Department of Computer Science and Engineering (Data Science)

Subject: Machine Learning – I (DJ19DSC402)

AY: 2022-23

JHANVI PAREKH

DATA SCIENCE

60009210033

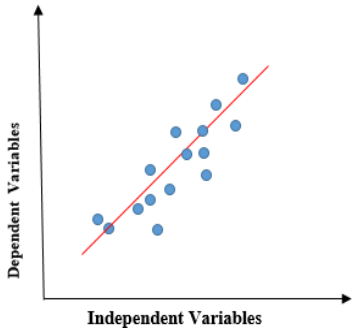
Experiment 1

(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.*

$$y = mx+b \implies y = a_0+a_1x$$

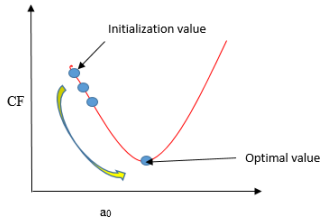
y= Dependent Variable; x= Independent Variable; a0= intercept; a1 = Linear regression coefficient.

- **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.

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

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

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

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

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

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

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

- **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

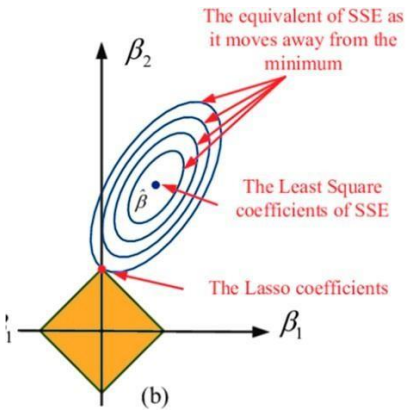
  1. Add more observations
  2. Regularization

  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.



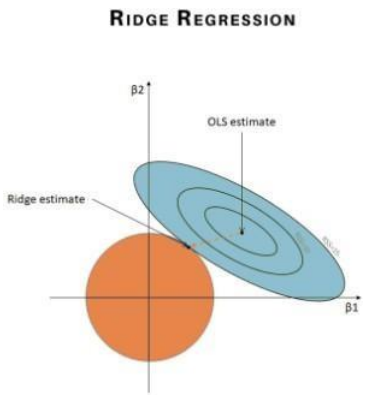
Residual Sum of Squares + λ * (Sum of the absolute value of the coefficients)

The equation looks like:

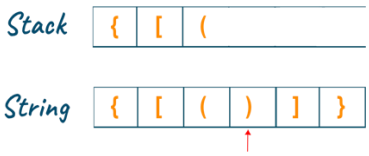
$$\sum_{i=1}^{n}(y_i - \sum_j x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p}|\beta_j|$$

- **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

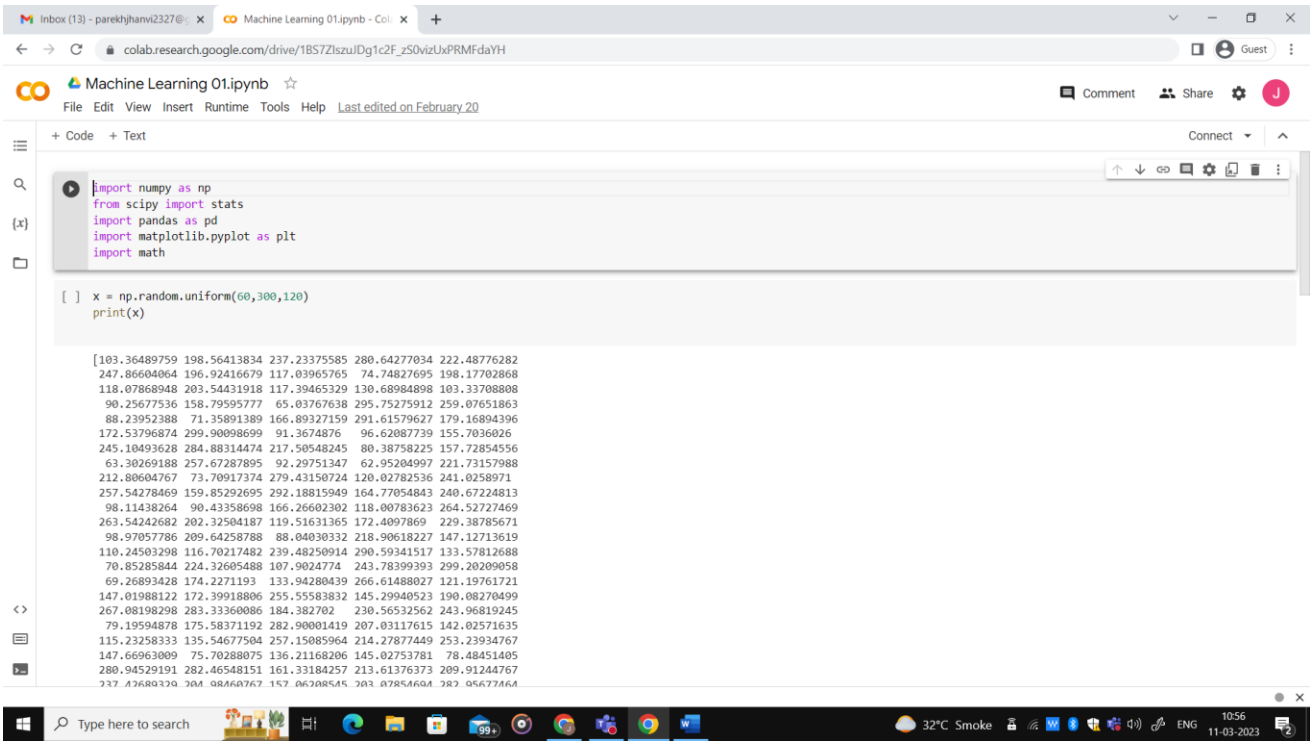
Use the given dataset and perform the following tasks:

Dataset 1: Simulate a sine curve between 60° and 300° with some random noise.

Dataset 2: food_truck_data.csv

Dataset 3: housing.csv

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



Machine Learning 01.ipynb

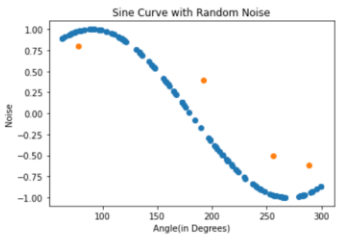
```
[ ] x = np.random.uniform(60,300,120)
    print(x)
```

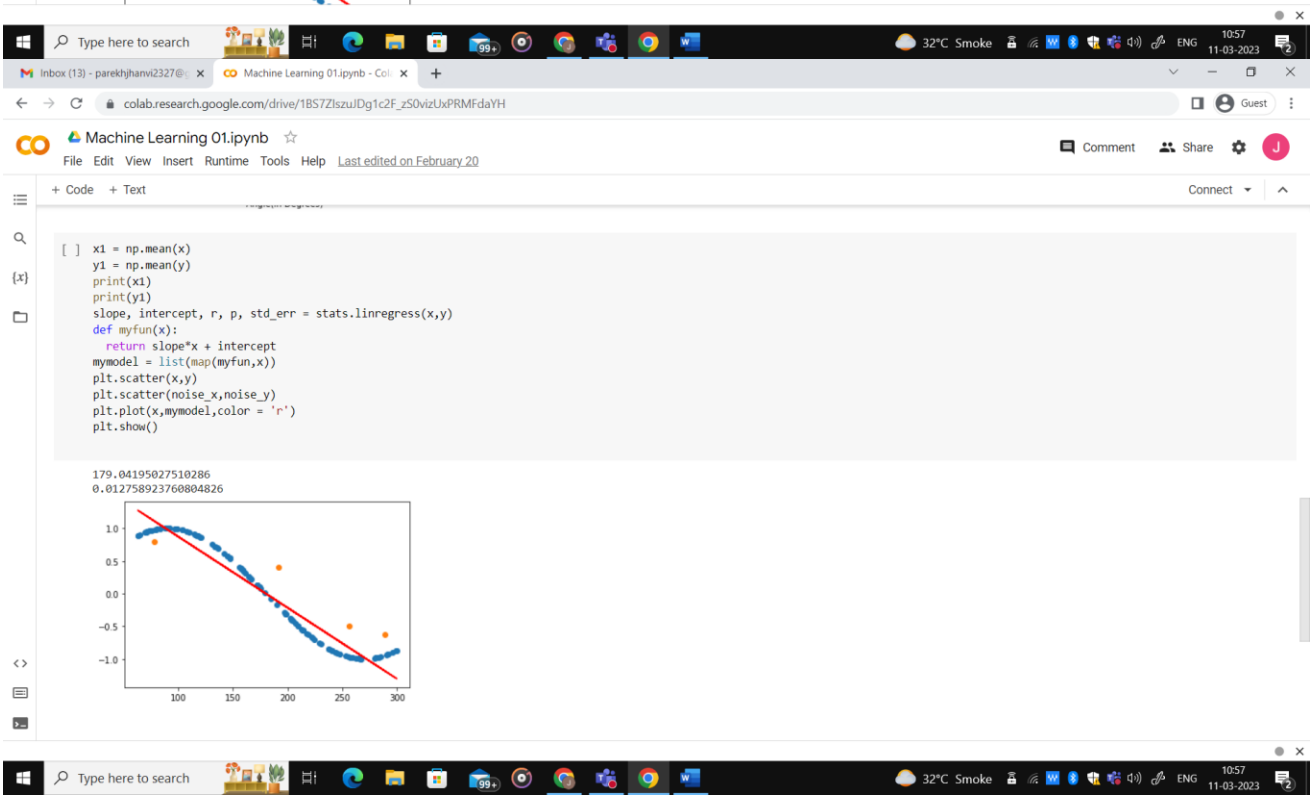
```
[103.36489759 198.56413834 237.23375585 280.64277034 222.48776282
 247.86604064 196.92416679 117.03965765  74.74827695 198.17702868
 118.07868948 203.54431918 117.39465329 130.68984898 103.33708808
  90.25677536 158.79595777  65.03767638 295.75275912 259.07651863
  88.23952388  71.35891389 166.89327159 291.61579627 179.16894396
 172.53796874 299.90098699  91.3674876   96.62087739 155.7036026
 245.10493628 284.88314474 217.50548245  80.38758225 157.72854556
  63.30269188 257.67287895  92.29751347  62.95204997 221.73157988
 212.80604767  73.70917374 279.43150724 120.02782536 241.0258971
 257.54278469 159.85292695 292.18815949 164.77054843 240.67224813
  98.11438264  90.43358698 166.26602302 118.00783623 264.52727469
 263.54242682 202.32504187 119.51631365 172.4097869  229.38785671
  98.97057786 209.64258788  88.04030332 218.90618227 147.12713619
 110.24503298 116.70217482 239.48250914 290.59341517 133.57812688
  70.85285844 224.32605488 107.9024774  243.78399393 299.20209058
  69.26893428 174.2271193  133.94280439 266.61488027 121.19761721
 147.01988122 172.39918806 255.55583832 145.29940523 190.08270499
 267.08198298 283.33360086 184.382702   230.56532562 243.96819245
  79.19594878 175.58371192 282.90001419 207.03117615 142.02571635
 115.23258333 135.54677504 257.15085964 214.27877449 253.23934767
 147.66963009  75.70288075 136.21168206 145.02753781  78.48451405
 280.94529191 282.46548151 161.33184257 213.61376373 209.91244767
 237.42689329 204.98460767 157.06208545 203.07854694 282.95677464
 262.38275871 164.76286398  83.34280514 114.26527909 214.3815309 ]
```

```
[ ] def mysin(x):
      return(math.sin(math.radians(x)))
    y = list(map(mysin,x))
    print(y)
```

```
[0.9729176765886686, -0.3183660348614734, -0.8408856054139683, -0.9827977587016083, -0.6754327251517862, -0.926305479009232, -0.29110574224158337, 0.890692078288907, 0.9647794133028663
```

```
[ ] noise_x = [78,192,256,289]
    noise_y = [0.8,0.4,-0.5,-0.62]
    plt.scatter(x,y)
    plt.scatter(noise_x,noise_y)
    plt.title("Sine Curve with Random Noise")
    plt.xlabel("Angle(in Degrees)")
    plt.ylabel("Noise")
    plt.show()
```

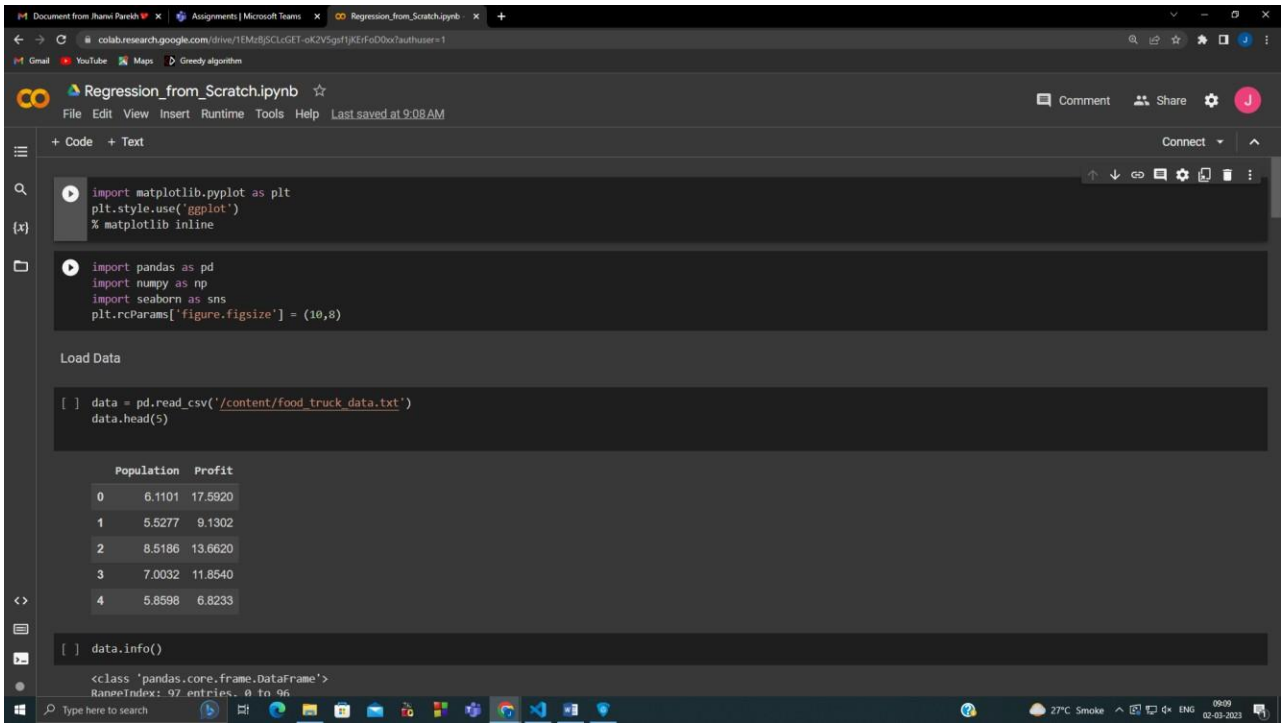




https://colab.research.google.com/drive/1BS7ZIszuJDg1c2F_zS0vizUxPRMFdaYH?usp=sharing

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.



Regression_from_Scratch.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:08 AM

Comment Share Connect

+ Code + Text

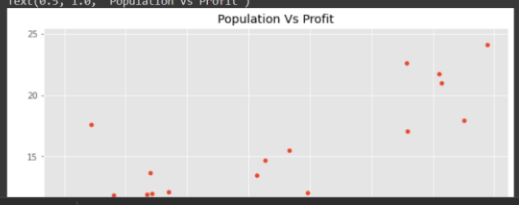
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 97 entries, 0 to 96
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Population  97 non-null     float64
 1   Profit      97 non-null     float64
dtypes: float64(2)
memory usage: 1.6 KB
```

visualise data

```
ax = sns.scatterplot(x = "Population", y = "Profit", data = data)
ax.set_title("Population Vs Profit")
```

```
Text(0.5, 1.0, 'Population Vs Profit')
```



Regression_from_Scratch.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:08 AM

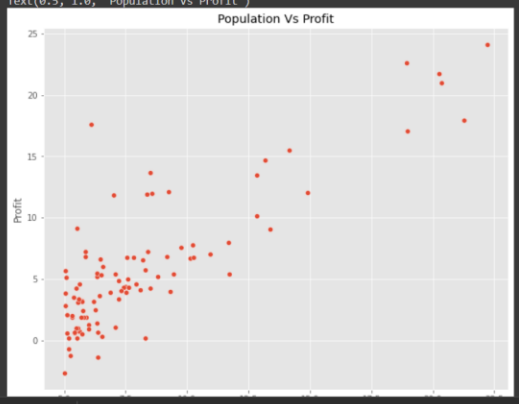
Comment Share Connect

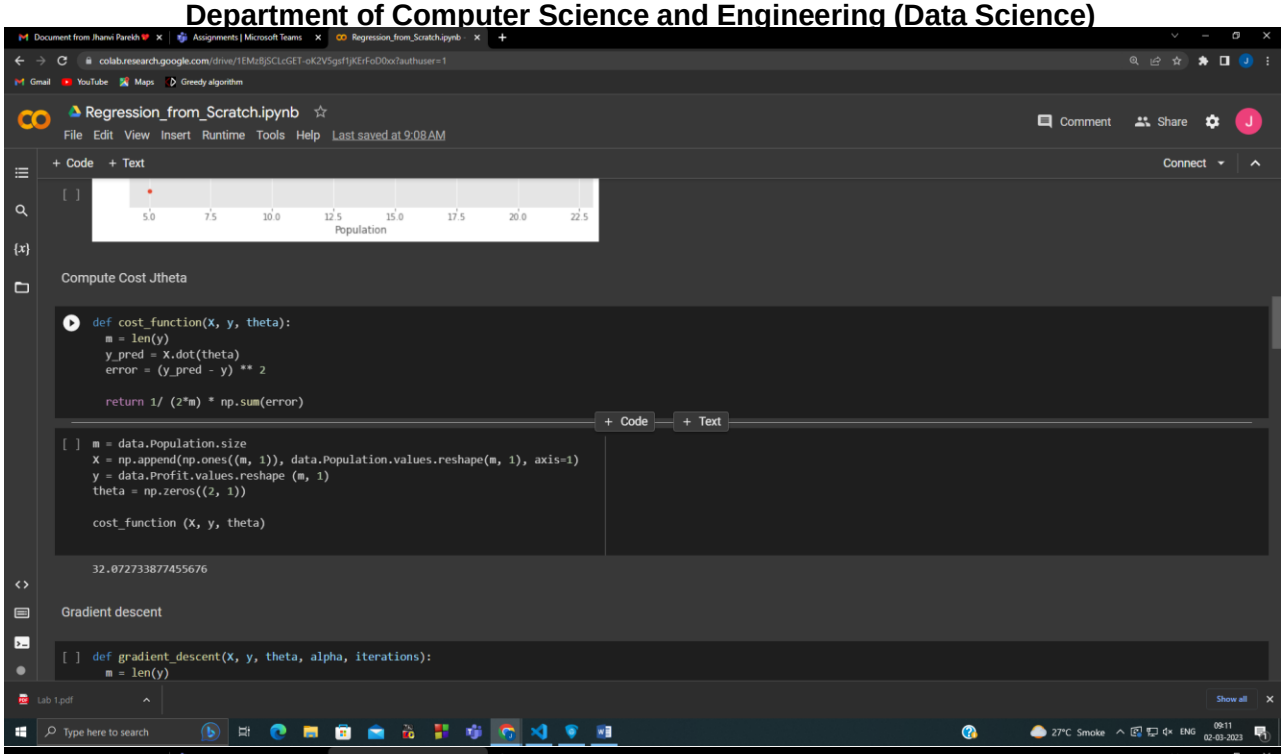
+ Code + Text

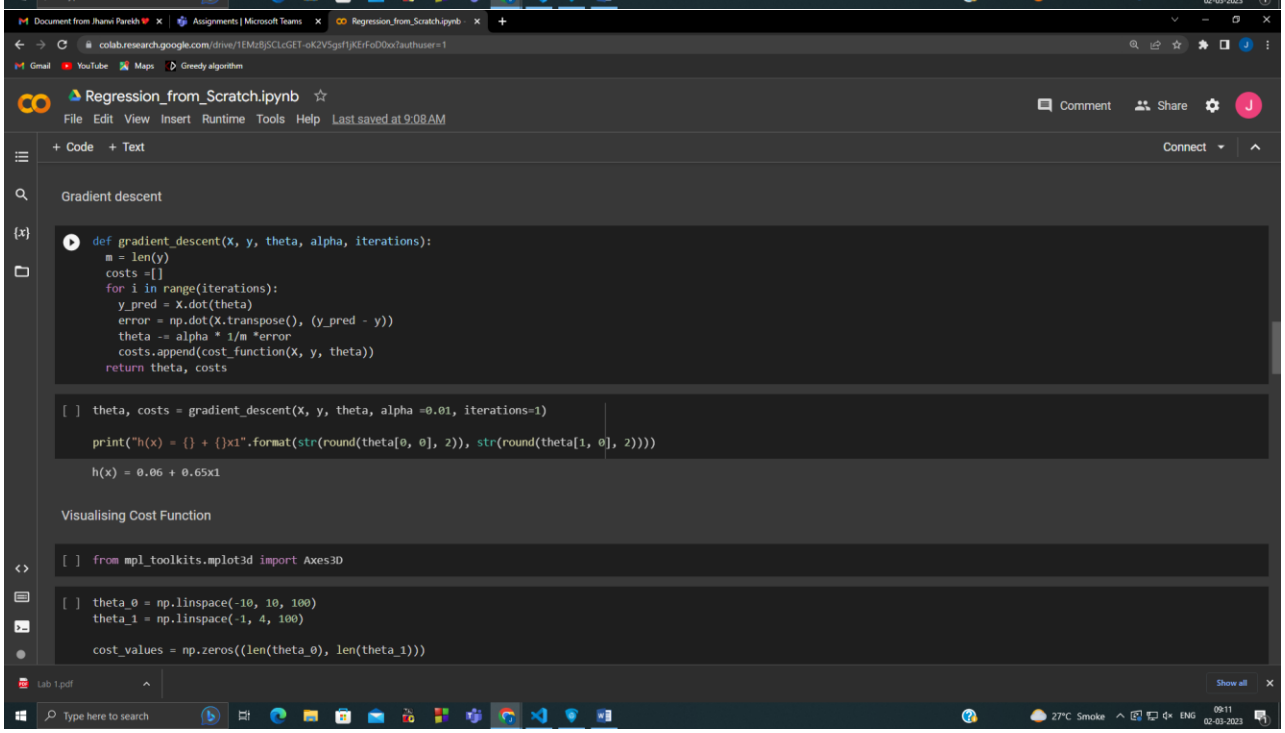
visualise data

```
ax = sns.scatterplot(x = "Population", y = "Profit", data = data)
ax.set_title("Population Vs Profit")
```

```
Text(0.5, 1.0, 'Population Vs Profit')
```




Regression_from_Scratch.ipynb
File Edit View Insert Runtime Tools Help Last saved at 9:08 AM
Comment
Share
Connect
+ Code + Text
5.0 7.5 10.0 12.5 15.0 17.5 20.0 22.5
Population
Compute Cost Jtheta
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)
+ Code + Text
m = data.Population.size
X = np.append(np.ones((m, 1)), data.Population.values.reshape(m, 1), axis=1)
y = data.Profit.values.reshape (m, 1)
theta = np.zeros((2, 1))
cost_function (X, y, theta)
32.072733877455676
Gradient descent
def gradient_descent(X, y, theta, alpha, iterations):
m = len(y)


Regression_from_Scratch.ipynb
File Edit View Insert Runtime Tools Help Last saved at 9:08 AM
Comment
Share
Connect
+ Code + Text
Gradient descent
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 -= alpha * 1/m *error
costs.append(cost_function(X, y, theta))
return theta, costs
theta, costs = gradient_descent(X, y, 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
Visualising 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)))

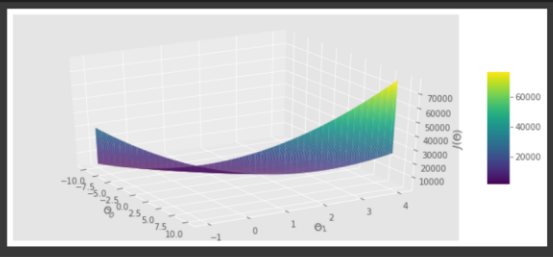
Regression_from_Scratch.ipynb

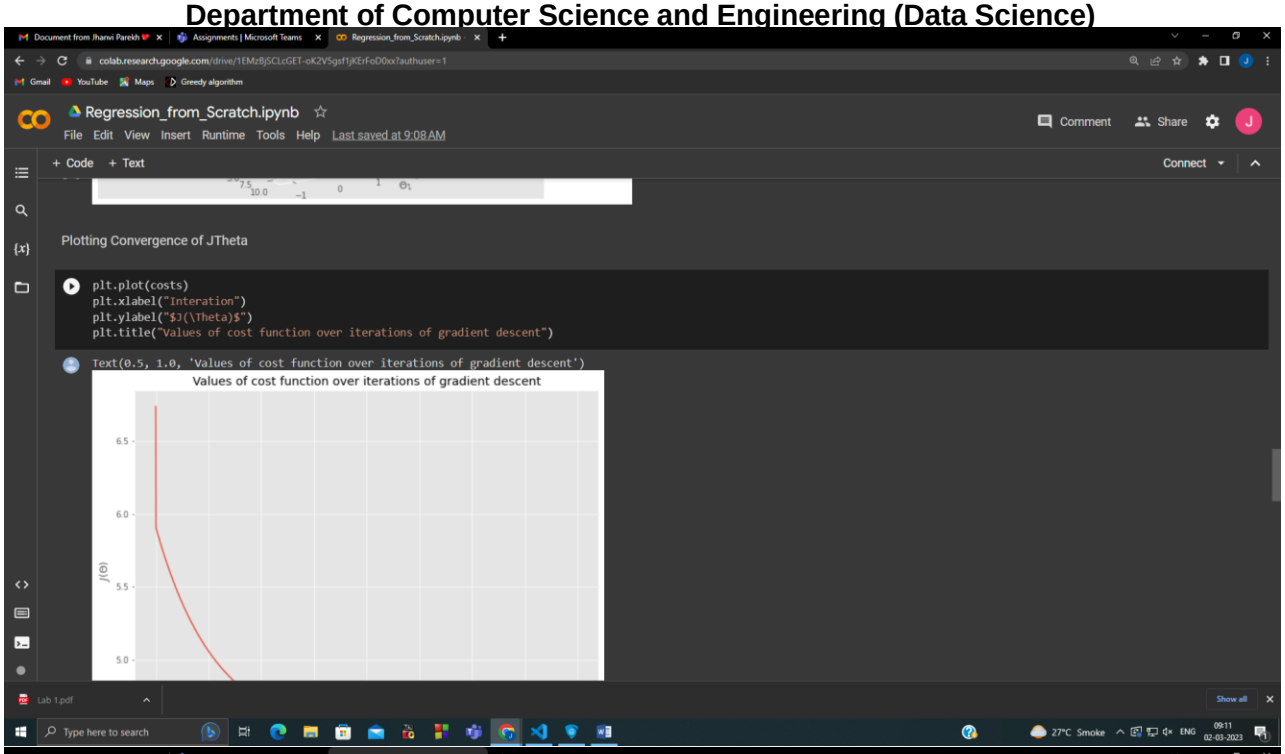
Visualising Cost Function

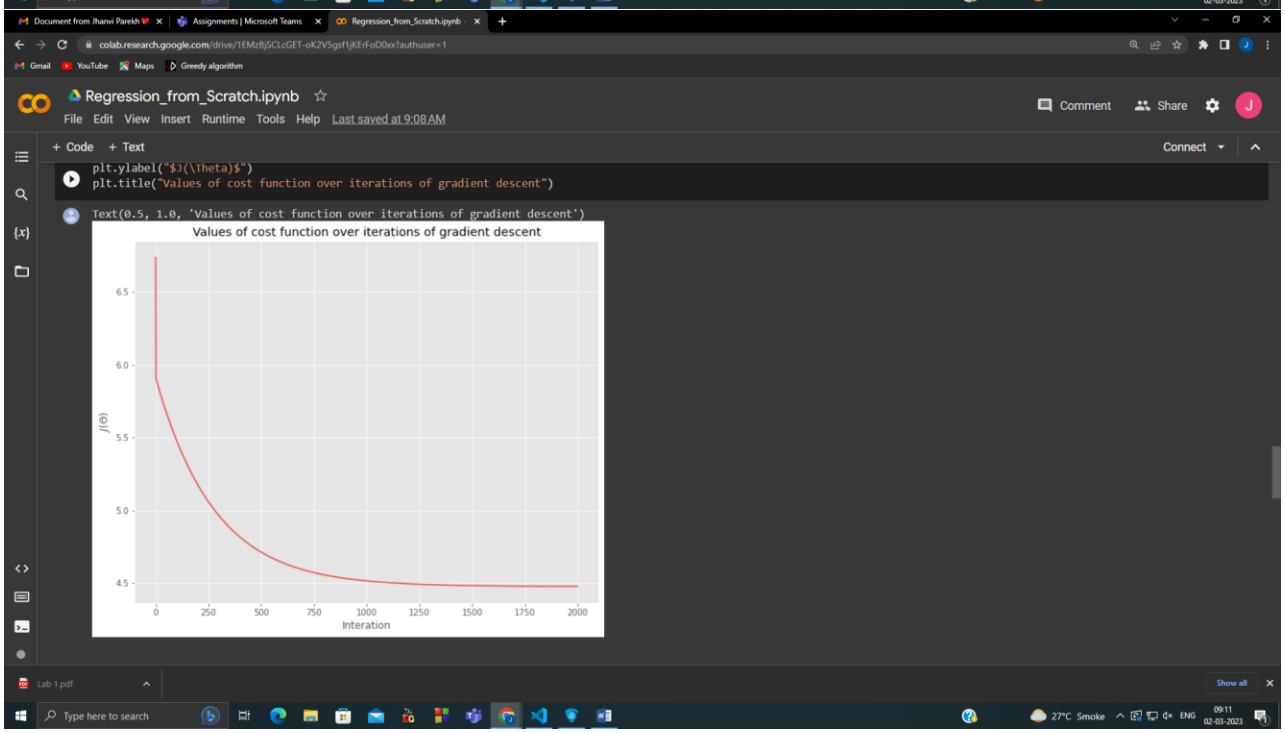
```
from mpl_toolkits.mplot3d import Axes3D
```

```
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(X,y,t)
```

```
fig = plt.figure(figsize = (12, 5))
ax = fig.gca(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("$J(\Theta)$")

ax.view_init(30, 330)

plt.show()
```




Regression_from_Scratch.ipynb
File Edit View Insert Runtime Tools Help Last saved at 9:08 AM
Comment
Share
Connect
+ Code + Text
Plotting Convergence of JTheta
plt.plot(costs)
plt.xlabel("Interation")
plt.ylabel("$J(\Theta)$")
plt.title("Values of cost function over iterations of gradient descent")
Text(0.5, 1.0, 'Values of cost function over iterations of gradient descent')
Values of cost function over iterations of gradient descent
J(Θ)
6.5
6.0
5.5
5.0

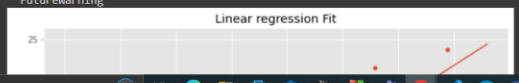

Regression_from_Scratch.ipynb
File Edit View Insert Runtime Tools Help Last saved at 9:08 AM
Comment
Share
Connect
+ Code + Text
plt.ylabel("$J(\Theta)$")
plt.title("Values of cost function over iterations of gradient descent")
Text(0.5, 1.0, 'Values of cost function over iterations of gradient descent')
Values of cost function over iterations of gradient descent
J(Θ)
6.5
6.0
5.5
5.0
4.5
0
250
500
750
1000
1250
1500
1750
2000
Interation

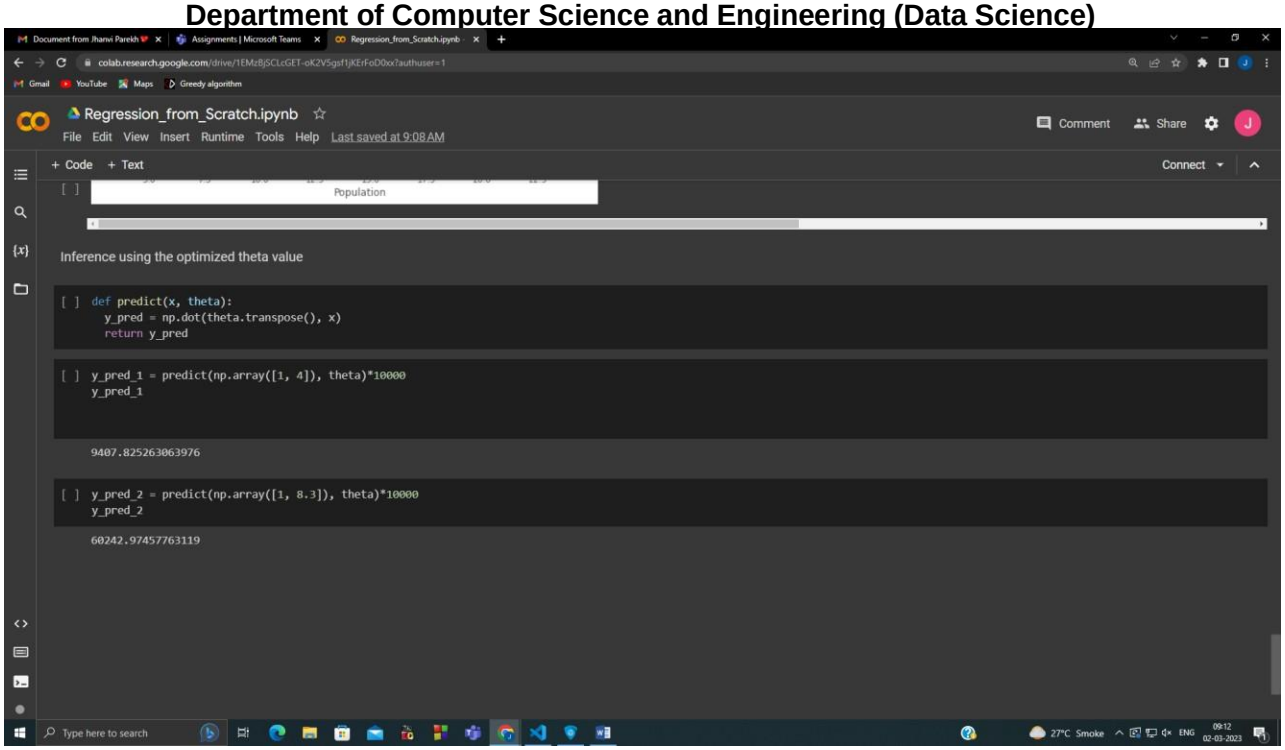
Regression_from_Scratch.ipynb

Training Data with Linear regression fit

```
[ ] theta.shape

(2, 1)

[ ] theta

array([[-3.78806857],
       [ 1.18221277]])

theta = np.squeeze(theta)

sns.scatterplot(x = "Population", y = "Profit", data = data)

x_value = [x for x in range(5, 25)]
y_value = [(x * theta[1] + theta[0]) for x in x_value]

sns.lineplot(x_value, y_value)

plt.xlabel("Population")
plt.ylabel("Profit")
plt.title("Linear regression Fit");
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argume
  FutureWarning
```





https://colab.research.google.com/drive/1EMzBjSCLcGET-oK2V5gsf1jKErFoD0xx?usp=sharing