

# Slide 11 - R Output

Elvan Ceyhan

11/14/2024

```
#Set Working Directory to Source File Location  
library("rstudioapi") # Load rstudioapi package  
#setwd(dirname(getActiveDocumentContext()$path)) # Set working directory to source file location  
#getwd()
```

Some required packages:

```
# Load packages  
library(rstan)  
library(TeachingDemos) # loading TeachingDemos package, to use hpd function  
library(pscl) # loading pscl package, to use inverse gamma distribution  
library(mvtnorm) # for multivariate normal simulations  
library(ggplot2)
```

## Example 1, Oxygen Uptake Data

```
# Data from Hoff (2009)  
y <- c(-0.87, -10.74, -3.27, -1.97, 7.50, -7.25, 17.05, 4.96, 10.40, 11.05, 0.26, 2.51)  
# y = change in maximal oxygen uptake (positive number = improvement)  
x1 <- c(0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1) # 0 = running group, 1 = aerobics group  
x2 <- c(23, 22, 22, 25, 27, 20, 31, 23, 27, 28, 22, 24) # ages of subjects  
x3 <- x1 * x2 # interaction term between group and age  
  
# Design matrix with an intercept term  
X <- cbind(1, x1, x2, x3)  
  
# Function to calculate posterior median and credible intervals for regression parameters  
t_CI_table <- function(coefs, cov_mat, level=0.95, degrees=Inf, quantiles=c(0.025, 0.50, 0.975)) {  
  # Compute standard errors  
  std_errors <- sqrt(diag(cov_mat))  
  
  # Generate quantile-based intervals by repeating coefs to match the shape of quantile_matrix  
  quantile_matrix <- outer(std_errors, qt(quantiles, degrees), "*")  
  quantile_matrix <- sweep(quantile_matrix, 1, coefs, "+")  
  
  # Combine mean, standard error, and quantiles into a result matrix  
  result <- cbind(Mean=coefs, Std_Error=std_errors, quantile_matrix)
```

```

# Set column names for quantiles
quantile_names <- paste0(quantiles * 100, "%")
colnames(result) <- c("Mean", "Std_Error", quantile_names)

return(round(result, 4))
}

### Noninformative Prior Analysis:
# Estimate regression coefficients (bhat) and residual variance (sig2hat)
XtX_inv <- solve(t(X) %*% X) # Store result to avoid recalculating
bhat <- XtX_inv %*% t(X) %*% y
residuals <- y - X %*% bhat
sig2hat <- crossprod(residuals) / (length(y) - ncol(X))

# Covariance matrix of the posterior for beta
my_Cov <- XtX_inv * as.numeric((length(y) - ncol(X)) * sig2hat / (length(y) - ncol(X) - 2))

# Posterior information about the coefficients (betas)
post_beta <- t_CI_table(bhat, my_Cov, degrees=length(y) - ncol(X))
rownames(post_beta) <- paste0("beta_", 0:(ncol(X) - 1))
print(post_beta)

```

```

##           Mean Std_Error      2.5%      50%      97.5%
## beta_0 -51.2939   14.1476 -83.9185 -51.2939 -18.6694
## beta_1  13.1071   18.2004 -28.8630  13.1071  55.0772
## beta_2   2.0947    0.6078  0.6931   2.0947   3.4963
## beta_3  -0.3182    0.7503 -2.0485  -0.3182   1.4120

```

```

# Posterior information about sigma^2 using inverse gamma parameters
alpha_post <- (length(y) - ncol(X) - 1) / 2
beta_post <- 0.5 * sig2hat * (length(y) - ncol(X))

# Posterior median and HPD for sigma^2
# Using `qgamma` with parameters converted to gamma distribution parameters if `qigamma` is unavailable
sig2_post_med <- qgamma(0.50, shape=alpha_post, rate=beta_post)
cat("Posterior median of sigma^2:", round(sig2_post_med, 3), "\n")

```

```
## Posterior median of sigma^2: 0.093
```

```

# Calculate HPD interval for sigma^2 using the gamma distribution
hpd_sig2 <- qgamma(c(0.025, 0.975), shape=alpha_post, rate=beta_post)
cat("HPD interval for sigma^2:", round(hpd_sig2, 3), "\n")

```

```
## HPD interval for sigma^2: 0.025 0.234
```

## Example 2, Automobile Data

```

# Load data
auto_data <- read.table("../STAT7630-Data/autoregresslarge.txt", header = TRUE)

```

```

y <- auto_data$mpg
X <- cbind(1, auto_data$displacement, auto_data$horsepower, auto_data$weight)

##### Setting up prior specification:
# 3 predictor variables, so we set up exactly 4 hypothetical "prior observations"
# Based on "expert opinion," we have the following estimates:

# Prior beliefs about mpg for specific car configurations
prior_data <- list(
  xtil = matrix(c(
    150, 100, 2000, # Observation 1
    200, 160, 4500, # Observation 2
    250, 140, 3000, # Observation 3
    100, 80, 1800   # Observation 4
  ), ncol=3, byrow=TRUE),
  ytil = c(25, 10, 20, 35)
)

#That is, prior beliefs or expert opinions:
#a car with displacement=150, horsepower=100, weight=2000 should have mpg around 25
#a car with displacement=200, horsepower=160, weight=4500 should have mpg around 10
#a car with displacement=250, horsepower=140, weight=3000 should have mpg around 20
#a car with displacement=100, horsepower=80, weight=1800 should have mpg around 35

# Include intercept in prior design matrix
xtil <- cbind(1, prior_data$xtil)

# Prior weights (D) for the prior observations, indicating confidence level
D <- diag(4, 4) # Equal weight (4) on each prior observation

# Prior mean on the beta vector induced from prior observations
pri_mean_beta <- as.vector(solve(t(xtil) %*% xtil) %*% t(xtil) %*% prior_data$ytil)
cat("Prior mean for beta vector:", round(pri_mean_beta, 3), "\n")

## Prior mean for beta vector: 82.5 0.325 -1.563 0.025

### Choosing prior parameters a and b for the gamma prior on tau:
# The parameter "a" reflects the confidence level in prior knowledge about tau.
a <- 0.5 # Sets prior knowledge to be equivalent to 1 sample observation

# Prior guess for tau, based on an expert's estimate of realistic mpg range
tau_guess <- 0.027 # Estimated based on  $(35 - 25)^2 / 1.645^2 = 36.97$ , thus  $\tau = 1/36.97$ 
b <- a / tau_guess # Set b such that prior mean of tau equals tau_guess
cat("Value of b based on tau guess:", round(b, 2), "\n")

## Value of b based on tau guess: 18.52

##### Posterior Information for tau and beta:

# Calculate posterior mean of beta (beta_hat) using the combined information from data and prior
XtX_inv <- solve(t(X) %*% X + t(xtil) %*% solve(D) %*% xtil) # Invert only once for efficiency
beta_hat <- XtX_inv %*% (t(X) %*% y + t(xtil) %*% solve(D) %*% prior_data$ytil)
cat("Posterior mean for beta:", round(beta_hat, 3), "\n")

```

```
## Posterior mean for beta: 43.204 -0.009 -0.035 -0.005
```

```
# Calculate posterior for s* (estimate of residual variance):
n <- length(y)
residual_data <- t(y - X %>% beta_hat) %>% (y - X %>% beta_hat)
residual_prior <- t(prior_data$ytil - xtil %>% beta_hat) %>% solve(D) %>% (prior_data$ytil - xtil %>% beta_hat)
sstar <- as.numeric((residual_data + residual_prior + 2 * b) / (n + 2 * a))

### Posterior Point Estimates for tau and sigma^2:
p_mean_tau <- 1 / sstar
p_mean_sig2 <- 1 / p_mean_tau
p_med_tau <- qgamma(0.50, shape=(n + 2 * a) / 2, rate=((n + 2 * a) / 2) * sstar)
p_med_sig2 <- 1 / p_med_tau

cat("Posterior mean for sigma^2:", round(p_mean_sig2, 3), "\n",
    "Posterior median for sigma^2:", round(p_med_sig2, 3), "\n")
```

```
## Posterior mean for sigma^2: 16.346
## Posterior median for sigma^2: 16.376
```

```
### 95% Marginal Interval Estimate for tau (and sigma^2):
hpd95_tau <- hpd(qgamma, shape=(n + 2 * a) / 2, rate=((n + 2 * a) / 2) * sstar)
hpd95_sig2 <- 1 / hpd95_tau

cat("95% Posterior Interval for sigma^2:", round(sort(hpd95_sig2), 3), "\n")
```

```
## 95% Posterior Interval for sigma^2: 14.259 19.08
```

```
### Point estimates for beta, given the point estimate for tau:
### NOT QUITE THE BEST APPROACH:
# - The code uses a single point estimate (the posterior median `p_med_tau`) for  $\tau$ 
# when calculating the posterior intervals for  $\beta$ .
# However, Bayesian inference typically involves integrating over the posterior distribution of  $\tau$ 
# not just using a single point estimate.
# - By using only the posterior median of  $\tau$ , the intervals for  $\beta$  do not fully
# reflect the uncertainty in  $\tau$ .
# This can lead to underestimation of the true posterior variance of  $\beta$ .
# Posterior mean, median and 95% intervals for the betas:
post_beta <- t_CI_table(beta_hat, (1/p_med_tau)*solve(t(X)%*%X + t(xtil)%*%solve(D)%*%xtil) )
# Set row names to beta_0, beta_1, beta_2, beta_3
rownames(post_beta) <- paste0("beta_", 0:3)
post_beta
```

```
##           Mean Std_Error   2.5%   50%   97.5%
## beta_0 43.2038   1.1646 40.9213 43.2038 45.4863
## beta_1 -0.0087   0.0064 -0.0211 -0.0087  0.0038
## beta_2 -0.0354   0.0123 -0.0595 -0.0354 -0.0112
## beta_3 -0.0050   0.0007 -0.0063 -0.0050 -0.0036
```

```
### A BETTER APPROACH TO GET POINT ESTIMATES FOR BETA:
# Generate posterior samples for tau and beta with 95% credible intervals
```

```

# Number of posterior samples to generate
Nsim <- 10000

# Generate posterior samples for tau based on its gamma distribution
# This reflects the uncertainty in tau based on observed data
tau_vals <- rgamma(Nsim, shape = (n + 2 * a) / 2, rate = ((n + 2 * a) / 2) * sstar)

# Generate posterior samples for beta based on tau samples
# For each sampled tau value, we sample beta from a multivariate normal distribution
# The distribution of beta depends on tau through its covariance matrix
covariance_matrix <- solve(t(X) %*% X + t(xtil) %*% solve(D) %*% xtil) # Compute once for efficiency
beta_vals <- t(apply(tau_vals, function(tau) {
  rmvnorm(1, mean = beta_hat, sigma = covariance_matrix / tau)
})))

# Calculate posterior summaries for beta: median and 95% credible interval bounds
# `apply` is used to obtain column-wise medians and quantiles across the samples
post_meds <- apply(beta_vals, 2, median) # Posterior median for each beta
post_low <- apply(beta_vals, 2, quantile, probs = 0.025) # Lower bound of 95% credible interval
post_up <- apply(beta_vals, 2, quantile, probs = 0.975) # Upper bound of 95% credible interval

# Summarizing posterior estimates for beta in a readable format
names_preds <- c("intercept", "displacement", "horsepower", "weight") # Predictor names
beta_post_summary <- data.frame(
  `.025Quantile` = post_low,
  `.5Quantile` = post_meds,
  `.975Quantile` = post_up,
  row.names = names_preds
)

# Display posterior summary table for beta coefficients
print(beta_post_summary)

```

```

##           X.025Quantile  X.5Quantile X.975Quantile
## intercept      40.94422804 43.183345476 45.472483654
## displacement  -0.02100904 -0.008745797  0.003689806
## horsepower    -0.05968009 -0.034888546 -0.011215164
## weight        -0.00630296 -0.004942381 -0.003641582

```

```

# Load another dataset to compare results: A smaller built-in R dataset for reference
# Load mpg (response) and key predictors from mtcars dataset for illustration
y <- mtcars$mpg
x1 <- mtcars$disp
x2 <- mtcars$hp
x3 <- mtcars$wt

# Redo analysis with both types of priors...

```

## Bayesian Regression with rstanarm

```
# Load necessary libraries
library(rstanarm)          # For Bayesian regression modeling

## Loading required package: Rcpp

## This is rstanarm version 2.32.1

## - See https://mc-stan.org/rstanarm/articles/priors for changes to default priors!

## - Default priors may change, so it's safest to specify priors, even if equivalent to the defaults.

## - For execution on a local, multicore CPU with excess RAM we recommend calling

##   options(mc.cores = parallel::detectCores())

##
## Attaching package: 'rstanarm'

## The following object is masked from 'package:rstan':
##
##   loo

library(tidyverse)        # For data manipulation and visualization

## -- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
## v dplyr      1.1.4      v readr      2.1.5
## v forcats    1.0.0      v stringr   1.5.1
## v lubridate  1.9.3      v tibble    3.2.1
## v purrr      1.0.2      v tidyr     1.3.1

## -- Conflicts ----- tidyverse_conflicts() --
## x tidyr::extract() masks rstan::extract()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

library(bayesplot)        # For visualizing MCMC diagnostics

## This is bayesplot version 1.11.1
## - Online documentation and vignettes at mc-stan.org/bayesplot
## - bayesplot theme set to bayesplot::theme_default()
##   * Does _not_ affect other ggplot2 plots
##   * See ?bayesplot_theme_set for details on theme setting
```

```

library(broom.mixed)    # For summarizing Bayesian model output
library(tidybayes)      # For handling Bayesian models in tidy format

# Notes on Prior Specification:
# The first value in `prior_intercept` (22 here) represents the expected mpg
# for a "typical" observation (e.g., a car with average characteristics).
# `prior` specifies weakly informative priors for predictors.

# Define the Bayesian linear regression model with rstanarm
car_mod <- stan_glm(
  mpg ~ displacement + horsepower + weight,
  data = auto_data,
  family = gaussian, # Gaussian family for linear regression
  prior_intercept = normal(22, 20), # Prior for intercept, centered at 22
  prior = normal(c(0.325, -1.563, 0.025), 40, autoscale = TRUE), # Priors for predictors
  prior_aux = exponential(1, autoscale = TRUE), # Prior for auxiliary (error) parameter
  chains = 4, # Number of MCMC chains
  iter = 10000 # Number of iterations per chain
)

```

```

##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 9.5e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.95 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [  0%] (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 1: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 1: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 1: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 1: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 1:
## Chain 1: Elapsed Time: 0.924 seconds (Warm-up)
## Chain 1:           1.016 seconds (Sampling)
## Chain 1:           1.94 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.3e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.13 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:

```

```

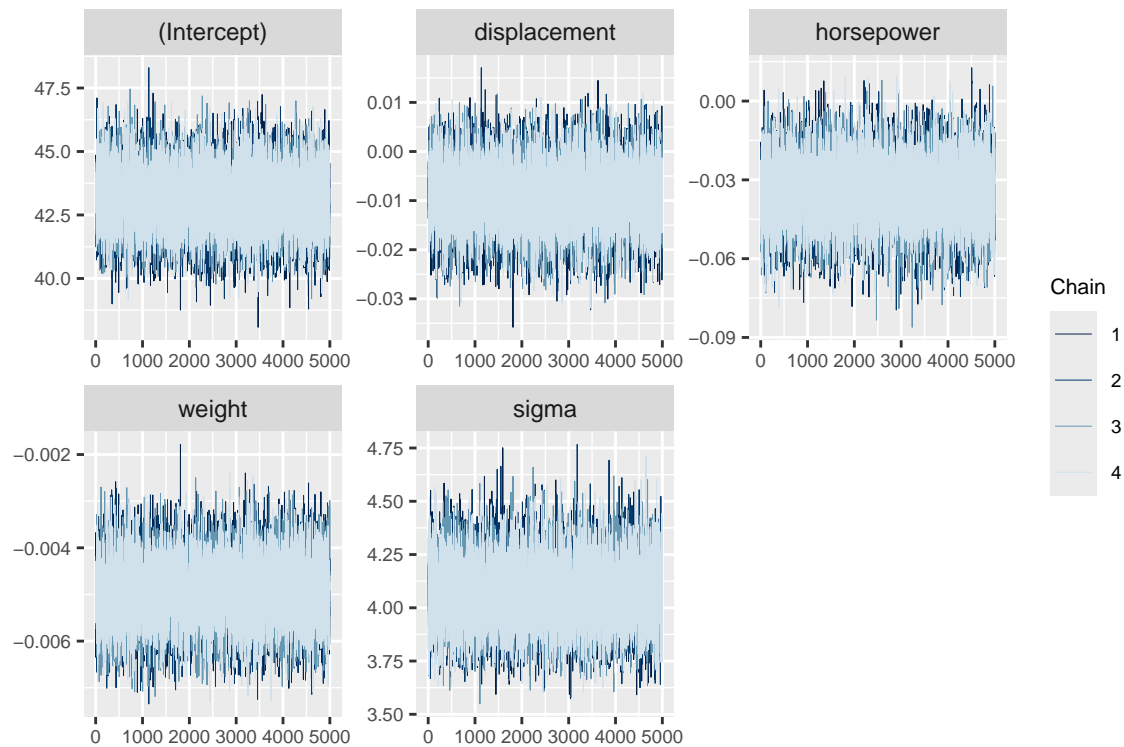
## Chain 2: Iteration:    1 / 10000 [ 0%] (Warmup)
## Chain 2: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 2: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 2: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 2: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 2: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 2: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 2: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 2: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 2: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 2: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 2: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 2:
## Chain 2: Elapsed Time: 0.99 seconds (Warm-up)
## Chain 2:                0.96 seconds (Sampling)
## Chain 2:                1.95 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 1.1e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.11 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 10000 [ 0%] (Warmup)
## Chain 3: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 3: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 3: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 3: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 3: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 3: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 3: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 3: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 3: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 3: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 3: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 3:
## Chain 3: Elapsed Time: 0.993 seconds (Warm-up)
## Chain 3:                1.048 seconds (Sampling)
## Chain 3:                2.041 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.1 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 10000 [ 0%] (Warmup)
## Chain 4: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 4: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 4: Iteration: 3000 / 10000 [ 30%] (Warmup)

```

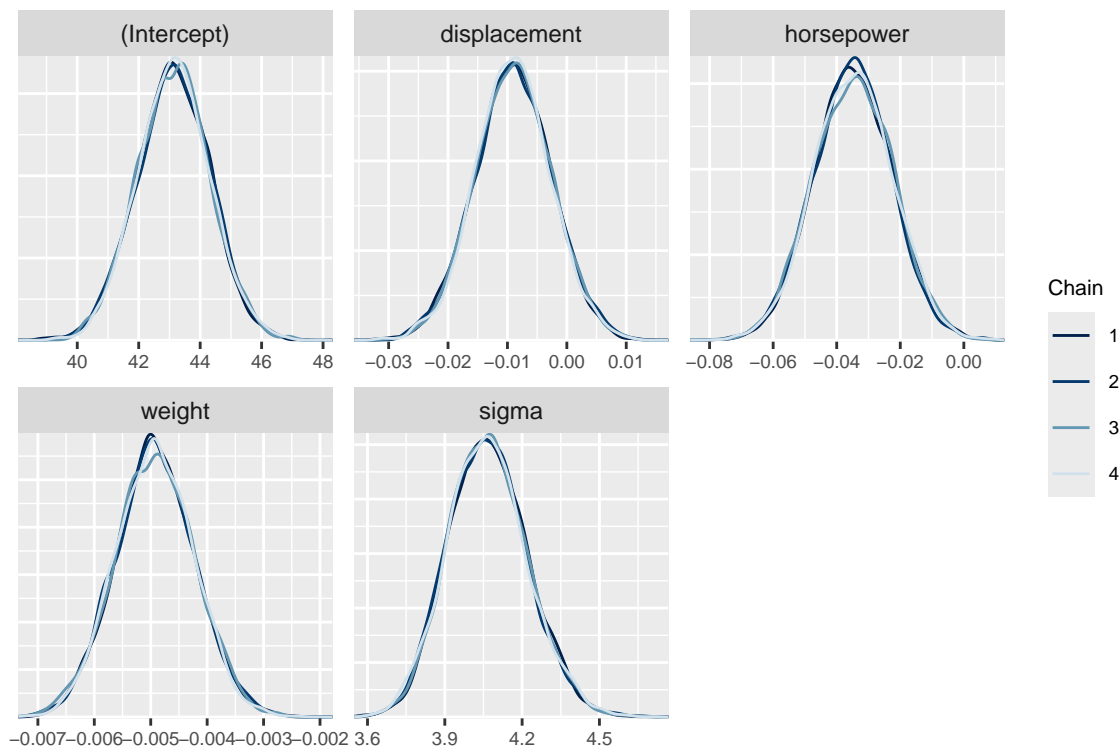


```
## Chain 4: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 4: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 4: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 4: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 4: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 4: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 4: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 4: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 4:
## Chain 4: Elapsed Time: 0.97 seconds (Warm-up)
## Chain 4:           0.973 seconds (Sampling)
## Chain 4:           1.943 seconds (Total)
## Chain 4:
```

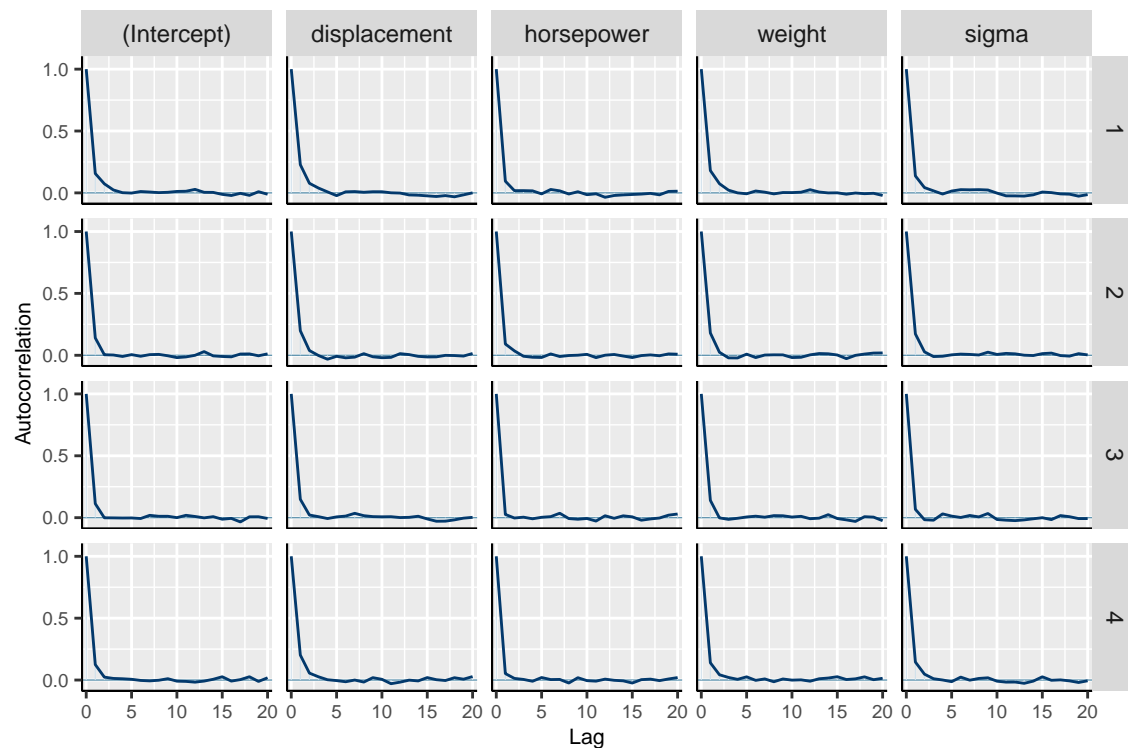
```
# MCMC Diagnostics to assess convergence and sampling quality
# Trace plot for checking mixing across chains
mcmc_trace(car_mod, size = 0.1)
```



```
# Density overlay plot to examine distribution of MCMC samples
mcmc_dens_overlay(car_mod)
```



```
# Autocorrelation plot to assess sampling independence
mcmc_acf(car_mod)
```



```
# Effective sample size ratio and R-hat statistic for convergence diagnostics
cat("Effective Sample Size Ratio:\n")
```

```
## Effective Sample Size Ratio:
```

```
print(neff_ratio(car_mod))
```

```
## (Intercept) displacement horsepower weight sigma
## 0.74235 0.65775 0.85280 0.71130 0.73605
```

```
cat("\nR-hat Diagnostics:\n")
```

```
##
```

```
## R-hat Diagnostics:
```

```
print(rhat(car_mod))
```

```
## (Intercept) displacement horsepower weight sigma
## 1.0000051 0.9999233 1.0000405 0.9999664 1.0000875
```

```
# Summarize the posterior distributions of parameters
# Provides median, 90% credible intervals, and auxiliary parameters
```

```
broom_summary <- broom.mixed::tidy(
  car_mod,
  effects = c("fixed", "aux"),
  conf.int = TRUE,
  conf.level = 0.90
)
```

```
cat("\nPosterior Summary with 90% Credible Intervals:\n")
```

```
##
```

```
## Posterior Summary with 90% Credible Intervals:
```

```
print(broom_summary)
```

```
## # A tibble: 6 x 5
## term estimate std.error conf.low conf.high
## <chr> <dbl> <dbl> <dbl> <dbl>
## 1 (Intercept) 43.1 1.19 41.2 45.1
## 2 displacement -0.00900 0.00638 -0.0196 0.00157
## 3 horsepower -0.0348 0.0125 -0.0551 -0.0145
## 4 weight -0.00493 0.000698 -0.00608 -0.00377
## 5 sigma 4.06 0.154 3.82 4.33
## 6 mean_PPD 22.8 0.301 22.3 23.3
```

```
# Detailed summary of posterior distributions with specific credible intervals
summary(car_mod, digits = 4, probs = c(0.025, 0.5, 0.975))
```

```
##
```

```
## Model Info:
```

```
## function: stan_glm
```

```

## family:      gaussian [identity]
## formula:     mpg ~ displacement + horsepower + weight
## algorithm:   sampling
## sample:      20000 (posterior sample size)
## priors:      see help('prior_summary')
## observations: 364
## predictors:  4
##
## Estimates:
##           mean      sd      2.5%      50%      97.5%
## (Intercept) 43.1331  1.1873 40.7951 43.1453 45.4576
## displacement -0.0090 0.0064 -0.0218 -0.0090  0.0036
## horsepower   -0.0348 0.0124 -0.0589 -0.0348 -0.0106
## weight       -0.0049 0.0007 -0.0063 -0.0049 -0.0035
## sigma        4.0676 0.1536  3.7841  4.0632  4.3840
##
## Fit Diagnostics:
##           mean      sd      2.5%      50%      97.5%
## mean_PPD 22.7639  0.3013 22.1687 22.7633 23.3476
##
## The mean_ppd is the sample average posterior predictive distribution of the outcome variable (for de
##
## MCMC diagnostics
##           mcse  Rhat  n_eff
## (Intercept) 0.0097 1.0000 14847
## displacement 0.0001 0.9999 13155
## horsepower    0.0001 1.0000 17056
## weight        0.0000 1.0000 14226
## sigma         0.0013 1.0001 14721
## mean_PPD      0.0023 0.9999 16662
## log-posterior 0.0175 1.0000  8147
##
## For each parameter, mcse is Monte Carlo standard error, n_eff is a crude measure of effective sample

# Classical regression model for comparison
freq_mod <- lm(mpg ~ displacement + horsepower + weight, data = auto_data)
cat("\nClassical (Frequentist) Regression Summary:\n")

##
## Classical (Frequentist) Regression Summary:

summary(freq_mod)

##
## Call:
## lm(formula = mpg ~ displacement + horsepower + weight, data = auto_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.9090  -2.6294  -0.4158   1.9631  16.8922
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)

```

```
## (Intercept) 43.1425283 1.1782187 36.617 < 2e-16 ***
## displacement -0.0089838 0.0064212 -1.399 0.16265
## horsepower -0.0349449 0.0123825 -2.822 0.00504 **
## weight -0.0049245 0.0006931 -7.105 6.51e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.054 on 360 degrees of freedom
## Multiple R-squared:  0.714, Adjusted R-squared:  0.7116
## F-statistic: 299.6 on 3 and 360 DF, p-value: < 2.2e-16
```

```
# Posterior Prediction for a New Observation
# Define new data for prediction (car with specific displacement, horsepower, and weight)
new_obs <- data.frame(displacement = 150, horsepower = 100, weight = 2.7)

# Generate posterior predictive samples
# This provides samples from the posterior predictive distribution for the new data point
shortcut_pred <- posterior_predict(car_mod, newdata = new_obs)

# Point Prediction - Calculate the median of the predictive distribution
point_pred <- median(shortcut_pred)
cat("Median of predictive distribution for new observation:", point_pred, "\n")
```

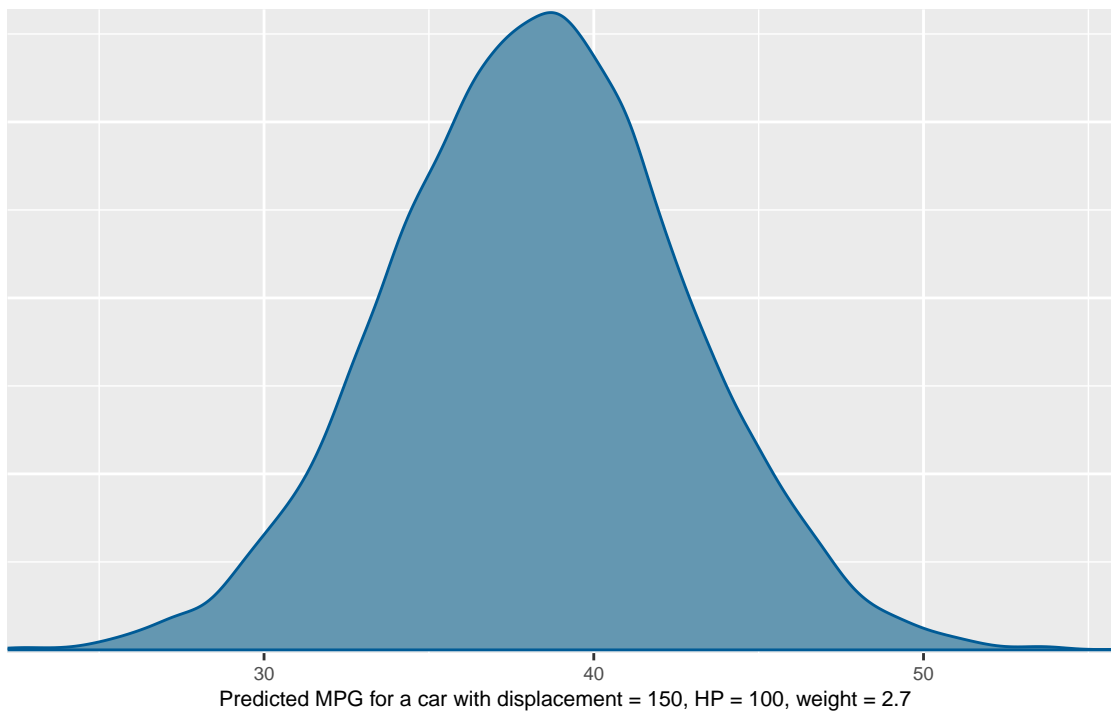
```
## Median of predictive distribution for new observation: 38.30484
```

```
# 95% Posterior Credible Interval for the Prediction
# Provides uncertainty bounds around the point prediction
cred_int <- quantile(shortcut_pred, probs = c(0.025, 0.975))
cat("95% posterior credible interval for prediction:", cred_int, "\n")
```

```
## 95% posterior credible interval for prediction: 29.53318 47.00301
```

```
# Plot the predictive distribution
# Visualizing the distribution of predictions for the new observation
mcmc_dens(shortcut_pred) +
  xlab("Predicted MPG for a car with displacement = 150, HP = 100, weight = 2.7") +
  ggtitle("Posterior Predictive Distribution for MPG")
```

Posterior Predictive Distribution for MPG



## A Bayesian Approach to Model Selection

```
# Load necessary libraries
library(MASS) # for matrix operations if needed

# Function to calculate the marginal probability of y given X using a Bayesian approach
# Inputs:
# y - response vector
# X - design matrix of predictors
# g - prior scaling parameter (default is length(y))
# nu0 - prior degrees of freedom
# s20 - prior guess for sigma^2 (default uses residual variance from linear model)
log_Py_x <- function(y, X, g = length(y), nu0 = 1,
                     s20 = try(summary(lm(y ~ -1 + X))$sigma^2, silent = TRUE)) {
  n <- nrow(X)
  p <- ncol(X)

  if (p == 0) {
    Hg <- 0
    s20 <- mean(y^2)
  } else {
    Hg <- (g / (g + 1)) * X %*% solve(t(X) %*% X) %*% t(X)
  }

  SSRg <- t(y) %*% (diag(1, nrow = n) - Hg) %*% y

  -.5 * (n * log(pi) + p * log(1 + g) + (nu0 + n) * log(nu0 * s20 + SSRg) - nu0 * log(nu0 * s20)) +
```

```

    lgamma((nu0 + n) / 2) - lgamma(nu0 / 2)
}

# Oxygen uptake data example
y <- c(-0.87, -10.74, -3.27, -1.97, 7.50, -7.25, 17.05, 4.96, 10.40, 11.05, 0.26, 2.51)
x1 <- c(0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1) # indicator for group (0 = running, 1 = aerobics)
x2 <- c(23, 22, 22, 25, 27, 20, 31, 23, 27, 28, 22, 24) # ages of subjects
x3 <- x1 * x2 # interaction term between group and age
X <- cbind(1, x1, x2, x3) # design matrix with intercept and predictors

### Initialize Gibbs Sampler for Bayesian Model Selection
z <- rep(1, ncol(X)) # start with all predictors in the model
log_Py_c <- log_Py_x(y, X[, z == 1, drop = FALSE]) # initial log-probability of y given X with all predictors
S <- 10000 # number of Gibbs sampler iterations
Z <- matrix(NA, nrow = S, ncol = ncol(X)) # storage matrix for sampled model indicators

### Gibbs Sampling Process for Model Selection
for (s in 1:S) {
  for (j in sample(seq_along(z))) { # randomly iterate over predictors
    zp <- z
    zp[j] <- 1 - zp[j] # toggle inclusion of predictor j
    lpy_p <- log_Py_x(y, X[, zp == 1, drop = FALSE]) # calculate log-marginal for proposed model

    # Calculate acceptance probability for the proposed model
    r <- (lpy_p - log_Py_c) * (-1)^(zp[j] == 0)
    z[j] <- rbinom(1, 1, 1 / (1 + exp(-r))) # accept/reject step based on acceptance probability
    if (z[j] == zp[j]) log_Py_c <- lpy_p # update log-probability if model changes
  }
  Z[s, ] <- z # store current model indicators
}

# Considering all possible subsets:
all_z_vecs <- unique(Z, MARGIN = 1) # unique models encountered in sampling

# Calculate the probability for each unique subset configuration
all_z_probs <- rowSums(apply(Z, 1, function(row) apply(all_z_vecs, 1, function(poss_row) all(row == poss_row)
sum(all_z_probs)

## [1] 1

# Combine model configurations and probabilities, then sort by probability
all_res <- cbind(all_z_vecs, all_z_probs)
all_res <- all_res[order(-all_z_probs), ]
print(all_res)

```

```

##           all_z_probs
## [1,] 1 1 1 0      0.3948
## [2,] 1 0 1 1      0.3329
## [3,] 1 1 1 1      0.1231
## [4,] 1 0 1 0      0.1155
## [5,] 0 1 0 1      0.0085
## [6,] 1 1 0 1      0.0071
## [7,] 0 1 1 1      0.0049

```

```
## [8,] 1 0 0 1      0.0033
## [9,] 0 0 0 1      0.0032
## [10,] 0 0 1 1     0.0022
## [11,] 0 1 0 0     0.0020
## [12,] 1 1 0 0     0.0009
## [13,] 0 0 0 0     0.0006
## [14,] 0 1 1 0     0.0006
## [15,] 0 0 1 0     0.0002
## [16,] 1 0 0 0     0.0002
```

```
# Constrained model selection: Include only models with interaction term if main effects are present
# Define constrained models manually
```

```
sub_z_vecs <- matrix(c(
  1, 0, 0, 0,
  1, 1, 0, 0,
  1, 0, 1, 0,
  1, 1, 1, 0,
  1, 1, 1, 1
), ncol = 4, byrow = TRUE)
```

```
# Calculate probabilities for each constrained model configuration
```

```
sub_z_probs <- rowSums(apply(Z, 1,
  function(row) apply(sub_z_vecs, 1, function(poss_row) all(row == poss_row))))
```

```
sub_z_probs = sub_z_probs / sum(sub_z_probs)
```

```
# Combine constrained results and sort by probability
```

```
sub_res <- cbind(sub_z_vecs, sub_z_probs)
sub_res <- sub_res[order(-sub_z_probs), ]
print(sub_res)
```

```
##          sub_z_probs
## [1,] 1 1 1 0 0.6222222222
## [2,] 1 1 1 1 0.1940110323
## [3,] 1 0 1 0 0.1820330969
## [4,] 1 1 0 0 0.0014184397
## [5,] 1 0 0 0 0.0003152088
```

```
# Small automobile data set analysis
```

```
# Using mtcars data for a similar analysis with different predictors
```

```
y <- mtcars$mpg
x1 <- mtcars$disp
x2 <- mtcars$hp
x3 <- mtcars$wt
X <- cbind(1, x1, x2, x3)
```

```
# Further analysis can be performed here for mtcars dataset using the same approach
```

## Posterior Predictive Distribution with Regression



```

# Load necessary libraries for multivariate t-distribution sampling
library(MASS) # for rmvt function

##### Regression example:
# Using the small automobile data set built into R
# Define response variable and predictors
y <- mtcars$mpg
X <- cbind(1, mtcars$disp, mtcars$hp, mtcars$wt) # Adding intercept directly

# Estimate regression coefficients (bhat) and residual variance (sig2hat)
XtX_inv <- solve(crossprod(X)) # Efficiently calculates (X'X)^-1
bhat <- XtX_inv %*% t(X) %*% y # Coefficients for predictors
sig2hat <- as.numeric(crossprod(y - X %*% bhat) / (nrow(X) - ncol(X))) # Residual variance estimate

# Set up posterior predictive sampling for observed X values
Xstar <- X
df <- nrow(X) - ncol(X) # Degrees of freedom for the t-distribution
sc_fact <- sig2hat * df / (df - 2) # Scale factor for predictive covariance

# Posterior predictive covariance matrix for observed X values
my_Sig <- sc_fact * (diag(nrow(Xstar)) + Xstar %*% XtX_inv %*% t(Xstar))

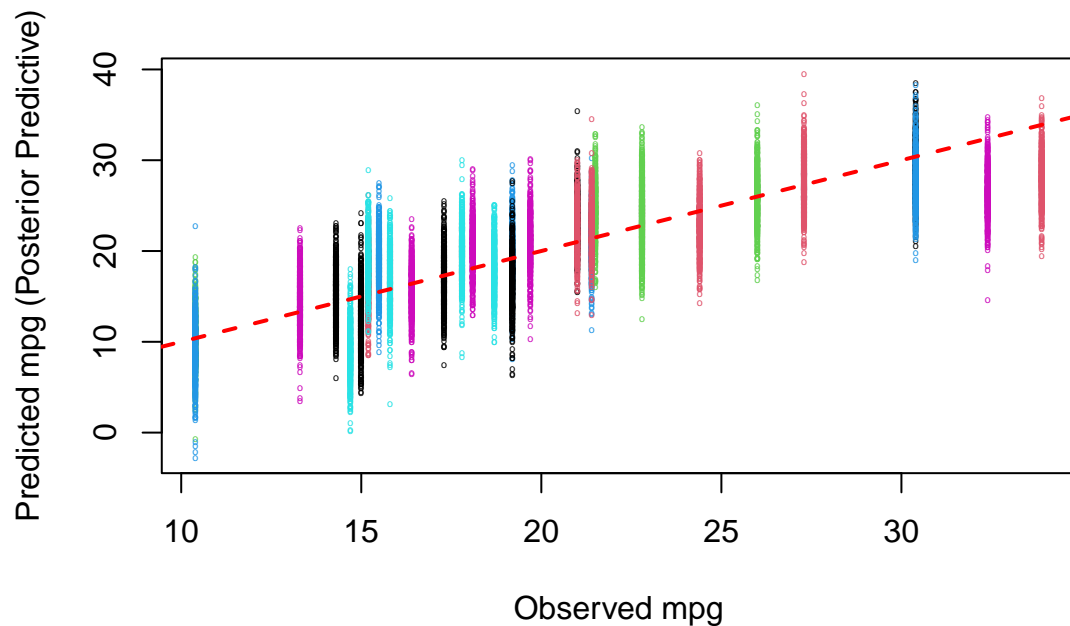
# Generate posterior predictive samples for observed X values
# Sampling from multivariate t-distribution centered at fitted values
post_pred_err <- rmvt(n = 500, sigma = my_Sig, df = df)
post_pred_samp <- matrix(Xstar %*% bhat, nrow = nrow(post_pred_err),
                        ncol = ncol(post_pred_err), byrow = TRUE) + post_pred_err

# Plot observed mpg against posterior predictive samples
mat_ys <- matrix(y, nrow = nrow(post_pred_samp), ncol = ncol(post_pred_samp), byrow = TRUE)

matplot(mat_ys, post_pred_samp, pch = 'o', cex = 0.3,
        xlab = "Observed mpg",
        ylab = "Predicted mpg (Posterior Predictive)",
        main = "Posterior Predictive Distribution of MPG")
abline(0, 1, col = "red", lty = 2, lwd = 2) # Add y=x reference line

```

## Posterior Predictive Distribution of MPG



```
# Examine model residuals to check for lack of fit
resids <- y - X %*% bhat
residual_analysis <- cbind(mtcars[, c("mpg", "disp")], Residuals = resids)[order(y), ]
print(residual_analysis)
```

	mpg	disp	Residuals
## Cadillac Fleetwood	10.4	472.0	0.0785315
## Lincoln Continental	10.4	460.0	1.0402079
## Camaro Z28	13.3	350.0	-1.2487773
## Duster 360	14.3	360.0	-1.2656477
## Chrysler Imperial	14.7	440.0	5.4885456
## Maserati Bora	15.0	301.0	2.1831584
## Merc 450SLC	15.2	275.8	-1.6715326
## AMC Javelin	15.2	304.0	-3.8911127
## Dodge Challenger	15.5	318.0	-3.2549189
## Ford Pantera L	15.8	351.0	-0.7024625
## Merc 450SE	16.4	275.8	0.6307257
## Merc 450SL	17.3	275.8	0.2384229
## Merc 280C	17.8	167.6	-2.2411432
## Valiant	18.1	225.0	-2.3721590
## Hornet Sportabout	18.7	360.0	0.4592780
## Merc 280	19.2	167.6	-0.8411432
## Pontiac Firebird	19.2	400.0	2.5361190
## Ferrari Dino	19.7	145.0	-1.2887756
## Mazda RX4	21.0	160.0	-2.5700299
## Mazda RX4 Wag	21.0	160.0	-1.6008028
## Hornet 4 Drive	21.4	258.0	0.1833269
## Volvo 142E	21.4	121.0	-1.6295873
## Toyota Corona	21.5	120.1	-3.1015898
## Datsun 710	22.8	108.0	-2.4886829
## Merc 230	22.8	140.8	0.7591033
## Merc 240D	24.4	146.7	1.4885011

```
## Porsche 914-2      26.0 120.3 -0.0236311
## Fiat X1-9          27.3  79.0 -0.3204259
## Honda Civic        30.4  75.7  1.1240053
## Lotus Europa       30.4  95.1  2.6550420
## Fiat 128           32.4  78.7  5.7865290
## Toyota Corolla     33.9  71.1  5.8609261

### Predicting mileage for a new car with specific characteristics
Xstar_new <- matrix(c(1, 150, 100, 2.7), nrow = 1) # New car with specified features

# Posterior predictive covariance for new prediction
my_Sig_new <- sc_fact * (1 + Xstar_new %*% XtX_inv %*% t(Xstar_new))

# Generate posterior predictive samples for new data
post_pred_err_new <- rmvt(n = 500, sigma = my_Sig_new, df = df)
post_pred_samp_new <- as.numeric(Xstar_new %*% bhat) + post_pred_err_new

# Calculate 95% prediction interval and median prediction for new data
pred_int <- quantile(post_pred_samp_new, probs = c(0.025, 0.5, 0.975))
names(pred_int) <- c("2.5%", "Median", "97.5%")
print(pred_int)

##      2.5%   Median    97.5%
## 17.83004 23.84256 29.09549
```

## Using ‘bayesrules’ Package for Posterior Predictive Analysis - Example: Bayesian Regression on the ‘mtcars’ Dataset

```
# Load necessary libraries
library(bayesrules)
library(ggplot2) # for plotting with ggplot2

# Load the 'mtcars' dataset
data("mtcars")

# Fit a Bayesian linear regression model to predict mpg based on displacement, horsepower, and weight
mod_mtcars <- stan_glm(
  mpg ~ disp + hp + wt,
  data = mtcars,
  family = gaussian,
  prior_intercept = normal(22, 20), # Prior for intercept centered around typical mpg
  prior = normal(c(0.325, -1.563, 0.025), 40, autoscale = TRUE), # Priors for predictors
  prior_aux = exponential(1, autoscale = TRUE), # Prior for residual scale parameter
  chains = 4, iter = 10000 # MCMC specifications
)

##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 1.7e-05 seconds
```

```

## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.17 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:      1 / 10000 [ 0%] (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 1: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 1: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 1: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 1: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 1:
## Chain 1: Elapsed Time: 0.805 seconds (Warm-up)
## Chain 1:              0.603 seconds (Sampling)
## Chain 1:              1.408 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.6e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.16 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:      1 / 10000 [ 0%] (Warmup)
## Chain 2: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 2: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 2: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 2: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 2: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 2: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 2: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 2: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 2: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 2: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 2: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 2:
## Chain 2: Elapsed Time: 0.862 seconds (Warm-up)
## Chain 2:              0.784 seconds (Sampling)
## Chain 2:              1.646 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 1.1e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.11 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:

```

```

## Chain 3: Iteration:    1 / 10000 [ 0%] (Warmup)
## Chain 3: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 3: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 3: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 3: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 3: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 3: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 3: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 3: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 3: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 3: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 3: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 3:
## Chain 3: Elapsed Time: 0.729 seconds (Warm-up)
## Chain 3:           0.847 seconds (Sampling)
## Chain 3:           1.576 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1.4e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.14 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 10000 [ 0%] (Warmup)
## Chain 4: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 4: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 4: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 4: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 4: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 4: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 4: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 4: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 4: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 4: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 4: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 4:
## Chain 4: Elapsed Time: 0.733 seconds (Warm-up)
## Chain 4:           0.584 seconds (Sampling)
## Chain 4:           1.317 seconds (Total)
## Chain 4:

```

```

# Generate posterior predictive samples for observed data in 'mtcars'
preds <- posterior_predict(mod_mtcars, newdata = mtcars)

# Visualize posterior predictive intervals for mpg as a function of weight
ppc_plot <- ppc_intervals(
  y = mtcars$mpg, # Observed values of mpg
  yrep = preds,   # Posterior predictive samples
  x = mtcars$wt,   # Predictor variable for x-axis
  prob = 0.5, prob_outer = 0.95 # Set inner and outer credible intervals
) +
  labs(x = "Weight (wt)",

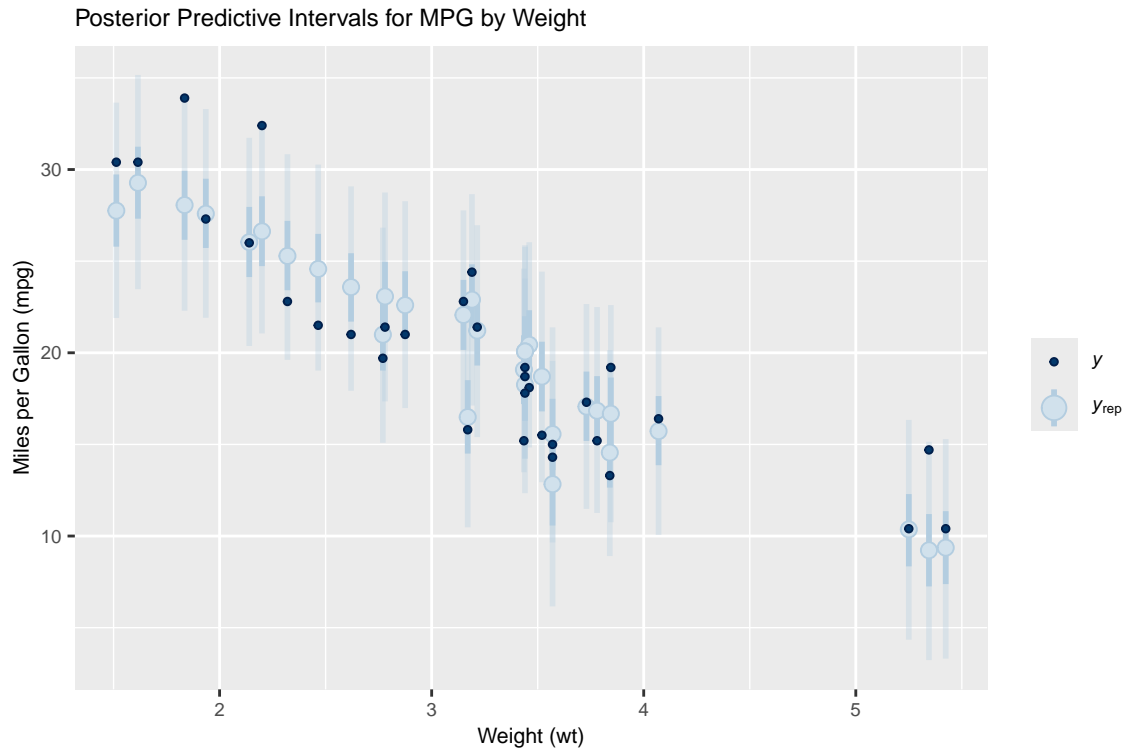
```

```

y = "Miles per Gallon (mpg)",
title = "Posterior Predictive Intervals for MPG by Weight")

# Display posterior predictive intervals plot
print(ppc_plot)

```



```

# Examine posterior predictive accuracy metrics
pred_summ <- prediction_summary(mod_mtcars, data = mtcars)
print(pred_summ)

```

```

##      mae mae_scaled within_50 within_95
## 1 1.54866 0.5403995    0.625    0.9375

```

```

# Metrics Explanation:
# - Median Absolute Error (MAE): Measures typical deviation between observed and predictive means.
# - Scaled MAE: Indicates typical deviation in terms of standard deviations from the predictive mean.
# - within_50 and within_95: Proportions of observed values within 50% and 95% posterior intervals.

#####
# Cross-Validation to Measure Predictive Accuracy
# Cross-validation divides data into 10 subsets, sequentially leaving out each subset for validation.

# Perform 10-fold cross-validation
set.seed(84735)
cv_res <- prediction_summary_cv(
  model = mod_mtcars, data = mtcars, k = 10
)

```

```
# Display cross-validation results for each fold
print(cv_res$folds)
```

```
##      fold      mae mae_scaled within_50 within_95
## 1      1 1.5668808 0.5541181 0.5000000 0.7500000
## 2      2 0.6597668 0.2265432 0.6666667 0.6666667
## 3      3 1.9209424 0.6842893 0.3333333 1.0000000
## 4      4 1.6647891 0.5671217 0.6666667 1.0000000
## 5      5 1.1289763 0.3588407 0.6666667 1.0000000
## 6      6 0.5857264 0.1900541 1.0000000 1.0000000
## 7      7 2.8143618 0.9643920 0.3333333 1.0000000
## 8      8 2.8943132 0.9460955 0.3333333 1.0000000
## 9      9 0.3336720 0.1088700 1.0000000 1.0000000
## 10     10 2.8075622 0.9966960 0.2500000 0.7500000
```

```
# Display average cross-validation metrics across all folds
print(cv_res$cv)
```

```
##      mae mae_scaled within_50 within_95
## 1 1.637699 0.5597021      0.575 0.9166667
```

## Model Comparison Using ELPD (loo) and BIC for Bayesian Models on mtcars Data

```
# Load necessary libraries
library(rstanarm)
library(loo)
```

```
## This is loo version 2.8.0
```

```
## - Online documentation and vignettes at mc-stan.org/loo
```

```
## - As of v2.0.0 loo defaults to 1 core but we recommend using as many as possible. Use the 'cores' arg
```

```
## - Windows 10 users: loo may be very slow if 'mc.cores' is set in your .Rprofile file (see https://gi
```

```
##
```

```
## Attaching package: 'loo'
```

```
## The following object is masked from 'package:rstan':
```

```
##
```

```
##      loo
```

```
library(bayesplot)
```

```
# Fit initial model with three predictors
mod_mtcars <- stan_glm(
```

```

mpg ~ disp + hp + wt,
data = mtcars,
family = gaussian,
prior_intercept = normal(82.5, 20), # Center prior intercept on typical mpg
prior = normal(c(0.325, -1.563, 0.025), 40, autoscale = TRUE), # Priors for predictors
prior_aux = exponential(1, autoscale = TRUE), # Prior for residual error scale
chains = 4, iter = 10000
)

```

```

##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 1.6e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.16 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [  0%] (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 1: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 1: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 1: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 1: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 1:
## Chain 1: Elapsed Time: 0.773 seconds (Warm-up)
## Chain 1:                0.928 seconds (Sampling)
## Chain 1:                1.701 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.3e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.13 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 10000 [  0%] (Warmup)
## Chain 2: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 2: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 2: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 2: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 2: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 2: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 2: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 2: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 2: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 2: Iteration: 9000 / 10000 [ 90%] (Sampling)

```



```

## Chain 2: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 2:
## Chain 2: Elapsed Time: 0.816 seconds (Warm-up)
## Chain 2: 0.707 seconds (Sampling)
## Chain 2: 1.523 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 1.6e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.16 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration: 1 / 10000 [ 0%] (Warmup)
## Chain 3: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 3: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 3: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 3: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 3: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 3: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 3: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 3: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 3: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 3: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 3: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 3:
## Chain 3: Elapsed Time: 0.677 seconds (Warm-up)
## Chain 3: 0.801 seconds (Sampling)
## Chain 3: 1.478 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1.4e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.14 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration: 1 / 10000 [ 0%] (Warmup)
## Chain 4: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 4: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 4: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 4: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 4: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 4: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 4: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 4: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 4: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 4: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 4: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 4:
## Chain 4: Elapsed Time: 0.794 seconds (Warm-up)
## Chain 4: 0.782 seconds (Sampling)

```

```
## Chain 4: 1.576 seconds (Total)
## Chain 4:
```

```
# Calculate ELPD for mod_mtcars
mod_elpd <- loo(mod_mtcars)
cat("ELPD Estimates for Initial Model (mpg ~ disp + hp + wt):\n")
```

```
## ELPD Estimates for Initial Model (mpg ~ disp + hp + wt):
```

```
print(mod_elpd$estimates)
```

```
##           Estimate      SE
## elpd_loo -79.945651 4.546744
## p_loo    4.853112 1.440363
## looic    159.891303 9.093489
```

```
# Define alternative model formulas to compare
model_formulas <- list(
  mpg ~ disp + hp,
  mpg ~ disp + wt
)

# Fit models based on alternative formulas, dynamically setting priors
models <- lapply(model_formulas, function(formula) {
  n_predictors <- length(all.vars(formula)) - 1 # Exclude the response variable
  stan_glm(
    formula, data = mtcars,
    family = gaussian,
    prior_intercept = normal(82.5, 20),
    prior = normal(rep(0, n_predictors), 40, autoscale = TRUE),
    prior_aux = exponential(1, autoscale = TRUE),
    chains = 4, iter = 10000
  )
})
```

```
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 1.7e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.17 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration: 1 / 10000 [ 0%] (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 1: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 1: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 1: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 1: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%] (Sampling)
```

```

## Chain 1: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 1:
## Chain 1: Elapsed Time: 0.48 seconds (Warm-up)
## Chain 1: 0.401 seconds (Sampling)
## Chain 1: 0.881 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.6e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.16 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration: 1 / 10000 [ 0%] (Warmup)
## Chain 2: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 2: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 2: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 2: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 2: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 2: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 2: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 2: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 2: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 2: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 2: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 2:
## Chain 2: Elapsed Time: 0.504 seconds (Warm-up)
## Chain 2: 0.388 seconds (Sampling)
## Chain 2: 0.892 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 1.2e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.12 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration: 1 / 10000 [ 0%] (Warmup)
## Chain 3: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 3: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 3: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 3: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 3: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 3: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 3: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 3: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 3: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 3: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 3: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 3:

```

```

## Chain 3: Elapsed Time: 0.472 seconds (Warm-up)
## Chain 3:           0.445 seconds (Sampling)
## Chain 3:           0.917 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1.3e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.13 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 10000 [ 0%] (Warmup)
## Chain 4: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 4: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 4: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 4: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 4: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 4: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 4: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 4: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 4: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 4: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 4: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 4:
## Chain 4: Elapsed Time: 0.517 seconds (Warm-up)
## Chain 4:           0.475 seconds (Sampling)
## Chain 4:           0.992 seconds (Total)
## Chain 4:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 1.5e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.15 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [ 0%] (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 1: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 1: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 1: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 1: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 1:
## Chain 1: Elapsed Time: 0.645 seconds (Warm-up)
## Chain 1:           0.592 seconds (Sampling)
## Chain 1:           1.237 seconds (Total)
## Chain 1:

```

```

##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.1e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.11 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 10000 [ 0%] (Warmup)
## Chain 2: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 2: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 2: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 2: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 2: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 2: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 2: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 2: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 2: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 2: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 2: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 2:
## Chain 2: Elapsed Time: 0.686 seconds (Warm-up)
## Chain 2:                0.505 seconds (Sampling)
## Chain 2:                1.191 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 1.2e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.12 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 10000 [ 0%] (Warmup)
## Chain 3: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 3: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 3: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 3: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 3: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 3: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 3: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 3: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 3: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 3: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 3: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 3:
## Chain 3: Elapsed Time: 0.569 seconds (Warm-up)
## Chain 3:                0.54 seconds (Sampling)
## Chain 3:                1.109 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1.1e-05 seconds

```

```
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.11 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:      1 / 10000 [ 0%] (Warmup)
## Chain 4: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 4: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 4: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 4: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 4: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 4: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 4: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 4: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 4: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 4: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 4: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 4:
## Chain 4: Elapsed Time: 0.626 seconds (Warm-up)
## Chain 4:           0.552 seconds (Sampling)
## Chain 4:           1.178 seconds (Total)
## Chain 4:
```

```
# Calculate ELPD and BIC for each model and store results, adding error handling
model_names <- c("mpg ~ disp + hp + wt", "mpg ~ disp + hp", "mpg ~ disp + wt")

# Calculate ELPD
elpd_res <- sapply(c(list(mod_mtcars), models), function(mod) {
  elpd <- tryCatch(loo(mod)$estimates[1, 1], error = function(e) NA) # Return NA if loo fails
})

elpd_res
```

```
## [1] -79.94565 -84.49547 -82.54823
```

```
# Identify the best model based on ELPD and BIC
best_model_elpd <- model_names[which.max(elpd_res) ]

cat("\nBest Model Based on ELPD:\n")
```

```
##
## Best Model Based on ELPD:
```

```
print(best_model_elpd)
```

```
## [1] "mpg ~ disp + hp + wt"
```

```
## Also see BIC values to compare regression models...
# Calculate BIC
# Define function to calculate BIC for a stan_glm model
calculate_bic <- function(model, data) {
  log_lik_matrix <- as.matrix(log_lik(model))
  log_lik <- sum(apply(log_lik_matrix, 2, mean)) # Total log-likelihood (mean over iterations)
```

```

num_params <- length(fixef(model)) # Number of effective parameters (fixed effects)
num_observations <- nrow(data) # Number of observations in the dataset
bic_value <- -2 * log_lik + num_params * log(num_observations) # BIC formula
return(bic_value)
}

```

```

# Calculate BIC for each model
bic_res <- sapply(c(list(mod_mtcars), models), calculate_bic, data = mtcars)
print(bic_res)

```

```
## [1] 167.9338 175.3980 170.9318
```

```

best_model_bic <- model_names[which.min(bic_res) ]
cat("\nBest Model Based on BIC:\n")

```

```

##
## Best Model Based on BIC:

```

```
print(best_model_bic)
```

```
## [1] "mpg ~ disp + hp + wt"
```