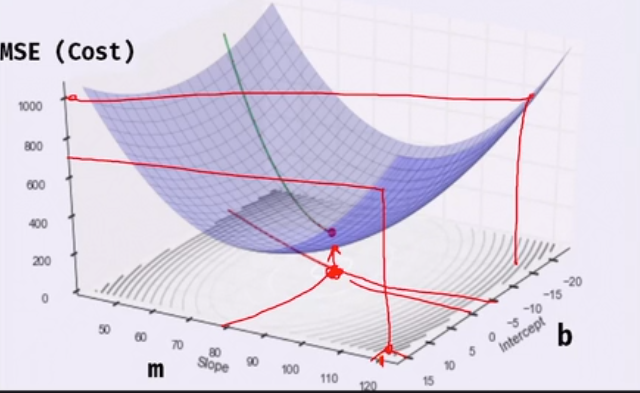
**Gradient Descent**

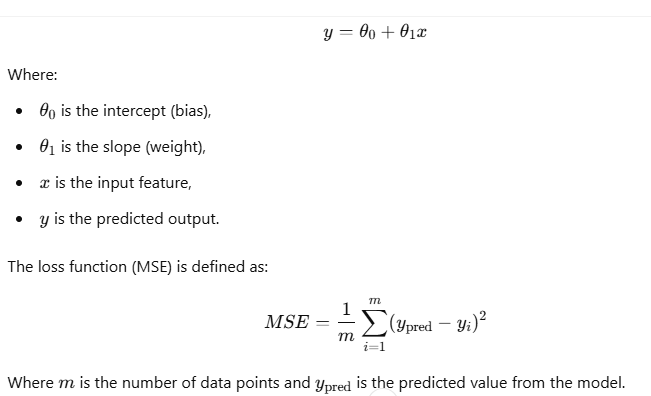
****

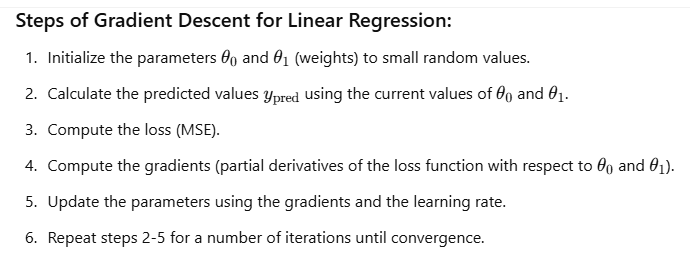
To emulate gradient descent in a linear regression model without using libraries like sklearn and instead implement it from scratch, here's a simple Python code and explanation:

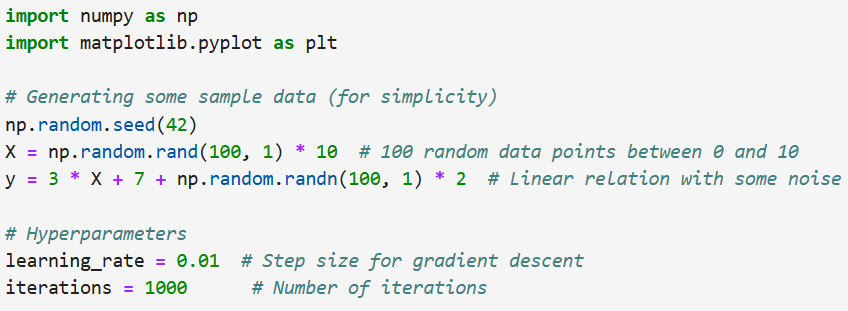
### Overview:

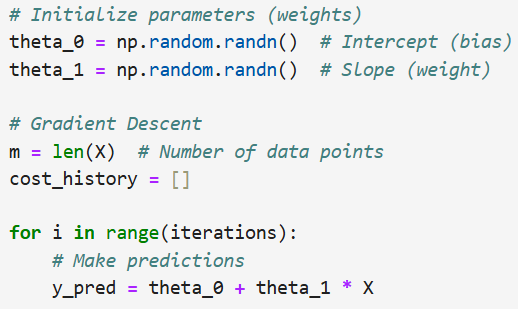
Gradient descent is an optimization algorithm used to minimize the loss function. For linear regression, we aim to minimize the Mean Squared Error (MSE) between the predicted values and actual values.

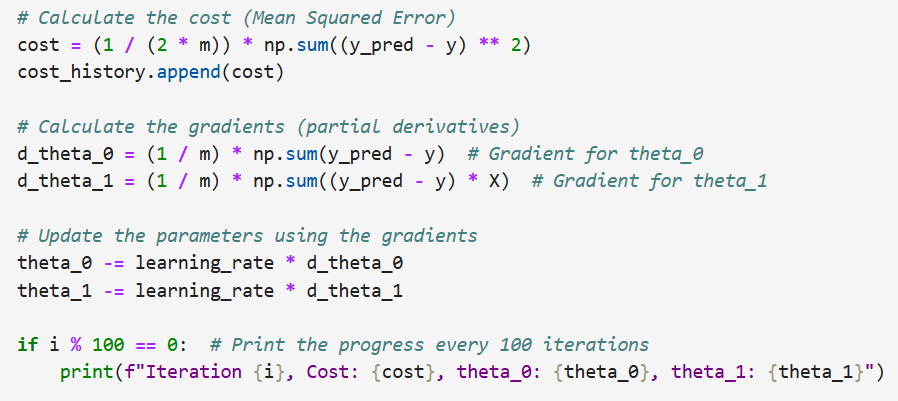
In linear regression, the model is typically represented by the equation:

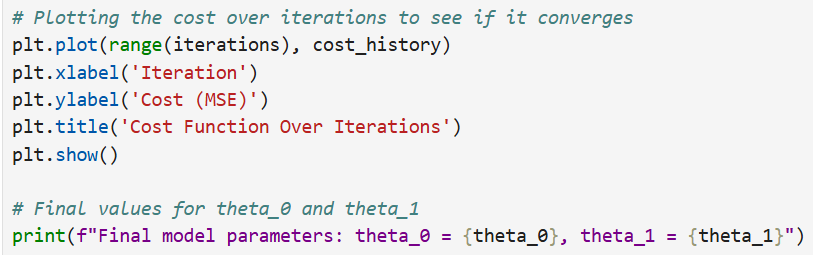


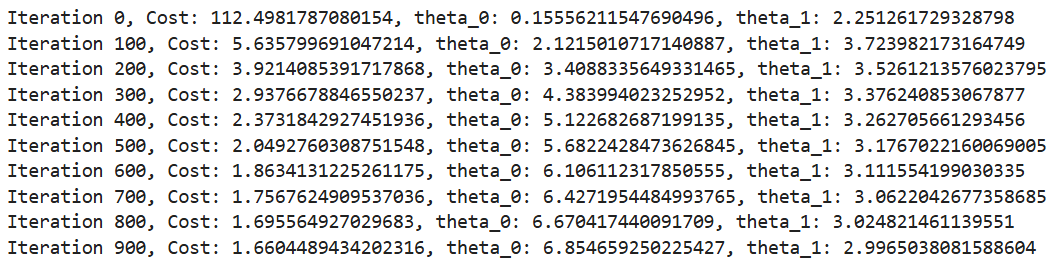


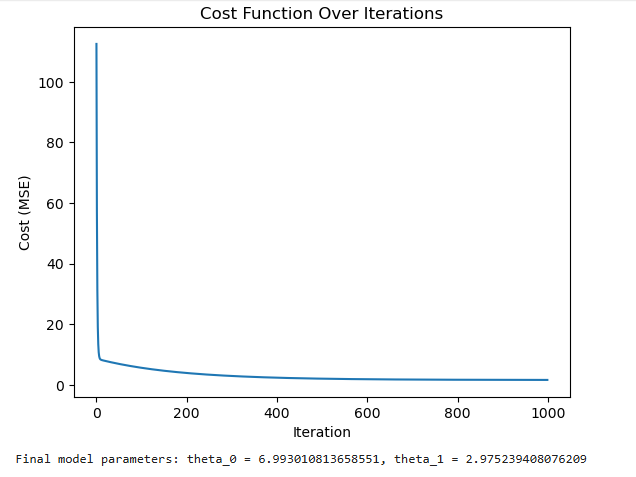












**Explanation of the Code:**

1. **Data Generation:**
   * We create a dataset with 100 data points where X is a set of random values and y is generated by a linear function y=3X+7 with some added Gaussian noise (np.random.randn).
2. **Hyperparameters:**
   * **Learning rate** controls how big a step we take during each update of the parameters.
   * **Iterations** specifies the number of times the gradient descent loop will run to optimize the parameters.
3. **Initialization:**
   * The parameters theta\_0 (intercept) and theta\_1​ (slope) are initialized to random values.
4. **Gradient Descent Loop:**
   * **Predictions:** We calculate the predicted values using the current values of theta\_0 and theta\_1​.
   * **Cost (MSE):** We compute the cost (MSE) to check how well our model is doing.
   * **Gradients:** We calculate the gradients (partial derivatives) of the cost with respect to theta\_0​ and theta\_1​.
   * **Update:** We adjust the parameters by subtracting the gradients multiplied by the learning rate. This is the key part of gradient descent.
5. **Plotting the Cost Function:**
   * The cost is plotted over the iterations to visualize the convergence. We expect to see a downward trend as the algorithm minimizes the error.
6. **Final Model Parameters:**
   * After 1000 iterations, we print the final values for theta\_0 and theta\_1​, which should approximate the true values (3 and 7 in this case, considering how we generated the data).

**Output:**

You will see the cost values printed every 100 iterations, showing how the model is progressively minimizing the error.

The final values of theta\_0​ and theta\_1​ should be close to 7 and 3 respectively.

**Key Points:**

* **Gradient Descent** is an iterative process of updating the parameters to minimize the cost function.
* The **Learning Rate** controls how big each update is. If it's too large, it can overshoot the minimum. If it's too small, the convergence might be slow.
* The model is trained by updating the parameters based on the gradients, which tell us how to move to reduce the cost.

This emulates the model.fit in regression but is done manually using gradient descent!