# CHANDAN DHAMANDE

20190802117

https://github.com/nitrogen404/Semester-5/blob/master/TC1/Labs/Lab1/gradientDescent.py

## Track Elevative
## Lab - 1

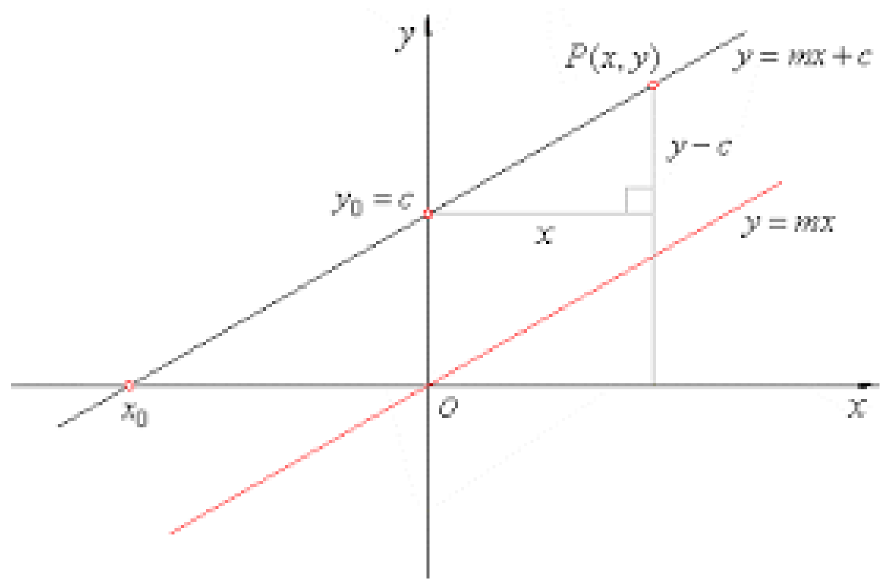
**Aim:** To find the parameters or coefficients of a function where the function has a minimum value using gradient Descent Optimizations algorithms

**Tools:** Python, Matplotlib

**Theory:** A linear regression model attempts to explain the relationship between a dependent variable and one or more independent variables using a straight line. The line can be represented using the equation of

$$y = mx + c$$

'y' being the dependent variable and 'x' as independent. 'm' is the slope of the line and 'c' as the y intercept.




Our goal is to minimize this error to obtain the most accurate value of m and c.
Step1. Find the difference between the actual y and predicted y value(y = mx + c), for a given x.
Step2. Square this difference.
Step3. Find the mean of the squares for every value in X.

# Gradient Descent Algorithm

1. Initially let m = 0 and c = 0. Let L be our learning rate, let's set it to 0.0001 for good accuracy.
2. Calculate the partial derivative of the loss function with respect to m, and plug in the current values of x, y, m and c in it to obtain the derivative value D.
3. Now we update the current value of m and c using the following equation.
4. We repeat this process until our loss function is a very small.

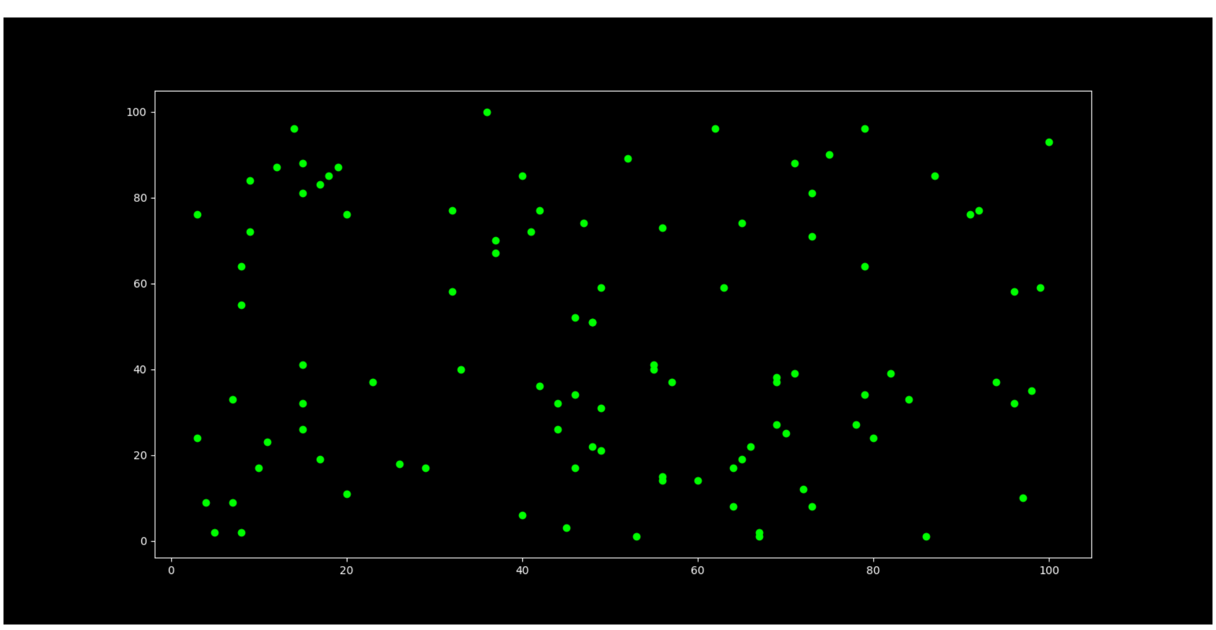
value or ideally 0. The value of m and c that we are left with now will be the optimum values.

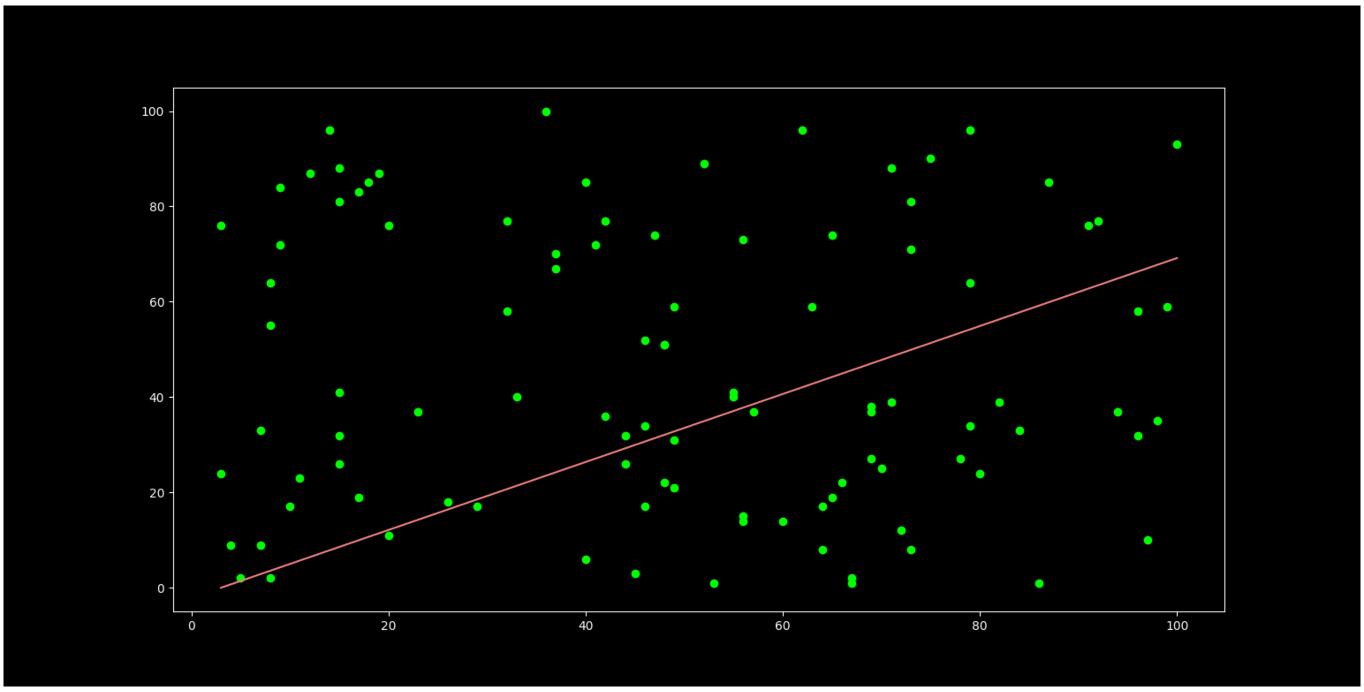
# Code:

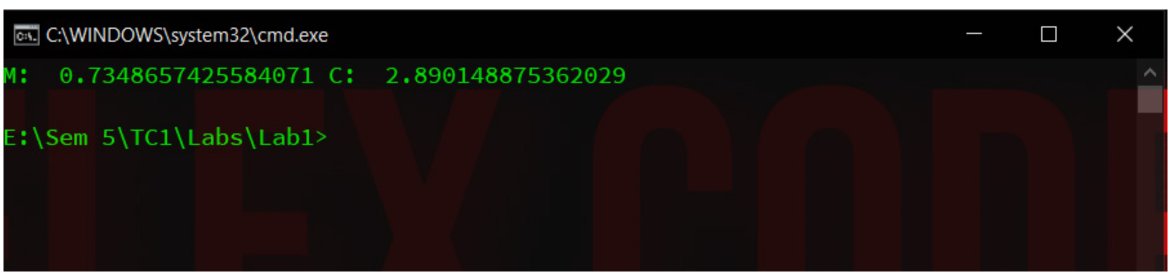
https://github.com/nitrogen404/Semester-5/blob/master/TC1/Labs/Lab1/gradientDescent.py

```
import matplotlib.pyplot as plt
import random
plt.style.use('dark_background')
x = [random.randint(1, 100) for i in range(100)]
y = [random.randint(1, 100) for i in range(100)]
plt.scatter(x, y, color='lime')
plt.show()

m = 0
c = 0
learning_rate = 0.0001
epochs = 1000
y_pred_values = []

for epoch in range(epochs):
    for value in range(len(x)):
        y_pred = m*x[value] + c
        y_pred_values.append(y_pred)
        partial_m = (-2 / float(len(x))) * (x[value] * (y[value]
- y_pred))
        partial_c = (-2 / float(len(x))) * (y[value] - y_pred)
        m = m - learning_rate * partial_m
        c = c - learning_rate * partial_c

print("M: ", m, "C: ", c)
plt.scatter(x, y, color='lime')
plt.plot([min(x), max(x)], [min(y_pred_values), max(y_pred_values)], color='lightcoral')  # regression line
plt.show()
```

# Output:






# Conclusion:

Studied linear regression using gradient descent approach. Optimized the equation of a line ***y = mx + c*** using gradient descent algorithm.