Department of Computer Science and Engineering (Data Science)

Subject: Machine Learning – I (DJ19MN4C2)

AY: 2022-23

Experiment 3

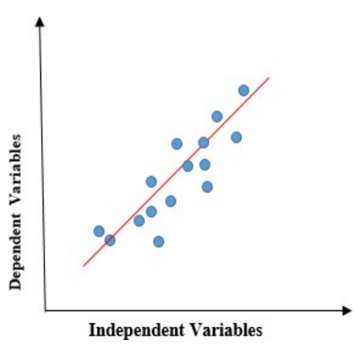
(Regression)

Name: Ayush Jain **SAPID:** 60004200132

Aim: Implement Linear Regression on the given Dataset and apply Regularization to overcome overfitting in the model.

Theory:

- **Linear Regression**: Linear regression is a quiet and simple statistical regression method used for predictive analysis and shows the relationship between the continuous variables. Linear regression shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis), consequently called linear regression. *If there is a single input variable (x), such linear regression is called **simple linear regression**. And if there is more than one input variable, such linear regression is called **multiple linear regression***. The linear regression model gives a sloped straight line describing the relationship within the variables.



The above graph presents the linear relationship between the dependent variable and independent variables. When the value of x (**independent variable**) increases, the value of y (**dependent variable**) is likewise increasing. The red line is referred to as the best fit straight line. Based on the given data points, we try to plot a line that models the points the best.

$$y = mx + b \implies y = a_0 + a_1 x$$

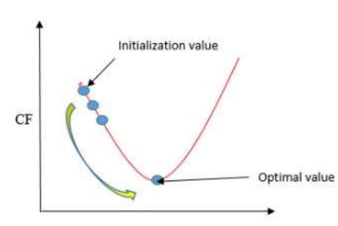
y= Dependent Variable; x= Independent Variable; a0= intercept; a1 = Linear regression coefficient.

- **Cost function:** The cost function helps to figure out the best possible values for a0 and a1, which provides the best fit line for the data points. Cost function optimizes the regression coefficients or weights and measures how a linear regression model is performing. The cost function is used to find the accuracy of the **mapping function** that maps the input variable to the output variable. This mapping function is also known as **the Hypothesis function**. In Linear Regression, **Mean Squared Error (MSE)** cost function is used, which is the average of squared error that occurred between the predicted values and actual values. *By simple linear equation y=mx+b we can calculate MSE as: Let's y = actual values, y_i = predicted values*

$$MSE = \frac{1}{N}\sum_{i=1}^{n}(y_i - (mx_i + b))^2$$

Using the MSE function, we will change the values of a0 and a1 such that the MSE value settles at the minima. Model parameters ***xi, b (a_0, a_1)*** can be manipulated to minimize the cost function. These parameters can be determined using the gradient descent method so that the cost function value is minimum.

- **Gradient descent:** Gradient descent is a method of updating a0 and a1 to minimize the cost function (MSE). A regression model uses gradient descent to update the coefficients of the line (a0, a1 => xi, b) by reducing the cost function by a random selection of coefficient values and then iteratively update the values to reach the minimum cost function.



To update a_0 and a_1, we take gradients from the cost function. To find these gradients, we take partial derivatives for a_0 and a_1.

$$J = \frac{1}{n}\sum_{i=1}^{n}(a_0 + a_1 \cdot x_i - y_i)^2$$

$$\frac{\partial J}{\partial a_0} = \frac{2}{n}\sum_{i=1}^{n}(a_0 + a_1 \cdot x_i - y_i)$$

$$\frac{\partial J}{\partial a_1} = \frac{2}{n}\sum_{i=1}^{n}(a_0 + a_1 \cdot x_i - y_i) \cdot x_i$$

$$\frac{\partial J}{\partial a_0} = \frac{2}{n}\sum_{i=1}^{n}(pred_i - y_i)$$

$$\frac{\partial J}{\partial a_1} = \frac{2}{n}\sum_{i=1}^{n}(pred_i - y_i) \cdot x_i$$

$$a_0 = a_0 - \alpha \cdot \frac{2}{n}\sum_{i=1}^{n}(pred_i - y_i)$$

$$a_1 = a_1 - \alpha \cdot \frac{2}{n}\sum_{i=1}^{n}(pred_i - y_i) \cdot x_i$$

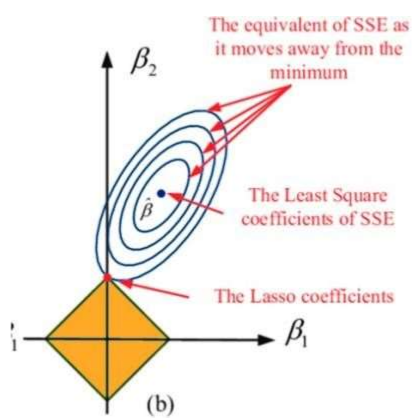
- **Regularization:** When linear regression is underfitting there is no other way (given you can't add more data) then to increase complexity of the model making it polynomial regression (cubic, quadratic, etc…) or using other complex model to capture data that linear regression cannot capture due to its simplicity. When linear regression is overfitting, number of columns(independent variables) approach number of observations there are two ways to mitigate it

  1. Add more observations
  2. Regularization

  Since adding more observations is time consuming and often not provided we will use regularization technique to mitigate overfitting. There are multiple regularization techniques, all share the same concept of **adding constraints on weights** of independent variables(except theta_0) however they differ in way of constraining. We will go through three most popular regularization techniques: Ridge regression (L2) and Lasso regression (L1)

- **Lasso Regression**

The word "LASSO" denotes Least Absolute Shrinkage and Selection Operator. Lasso regression follows the regularization technique to create prediction. It is given more priority over the other regression methods because it gives an accurate prediction. Lasso regression model uses shrinkage technique. In this technique, the data values are shrunk towards a central point similar to the concept of mean. The lasso regression algorithm suggests a simple, sparse models (i.e. models with fewer parameters), which is well-suited for models or data showing high levels of multicollinearity or when we would like to automate certain parts of model selection, like variable selection or parameter elimination using feature engineering. Lasso Regression algorithm utilises L1 regularization technique It is taken into consideration when there are more number of features because it automatically performs feature selection.



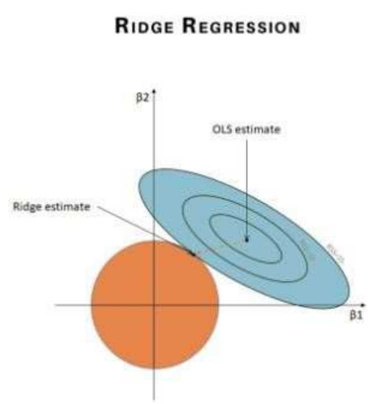
Residual Sum of Squares + λ * (Sum of the absolute value of the coefficients)

The equation looks like:

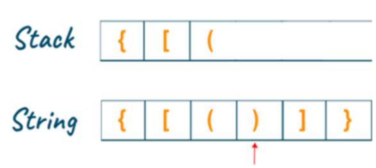
$$\sum_{i=1}^{n}(y_i - \sum_{j} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

- **Ridge Regression**

Ridge Regression is another type of regression algorithm in data science and is usually considered when there is a high correlation between the independent variables or model parameters. As the value of correlation increases the least square estimates evaluates unbiased values. But if the collinearity in the dataset is very high, there can be some bias value. Therefore, we create a bias matrix in the equation of Ridge Regression algorithm. It is a useful regression method in which the model is less susceptible to overfitting and hence the model works well even if the dataset is very small.



The cost function for ridge regression algorithm is:



Where λ is the penalty variable. λ given here is denoted by an alpha parameter in the ridge function. Hence, by changing the values of alpha, we are controlling the penalty term. Greater the values of alpha, the higher is the penalty and therefore the magnitude of the coefficients is reduced.We can conclude that it shrinks the parameters. Therefore, it is used to prevent multicollinearity, it also reduces the model complexity by shrinking the coefficient.

Lab Assignments to complete in this session

Use the given dataset and perform the following tasks:

Dataset 1: food_truck_data.csv

Dataset 2: housing.csv

1. Perform Linear Regression on Dataset 1 by computing cost function and gradient descent from scratch.

2. Use sklearn to perform linear regression on Dataset 2, show the scatter plot for best fit line using matplotlib and show the results using MSE.

3. To perform regularization on linear model build using Linear Regression on Dataset2.

Code:

Importing Libraries

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

Loading the dataset

```
df=pd.read_csv('/content/foodtruck (1).txt',sep=",")
df
```

	Population	Profit
0	6.1101	17.59200
1	5.5277	9.13020
2	8.5186	13.66200
3	7.0032	11.85400
4	5.8598	6.82330
...	...	...
92	5.8707	7.20290
93	5.3054	1.98690
94	8.2934	0.14454
95	13.3940	9.05510
96	5.4369	0.61705

97 rows × 2 columns

```
x=df.iloc[:,0].values
y=df.iloc[:,1].values
print(x)
print(y)
```

```
[ 6.1101  5.5277  8.5186  7.0032  5.8598  8.3829  7.4764  8.5781  6.4862
  5.0546  5.7107 14.164   5.734   8.4084  5.6407  5.3794  6.3654  5.1301
  6.4296  7.0708  6.1891 20.27    5.4901  6.3261  5.5649 18.945  12.828
 10.957  13.176  22.203   5.2524  6.5894  9.2482  5.8918  8.2111  7.9334
  8.0959  5.6063 12.836   6.3534  5.4069  6.8825 11.708   5.7737  7.8247
  7.0931  5.0702  5.8014 11.7     5.5416  7.5402  5.3077  7.4239  7.6031
  6.3328  6.3589  6.2742  5.6397  9.3102  9.4536  8.8254  5.1793 21.279
 14.908  18.959   7.2182  8.2951 10.236   5.4994 20.341  10.136   7.3345
  6.0062  7.2259  5.0269  6.5479  7.5386  5.0365 10.274   5.1077  5.7292
  5.1884  6.3557  9.7687  6.5159  8.5172  9.1802  6.002   5.5204  5.0594
  5.7077  7.6366  5.8707  5.3054  8.2934 13.394   5.4369]
[17.592    9.1302  13.662   11.854    6.8233  11.886    4.3483  12.
  6.5987   3.8166   3.2522  15.505    3.1551   7.2258   0.71618  3.5129
  5.3048   0.56077  3.6518   5.3893   3.1386  21.767    4.263    5.1875
  3.0825  22.638   13.501    7.0467  14.692   24.147   -1.22     5.9966
 12.134    1.8495   6.5426   4.5623   4.1164   3.3928  10.117    5.4974
  0.55657  3.9115   5.3854   2.4406   6.7318   1.0463   5.1337   1.844
  8.0043   1.0179   6.7504   1.8396   4.2885   4.9981   1.4233  -1.4211
  2.4756   4.6042   3.9624   5.4141   5.1694  -0.74279 17.929   12.054
 17.054    4.8852   5.7442   7.7754   1.0173  20.992    6.6799   4.0259
  1.2784   3.3411  -2.6807   0.29678  3.8845   5.7014   6.7526   2.0576
  0.47953  0.20421  0.67861  7.5435   5.3436   4.2415   6.7981   0.92695
  0.152    2.8214   1.8451   4.2959   7.2029   1.9869   0.14454  9.0551
  0.61705]
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=42)
print(x_train)
print(x_test)
print(y_train)
print(y_test)
```

```
[ 5.8918   5.0546   5.7292 14.164    7.2182 13.394    5.2524 13.176    6.002
  8.3829   7.0931 20.341    7.9334   6.3654   6.0062   8.2111   8.5781   6.3589
 10.957    7.0708   5.1077 18.945    7.6031   8.4084   5.5649   7.0032   5.1301
 12.836    6.4862   7.5386   7.4764 10.274    8.0959   8.5172   6.2742   5.4369
  6.3328   5.7737   7.5402   8.2951   5.0702 10.236    5.1793   8.2934   5.0365
  6.8825   9.3102 11.7      6.5159   5.6397   9.2482   7.6366   9.4536 14.908
  5.7077   5.6063 22.203    5.5277   7.4239 20.27     8.5186   6.3261   5.5204
  5.0269   9.1802   6.3557   6.1891   8.8254   7.3345   5.6407   5.8707   5.3077]
[21.279    5.4069   5.3054   6.4296   5.1884   9.7687 18.959   11.708    5.7107
  6.1101   6.5894   6.5479   5.8014 12.828    7.8247   5.8598   5.4901   5.734
  5.0594   7.2259   5.5416 10.136    5.4994   5.3794   6.3534]
[ 1.8495   3.8166   0.47953 15.505    4.8852   9.0551  -1.22    14.692
  0.92695 11.886    1.0463  20.992    4.5623   5.3048   1.2784   6.5426
 12.      -1.4211   7.0467   5.3893   2.0576  22.638    4.9981   7.2258
  3.0825  11.854    0.56077 10.117    6.5987   3.8845   4.3483   6.7526
  4.1164   4.2415   2.4756   0.61705  1.4233   2.4406   6.7504   5.7442
  5.1337   7.7754  -0.74279  0.14454  5.7014   3.9115   3.9624   8.0043
  5.3436   4.6042  12.134    4.2959   5.4141  12.054    1.8451   3.3928
 24.147    9.1302   4.2885  21.767   13.662    5.1875   0.152   -2.6807
  6.7981   0.67861  3.1386   5.1694   4.0259   0.71618  7.2029   1.8396 ]
[17.929    0.55657  1.9869   3.6518   0.20421  7.5435  17.054    5.3854
  3.2522  17.592    5.9966   0.29678  1.844   13.501    6.7318   6.8233
  4.263    3.1551   2.8214   3.3411   1.0179   6.6799   1.0173   3.5129
  5.4974 ]
```

```
[ ] from sklearn.linear_model import LinearRegression
x_train=np.reshape(x_train,(-1,1))
x_test=np.reshape(x_test,(-1,1))
reg = LinearRegression().fit(x_train, y_train)
pred=reg.predict(x_test)
print(pred)
```

```
[22.72210566  2.23334237  2.10231941  3.55351161  1.95128802  7.86384417
 19.72729521 10.36722171  2.62550763  3.14107974  3.75979209  3.70622112
  2.74258923 11.81299227  5.35439956  2.81797584  2.34074247  2.65558482
  1.78476623  4.58142866  2.40722209  8.33797946  2.35274753  2.19784354
  3.45514758]
```

```
reg.coef_
```

```
array([1.29086657])
```

```
[ ] reg.n_features_in_
```

```
1
```

```
[ ] reg.rank_
```

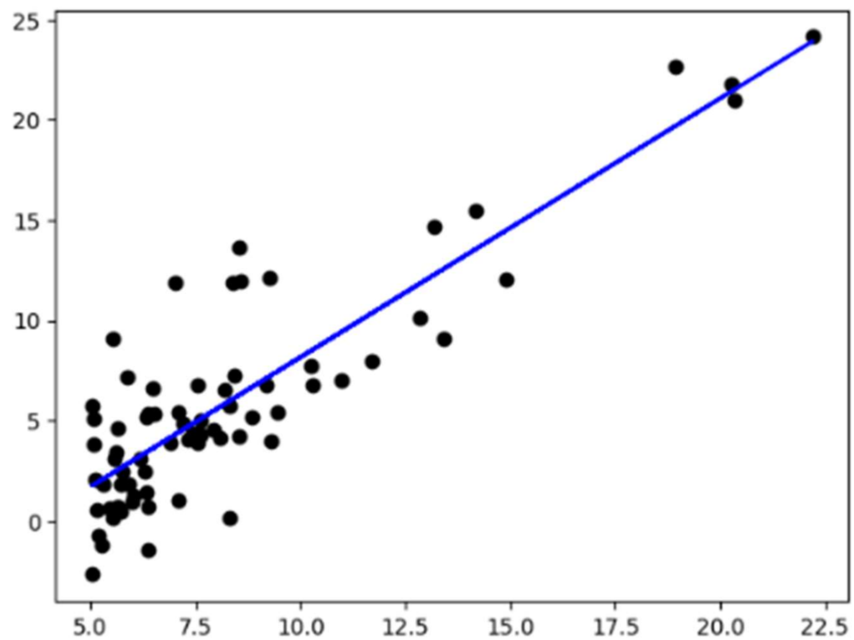
```
1
```

```
[ ] reg.score(x_test,y_test)
```

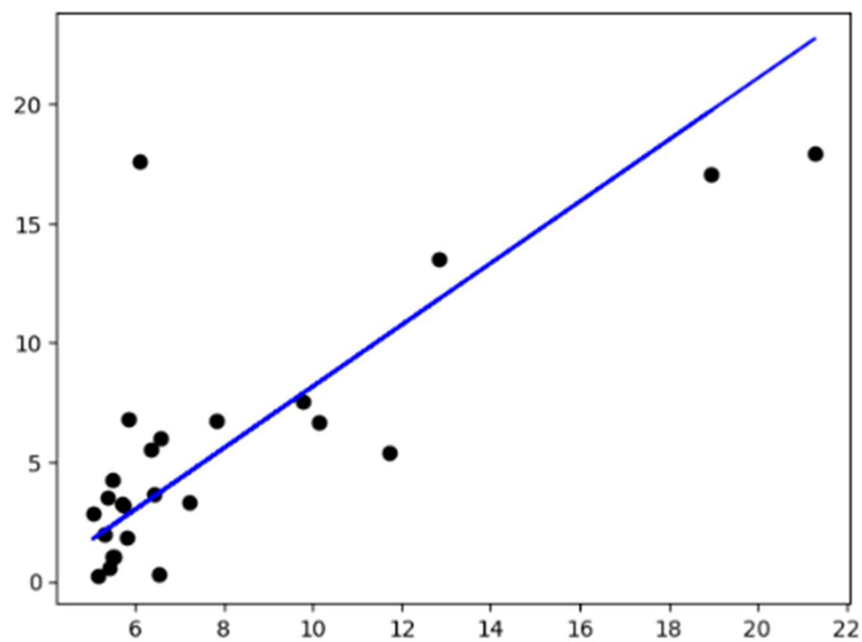
```
0.5210382872605228
```

```
[ ] pred_train=reg.predict(x_train)
```

```
plt.scatter(x_train,y_train,color='black')
plt.plot(x_train,pred_train,color='blue')
plt.show()
```



```
plt.scatter(x_test,y_test, color="black")
plt.plot(x_test,pred,color='blue')
plt.show()
```



For DATASET 2:

```
[1] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.linear_model import LinearRegression
    from sklearn.metrics import mean_squared_error
```

```python
df=pd.read_csv('Housing (1).csv')
df
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished
...	...	...	...	...	...	...	...	...	...	...	...	...	...
540	1820000	3000	2	1	1	yes	no	yes	no	no	2	no	unfurnished
541	1767150	2400	3	1	1	no	no	no	no	no	0	no	semi-furnished
542	1750000	3620	2	1	1	yes	no	no	no	no	0	no	unfurnished
543	1750000	2910	3	1	1	no	no	no	no	no	0	no	furnished
544	1750000	3850	3	1	2	yes	no	no	no	no	0	no	unfurnished

545 rows × 13 columns

```python
x=df.iloc[:,0].values
y=df.iloc[:,1].values
print(x)
print(y)
```

```
 3118850 3115000 3115000 3115000 3087000 3080000 3080000 3080000
 3080000 3045000 3010000 3010000 3010000 3010000 3010000 3010000
 3010000 3003000 2975000 2961000 2940000 2940000 2940000 2940000
 2940000 2940000 2940000 2940000 2870000 2870000 2870000 2870000
 2852500 2835000 2835000 2835000 2800000 2800000 2730000 2730000
 2695000 2660000 2660000 2660000 2660000 2660000 2660000 2660000
 2653000 2653000 2604000 2590000 2590000 2590000 2520000 2520000
 2520000 2485000 2485000 2450000 2450000 2450000 2450000 2450000
 2450000 2408000 2380000 2380000 2380000 2345000 2310000 2275000
 2275000 2275000 2240000 2233000 2135000 2100000 2100000 2100000
 1960000 1890000 1890000 1855000 1820000 1767150 1750000 1750000
 1750000]
[ 7420  8960  9960  7500  7420  7500  8580 16200  8100  5750 13200  6000
  6550  3500  7800  6000  6600  8500  4600  6420  4320  7155  8050  4560
  8800  6540  6000  8875  7950  5500  7475  7000  4880  5960  6840  7000
  7482  9000  6000  6000  6550  6360  6480  6000  6000  6000  6000  6600
  4300  7440  7440  6325  6000  5150  6000  6000 11440  9000  7680  6000
  6000  8880  6240  6360 11175  8880 13200  7700  6000 12090  4000  6000
  5020  6600  4040  4260  6420  6500  5700  6000  6000  4000 10500  6000
  3760  8250  6670  3960  7410  8580  5000  6750  4800  7200  6000  4100
  9000  6400  6600  6000  6600  5500  5500  6350  5500  4500  5450  6420
  3240  6615  6600  8372  4300  9620  6800  8000  6900  3700  6420  7020
  6540  7231  6254  7320  6525 15600  7160  6500  5500 11460  4800  5828
  5200  4800  7000  6000  5400  4640  5000  6360  5800  6660 10500  4800
  4700  5000 10500  5500  6360  6600  5136  4400  5400  3300  3650  6100
  6900  2817  7980  3150  6210  6100  6600  6825  6710  6450  7800  4600
```

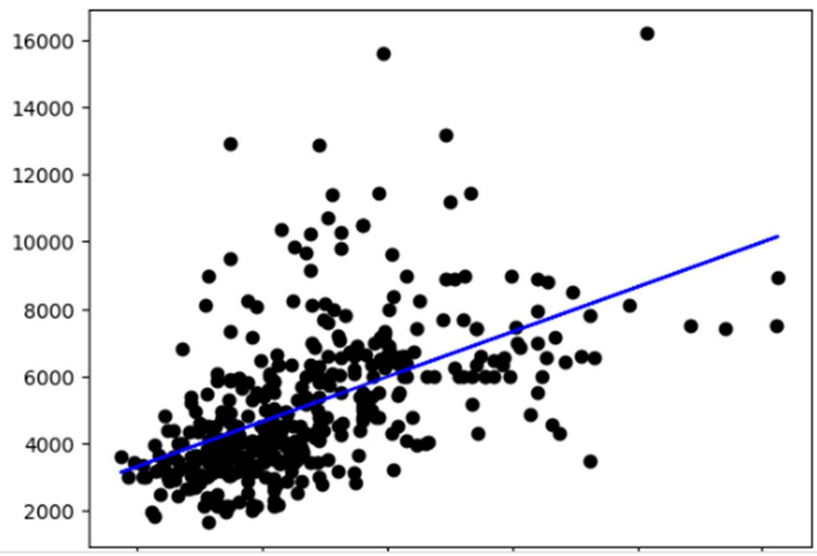
```
[6] from sklearn.model_selection import train_test_split
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=42)
```

```
from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(x_train, y_train)
y_pred=reg.predict(x_test)
print(y_pred)
```

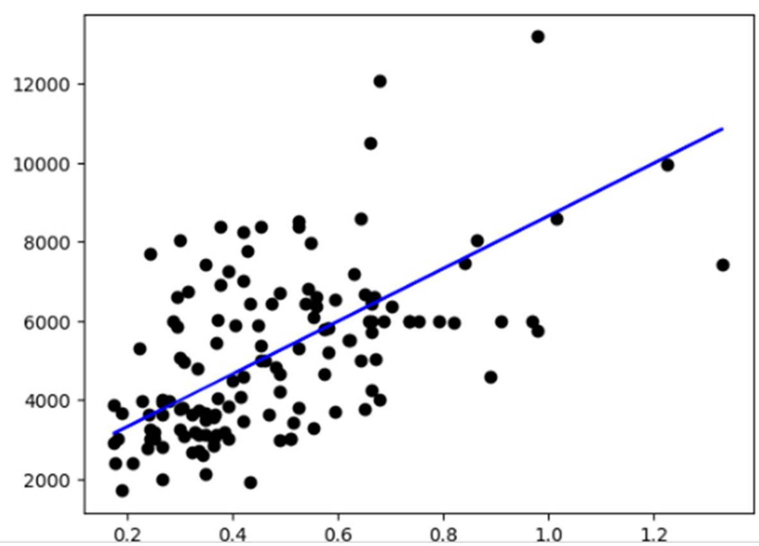
```
[ 4687.97629657  6417.00136832  4454.32425985  6276.81014628
  3846.82896437  5248.7411847   5482.39322143  5010.41610725
  3613.17692765  4215.99918239  8753.52173554  3753.36814968
  4220.67222312  4220.67222312  3496.35090928  3753.36814968
  3753.36814968  6884.30544176  3940.28977906  3893.55937171
  6463.73177566  5599.21923979  3239.33366889  5482.39322143
  4776.76407052 10155.43395588  4033.75059375  5388.93240674
  8519.86969882  3659.90733499  6510.462183    4314.13303781
  6417.00136832  3940.28977906  4197.30701945  4781.43711126
  5248.7411847   4173.94181578  4314.13303781  3566.4465203
  5645.94964713  4430.95905617  6417.00136832  5253.41422544
  4080.48100109  4968.35874064  6370.27096097  5809.50607284
  3982.34714567  3192.60326154  7585.26155193  3613.17692765
  4828.1675186   4652.92849106  4136.5574899   3145.8728542
  8519.86969882  3379.52489092  4874.89792595  4010.38539007
  4547.78507454  4314.13303781  5150.60732928  4033.75059375
  5015.08914798  6323.54055363  6440.36657199  5388.93240674
  5015.08914798  6417.00136832  4407.5938525   5716.04525815
  4501.05466719  6393.63616464  4127.21140844  6417.00136832
  5108.54996267  5202.01077736  6557.19259035  3468.31266488
  6674.01860871  4758.07190759  6113.25372058  6323.54055363
```

```
[8] pred_train=reg.predict(x_train)
```

```
plt.scatter(x_train,y_train,color='black')
plt.plot(x_train,pred_train,color='blue')
plt.show()
```




```
plt.scatter(x_test,y_test, color="black")
plt.plot(x_test,y_pred,color='blue')
plt.show()
```



```
[27] from sklearn.metrics import mean_squared_error
     mse = mean_squared_error(y_test, y_pred)
     print('Mean Squared Error:', mse)

     Mean Squared Error: 3033418.448648035
```

```
from sklearn.linear_model import Ridge
ridge = Ridge(alpha=1)
ridge.fit(x_train, y_train)
```

```
Ridge
Ridge(alpha=1)
```

```
y_pred = ridge.predict(x_test)
mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error:', mse)

Mean Squared Error: 3033418.448648034
```

Conclusion : We successfully implemented Linear Regression on the given Dataset and apply Regularization to overcome overfitting in the model.