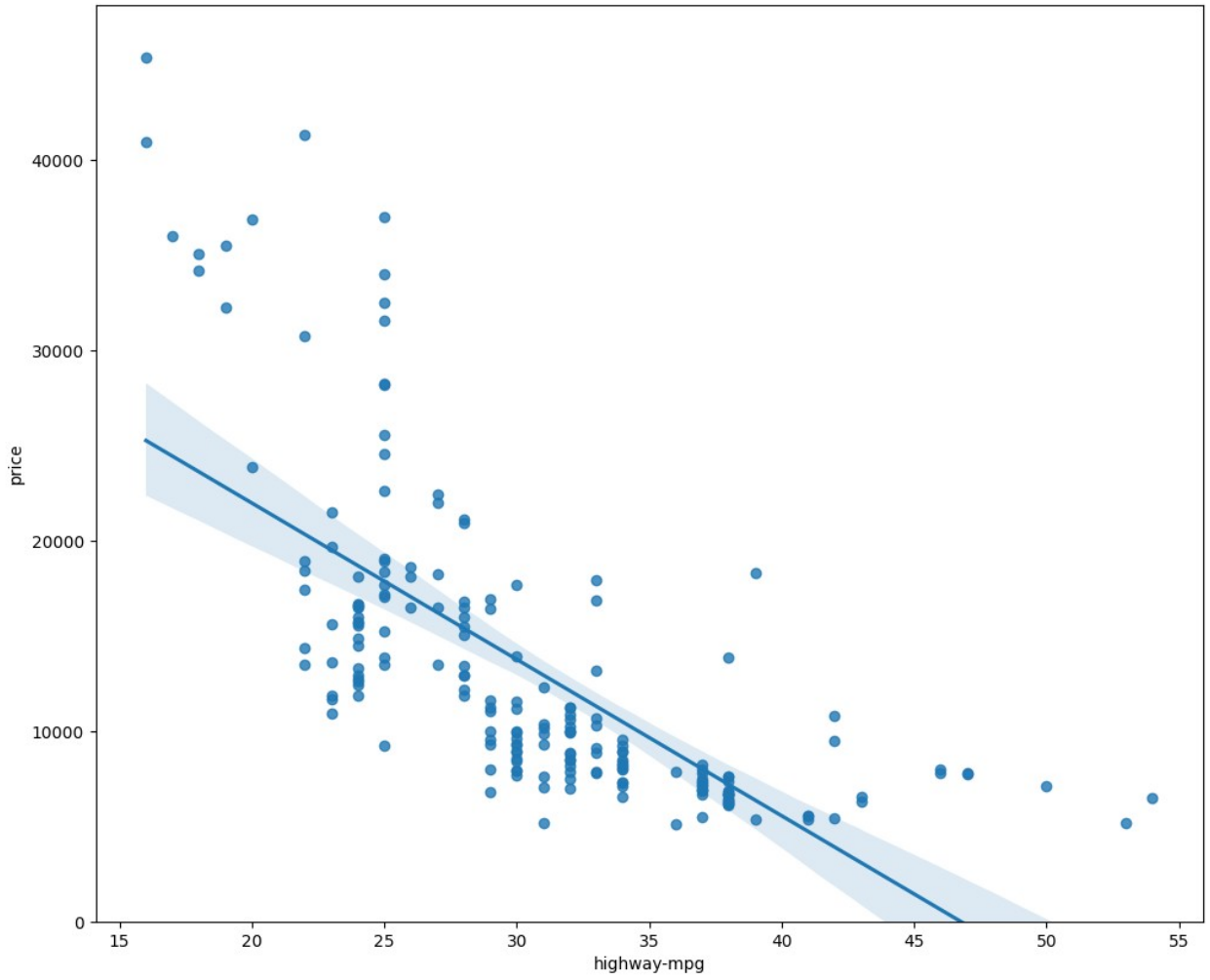
Import the visualization package, seaborn:

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

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

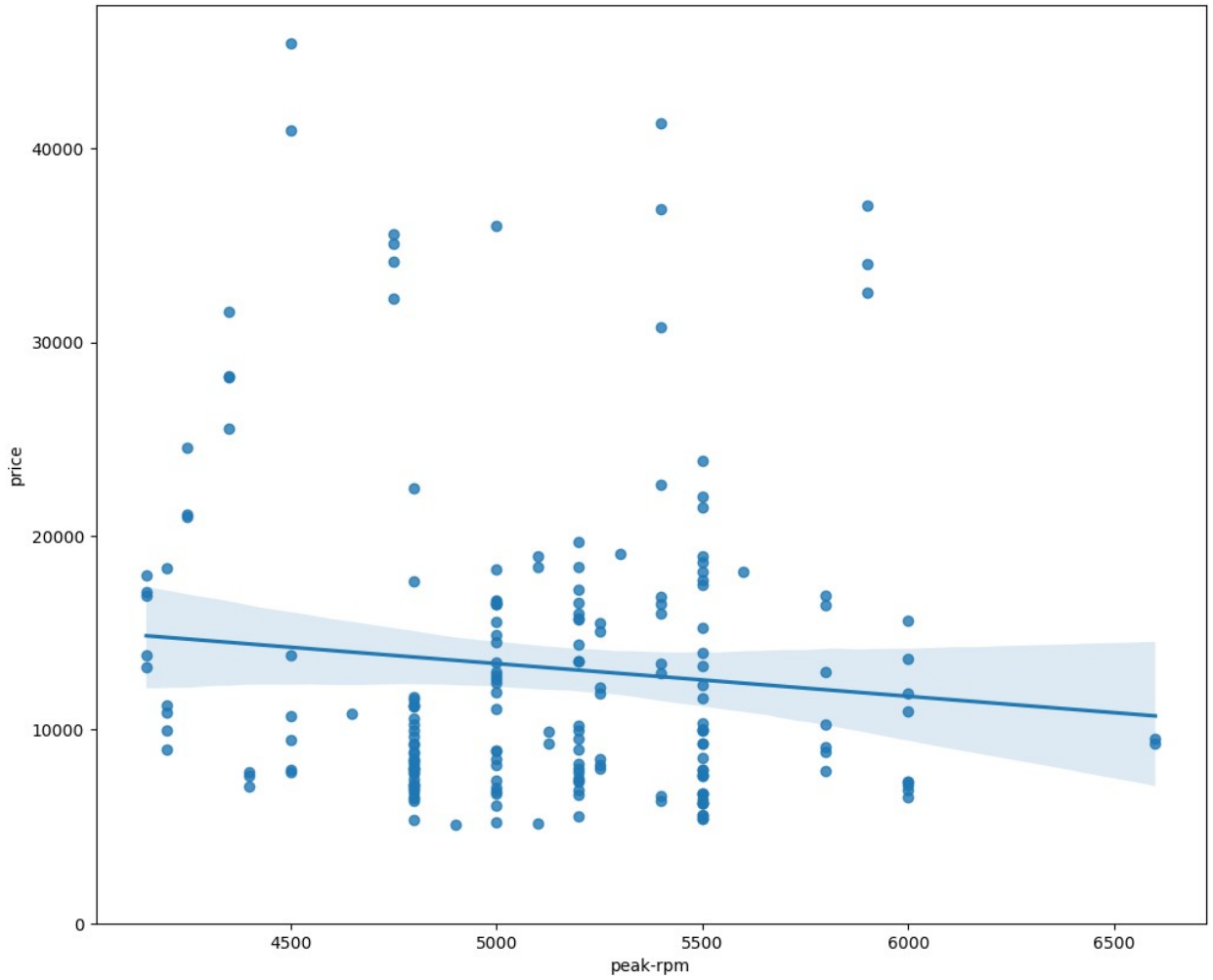
```
width = 12
height = 10
plt.figure(figsize=(width, height))
sns.regplot(x="highway-mpg", y="price", data=df)
plt.ylim(0,)
```

```
(0.0, 48180.777795527756)
```




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

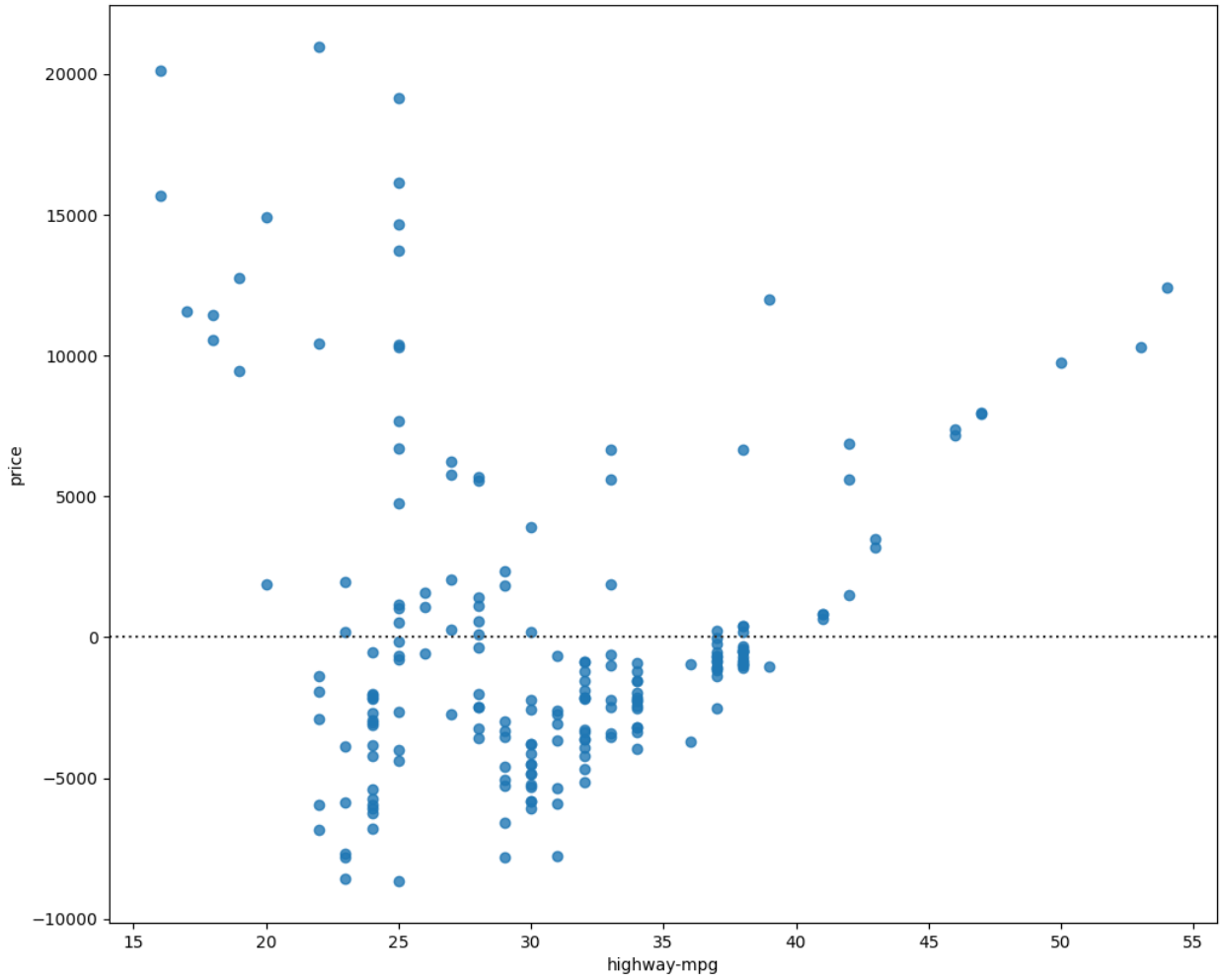
```
(0.0, 47414.1)
```




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

```
             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
```

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

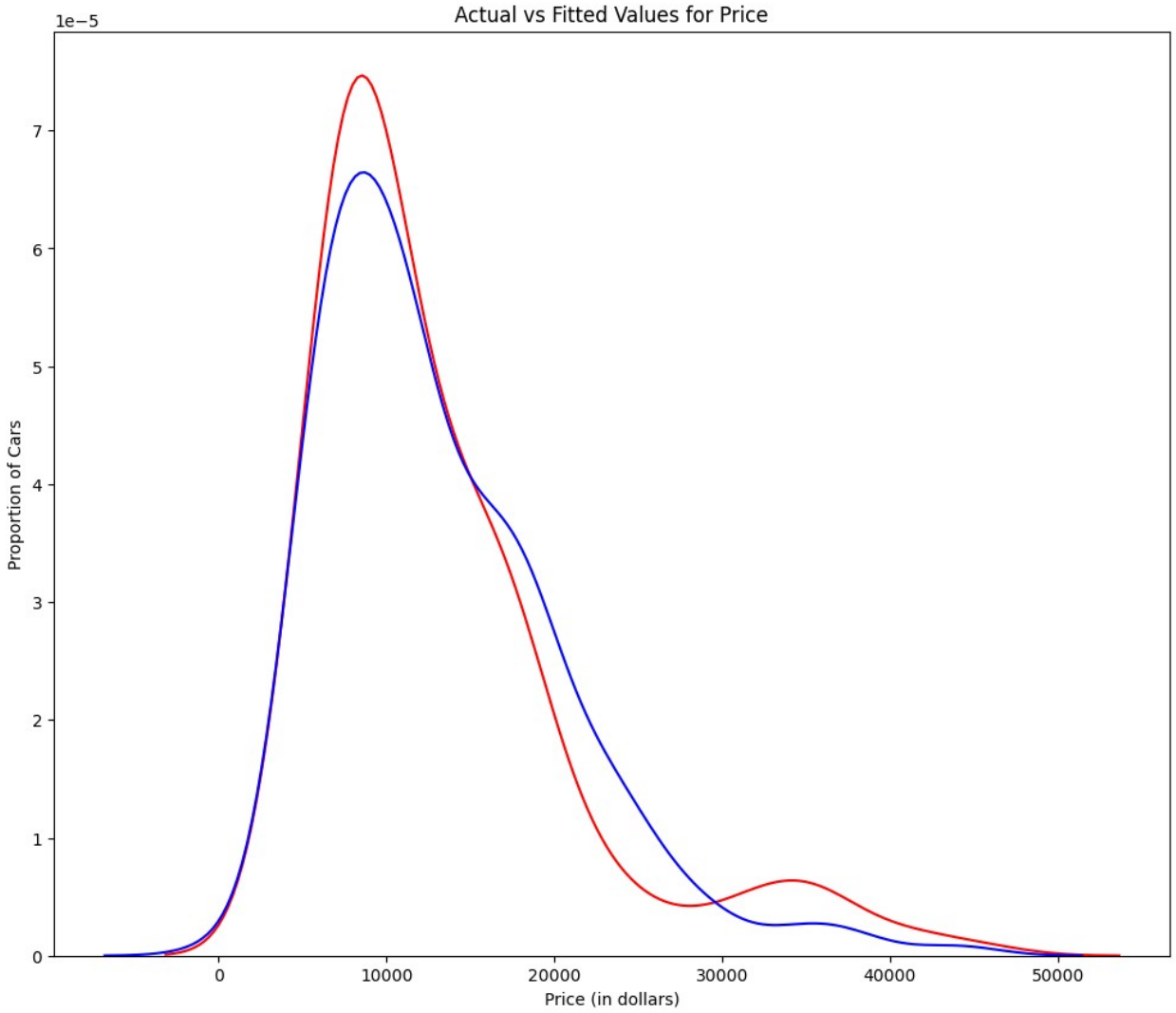


What is this plot telling us?

First, let's make a prediction:

```
Y_hat = lm.predict(Z)

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()

<ipython-input-109-7377bca648c1>:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

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).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax1 = sns.distplot(df['price'], hist=False, color="r", label="Actual
Value")
<ipython-input-109-7377bca648c1>:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

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).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(Y_hat, hist=False, color="b", label="Fitted Values" ,
ax=ax1)
```



$$Yhat = a + b_1 X + b_2 X^2$$

$$ Yhat = a + b_1 X +b_2 X^2 +b_3 X^3\\\\\\ $$

$$ Y = a + b_1 X +b_2 X^2 +b_3 X^3 ....\\\\ $$

```
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:

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

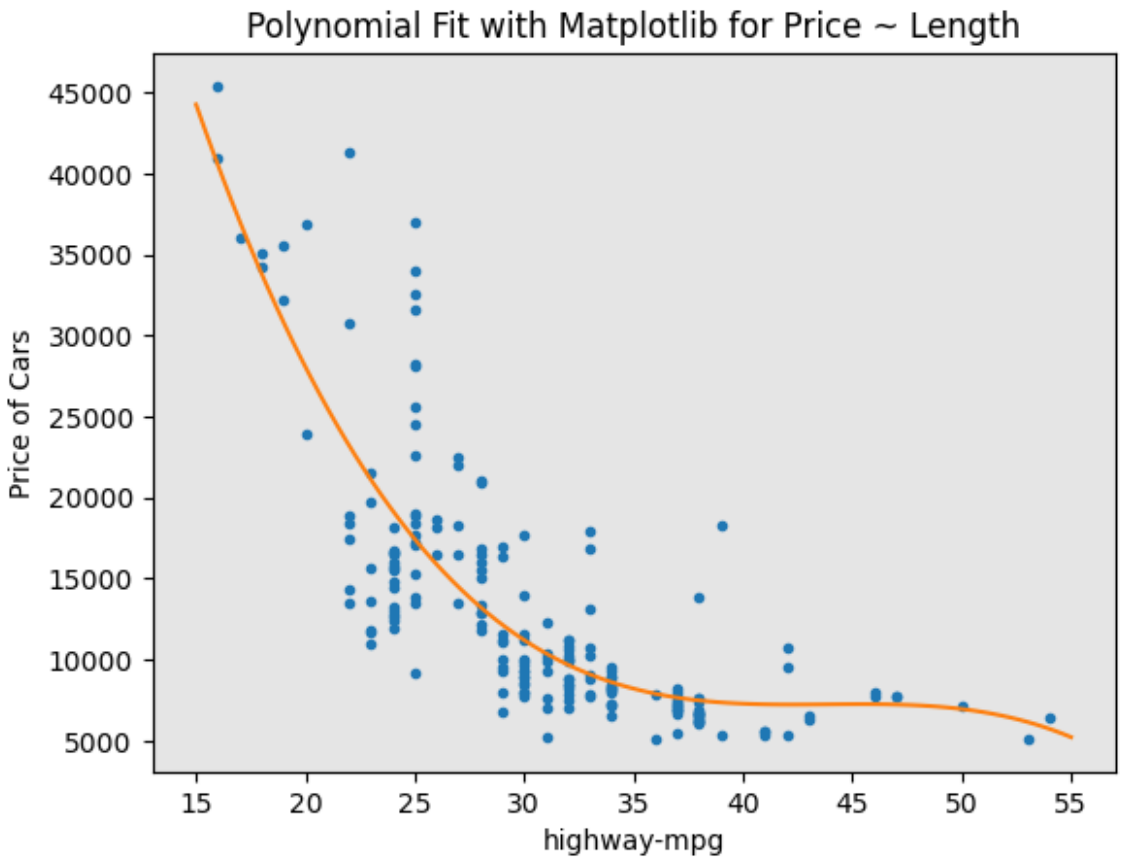
Let's fit the polynomial using the function polyfit, then use the function poly1d to display the polynomial function.

```
# 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:

```
PlotPolly(p, x, y, 'highway-mpg')
```

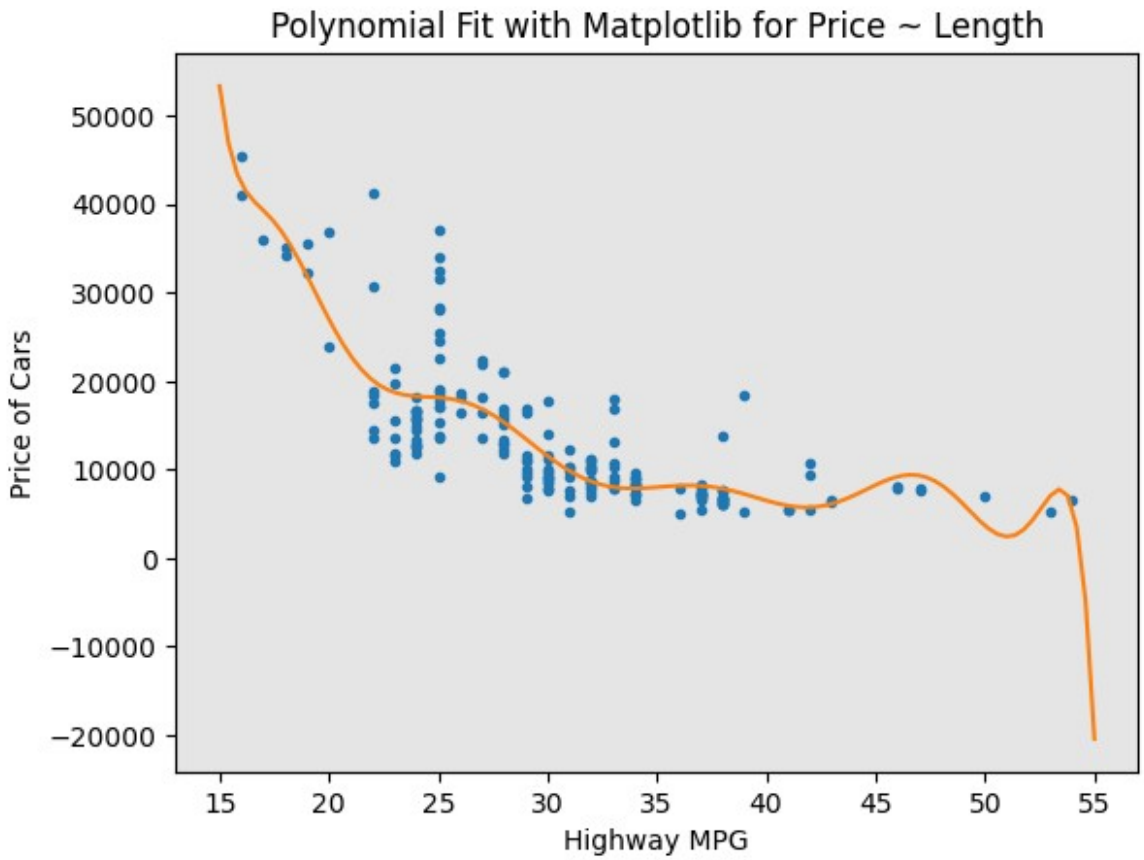



```
np.polyfit(x, y, 3)
```

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

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



$$Yhat = a + b_1 X_1 + b_2 X_2 + b_3 X_1 X_2 + b_4 X_1^2 + b_5 X_2^2$$

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

```
from sklearn.preprocessing import PolynomialFeatures
```

We create a PolynomialFeatures object of degree 2:

```
pr=PolynomialFeatures(degree=2)
pr

PolynomialFeatures()

Z_pr=pr.fit_transform(Z)
```

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

```
Z.shape

(201, 4)
```

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

```
Z_pr.shape
```

```
(201, 15)
```

```
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.

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

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

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

```
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.

```
Z = Z.astype(float)
pipe.fit(Z,y)
```

```
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.

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

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

```
# Write your code below and press Shift+Enter to execute
Input=[('scale',StandardScaler()),('model',LinearRegression())]

pipe=Pipeline(Input)
```

```
pipe.fit(Z,y)

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

```
array([13699.11161184, 13699.11161184, 19051.65470233, 10620.36193015,
       15521.31420211, 13869.66673213, 15456.16196732, 15974.00907672,
       17612.35917161, 10722.32509097])
```

R-squared

Mean Squared Error (MSE)

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))
```

```
The R-square is:  0.4965911884339176
```

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])
```

```
The output of the first four predicted value is:  [16236.50464347
16236.50464347 17058.23802179 13771.3045085 ]
```

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)
```

```
The mean square error of price and predicted value is:
31635042.944639888
```

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']))
```

```
The R-square is:  0.8093562806577457
```

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))
```

```
The mean square error of price and predicted value using multifit is:
11980366.87072649
```

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)
```

```
The R-square value is:  0.674194666390652
```

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

We can also calculate the MSE:

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

```
20474146.426361218
```

```
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
```

```
LinearRegression()
```

Produce a prediction:

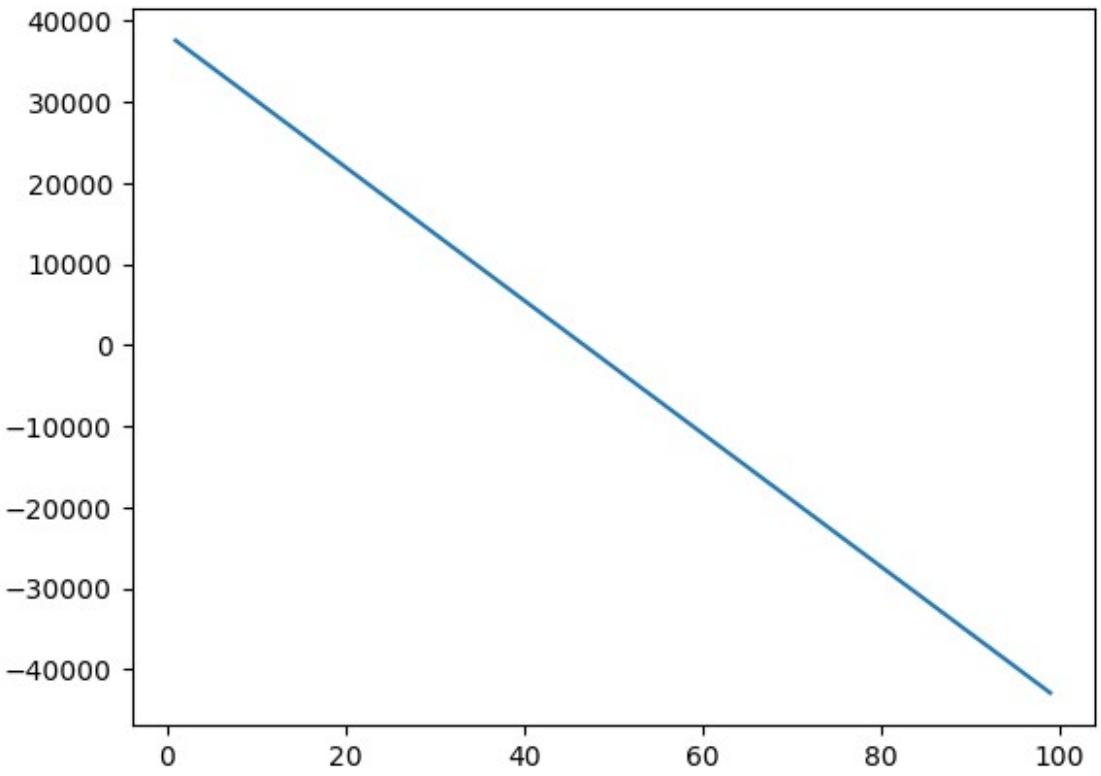
```
yhat=lm.predict(new_input)
yhat[0:5]
```

```
/lib/python3.11/site-packages/sklearn/base.py:465: UserWarning: X does
not have valid feature names, but LinearRegression was fitted with
feature names
  warnings.warn(
```

```
array([37601.57247984, 36779.83910151, 35958.10572319, 35136.37234487,
       34314.63896655])
```

We can plot the data:

```
plt.plot(new_input, yhat)
plt.show()
```



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

Thank you for completing this lab!

Author

Joseph Santarcangelo

Other Contributors

Mahdi Noorian PhD

Bahare Talayian

Eric Xiao

Steven Dong

Parizad

Hima Vasudevan

Fiorella Wenver

Yi Yao.

Abhishek Gagneja

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2023-09-28	2.3	Abhishek Gagneja	Updated instructions
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