California House Price Predictor

Comparative Study Using Pure Python, NumPy, and Scikit-Learn

Panshull Choudhary
Indian Institute of Technology (BHU), Varanasi

May 19, 2025

Abstract

This report presents a comprehensive study of predicting California housing prices using multivariable linear regression, implemented in three stages. The project aims to understand how different programming paradigms affect performance and code complexity. The three approaches include a manual implementation in pure Python, a vectorized version using NumPy, and a library-based model using Scikit-learn. The California Housing Dataset from Kaggle is used throughout the study. The results reveal trade-offs in terms of code readability, execution time, and predictive accuracy.

1 Introduction

1.1 Motivation

House price prediction is a classic regression problem with real-world applications in real estate valuation and urban economics. This project explores how different computational tools affect the implementation and performance of a regression model.

1.2 Objective

To compare and contrast three implementations of multivariable linear regression:

- Pure Python
- NumPy-based
- Scikit-learn-based

2 Dataset Description

2.1 Source

The dataset is sourced from Kaggle: California Housing Prices Dataset.

2.2 Features

- **MedInc**, **HouseAge**, **AveRooms**, **AveBedrms**
- **Population**, **AveOccup**, **Latitude**, **Longitude**
- **MedHouseVal** (Target)

2.3 Preprocessing

- Removed missing values
- Normalized/standardized features
- Train-test split (80/20)

3 Phase 1: Pure Python Implementation

3.1 Theory

Linear regression model: $\hat{y} = Xw + b$
Loss function (MSE):

$$J(w, b) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})^2$$

3.2 Implementation Details

- Manual matrix operations
- Learning rate: 0.01, Iterations: 1000

3.3 Challenges

- Slow execution
- Debugging matrix operations

4 Phase 2: NumPy Implementation

4.1 Improvements

- Vectorized operations using NumPy
- Significant performance boost

4.2 Results

- Faster and more accurate

5 Phase 3: Scikit-learn Implementation

5.1 Why Scikit-learn?

- Clean API, optimized performance
- Built-in support for model evaluation

5.2 Code Sample

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

6 Visual Comparison

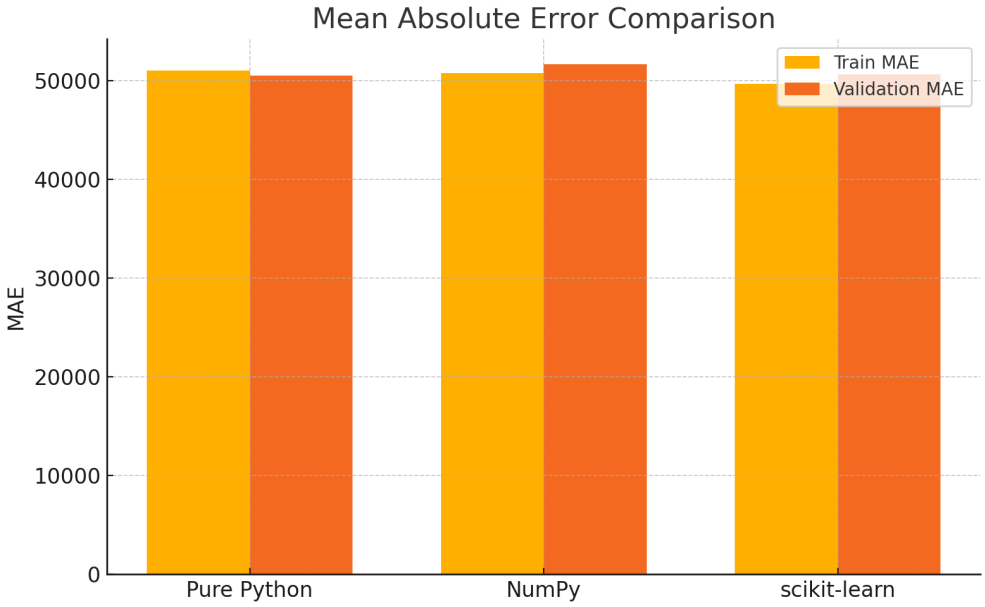

Figure 1: Mean Absolute Error Comparison

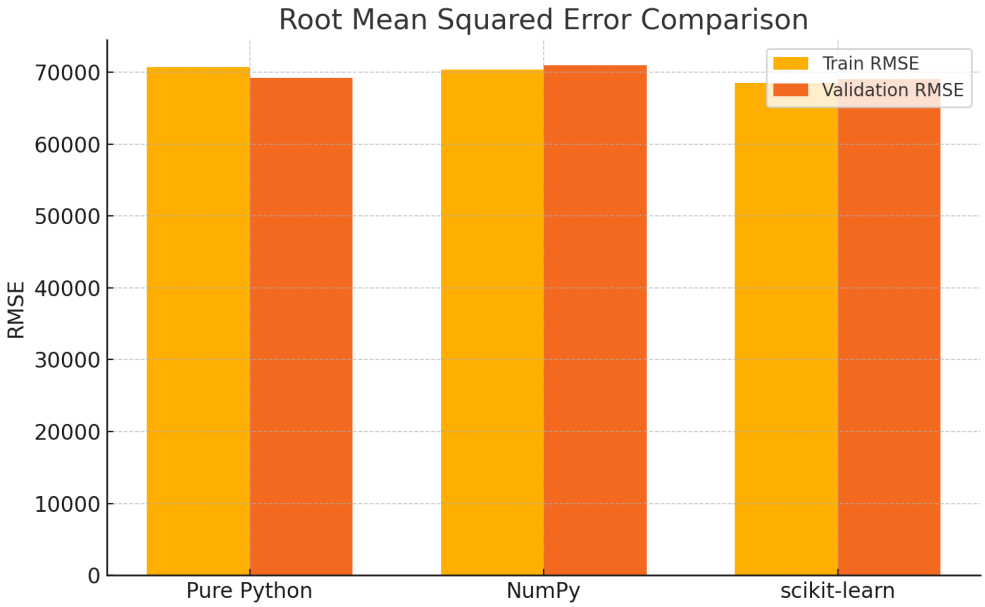

Figure 2: Root Mean Squared Error Comparison

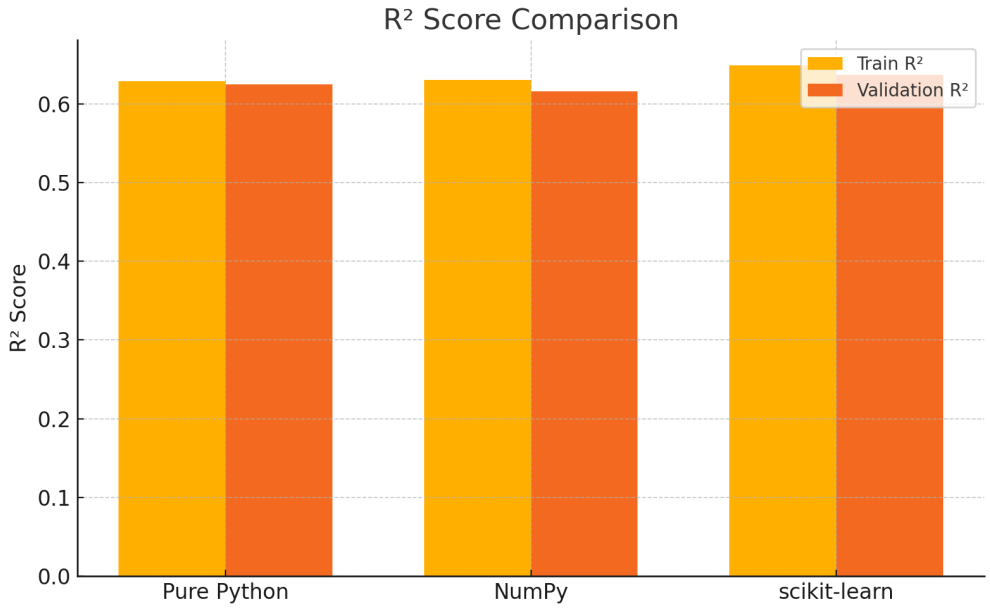

Figure 3: R^2 Score Comparison

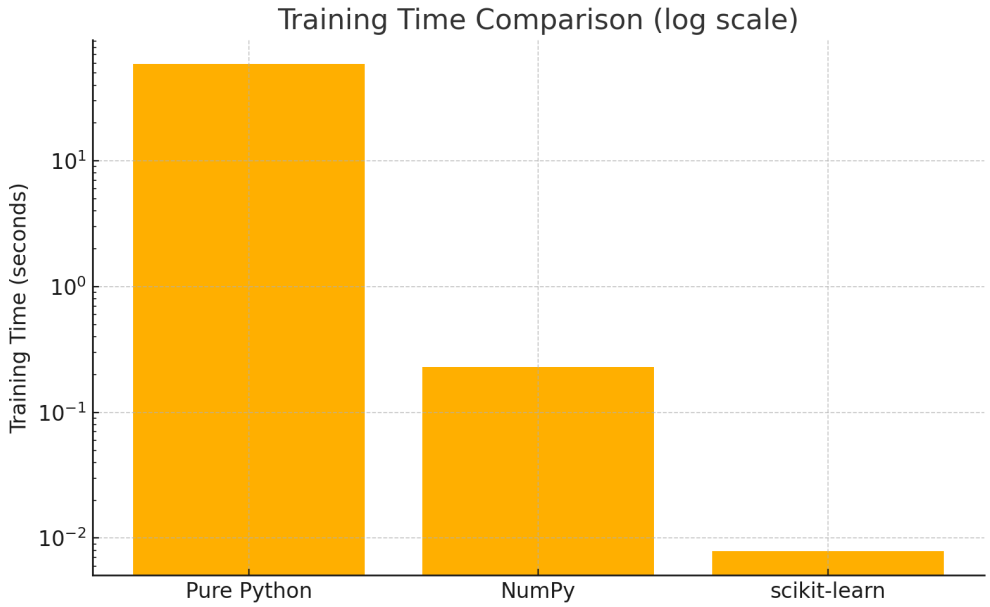

Figure 4: Training Time Comparison (log scale)

7 Conclusion

This project highlights the progressive impact of computational optimization and abstraction on model training efficiency and performance.

- The Pure Python implementation is educational but computationally expensive.

- The NumPy implementation is an efficient upgrade that uses vectorization.
- The Scikit-learn implementation is the most performant and robust.

Key Insight: Abstraction and library support not only reduce development effort but also improve accuracy and efficiency. In real-world scenarios, libraries like Scikit-learn strike the best balance between usability and performance.

Appendix

References

- `https://www.kaggle.com/datasets/camnugent/california-housing-prices`
- `https://scikit-learn.org/stable/modules/linear_model.html`