Linear Regression using Gradient Descent and ML Library

Your Name

September 20, 2025

1 Dataset and Preprocessing

For both parts of this assignment, we used the **Daily Demand Forecasting Orders** dataset (UCI Repository, ID 409). This dataset contains 60 daily records with 12 predictive attributes and 1 target variable (total orders).

Preprocessing steps included:

- Removal of duplicate rows (none found)
- Conversion of categorical to numerical variables (not needed)
- Standardization of features using `StandardScaler`
- Removal of weakly correlated features with correlation < 0.3 with the target (see Figure 1)

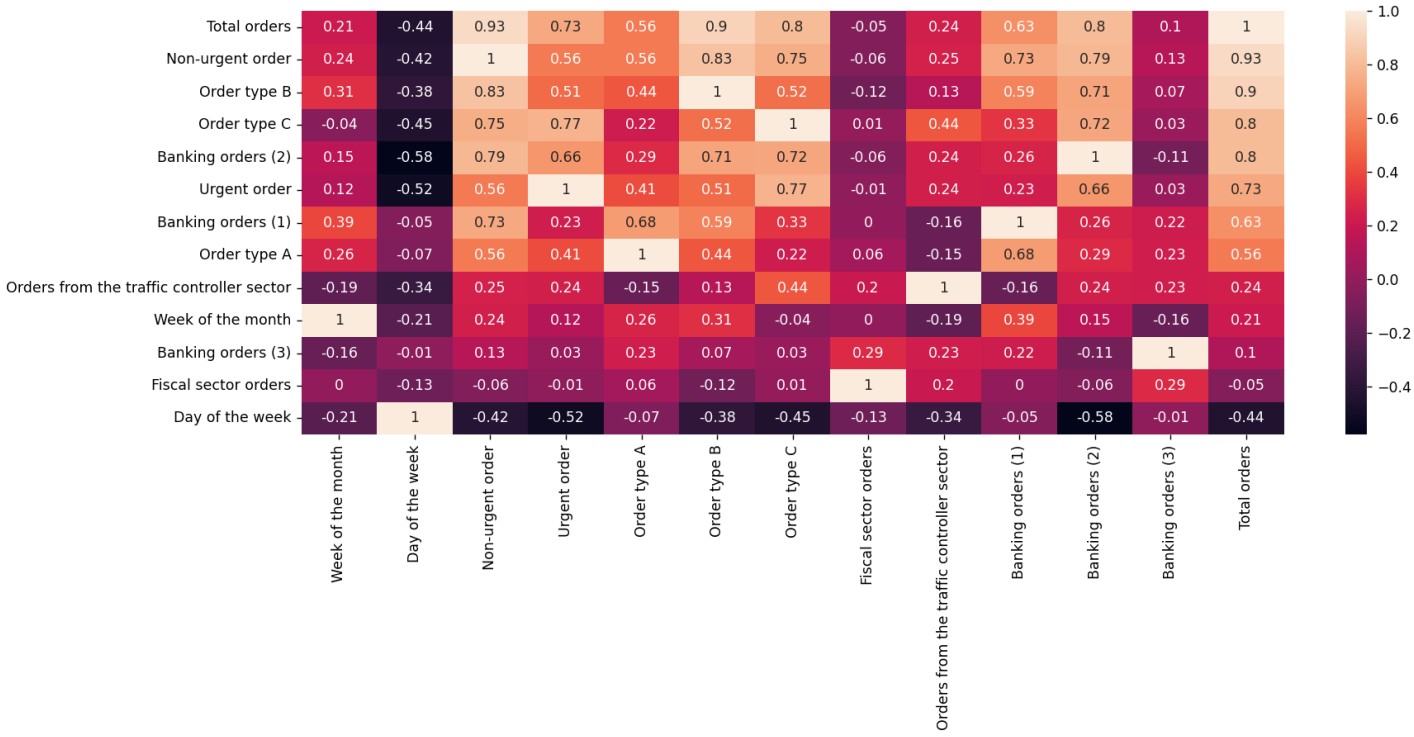

Figure 1: Correlation Matrix Heatmap

Figure 1 is a Correlation Matrix Heatmap ordered by correlation to the target variable, total orders. After removing 4 columns ("Fiscal sector orders", "Banking orders (3)", "Week of the month", "Orders from the traffic controller sector"), the final dataset contained 8 predictive attributes. I then used an 80/20 train-test split for the model.

2 Part 1: Linear Regression with Gradient Descent

We implemented a custom gradient descent algorithm from scratch in Python (no regression libraries). The update rule for weights is given by:

$$w := w - \eta \cdot \nabla_w J(w, b), \quad b := b - \eta \cdot \nabla_b J(w, b)$$

where J is the mean squared error (MSE) loss function.

2.1 Hyperparameter Tuning

We experimented with the learning rate (η), the number of iterations, and the tolerance (ϵ). The log of trials is shown in Table 1.

Table 1: Hyperparameter tuning log for Part 1

Learning Rate	Iterations	Tolerance	Train MSE	Test MSE
0.001	1000	1×10^{-6}	1.9445	0.3092
0.01	1000	1×10^{-6}	1.9445	0.3092
0.1	1000	1×10^{-6}	1.9445	0.3092
0.001	5000	1×10^{-6}	1.9445	0.3092
0.01	5000	1×10^{-6}	1.9445	0.3092
0.1	5000	1×10^{-6}	1.9445	0.3092
0.001	1000	1×10^{-3}	2.0811	0.3428
0.01	1000	1×10^{-3}	2.0811	0.3428
0.1	1000	1×10^{-3}	2.0811	0.3428
0.001	5000	1×10^{-3}	2.0811	0.3428
0.01	5000	1×10^{-3}	2.0811	0.3428
0.1	5000	1×10^{-3}	2.0811	0.3428
0.001	1000	1×10^{-9}	1.9445	0.3092
0.01	1000	1×10^{-9}	1.9445	0.3092
0.1	1000	1×10^{-9}	1.9445	0.3092
0.001	5000	1×10^{-9}	1.9445	0.3092
0.01	5000	1×10^{-9}	1.9445	0.3092
0.1	5000	1×10^{-9}	1.9445	0.3092

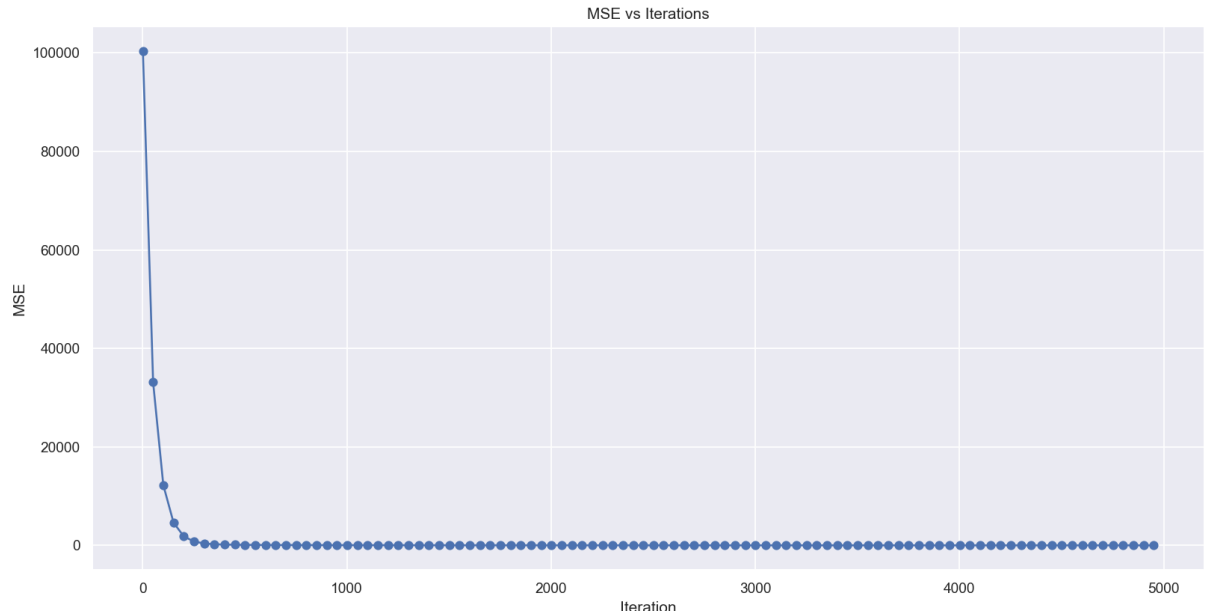

Figure 2: MSE vs Iterations for Gradient Descent

2.2 Results

The best set of parameters achieved:

- Training MSE: 1.9445, MAE: 0.9645, R^2: 0.9998, EV: 0.9998
- Testing MSE: 0.3092, MAE: 0.3953, R^2: 1.0000, EV: 1.0000

As shown in Table 1, various learning rates, iteration numbers, and tolerances were experimented with. The MSE did not significantly change across most variations, indicating that the gradient descent quickly reached a stable solution for this dataset. Therefore, I chose the hyperparameters that minimized the work the model needed to do to achieve the optimal solution.

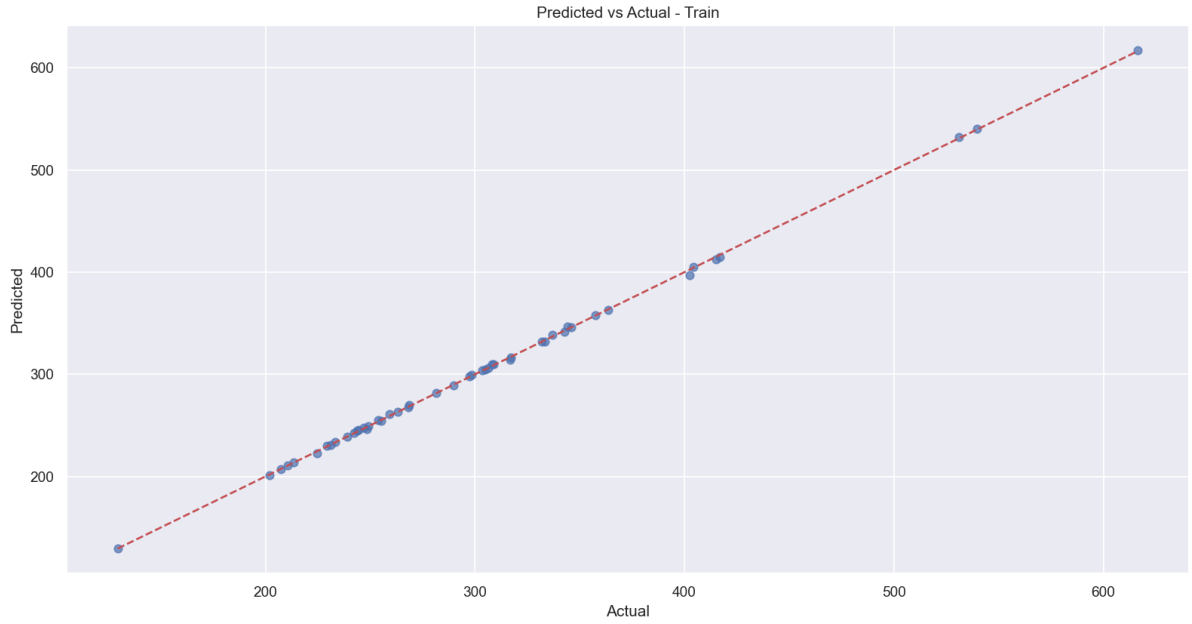

Figure 3: Predicted vs Actual (Train)

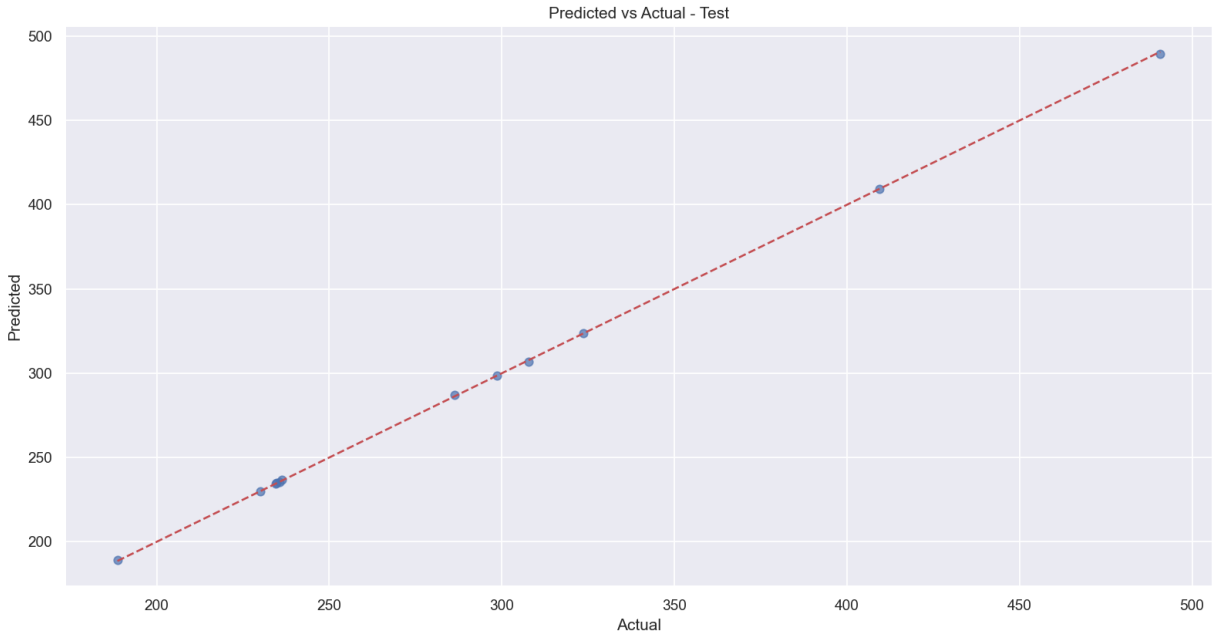

Figure 4: Predicted vs Actual (Test)

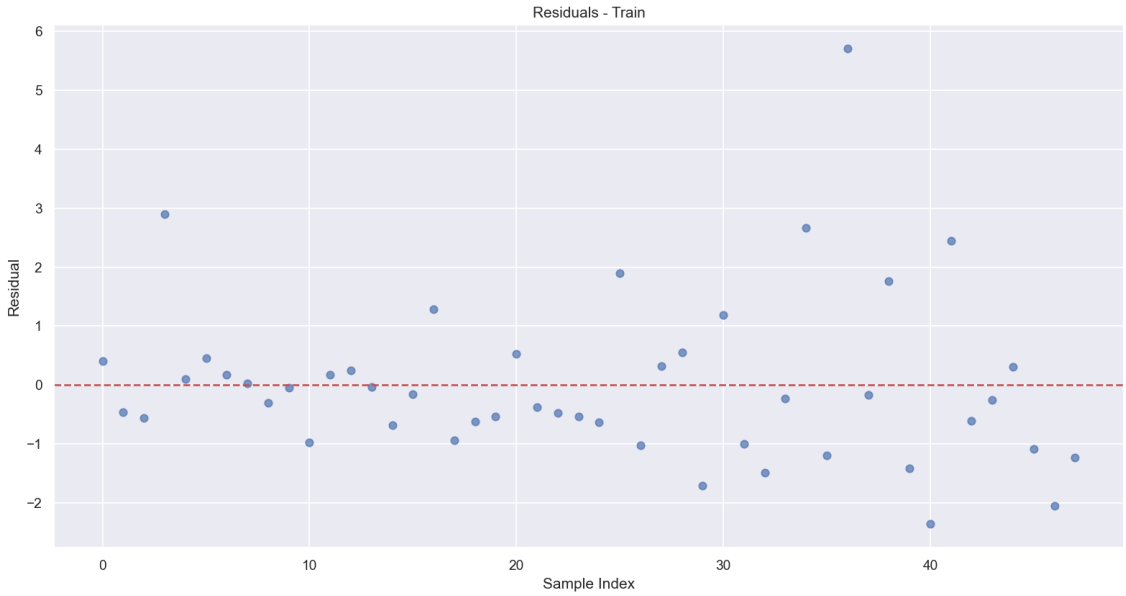

Figure 5: Residuals (Train)

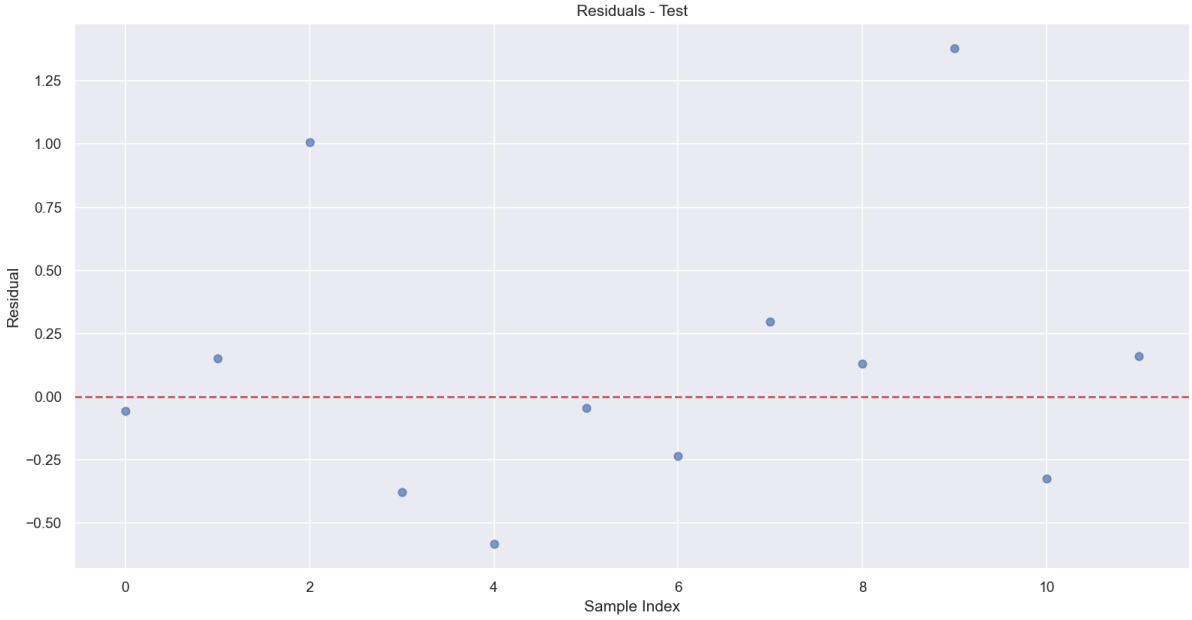

Figure 6: Residuals (Test)

Therefore, we selected:

- Learning Rate = 0.01
- Iterations = 1000

- Tolerance = 1e-6

as the most stable and optimal parameters for the model.

I am satisfied that this is the best solution, as further tuning of the hyperparameters did not produce any significant improvements in training or testing performance. Also, the metrics support the model being very accurate with low error and being able to explain the majority of the variance in the data.

3 Part 2: Linear Regression with `SGDRegressor`

We next applied `SGDRegressor` from scikit-learn on the same dataset. We used $max\_iter = 5000$, $tol = 1 \times 10^{-3}$, and $random\_state = 5$.

3.1 Results

The model achieved:

- Training MSE: 1.3964, MAE: 0.8174, R^2: 0.9998, EV: 0.9998
- Testing MSE: 0.2064, MAE: 0.3174, R^2: 1.0000, EV: 1.0000

Cross-validation confirmed stability:

- Average MSE: 20.7646 ($\pm$38.5352)
- Average R^2: 0.9970 ($\pm$ 0.0048)

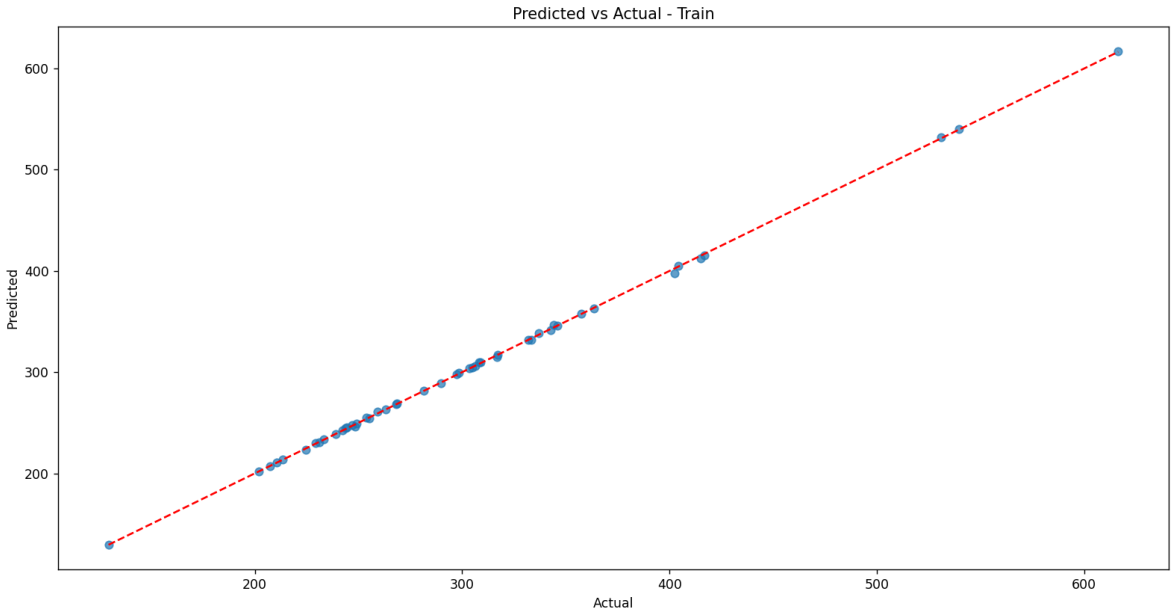

Figure 7: Predicted vs Actual (Train)

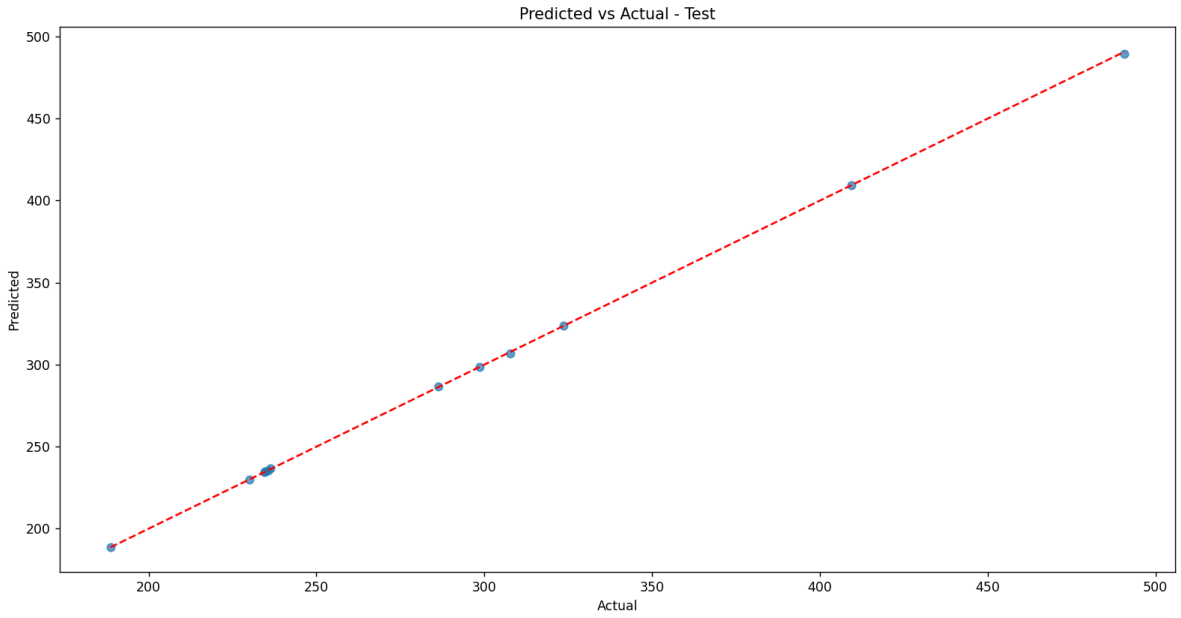

Figure 8: Predicted vs Actual (Test)

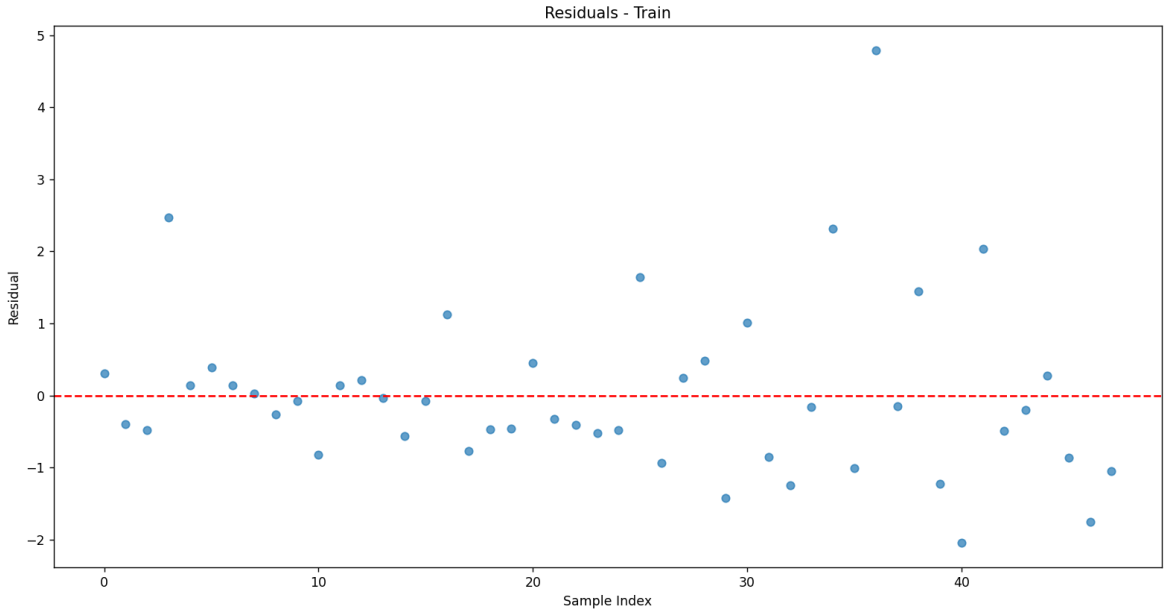

Figure 9: Residuals (Train)

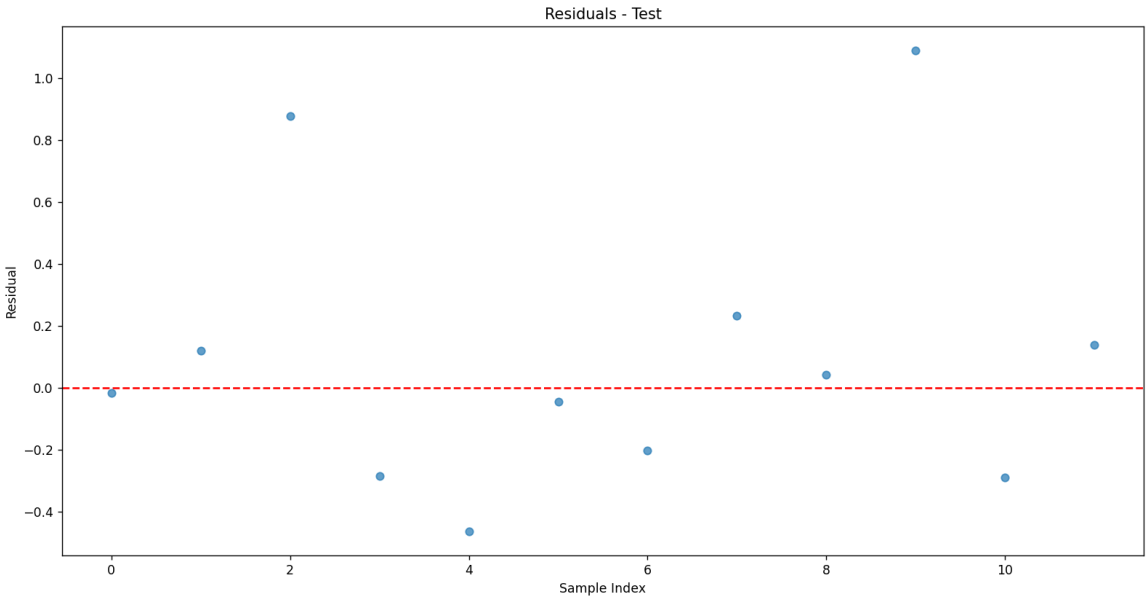

Figure 10: Residuals (Test)

I am satisfied that the package has found the best solution. It achieved better results compared to the linear regression model created with custom gradient descent. To ensure that the model was not overfitting, I performed cross-validation, which confirmed that the model generalizes well to unseen data. This demonstrates that the solution provided by the package is optimal and robust.

4 Conclusion

I am satisfied that both implementations found near-optimal solutions. The ML library implementation converged faster and gave slightly better results, but the custom implementation demonstrated a clear understanding of the gradient descent process. The library-based `SGDRegressor` slightly outperformed the custom implementation, achieving lower MSE values, but both models achieved excellent performance $R^2 \approx 1.0$. This is also illustrated by all of the figures looking almost identical between the linear regression model created with gradient descent and the SGDRegressor model.