Assignment 01

For this task, you will be using the baseline code for regression analysis provided in the class jupyter notebook. **Do not** use any libraries.

Task 0: Getting Ready with your Dataset

The dataset we will be using is diamonds.csv, which is a public dataset for prices (US $) along with attributes of 54,000 diamond prices.[1] The columns of the dataset are:

1. Carats (Diamond Weight 0.2 – 5.01)
2. Cut type (1: Fair, 2: Good, 3: Very Good, 4: Premium, 5: Ideal)
3. Color; (1: Worst, 2, 3, 4, 5, 6, 7: Best)
4. Clarity (1: Worst, 2, 3, 4, 5, 6, 7, 8: Best)
5. Depth as Percentage, calculated as: z / mean (x, y) or 2z / (x+y)
6. Table; width of diamond top relative to widest point (43 - 95)
7. Price in US $; (326 – 18,823)
8. x length (0 – 10.74 mm)
9. y length (0 – 58.9 mm)
10. z length (0 – 31.8 mm)

Plotting code is:

```
points = np.genfromtxt('data/diamonds.csv', delimiter=',')
points = points[1:len(points)]

carat    = points[:,0]
cut      = points[:,1]
color    = points[:,2]
clarity  = points[:,3]
depth    = points[:,4]
table    = points[:,5]
price    = points[:,6]
xlen     = points[:,7]
ylen     = points[:,8]
zlen     = points[:,9]

x = carat
y = price

plt.figure(figsize=(5,2))
plt.scatter(x,y,s=0.5)
plt.xlabel('Carats')
plt.ylabel('price')

plt.show()
```

1 Available Online: https://www.kaggle.com/datasets/shivam2503/diamonds, Retrieved: 19 March 2025

Keeping the price (column 7) on y-axis, play around with the data and answer the following questions.

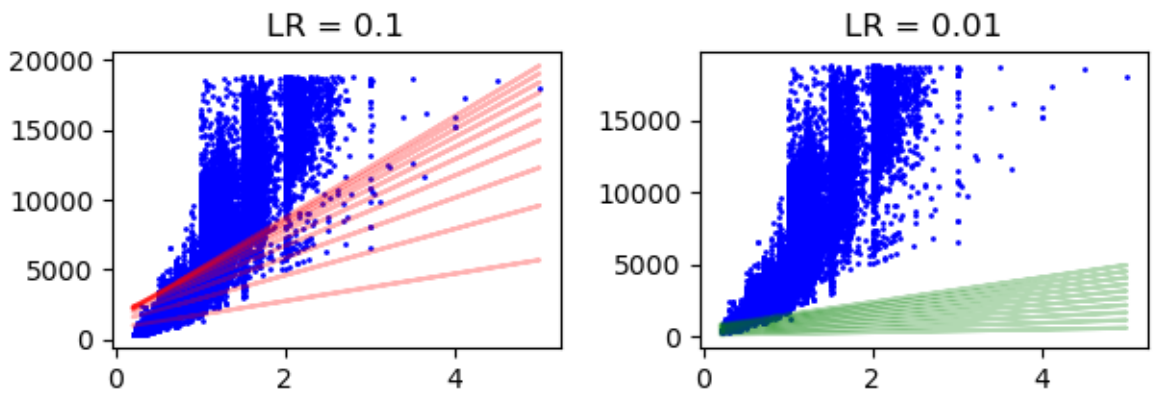
	Question	Answer
1	Identify which columns are most suitable to provide a linear/multilinear/polynomial relationship to price?	
2	Explain why some plots are appearing as vertical or horizontal lines?	

Task 1: Relationship between Learning Rate and Iterations

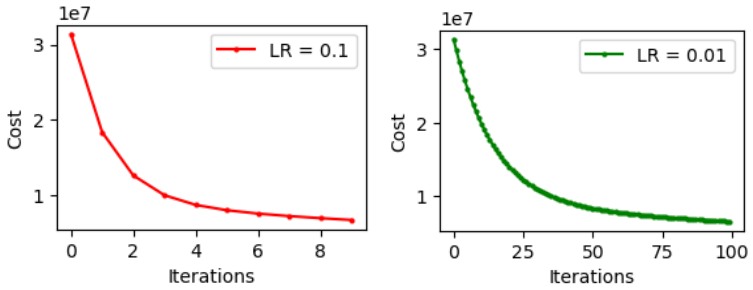
The code below will help illustrate the relationship between learning rate and number of iterations (See how done through for loop). The case is that of scatter plot of diamond carats and price.

```
for learning_rate, color, num_iterations
          in zip([0.1, 0.01], ['r', 'g'], [10, 10]):

    b = 0
    m = 0

    plt.figure(figsize=(3,2))
    plt.scatter(x, y, s=1, c='b')
    plt.title(f'LR = {learning_rate}')

    for i in range(num_iterations):

        i = i+1
        m_gradient = 0
        b_gradient = 0
        N = float(len(points))

        for j in range(0, len(points)):

            m_gradient += (2/N) * x[j] * (y[j] - (m * x[j] + b))
            b_gradient += (2/N) *        (y[j] - (m * x[j] + b))


        m = m + learning_rate * m_gradient
        b = b + learning_rate * b_gradient

        pred = m * x + b
        plt.plot(x, pred, c=color, alpha=0.1)

    plt.show()
```

In its current form, the linear regression line for 0.1 and 0.01 are shown below:



Attempt the following questions:

	Question	Answer
1	How many iterations would be needed for Learning Rate 0.01 to make its linear regression line similar to Learning Rate 0.1. Devise a scheme and show your output.	
2	Refer to your class jupyter notebook and get the formula for cost/error. a) Modify your code so that the cost function for Learning Rate 0.1 and 0.01 is determined. Show your code change. The output at my end is:  b) What is the minimum error in both cases?	
3	Modify your code so that Learning Rte of 0.001 is used, and it gives **exactly** the same output as Learning Rate 0.1. Comment on the nature of execution. What is happening? Would you be willing to try learning rate of 0.0001?	

4	What is the behavior of the linear regression line? Is it smoothly progressing in one direction or is it oscillating? Tip: You are plotting the movement of the line using:
	`plt.plot(x, pred, c=color, alpha=0.1)`
	When the loop finishes, place another plot with a different color to see the final plot. E.g.
	`plt.plot(x, pred, c='b', alpha=0.1)`	

Task 2: Converting to Stochastic Gradient Descent

The basic premise of gradient descent can be captured from the following:

```
for j in range(0, len(points)):
    m_gradient += (2/len(points)) * x[j] * (y[j] - (m * x[j] + b))

m = m + learning_rate * m_gradient
```

Where m_gradient is the slope representing contribution of error term in determining m. As can be seen, it is traversing all the points in the dataset.

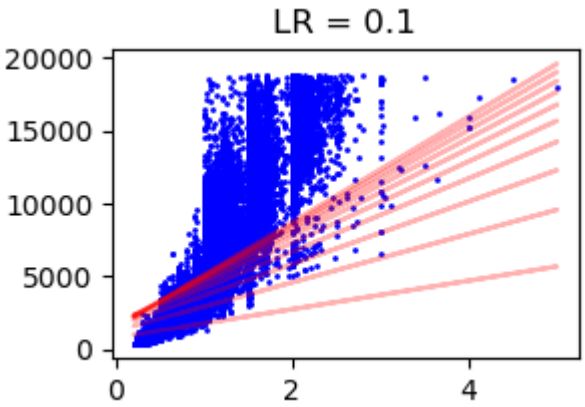
The stochastic gradient descent simply says don't go through all the points, but only a few points randomly. To do this, the following steps are carried out:

1. Determine a random number idx between 0 and len(points), i.e. `idx = np.random.randint(0, len(points))`
2. Pick up the x[idx] and y[idx] points corresponding to this random point, i.e. `x_j, y_j = x[idx], y[idx]`
3. Then use these points into your gradient calculation, which will become:

```
idx = np.random.randint(0, len(points))
x_j, y_j = x[idx], y[idx]

m_gradient += 2 * x_j * (y_j - (m * x_j + b))
```

```
m = m + learning_rate * m_gradient
```

Thus essentially the for loop is removed. Note that it is no longer needed to divide 2 by N, as only 1 value from the loop is being considered. Using the above as an example, modify the code from task 1 to convert from gradient descent to stochastic gradient descent. Then, answer the following questions:

	Question	Answer
1	What is the behavior of the linear regression line? Is it smoothly progressing in one direction or is it oscillating?	
2	Determine the stochastic gradient scolution for learning rates 0.1, 0.01, 0.001, 0.0001, 0.00001, and show the output graphs, just like we computed the following output graph of previous task: 	
3	Compute the total cost using: `total_cost = np.mean((y_j - (m * x_j + b)) ** 2)` `cost_graph.append(total_cost)` And then plot the cost_graph for all learning rates, i.e. 0.1, 0.01, 0.001, 0.0001, 0.00001.	
4	From 3 above, explain why these costs are not smoothly progressing and why are they oscillating?	

Task 3: Fitting to Polynomial Regression

From the class jupyter notebook, fetch the code for the polynomial regression (and convert it into stochastic gradient descent). The basic syntax of your code should be similar to the following (case of polynomial regression of order 2):

```
for learning_rate, color, num_iterations in
     zip([0.1, 0.01, 0.001, ['r', 'g', 'b'], [100, 100, 100]):

    c0 = 0
    c1 = 0

    plt.figure(figsize=(3,2))
    plt.scatter(x, y, s=1, c='b')
    plt.title(f'LR = {learning_rate}')

    for i in range(num_iterations):

        c0_gradient = 0
        c1_gradient = 0

        N = float(len(points))
```

```
        total_cost = 0

        idx = np.random.randint(0, len(points))
        x_j, y_j = x[idx], y[idx]

        c0_gradient += 2 *          (y_j - (c0 + c1 * x_j))
        c1_gradient += 2 * x_j    * (y_j - (c0 + c1 * x_j))

        c0 = c0 + learning_rate * c0_gradient
        c1 = c1 + learning_rate * c1_gradient

        x_new = x[np.argsort(x)]
        pred = c0 + c1 * x_new

        plt.plot(x_new, pred, c=color, alpha=0.1)

    plt.show()
```

When done, answer the following:

	Question	Answer
1	Add the cost / error calculation into the code and then show the convergence of the error + the fitted polynomial fitted line.	
2	Convert the code into a order 3 polynomial, i.e. with c0, c1, and c2. When done, show the convergence of the error + the fitted polynomial line.	
3	Convert the code into an order 4 polynomial, i.e. with c0, c1, c2, and c3. When done, show the convergence of the error + the fitted polynomial line.	

Deliverable

Submit a report containing answers asked, along with your full jupyter-notebook working.