Visualizing the Bayesian Workflow

Monica Alexander

March 2 2021

Contents

1 Introduction

This lab will be looking at trying to replicate some of the visualizations in the lecture notes, involving prior and posterior predictive checks, and LOO model comparisons.

The dataset is a 0.1% of all births in the US in 2017. I've pulled out a few different variables, but as in the lecture, we'll just focus on birth weight and gestational age.

2 The data

Read it in, along with all our packages.

```
# the ol' faves
library(tidyverse)
library(tidyr)
library(here)
# for bayes stuff
library(rstan)
library(bayesplot) # PPCs
```

```
library(loo) # does what it says on the packet
library(tidybayes) # may or may not be needed, but I like it

ds <- read_rds(here("data","births_2017_sample.RDS"))
head(ds)
```

```
## # A tibble: 6 x 8
##   mager mracehisp meduc   bmi sex   combgest  dbwt ilive
##   <dbl>     <dbl> <dbl> <dbl> <chr>    <dbl> <dbl> <chr>
## 1    16         2     2  23   M           39  3.18 Y
## 2    25         7     2  43.6 M           40  4.14 Y
## 3    27         2     3  19.5 F           41  3.18 Y
## 4    26         1     3  21.5 F           36  3.40 Y
## 5    28         7     2  40.6 F           34  2.71 Y
## 6    31         7     3  29.3 M           35  3.52 Y
```

Brief overview of variables:

- `mager` mum's age
- `mracehisp` mum's race/ethnicity see here for codes: https://data.nber.org/natality/2017/natl2017.pdf page 15
- `meduc` mum's education see here for codes: https://data.nber.org/natality/2017/natl2017.pdf page 16
- `bmi` mum's bmi
- `sex` baby's sex
- `combgest` gestational age in weeks
- `dbwt` birth weight in kg
- `ilive` alive at time of report y/n/ unsure

I'm going to rename some variables, remove any observations with missing gestational age or birth weight, restrict just to babies that were alive, and make a preterm variable.

```
ds <- ds %>%
  rename(birthweight = dbwt, gest = combgest) %>%
  mutate(preterm = ifelse(gest<32, "Y", "N")) %>%
  filter(ilive=="Y",gest< 99, birthweight<9.999)
```

2.1 Question 1

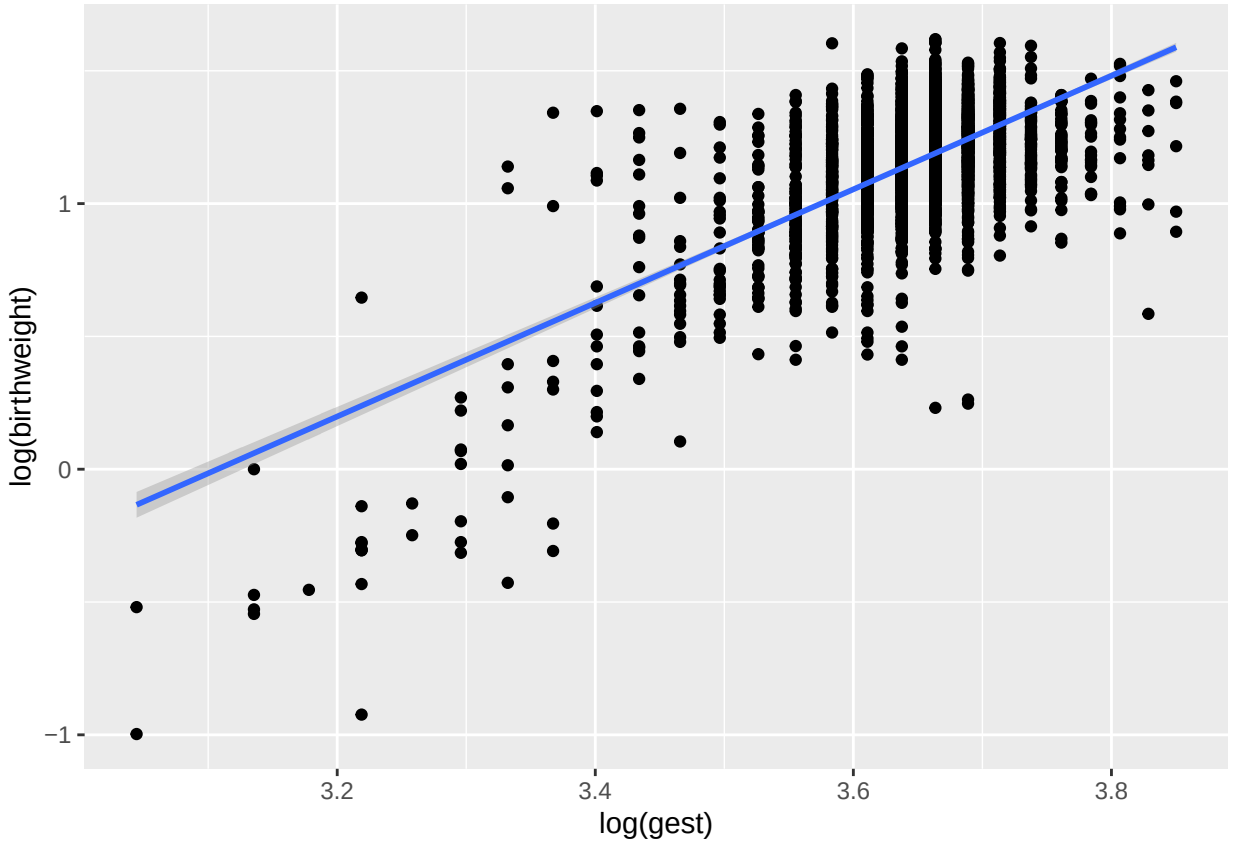
Should sound familiar by now: use plots or tables to show three interesting observations about the data. Remember:

- 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

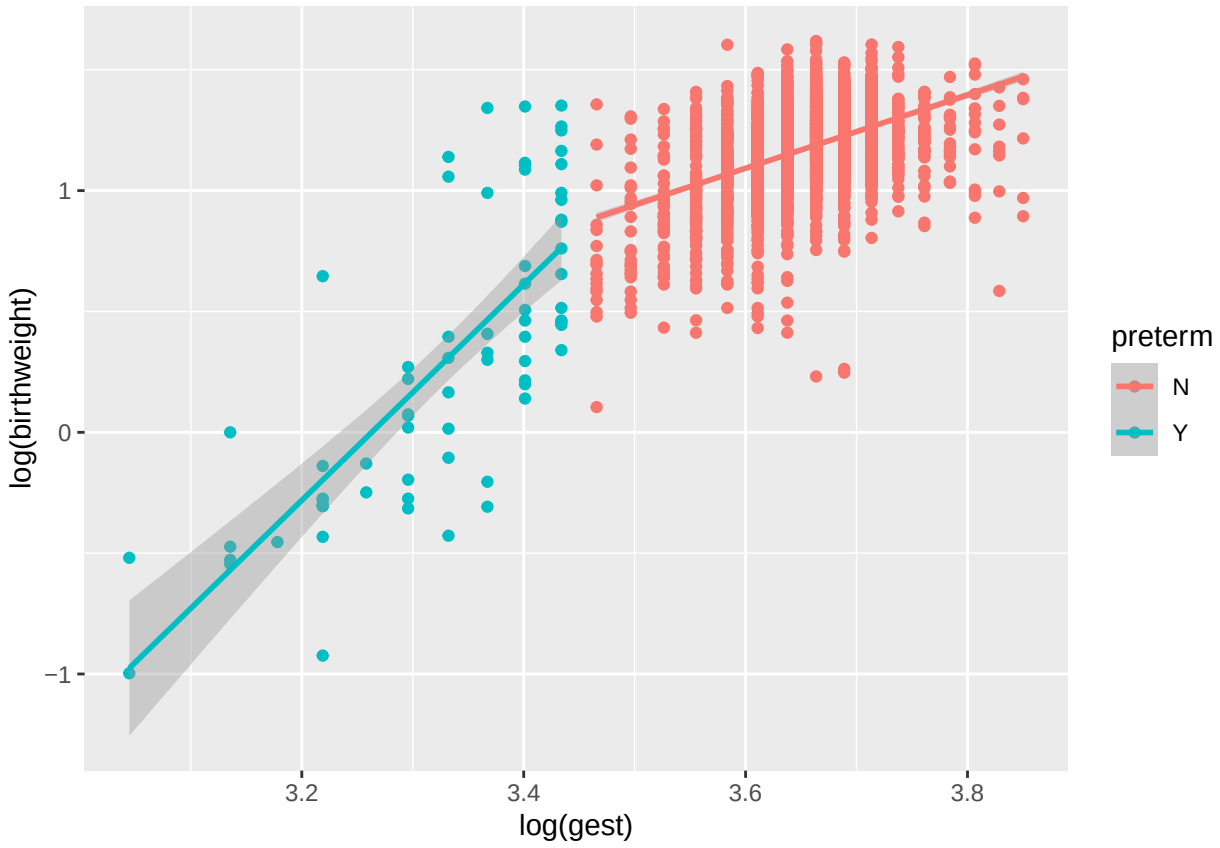
###Answer:

- First, from the first plot we could see the log(gestational weeks) and log(birth weights) have positive linear relation. The weights of birth increase significantly as the weeks are larger.
- Second, from the second plot we could see the after 32 weeks. The weights gain linear model have a smaller sloper as the weights gain before 32 weeks. Means the major weights gain happened before 32 weeks.
- However, the preterm samples are significantly imbalanced. We need to be careful on the choice of models.

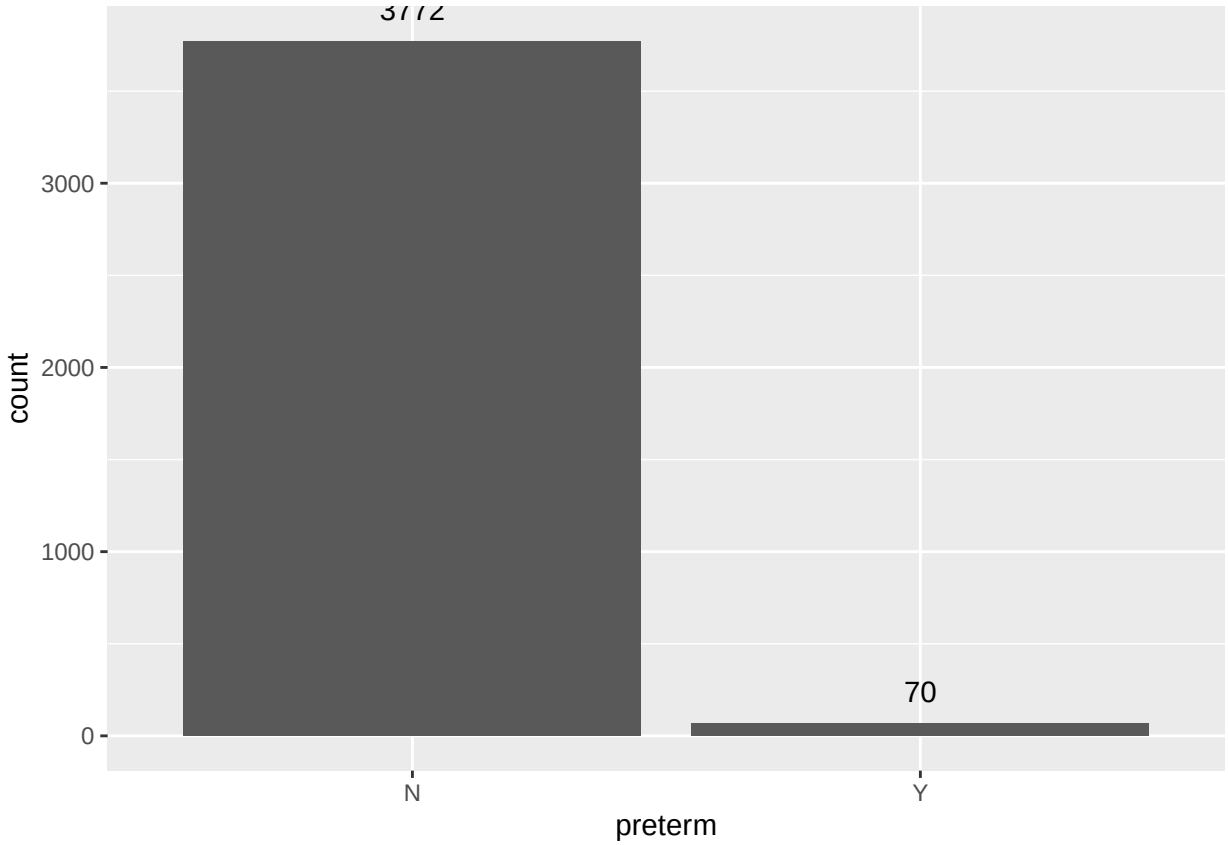
```
ggplot(ds,aes(log(gest), log(birthweight))) +
  geom_point() +
  geom_smooth(method='lm', formula= y~x)
```




```
ggplot(ds,aes(log(gest), log(birthweight), color = preterm)) +
  geom_point() +
  geom_smooth(method='lm', formula= y~x)
```




```
ggplot(ds, aes(x=preterm)) +
  geom_bar() +
  geom_text(stat='count', aes(label=..count..), vjust=-1)
```



Feel free to replicate one of the scatter plots in the lectures as one of the interesting observations, as those form the basis of our models.

3 The model

As in lecture, we will look at two candidate models

Model 1 has log birth weight as a function of log gestational age

$$\log(y_i) \sim N(\beta_1 + \beta_2 \log(x_i), \sigma^2)$$

Model 2 has an interaction term between gestation and prematurity

$$\log(y_i) \sim N(\beta_1 + \beta_2 \log(x_i) + \beta_2 z_i + \beta_4 \log(x_i) z_i, \sigma^2)$$

- y_i is weight in kg
- x_i is gestational age in weeks, CENTERED AND STANDARDIZED
- z_i is preterm (0 or 1, if gestational age is less than 32 weeks)

4 Prior predictive checks

Let's put some weakly informative priors on all parameters i.e. for the βs

$$\beta \sim N(0, 1)$$

and for σ

$$\sigma \sim N^{+}(0, 1)$$

where the plus means positive values only i.e. Half Normal.

Let's check to see what the resulting distribution of birth weights look like given Model 1 and the priors specified above, assuming we had no data on birth weight (but observations of gestational age).

4.1 Question 2

For Model 1, simulate values of βs and σ based on the priors above. Use these values to simulate (log) birth weights from the likelihood specified in Model 1, based on the set of observed gestational weights. Plot the resulting distribution of simulated (log) birth weights. Do 1000 simulations. **Remember the gestational weights should be centered and standardized**.

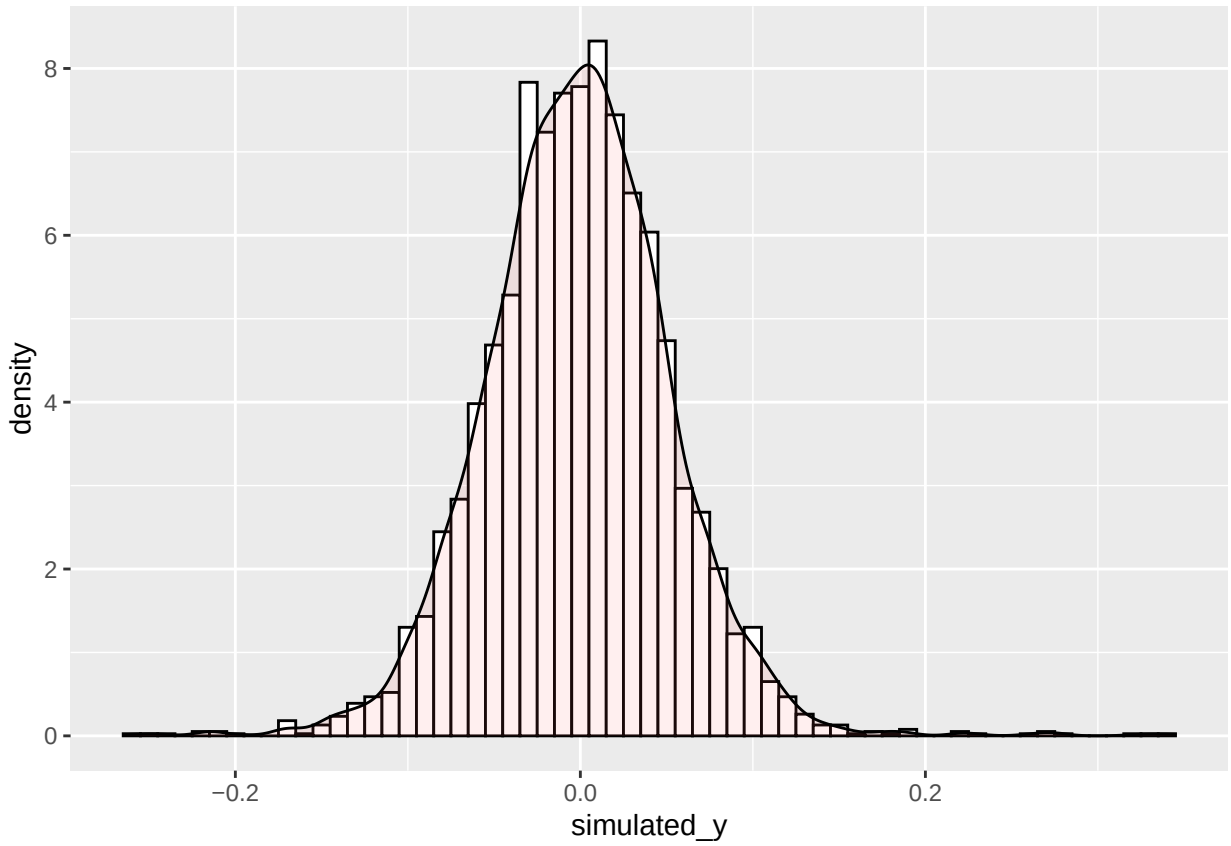
```
sample_cus <- function(data){
  beta_1 = rnorm(1)
  beta_2 = rnorm(1)
  sigma = abs(rnorm(1))
  result = rnorm(1, beta_1 + beta_2 * data, sigma)
}

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

log_y = rep(0, 3842)
for(counter in c(1:1000)){
  log_y = log_y + sapply(ds$log_gest_c, sample_cus)
}

ds$simulated_y = log_y/1000

ggplot(ds,aes(x = simulated_y)) +
  geom_histogram(aes(y=..density..),
                 binwidth=.01,
                 colour="black", fill="white") +
    geom_density(alpha=.1, fill="#FF6666")
```



5 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))
N = nrow(ds)
# 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 = "models/simple_weight.stan",
             iter = 500,
             seed = 243)
```

```
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -mmacosx-version-min=10.13 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/inclu
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/include/Eigen/src/Core/util/
```

```
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/include/Eigen/Core:96:10: fa
## #include <complex>
##          ^~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL 'simple_weight' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000495 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 4.95 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: 0.426203 seconds (Warm-up)
## Chain 1:                0.336302 seconds (Sampling)
## Chain 1:                0.762505 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'simple_weight' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.000183 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 1.83 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: 0.379252 seconds (Warm-up)
## Chain 2:                0.346475 seconds (Sampling)
## Chain 2:                0.725727 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'simple_weight' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0.000184 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 1.84 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: 0.383702 seconds (Warm-up)
## Chain 3:                0.339237 seconds (Sampling)
## Chain 3:                0.722939 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'simple_weight' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0.000132 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 1.32 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: 0.366536 seconds (Warm-up)
## Chain 4:                0.329182 seconds (Sampling)
## Chain 4:                0.695718 seconds (Total)
## Chain 4:
```

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

```
##              mean      se_mean          sd      2.5%       25%       50%
## beta[1] 1.1626250 7.634607e-05 0.002583881 1.1575321 1.1609497 1.1626383
## beta[2] 0.1436183 8.105504e-05 0.002791943 0.1380281 0.1417563 0.1436199
## sigma   0.1689127 1.051837e-04 0.001979909 0.1650908 0.1676042 0.1688619
##               75%     97.5%     n_eff      Rhat
## beta[1] 1.1643919 1.1677313 1145.4383 0.9970543
## beta[2] 0.1455075 0.1489575 1186.4598 0.9984953
## sigma   0.1701148 0.1728405  354.3181 1.0046933
```

5.1 Question 3

Write a stan model to run Model 2, and run it. There are three options (probably more) to alter the existing stan code

1. add in prematurity and interaction betas to the equation, pass the interaction covariate in as data
2. add in prematurity and interaction betas to the equation, calculate the interaction in a `transformed data` block in the stan model (put it after the data block). this would look something like
3. change the whole format of the model to be similar to the kids examples from last time where the design matrix was being inputted, rather than individual variables.

To run the model, your code should look something like this (set `eval = T` to run)

```
preterm <- ifelse(ds$preterm=="Y", 1, 0)
# add preterm to list
# note if you are also inputting interaction you will need to add this
stan_data[["preterm"]] <- preterm
stan_data[["inter"]] <- preterm * ds$log_gest_c

mod2 <- stan(data = stan_data,
             file = "models/simple_weight_preterm_int.stan",
             iter = 300,
             seed = 243)
```

```
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -mmacosx-version-min=10.13 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/inclu
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
##                 ^
##                 ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/StanHeaders/incl
```

```
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/include/Eigen/Core:96:10: fa
## #include <complex>
##          ^~~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL 'simple_weight_preterm_int' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.001339 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 13.39 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 300 [  0%]  (Warmup)
## Chain 1: Iteration:  30 / 300 [ 10%]  (Warmup)
## Chain 1: Iteration:  60 / 300 [ 20%]  (Warmup)
## Chain 1: Iteration:  90 / 300 [ 30%]  (Warmup)
## Chain 1: Iteration: 120 / 300 [ 40%]  (Warmup)
## Chain 1: Iteration: 150 / 300 [ 50%]  (Warmup)
## Chain 1: Iteration: 151 / 300 [ 50%]  (Sampling)
## Chain 1: Iteration: 180 / 300 [ 60%]  (Sampling)
## Chain 1: Iteration: 210 / 300 [ 70%]  (Sampling)
## Chain 1: Iteration: 240 / 300 [ 80%]  (Sampling)
## Chain 1: Iteration: 270 / 300 [ 90%]  (Sampling)
## Chain 1: Iteration: 300 / 300 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 1.08719 seconds (Warm-up)
## Chain 1:                0.880776 seconds (Sampling)
## Chain 1:                1.96796 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'simple_weight_preterm_int' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.000333 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 3.33 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 300 [  0%]  (Warmup)
## Chain 2: Iteration:  30 / 300 [ 10%]  (Warmup)
## Chain 2: Iteration:  60 / 300 [ 20%]  (Warmup)
## Chain 2: Iteration:  90 / 300 [ 30%]  (Warmup)
## Chain 2: Iteration: 120 / 300 [ 40%]  (Warmup)
## Chain 2: Iteration: 150 / 300 [ 50%]  (Warmup)
## Chain 2: Iteration: 151 / 300 [ 50%]  (Sampling)
## Chain 2: Iteration: 180 / 300 [ 60%]  (Sampling)
## Chain 2: Iteration: 210 / 300 [ 70%]  (Sampling)
## Chain 2: Iteration: 240 / 300 [ 80%]  (Sampling)
## Chain 2: Iteration: 270 / 300 [ 90%]  (Sampling)
## Chain 2: Iteration: 300 / 300 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 1.03733 seconds (Warm-up)
## Chain 2:                0.823427 seconds (Sampling)
```

```
## Chain 2:                1.86076 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'simple_weight_preterm_int' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0.000334 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 3.34 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:   1 / 300 [  0%]  (Warmup)
## Chain 3: Iteration:  30 / 300 [ 10%]  (Warmup)
## Chain 3: Iteration:  60 / 300 [ 20%]  (Warmup)
## Chain 3: Iteration:  90 / 300 [ 30%]  (Warmup)
## Chain 3: Iteration: 120 / 300 [ 40%]  (Warmup)
## Chain 3: Iteration: 150 / 300 [ 50%]  (Warmup)
## Chain 3: Iteration: 151 / 300 [ 50%]  (Sampling)
## Chain 3: Iteration: 180 / 300 [ 60%]  (Sampling)
## Chain 3: Iteration: 210 / 300 [ 70%]  (Sampling)
## Chain 3: Iteration: 240 / 300 [ 80%]  (Sampling)
## Chain 3: Iteration: 270 / 300 [ 90%]  (Sampling)
## Chain 3: Iteration: 300 / 300 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 1.05746 seconds (Warm-up)
## Chain 3:                1.08989 seconds (Sampling)
## Chain 3:                2.14735 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'simple_weight_preterm_int' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0.000362 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 3.62 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:   1 / 300 [  0%]  (Warmup)
## Chain 4: Iteration:  30 / 300 [ 10%]  (Warmup)
## Chain 4: Iteration:  60 / 300 [ 20%]  (Warmup)
## Chain 4: Iteration:  90 / 300 [ 30%]  (Warmup)
## Chain 4: Iteration: 120 / 300 [ 40%]  (Warmup)
## Chain 4: Iteration: 150 / 300 [ 50%]  (Warmup)
## Chain 4: Iteration: 151 / 300 [ 50%]  (Sampling)
## Chain 4: Iteration: 180 / 300 [ 60%]  (Sampling)
## Chain 4: Iteration: 210 / 300 [ 70%]  (Sampling)
## Chain 4: Iteration: 240 / 300 [ 80%]  (Sampling)
## Chain 4: Iteration: 270 / 300 [ 90%]  (Sampling)
## Chain 4: Iteration: 300 / 300 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 1.17054 seconds (Warm-up)
## Chain 4:                1.13653 seconds (Sampling)
## Chain 4:                2.30707 seconds (Total)
## Chain 4:
```

```r
summary(mod2)$summary[c("beta[1]", "beta[2]", "beta[3]", "beta[4]", "sigma"),]
```

```
##              mean      se_mean          sd       2.5%        25%       50%
## beta[1] 1.1697017 9.990786e-05 0.002633970 1.16475367 1.16786294 1.1697858
## beta[2] 0.1021125 1.494450e-04 0.003527920 0.09496695 0.09997127 0.1020169
## beta[3] 0.5627788 6.071466e-03 0.062298115 0.42410107 0.52167902 0.5625263
## beta[4] 0.1980668 1.271243e-03 0.013394524 0.17181907 0.18894727 0.1979953
## sigma   0.1612023 9.179934e-05 0.001876482 0.15775970 0.15991641 0.1611336
##               75%     97.5%    n_eff     Rhat
## beta[1] 1.1714700 1.1751558 695.0599 1.002326
## beta[2] 0.1043538 0.1087593 557.2815 1.004405
## beta[3] 0.6088776 0.6834738 105.2841 1.029855
## beta[4] 0.2079308 0.2237020 111.0191 1.034787
## sigma   0.1626033 0.1648317 417.8398 1.002012
```

5.2 Question 4

For reference I have uploaded some model 2 results. Check your results are similar.

Answer: The mean results I acquired is beta[1] 1.1697017, beta[2] 0.1021125, beta[3] 0.5627788, beta[4] 0.1980668, sigma 0.1612023. They are quite clost to the real results.

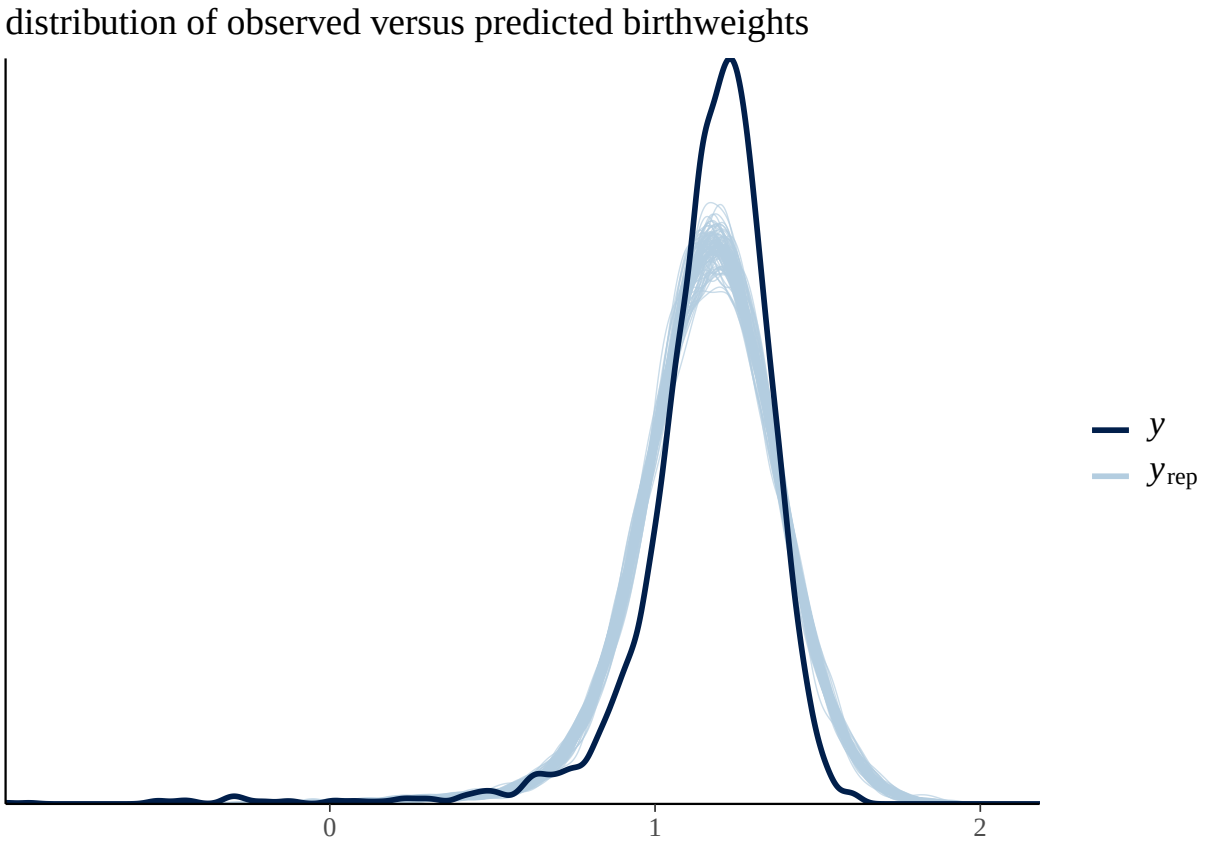
```r
load(here("output", "mod2.Rda"))
summary(mod2)$summary[c(paste0("beta[", 1:4, "]"), "sigma"),]
```

```
##              mean      se_mean          sd       2.5%        25%       50%
## beta[1] 1.1697241 1.385590e-04 0.002742186 1.16453578 1.16767109 1.1699278
## beta[2] 0.5563133 5.835253e-03 0.058054991 0.43745504 0.51708255 0.5561553
## beta[3] 0.1020960 1.481816e-04 0.003669476 0.09459462 0.09997153 0.1020339
## beta[4] 0.1967671 1.129799e-03 0.012458398 0.17164533 0.18817091 0.1974114
## sigma   0.1610727 9.950037e-05 0.001782004 0.15784213 0.15978020 0.1610734
##               75%     97.5%     n_eff      Rhat
## beta[1] 1.1716235 1.1750167 391.67359 1.0115970
## beta[2] 0.5990427 0.6554967  98.98279 1.0088166
## beta[3] 0.1044230 0.1093843 613.22428 0.9978156
## beta[4] 0.2064079 0.2182454 121.59685 1.0056875
## sigma   0.1623019 0.1646189 320.75100 1.0104805
```

6 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:

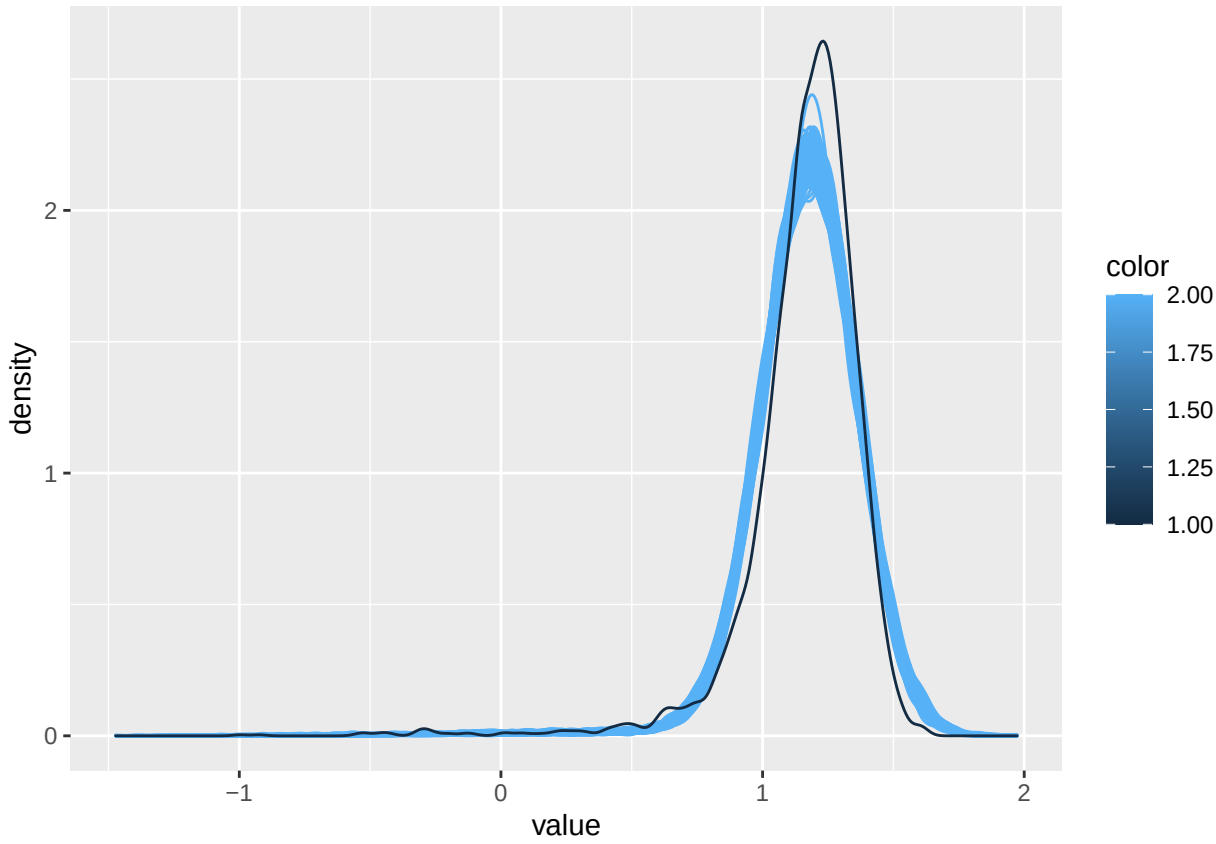
```r
set.seed(1856)
y <- ds$log_weight
yrep1 <- extract(mod1)[["log_weight_rep"]]
yrep2 <- extract(mod2)[["log_weight_rep"]] # will need mod2 for later
samp100 <- sample(nrow(yrep1), 100)
ppc_dens_overlay(y, yrep1[samp100, ])  + ggtitle("distribution of observed versus predicted birthweights
```



6.1 Question 5

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`)

```
samp100 <- sample(nrow(yrep2), 100)
simulated = yrep2[samp100, ]
plot_data = as.data.frame(t(rbind(y, simulated)))
test_long = gather(plot_data)
test_long$color = 2
test_long[test_long$key == "y", 3] = 1
ggplot(test_long, aes(x = value, color = color, group = key)) +
    geom_density()
```

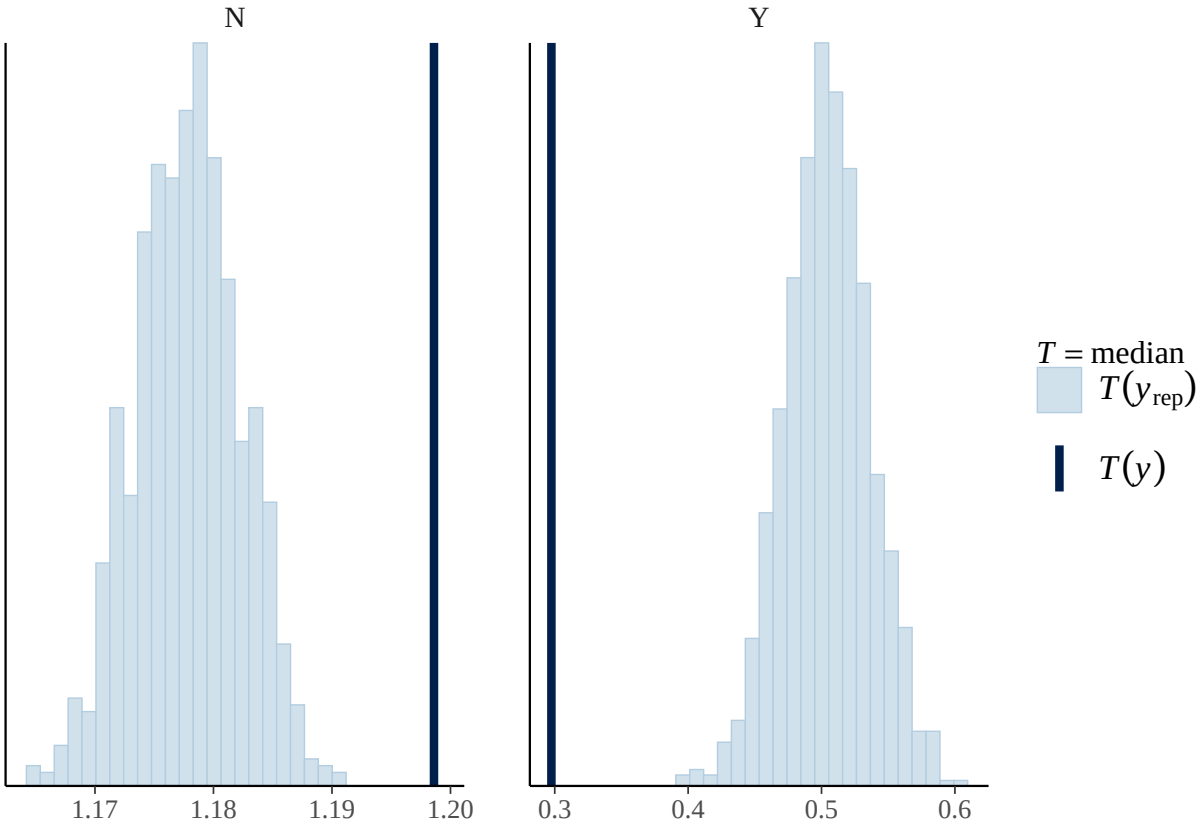


6.2 Test statistics

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')
```

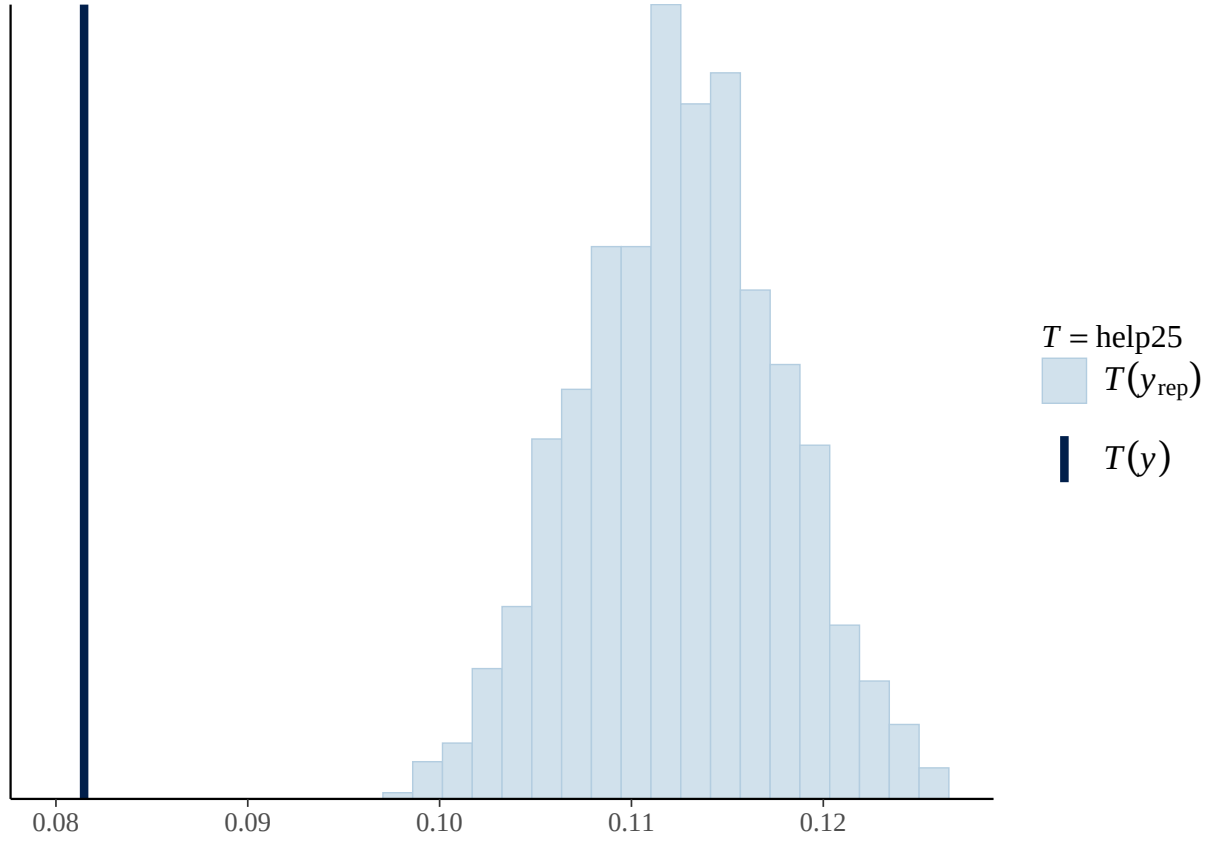


6.3 Question 6

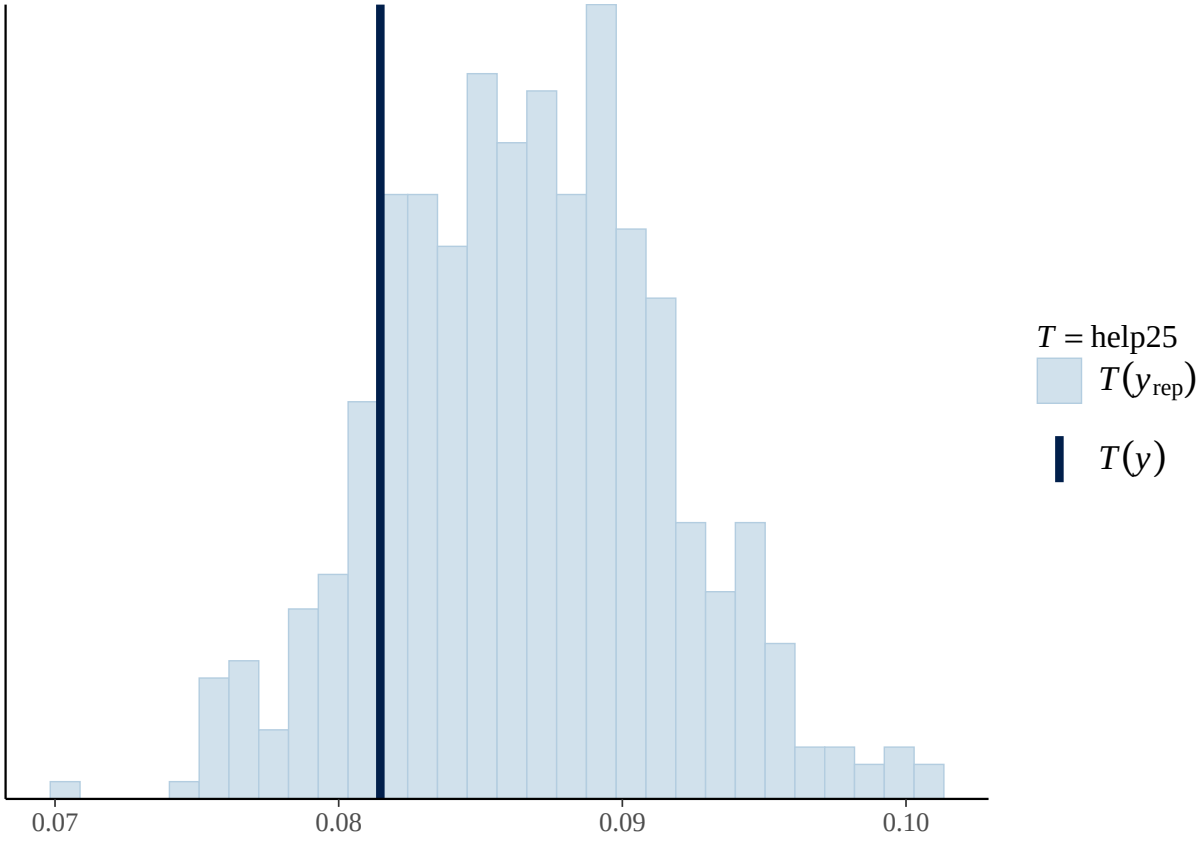
Use a test statistic of the proportion of births under 2.5kg. Calculate the test statistic for the data, and the posterior predictive samples for both models, and plot the comparison (one plot per model).

```
help25 <- function(x){
  result = sum(exp(x) < 2.5)/length(x)
  return(result)
}

ppc_stat(ds$log_weight, yrep1, stat = help25)
```




```
ppc_stat(ds$log_weight, yrep2, stat = help25)
```



7 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 <- extract(mod1)[["log_lik"]]
loglik2 <- extract(mod2)[["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)
loo2 <- loo(loglik2, save_psis = TRUE)
```

Look at the output:

```
loo1
```

```
##
## Computed from 1000 by 3842 log-likelihood matrix
##
##          Estimate    SE
## elpd_loo   1377.2  72.6
## p_loo         9.6   1.5
## looic     -2754.5 145.2
## ------
## 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.
```

```
loo2
```

```
##
## Computed from 500 by 3842 log-likelihood matrix
##
##          Estimate    SE
## elpd_loo   1552.8  70.0
## p_loo        14.8   2.3
## looic     -3105.6 139.9
## ------
## Monte Carlo SE of elpd_loo is 0.2.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

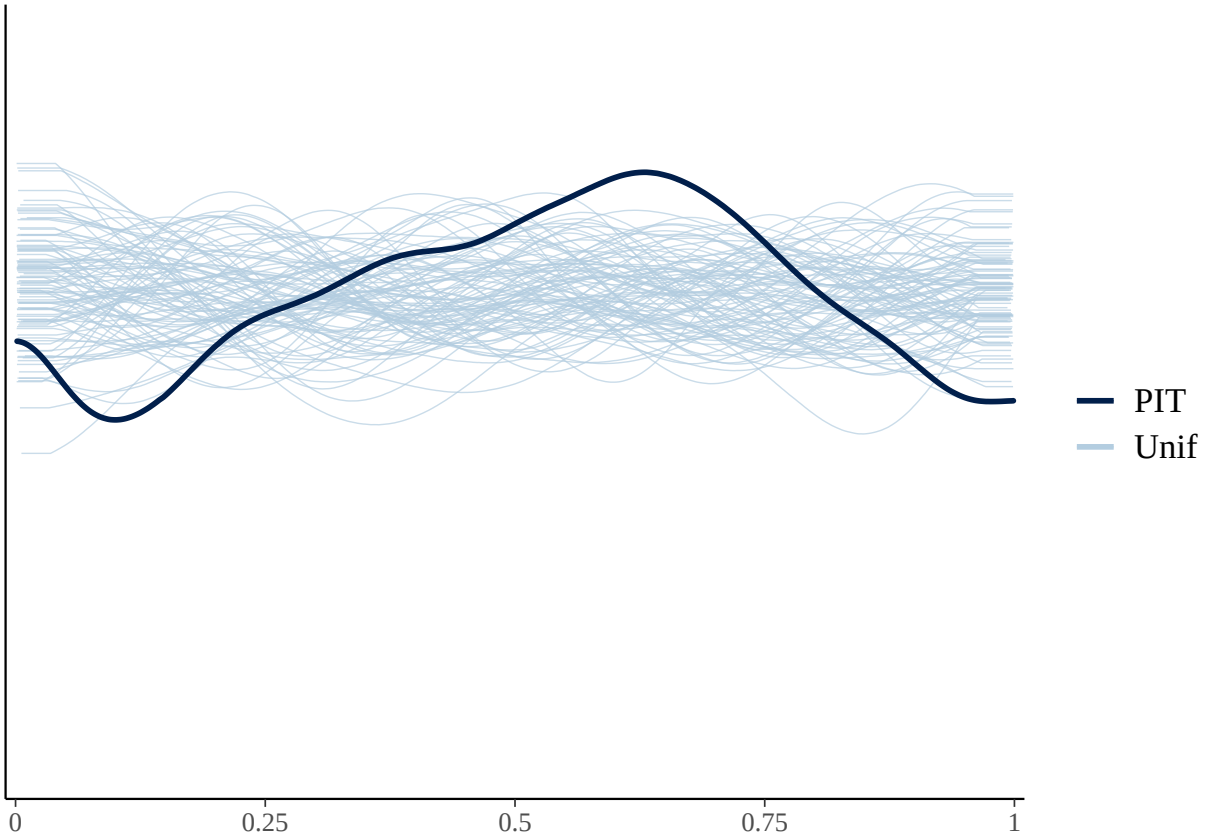
Comparing the two models tells us Model 2 is better:

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

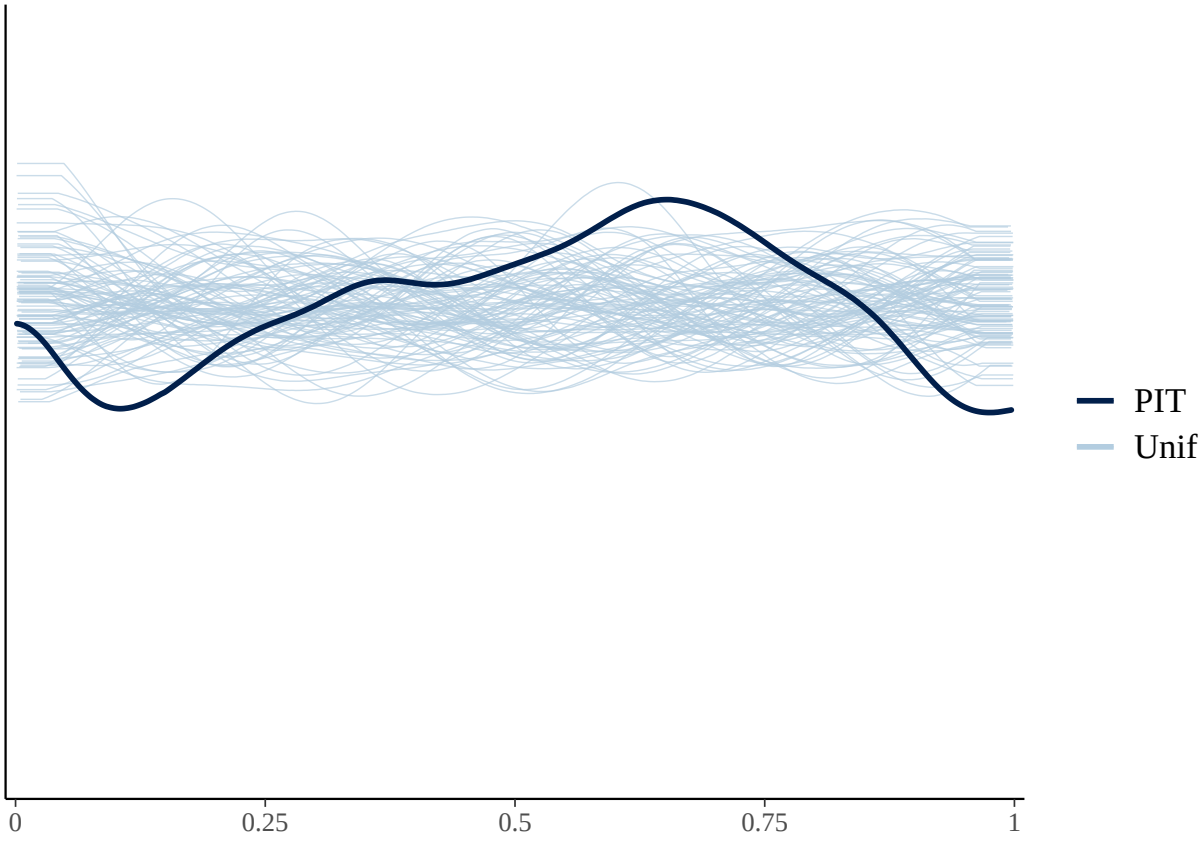
```
##        elpd_diff se_diff
## model2    0.0       0.0
## model1 -175.6      36.3
```

We can also compare the LOO-PIT of each of the models to standard uniforms. The both do pretty well.

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




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



7.1 Bonus question

Create your own PIT histogram "from scratch" for Model 2.