

MINERVA INVESTMENT MANAGEMENT SOCIETY
Quantitative Research Division



Forecasting 101

Bayesian Regression

Federico Lazzarini Head of Quantitative Research
Ianina Țarcă Quantitative Research Analyst

Academic Year 2024/2025

Abstract

In this technical series on econometrics and statistics, we explore various econometric tools, providing the underlying theory behind these models while analyzing a dataset of house prices in California, which is featured in the book *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow* by Aurélien Géron.

DISCLAIMER

This is an academic paper related to an academic project that does not represent any investment recommendation nor solicitation to buy or sell securities. The opinions expressed are subject to change. References to specific securities, asset classes and financial markets are for illustrative purposes only and are not intended to be interpreted as recommendations. Reliance upon information in this material is at the sole risk and discretion of the reader. The material was prepared only in regard to the research activity of Minerva Investment Management Society.

1 Introduction

The objective of this analysis is to apply Bayesian linear regression to model the relationship between various housing features and the median house value in a dataset. Unlike traditional regression techniques, Bayesian linear regression incorporates prior beliefs about the parameters and updates these beliefs based on the observed data, resulting in a posterior distribution. This approach provides not only point estimates for predictions but also quantifies uncertainty, which is valuable for decision-making in fields like finance and real estate.

In this implementation, we enhance Bayesian linear regression by using kernel-based feature expansion, which allows for capturing non-linear relationships between the features and the target variable, effectively extending linear regression into a higher-dimensional space where complex patterns can be more easily identified.

2 Theoretical Background

In traditional linear regression, we assume a linear relationship between inputs $\mathbf{x}$ and outputs y :

$$y = \mathbf{x}^T \boldsymbol{\beta} + \epsilon, \quad (1)$$

where:

- $\mathbf{x}$ is the input vector (features of a data point),
- $\boldsymbol{\beta}$ is the vector of model parameters (weights),
- $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is Gaussian noise with mean 0 and variance σ^2 .

In Bayesian linear regression, instead of estimating a single value for $\boldsymbol{\beta}$, we place a prior distribution over $\boldsymbol{\beta}$ and update this with the observed data to obtain a posterior distribution to capture uncertainty in the model parameters.

Nonlinear Relationships with Basis Functions

Linear regression assumes a direct linear relationship between the inputs $\mathbf{x}$ and the output y . However, this assumption can be too simplistic, especially for complex data. To make the model more flexible, we transform the inputs into a higher-dimensional space using basis functions (or kernels):

$$y = \phi(\mathbf{x})^T \boldsymbol{\beta} + \epsilon,$$

where $\phi(\mathbf{x})$ represents the transformed input vector in a higher-dimensional space.

Gaussian (RBF) Kernel: A common choice for $\phi(\mathbf{x})$ is the Gaussian kernel (or RBF kernel), defined as:

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2}\right),$$

where ℓ is a length scale parameter that controls the “smoothness” of the kernel.

- The RBF kernel maps the distance between two points $\mathbf{x}$ and $\mathbf{x}'$ to a similarity value between 0 and 1.

- If $\mathbf{x}$ and $\mathbf{x}'$ are close, $\|\mathbf{x} - \mathbf{x}'\|$ is small, making $k(\mathbf{x}, \mathbf{x}')$ close to 1, which indicates high similarity.
- If $\mathbf{x}$ and $\mathbf{x}'$ are far apart, $\|\mathbf{x} - \mathbf{x}'\|$ is large, so $k(\mathbf{x}, \mathbf{x}')$ is close to 0, indicating low similarity.

The RBF kernel thus transforms the distance between points into a similarity measure that decays exponentially with distance.

Polynomial Kernel The polynomial kernel introduces polynomial terms into the feature space, allowing the model to learn relationships that are more complex than linear but still interpretable. The kernel function for two input vectors $\mathbf{x}$ and $\mathbf{x}'$ is defined as:

$$k(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^T \mathbf{x}')^d,$$

where d is the degree of the polynomial.

The degree parameter d controls the complexity of the polynomial transformation. Higher degrees allow the model to capture more intricate patterns but can also lead to overfitting if not tuned carefully.

Matern Kernel The Matern kernel is a kernel function that interpolates between the RBF kernel and the absolute exponential kernel. It is commonly used in Gaussian processes and Bayesian linear regression to control the smoothness of the function. The Matern kernel with parameter $\nu = 1.5$ is given by:

$$k(\mathbf{x}, \mathbf{x}') = \left(1 + \frac{\sqrt{3}\|\mathbf{x} - \mathbf{x}'\|}{\ell}\right) \exp\left(-\frac{\sqrt{3}\|\mathbf{x} - \mathbf{x}'\|}{\ell}\right),$$

where ℓ is the length scale parameter. Similar to the RBF kernel, the length scale ℓ controls the smoothness of the resulting function. A smaller ℓ allows the model to be more responsive to changes in the input space, while a larger ℓ leads to a smoother function.

Assuming:

- The input data x has N data points.
- We have selected M basis points.
- An extra column is added for a bias (intercept) term.

Then, Φ will be an $N \times (M + 1)$ matrix with the following structure:

$$\Phi = \begin{bmatrix} 1 & k(x_1, b_1) & k(x_1, b_2) & \dots & k(x_1, b_M) \\ 1 & k(x_2, b_1) & k(x_2, b_2) & \dots & k(x_2, b_M) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & k(x_N, b_1) & k(x_N, b_2) & \dots & k(x_N, b_M) \end{bmatrix}$$

where:

- Each row represents a transformed feature vector for a data point x_i .
- The first column is all ones, representing the bias term.
- Each entry $k(x_i, b_j)$ is the kernel similarity between data point x_i and basis point b_j .

Instead of comparing every data point to all others (which is computationally expensive and could lead to overfitting), we compare each data point to a smaller subset of basis points.

In Bayesian linear regression, basis point selection plays a crucial role in transforming the input space and effectively capturing data patterns. Two methods for selecting basis points are **k-means clustering** and **random sampling**.

Random sampling selects basis points randomly from the input data without considering the data's spatial structure.

In contrast, k-means clustering partitions the data into clusters based on similarity and selects the centroids as basis points, which represent different regions in the data. It reduces redundancy by ensuring that basis points represent distinct clusters, leading to a more informative design matrix, and overall model performance.

Posterior Inference

To express our initial beliefs about β before observing any data, we place a Gaussian prior distribution over β :

$$\beta \sim \mathcal{N}(0, \alpha^2 I)$$

where:

- The prior mean is 0, representing an initial belief that the weights are centered around zero,
- $\alpha^2 I$ is the prior covariance matrix, where α^2 is a hyperparameter controlling the spread or uncertainty in the prior distribution of β .

Bayesian inference combines prior information with observed data to update beliefs about β . Using Bayes' theorem, the posterior distribution of β given Φ (the design matrix of transformed inputs) and y is:

$$p(\beta|\Phi, y) \propto p(y|\Phi, \beta) \cdot p(\beta)$$

where:

- $p(y|\Phi, \beta)$ is the likelihood, expressing how likely the observed data is given a particular β ,
- $p(\beta)$ is the prior, encoding our initial beliefs about β .

With a Gaussian prior and Gaussian likelihood, this posterior also has a Gaussian form:

$$\beta|\Phi, y \sim \mathcal{N}(m, V_n)$$

where m and V_n are the posterior mean and covariance, respectively.

The posterior mean m and posterior covariance V_n are given by:

$$V_n = \left(\alpha^{-2} I + \sigma^{-2} \Phi^T \Phi \right)^{-1}$$

$$m = \sigma^{-2} V_n \Phi^T y$$

where:

- V_n quantifies our updated uncertainty about β after observing the data, incorporating both the prior and the data,
- m is the mean of the posterior distribution, representing the most likely values for β given both the prior information and the observed data.

The Gaussian prior and Gaussian likelihood form a *conjugate prior* pair, meaning that the prior and likelihood belong to the same family of distributions. This conjugate relationship between the Gaussian prior on β and the Gaussian likelihood for y allows for a Gaussian posterior distribution as well.

Conjugacy is advantageous because the resulting posterior distribution remains Gaussian, allowing us to derive closed-form expressions for the posterior mean and covariance. This enables efficient computation without the need for iterative sampling methods, such as Markov Chain Monte Carlo (MCMC).

This conjugate setup is especially beneficial in linear models, where the Gaussian assumption aligns well with the common assumption of normally distributed errors.

Once the posterior distribution of β is obtained, it can be used to make predictions for new inputs. For a new input x_* , we compute the distribution over possible values of the output y_* :

$$y_*|x_*, \Phi, y \sim \mathcal{N}(\phi(x_*)^T m, \phi(x_*)^T V_n \phi(x_*) + \sigma^2)$$

where:

- $\phi(x_*)^T m$ is the mean prediction, providing the expected value of y_* ,
- $\phi(x_*)^T V_n \phi(x_*) + \sigma^2$ is the predictive variance, capturing uncertainty due to parameter estimation (from V_n) and observational noise (from σ^2).

However, not all problems have such conjugate structures, and in many real-world cases:

- The likelihood may not be Gaussian (e.g., for binary outcomes),
- The prior may be non-Gaussian, or
- The model may involve complex hierarchical structures that do not allow for closed-form solutions.

In these cases, MCMC provides a powerful alternative by enabling us to sample directly from the posterior distribution, even if we cannot solve it analytically.

How MCMC Works in Bayesian Inference

MCMC is a computational method that generates samples from the posterior distribution of model parameters by constructing a Markov Chain, a stochastic process whose current state depends only on the previous state and is independent conditionally on the previous state from all other previous states, that moves through the parameter space. Over many iterations, this chain converges toward the target posterior distribution. The distribution of these samples approximates the true posterior distribution, allowing us to estimate parameters and uncertainties.

MCMC Steps Applied to Bayesian Linear Regression

1. Specify the Model and Likelihood:

We specify the model as

$$y = \Phi\beta + \epsilon,$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

For each data point y_i given Φ_i and β , the likelihood is

$$p(y_i|\Phi_i, \beta) \sim \mathcal{N}(\Phi_i\beta, \sigma^2).$$

2. Define the Prior:

We place a Gaussian prior over β :

$$\beta \sim \mathcal{N}(0, \alpha^2 I).$$

3. Posterior Calculation Using MCMC:

- **Posterior:** Since we want to sample from

$$p(\beta|\Phi, y) \propto p(y|\Phi, \beta) \cdot p(\beta),$$

we combine our likelihood and prior.

- **Sampling:** Using an MCMC algorithm (e.g., Metropolis-Hastings or Hamiltonian Monte Carlo), we generate a sequence of samples $\{\beta^{(1)}, \beta^{(2)}, \dots, \beta^{(N)}\}$ that approximate the posterior distribution of β .

4. Generating Predictions:

With MCMC, we do not get a single posterior mean m and covariance V_n as in the analytical solution. Instead, we obtain a set of sampled β values.

For each sampled $\beta^{(k)}$, we can compute predictions $y_*^{(k)}$ for a new input x_* using:

$$y_*^{(k)} = \phi(x_*)^T \beta^{(k)}.$$

The mean and variance of these predictions across samples k provide the final prediction mean and uncertainty.

3 Methodology

Data Preprocessing

The dataset used in this study contains information about housing values in various regions in California, with 20,640 entries. Each entry represents a unique district, with features describing the geographic location, demographics, and housing characteristics of that area. Key variables include *longitude*, *latitude*, *housing age*, *number of rooms*, *population*, *number of households*, *median income*, and *proximity to the ocean*, as well as the target variable, *median_house_value*, which indicates the median value of houses in each district.

Preprocessing Step	Description
Handling Missing Values	Removed rows with missing values, especially in <code>total_bedrooms</code> , to retain data integrity.
Feature Scaling	Standardized numeric features to a mean of 0 and standard deviation of 1, ensuring proportional regularization.
Categorical Encoding	One-hot encoded <code>ocean_proximity</code> , creating binary features for categorical values.
Train-Validation Split	Split data into 80% training and 20% validation to enable independent assessment of model performance.

Table 1: Summary of Data Preprocessing Steps

Model Initialization and Setup

Firstly, the Bayesian linear regression model is initialized and we define the model's key hyperparameters:

- `kernel_params`: a dictionary containing parameters specific to the chosen kernel, such as the `length_scale` for RBF and Matern kernels or `degree` for the Polynomial kernel, and `number_of_bases`, the number of basis points to be selected.
- `sigma2`: the variance of the observation noise, representing the uncertainty in the data.
- `alpha2`: the prior variance of the weights, capturing the prior uncertainty about the model parameters.
- `kernel_type`: specifies the type of kernel function to be used, such as "RBF," "Polynomial," or "Matern."

Basis Point Selection

To select basis points, two methods are defined:

- `k_means_bases`: This method applies k-means clustering to the input data and returns the centroids of the clusters as basis points, effectively capturing different regions in the data space.
- `random_bases`: This method selects basis points randomly from the input data, providing a simpler and computationally efficient option.

In the `create_bases` function, the model checks which basis point selection method has been specified. If the selected `number_of_bases` exceeds the data size, the value is adjusted to avoid indexing errors. This flexibility ensures that basis points are chosen effectively, whether for simplicity (random sampling) or representativeness (k-means clustering).

Kernel Transformation

The `kernel_function` method calculates the similarity between data points and basis points, projecting the data into a higher-dimensional space. Each kernel, RBF, Polynomial or Matern, provides a different transformation approach.

Design Matrix Construction

The `create_design_matrix` method constructs the design matrix Φ , where each row represents a transformed data point in relation to each basis point. This matrix Φ serves as the input for the Bayesian linear regression model, capturing the structure of the transformed feature space. With the design matrix constructed, the model is ready to proceed to posterior inference and prediction steps.

Posterior Inference for Parameter Estimation

1. We want to calculate the posterior distribution over the weights β after observing the data.

2. **Steps:**

- **Prior Distribution:** We assume a Gaussian prior on β , defined as:

$$\beta \sim \mathcal{N}(0, \alpha^2 I),$$

where α^2 is the prior variance. This reflects our initial uncertainty about the weights before observing any data. In the code, this prior variance is set by the `alpha2` parameter during model initialization.

- **Posterior Mean and Covariance:**

- The **posterior covariance** V_n is computed as:

$$V_n = \left(\alpha^{-2} I + \frac{1}{\sigma^2} \Phi^T \Phi \right)^{-1},$$

where Φ is the design matrix created by the `create_design_matrix` method. This matrix V_n quantifies uncertainty in the parameter estimates.

- The **posterior mean** $\mathbf{m}$, representing the expected values of the weights given the observed data, is calculated as:

$$\mathbf{m} = \frac{1}{\sigma^2} V_n \Phi^T \mathbf{y},$$

where σ^2 is the observation noise variance, set in the `sigma2` parameter during initialization, and $\mathbf{y}$ is the target vector.

- The posterior mean $\mathbf{m}$ serves as the best estimate for the weights after observing the data, while V_n reflects the remaining uncertainty in these estimates.
- The `fit` method in the code computes V_n and $\mathbf{m}$ using matrix operations on Φ , making the inference efficient and eliminating the need for iterative sampling. These calculations yield the posterior distribution of the model weights.

Prediction with Uncertainty

1. Now, we aim to generate predictions for new data points x_* with associated uncertainty, providing a full probabilistic output.

2. **Predictive Mean and Variance:**

- For a new data point x_* , we first construct Φ_* , the transformed vector based on kernel similarities between x_* and each basis point. This transformation is handled in the `predict` method by calling `create_design_matrix` on the test data.

- **Predictive Mean** $\mathbb{E}[y_*]$:

$$\mathbb{E}[y_*] = \Phi_*^T \mathbf{m},$$

where $\mathbf{m}$ is the posterior mean computed during the `fit` process. This is the expected prediction value for y_* , representing the most likely outcome given the observed data.

- **Predictive Variance** $\text{Var}(y_*)$:

$$\text{Var}(y_*) = \Phi_*^T V_n \Phi_* + \sigma^2,$$

where V_n is the posterior covariance, and σ^2 is the observation noise. This variance reflects both parameter uncertainty (captured by V_n) and observation noise (captured by σ^2), providing a full measure of predictive uncertainty.

3. This Bayesian approach allows the model to output not only predictions but also a confidence measure for each prediction. The predictive variance quantifies this uncertainty, with the square root of the variance providing the predictive standard deviation, which is returned in the `predict` method.

Hyperparameter Tuning

1. To optimize model performance, hyperparameters such as the kernel type, length scale, number of basis points, and polynomial degree are tuned.

- **Grid Search:** We perform a grid search over various combinations of hyperparameters, including:
 - `kernel_type`: Specifies the type of kernel function (e.g., "RBF," "Polynomial," or "Matern").
 - `kernel_params['length_scale']`: Controls the spread of influence in the RBF and Matern kernels.
 - `kernel_params['degree']`: Specifies the degree for the Polynomial kernel.
 - `number_of_bases`: The number of basis points used to represent the data structure.
- For each configuration, the model is instantiated with the specific hyperparameter values, and the `fit` method is called on the training data. Predictions are made on the validation set using the `predict` method, and the mean squared error (MSE) is calculated as a performance metric.

2. The hyperparameter configuration that yields the lowest MSE on the validation set is selected as the optimal model configuration. This ensures that the model achieves the best balance between fit and generalization.
3. The hyperparameter tuning is performed in a loop outside of the class definition, iterating over each combination of hyperparameters, fitting the model, and evaluating MSE. The best configuration is stored and reported based on the lowest MSE achieved.

Model Evaluation and Final Training with Best Parameters

After identifying the best hyperparameter configuration through grid search, we evaluated the model's performance on the test dataset and trained the final model using the optimal parameters.

1. Unpacking the Best Parameters

The optimal parameters, stored in `best_params`, include:

- **kernel_type**: Specifies the type of kernel function (e.g., "RBF," "Polynomial," or "Matern").
- **length_scale**: Controls the spread for the RBF and Matern kernels.
- **number_of_bases**: The number of basis points used to represent the data structure.

For polynomial kernels, if the parameter **degree** is present, it is also extracted. A dictionary `kernel_params` is created to store these values and specify the basis sampling method as "KMeans" for consistent basis point selection.

2. Evaluating the Best Model on the Test Data

Using the test dataset, the model's performance is evaluated with two key metrics:

- **Mean Squared Error (MSE)**: Calculated as

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

where y_i is the true value, $\hat{y}_i$ is the predicted value, and N is the number of data points. Lower MSE values indicate a better model fit.

- **R-squared Score (R^2)**: Computed to assess how well the model explains the variability in the data. R^2 values close to 1 indicate a better fit, demonstrating the model's ability to capture patterns in the data.

The code calculates MSE using the unscaled data `mean_squared_error(y_test_original, y_pred_original)` and R^2 using `r2_score(y_test_original, y_pred_original)`.

4 Results

Posterior Summaries of Model Parameters

The Bayesian linear regression model provided insights into the estimated parameters by calculating their posterior means and 95% credible intervals. Table 2 lists the posterior mean and credible interval for each parameter, indicating the expected values of the coefficients and their uncertainty.

Table 2: Posterior Means and 95% Credible Intervals for Model Parameters

Parameter	Mean	95% Credible Interval
β_0	-0.5039	(-1.2265, 0.2187)
β_1	-0.9369	(-1.9209, 0.0470)
β_2	-0.0705	(-1.0382, 0.8973)

For each parameter β_i , the credible interval represents the range within which the parameter lies with 95% probability, given the observed data and prior beliefs.

Predictive Performance

The model's predictions were evaluated on the test set, assessing both the predictive accuracy and the associated uncertainty of predictions. The predictive mean for each test point was calculated along with its 95% predictive credible interval, which accounts for both parameter uncertainty and observation noise.

- **Mean Squared Error (MSE):** 1787904395.46
- **R-squared (R^2):** 0.7183

These metrics indicate that the model explains approximately 71.8% of the variance in housing prices and achieves a mean squared error of 1787904395.46, demonstrating a reasonably accurate fit.

Predictive Summaries

For each test point, the mean prediction and 95% credible interval were calculated. Table 3 provides a summary for a few representative test points.

Table 3: Predictive Means and 95% Credible Intervals for Test Points

Test Point	Predictive Mean	95% Credible Interval
1	0.154	(0.075, 0.233)
2	0.201	(0.125, 0.277)

These intervals reflect the model's uncertainty in each prediction, capturing the possible range of outcomes.

5 Results and Analysis

True vs. Predicted Values Plot

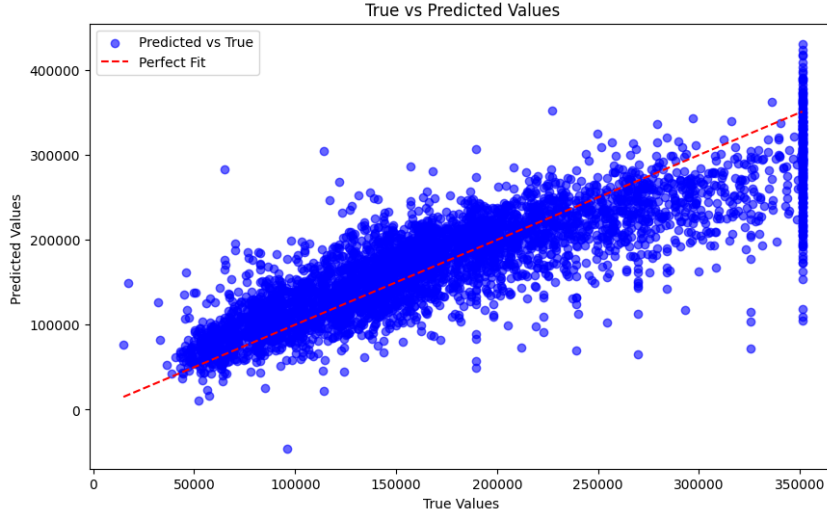


Figure 1: True vs. Predicted Values for Bayesian Linear Regression

This scatter plot (Figure 1) shows the predicted values (y-axis) against the true values (x-axis) of the target variable after applying Bayesian linear regression. The red dashed line represents the line of perfect prediction, where the predicted values would equal the true values.

The spread of points around the red line in Figure 1 indicates the accuracy of the predictions. Ideally, the points should align closely along the perfect fit line if the model accurately predicts the true values.

Bayesian Linear Regression Predictions with Uncertainty

This plot (Figure 2) displays a subset of predicted values along with their 95% credible intervals, represented by vertical lines around each prediction point. True values are shown in red for comparison. The credible intervals indicate the model's uncertainty for each prediction. Larger intervals show higher uncertainty, while smaller intervals suggest more confidence in the prediction.

6 Analysis and Conclusion

The following analysis compares the Bayesian Linear Regression model with traditional OLS, Lasso, and Ridge models across several metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2). Table 4 summarizes the performance of each model.

Performance Comparison

- **Mean Absolute Error (MAE):** The Bayesian Linear Regression model achieves a lower MAE (30960.95) than the OLS, Lasso, and Ridge models, all of which

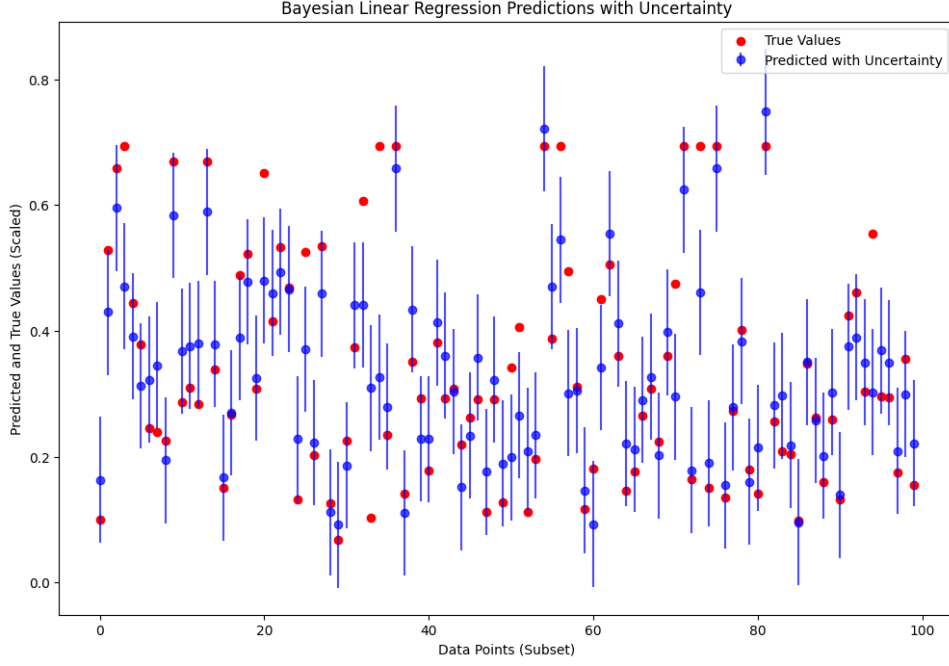


Figure 2: Bayesian Linear Regression Predictions with Uncertainty

Model	MAE	MSE	RMSE	R ²
Lasso	50500.43	4784523113.39	69170.25	0.6466
Ridge	50491.27	4783140549.35	69160.25	0.6467
OLS	50493.54	4782679151.64	69156.92	0.6467
Bayesian Linear Regression	30960.9	1787904818.8	42283.6	0.7183

Table 4: Performance Comparison of Bayesian Linear Regression, OLS, Lasso, and Ridge Models

have MAE values around 50,500. A lower MAE indicates that the Bayesian model predictions align more closely with true values.

- **Mean Squared Error (MSE):** The MSE for Bayesian Linear Regression is 1787904818.8. This reduction highlights the Bayesian model’s strength in minimizing larger prediction errors.
- **Root Mean Squared Error (RMSE):** The Bayesian model’s RMSE (42283.6) is also lower than that of the OLS, Lasso, and Ridge models (all around 69,160).
- **R-squared (R²):** The Bayesian model achieves a higher R² value of 0.7183, compared to around 0.6467 for the OLS, Lasso, and Ridge models. This indicates that approximately 71.83% of the variance in the target variable is explained by the Bayesian model, making it a superior fit.

Conclusion

The results suggest that Bayesian Linear Regression generally outperforms traditional models in terms of error reduction and explained variance. The Bayesian model’s lower

MAE, MSE, and RMSE indicate greater predictive accuracy, while its higher R^2 shows it captures more variance in the data.

Consequently, Bayesian Linear Regression proves to be a more robust choice for this dataset compared to traditional regression techniques.