Department of Computer Science and Engineering (Data Science)

Subject: Machine Learning – I (DJ19DSC402)

AY: 2023-24

Experiment 1

Name : Sayantan Mukherjee **SAP-ID:60009220131**

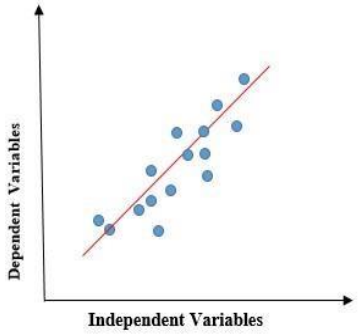
Batch:D2-2 **Date:08/02/2024**

(Regression)

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. *To calculate best-fit line linear regression uses a traditional slope-intercept form.*

$$y = mx+b \quad \Longrightarrow \quad y = a_0+a_1x$$

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

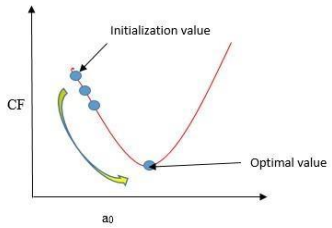
- **Cost function:** The cost function helps to figure out the best possible values for a0 and a1, which provides the best fit line for the data points. Cost function optimizes the regression coefficients or weights and measures how a linear regression model is performing. The cost function is used to 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.

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

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

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

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

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

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

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

partial derivates are the gradients and they are used to update the

- **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,

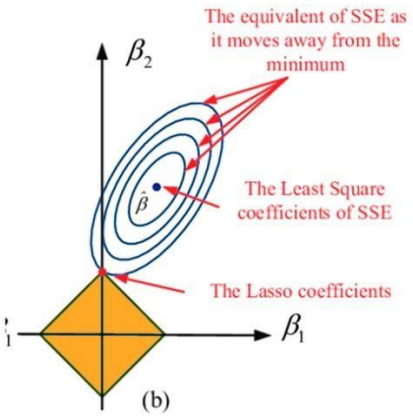
quadratic, etc…) or using other complex model to capture data that linear regression cannot capture due to its simplicity. When linear regression is overfitting, number of columns(independent variables) approach number of observations there are two ways to mitigate it

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.

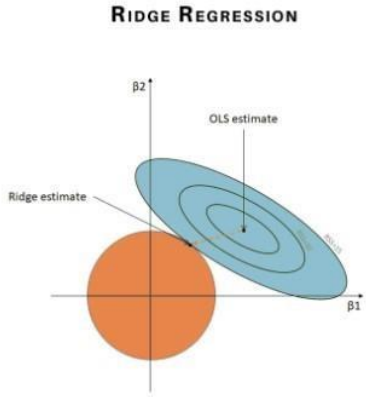


Residual Sum of Squares + λ * (Sum of the absolute value of the coefficients) The equation looks like:

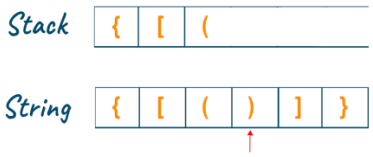
$$\sum_{i=1}^{n}(y_i - \sum_j x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

- **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: Simulate a sine curve between 60° and 300° with some random noise.

Dataset 2: food_truck_data.csv

Dataset 3: home_data.csv

Department of Computer Science and Engineering (Data Science)

1. Perform Linear Regression on Dataset 1 and Dataset 2 by computing cost function and gradient descent from scratch.

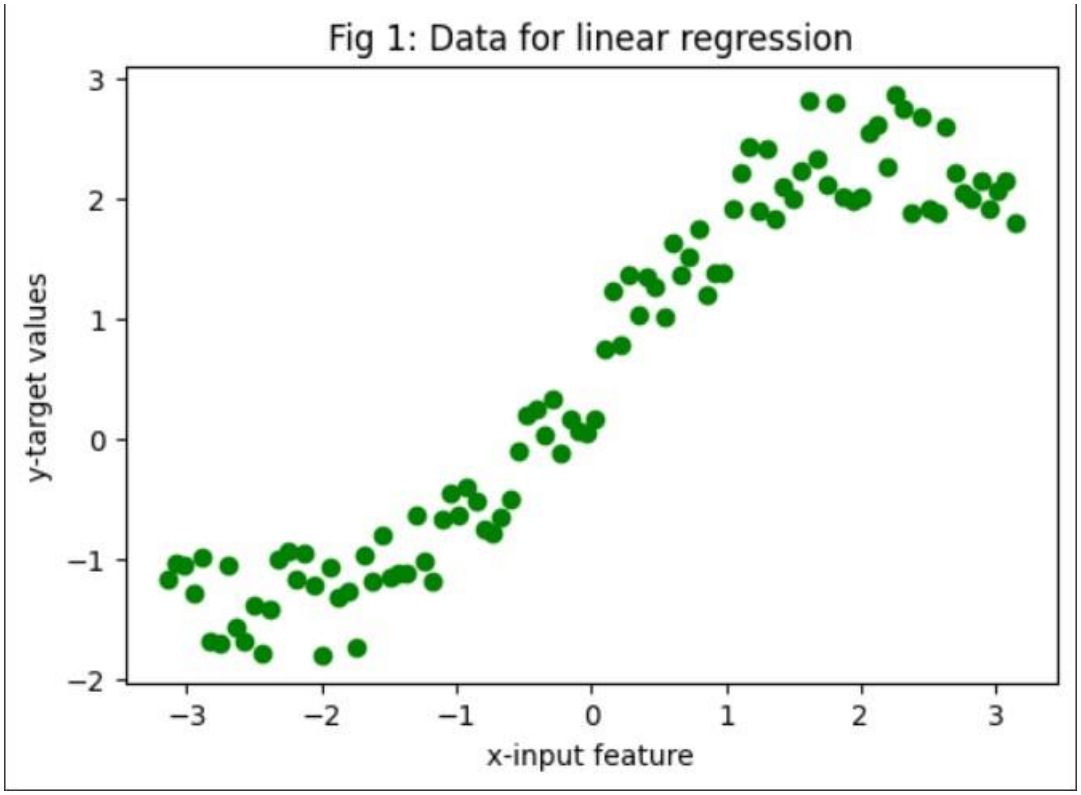
Linear Regression Experiment 1

Import Necessary Libararies

```
import pandas as pd
import seaborn as sns
import numpy as np
import math
import matplotlib.pyplot as plt
```

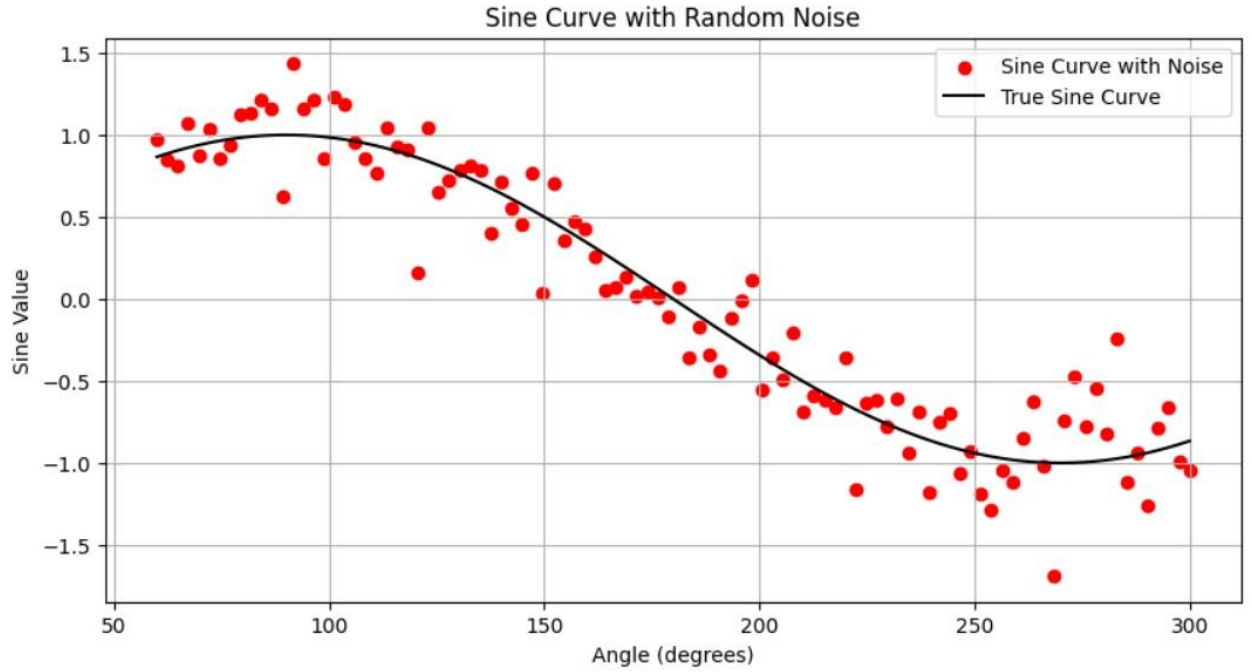
Sine Curve with Random Noise

```
number_of_samples = 100
x = np.linspace(-np.pi, np.pi, number_of_samples)
y = 0.5*x+np.sin(x)+np.random.random(x.shape)
plt.figure(figsize=(6,4))
plt.scatter(x,y,color='green')
plt.xlabel('x-input feature')
plt.ylabel('y-target values')
plt.title('Fig 1: Data for linear regression')
plt.show()
```



Sine Curve 60 to 300 Degree

```
angles = np.linspace(60, 300, 100)
sine_values = np.sin(np.radians(angles))
noise = np.random.normal(0, 0.2, len(angles))
sine_values_with_noise = sine_values + noise
plt.figure(figsize=(10, 5))
plt.scatter(angles, sine_values_with_noise, label='Sine Curve with Noise', color='red')
plt.plot(angles, sine_values, label='True Sine Curve', color='black')

plt.title('Sine Curve with Random Noise')
plt.xlabel('Angle (degrees)')
plt.ylabel('Sine Value')
plt.legend()
plt.grid(True)
plt.show()
```


Sine Curve with Random Noise
Sine Curve with Noise
True Sine Curve
Sine Value
Angle (degrees)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
50
100
150
200
250
300

Linear Regression on Food Truck Data Set

Reading Data

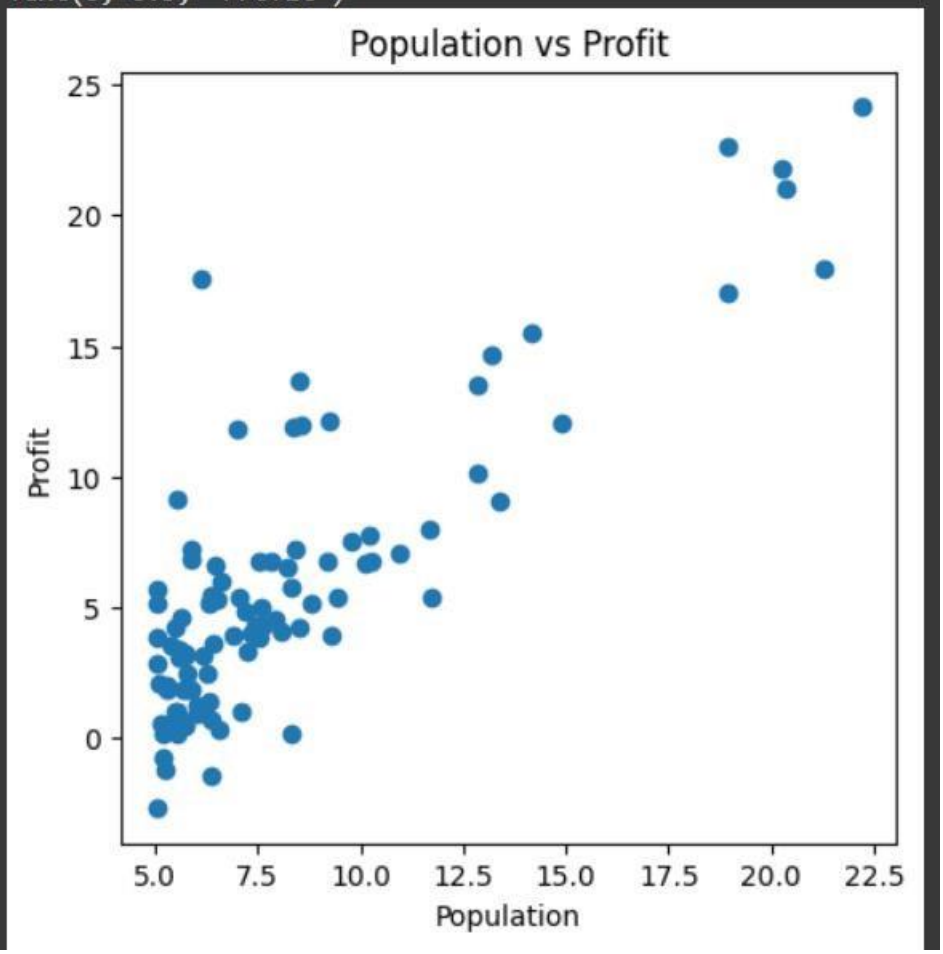
```
#First Five Row Values
df2=pd.read_csv('food_truck_data.txt')

df2.head(5)
```

	Population	Profit
0	6.1101	17.5920
1	5.5277	9.1302
2	8.5186	13.6620
3	7.0032	11.8540
4	5.8598	6.8233

Scatter Plot of Population vs Profit

```
plt.figure(figsize=(5,5))
plt.scatter(df2['Population'],df2['Profit'])
plt.title("Population vs Profit")
plt.xlabel('Population')
plt.ylabel('Profit')
```

```
Text(0, 0.5, 'Profit')
```



Cost Function

```
[15] def cost_function(x,y,theta):
         m=len(y)
         y_pred=x.dot(theta)
         error=(y_pred-y)**2
         return 1/(2*m)*np.sum(error)


     m=df2.Population.size
     X2=np.append(np.ones((m,1)),df2.Population.values.reshape(m,1),axis=1)
     Y2=df2.Profit.values.reshape(m,1)
     theta=np.zeros((2,1))
     cost_function(X2,Y2,theta)

     32.072733877455676
```

Gradient Descent

```
[16] def gradient_descent(x,y,theta,alpha,iterations):
         m=len(y)
         costs=[]
         for i in range(iterations):
             y_pred=x.dot(theta)
             error=np.dot(x.transpose(),(y_pred-y))
             theta=theta-alpha*1/m*error
             costs.append(cost_function(x,y,theta))
         return theta,costs
```

Regression Line

```
[7] theta,costs=gradient_descent(X2,Y2,theta,alpha=0.01,iterations=1)
    print("h(x)= {} + {}x1".format(str(round(theta[0,0],2)),str(round(theta[1,0],2))))
```

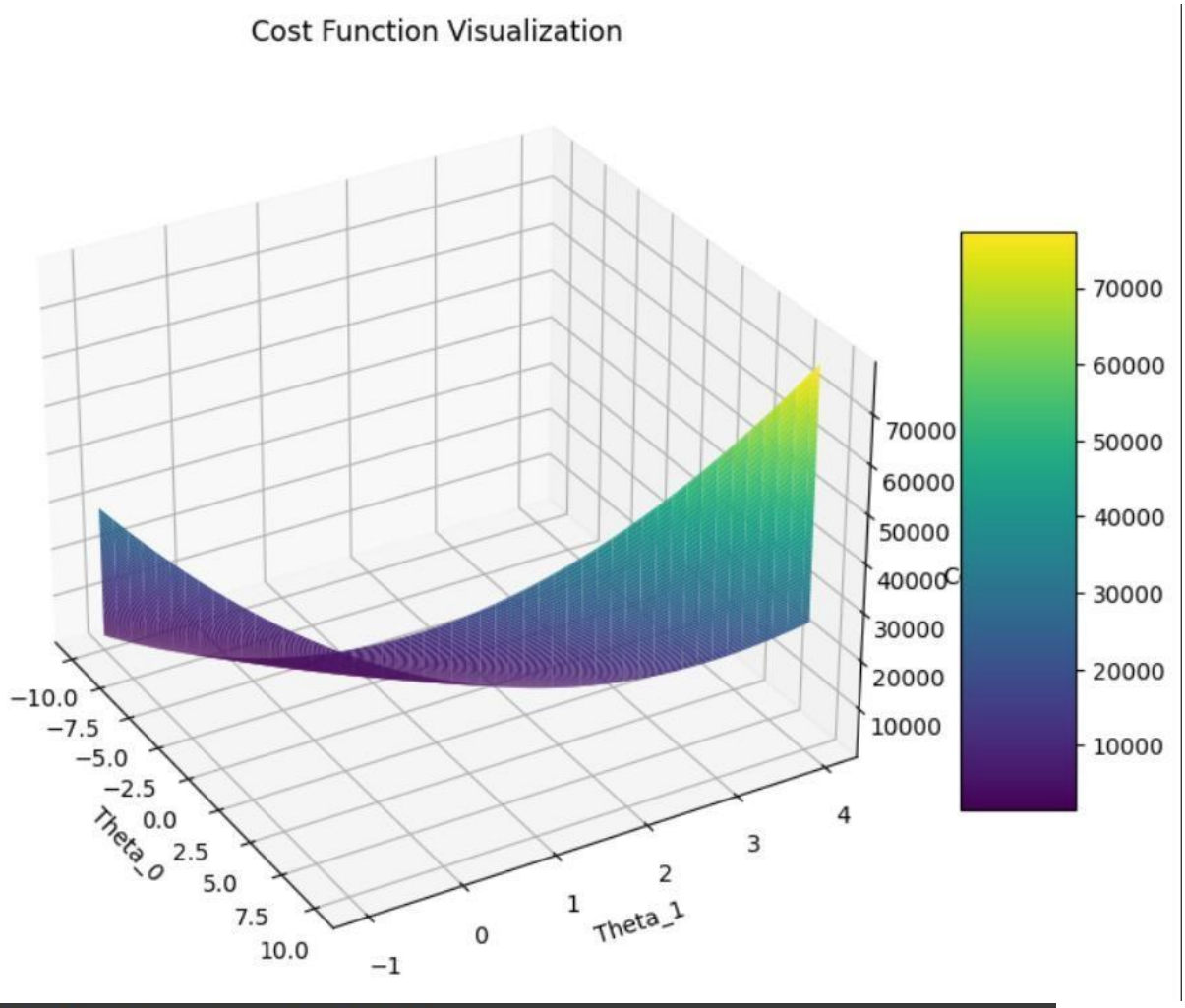
```
h(x)= 0.06 + 0.65x1
```

Prediction Function

```
[18] def predict(x,theta):
         y_pred=theta[0][0]+x*theta[1][0]
         return y_pred
```

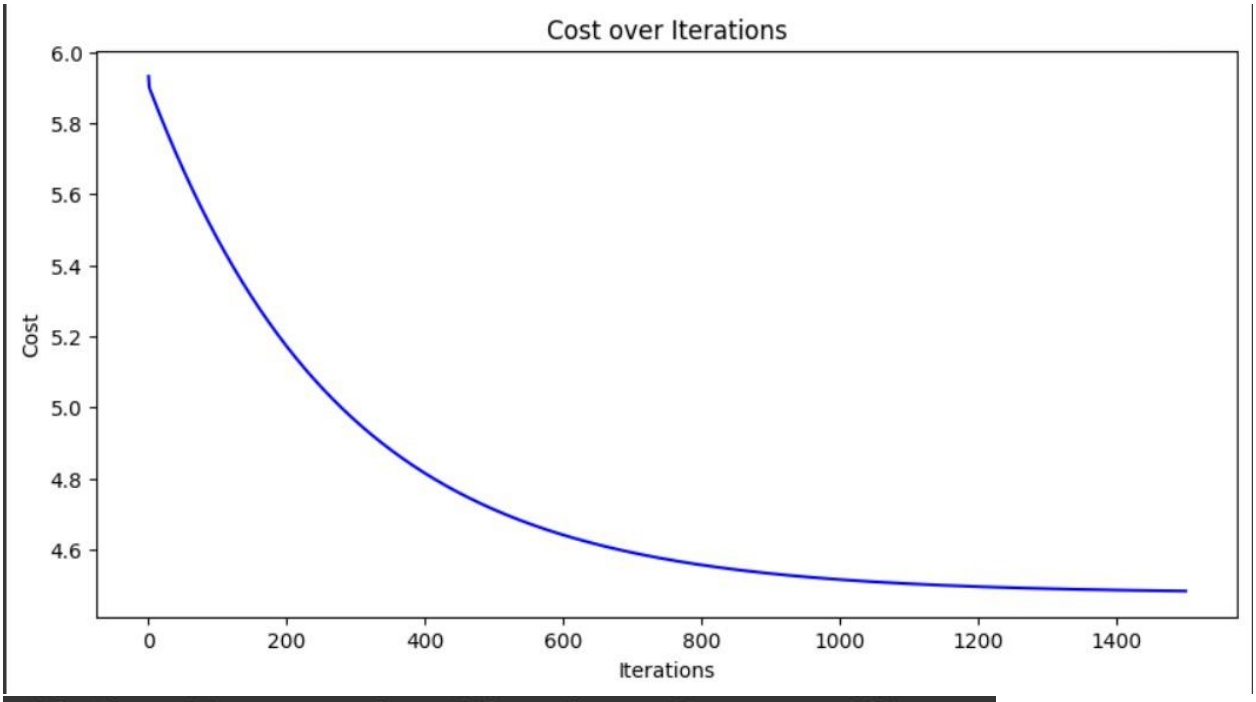
Visualization of Cost Function

```
from mpl_toolkits.mplot3d import Axes3D
theta_0 = np.linspace(-10, 10, 100)
theta_1 = np.linspace(-1, 4, 100)
cost_values = np.zeros((len(theta_0), len(theta_1)))

for i in range(len(theta_0)):
    for j in range(len(theta_1)):
        t = np.array([theta_0[i], theta_1[j]])
        cost_values[i, j] = cost_function(X2, Y2, t)

fig = plt.figure(figsize=(9, 9))
ax = fig.add_subplot(111,projection='3d')
surf = ax.plot_surface(theta_0, theta_1, cost_values, cmap='viridis')
fig.colorbar(surf, shrink=0.5, aspect=5)
plt.xlabel("Theta_0")
plt.ylabel("Theta_1")
ax.set_zlabel("Cost")
ax.view_init(30, 330)
plt.title("Cost Function Visualization")
plt.show()
```



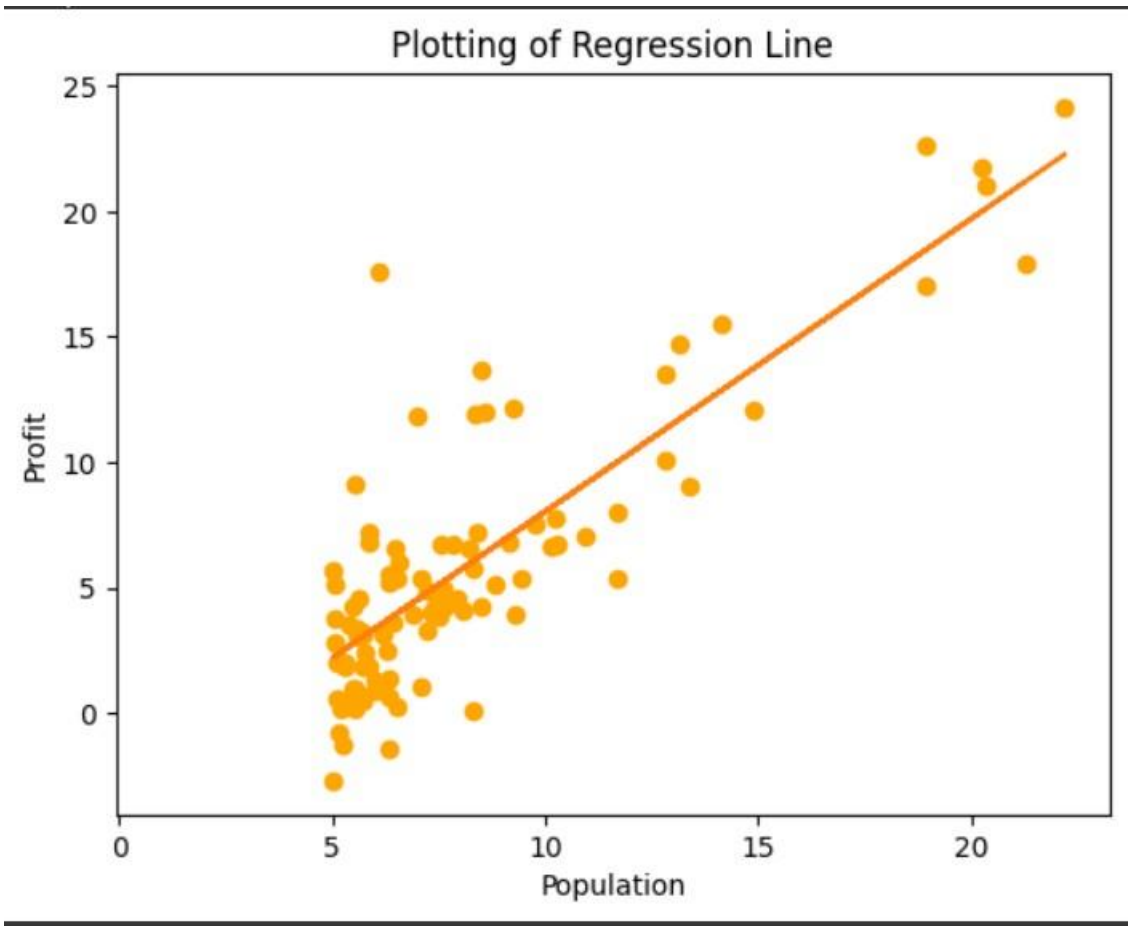
Plot of Cost Drop Over Iterations

```
alpha = 0.01
iterations = 1500
theta, costs = gradient_descent(X2, Y2, theta, alpha, iterations)
plt.figure(figsize=(10, 5))
plt.plot(range(1, iterations + 1), costs, color='blue')
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Cost over Iterations')
plt.show()
```



Fitting Regression Line into Scatter Plot

```
Y2P=[]
for i in X2:
    Y2P.append(predict(i,theta))
plt.plot(X2,Y2P)
plt.xlabel("Population")
plt.ylabel("Profit")
plt.title("Plotting of Regression Line")
plt.scatter(df2['Population'],df2['Profit'],color='orange')
```



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.

House Price Dataset

```
#First Five Rows
df1=pd.read_csv('home_data.csv')
df1.head(5)
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basemer
0	7129300520	20141013T000000	221900	3	1.00	1180	5650	1.0	0	0	...	7	1180	
1	6414100192	20141209T000000	538000	3	2.25	2570	7242	2.0	0	0	...	7	2170	4(
2	5631500400	20150225T000000	180000	2	1.00	770	10000	1.0	0	0	...	6	770	
3	2487200875	20141209T000000	604000	4	3.00	1960	5000	1.0	0	0	...	7	1050	9
4	1954400510	20150218T000000	510000	3	2.00	1680	8080	1.0	0	0	...	8	1680	

5 rows × 21 columns

Size Vs Price Prediction

```
8] x=np.array(df1['sqft_living'].values)
   y=np.array(df1['price'].values)

0] from sklearn.linear_model import LinearRegression
   from sklearn.model_selection import train_test_split

5] #Reshaping Dataset into 2-D Array
   x=x.reshape(-1,1)
```

Split Dataset individually into Train and Test

```
] x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_state=42)
```

ting Model

```
]
  model = LinearRegression()
  model.fit(x_train,y_train)
```

▾ LinearRegression
LinearRegression()

```
  print("Train Score",model.score(x_train,y_train))
  print("Test Score",model.score(x_test,y_test))
```

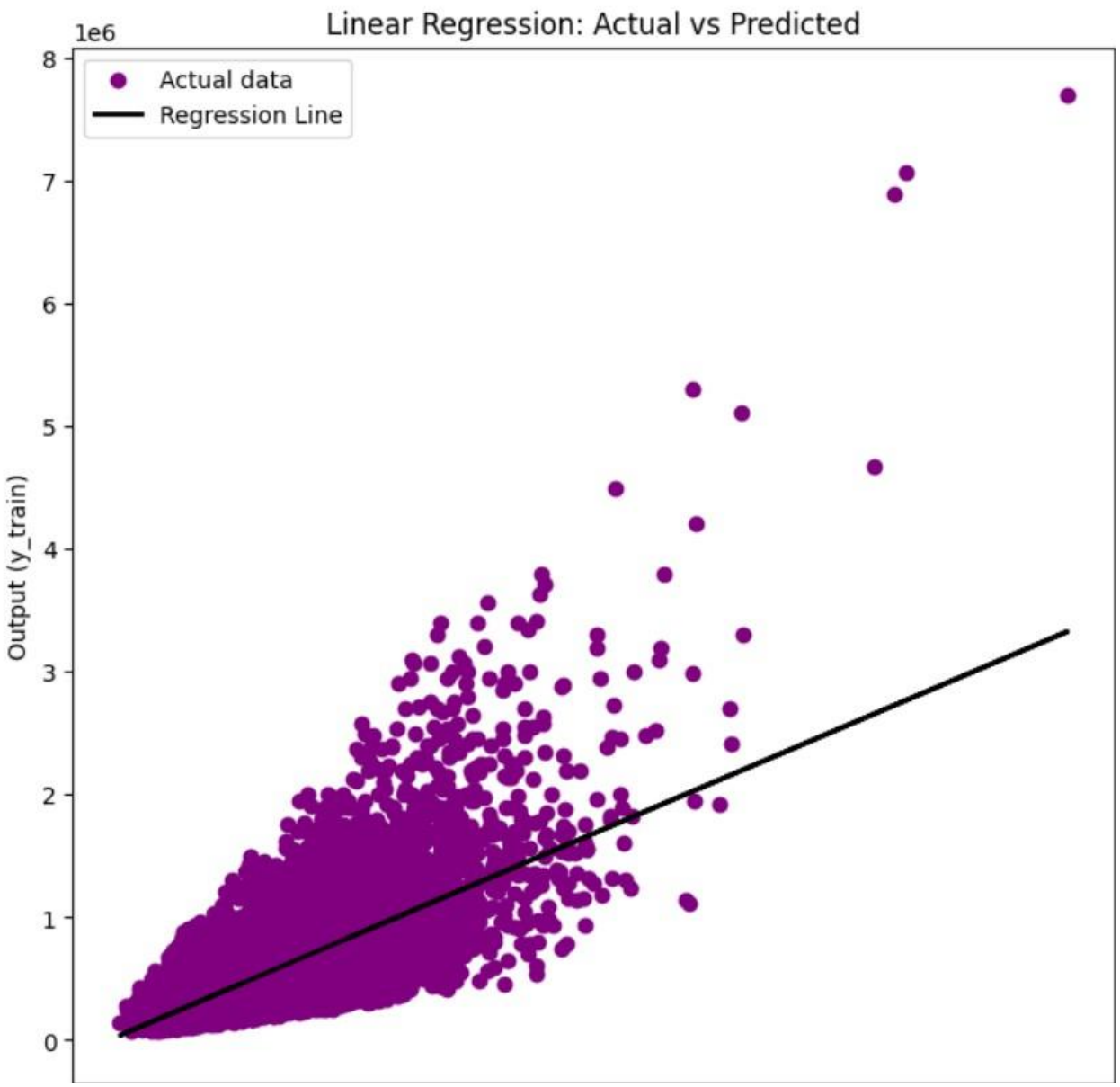
```
Train Score 0.4923838367820874
Test Score 0.4940690470123985
```

Plot of Model

Plot of Model

Loading...

```
y_pred = model.predict(x_train)
plt.figure(figsize=(8,8))
plt.scatter(x_train, y_train, color='purple', label='Actual data')
plt.plot(x_train, y_pred, color='black', linewidth=2, label='Regression Line')
plt.xlabel('Input Feature (x_train)')
plt.ylabel('Output (y_train)')
plt.title('Linear Regression: Actual vs Predicted')
plt.legend()
plt.show()
```



3. To perform regularization on linear model build using Linear Regression on Dataset3.

Ridge linear Regression

```
1] from sklearn.linear_model import Ridge
   ridgemodel=Ridge(alpha=50,max_iter=100,tol=0.1)
   ridgemodel.fit(x_train,y_train)
```

```
Ridge
Ridge(alpha=50, max_iter=100, tol=0.1)
```

ACCURACY

```
2] print("Train Score",ridgemodel.score(x_train,y_train))
   print("Test Score",ridgemodel.score(x_test,y_test))
```

```
Train Score 0.4923838367820874
Test Score 0.4940690469535862
```

Lasso Linear Regression

```
[43] from sklearn.linear_model import Lasso
     lassomodel=Lasso(alpha=50,max_iter=100,tol=0.1)
     lassomodel.fit(x_train,y_train)
```

```
Lasso
Lasso(alpha=50, max_iter=100, tol=0.1)
```

ACCURACY

```
print("Train Score",lassomodel.score(x_train,y_train))
print("Test Score",lassomodel.score(x_test,y_test))
```

```
Train Score 0.4923838367820642
Test Score 0.4940690433749376
```

MY COLLABORATORY LINK:

https://colab.research.google.com/drive/1uH0OpSuWUVQoq1St2-xm7p-mLEWeWBu-#scrollTo=lTff4XjU0JDv