Visualizing the Bayesian Workflow

Michal Malyska

February 26 2020

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(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

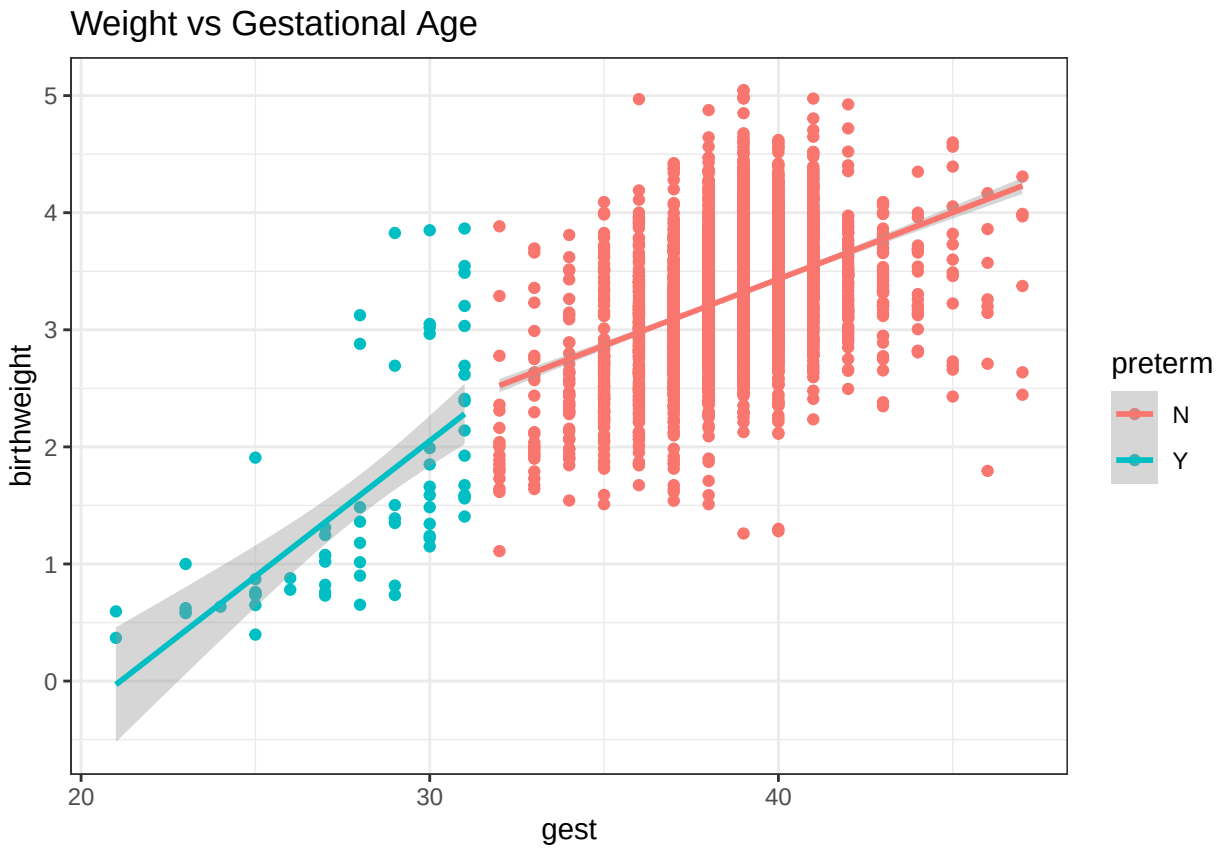
I'm going to rename some variables, remove any observations with missing gestational age or birth weight, restrict just to babies that were alive, and make a preterm variable.

```
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

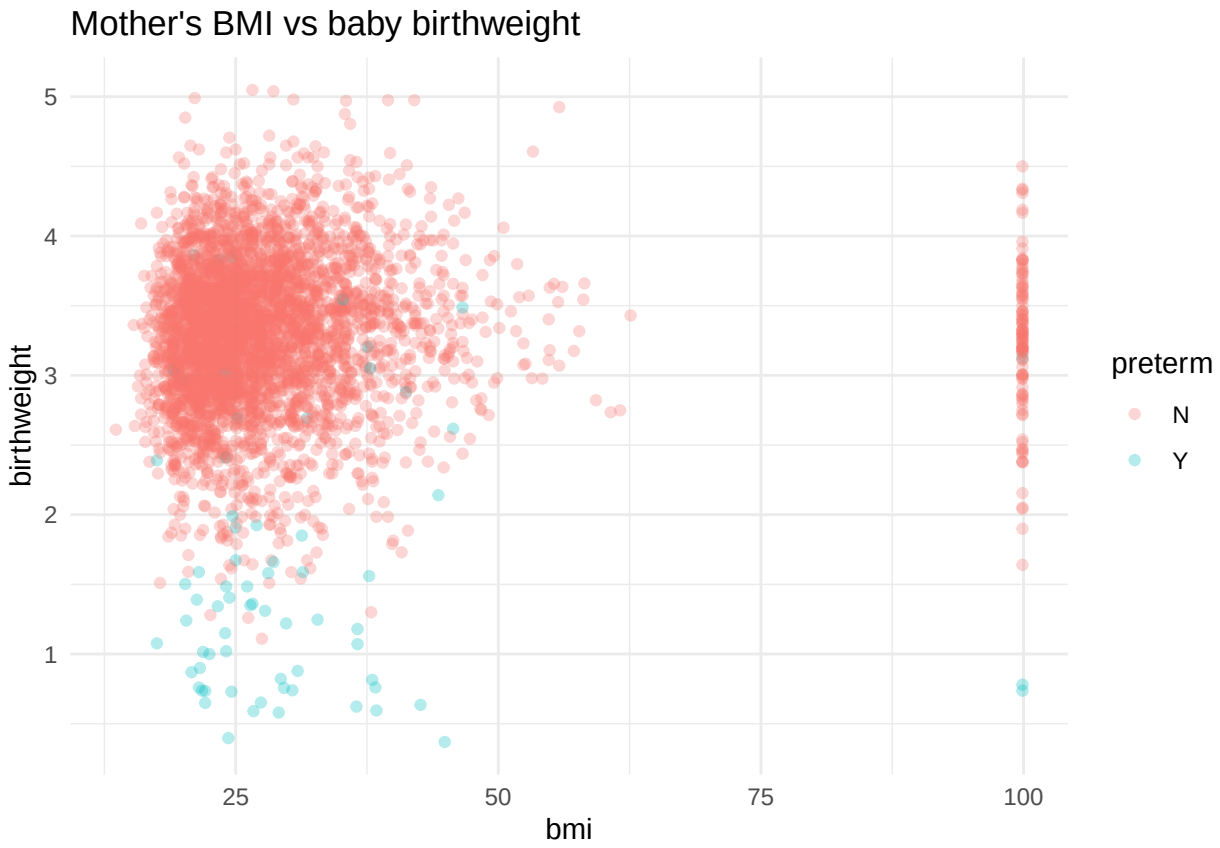
Should sound familiar by now: use plots or tables to show three interesting observations about the data. Remember:

```
ds %>%
  ggplot() +
  aes(x = gest, y = birthweight, color = preterm, group = preterm) +
  geom_point() +
  theme_bw() +
  geom_smooth(method = "lm") +
  labs(title = "Weight vs Gestational Age")
```



First I reproduced the gestation vs birthweight plot from the slides - It shows that there seems to be a difference in slopes for preterm vs full term babies for birthweight as a function of gestation. However since the cutoff for pre-term is not data dependent but rather given it seems a bit arbitrary and moving the cutoff a bit to the right (>32) would result in slopes getting closer. Plus there is a big uncertainty in the line since there are significantly fewer datapoints for preterm babies.

```
ds %>%
  ggplot() +
  aes(x = bmi, y = birthweight, color = preterm) +
  geom_point(alpha = 0.3) +
  theme_minimal() +
  labs(title = "Mother's BMI vs baby birthweight")
```



I wanted to see whether mother's BMI could be indicative of anything regarding our response - birthweight, however what I learned is that the data is clearly not reliable - people with BMI of 100 don't exist. That's like someone who is 100 kg and only 1m tall or someone who is 400kg and 2m tall. That would mean a woman that's around average height (1.65m) would weight ~270kg when giving birth. Those probably should be recoded as NA's.

```
ds %>%
  mutate(race = as_factor(case_when(
    mracehisp == 1 ~ "NHW",
    mracehisp == 2 ~ "NHB",
    mracehisp == 3 ~ "NHAIAN",
    mracehisp == 4 ~ "NHA",
    mracehisp == 5 ~ "NHOPI",
    mracehisp == 6 ~ "Hisp >1 race",
    mracehisp == 7 ~ "Hisp",
    mracehisp == 8 ~ "Unknown"
  ))) %>%
  group_by(race, sex) %>%
  summarize(
    n = n(),
    mean_gest = mean(gest),
    med_gest = median(gest),
    var_gest = var(gest),
    mean_bw = mean(birthweight),
    med_bw = median(birthweight),
    var_bw = var(birthweight),
    num_preterm = sum(preterm == "Y"),
    prop_preterm = mean(preterm == "Y")
```

```
  ) %>%
  arrange(desc(n))
```

```
## # A tibble: 16 x 11
## # Groups:   race [8]
##    race  sex       n mean_gest med_gest var_gest mean_bw med_bw var_bw
##    <fct> <chr> <int>     <dbl>    <dbl>    <dbl>   <dbl>  <dbl>  <dbl>
##  1 NHW   M      1020      38.8     39       5.17    3.41   3.46  0.322
##  2 NHW   F       995      38.8     39       5.20    3.27   3.31  0.316
##  3 Hisp  F       447      38.6     39       6.03    3.19   3.23  0.305
##  4 Hisp  M       426      38.4     39       5.54    3.30   3.33  0.313
##  5 NHB   F       293      38.0     38       7.30    2.99   3.06  0.382
##  6 NHB   M       281      38.2     39       7.39    3.17   3.18  0.425
##  7 NHA   M       123      38.4     39       3.30    3.17   3.20  0.259
##  8 NHA   F       118      38.6     39       5.65    3.16   3.14  0.245
##  9 Hisp~ M        38      38.2     38       6.77    3.18   3.37  0.447
## 10 Hisp~ F        36      38.4     39       4.69    3.21   3.28  0.424
## 11 Unkn~ F        17      38.8     39       4.28    3.25   3.3   0.297
## 12 Unkn~ M        15      37.7     37       2.07    3.20   3.32  0.463
## 13 NHAI~ F        14      38.4     39      27.3     3.10   3.18  0.946
## 14 NHAI~ M        12      39.2     39       6.20    3.71   3.72  0.120
## 15 NHOPI M         4      40.2     39.5    18.9     3.21   3.11  0.210
## 16 NHOPI F         3      35       36       7       2.78   2.98  0.130
## # ... with 2 more variables: num_preterm <int>, prop_preterm <dbl>
```

Since we are going to model data for babies and we have race of mothers and sex of babies available it is good to make a check whether we will be doing a disfavour to some group by not modelling them separately / not adding a race and sex as a factor to account for the differences. I outputted a numerical summary to look at gestation time and birthweight plus numbers of observations in each group to see if there are any differences that are clear. There does not seem to be too much variation in things like proportion of babies born pre-term but there are some things I would try to look more in depth at like why is the variance of gestation times so much higher for NHB than others (I am not comparing to the ones with very little data)

- 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

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

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

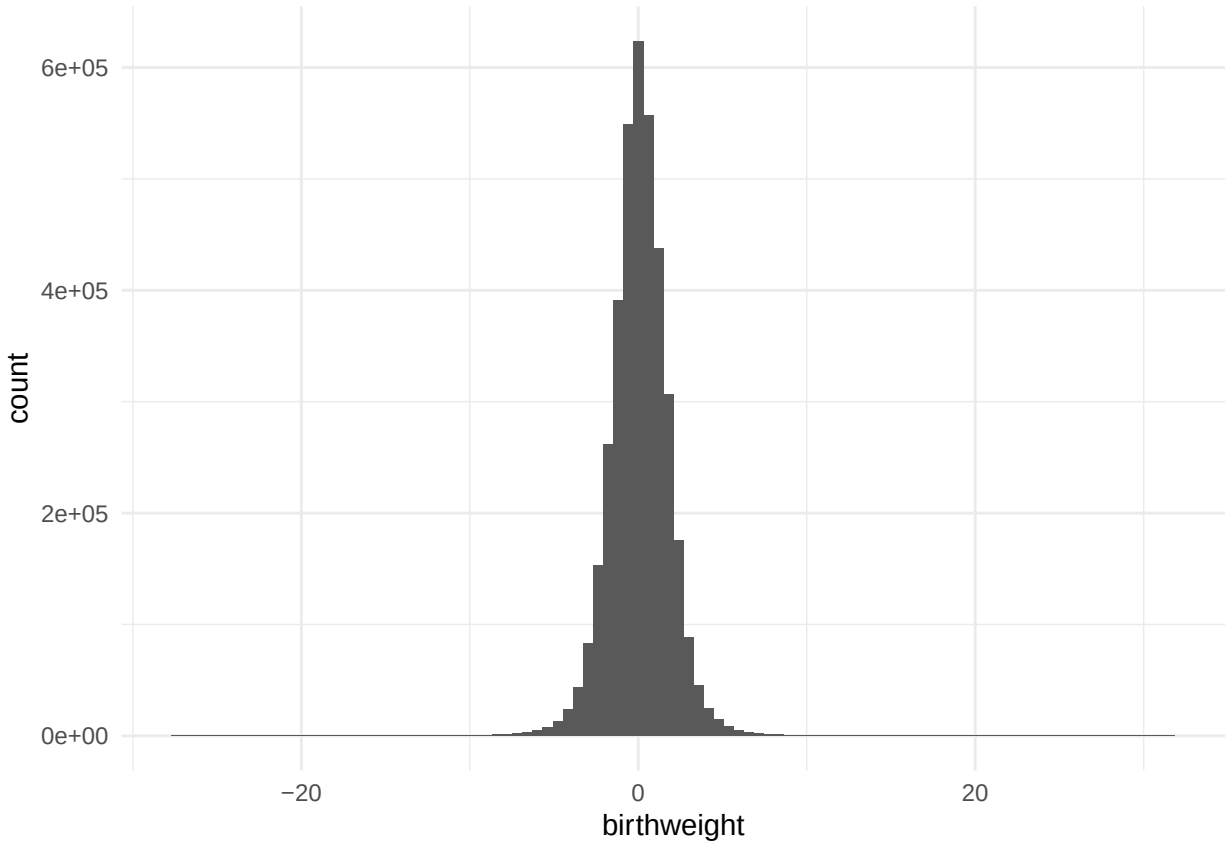
where the plus means positive values only i.e. Half Normal.

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. Here's some skeleton code. Remember to set `eval = TRUE` before you submit. **Also the gestational weights should be centered and standardized**.

```
set.seed(182)
nsims <- 1000
sigma <- abs(rnorm(nsims))
beta0 <- rnorm(nsims)
beta1 <- rnorm(nsims)

# a tibble to store simulations
# we will calculate likelihood based on observed set of (log, centered, standarized) gestational length
lgc <- ds %>% mutate(log_gests_centered = scale(log(gest))) %>% pull(log_gests_centered)
dsims <- tibble(log_gest_c = lgc)

for (i in 1:nsims) {
  this_mu <- beta0[i] + beta1[i] * dsims$log_gest_c
  dsims[paste0(i)] <- rnorm(3842, mean = this_mu, sd = sigma[i])
}

# plot histogram
data_for_plot <- dsims %>% select(-log_gest_c) %>% pivot_longer(cols = everything())
data_for_plot %>%
  ggplot(aes(x = value)) +
  geom_histogram(bins = 100) +
  theme_minimal() +
  scale_x_continuous(name = "birthweight")
```



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))

# 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 = here("code/models/", "simple_weight.stan"),
             iter = 500,
             seed = 243)
```

```
##
## SAMPLING FOR MODEL 'simple_weight' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000716 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 7.16 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: 0.774762 seconds (Warm-up)
## Chain 1:                0.762955 seconds (Sampling)
## Chain 1:                1.53772 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'simple_weight' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.000263 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 2.63 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: 0.827773 seconds (Warm-up)
## Chain 2:                0.686152 seconds (Sampling)
## Chain 2:                1.51392 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'simple_weight' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0.000338 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 3.38 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: 0.952813 seconds (Warm-up)
## Chain 3:                0.984629 seconds (Sampling)
## Chain 3:                1.93744 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'simple_weight' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0.000253 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 2.53 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: 0.863182 seconds (Warm-up)
## Chain 4:                0.808933 seconds (Sampling)
## Chain 4:                1.67212 seconds (Total)
## Chain 4:
```

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

```
##              mean      se_mean          sd      2.5%      25%       50%
## beta[1] 1.1625715 7.956621e-05 0.002739335 1.1574710 1.160660 1.1626397
## beta[2] 0.1438563 7.557017e-05 0.002662185 0.1386213 0.142105 0.1438678
## sigma   0.1689489 1.072462e-04 0.001856871 0.1654528 0.167652 0.1689291
##               75%     97.5%     n_eff      Rhat
## beta[1] 1.1644525 1.1678794 1185.3127 0.9988721
## beta[2] 0.1456511 0.1490625 1241.0112 0.9971666
## sigma   0.1702214 0.1725322  299.7777 1.0173937
```

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

```
transformed data {
  vector[N] inter;           // interaction
  inter     = log_gest .* preterm;
}
```

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

mod2 <- stan(data = stan_data,
             file = here("code/models/", "simple_weight_interaction.stan"),
             iter = 250,
             seed = 243)
```

```
##
## SAMPLING FOR MODEL 'simple_weight_interaction' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.001047 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 10.47 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: WARNING: There aren't enough warmup iterations to fit the
## Chain 1:          three stages of adaptation as currently configured.
## Chain 1:          Reducing each adaptation stage to 15%/75%/10% of
## Chain 1:          the given number of warmup iterations:
## Chain 1:            init_buffer = 18
## Chain 1:            adapt_window = 95
## Chain 1:            term_buffer = 12
## Chain 1:
## Chain 1: Iteration:   1 / 250 [  0%]  (Warmup)
## Chain 1: Iteration:  25 / 250 [ 10%]  (Warmup)
## Chain 1: Iteration:  50 / 250 [ 20%]  (Warmup)
## Chain 1: Iteration:  75 / 250 [ 30%]  (Warmup)
## Chain 1: Iteration: 100 / 250 [ 40%]  (Warmup)
## Chain 1: Iteration: 125 / 250 [ 50%]  (Warmup)
## Chain 1: Iteration: 126 / 250 [ 50%]  (Sampling)
## Chain 1: Iteration: 150 / 250 [ 60%]  (Sampling)
## Chain 1: Iteration: 175 / 250 [ 70%]  (Sampling)
## Chain 1: Iteration: 200 / 250 [ 80%]  (Sampling)
## Chain 1: Iteration: 225 / 250 [ 90%]  (Sampling)
## Chain 1: Iteration: 250 / 250 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 13.6328 seconds (Warm-up)
## Chain 1:                14.003 seconds (Sampling)
```

```
## Chain 1:                27.6358 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'simple_weight_interaction' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.000394 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 3.94 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: WARNING: There aren't enough warmup iterations to fit the
## Chain 2:          three stages of adaptation as currently configured.
## Chain 2:          Reducing each adaptation stage to 15%/75%/10% of
## Chain 2:          the given number of warmup iterations:
## Chain 2:            init_buffer = 18
## Chain 2:            adapt_window = 95
## Chain 2:            term_buffer = 12
## Chain 2:
## Chain 2: Iteration:   1 / 250 [  0%]  (Warmup)
## Chain 2: Iteration:  25 / 250 [ 10%]  (Warmup)
## Chain 2: Iteration:  50 / 250 [ 20%]  (Warmup)
## Chain 2: Iteration:  75 / 250 [ 30%]  (Warmup)
## Chain 2: Iteration: 100 / 250 [ 40%]  (Warmup)
## Chain 2: Iteration: 125 / 250 [ 50%]  (Warmup)
## Chain 2: Iteration: 126 / 250 [ 50%]  (Sampling)
## Chain 2: Iteration: 150 / 250 [ 60%]  (Sampling)
## Chain 2: Iteration: 175 / 250 [ 70%]  (Sampling)
## Chain 2: Iteration: 200 / 250 [ 80%]  (Sampling)
## Chain 2: Iteration: 225 / 250 [ 90%]  (Sampling)
## Chain 2: Iteration: 250 / 250 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 13.3085 seconds (Warm-up)
## Chain 2:                16.7213 seconds (Sampling)
## Chain 2:                30.0297 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'simple_weight_interaction' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0.000407 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 4.07 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: WARNING: There aren't enough warmup iterations to fit the
## Chain 3:          three stages of adaptation as currently configured.
## Chain 3:          Reducing each adaptation stage to 15%/75%/10% of
## Chain 3:          the given number of warmup iterations:
## Chain 3:            init_buffer = 18
## Chain 3:            adapt_window = 95
## Chain 3:            term_buffer = 12
## Chain 3:
## Chain 3: Iteration:   1 / 250 [  0%]  (Warmup)
## Chain 3: Iteration:  25 / 250 [ 10%]  (Warmup)
## Chain 3: Iteration:  50 / 250 [ 20%]  (Warmup)
```

```
## Chain 3: Iteration:  75 / 250 [ 30%]  (Warmup)
## Chain 3: Iteration: 100 / 250 [ 40%]  (Warmup)
## Chain 3: Iteration: 125 / 250 [ 50%]  (Warmup)
## Chain 3: Iteration: 126 / 250 [ 50%]  (Sampling)
## Chain 3: Iteration: 150 / 250 [ 60%]  (Sampling)
## Chain 3: Iteration: 175 / 250 [ 70%]  (Sampling)
## Chain 3: Iteration: 200 / 250 [ 80%]  (Sampling)
## Chain 3: Iteration: 225 / 250 [ 90%]  (Sampling)
## Chain 3: Iteration: 250 / 250 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 13.829 seconds (Warm-up)
## Chain 3:                18.4857 seconds (Sampling)
## Chain 3:                32.3147 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'simple_weight_interaction' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0.000418 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 4.18 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: WARNING: There aren't enough warmup iterations to fit the
## Chain 4:          three stages of adaptation as currently configured.
## Chain 4:          Reducing each adaptation stage to 15%/75%/10% of
## Chain 4:          the given number of warmup iterations:
## Chain 4:            init_buffer = 18
## Chain 4:            adapt_window = 95
## Chain 4:            term_buffer = 12
## Chain 4:
## Chain 4: Iteration:   1 / 250 [  0%]  (Warmup)
## Chain 4: Iteration:  25 / 250 [ 10%]  (Warmup)
## Chain 4: Iteration:  50 / 250 [ 20%]  (Warmup)
## Chain 4: Iteration:  75 / 250 [ 30%]  (Warmup)
## Chain 4: Iteration: 100 / 250 [ 40%]  (Warmup)
## Chain 4: Iteration: 125 / 250 [ 50%]  (Warmup)
## Chain 4: Iteration: 126 / 250 [ 50%]  (Sampling)
## Chain 4: Iteration: 150 / 250 [ 60%]  (Sampling)
## Chain 4: Iteration: 175 / 250 [ 70%]  (Sampling)
## Chain 4: Iteration: 200 / 250 [ 80%]  (Sampling)
## Chain 4: Iteration: 225 / 250 [ 90%]  (Sampling)
## Chain 4: Iteration: 250 / 250 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 12.0316 seconds (Warm-up)
## Chain 4:                13.5223 seconds (Sampling)
## Chain 4:                25.5539 seconds (Total)
## Chain 4:
```

```
summary(mod2)$summary[c(paste0("beta[", 1:4, "]"), "sigma"),]
```

```
##                mean      se_mean          sd       2.5%        25%         50%
## beta[1]  1.16239521 1.159396e-04 0.002720126  1.1570368  1.1605675  1.16235144
## beta[2]  0.14402374 1.142597e-04 0.002675444  0.1387461  0.1423456  0.14408656
## beta[3] -0.18432614 1.224303e-01 1.117927599 -2.5900538 -0.9880317 -0.07037156
## beta[4]  0.07881893 1.265887e-01 1.200923760 -2.2246796 -0.6421216 -0.07528556
```

```
## sigma    0.16898028 8.573476e-05 0.001910467  0.1653289  0.1677819  0.16898345
##               75%     97.5%     n_eff      Rhat
## beta[1] 1.1641920 1.1680463 550.44573 1.0012639
## beta[2] 0.1456805 0.1492010 548.28439 0.9984674
## beta[3] 0.6119829 1.8029537  83.37764 1.0840081
## beta[4] 0.8909053 2.7388019  89.99967 1.0196837
## sigma   0.1701640 0.1728563 496.55266 0.9968014
```

5.2 Question 4

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

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

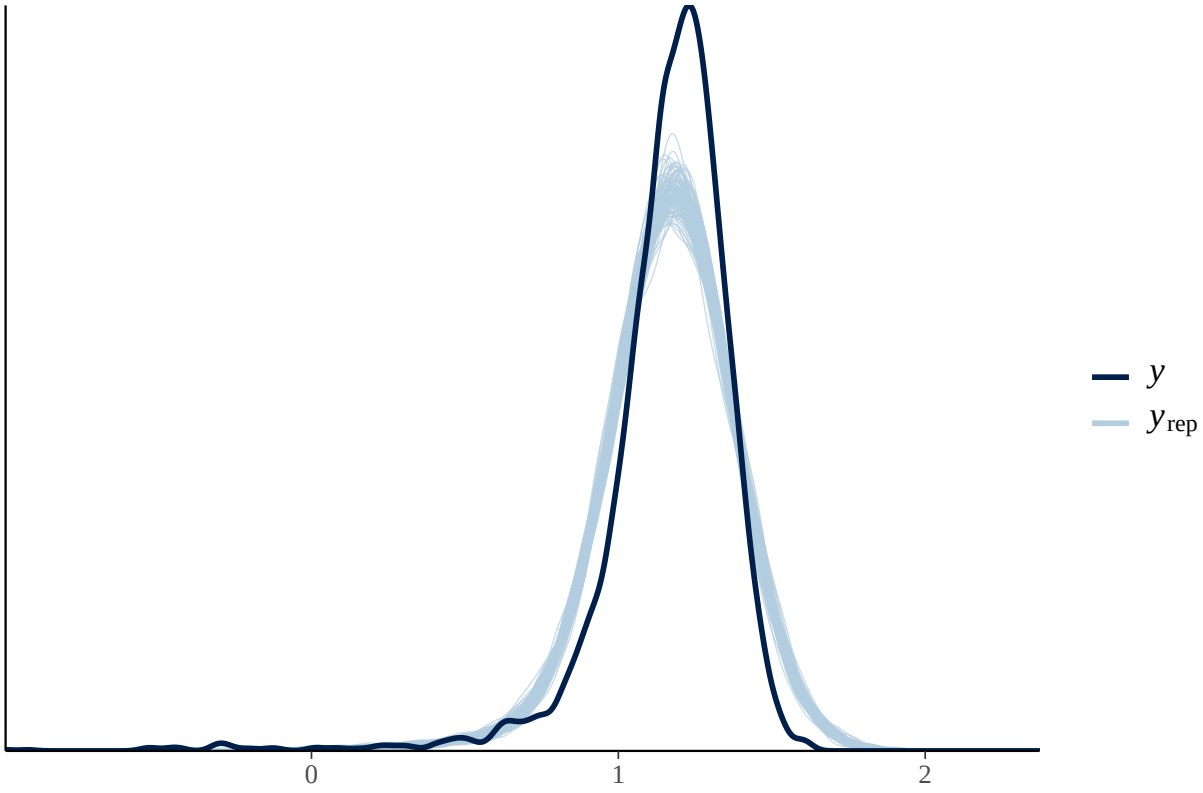
```
##              mean      se_mean          sd       2.5%        25%       50%
## beta[1] 1.1697241 1.385590e-04 0.002742186 1.16453578 1.16767109 1.1699278
## beta[2] 0.5563133 5.835253e-03 0.058054991 0.43745504 0.51708255 0.5561553
## beta[3] 0.1020960 1.481816e-04 0.003669476 0.09459462 0.09997153 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:

```
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
```

distribution of observed versus predicted birthweights

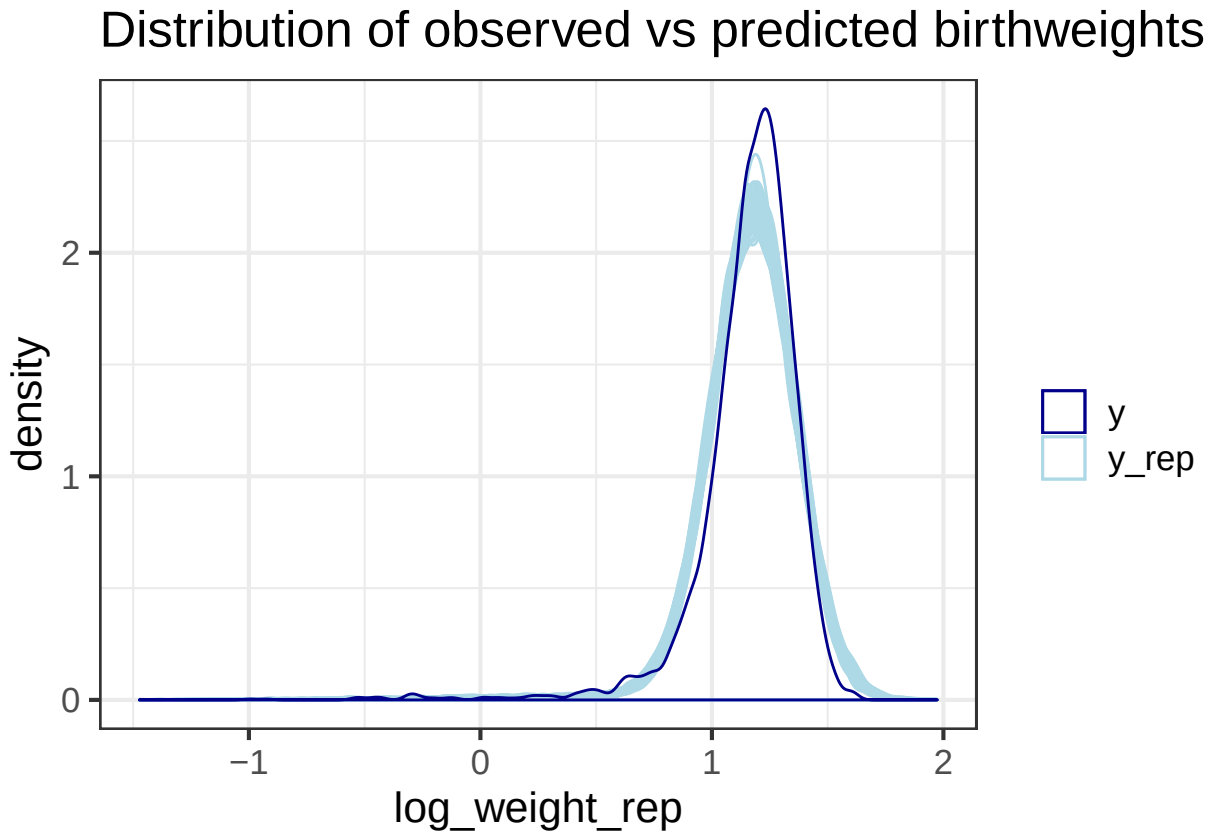


6.1 Question 5

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`)

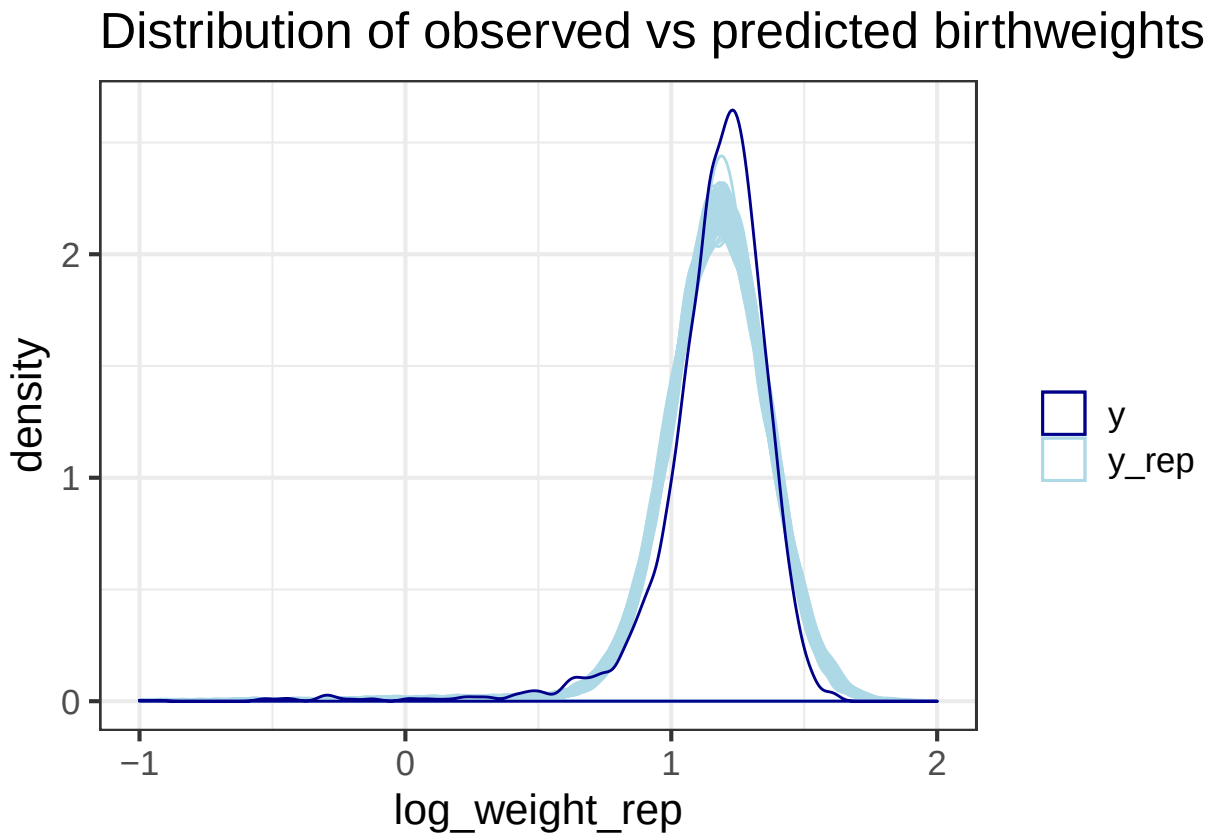
```
samp100_2 <- sample(nrow(yrep2), 100)
y2 <- as_tibble(t(yrep2[samp100_2, ]))

# data easier to plot
colnames(y2) <- 1:100

dr <- as_tibble(y2)
dr <- dr %>% bind_cols(i = 1:3842, log_weight_obs = log(ds$birthweight))

dr <- dr %>%
  pivot_longer(`1`:`100`, names_to = "sim", values_to = "log_weight_rep")

# plot densities for 100 samples

set.seed(176)

dr %>%
  ggplot(aes(log_weight_rep, group = sim)) +
  geom_density(alpha = 0.2, aes(color = "y_rep")) +
  geom_density(data = ds %>% mutate(sim = 1),
               aes(x = log(birthweight), col = "y")) +
  scale_color_manual(name = "",
```

```
                    values = c("y" = "darkblue",
                               "y_rep" = "lightblue")) +
  ggtitle("Distribution of observed vs predicted birthweights") +
  theme_bw(base_size = 16)
```



This seemed super narrow at first but it is just because of the x axis scale.

```
dr %>%
  ggplot(aes(log_weight_rep, group = sim)) +
  geom_density(alpha = 0.2, aes(color = "y_rep")) +
  geom_density(data = ds %>% mutate(sim = 1),
               aes(x = log(birthweight), col = "y")) +
  scale_color_manual(name = "",
                     values = c("y" = "darkblue",
                                "y_rep" = "lightblue")) +
  ggtitle("Distribution of observed vs predicted birthweights") +
  theme_bw(base_size = 16) +
  scale_x_continuous(limits = c(-1, 2))
```

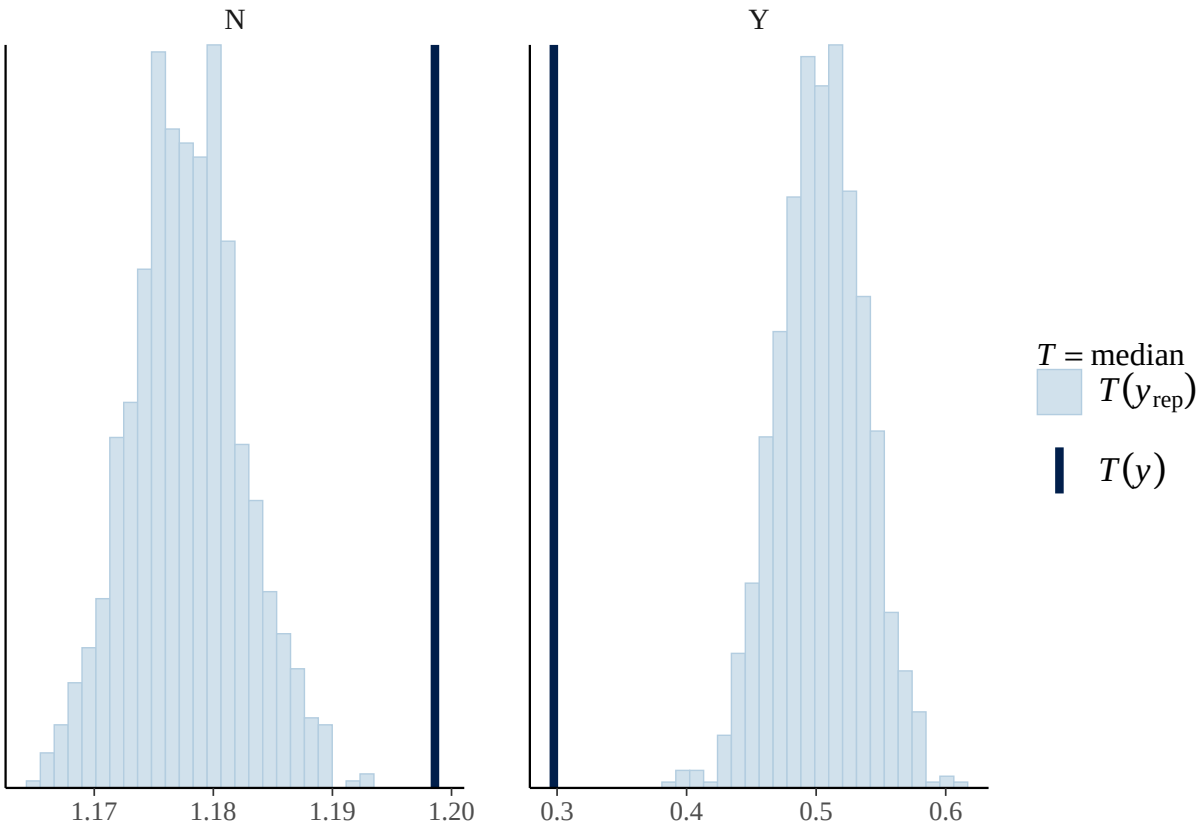


6.2 Test statistics

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

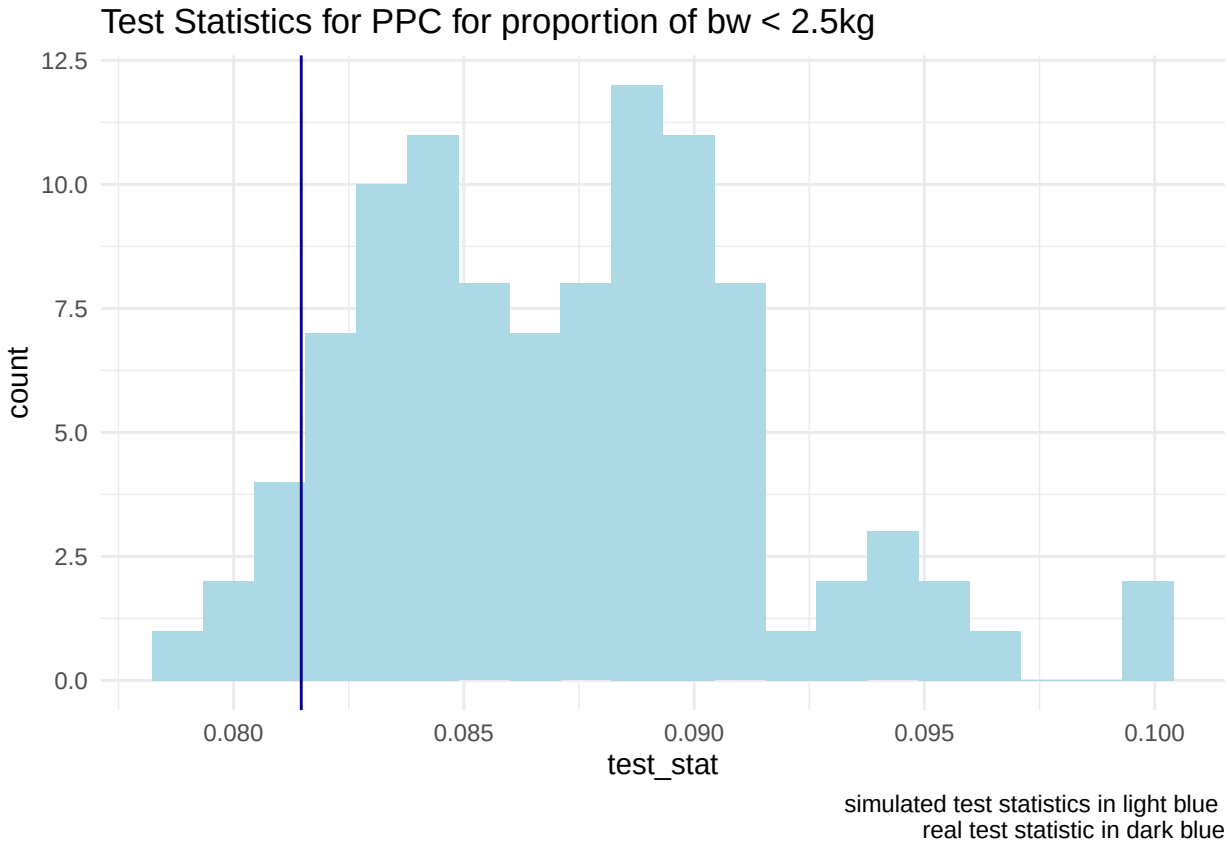
```
ppc_stat_grouped(ds$log_weight, yrep1, group = ds$preterm, stat = 'median')
```



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).

```
test_stat_real <- mean(ds$birthweight < 2.5)
test_stat_rep <- dr %>% group_by(sim) %>%
  summarize(test_stat = mean(exp(log_weight_rep) < 2.5))

test_stat_rep %>%
  ggplot(aes(x = test_stat)) +
  geom_histogram(bins = 20, fill = "lightblue") +
  geom_vline(xintercept = test_stat_real, color = "darkblue") +
  theme_minimal() +
  labs(caption = "simulated test statistics in light blue \n real test statistic in dark blue",
       title = "Test Statistics for PPC for proportion of bw < 2.5kg")
```



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   1377.8  72.4
## p_loo         8.7   1.3
## looic     -2755.5 144.8
## ------
## 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.
```

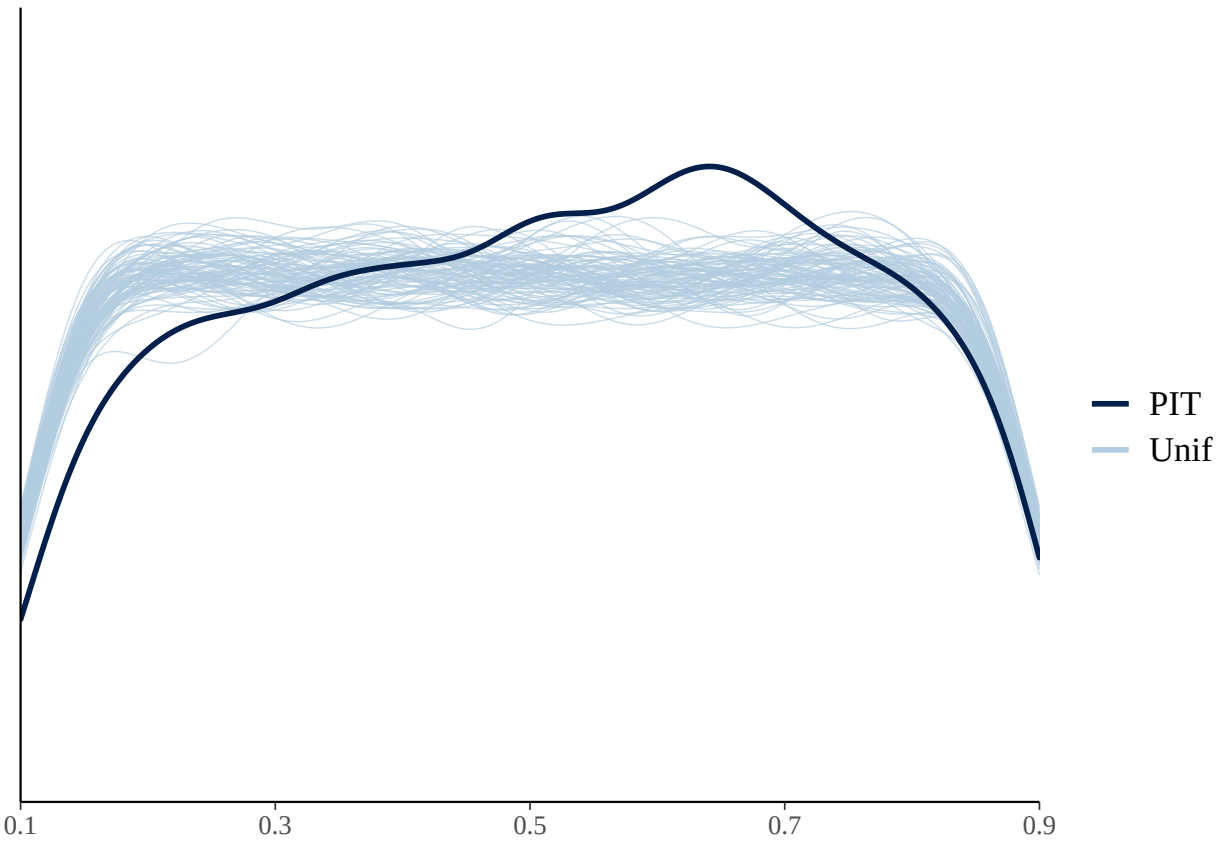
Comparing the two models tells us Model 2 is better:

```
compare(loo1, loo2)
```

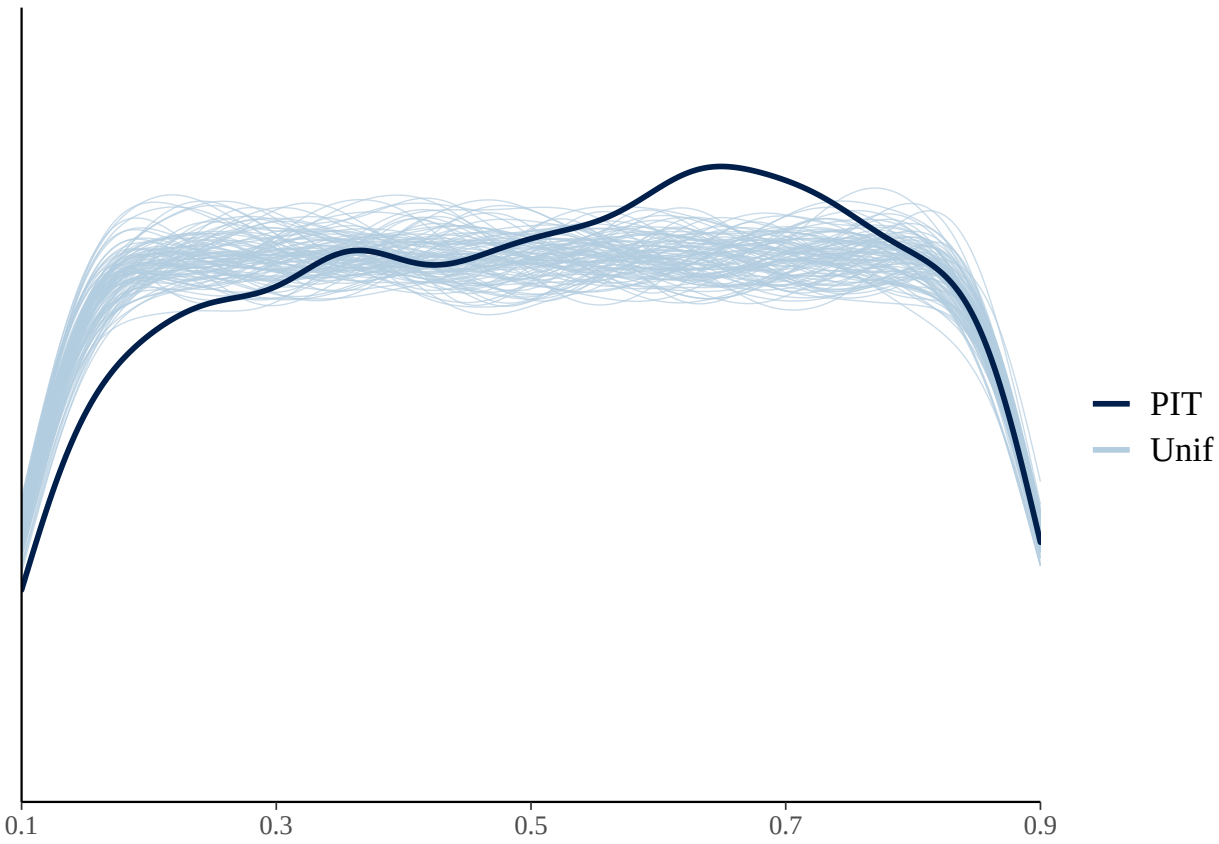
```
## elpd_diff        se
##     175.1      36.1
```

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))
```


0.1
0.3
0.5
0.7
0.9
PIT
Unif