Hands-on practice lab: Model Development

Estimated time needed: **45** minutes

In this lab, you will use the skills acquired in throughout the module, and use linear regression principles to create a model that predicts the Price of the laptop, based on one or more attributes of the dataset.

Objectives

After completing this lab you will be able to:

- Use Linear Regression in one variable to fit the parameters to a model
- Use Linear Regression in multiple variables to fit the parameters to a model
- Use Polynomial Regression in single variable tofit the parameters to a model
- Create a pipeline for performing linear regression using multiple features in polynomial scaling
- Evaluate the performance of different forms of regression on basis of MSE and R^2 parameters

Setup

For this lab, we will be using the following libraries:

- `pandas` for managing the data.
- `numpy` for mathematical operations.
- `sklearn` for machine learning and machine-learning-pipeline related functions.
- `seaborn` for visualizing the data.
- `matplotlib` for additional plotting tools.

The following required libraries are **not** pre-installed in the Skills Network Labs environment. **You will need to run the following cell** to install them:

```
import piplite
await piplite.install('seaborn')
```

Importing Required Libraries

We recommend you import all required libraries in one place (here):

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, r2_score
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
%matplotlib inline
```

Importing the dataset

Run the cell below to download the dataset into this environment.

This function will download the dataset into your browser

```
#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())
```

We put the file path along with a quotation mark so that pandas will read the file into a dataframe from that address. The file path can be either an URL or your local file address.

```
path = "https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DA0101EN-Coursera/
laptop_pricing_dataset_mod2.csv"
```

You will need to download the dataset using the download() function:

```
#you will need to download the dataset;
await download(path, "laptops.csv")
file_name="laptops.csv"
```

Load the dataset into a pandas dataframe

```
df = pd.read_csv(file_name, header=0)
```

> 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.

```
#https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
IBMDeveloperSkillsNetwork-DA0101EN-Coursera/
laptop_pricing_dataset_mod2.csv"
#df = pd.read_csv(filepath, header=None)

# show the first 5 rows using dataframe.head() method
print("The first 5 rows of the dataframe")
df.head(5)
```

```
The first 5 rows of the dataframe

   Unnamed: 0.1  Unnamed: 0 Manufacturer  Category  GPU  OS  CPU_core
\
0             0           0         Acer         4    2   1         5

1             1           1         Dell         3    1   1         3

2             2           2         Dell         3    1   1         7

3             3           3         Dell         4    2   1         5

4             4           4           HP         4    2   1         7


   Screen_Size_inch  CPU_frequency  RAM_GB  Storage_GB_SSD
Weight_pounds  \
0              14.0       0.551724       8             256
3.52800
1              15.6       0.689655       4             256
4.85100
2              15.6       0.931034       8             256
4.85100
3              13.3       0.551724       8             128
2.69010
4              15.6       0.620690       8             256
4.21155

   Price Price-binned  Screen-Full_HD  Screen-IPS_panel
0    978          Low               0                 1
1    634          Low               1                 0
2    946          Low               1                 0
3   1244          Low               0                 1
4    837          Low               1                 0
```

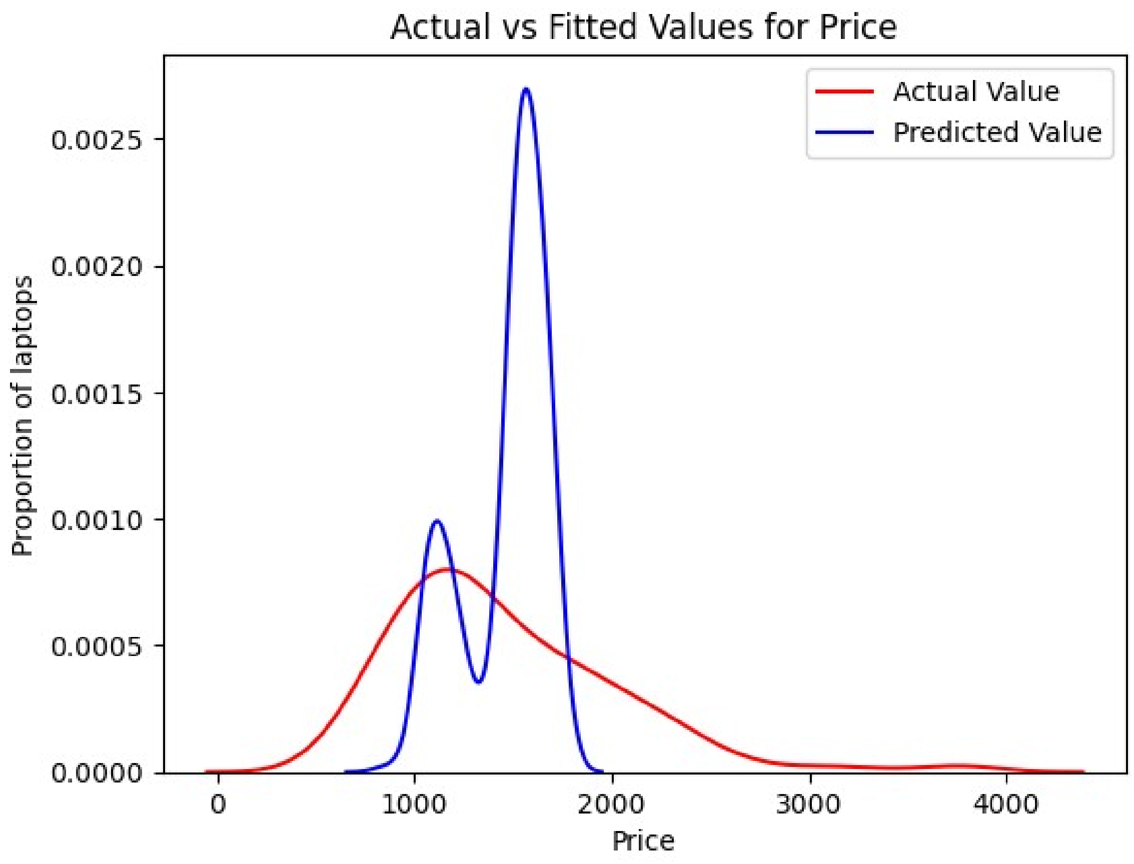
Task 1 : Single Linear Regression

You have learnt that "CPU_frequency" is the parameter with the lowest p-value among the different features of the dataset. Create a single feature Linear Regression model that fits the pair of "CPU_frequency" and "Price" to find the model for prediction.

```
# Write your code below and press Shift+Enter to execute
lm = LinearRegression()
X = df[['CPU_frequency']]
Y = df['Price']
lm.fit(X,Y)
Y_hat=lm.predict(X)
```

Generate the Distribution plot for the predicted values and that of the actual values. How well did the model perform?

```
# Write your code below and press Shift+Enter to execute
ax1 = sns.distplot(df['Price'], hist=False, color="r", label="Actual
Value")

# Create a distribution plot for predicted values
sns.distplot(Yhat, hist=False, color="b", label="Fitted Values" ,
ax=ax1)

plt.title('Actual vs Fitted Values for Price')
plt.xlabel('Price')
plt.ylabel('Proportion of laptops')
plt.legend(['Actual Value', 'Predicted Value'])
plt.show()
```



Evaluate the Mean Squared Error and R^2 score values for the model.

```
# Write your code below and press Shift+Enter to execute
mse_slr = mean_squared_error(df['Price'], Y_hat)
r2_score_slr = lm.score(X, Y)
print('The R-square for Linear Regression is: ', r2_score_slr)
print('The mean square error of price and predicted value is: ',
mse_slr)
```

```
The R-square for Linear Regression is:  0.13444363210243238
The mean square error of price and predicted value is:
284583.44058686297
```

Task 2 - Multiple Linear Regression

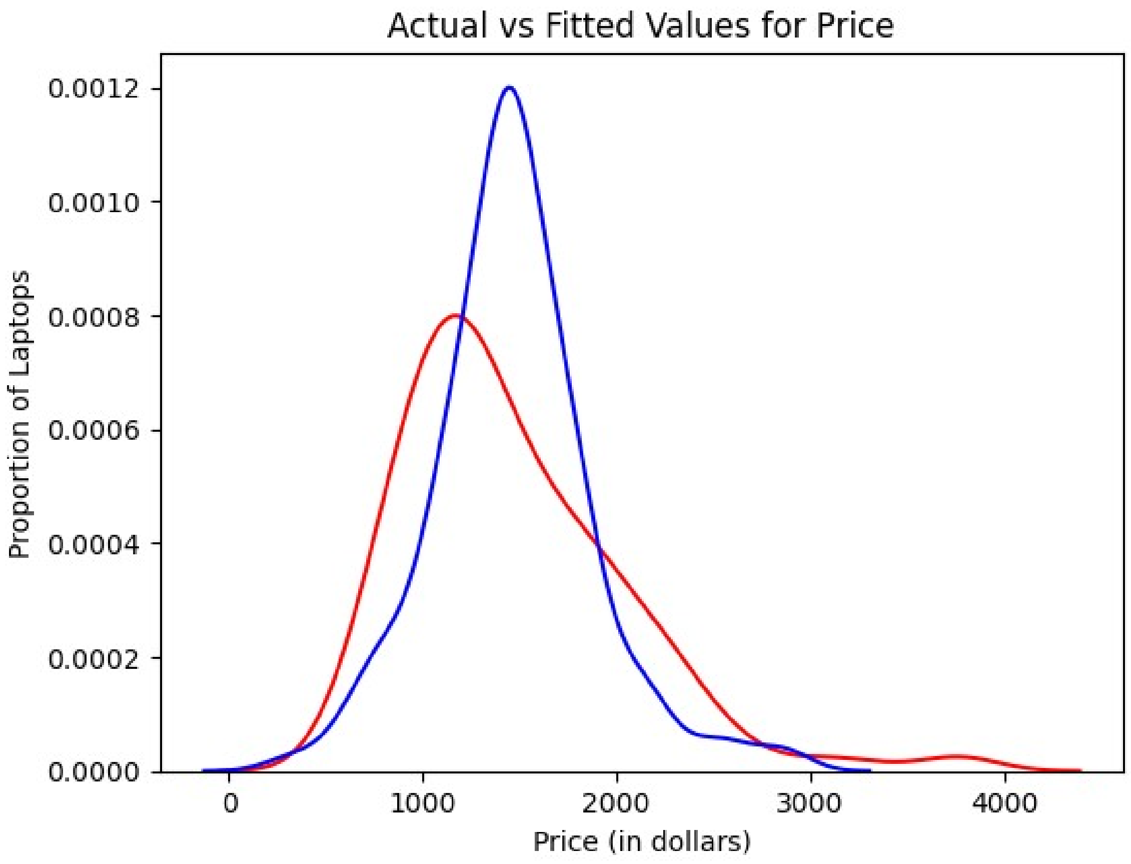
The parameters which have a low enough p-value so as to indicate strong relationship with the 'Price' value are 'CPU_frequency', 'RAM_GB', 'Storage_GB_SSD', 'CPU_core', 'OS', 'GPU' and 'Category'. Use all these variables to create a Multiple Linear Regression system.

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

```
Z =
df[['CPU_frequency','RAM_GB','Storage_GB_SSD','CPU_core','OS','GPU','C
ategory']]
lm1.fit(Z,Y)
Y_hat = lm1.predict(Z)
```

Plot the Distribution graph of the predicted values as well as the Actual values

```
# Write your code below and press Shift+Enter to execute
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')
plt.ylabel('Proportion of Laptops')
plt.show()
plt.close()
```



Find the R^2 score and the MSE value for this fit. Is this better or worst than the performance of Single Linear Regression?

```
# Write your code below and press Shift+Enter to execute
Y_predict_multifit = lm1.predict(Z)
lm1.fit(Z, df['Price'])
print('The R-square is: ', lm1.score(Z, df['Price']))
print('The mean square error of price and predicted value using 
multifit is: ', \
      mean_squared_error(df['Price'], Y_predict_multifit))

The R-square is:  0.5082509055187374
The mean square error of price and predicted value using multifit is: 
161680.57263893107
```

Task 3 - Polynomial Regression

Use the variable "CPU_frequency" to create Polynomial features. Try this for 3 different values of polynomial degrees. Remember that polynomial fits are done using `numpy.polyfit`.

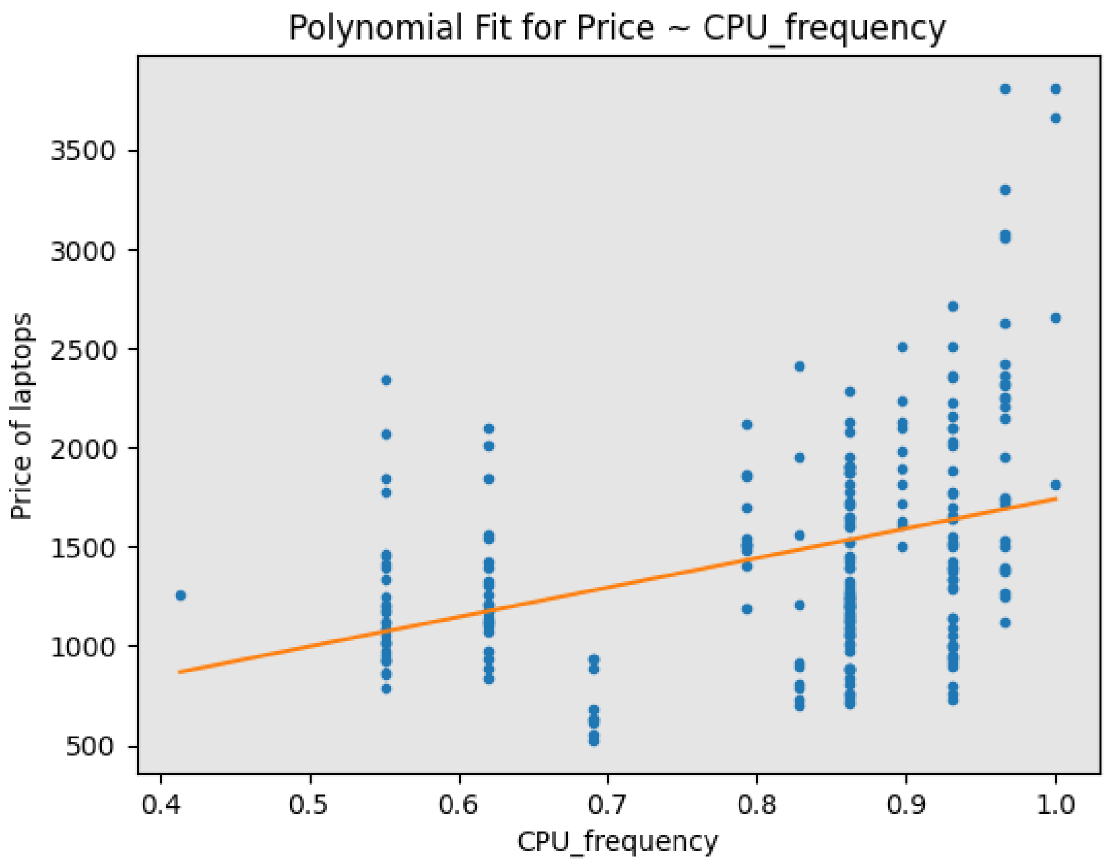
```
#  Write your code below and press Shift+Enter to execute
X = X.to_numpy().flatten()
f1 = np.polyfit(X, Y, 1)
p1 = np.poly1d(f1)

f3 = np.polyfit(X, Y, 3)
p3 = np.poly1d(f3)

f5 = np.polyfit(X, Y, 5)
p5 = np.poly1d(f5)
```

Plot the regression output against the actual data points to note how the data fits in each case. To plot the polynomial response over the actual data points, you have the function shown below.

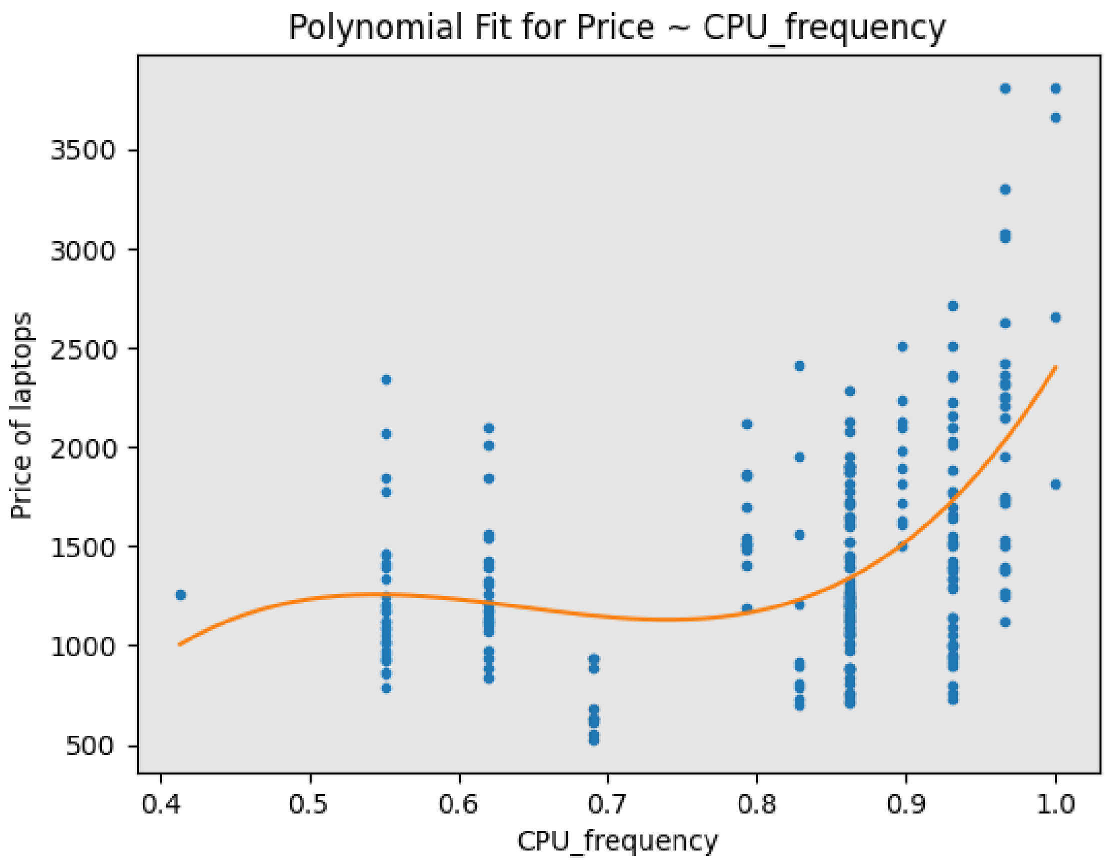
```
def PlotPolly(model, independent_variable, dependent_variabble, Name):
    x_new = 
np.linspace(independent_variable.min(),independent_variable.max(),100)
    y_new = model(x_new)

    plt.plot(independent_variable, dependent_variabble, '.', x_new, 
y_new, '-')
    plt.title(f'Polynomial Fit for Price ~ {Name}')
    ax = plt.gca()
    ax.set_facecolor((0.898, 0.898, 0.898))
    fig = plt.gcf()
    plt.xlabel(Name)
    plt.ylabel('Price of laptops')
```

Call this function for the 3 models created and get the required graphs.

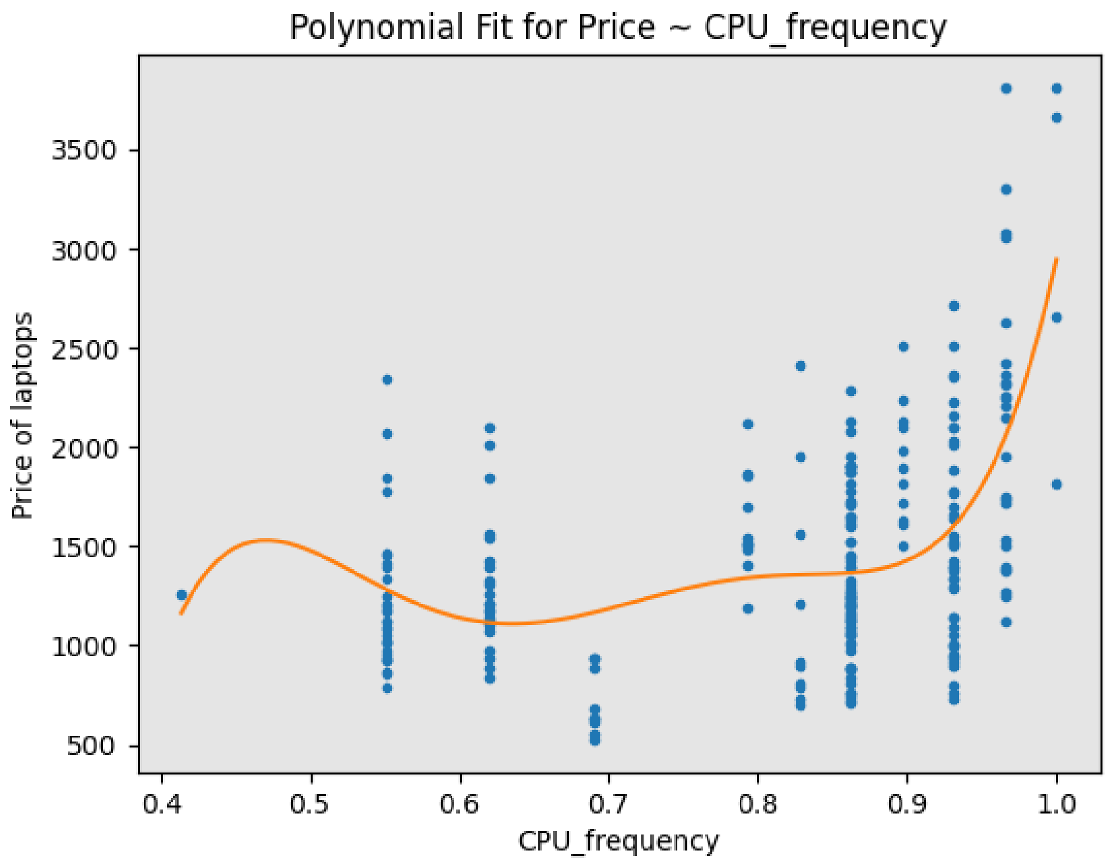
```
#  Write your code below and press Shift+Enter to execute
# Call for function of degree 1
PlotPolly(p1, X, Y, 'CPU_frequency')
```



```
#  Write your code below and press Shift+Enter to execute
# Call for function of degree 3
PlotPolly(p3, X, Y, 'CPU_frequency')
```



```
#  Write your code below and press Shift+Enter to execute
# Call for function of degree 5
PlotPolly(p5, X, Y, 'CPU_frequency')
```



Also, calculate the R^2 and MSE values for these fits. For polynomial functions, the function sklearn.metrics.r2_score will be used to calculate R^2 values.

```
#  Write your code below and press Shift+Enter to execute
r_squared_1 = r2_score(Y, p1(X))
mse_slr_1 = mean_squared_error(Y,p1(X))
print('The R-square value for 1st degree polynomial is: ',
r_squared_1)
print('The MSE value for 1st degree polynomial is: ', mse_slr_1)
r_squared_3 = r2_score(Y, p3(X))
mse_slr_3 = mean_squared_error(Y,p3(X))
print('The R-square value for 1st degree polynomial is: ',
r_squared_3)
print('The MSE value for 1st degree polynomial is: ', mse_slr_3)
r_squared_5 = r2_score(Y, p5(X))
mse_slr_5 = mean_squared_error(Y,p5(X))
print('The R-square value for 1st degree polynomial is: ',
r_squared_5)
print('The MSE value for 1st degree polynomial is: ', mse_slr_5)
```

```
The R-square value for 1st degree polynomial is:  0.13444363210243282
The MSE value for 1st degree polynomial is:  284583.4405868628
The R-square value for 1st degree polynomial is:  0.26692640796530986
The MSE value for 1st degree polynomial is:  241024.8630384881
The R-square value for 1st degree polynomial is:  0.3030822706443803
The MSE value for 1st degree polynomial is:  229137.29548053825
```

Task 4 - Pipeline

Create a pipeline that performs parameter scaling, Polynomial Feature generation and Linear regression. Use the set of multiple features as before to create this pipeline.

```
#  Write your code below and press Shift+Enter to execute
Input=[('scale',StandardScaler()), ('polynomial',
PolynomialFeatures(include_bias=False)), ('model',LinearRegression())]
pipe=Pipeline(Input)
Z = Z.astype(float)
pipe.fit(Z,Y)
ypipe=pipe.predict(Z)
```

Evaluate the MSE and R^2 values for the this predicted output.

```
#  Write your code below and press Shift+Enter to execute
print('MSE for multi-variable polynomial pipeline is: ',
mean_squared_error(Y, ypipe))
print('R^2 for multi-variable polynomial pipeline is: ', r2_score(Y,
ypipe))

MSE for multi-variable polynomial pipeline is:  244507.9894957983
R^2 for multi-variable polynomial pipeline is:  0.2563325298426047
```

You should now have seen that the values of R^2 increase as we go from Single Linear Regression to Multiple Linear Regression. Further, if we go for multiple linear regression extended with polynomial features, we get an even better R^2 value.

Congratulations! You have completed the lab

Authors

Abhishek Gagneja

Vicky Kuo

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2023-09-16	0.1	Abhishek Gagneja	Initial Version

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2023-09-19	0.2	Vicky Kuo	Created Reviewed and Revised