```r
optimal_train_prop <- 0.8  # Proportion of the dataset used for training.

# Set a random seed for reproducibility.
set.seed(123)

# Total number of samples.
n_total <- nrow(X)

# Randomly shuffle indices for splitting data.
indices <- sample(1:n_total)

# Calculate sizes of training, validation, and test sets.
n_train <- floor(optimal_train_prop * n_total)  # Number of training samples
n_remaining <- n_total - n_train                # Remaining samples for validation and test
n_val <- floor(n_remaining / 2)                 # Number of validation samples
n_test <- n_remaining - n_val                   # Number of test samples

# Split data into training, validation, and test sets.
train_indices <- indices[1:n_train]
val_indices <- indices[(n_train + 1):(n_train + n_val)]
test_indices <- indices[(n_train + n_val + 1):n_total]

# Create training set.
X_train <- X[train_indices, ]
y_train <- y[train_indices]

# Create validation set.
X_val <- X[val_indices, ]
y_val <- y[val_indices]

# Create test set.
X_test <- X[test_indices, ]
y_test <- y[test_indices]

#### Step 5: Loop Over Different Prior Variances (tau_squared) ####
# In this step, we evaluate model performance for each tau_squared value.
for (i in seq_along(tau_squared_values)) {
  tau_squared <- tau_squared_values[i]  # Current tau_squared value

  # Define prior distributions for model parameters.
  beta0 <- rep(0, ncol(X))           # Prior mean vector for regression coefficients
  V0 <- diag(tau_squared, ncol(X))  # Prior covariance matrix scaled by tau_squared
  a0 <- 0.01                  # Shape parameter for inverse-gamma prior on variance
  b0 <- 0.01                  # Scale parameter for inverse-gamma prior on variance

  # Set parameters for the Gibbs sampling algorithm.
  n_iter <- 5000 # Total number of iterations
  burn_in <- 1000 # Number of burn-in samples to discard

  # Fit the model and obtain posterior samples.
  results <- bayesian_linear_regression(
    X_train, y_train, beta0, V0, a0, b0,
    n_iter = n_iter, burn_in = burn_in
  )

  # Estimate regression coefficients using posterior means.
  beta_posterior_mean <- colMeans(results$beta_samples)  # Posterior mean estimates

  # Evaluate model performance on training data.
  # Predictions on training data
  y_pred_train <- X_train %*% beta_posterior_mean  # Predicted values for training data.
  rmse_train_hyper[i] <- sqrt(mean((y_train - y_pred_train)^2))  # RMSE for training data

  # Predictions on validation data
  y_pred_val <- X_val %*% beta_posterior_mean  # Predicted values for validation data
  rmse_val_hyper[i] <- sqrt(mean((y_val - y_pred_val)^2))  # Validation RMSE
}
#### Step 5: Combine RMSE Results into a Data Frame ####
# In this step , we organize RMSE results for further analysis and plotting.
rmse_hyper_results <- data.frame(
  TauSquared = tau_squared_values,      # TauSquared values tested
  RMSE_Train = rmse_train_hyper,        # RMSE values for training data
  RMSE_Validation = rmse_val_hyper      # RMSE values for validation data
)

# Print RMSE results
print(rmse_hyper_results)
```

```
##   TauSquared RMSE_Train RMSE_Validation
## 1      1e-02   7.005826        6.882651
## 2      1e-01   5.625432        5.281985
## 3      1e+00   4.958987        4.434398
## 4      1e+01   4.843736        4.307508
## 5      1e+02   4.679821        4.367329
## 6      1e+03   4.655981        4.444334
```

```r
# Identify the optimal TauSquared value based on validation RMSE
optimal_index <- which.min(rmse_val_hyper)          # Index of the minimum validation RMSE
optimal_tau_squared <- tau_squared_values[optimal_index]  # Optimal TauSquared value
optimal_rmse <- rmse_val_hyper[optimal_index]       # Minimum validation RMSE

# Output the optimal TauSquared value and corresponding validation RMSE
cat("Optimal TauSquared Value:", optimal_tau_squared, "\n")
```

```
## Optimal TauSquared Value: 10
```

```r
cat("Validation RMSE at Optimal TauSquared Value:", optimal_rmse, "\n")
```

```
## Validation RMSE at Optimal TauSquared Value: 4.307508
```

```r
# Load necessary packages
library(reshape2)
library(dplyr)
library(ggplot2)

# Melt the RMSE results into a long format
rmse_hyper_melted <- melt(
  rmse_hyper_results,
  id.vars = "TauSquared",
  variable.name = "Dataset",
  value.name = "RMSE"
)

# Convert the wide-format data frame into a long-format data frame.

# Modify the Dataset column for descriptive labels
rmse_hyper_melted$Dataset <- recode(
  rmse_hyper_melted$Dataset,              # The Dataset column in the melted data frame
  "RMSE_Train" = "Train RMSE (TauSquared)",
  "RMSE_Validation" = "Validation RMSE (TauSquared)"
)


# Plot RMSE vs. TauSquared for Training and Validation using ggplot2
tau_squared_rmse_plot <- ggplot(
  rmse_hyper_melted,
  aes(x = log10(TauSquared), y = RMSE, color = Dataset)
) +
  geom_point(size = 3) +
  geom_line() +
  labs(
    title = "Impact of Prior Variance (TauSquared) on RMSE for Bayesian Linear Regression",  # Plot title
    x = "Log10(TauSquared)",
    y = "RMSE",
    color = "Data Type"
  ) +
  theme_bw()


# Print the plot
print(tau_squared_rmse_plot)
```

section 5: Further Exploring Model Performance

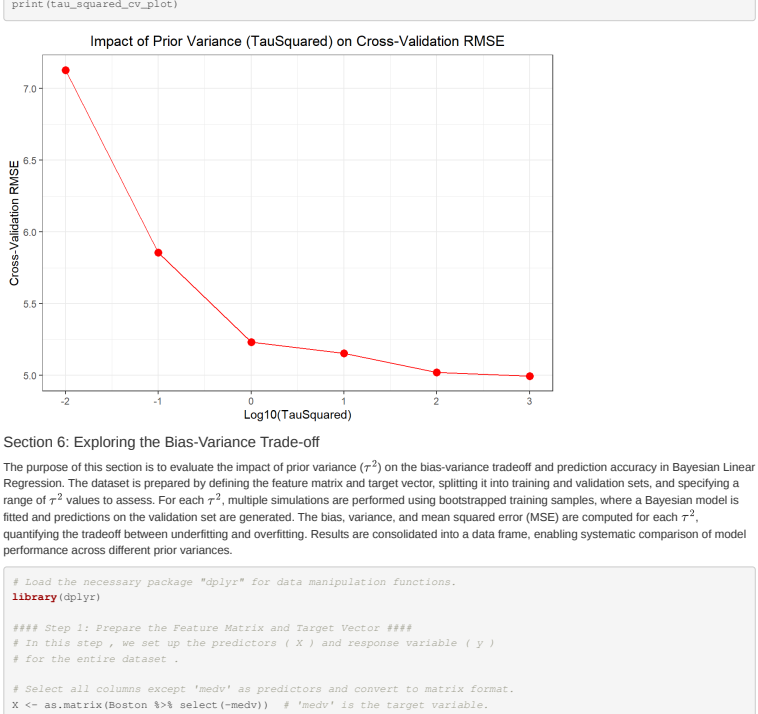
The purpose of this section is to evaluate the performance of a Bayesian Linear Regression model on a test set using an optimal prior variance ($\tau^2 = 10$) determined from prior analysis. The process begins by merging the training and validation sets to ensure the model is trained on all available data except the test set, enhancing robustness. The prior parameters, including a zero-initialized mean vector for β and a diagonal covariance matrix scaled by the optimal τ^2, are defined alongside inverse-gamma hyperparameters for σ^2. Gibbs sampling is then performed to estimate the posterior distributions of β and σ^2, with a burn-in period to discard initial samples and ensure convergence. The posterior mean of β is extracted to make predictions on the test set, and the model's accuracy is quantified using the Root Mean Square Error (RMSE), which reflects the average deviation of predicted values from the true test values. The test RMSE is finally printed to summarize the model's predictive performance.

```r
#### Step 1: Merge training and validation sets ####
# In this step, we create a combined dataset for model retraining.
# This ensures the model uses all available data (except the test set)
# for a more robust evaluation.

# Combine rows of training and validation predictors
X_train_full <- rbind(X_train, X_val)
# Combine responses from training and validation sets
y_train_full <- c(y_train, y_val)

#### Step 2: Set Prior Parameters with Optimal Tau Squared ####
# In this step, we define the prior distributions for the model parameters using # the optimal tau_squared value.

# Prior mean vector for regression coefficients (initialized to zeros).
beta0 <- rep(0, ncol(X))

# Prior covariance matrix for beta with optimal tau_squared = 10
V0 <- diag(10, ncol(X))

# Shape and scale parameters for the inverse-gamma prior on sigma^2.
a0 <- 0.01 # Shape parameter for inverse-gamma prior on sigma^2
b0 <- 0.01 # Scale parameter for inverse-gamma prior on sigma^2

#### Step 3: Set Gibbs Sampling Parameters ####
# In this step, we specify the number of iterations and burn-in period
# for the Gibbs sampler.

n_iter <- 5000 # Total number of iterations
burn_in <- 1000 # Number of burn-in iterations to discard

# Set a random seed for reproducibility
set.seed(123)

#### Step 4: Retrain the Model on the Combined Training and Validation Sets ####
# In this step, we fit the Bayesian Linear Regression model using
# the combined dataset.
results_final <- bayesian_linear_regression(
  X_train_full, y_train_full, beta0, V0, a0, b0,  # Input data and prior parameters
  n_iter = n_iter, burn_in = burn_in # Sampling parameters
)

#### Step 5: Obtain Posterior Mean of Beta ####
# We compute the posterior mean of the regression coefficients.
beta_posterior_mean_final <- colMeans(results_final$beta_samples)

#### Step 6: Predict on the Test Set ####
# We use the estimated coefficients to predict the response variable
# on the test set.
y_pred_test <- X_test %*% beta_posterior_mean_final

#### Step 7: Calculate RMSE on the Test Set ####
# In this setp, we compute the Root Mean Squared Error (RMSE) to
# evaluate model performance.
```

The purpose of this section is to evaluate the performance of different prior variances (τ^2) for Bayesian Linear Regression using k-fold cross-validation. The first setp is define the number of folds, and observations are randomly assigned to one of the k folds to ensure reproducibility. For each τ^2 value, the code iterates over the folds, splitting the data into training and validation sets, where one fold is used for validation and the remaining folds for training. The Bayesian model is trained on the training set using Gibbs sampling, and the posterior mean of β is used to predict the target variable on the validation set. The Root Mean Square Error (RMSE) is calculated for each fold to measure prediction accuracy. Finally, the mean RMSE across all folds is computed for each τ^2, providing a comprehensive evaluation of how prior variance affects model performance.

```r
#### Step 1: Set Up Cross-Validation Parameters ####
# In this step, we define the number of folds for k-fold cross-validation
k_folds <- 5  # Number of folds for cross-validation

# Set a random seed for reproducibility
set.seed(123)

#### Step 2: Create Fold Assignments ####
# In this step, we randomly assign each observation to one of the k folds.
n_total <- nrow(X)  # Total number of observations (ensure X is defined)
folds <- sample(rep(1:k_folds, length.out = n_total))  # Assign observations to folds

#### Step 3: Initialize Storage for RMSE Values ####
# In this step, we create a matrix to store RMSE values for
# each combination of tau_squared and fold.
tau_squared_values <- c(0.01, 0.1, 1, 10, 100, 1000)
rmse_cv <- matrix(0, nrow = length(tau_squared_values), ncol = k_folds)

# Notice that:
# - Rows correspond to different tau_squared values (hyperparameters).
# - Columns correspond to different folds in the cross-validation.

#### Step 4: Loop Over Tau Squared Values ####
# In this step, we evaluate model performance for each tau_squared value
# using cross-validation.
for (i in seq_along(tau_squared_values)) {
  tau_squared <- tau_squared_values[i]  # Select the current tau_squared value

  # Perform k-fold cross-validation for the current tau_squared value.
  for (k in 1:k_folds) {

    # Use fold 'k' as the validation set and the remaining folds
    # as the training set.
    test_indices_cv <- which(folds == k)     # Indices for the validation set
    train_indices_cv <- which(folds != k)    # Indices for the training set

    # Subset training and validation data
    X_train_cv <- X[train_indices_cv, ]     # Training predictors
    y_train_cv <- y[train_indices_cv]       # Training response variable

    X_val_cv <- X[test_indices_cv, ]        # Validation predictors
    y_val_cv <- y[test_indices_cv]          # Validation response variable

    # Define prior parameters for Bayesian Linear Regression.
    beta0 <- rep(0, ncol(X))                # Prior mean vector for beta
    V0 <- diag(tau_squared, ncol(X))        # Prior covariance matrix for beta
    a0 <- 0.01                # Shape parameter for inverse-gamma prior on sigma^2
    b0 <- 0.01                # Scale parameter for inverse-gamma prior on sigma^2

    # Notice that:
    # - The prior parameters reflect theinitial beliefs about the parameters.
    # - Varying tau_squared allows us to assess the impact of the prior variance.

    # Set parameters for the Gibbs sampling algorithm.
    n_iter <- 5000          # Total number of iterations (ensure n_iter is defined)
    burn_in <- 1000         # Number of burn-in samples (ensure burn_in is defined)

    # Notice that:
    # - These parameters control the convergence and accuracy of the Gibbs sampler
    # - Burn-in period allows the Markov chain to reach its stationary distribution

    # Fit the model using the training set for the current fold.
    results_cv <- bayesian_linear_regression(
      X_train_cv, y_train_cv, beta0, V0, a0, b0,
      n_iter = n_iter, burn_in = burn_in
    )

    # Notice that:
    # - The model is trained using Gibbs sampling to obtain posterior samples.
    # - Training is done on the training set specific to the current fold.

    # Obtain the posterior mean of the regression coefficients.
    beta_posterior_mean_cv <- colMeans(results_cv$beta_samples)

    # Notice that The posterior mean of beta serves as a point estimate for predictions,
    # it averages over the posterior distribution of beta.

    # Purpose: Use the posterior mean of beta to predict on the validation set.
    y_pred_cv <- X_val_cv %*% beta_posterior_mean_cv

    # Notice that: Predictions are made by applying the estimated coefficients to the validation data.

    # Evaluate the model's prediction accuracy on the validation set.
    rmse_cv[i, k] <- sqrt(mean((y_val_cv - y_pred_cv)^2))

    # Notice that:
    # - RMSE measures the average magnitude of prediction errors.
    # - Lower RMSE indicates better predictive performance on the validation set.

  }
}

#### Step 5: Compute Mean RMSE Across Folds ####
# In this step, we aggregate the RMSE across all folds for each tau_squared value.
rmse_cv_mean <- rowMeans(rmse_cv)  # Average RMSE across folds for each tau_squared value

# Notice that the averaging RMSE across folds provides a more stable estimate of
# model performance, it reduces variability due to different data splits.

#### Step 6: Compile Results into Data Frame ####
# In this step, Organize the results for further analysis.
data_cv <- data.frame(
  tau_squared = tau_squared_values,
  rmse = rmse_cv_mean
)
```

```r
# Load the package 'ggplot2'
library(ggplot2)

# Plot Cross-Validation RMSE vs. TauSquared
tau_squared_cv_plot <- ggplot(data_cv, aes(x = log10(tau_squared), y = rmse)) +
  geom_point(size = 3, color = "red") +
  geom_line(color = "red") +
  labs(
    title = "Impact of Prior Variance (TauSquared) on Cross-Validation RMSE",
    x = "Log10(TauSquared)",
    y = "Cross-Validation RMSE"
  ) +
  theme_bw() +
  theme(
    plot.title = element_text(hjust = 0.5)
  )

print(tau_squared_cv_plot)
```



Section 6: Exploring the Bias-Variance Trade-off

The purpose of this section is to evaluate the impact of prior variance (τ^2) on the bias-variance tradeoff and prediction accuracy in Bayesian Linear Regression. The dataset is prepared by defining the feature matrix and target vector, splitting it into training and validation sets, and specifying a range of τ^2 values to assess. For each τ^2, multiple simulations are performed using bootstrapped training samples, where a Bayesian model is fitted and predictions on the validation set are generated. The bias, variance, and mean squared error (MSE) are computed for each τ^2, quantifying the tradeoff between underfitting and overfitting. Results are consolidated into a data frame, enabling systematic comparison of model performance across different prior variances.

```r
# Load the necessary package "dplyr" for data manipulation functions.
library(dplyr)

#### Step 1: Prepare the Feature Matrix and Target Vector ####
# In this step , we set up the predictors ( X ) and response variable ( y )
# for the entire dataset .

# Select all columns except 'medv' as predictors and convert to matrix format.
X <- as.matrix(Boston %>% select(-medv))  # 'medv' is the target variable.
# Set the response variable as the 'medv' column.
y <- Boston$medv  # Target variable (median house values).
# Add an intercept term to the feature matrix.
X <- cbind(Intercept = 1, X)  # Adds a column of ones for the intercept term.

#### Step 2: Split Data into Training and Validation Sets ####
# In this step, we divide the dataset into training and validation sets based on the optimal training proportion.

# Set the optimal training set proportion 80%.
optimal_train_prop <- 0.8  # 80% of the data used for training.

# Set a random seed for reproducibility.
set.seed(123)

# Total number of observations.
n_total <- nrow(X)

# Randomly shuffle indices to randomize data splitting.
indices <- sample(1:n_total)

# Calculate the number of training and validation samples.
n_train <- floor(optimal_train_prop * n_total)  # Number of training samples.
n_val <- n_total - n_train                      # Number of validation samples.

# Split indices for training and validation sets.
train_indices <- indices[1:n_train]             # Indices for training set.
val_indices <- indices[(n_train + 1):n_total]   # Indices for validation set.

# Create validation set (training set will be sampled during simulations).
X_val <- X[val_indices, ]                       # Validation predictors.
y_val <- y[val_indices]                         # Validation target values.

#### Step 4: Define Tau Squared Values ####
# In this step, we specify different prior variances (tau_squared)
# to evaluate their impact on the model.
tau_squared_values <- c(0.01, 0.1, 1, 10, 100, 1000)

# Initialize vectors to store bias, variance, and MSE for each tau_squared value.
bias_list <- numeric(length(tau_squared_values))
variance_list <- numeric(length(tau_squared_values))
mse_list <- numeric(length(tau_squared_values))

#### Step 5: Set Number of Simulations ####
# In this step, we define the number of simulations to perform for each tau_squared value.
n_simulations <- 50

#### Step 6: Evaluate Bias, Variance, and MSE for Each Tau Squared Value ####
# Loop over tau_squared values to compute bias, variance, and MSE.
for (i in seq_along(tau_squared_values)) {
  tau_squared <- tau_squared_values[i]  # Current tau_squared value.

  # Create a matrix to store predictions from each simulation.
  predictions_matrix <- matrix(0, nrow = nrow(X_val), ncol = n_simulations)

  # Estimate the model multiple times to compute bias and variance.
  for (sim in 1:n_simulations) {
    # Randomly sample training data with replacement (bootstrap sampling).
    train_sample_indices <- sample(train_indices, size = n_train, replace = TRUE)
    X_train <- X[train_sample_indices, ]  # Bootstrapped training predictors.
    y_train <- y[train_sample_indices]    # Bootstrapped training target values.

    # Purpose: Set prior distributions for Bayesian Linear Regression.
    beta0 <- rep(0, ncol(X))                  # Prior mean vector for beta.
    V0 <- diag(tau_squared, ncol(X))          # Prior covariance matrix for beta.
    a0 <- 0.01                                # Shape parameter for inverse-gamma prior on sigma^2.
    b0 <- 0.01                                # Scale parameter for inverse-gamma prior on sigma^2.

    # Set parameters for the Gibbs sampling algorithm.
    n_iter <- 2000                            # Total number of iterations.
    burn_in <- 500                            # Number of burn-in iterations to discard.

    # Train the model using the bootstrapped training data.
    results <- bayesian_linear_regression(
      X_train, y_train, beta0, V0, a0, b0,
      n_iter = n_iter, burn_in = burn_in
    )

    # Obtain point estimates of the regression coefficients.
    beta_posterior_mean <- colMeans(results$beta_samples)

    # Use the estimated coefficients to predict on the validation set.
    y_pred_val <- X_val %*% beta_posterior_mean
    predictions_matrix[, sim] <- y_pred_val
  }

  # Calculate performance metrics based on the predictions:

  # Mean prediction for each validation sample across all simulations.
  y_pred_mean <- rowMeans(predictions_matrix)

  # Calculate the squared bias.
  bias_squared <- mean((y_pred_mean - y_val)^2)
  bias_list[i] <- bias_squared  # Store bias for the current tau_squared.

  # Calculate the variance of the predictions.
  variance <- mean(apply(predictions_matrix, 1, var))
  variance_list[i] <- variance  # Store variance for the current tau_squared.

  # Calculate the Mean Squared Error (MSE).
  mse <- bias_squared + variance
  mse_list[i] <- mse  # Store MSE for the current tau_squared.
}

#### Step 7: Compile Results into Data Frame ####
# In this step, we organize the bias, variance, and MSE results
# for further analysis.
bias_variance_df <- data.frame(
  TauSquared = tau_squared_values,
  BiasSquared = bias_list,
  Variance = variance_list,
  MSE = mse_list
)
```

```r
# Load necessary packages
library(reshape2)
library(ggplot2)

# Reshape the data frame for plotting
# Convert the wide-format data frame (bias_variance_df) into a long-format data frame
# with 'TauSquared' as the identifier, and 'Component' and 'Value' as columns.
bias_variance_melted <- melt(bias_variance_df, id.vars = 'TauSquared', variable.name = 'Component', value.name = 'Value')

# Plot the bias_variance_plot
bias_variance_plot <- ggplot(bias_variance_melted, aes(x = log10(TauSquared), y = Value, color = Component, linetype = Component, shape = Component)) +
  geom_line() +
  geom_point(size = 2, show.legend = FALSE) +
  theme_bw() +
  ggtitle("Bias-Variance Trade-off in Bayesian Linear Regression") +
  xlab("Log10(TauSquared)") +
  ylab("Metric Value") +
  scale_color_manual(
    values = c("BiasSquared" = "blue",
               "Variance" = "forestgreen",
               "MSE" = "firebrick"),
    breaks = c("MSE", "BiasSquared", "Variance"),
    labels = c("Bias^2", "Variance", "MSE")
  ) +
  scale_linetype_manual(
    values = c("BiasSquared" = "longdash",
               "Variance" = "dashed",
               "MSE" = "solid"),
    breaks = c("MSE", "BiasSquared", "Variance"),
    labels = c("Bias^2", "Variance", "MSE")
  ) +
  scale_shape_manual(
    values = c("BiasSquared" = 16,
               "Variance" = 17,
               "MSE" = 15),
    labels = c("Bias^2", "Variance", "MSE"),
    breaks = c("MSE", "BiasSquared", "Variance"),
  ) +
  labs(
    color = "Performance Metrics",
    linetype = "Performance Metrics",
    shape = "Performance Metrics"
  ) +
  theme(
    legend.position = "right",
    legend.text = element_text(size = 11)
  )

print(bias_variance_plot)
```