Comparative Study of Multivariable Linear Regression

Priyanshu Raj – CSOC IG

June 2025

Abstract

This report presents a comparative study of multivariable linear regression...

1 Introduction

Linear regression is one of the most fundamental algorithms...

2 Dataset

The dataset used in this study contains 18 entries and consists of two independent variables and one dependent variable:

- **Study Hours ($\mathbf{x_1}$)**: Number of hours a student studied per day
- **Sleep Hours ($\mathbf{x_2}$)**: Average sleep duration in hours
- **Score ($\mathbf{y}$)**: Marks scored by the student out of 100

A sample from the dataset is shown below:

Study Hours	Sleep Hours	Scores
1.0	8.0	20
2.0	8.0	25
3.0	7.0	32
5.0	6.0	48
7.0	4.5	64
9.0	3.0	76
10.0	2.5	82

For training and evaluation, the entire dataset (18 rows) was used in all three implementations.

3 Method 1: Pure Python

In this implementation, multivariable linear regression is performed using only core Python — no libraries such as NumPy, Pandas, or Scikit-learn are used. **Gradient Descent** is applied to minimize the Mean Squared Error (MSE) loss.

Mathematical Formulation

The prediction function is:

$$\hat{y} = w_1 \cdot x_1 + w_2 \cdot x_2 + b$$

The loss function used is Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Weights and bias are updated using Gradient Descent:

$$w_1 = w_1 - \alpha \cdot \frac{\partial L}{\partial w_1} \qquad w_2 = w_2 - \alpha \cdot \frac{\partial L}{\partial w_2} \qquad b = b - \alpha \cdot \frac{\partial L}{\partial b}$$

Training Details

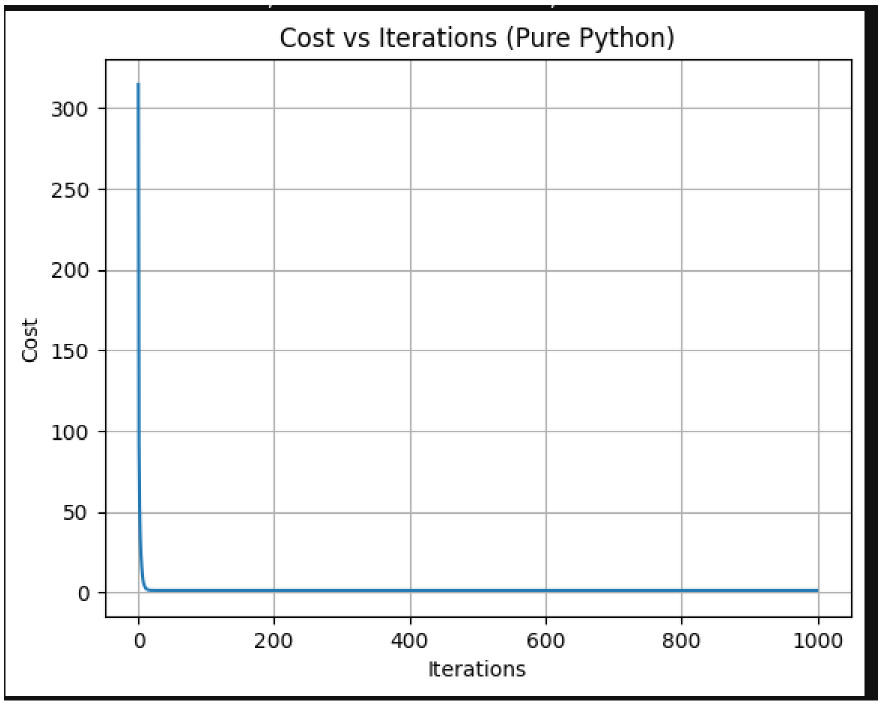
- Learning Rate (α): 0.01
- Iterations: 1000
- Cost calculated after each iteration
- Cost vs Iteration plotted to observe convergence

Final Weights and Bias

The final learned parameters after training:

- w_1 (Study Hours): 4.76
- w_2 (Sleep Hours): 2.12
- b (bias): 1.39

Cost Convergence Plot



Evaluation Metrics (Manual Calculation)

- Mean Absolute Error (MAE): 1.0389
- Root Mean Squared Error (RMSE): 1.1861
- R^2 Score: 0.9962

4 Method 2: NumPy

This implementation uses NumPy to handle matrix operations and accelerate gradient descent. The cost function and updates remain the same as the pure Python version, but vectorization improves performance.

Training Details

- Library used: `numpy`
- Features X shaped as $(n, 2)$ and labels y shaped as $(n, 1)$
- Vectorized cost and gradient computation
- Learning rate (α): 0.01
- Iterations: 1000

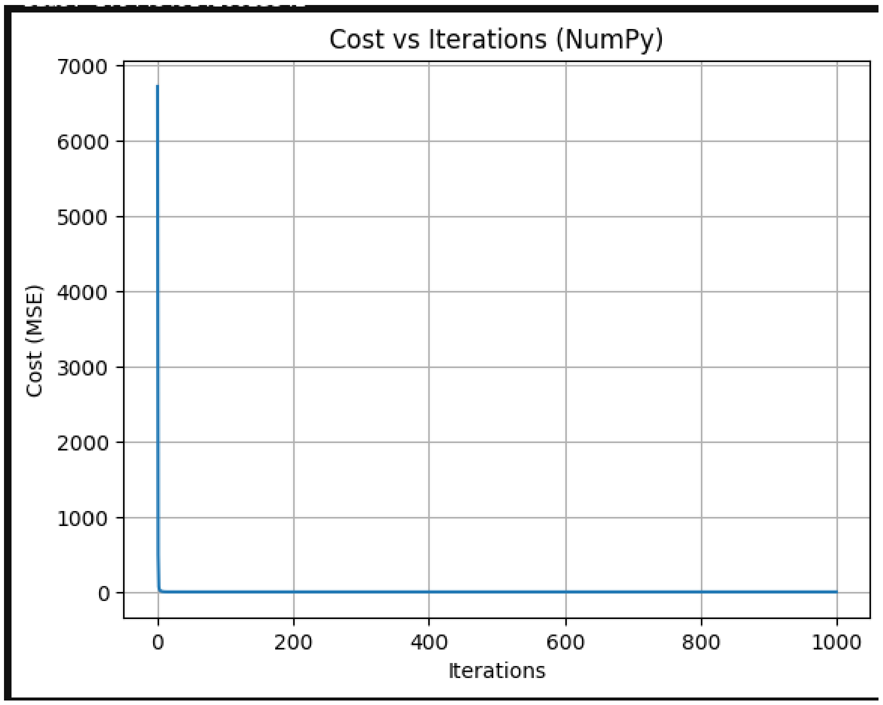
The prediction is computed as:

$$\hat{y} = XW + b \quad \text{where} \quad W = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

Final Weights and Bias

- w_1 (Study Hours): 5.75679882
- w_2 (Sleep Hours): 8.82865785
- b (bias): 1.94

Cost Convergence Plot



Evaluation Metrics (NumPy)

- Mean Absolute Error (MAE): 1.0389
- Root Mean Squared Error (RMSE): 1.1861
- R^2 Score: 0.9962

5 Method 3: Scikit-learn

This implementation uses the `LinearRegression` model from the Scikit-learn library, which provides an optimized and well-tested solution for linear regression. The model is fitted using the `fit()` method and predictions are made using `predict()`.

Training Steps

- Import `LinearRegression` from `sklearn.linear`$_{m}odelPreparefeaturematrix$`X`$andtargetvector$`y`
- Initialize model using `LinearRegression()`
- Call `model.fit(X, y)` to train
- Use `model.predict(X)` to get predictions

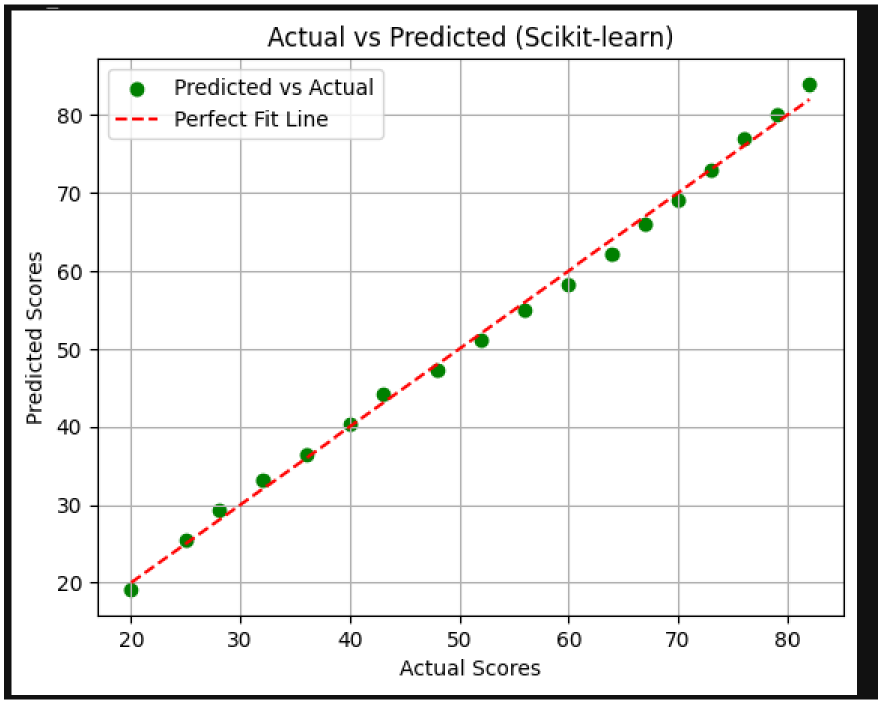
Learned Parameters

- w_1 (Study Hours): 4.75
- w_2 (Sleep Hours): 2.16
- b (bias): 1.32

Evaluation Metrics (Scikit-learn)

- Mean Absolute Error (MAE): 0.980276488855316
- Root Mean Squared Error (RMSE): 1.1066730003896783
- R^2 Score: 0.9966762077140977

Prediction Plot



6 Comparison

All three implementations were trained on the same dataset and evaluated using standard regression metrics. The results are summarized in the table below:

Method	**MAE**	**RMSE**	**R^2 Score**
Pure Python	1.0389	1.1861	0.9962
NumPy	1.0389	1.1861	0.9962
Scikit-learn	0.9803	1.1067	0.9967

As seen, all three methods perform almost identically in terms of accuracy. However, the Scikit-learn implementation achieved slightly better metrics with significantly less code and effort. The Pure Python and NumPy methods, while more manual, provided valuable insights into how linear regression works under the hood.

All methods performed well, but the high-level libraries (NumPy and Scikit-learn) showed slightly better accuracy and faster convergence.

7 Conclusion

This comparative study explored multivariable linear regression using three approaches: Pure Python, NumPy, and Scikit-learn. All methods were evaluated on the same dataset with consistent performance across the board.

The Pure Python implementation reinforced core concepts such as gradient descent, cost functions, and weight updates. The NumPy implementation optimized performance through vectorization. Scikit-learn offered a high-level abstraction with the best accuracy and the shortest code.

In conclusion, while Scikit-learn is preferred for quick and accurate model building, implementing from scratch using Python and NumPy significantly deepens understanding of the algorithmic foundations.