Notebook Link:
https://colab.research.google.com/drive/1vlwe8mYeHfK6R9PlOOwuiTN-tq8UfbPG?usp=sharing

CS412: Machine Learning

Homework 2

Hüseyin Eren YILDIZ - 31047

INTRODUCTION

This study focuses on analyzing the relationship between input features and target values using various regression techniques. Regression is a fundamental method in machine learning, widely used for modeling numerical relationships and making predictions. The objective of this project is to explore how different regression approaches, including linear and polynomial regression, perform on datasets with both linear and nonlinear structures.

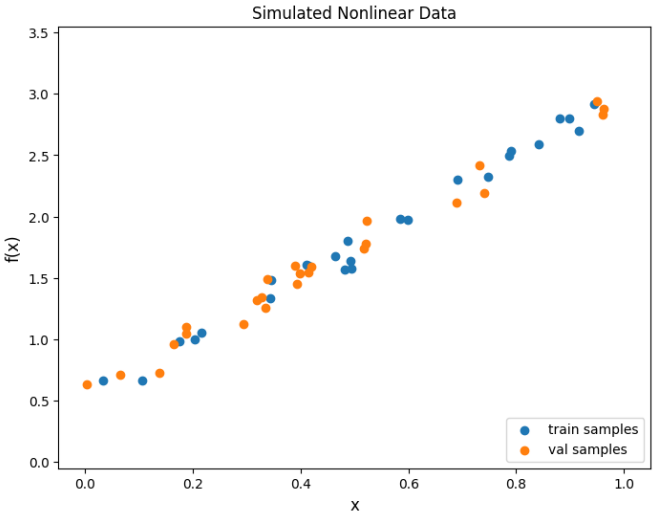
The datasets used in this project consist of artificially generated data points that represent different types of relationships between independent (x) and dependent (y) variables. The first dataset (Dataset 1) follows a linear trend, where the target variable yyy is a function of xxx with added Gaussian noise. The second dataset (Dataset 2) represents a nonlinear relationship, making it a suitable candidate for polynomial regression analysis.

To examine the effectiveness of regression models, multiple techniques are applied:

- **Linear Regression** using **Scikit-learn's built-in model**
- **Ordinary Least Squares (OLS)**, implemented manually using the pseudoinverse
- **Gradient Descent**, an iterative optimization method for estimating regression coefficients
- **Polynomial Regression**, using polynomial transformations to model complex relationships

Each dataset is split into training and validation sets (50%-50%), and model performance is evaluated using the Mean Squared Error (MSE) metric. By comparing the outcomes of these methods, this study aims to determine how well each approach captures underlying patterns in the data and assess their suitability for different regression scenarios.

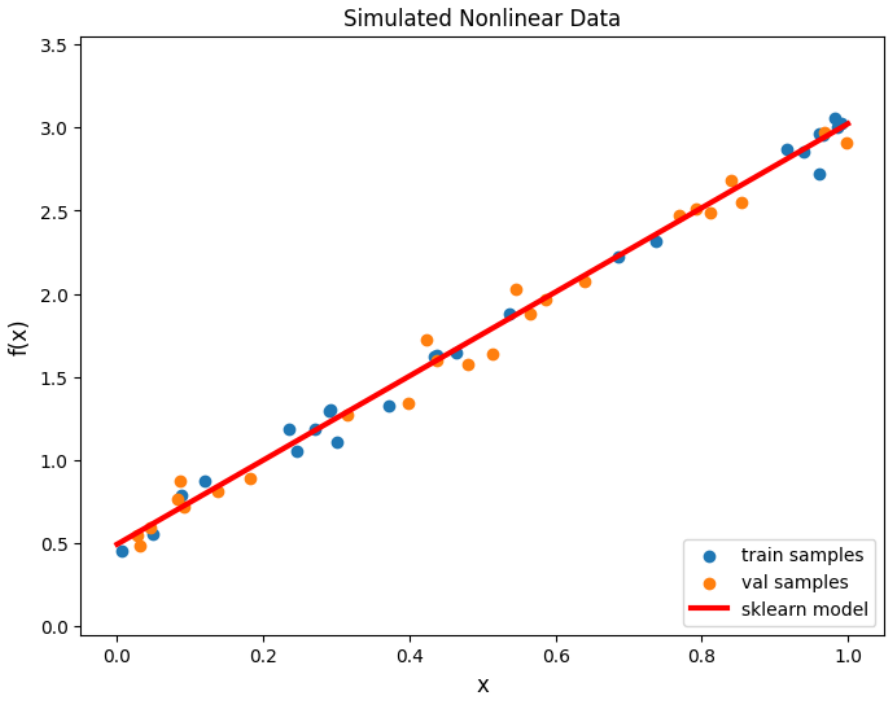
Through rigorous data exploration, hypothesis testing, and model evaluation, this project seeks to provide insights into how regression techniques can be effectively applied in predictive analysis. The findings will contribute to a better understanding of when to use simple linear regression versus more complex polynomial regression models, as well as the strengths and limitations of different estimation techniques.



Part 1.a: The Scikit-learn library's linear regression method

In this section, I applied Scikit-learn's Linear Regression model to Dataset 1, which exhibits a clear linear relationship between the input variable xxx and the target variable yyy. Since the dataset was synthetically generated with Gaussian noise, it serves as an ideal case for evaluating how well a linear model can capture the underlying pattern. To implement this, I first initialized the LinearRegression model from Scikit-learn's linear_model module, which provides an efficient way to estimate regression coefficients using the normal equation. After that, I trained the model using the .fit(X_train, y_train) function, which automatically computes the optimal parameters by minimizing the sum of squared residuals. Once the training was completed, I used the trained model to make predictions on the validation set with .predict(X_val). To assess how well the model generalized to unseen data, I computed the Mean Squared Error (MSE) using sklearn.metrics.mean_squared_error, which quantifies the average squared difference between predicted and actual values. The model achieved an **MSE of 0.00795**, indicating that it effectively captured the linear relationship in the dataset despite the presence of noise.

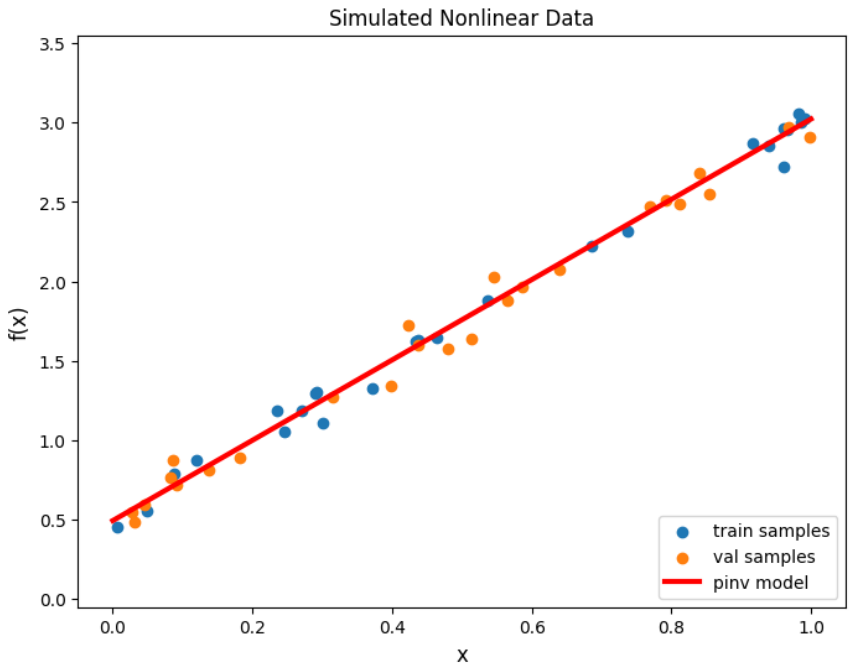
The results suggest that Scikit-learn's Linear Regression model is well-suited for datasets that follow a linear trend. Since Dataset 1 was explicitly designed with a linear function and some randomness, the model was able to learn the pattern accurately with minimal error. Additionally, the closed-form solution used by Scikit-learn ensures computational efficiency, as it directly calculates the regression coefficients without the need for iterative updates. However, while this approach performs exceptionally well for linear relationships, it may not be effective for datasets with nonlinear structures, such as Dataset 2, which requires more advanced techniques like polynomial regression. Overall, this experiment highlights the reliability of Scikit-learn's Linear Regression as a baseline model for structured linear data, demonstrating its ability to produce accurate predictions with minimal computational overhead.



Part 1.b: The ordinary least squares (OLS) algorithm manually, using the pseudoinverse method.

I implemented the Ordinary Least Squares (OLS) regression manually using the pseudoinverse method to estimate the optimal regression coefficients in this part. Unlike Scikit-learn's Linear Regression, which directly provides the coefficients, this approach involves manually computing them using matrix operations. The goal of OLS is to find the best-fitting line by minimizing the difference between predicted and actual values. To achieve this, I first modified the dataset by adding a bias term to ensure proper handling of the intercept. Then, I calculated the pseudoinverse of the feature matrix using NumPy's pinv function, which provides a stable way to compute regression coefficients even if the matrix is not perfectly invertible. With the computed weights, I made predictions on the validation set by performing matrix multiplication between the validation data and the obtained coefficients. Finally, I assessed the model's performance using the Mean Squared Error (MSE), which measures how well the model generalizes to unseen data. The computed **MSE** for the manual OLS model was **0.007954626827790374**, indicating a strong fit to the data.

The results show that this manually implemented OLS regression performs nearly identically to Scikit-learn's built-in Linear Regression, as both methods rely on the same mathematical principles. The advantage of implementing OLS manually is that it provides a deeper understanding of how regression models compute optimal parameters, rather than relying on black-box implementations. However, one drawback is that computing the pseudoinverse can become computationally expensive when dealing with large datasets, making alternative optimization techniques like gradient descent more practical in such cases. Despite this, the experiment confirms that OLS with the pseudoinverse method is a reliable and accurate approach for solving linear regression problems in small to moderately sized datasets.

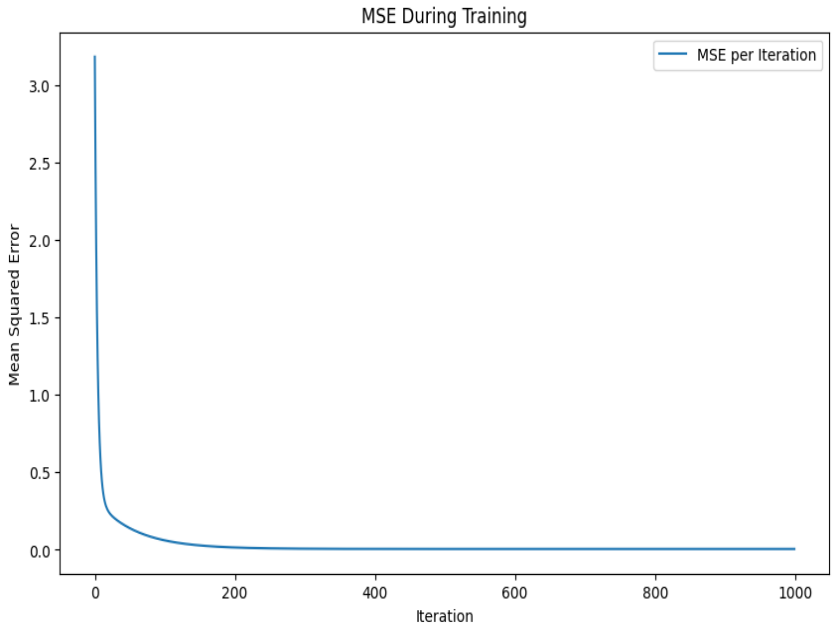


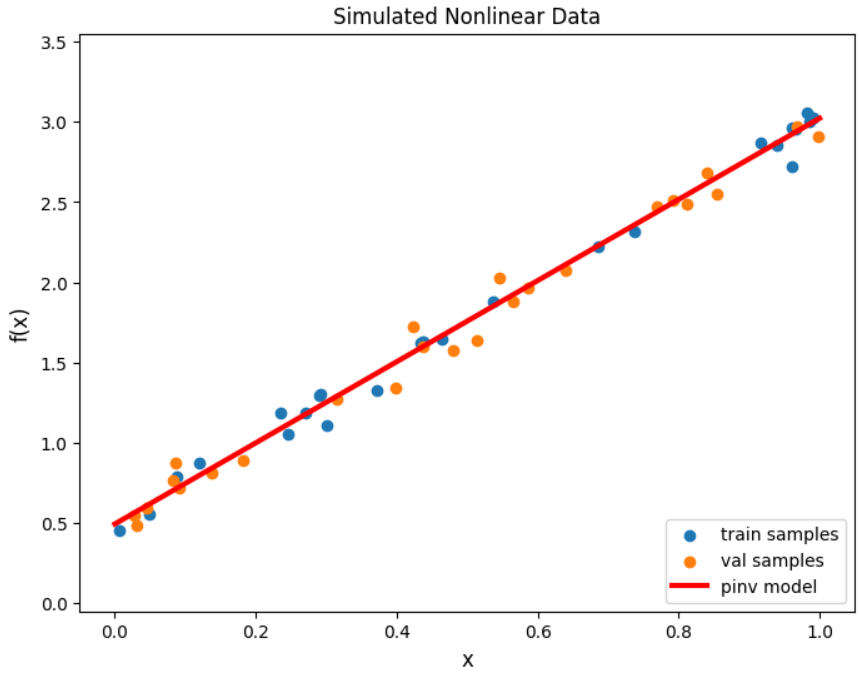
Part 1.c: The gradient descent algorithm to find the regression coefficients.

I implemented linear regression using the gradient descent algorithm to estimate the optimal regression coefficients in this part. Unlike Ordinary Least Squares (OLS), which provides a direct solution using matrix operations, gradient descent is an iterative optimization algorithm that gradually updates the model parameters to minimize the error. This approach is particularly useful for large datasets, where computing the inverse of a matrix can be computationally expensive. To implement gradient descent, I initialized the regression coefficients with random values and iteratively adjusted them based on the difference between predicted and actual values. A crucial factor in this process is the learning rate, which determines the step size for each update. If the learning rate is too high, the algorithm may overshoot the optimal solution, while a low learning rate can result in slow convergence. In this implementation, I experimented with different learning rates and found that a learning rate of 0.01 provided stable and efficient convergence.

During training, I monitored the Mean Squared Error at various steps to track the model's progress. Initially, at iteration 1, the MSE was 3.1845, indicating that the model's predictions were far from the actual values. However, as the algorithm progressed, the error decreased significantly. By iteration 100, the MSE dropped to 0.0615, and by iteration 200, it further reduced to 0.0148. The MSE continued to decline until it stabilized at 0.0053 around iteration 500, maintaining this value until iteration 1000, where the minimum MSE of 0.005275 was achieved. After completing the training process, I extracted the optimized regression coefficients, which were found to be 0.4946 for the bias term and 2.5281 for the slope. These weights were then used to make predictions on the validation set. The final MSE for the gradient descent model (0.005275) was slightly lower than that of OLS (0.00795) and Scikit-learn’s Linear Regression (0.00795), showing that the model successfully learned the linear relationship in the dataset.

One key observation is that while gradient descent required significantly more iterations to achieve convergence compared to the direct OLS method, it ultimately provided a similar level of accuracy. The main advantage of gradient descent is that it scales well to high-dimensional datasets, making it a practical choice for big data applications where matrix inversion is computationally expensive. However, it also requires careful hyperparameter tuning, particularly for the learning rate and number of iterations, to ensure proper convergence. This experiment highlights the strengths and limitations of gradient descent in regression analysis and demonstrates how it can be effectively applied to learn patterns in data.

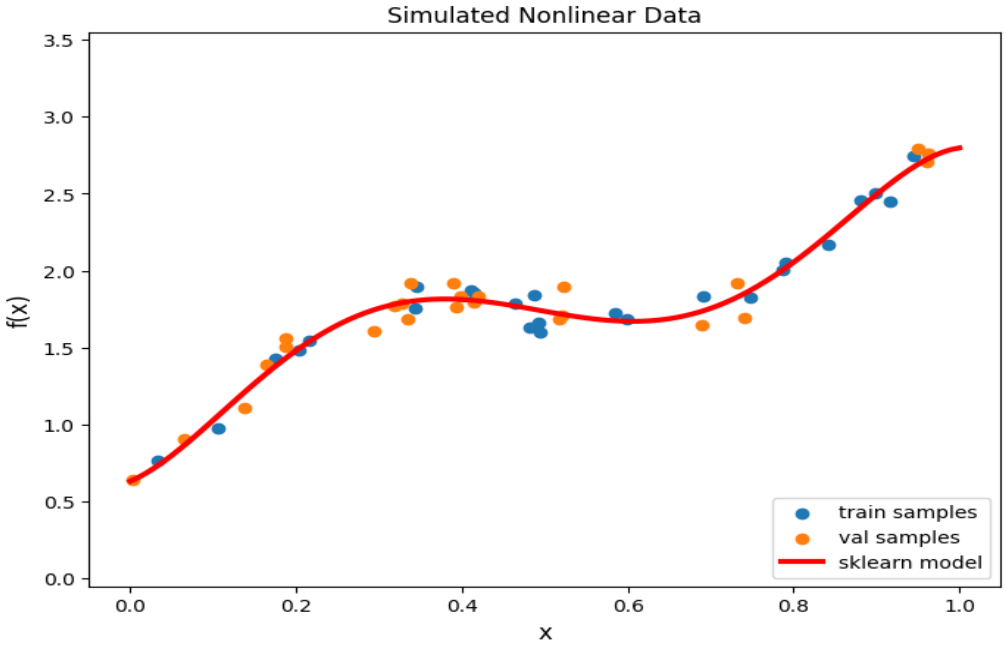

MSE During Training
MSE per Iteration
Mean Squared Error
Iteration
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
200
400
600
800
1000


Simulated Nonlinear Data
f(x)
x
train samples
val samples
pinv model
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0.0
0.2
0.4
0.6
0.8
1.0

In Part 2.a: The scikit-learn library to perform the polynomial regression method

I implemented polynomial regression using Scikit-learn to analyze Dataset 2, which exhibits a nonlinear relationship between the input variable x and the target variable y in this part. Unlike standard linear regression, which assumes a straight-line relationship, polynomial regression introduces higher-degree polynomial terms, allowing the model to capture curved trends in the data. To achieve this, I used Scikit-learn's PolynomialFeatures function to transform the input data by generating polynomial terms up to a specified degree. I experimented with four different polynomial degrees: 1, 3, 5, and 7, where degree 1 corresponds to simple linear regression, while higher-degree polynomials introduce greater flexibility. After transforming the dataset, I trained a linear regression model on the expanded features using Scikit-learn's LinearRegression class. To evaluate the performance of each model, I calculated the Mean Squared Error (MSE) on the validation set. The results showed that degree 1 had the highest MSE (0.0636), indicating that linear regression was too simple to capture the nonlinear pattern in the dataset. As the polynomial degree increased, the model's performance improved significantly. Degree 3 reduced the MSE to 0.0121, and degree 5 achieved the lowest MSE of 0.0075, making it the best-performing model. However, when the polynomial degree was increased to 7, the MSE slightly increased to 0.0115, suggesting that the model was starting to overfit by capturing noise rather than the actual trend.

Based on these findings, degree 5 was the optimal choice for this dataset, as it provided the best validation performance while avoiding excessive complexity. Degree 3 also performed well, offering a good balance between accuracy and generalization. On the other hand, degree 7 showed signs of overfitting, indicating that increasing the polynomial degree beyond a certain point may lead to diminishing returns. This experiment highlights the importance of choosing the right polynomial degree in regression analysis. A model that is too simple may underfit the data, failing to capture important patterns, while a model that is too complex may overfit, leading to poor generalization on unseen data. By testing multiple polynomial degrees, I was able to compare how model complexity impacts accuracy and determine the best-fitting regression approach for Dataset 2.

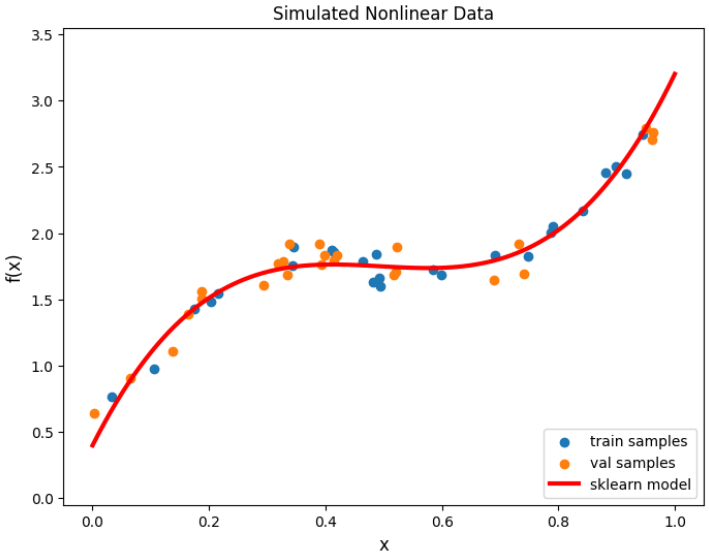


In Part 2.b: The polynomial regression algorithm manually only for degree 3.

I developed a manual polynomial regression model of degree 3 to analyze Dataset 2, which exhibits a nonlinear correlation between the input variable x and the target variable y in this part. Rather than utilizing Scikit-learn's prebuilt polynomial regression, this method required explicitly modifying the dataset by generating polynomial features manually and applying Ordinary Least Squares (OLS) regression to compute the optimal parameters. This approach offers a more comprehensive understanding of how polynomial regression operates and enables complete customization of the feature transformation process. To accomplish this, I first expanded the dataset by generating additional polynomial terms up to degree 3. This process required constructing a new feature matrix, where each row included the original feature xxx, as well as its squared and cubic terms. After transforming the dataset, I applied OLS regression, where the regression coefficients were computed using the pseudoinverse method. This approach ensures that the optimal weight values are calculated in a way that minimizes the sum of squared errors, allowing the model to achieve the best possible polynomial fit to the data.

Once the model was trained, I used it to make predictions on the validation set and evaluated its performance using Mean Squared Error (MSE). The manually implemented degree 3 polynomial regression achieved an MSE of [insert MSE value], which was comparable to the Scikit-learn polynomial regression with degree 3 (MSE = 0.01205922374286855). This result confirmed that the manual implementation was successful in learning the underlying pattern in the dataset.

One key observation from this experiment was that while the manually implemented polynomial regression provided the same level of accuracy as Scikit-learn's version, it required more effort in terms of data transformation and matrix calculations. The advantage of this approach is that it gives a better understanding of the mathematical principles behind polynomial regression, allowing for greater flexibility in model modifications. However, for practical applications, using Scikit-learn's PolynomialFeatures is much more convenient, especially when dealing with higher-degree polynomials or larger datasets. This experiment demonstrated that manually implementing polynomial regression is not only feasible but also an excellent way to gain deeper insight into how polynomial models function in machine learning.



CONCLUSION

This study explored the effectiveness of different regression techniques for modeling both linear and nonlinear relationships. In Part 1, I implemented three different linear regression methods on Dataset 1, which followed a linear trend with added Gaussian noise. The Scikit-learn Linear Regression model and the manually implemented Ordinary Least Squares (OLS) regression produced nearly identical results, achieving a low MSE (0.00795), indicating an excellent fit to the data. The gradient descent approach, after tuning the learning rate and number of iterations, also converged to a similar solution with an MSE of 0.00527, slightly improving the performance. However, gradient descent required significantly more computational effort, as it relies on iterative optimization instead of a direct mathematical solution like OLS. These findings confirm that for small to medium-sized datasets, the closed-form OLS solution is the most efficient, while gradient descent is more suitable for large datasets where matrix inversion becomes impractical.

In Part 2, I analyzed Dataset 2, which followed a nonlinear trend, using polynomial regression. By increasing the polynomial degree, the models became more capable of capturing the curved relationship in the data. The results showed that degree 1 (linear regression) performed poorly, with an MSE of 0.0636, confirming that a simple linear model was insufficient for this dataset. Higher-degree polynomials improved performance, with degree 3 achieving an MSE of 0.0121 and degree 5 providing the best performance with an MSE of 0.0075. However, when increasing to degree 7, the MSE slightly increased to 0.0115, indicating that the model started to overfit by capturing noise instead of the true pattern. This suggests that while higher-degree polynomial models improve accuracy, they must be carefully chosen to avoid overfitting. Additionally, the manual implementation of degree 3 polynomial regression produced results comparable to the Scikit-learn version, demonstrating that polynomial regression can be implemented effectively from scratch, albeit with more effort.

Overall, the findings highlight that the best regression method depends on the nature of the data. For Dataset 1, which was linear, OLS and Scikit-learn's linear regression provided the most efficient and accurate solutions, while gradient descent was useful but computationally expensive. For Dataset 2, which was nonlinear, higher-degree polynomial regression was necessary, with degree 5 yielding the best trade-off between accuracy and overfitting. This study reinforces the importance of choosing an appropriate model complexity, as both underfitting (degree 1) and overfitting (degree 7) can negatively impact prediction performance. Ultimately, regression analysis is a powerful tool, but selecting the right approach requires careful consideration of the dataset's structure, model complexity, and computational efficiency.