# Lab 6

Chenxi Liu 1010615050

2024-02-19

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.3     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## 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 become errors
```

```
library(here)
```

```
## here() starts at /Users/dawn/Desktop/uoft/sta2201/HW
```

```
# for bayes stuff
library(rstan)
```

```
## Loading required package: StanHeaders
##
## rstan version 2.32.5 (Stan version 2.32.2)
##
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
## For within-chain threading using ‘reduce_sum()‘ or ‘map_rect()‘ Stan functions,
## change ‘threads_per_chain‘ option:
## rstan_options(threads_per_chain = 1)
##
##
## Attaching package: 'rstan'
##
## The following object is masked from 'package:tidyr':
##
##     extract
```

```
library(bayesplot)
```

```
## 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(loo)
```

```
## This is loo version 2.6.0
## - 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' ar
##
## Attaching package: 'loo'
##
## The following object is masked from 'package:rstan':
##
##     loo
```

```
library(tidybayes)
```

```
ds <- read_rds(file = "/Users/dawn/Desktop/uoft/sta2201/HW/births_2017_sample.RDS")
ds <- ds %>%
  rename(birthweight = dbwt, gest = combgest) %>%
  mutate(preterm = ifelse(gest<32, "Y", "N")) %>%
  filter(ilive=="Y",gest< 99, birthweight<9.999)
head(ds)
```

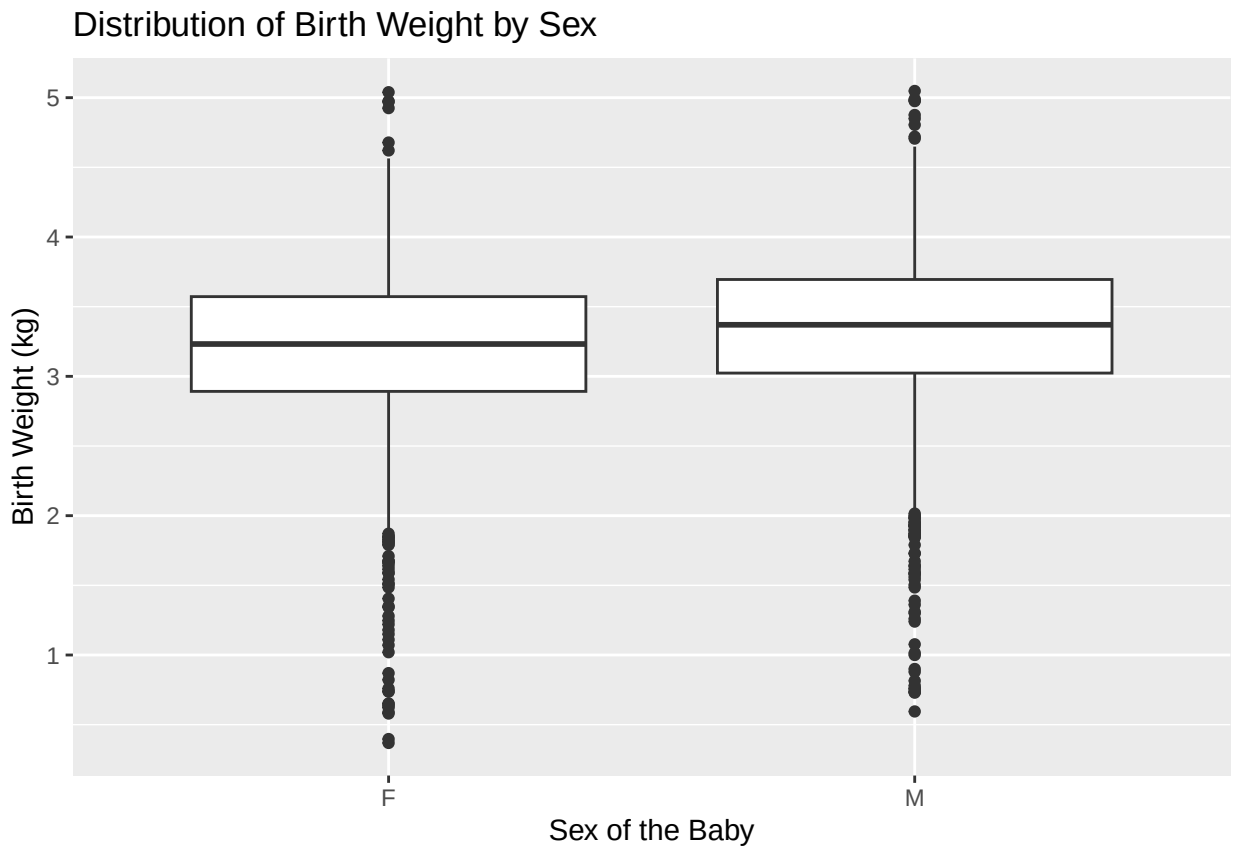
```
## # A tibble: 6 x 9
##   mager mracehisp meduc   bmi sex    gest birthweight ilive preterm
##   <dbl>     <dbl> <dbl> <dbl> <chr> <dbl>       <dbl> <chr> <chr>
## 1    16         2     2  23   M        39        3.18 Y     N
## 2    25         7     2  43.6 M        40        4.14 Y     N
## 3    27         2     3  19.5 F        41        3.18 Y     N
## 4    26         1     3  21.5 F        36        3.40 Y     N
## 5    28         7     2  40.6 F        34        2.71 Y     N
## 6    31         7     3  29.3 M        35        3.52 Y     N
```

## Question 1

Use plots or tables to show three interesting observations about the data. Remember:

- Explain what your graph/ tables show
- Choose a graph type that's appropriate to the data type
- If you use `geom_smooth`, please also plot the underlying data

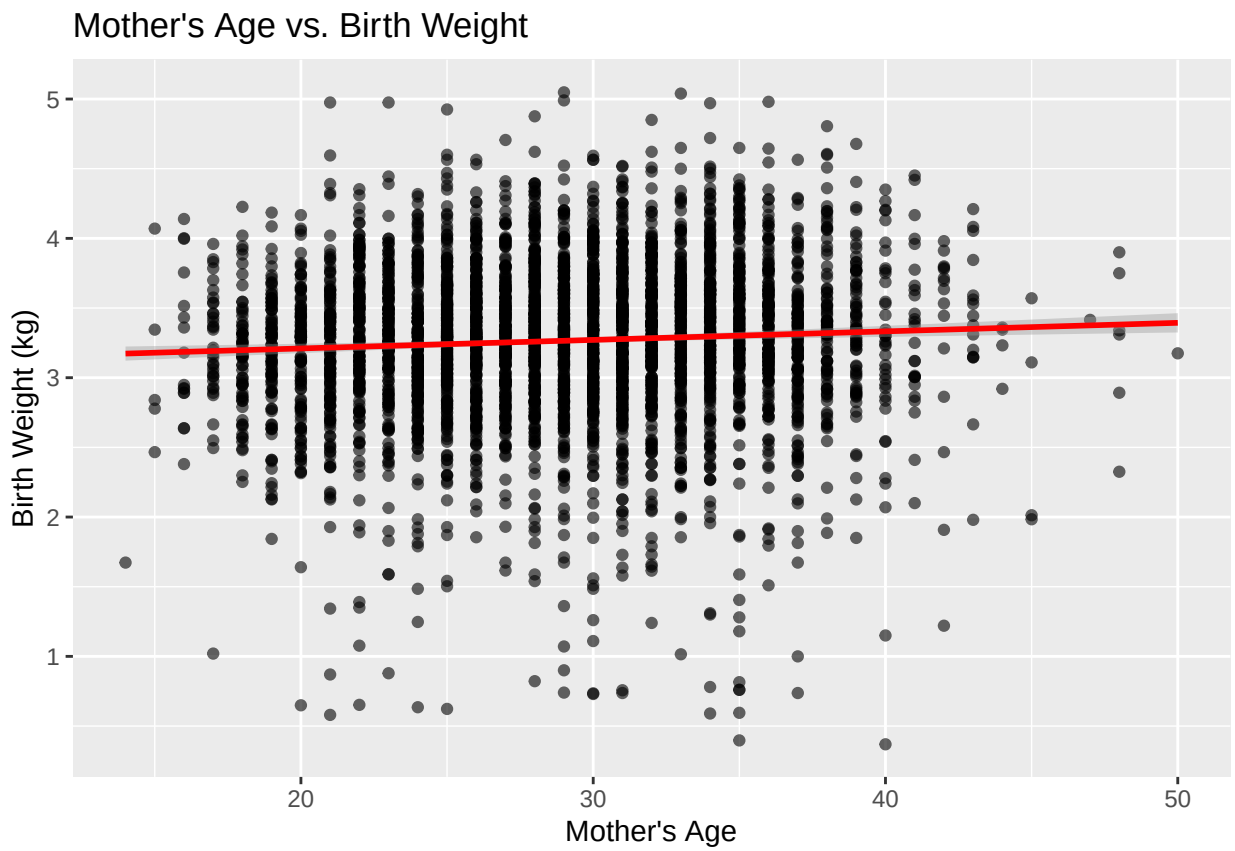
```r
# Observation 1: Distribution of Birth Weight by Sex
# Using a boxplot
p1 <- ggplot(ds, aes(x = sex, y = birthweight)) +
      geom_boxplot() +
      labs(title = "Distribution of Birth Weight by Sex", x = "Sex of the Baby", y = "Birth Weight (kg)")
p1
```




The mean of birth weight of baby boys is higher than that of girls, 3.4 and 3.3 approximately respectively.

```r
# Observation 2: Relationship between Mother's Age and Birth Weight
# Using a scatter plot with a smooth line
p2 <- ggplot(ds, aes(x = mager, y = birthweight)) +
      geom_point(alpha = 0.6) +
      geom_smooth(method = "lm", color = "red") +
      labs(title = "Mother's Age vs. Birth Weight", x = "Mother's Age", y = "Birth Weight (kg)")
p2
```

```
## `geom_smooth()` using formula = 'y ~ x'
```




It seems like the age of mother has a slightly positive relationship with the weight of baby, but the values vary too much with lots of outliers.

```
# Observation 3: Average Birth Weight by Gestational Age
# Creating a summary table
avg_birthweight_by_gest <- ds %>%
                           group_by(gest) %>%
                           summarize(AverageBirthWeight = mean(birthweight)) %>%
                           arrange(gest)
avg_birthweight_by_gest
```

```
## # A tibble: 26 x 2
##     gest AverageBirthWeight
##    <dbl>              <dbl>
##  1    21              0.482
##  2    23              0.698
##  3    24              0.635
##  4    25              0.839
##  5    26              0.830
##  6    27              1.00
##  7    28              1.57
##  8    29              1.76
##  9    30              2.03
## 10    31              2.44
## # i 16 more rows
```

This table summarizes the average birth weight for each gestational week. It can be clearly seen that when

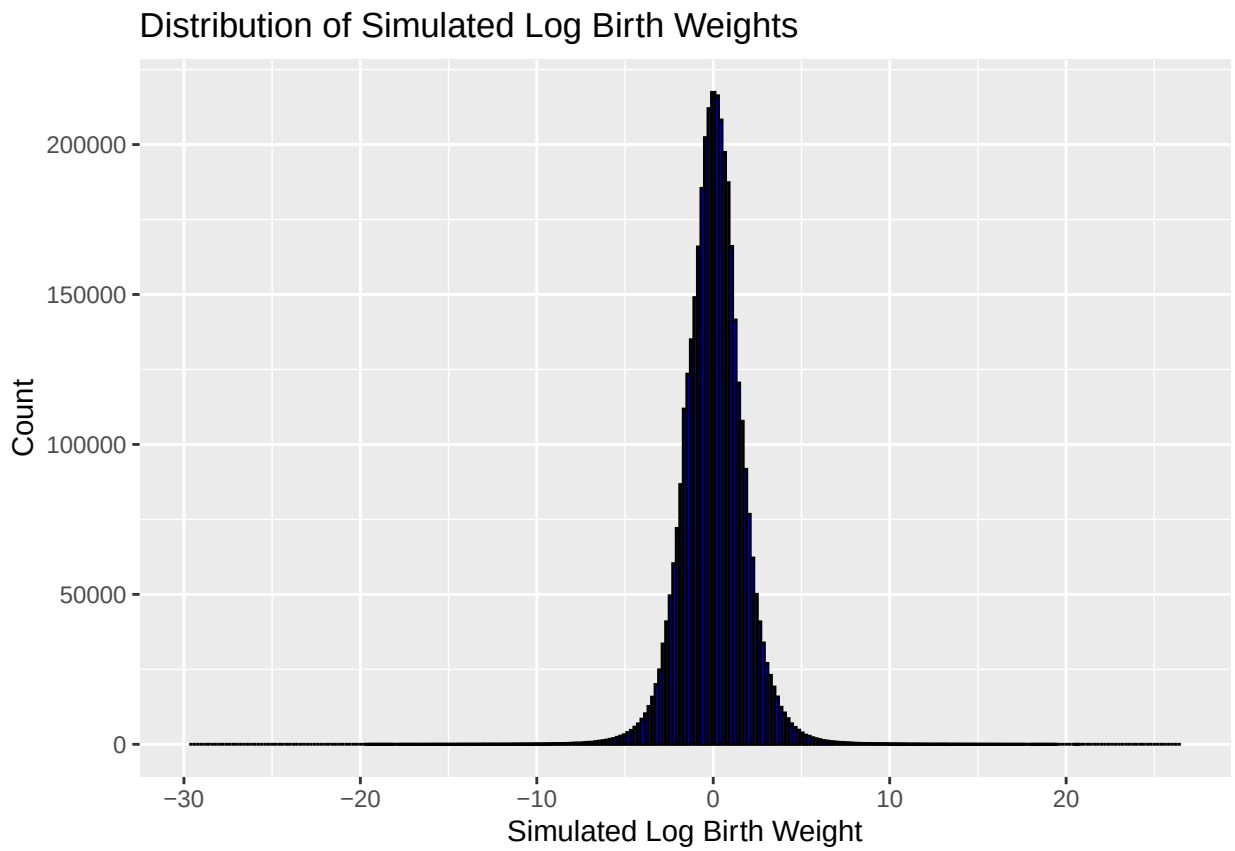
the gest is longer, the weight of baby is higher, which also means healthier.

## Question 2

For Model 1, simulate values of $\beta$s and $\sigma$ based on the priors above. Do 1000 simulations. Use these values to simulate (log) birth weights from the likelihood specified in Model 1, based on the set of observed gestational weights. **Remember the gestational weights should be centered and standardized**.

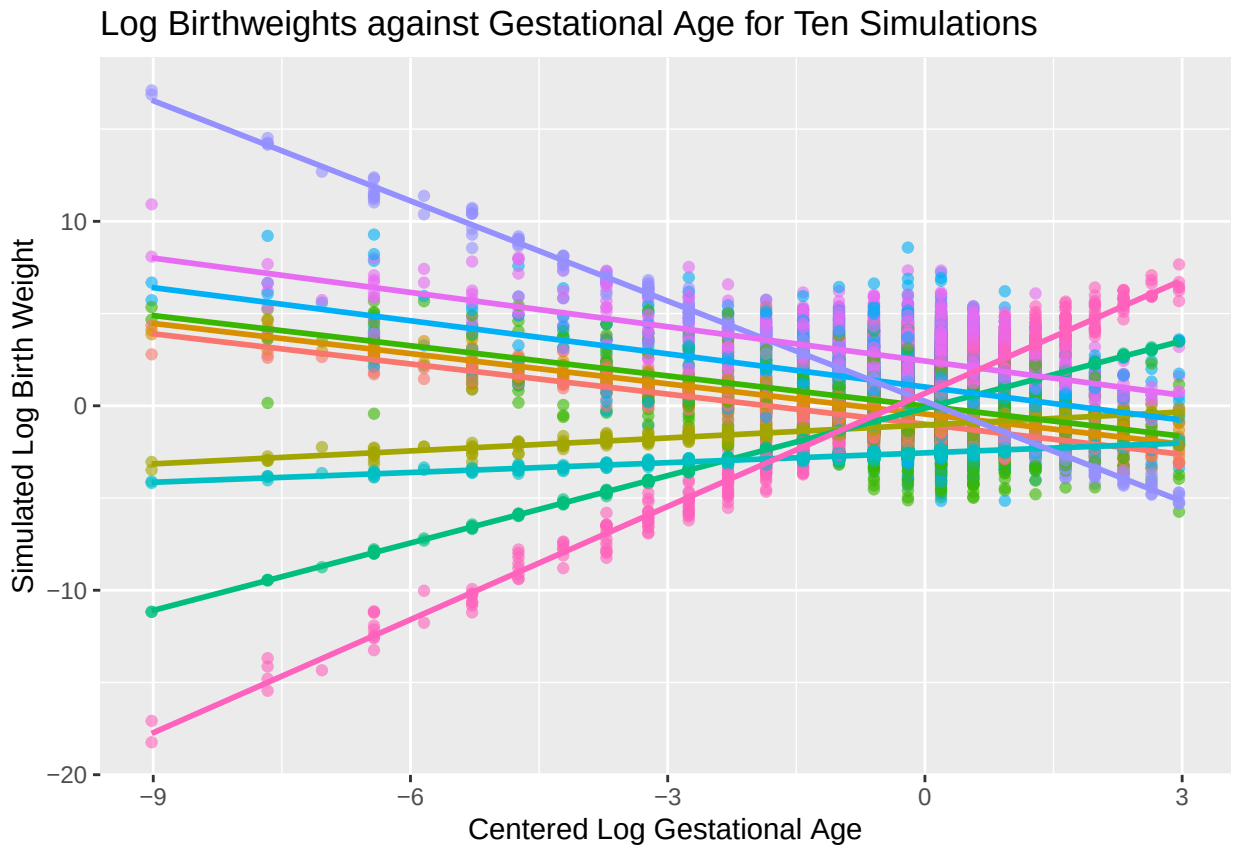
- Plot the resulting distribution of simulated (log) birth weights.
- Plot ten simulations of (log) birthweights against gestational age.

```
set.seed(123)

n <- 1000
sigma <- abs(rnorm(n, 0, 1))
beta0 <- rnorm(n, 0, 1)
beta1 <- rnorm(n, 0, 1)

center_gest <- tibble(log_gest = (log(ds$gest) - mean(log(ds$gest)))/sd(log(ds$gest)))

for(i in 1:n){
  mu <- beta0[i] + beta1[i]*center_gest$log_gest
  center_gest[paste0(i)] <- mu + rnorm(nrow(center_gest), 0, sigma[i])
}

df1 <- center_gest %>%
  pivot_longer(cols = -log_gest, names_to = "simulation", values_to = "simu_weight")

# Plot the distribution of simulated (log) birth weights
p1 <- ggplot(df1, aes(x = simu_weight)) +
  geom_histogram(binwidth = 0.2, color = "black", fill = "blue") +
  labs(title = "Distribution of Simulated Log Birth Weights", x = "Simulated Log Birth Weight", y = "Cou

# Plot ten simulations of (log) birthweights against gestational age
# Selecting first 10 simulations for plotting
df2 <- df1 %>%
  filter(simulation %in% as.character(1:10))

p2 <- ggplot(df2, aes(x = log_gest, y = simu_weight, color = simulation)) +
  geom_point(alpha = 0.6) +
  geom_smooth(se = FALSE, method = "lm") +
  labs(title = "Log Birthweights against Gestational Age for Ten Simulations",
       x = "Centered Log Gestational Age",
       y = "Simulated Log Birth Weight") +
  theme(legend.position = "none")

p1
```




```
p2
```

```
## `geom_smooth()` using formula = 'y ~ x'
```




## Run the model

Now we're going to run Model 1 in Stan. The stan code is in the `code/models` folder.

First, get our data into right form for input into stan.

```
ds$log_weight <- log(ds$birthweight)
ds$log_gest_c <- (log(ds$gest) - mean(log(ds$gest)))/sd(log(ds$gest))

# put into a list
stan_data <- list(N = nrow(ds),
                  log_weight = ds$log_weight,
                  log_gest = ds$log_gest_c)
```

Now fit the model

```
mod1 <- stan(data = stan_data,
             file = "/Users/dawn/Desktop/uoft/sta2201/HW/simple_weight.stan",
             iter = 500,
             seed = 243)
```

```
## Warning in readLines(file, warn = TRUE): incomplete final line found on
## '/Users/dawn/Desktop/uoft/sta2201/HW/simple_weight.stan'
```

```
## Trying to compile a simple C file

## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## using C compiler: 'Apple clang version 14.0.3 (clang-1403.0.22.14.1)'
## using SDK: 'MacOSX13.3.sdk'
## clang -arch arm64 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Library/Framew
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/StanHeade
## In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen/
## In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen/
## /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen/include/Eigen/src/Core
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen/include/Eigen/src/Core
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/StanHeade
## In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen/
## /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen/include/Eigen/Core:96
## #include <complex>
##          ^~~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000113 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 1.13 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 500 [  0%]  (Warmup)
## Chain 1: Iteration:  50 / 500 [ 10%]  (Warmup)
## Chain 1: Iteration: 100 / 500 [ 20%]  (Warmup)
## Chain 1: Iteration: 150 / 500 [ 30%]  (Warmup)
## Chain 1: Iteration: 200 / 500 [ 40%]  (Warmup)
## Chain 1: Iteration: 250 / 500 [ 50%]  (Warmup)
## Chain 1: Iteration: 251 / 500 [ 50%]  (Sampling)
## Chain 1: Iteration: 300 / 500 [ 60%]  (Sampling)
## Chain 1: Iteration: 350 / 500 [ 70%]  (Sampling)
## Chain 1: Iteration: 400 / 500 [ 80%]  (Sampling)
## Chain 1: Iteration: 450 / 500 [ 90%]  (Sampling)
## Chain 1: Iteration: 500 / 500 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.209 seconds (Warm-up)
## Chain 1:                0.164 seconds (Sampling)
## Chain 1:                0.373 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 7e-05 seconds
```

```
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.7 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 500 [  0%]  (Warmup)
## Chain 2: Iteration:  50 / 500 [ 10%]  (Warmup)
## Chain 2: Iteration: 100 / 500 [ 20%]  (Warmup)
## Chain 2: Iteration: 150 / 500 [ 30%]  (Warmup)
## Chain 2: Iteration: 200 / 500 [ 40%]  (Warmup)
## Chain 2: Iteration: 250 / 500 [ 50%]  (Warmup)
## Chain 2: Iteration: 251 / 500 [ 50%]  (Sampling)
## Chain 2: Iteration: 300 / 500 [ 60%]  (Sampling)
## Chain 2: Iteration: 350 / 500 [ 70%]  (Sampling)
## Chain 2: Iteration: 400 / 500 [ 80%]  (Sampling)
## Chain 2: Iteration: 450 / 500 [ 90%]  (Sampling)
## Chain 2: Iteration: 500 / 500 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.197 seconds (Warm-up)
## Chain 2:                0.188 seconds (Sampling)
## Chain 2:                0.385 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 6.8e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.68 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:   1 / 500 [  0%]  (Warmup)
## Chain 3: Iteration:  50 / 500 [ 10%]  (Warmup)
## Chain 3: Iteration: 100 / 500 [ 20%]  (Warmup)
## Chain 3: Iteration: 150 / 500 [ 30%]  (Warmup)
## Chain 3: Iteration: 200 / 500 [ 40%]  (Warmup)
## Chain 3: Iteration: 250 / 500 [ 50%]  (Warmup)
## Chain 3: Iteration: 251 / 500 [ 50%]  (Sampling)
## Chain 3: Iteration: 300 / 500 [ 60%]  (Sampling)
## Chain 3: Iteration: 350 / 500 [ 70%]  (Sampling)
## Chain 3: Iteration: 400 / 500 [ 80%]  (Sampling)
## Chain 3: Iteration: 450 / 500 [ 90%]  (Sampling)
## Chain 3: Iteration: 500 / 500 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.205 seconds (Warm-up)
## Chain 3:                0.178 seconds (Sampling)
## Chain 3:                0.383 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 7.9e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.79 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
```

```
## Chain 4: Iteration:   1 / 500 [  0%]  (Warmup)
## Chain 4: Iteration:  50 / 500 [ 10%]  (Warmup)
## Chain 4: Iteration: 100 / 500 [ 20%]  (Warmup)
## Chain 4: Iteration: 150 / 500 [ 30%]  (Warmup)
## Chain 4: Iteration: 200 / 500 [ 40%]  (Warmup)
## Chain 4: Iteration: 250 / 500 [ 50%]  (Warmup)
## Chain 4: Iteration: 251 / 500 [ 50%]  (Sampling)
## Chain 4: Iteration: 300 / 500 [ 60%]  (Sampling)
## Chain 4: Iteration: 350 / 500 [ 70%]  (Sampling)
## Chain 4: Iteration: 400 / 500 [ 80%]  (Sampling)
## Chain 4: Iteration: 450 / 500 [ 90%]  (Sampling)
## Chain 4: Iteration: 500 / 500 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.201 seconds (Warm-up)
## Chain 4:                0.178 seconds (Sampling)
## Chain 4:                0.379 seconds (Total)
## Chain 4:

## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#bulk-ess

## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quanti
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#tail-ess
```

```
summary(mod1)$summary[c("beta[1]", "beta[2]", "sigma"),]
```

```
##              mean      se_mean          sd      2.5%       25%       50%
## beta[1] 1.1626293 8.109795e-05 0.002794886 1.1571530 1.1607401 1.1626100
## beta[2] 0.1437074 7.050797e-05 0.002760482 0.1381359 0.1418908 0.1436313
## sigma   0.1688448 1.025235e-04 0.001845326 0.1652251 0.1675565 0.1689364
##               75%     97.5%    n_eff      Rhat
## beta[1] 1.1646213 1.1679552 1187.705 0.9970491
## beta[2] 0.1454944 0.1491751 1532.828 0.9972723
## sigma   0.1700804 0.1722141  323.966 1.0095667
```

## Question 3

Based on Model 1, give an estimate of the expected birthweight of a baby who was born at a gestational age of 37 weeks.

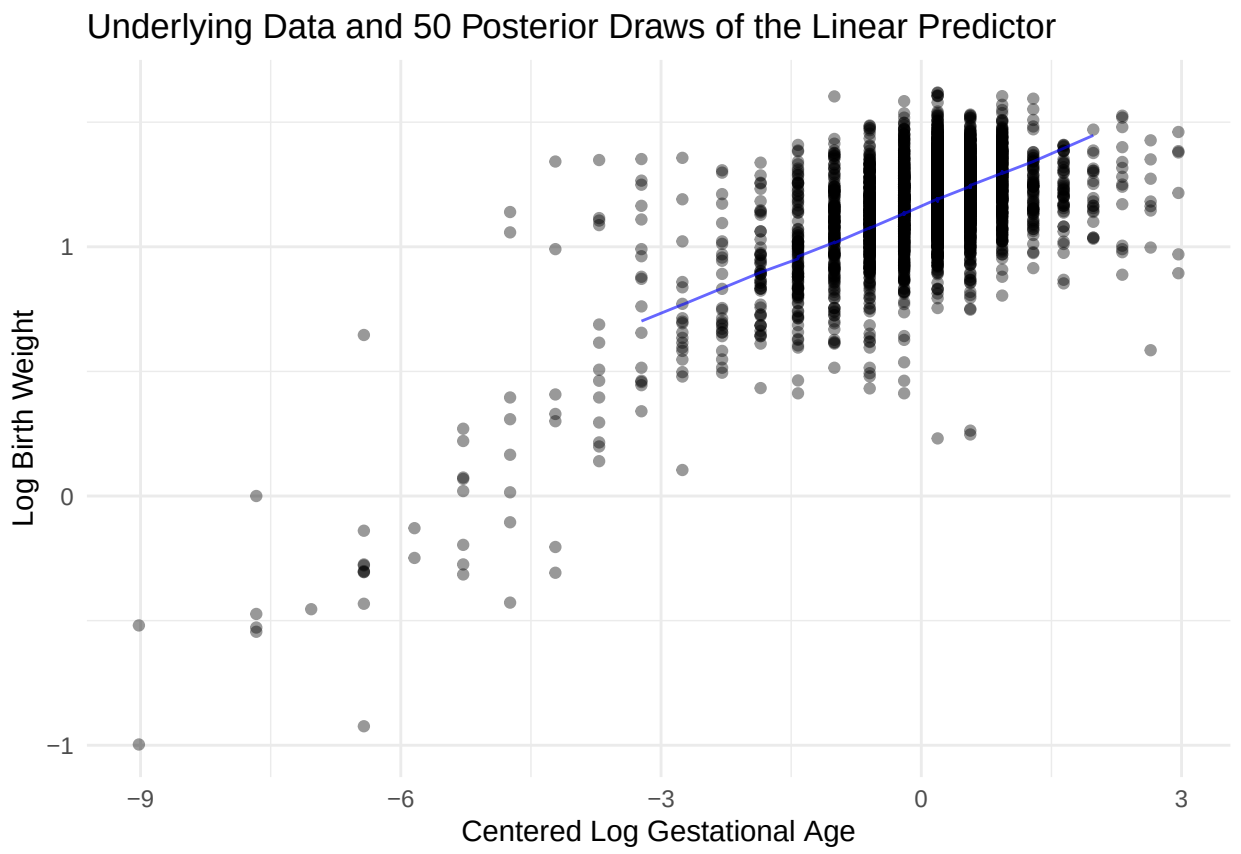
```
gest_37 <- (log(37) - mean(log(ds$gest)))/sd(log(ds$gest))
samps <- rstan::extract(mod1)
median(exp(samps[["beta"]][,1] + gest_37*samps[["beta"]][,2]))
```

```
## [1] 2.936571
```

## Question 4

Based on Model 1, create a scatter plot showing the underlying data (on the appropriate scale) and 50 posterior draws of the linear predictor.

```r
# Extract samples from the model
samps <- rstan::extract(mod1)

# Take 50 posterior draws for the linear predictor for gestational age of 37
posterior_draws <- samps[["beta"]][,1] + ds$log_gest_c[1:2000] * samps[["beta"]][,2]
posterior_draws_50 <- matrix(posterior_draws[1:50], ncol=1)  # Assuming the draws are in rows

# Data frame for plotting
posterior_data <- data.frame(
  log_gest_c = ds$log_gest_c[1:50] ,
  log_weight_draws = posterior_draws_50
)

# Scatter plot with underlying data
p <- ggplot() +
  geom_point(data = ds, aes(x = log_gest_c, y = log_weight), alpha = 0.4) +
  geom_line(data = posterior_data, aes(x = log_gest_c, y = log_weight_draws), color = 'blue', alpha = 0
  labs(title = "Underlying Data and 50 Posterior Draws of the Linear Predictor",
       x = "Centered Log Gestational Age",
       y = "Log Birth Weight") +
  theme_minimal()

p
```




## Question 5

Write a Stan model to run Model 2, and run it. Report a summary of the results, and interpret the coefficient

estimate on the interaction term.

```
# Define the Stan model code as a character string
stan_model_code <- '
data {
  int<lower=1> N;              // number of observations
  vector[N] log_gest;          // log gestational age
  vector[N] log_weight;        // log birth weight
  int<lower=0,upper=1> z_i[N]; // preterm indicator (0 or 1)
}
parameters {
  real beta_1;                 // intercept
  real beta_2;                 // coef for log gestational age
  real beta_3;                 // coef for interaction term
  real<lower=0> sigma;         // error sd for Gaussian likelihood
}
model {
  // Convert z_i to a vector for element-wise multiplication
  vector[N] z_i_vec = to_vector(z_i);

  // Log-likelihood
  target += normal_lpdf(log_weight | beta_1 + beta_2 * log_gest + beta_3 * log_gest .* z_i_vec, sigma);

  // Log-priors
  target += normal_lpdf(sigma | 0, 1)
          + normal_lpdf(beta_1 | 0, 1)
          + normal_lpdf(beta_2 | 0, 1)
          + normal_lpdf(beta_3 | 0, 1);
}
generated quantities {
  vector[N] log_lik;           // pointwise log-likelihood for LOO
  vector[N] log_weight_rep;    // replications from posterior predictive dist

  for (n in 1:N) {
    real log_weight_hat_n = beta_1 + beta_2 * log_gest[n] + beta_3 * log_gest[n] * z_i[n];
    log_lik[n] = normal_lpdf(log_weight[n] | log_weight_hat_n, sigma);
    log_weight_rep[n] = normal_rng(log_weight_hat_n, sigma);
  }
}
'
cat(stan_model_code)
```

```
##
## data {
##   int<lower=1> N;              // number of observations
##   vector[N] log_gest;          // log gestational age
##   vector[N] log_weight;        // log birth weight
##   int<lower=0,upper=1> z_i[N]; // preterm indicator (0 or 1)
## }
## parameters {
##   real beta_1;                 // intercept
##   real beta_2;                 // coef for log gestational age
##   real beta_3;                 // coef for interaction term
##   real<lower=0> sigma;         // error sd for Gaussian likelihood
```

```
## }
## model {
##   // Convert z_i to a vector for element-wise multiplication
##   vector[N] z_i_vec = to_vector(z_i);
##
##   // Log-likelihood
##   target += normal_lpdf(log_weight | beta_1 + beta_2 * log_gest + beta_3 * log_gest .* z_i_vec, sigma
##
##   // Log-priors
##   target += normal_lpdf(sigma | 0, 1)
##           + normal_lpdf(beta_1 | 0, 1)
##           + normal_lpdf(beta_2 | 0, 1)
##           + normal_lpdf(beta_3 | 0, 1);
## }
## generated quantities {
##   vector[N] log_lik;          // pointwise log-likelihood for LOO
##   vector[N] log_weight_rep;   // replications from posterior predictive dist
##
##   for (n in 1:N) {
##     real log_weight_hat_n = beta_1 + beta_2 * log_gest[n] + beta_3 * log_gest[n] * z_i[n];
##     log_lik[n] = normal_lpdf(log_weight[n] | log_weight_hat_n, sigma);
##     log_weight_rep[n] = normal_rng(log_weight_hat_n, sigma);
##   }
## }
```

```
ds <- ds %>%
  mutate(
    preterm = if_else(gest < 32, 1, 0), # Create preterm indicator
    log_weight = log(birthweight), # Take log of birthweight
    log_gest = (log(gest) - mean(log(gest)))/sd(log(gest)) # Center and scale log gest
  )

# Prepare your data for Stan
N <- nrow(ds)
log_weight <- ds$log_weight
log_gest <- ds$log_gest
z_i <- ds$preterm

stan_data <- list(N = N, log_gest = log_gest, log_weight = log_weight, z_i = z_i)

# Define your Stan model code (make sure to define 'stan_model_code' with Model 2)
# stan_model_code <- ' ... '

# Fit the model
mod2 <- stan(model_code = stan_model_code, data = stan_data, iter = 500, chains = 4,seed = 243)
```

```
## Trying to compile a simple C file

## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## using C compiler: 'Apple clang version 14.0.3 (clang-1403.0.22.14.1)'
## using SDK: 'MacOSX13.3.sdk'
## clang -arch arm64 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Library/Framew
## In file included from <built-in>:1:
```

```
## In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/StanHeade
## In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen/
## In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen/
## /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen/include/Eigen/src/Cor
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen/include/Eigen/src/Cor
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/StanHeade
## In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen/
## /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen/include/Eigen/Core:96
## #include <complex>
##          ^~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000266 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 2.66 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 500 [  0%]  (Warmup)
## Chain 1: Iteration:  50 / 500 [ 10%]  (Warmup)
## Chain 1: Iteration: 100 / 500 [ 20%]  (Warmup)
## Chain 1: Iteration: 150 / 500 [ 30%]  (Warmup)
## Chain 1: Iteration: 200 / 500 [ 40%]  (Warmup)
## Chain 1: Iteration: 250 / 500 [ 50%]  (Warmup)
## Chain 1: Iteration: 251 / 500 [ 50%]  (Sampling)
## Chain 1: Iteration: 300 / 500 [ 60%]  (Sampling)
## Chain 1: Iteration: 350 / 500 [ 70%]  (Sampling)
## Chain 1: Iteration: 400 / 500 [ 80%]  (Sampling)
## Chain 1: Iteration: 450 / 500 [ 90%]  (Sampling)
## Chain 1: Iteration: 500 / 500 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.431 seconds (Warm-up)
## Chain 1:                0.376 seconds (Sampling)
## Chain 1:                0.807 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.00018 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 1.8 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 500 [  0%]  (Warmup)
## Chain 2: Iteration:  50 / 500 [ 10%]  (Warmup)
## Chain 2: Iteration: 100 / 500 [ 20%]  (Warmup)
```

```
## Chain 2: Iteration: 150 / 500 [ 30%]  (Warmup)
## Chain 2: Iteration: 200 / 500 [ 40%]  (Warmup)
## Chain 2: Iteration: 250 / 500 [ 50%]  (Warmup)
## Chain 2: Iteration: 251 / 500 [ 50%]  (Sampling)
## Chain 2: Iteration: 300 / 500 [ 60%]  (Sampling)
## Chain 2: Iteration: 350 / 500 [ 70%]  (Sampling)
## Chain 2: Iteration: 400 / 500 [ 80%]  (Sampling)
## Chain 2: Iteration: 450 / 500 [ 90%]  (Sampling)
## Chain 2: Iteration: 500 / 500 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.406 seconds (Warm-up)
## Chain 2:                0.381 seconds (Sampling)
## Chain 2:                0.787 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0.000154 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 1.54 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:   1 / 500 [  0%]  (Warmup)
## Chain 3: Iteration:  50 / 500 [ 10%]  (Warmup)
## Chain 3: Iteration: 100 / 500 [ 20%]  (Warmup)
## Chain 3: Iteration: 150 / 500 [ 30%]  (Warmup)
## Chain 3: Iteration: 200 / 500 [ 40%]  (Warmup)
## Chain 3: Iteration: 250 / 500 [ 50%]  (Warmup)
## Chain 3: Iteration: 251 / 500 [ 50%]  (Sampling)
## Chain 3: Iteration: 300 / 500 [ 60%]  (Sampling)
## Chain 3: Iteration: 350 / 500 [ 70%]  (Sampling)
## Chain 3: Iteration: 400 / 500 [ 80%]  (Sampling)
## Chain 3: Iteration: 450 / 500 [ 90%]  (Sampling)
## Chain 3: Iteration: 500 / 500 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.442 seconds (Warm-up)
## Chain 3:                0.368 seconds (Sampling)
## Chain 3:                0.81 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0.000153 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 1.53 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:   1 / 500 [  0%]  (Warmup)
## Chain 4: Iteration:  50 / 500 [ 10%]  (Warmup)
## Chain 4: Iteration: 100 / 500 [ 20%]  (Warmup)
## Chain 4: Iteration: 150 / 500 [ 30%]  (Warmup)
## Chain 4: Iteration: 200 / 500 [ 40%]  (Warmup)
## Chain 4: Iteration: 250 / 500 [ 50%]  (Warmup)
## Chain 4: Iteration: 251 / 500 [ 50%]  (Sampling)
```

```
## Chain 4: Iteration: 300 / 500 [ 60%]  (Sampling)
## Chain 4: Iteration: 350 / 500 [ 70%]  (Sampling)
## Chain 4: Iteration: 400 / 500 [ 80%]  (Sampling)
## Chain 4: Iteration: 450 / 500 [ 90%]  (Sampling)
## Chain 4: Iteration: 500 / 500 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.392 seconds (Warm-up)
## Chain 4:                0.354 seconds (Sampling)
## Chain 4:                0.746 seconds (Total)
## Chain 4:
```

```
## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#bulk-ess
```

```
## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quanti
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#tail-ess
```

```
# Summarize the results
summary(mod2)$summary[c("beta_1", "beta_2", "beta_3", "sigma"), ]
```

```
##              mean      se_mean          sd       2.5%        25%       50%
## beta_1 1.17061716 8.875827e-05 0.002777351 1.16498801 1.16870684 1.1706511
## beta_2 0.10191878 1.453365e-04 0.003699757 0.09475329 0.09950080 0.1017936
## beta_3 0.09157266 2.399446e-04 0.005503417 0.08019250 0.08793668 0.0917572
## sigma  0.16286774 1.059498e-04 0.001793397 0.15918758 0.16165598 0.1629723
##               75%     97.5%    n_eff      Rhat
## beta_1 1.17246244 1.1758366 979.1383 0.9979351
## beta_2 0.10441540 0.1096713 648.0330 1.0004655
## beta_3 0.09514565 0.1021709 526.0695 1.0022442
## sigma  0.16401570 0.1663456 286.5186 1.0120029
```

The coefficient estimate for the interaction term in Stan model, beta_3, is 0.09157, with a 95% credible interval ranging from approximately 0.08019 to 0.10217. The positive beta_3 estimate suggests that for preterm babies, as gestational age increases, there is an additional positive effect on birth weight beyond the effect seen in full-term babies.

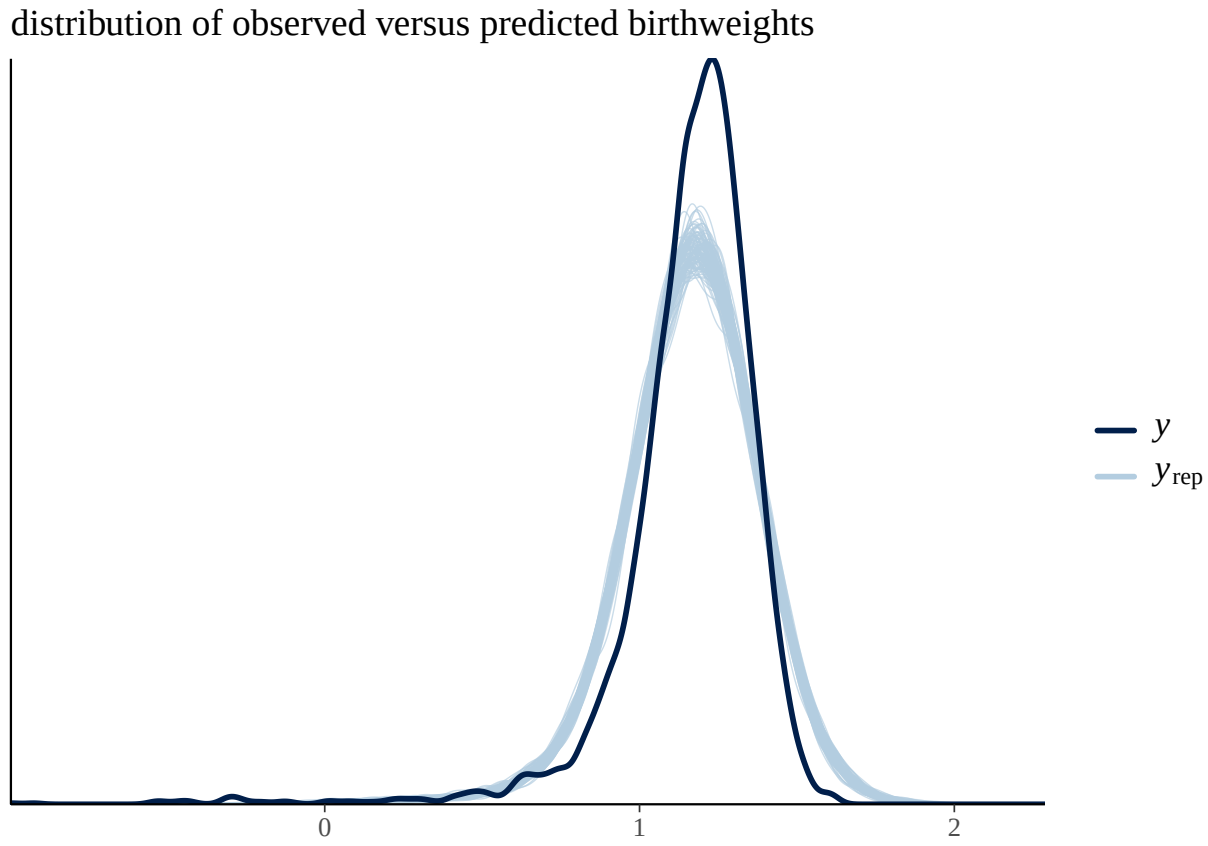
# PPCs

Now we've run two candidate models let's do some posterior predictive checks. The `bayesplot` package has a lot of inbuilt graphing functions to do this. For example, let's plot the distribution of our data (y) against 100 different datasets drawn from the posterior predictive distribution:

```
set.seed(1856)
y <- ds$log_weight
yrep1 <-rstan::extract(mod1)[["log_weight_rep"]]
dim(yrep1)
```

```
## [1] 1000 3842
```

```
samp100 <- sample(nrow(yrep1), 100)
ppc_dens_overlay(y, yrep1[samp100, ])  + ggtitle("distribution of observed versus predicted birthweights
```




## Question 6

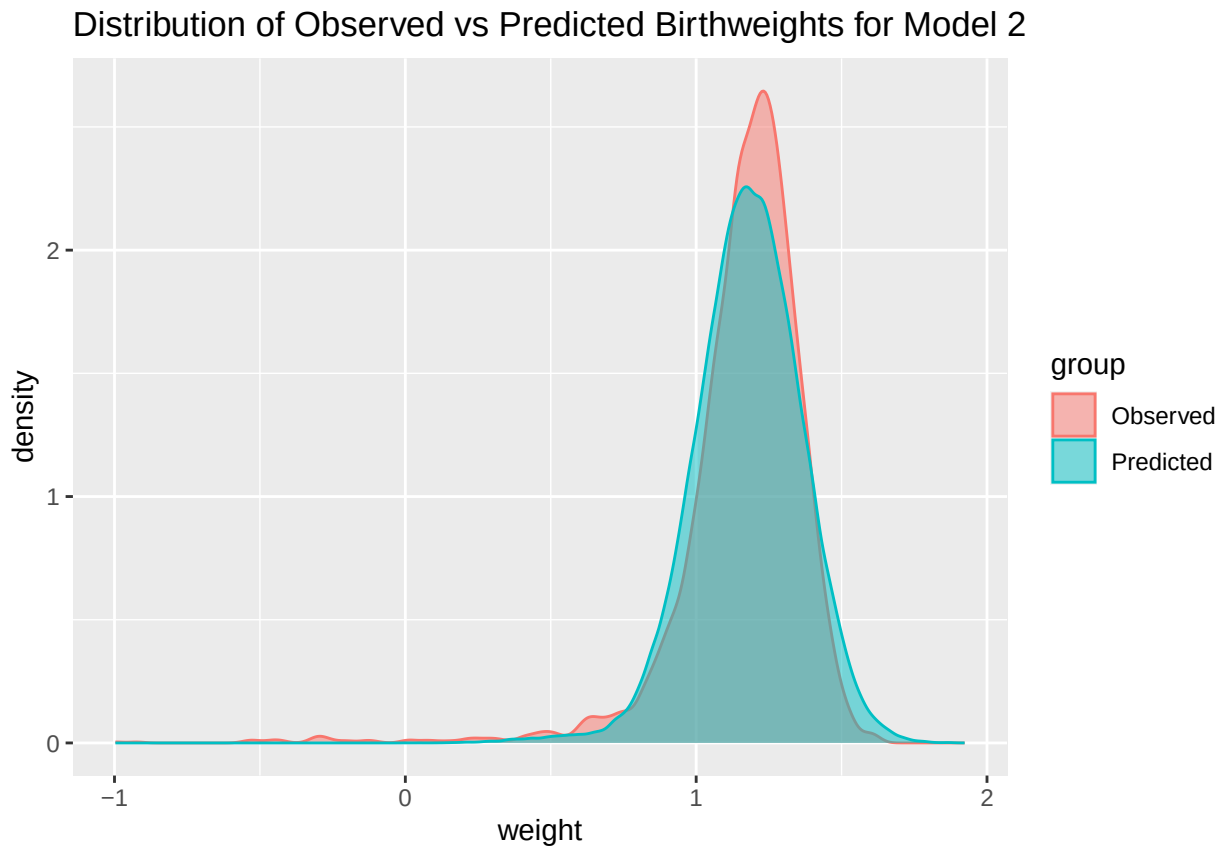
Make a similar plot to the one above but for Model 2, and **not** using the bayes plot in built function (i.e. do it yourself just with `geom_density`)

```
set.seed(1856)
y <- ds$log_weight
yrep2 <- rstan::extract(mod2)[["log_weight_rep"]] # Replace with mod2 extraction
dim(yrep2)
```

```
## [1] 1000 3842
```

```
samp100 <- sample(nrow(yrep2), 100)

# Creating a data frame for plotting
observed_data <- data.frame(weight = y, group = 'Observed')
predicted_data <- data.frame(weight = as.vector(yrep2[, samp100]), group = 'Predicted')

# Binding the observed and predicted data together
combined_data <- rbind(observed_data, predicted_data)

# Plotting the densities using ggplot2
ggplot(combined_data, aes(x = weight, fill = group, color = group)) +
```

```
  geom_density(alpha = 0.5) +
  ggtitle("Distribution of Observed vs Predicted Birthweights for Model 2")
```



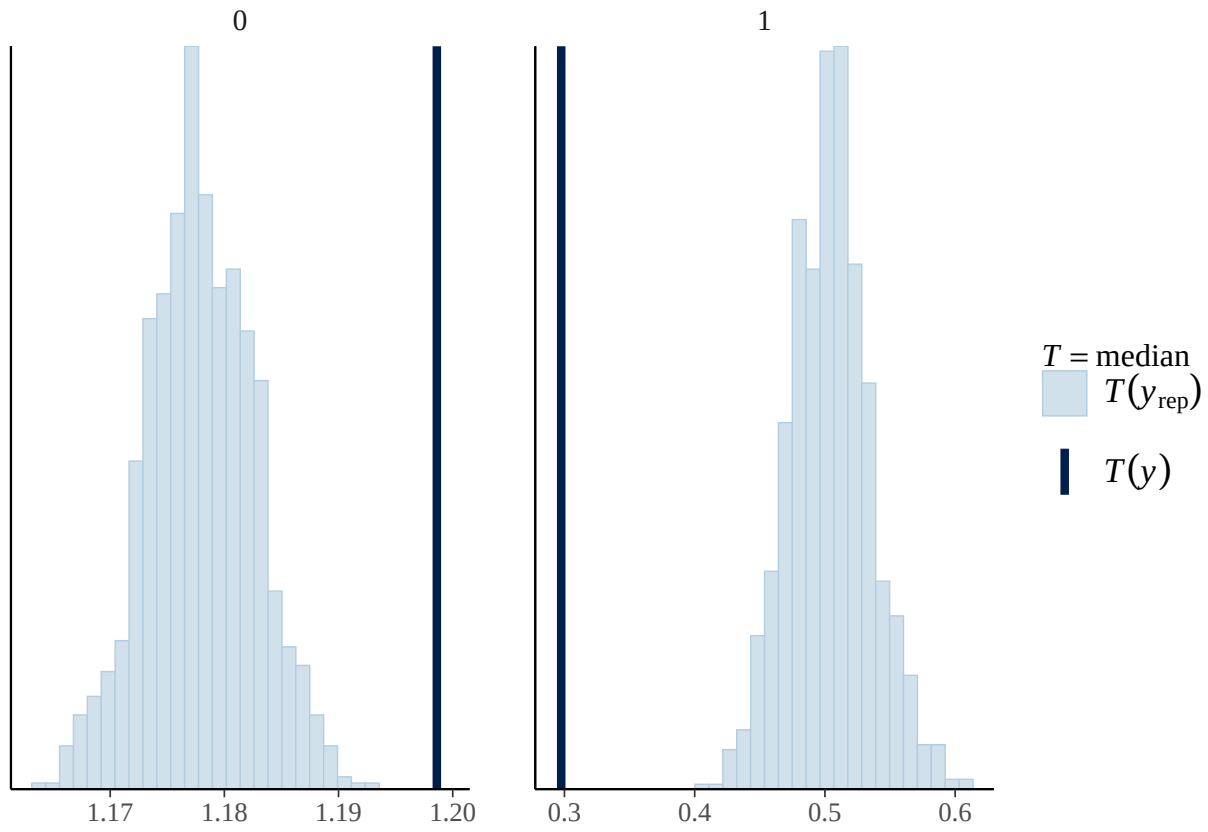

## Test statistics

We can also look at some summary statistics in the PPD versus the data, again either using `bayesplot` – the function of interest is `ppc_stat` or `ppc_stat_grouped` – or just doing it ourselves using ggplot.

E.g. medians by prematurity for Model 1

```
ppc_stat_grouped(ds$log_weight, yrep1, group = ds$preterm, stat = 'median')
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




## Question 7

Use a test statistic of the proportion of births under 2.5kg. Calculate the test statistic for the data, and the posterior predictive samples for both models, and plot the comparison (one plot per model).

```
# Define the test statistic function
test_statistic <- function(weights) {
  sum(weights < 2.5) / length(weights)
}

# Calculate the test statistic for the observed data
observed_stat <- test_statistic(ds$birthweight)

# Extract the posterior predictive samples for both models
yrep1 <- rstan::extract(mod1)[["log_weight_rep"]]
yrep2 <- rstan::extract(mod2)[["log_weight_rep"]]

# Calculate the test statistic for the posterior predictive samples
test_stats_mod1 <- apply(exp(yrep1), 2, test_statistic)
test_stats_mod2 <- apply(exp(yrep2), 2, test_statistic)

# Create a data frame for plotting
test_stats_data_mod1 <- data.frame(statistic = test_stats_mod1, model = 'Model 1')
test_stats_data_mod2 <- data.frame(statistic = test_stats_mod2, model = 'Model 2')
```

```r
# Combine the data for plotting
combined_test_stats_data <- rbind(
  test_stats_data_mod1,
  test_stats_data_mod2,
  data.frame(statistic = observed_stat, model = 'Observed')
)

# Plot the comparison for Model 1
ggplot(combined_test_stats_data %>% filter(model %in% c('Model 1', 'Observed')), aes(x = statistic, fil
  geom_density(alpha = 0.5) +
  ggtitle('Comparison of Test Statistic for Model 1')
```

```
## Warning: Groups with fewer than two data points have been dropped.
```

```
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning
## -Inf
```


```r
# Plot the comparison for Model 2
ggplot(combined_test_stats_data %>% filter(model %in% c('Model 2', 'Observed')), aes(x = statistic, fil
  geom_density(alpha = 0.5) +
  ggtitle('Comparison of Test Statistic for Model 2')
```

```
## Warning: Groups with fewer than two data points have been dropped.
## no non-missing arguments to max; returning -Inf
```


# LOO

Finally let's calculate the LOO elpd for each model and compare. The first step of this is to get the point-wise log likelihood estimates from each model:

```
loglik1 <- rstan::extract(mod1)[["log_lik"]]
```

And then we can use these in the `loo` function to get estimates for the elpd. Note the `save_psis = TRUE` argument saves the calculation for each simulated draw, which is needed for the LOO-PIT calculation below.

```
loo1 <- loo(loglik1, save_psis = TRUE)
```

```
## Warning: Relative effective sample sizes ('r_eff' argument) not specified.
## For models fit with MCMC, the reported PSIS effective sample sizes and
## MCSE estimates will be over-optimistic.
```

Look at the output:

```
loo1
```

```
##
## Computed from 1000 by 3842 log-likelihood matrix
##
##          Estimate    SE
## elpd_loo   1377.4  72.6
## p_loo         9.3   1.4
```

```
## looic     -2754.8 145.2
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

## Question 8

Get the LOO estimate of elpd for Model 2 and compare the two models with the `loo_compare` function. Interpret the results.

We can also compare the LOO-PIT of each of the models to standard uniforms. For example for Model 1:

```
loglik2 <- rstan::extract(mod2)[["log_lik"]]
loo2 <- loo(loglik2, save_psis = TRUE)
```

```
## Warning: Relative effective sample sizes ('r_eff' argument) not specified.
## For models fit with MCMC, the reported PSIS effective sample sizes and
## MCSE estimates will be over-optimistic.
```

```
loo2
```

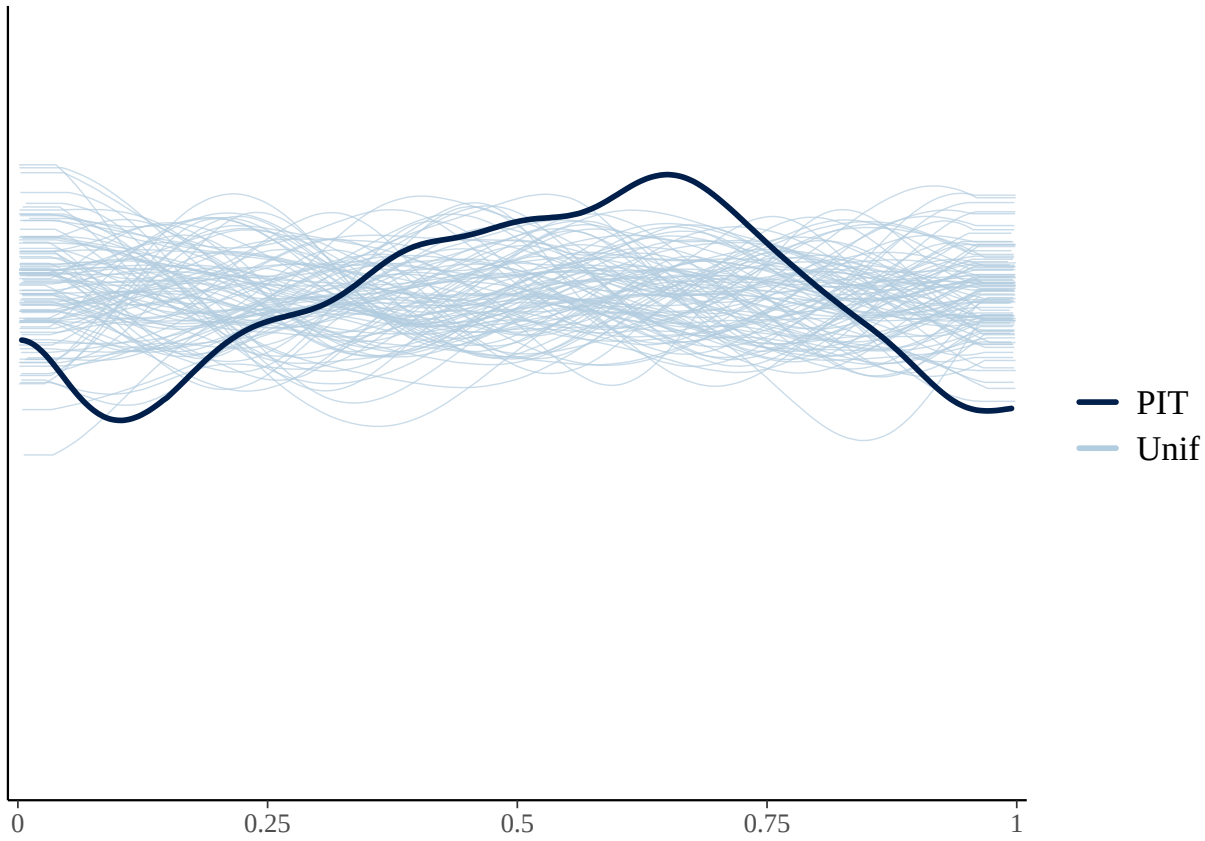
```
##
## Computed from 1000 by 3842 log-likelihood matrix
##
##          Estimate    SE
## elpd_loo   1517.0  72.8
## p_loo        11.6   1.6
## looic     -3034.0 145.5
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

```
loo_compare(loo1, loo2)
```

```
##        elpd_diff se_diff
## model2    0.0       0.0
## model1 -139.6      36.0
```

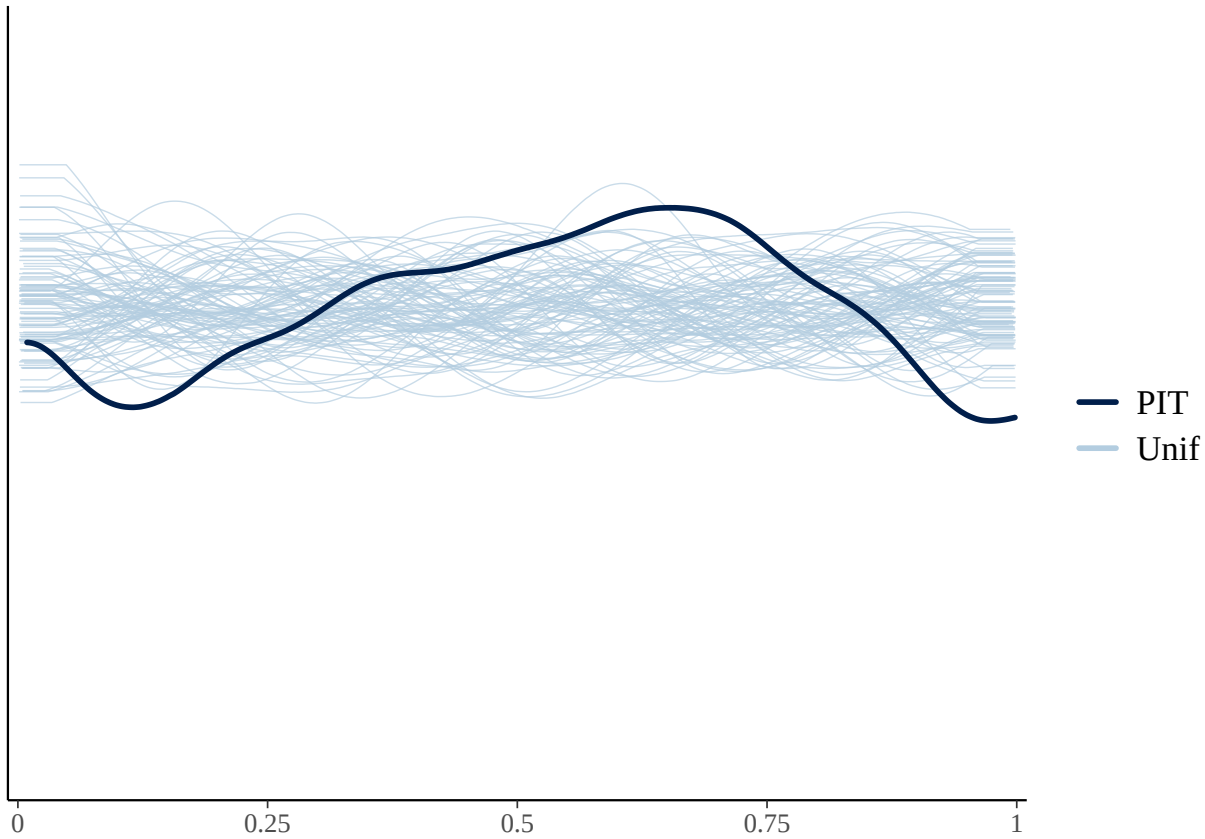
```
ppc_loo_pit_overlay(yrep = yrep1, y = y, lw = weights(loo1$psis_object))
```

```
## NOTE: The kernel density estimate assumes continuous observations and is not optimal for discrete ob
```




```
ppc_loo_pit_overlay(yrep = yrep2, y = y, lw = weights(loo2$psis_object))
```

```
## NOTE: The kernel density estimate assumes continuous observations and is not optimal for discrete ob
```




Since model2 has a better (higher) elpd_loo, it is considered to be the model with better out-of-sample predictive performance compared to model1.