SGupta_HW05_CodeOtherProblem

```
library(tidyverse)
```

```
-- 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 ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
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 becor
```

```
library(invgamma)

# Read the diabetes dataset
diabetes <- read_csv("diabetes-dataset.csv")
```

```
Rows: 2000 Columns: 9
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (9): Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, D...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# Logit function
logit <- function(p) {
  log(p / (1 - p))
}
```

```r
# Expit function
expit <- function(x) {
  exp(x) / (1 + exp(x))
}

# Log posterior function
log_posterior <- function(y, x, beta) {
  # Extract beta values
  beta0 <- beta[1]
  beta1 <- beta[2]

  # Calculate the log likelihood
  log_likelihood <- sum(y * log(expit(beta0 + beta1 * x)) +
                          (1 - y) * log(1 - expit(beta0 + beta1 * x)))

  # Calculate log prior
  log_prior <- sum(dnorm(beta0, 0, 15, log = TRUE)) +
    sum(dnorm(beta1, 0, 15, log = TRUE))

  # Return the log posterior
  return(log_likelihood + log_prior)
}

# MCMC sampler function
bayes_logistic <- function(y, x, init_beta, n_samples = 100000, burn_in = 10000, can_SD = c(
  # Initialize beta
  beta <- init_beta

  # Data frame to keep track of samples
  keep_beta <- data.frame(beta0 = numeric(n_samples), beta1 = numeric(n_samples))
  keep_beta[1, ] <- beta

  # Current log posterior
  cur_log_post <- log_posterior(y, x, beta)

  for (i in 2:n_samples) {
    # Update beta0 using MH sampling
    can_beta0 <- rnorm(1, beta[1], can_SD[1])
    can_beta <- c(can_beta0, beta[2])

    can_log_post <- log_posterior(y, x, can_beta)
```

```r
    # Calculate log acceptance ratio
    log_R0 <- can_log_post - cur_log_post
    log_U <- log(runif(1))

    if (log_U < log_R0) {  # Accept
      beta[1] <- can_beta0
      cur_log_post <- can_log_post
    }

    # Update beta1 using MH sampling
    can_beta1 <- rnorm(1, beta[2], can_SD[2])
    can_beta <- c(beta[1], can_beta1)

    can_log_post <- log_posterior(y, x, can_beta)

    # Calculate log acceptance ratio
    log_R1 <- can_log_post - cur_log_post
    log_U <- log(runif(1))

    if (log_U < log_R1) {  # Accept
      beta[2] <- can_beta1
      cur_log_post <- can_log_post
    }

    # Store current beta values
    keep_beta[i, ] <- beta
  }

  # Return posterior samples
  return(keep_beta)
}

# Fit the model using the diabetes data
burn_in <- 10000  # Define the burn-in period
fit <- bayes_logistic(y = diabetes$Outcome,
                      x = diabetes$Glucose,
                      init_beta = c(0, 0))

# Analyze the results after burn-in
burned_fit <- fit[-(1:burn_in), ]

# Report mean and standard deviation of posteriors for beta0 and beta1
```

```r
beta0_summary <- c(mean(burned_fit$beta0), sd(burned_fit$beta0))
beta1_summary <- c(mean(burned_fit$beta1), sd(burned_fit$beta1))

# Get 95% credible intervals for beta0 and beta1
beta0_CI <- quantile(burned_fit$beta0, c(0.025, 0.975))
beta1_CI <- quantile(burned_fit$beta1, c(0.025, 0.975))

# Print summaries and intervals
cat("Posterior mean and SD for beta0:", beta0_summary, "\n")
```

```
Posterior mean and SD for beta0: -5.243937 0.2615598
```

```r
cat("95% credible interval for beta0:", beta0_CI, "\n")
```

```
95% credible interval for beta0: -5.74896 -4.723065
```

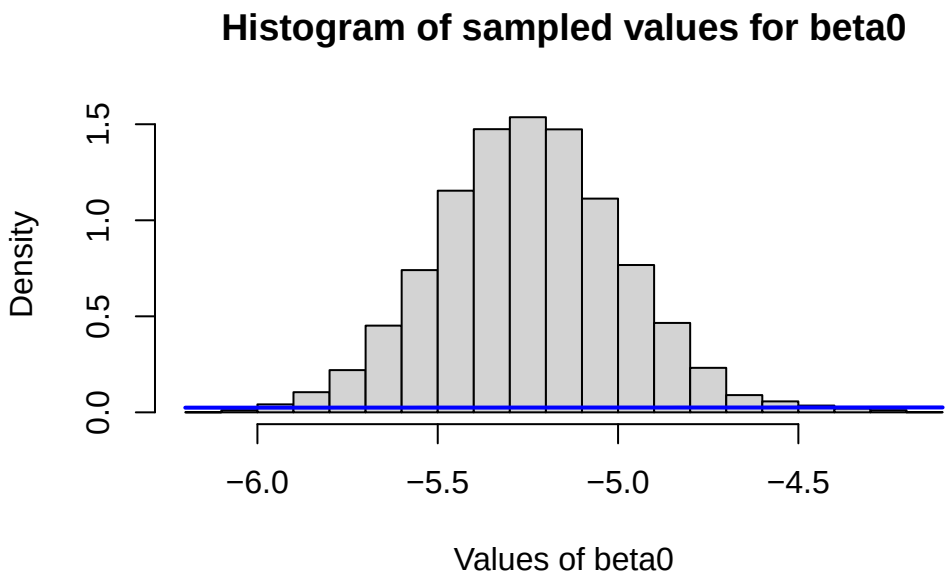
```r
cat("Posterior mean and SD for beta1:", beta1_summary, "\n")
```

```
Posterior mean and SD for beta1: 0.03660986 0.002005417
```

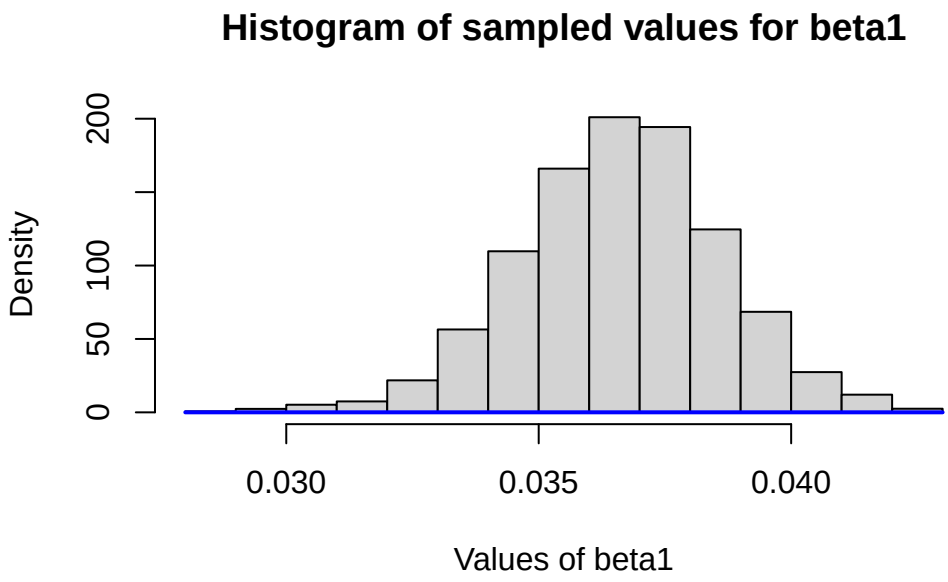
```r
cat("95% credible interval for beta1:", beta1_CI, "\n")
```

```
95% credible interval for beta1: 0.03262847 0.04050688
```

```r
# Plot histogram of beta0 samples
hist(burned_fit$beta0, prob = TRUE,
     xlab = "Values of beta0",
     main = "Histogram of sampled values for beta0")
# Overlay the normal density for the prior
curve(dnorm(x, mean = 0, sd = 15), add = TRUE, col = "blue", lwd = 2)
```

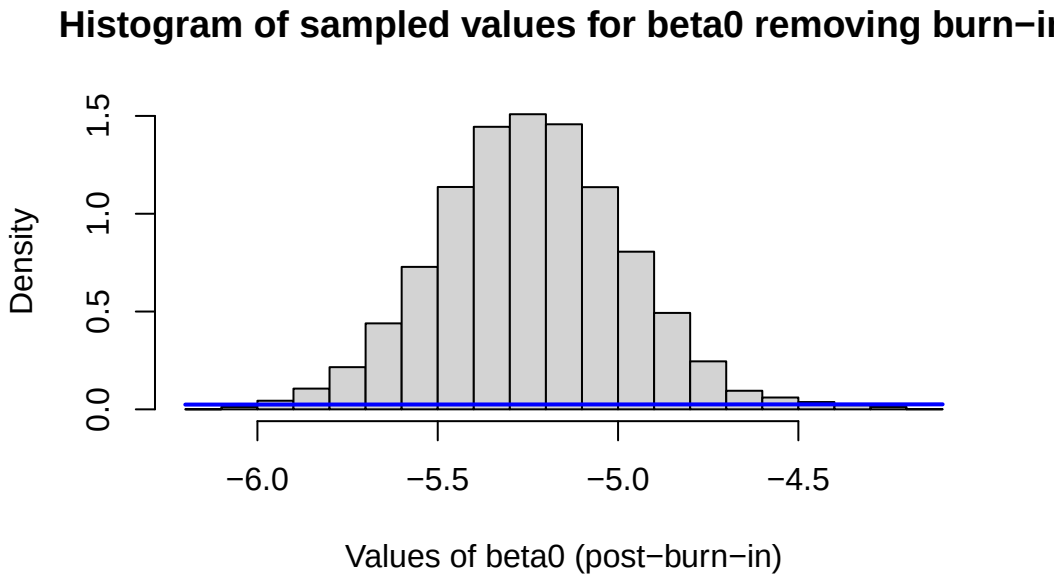



```r
# Plot histogram of beta1 samples
hist(burned_fit$beta1, prob = TRUE,
     xlab = "Values of beta1",
     main = "Histogram of sampled values for beta1")
# Overlay the normal density for the prior
curve(dnorm(x, mean = 0, sd = 15), add = TRUE, col = "blue", lwd = 2)
```

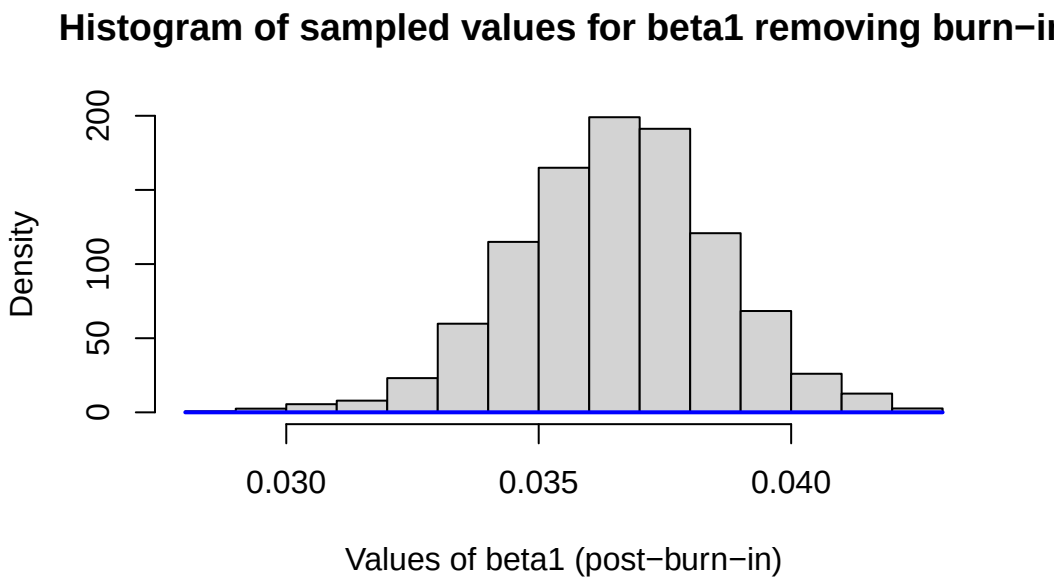



```
# You can discard the first 5000 random draws as "burn-in" if needed
burnin <- 5000

# Re-make the plot for beta0 without burn-in
hist(burned_fit$beta0[-c(1:burnin)], prob = TRUE,
     xlab = "Values of beta0 (post-burn-in)",
     main = "Histogram of sampled values for beta0 removing burn-in")
curve(dnorm(x, mean = 0, sd = 15), add = TRUE, col = "blue", lwd = 2)
```




```
# Re-make the plot for beta1 without burn-in
hist(burned_fit$beta1[-c(1:burnin)], prob = TRUE,
     xlab = "Values of beta1 (post-burn-in)",
     main = "Histogram of sampled values for beta1 removing burn-in")
curve(dnorm(x, mean = 0, sd = 15), add = TRUE, col = "blue", lwd = 2)
```


Histogram of sampled values for beta1 removing burn−in
Density
0
50
100
200
0.030
0.035
0.040
Values of beta1 (post−burn−in)