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: