

Visualizing the Bayesian Workflow

Harriet Ware

March 5 2021

Contents

1	Introduction	1
2	The data	2
2.1	Question 1	2
3	The model	4
4	Prior predictive checks	4
4.1	Question 2	5
5	Run the model	6
5.1	Question 3	9
5.2	Question 4	12
6	PPCs	12
6.1	Question 5	13
6.2	Test statistics	14
6.3	Question 6	15
7	LOO	17
7.1	Bonus question	20

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/natalty/2017/natl2017.pdf> page 15
- `meduc` mum's education see here for codes: <https://data.nber.org/natalty/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

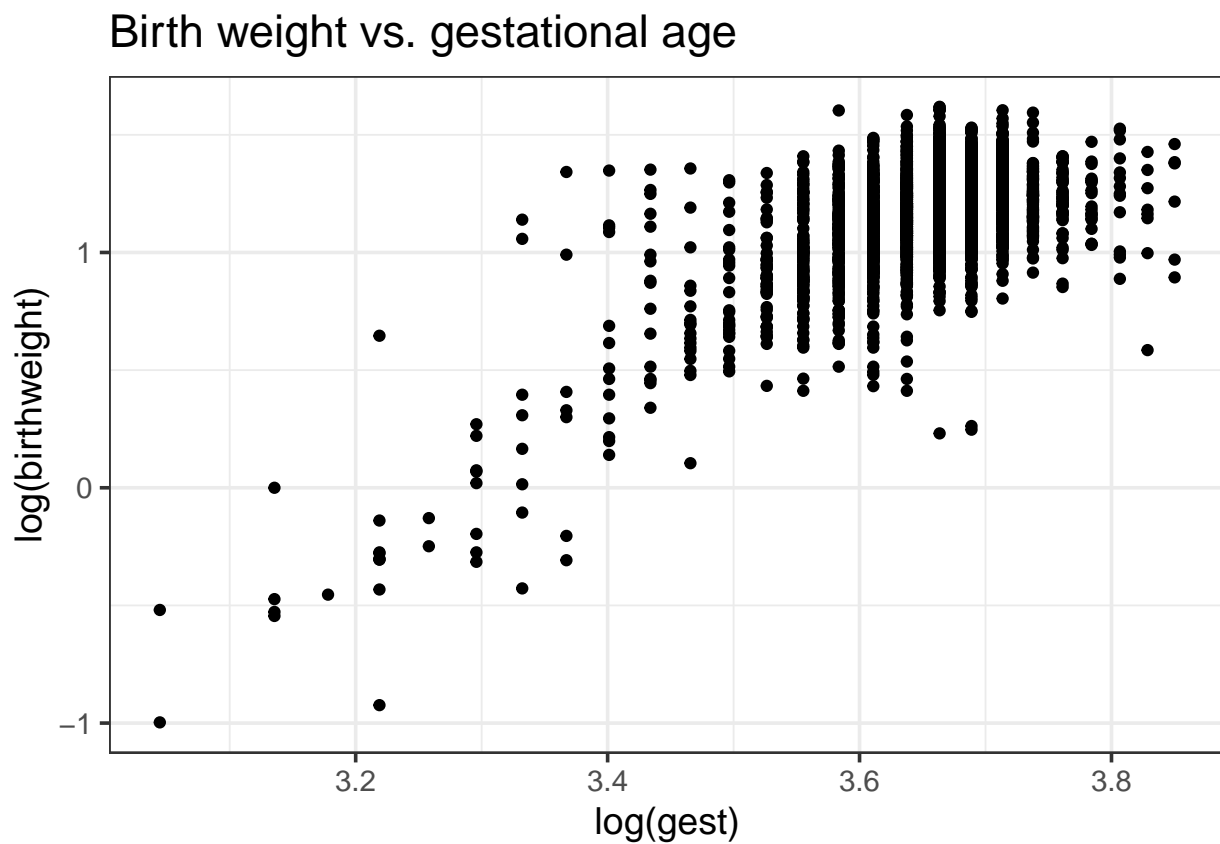
Should sound familiar by now: use plots or tables to show three interesting observations about the data. Remember:

- 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

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.

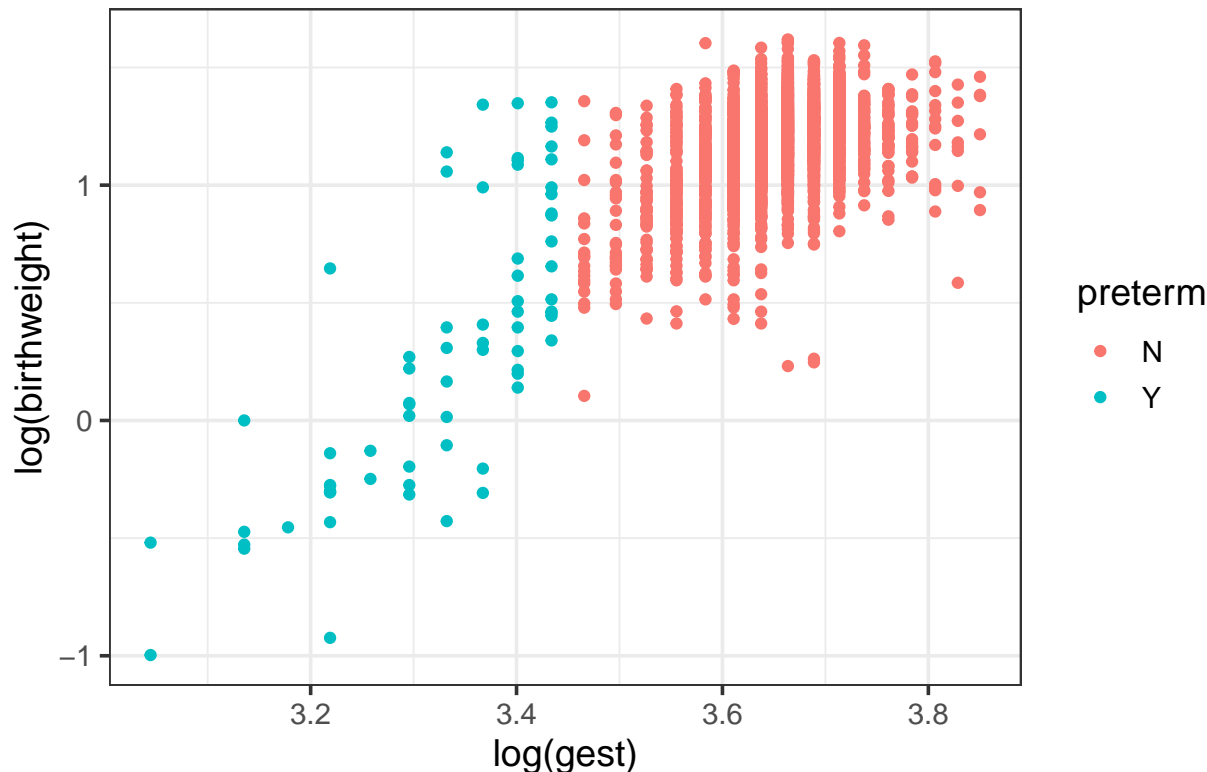
```
ds %>%
  ggplot(aes(log(gest), log(birthweight))) +
  geom_point() +
  theme_bw(base_size = 14) +
  ggtitle("Birth weight vs. gestational age")
```



There seems to be a positive relationship between birth weight and gestational age (on the log scale).

```
ds %>%
  ggplot(aes(log(gest), log(birthweight), color = preterm)) +
  geom_point() +
  theme_bw(base_size = 14) +
  ggtitle("Birth weight vs. gestational age")
```

Birth weight vs. gestational age



If we allow the relationship between weight and gestational age to vary by prematurity, there seems to be a different relationship between the two variables among preterm and not preterm babies.

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

$$\log(y_i) \sim N(\beta_1 + \beta_2 \log(x_i) + \beta_3 z_i + \beta_4 \log(x_i) z_i, \sigma^2)$$

- 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 β s

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

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 β s 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.**

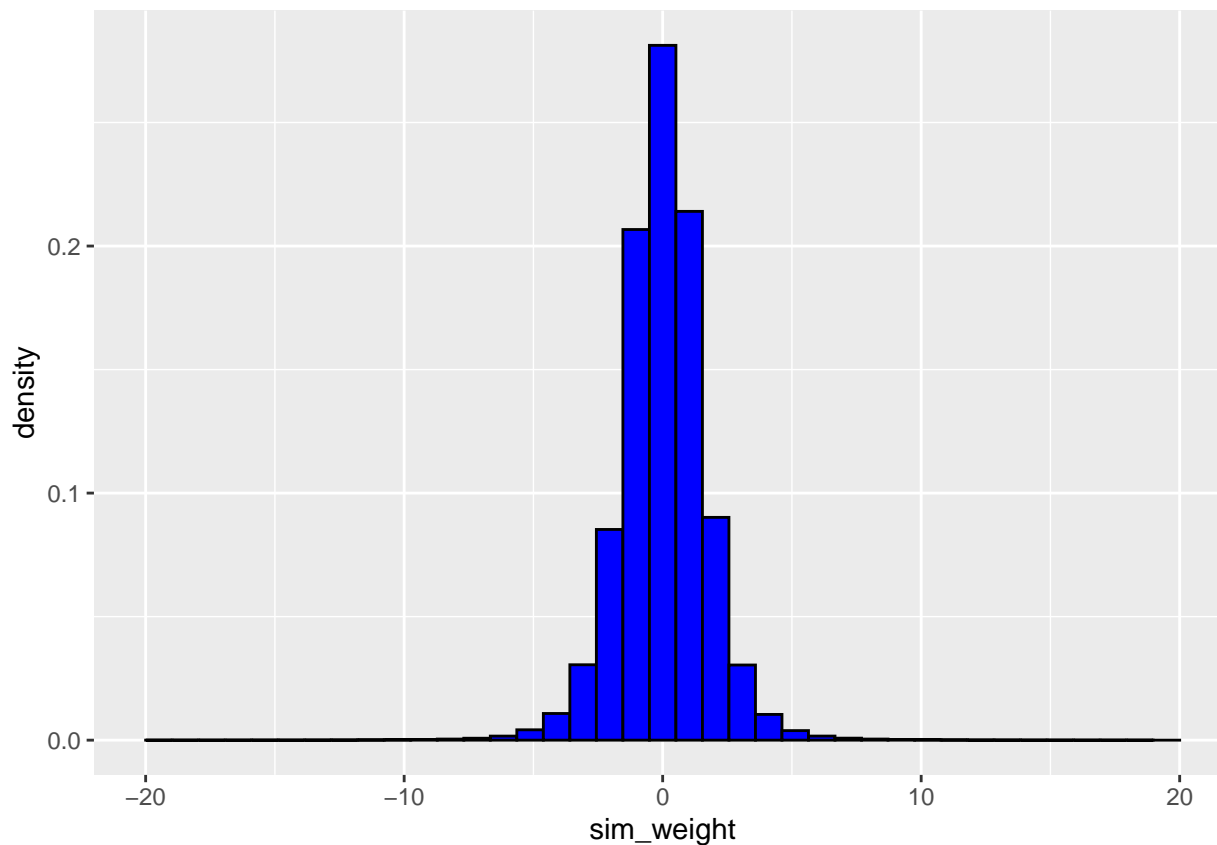
```
nsims<-1000
sigma <- abs(rnorm(nsims, 0, 1))
beta0 <- rnorm(nsims, 0, 1)
beta1 <- rnorm(nsims, 0, 1)

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

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

dslong <- dsims %>%
  pivot_longer('1':'1000', names_to = "sim", values_to = "sim_weight")

dslong %>%
  ggplot(aes(sim_weight)) + geom_histogram(aes(y = ..density..), bins = 40, fill = "blue", color = "black",
  xlim(c(-20,20))
```



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

```
##
## SAMPLING FOR MODEL 'simple_weight' NOW (CHAIN 1).
## Chain 1:
```

```

## Chain 1: Gradient evaluation took 0.001 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 10 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: 1.584 seconds (Warm-up)
## Chain 1:                1.202 seconds (Sampling)
## Chain 1:                2.786 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'simple_weight' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 1.299 seconds (Warm-up)
## Chain 2:                1.268 seconds (Sampling)
## Chain 2:                2.567 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'simple_weight' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 1.377 seconds (Warm-up)
## Chain 3: 1.103 seconds (Sampling)
## Chain 3: 2.48 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'simple_weight' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 1.362 seconds (Warm-up)
## Chain 4: 1.269 seconds (Sampling)
## Chain 4: 2.631 seconds (Total)
## Chain 4:

```

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

```

##           mean      se_mean      sd      2.5%      25%      50%
## beta[1] 1.1626507 8.110475e-05 0.002714086 1.1573955 1.1608221 1.1626281
## beta[2] 0.1436061 7.908990e-05 0.002793714 0.1380281 0.1416853 0.1436199
## sigma   0.1688517 1.090725e-04 0.002019961 0.1650908 0.1674982 0.1687796
##           75%      97.5%      n_eff      Rhat
## beta[1] 1.1645352 1.1678892 1119.8364 0.9973582
## beta[2] 0.1454956 0.1489020 1247.7331 0.9981180
## sigma   0.1700702 0.1729636 342.9693 1.0067923

```


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
```

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

```
##  
## SAMPLING FOR MODEL 'simple_weight_preterm_int' NOW (CHAIN 1).  
## Chain 1:  
## Chain 1: Gradient evaluation took 0.003 seconds  
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 30 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: 4.006 seconds (Warm-up)
## Chain 1: 4.636 seconds (Sampling)
## Chain 1: 8.642 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'simple_weight_preterm_int' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.003 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 30 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: 4.334 seconds (Warm-up)
## Chain 2: 4.208 seconds (Sampling)
## Chain 2: 8.542 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'simple_weight_preterm_int' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 3.916 seconds (Warm-up)
## Chain 3:          4.871 seconds (Sampling)
## Chain 3:          8.787 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'simple_weight_preterm_int' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0.001 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 10 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: 4.159 seconds (Warm-up)
## Chain 4:          5.448 seconds (Sampling)

```

```
## Chain 4:          9.607 seconds (Total)
## Chain 4:
```

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

```
##           mean      se_mean      sd      2.5%      25%      50%
## beta[1] 1.1695857 1.168847e-04 0.002603743 1.16468248 1.16772116 1.1694728
## beta[2] 0.1018581 1.797439e-04 0.003645742 0.09460536 0.09944062 0.1018158
## beta[3] 0.5634473 7.055740e-03 0.060305406 0.43998076 0.52394276 0.5649192
## beta[4] 0.1985907 1.367117e-03 0.012573395 0.17385085 0.19058615 0.1990943
## sigma   0.1612254 9.820237e-05 0.001911679 0.15758325 0.15982619 0.1612172
##           75%      97.5%    n_eff    Rhat
## beta[1] 1.1714010 1.1749522 496.22776 1.0013075
## beta[2] 0.1042765 0.1089305 411.39927 0.9989007
## beta[3] 0.6006719 0.6712653  73.05121 1.0235098
## beta[4] 0.2071440 0.2216538  84.58504 1.0191976
## sigma   0.1624726 0.1651577 378.95353 0.9956560
```

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"),]
```

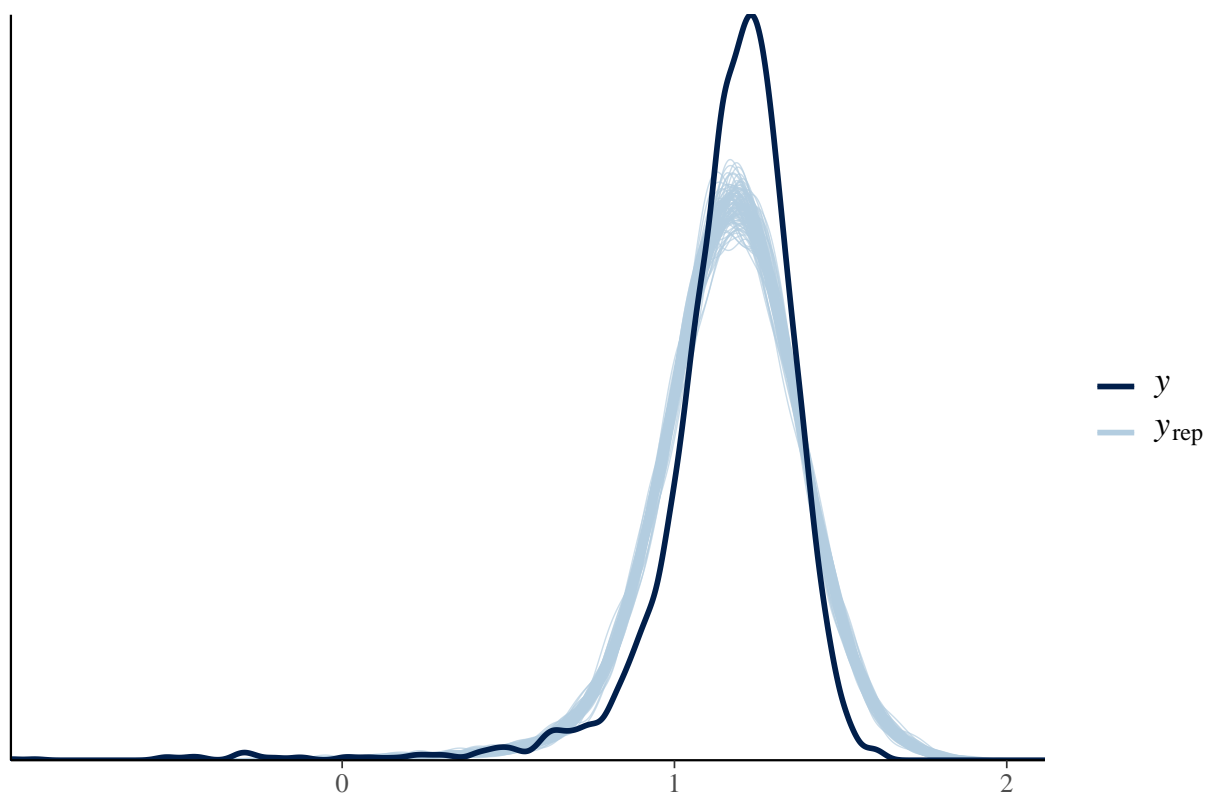
```
##           mean      se_mean      sd      2.5%      25%      50%
## beta[1] 1.1697241 1.385590e-04 0.002742186 1.16453578 1.16767109 1.1699278
## beta[2] 0.5563133 5.835253e-03 0.058054991 0.43745504 0.51708255 0.5561553
## beta[3] 0.1020960 1.481816e-04 0.003669476 0.09459462 0.09997153 0.1020339
## 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 birthweight")
```

distribution of observed versus predicted birthweights



6.1 Question 5

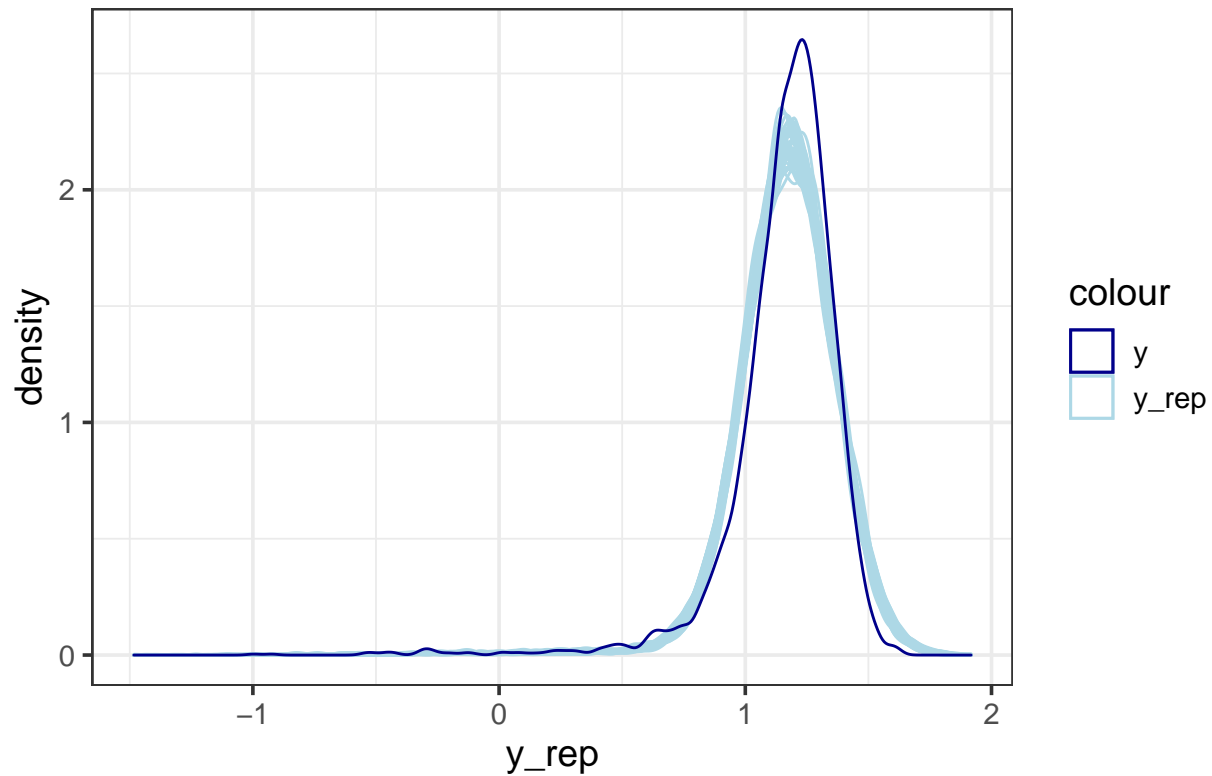
Make a similar plot to the one above but for model 2, and **not** using the bayes plot in built function (i.e. do it yourself just with `geom_density`)

```
rownames(yrep2) <- 1:nrow(yrep2)
dr <- as_tibble(t(yrep2))
dr <- bind_cols(dr, log_weight_obs = log(ds$birthweight))

dr_long <- dr %>%
  pivot_longer(-log_weight_obs, names_to = "sim", values_to = "y_rep")

dr_long %>%
  filter(sim %in% samp100) %>%
  ggplot(aes(y_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(values = c("y" = "darkblue",
                                "y_rep" = "lightblue")) +
  ggtitle("distribution of observed vs predicted birthweights") +
  theme_bw(base_size = 14)
```

distribution of observed vs predicted birthweights

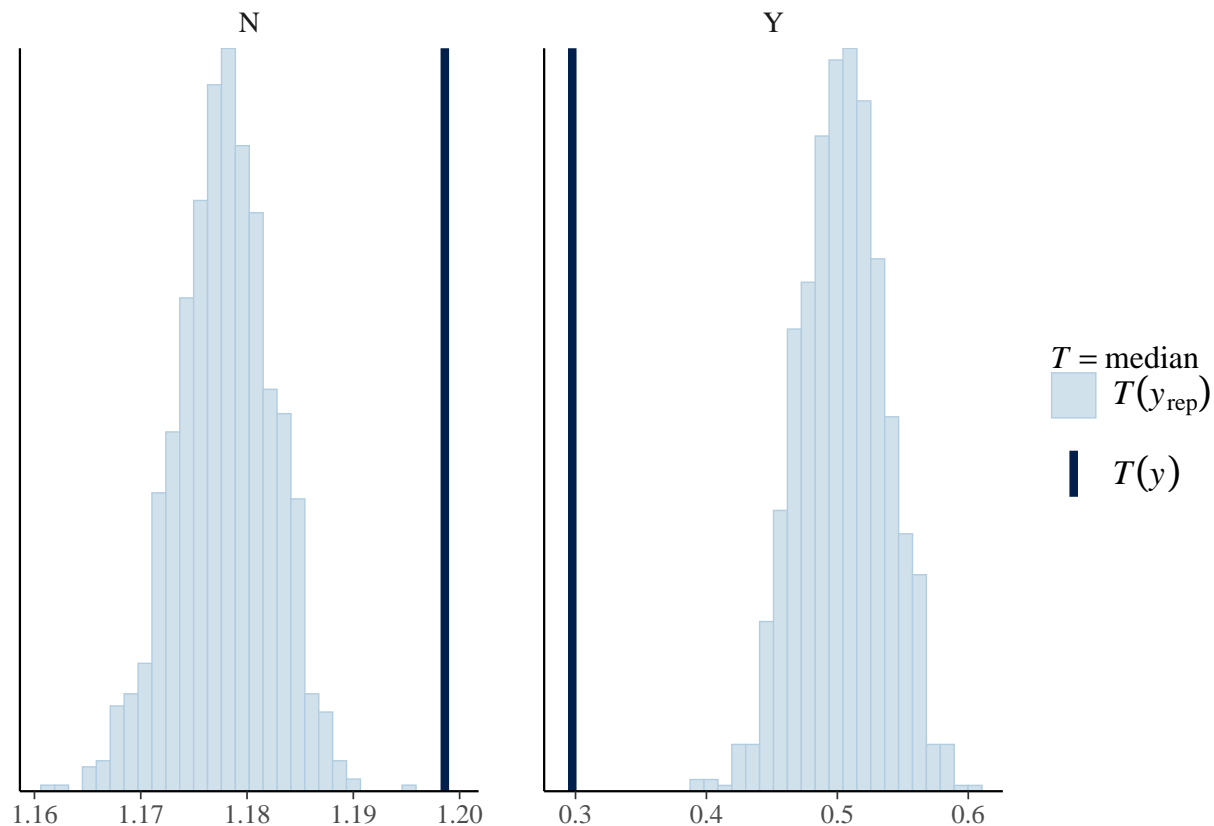


6.2 Test statistics

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')
```



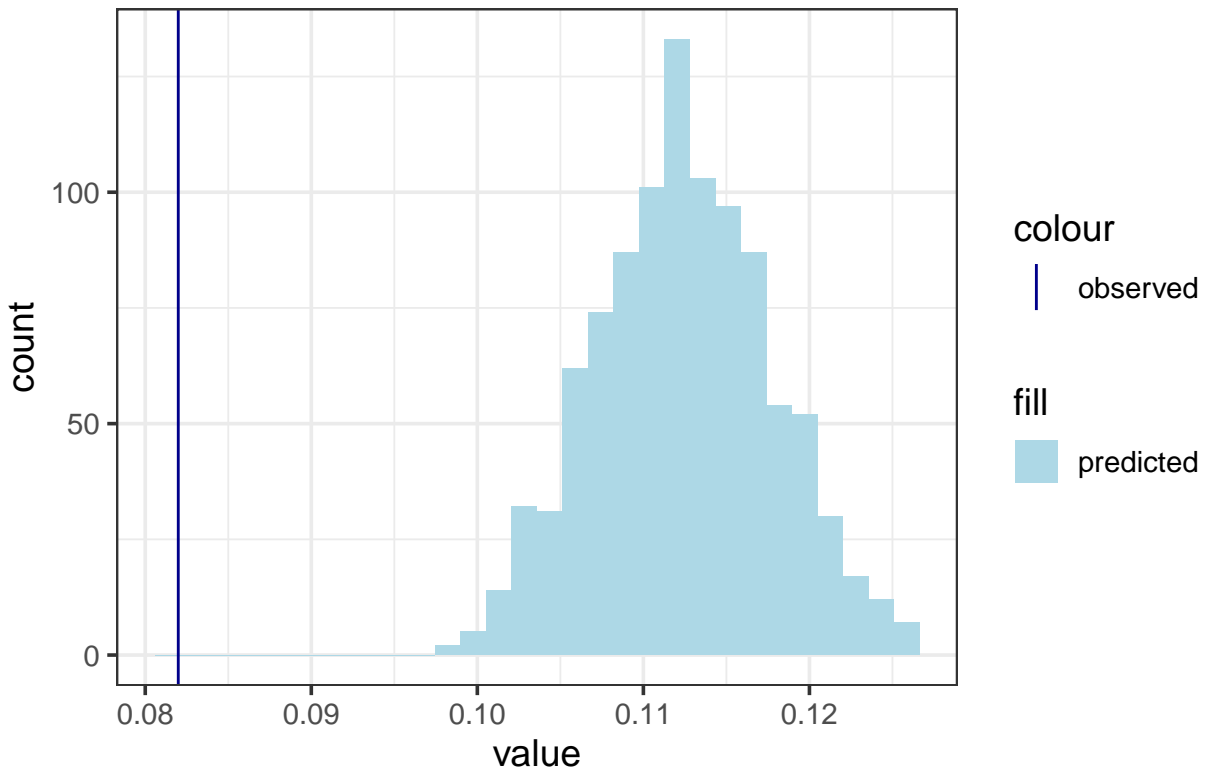
6.3 Question 6

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).

```
t_y <- mean(y<=log(2.5))
t_y_rep <- sapply(1:nrow(yrep1), function(x) mean(yrep1[x,]<=log(2.5)))
t_y_rep_2 <- sapply(1:nrow(yrep2), function(x) mean(yrep2[x,]<=log(2.5)))

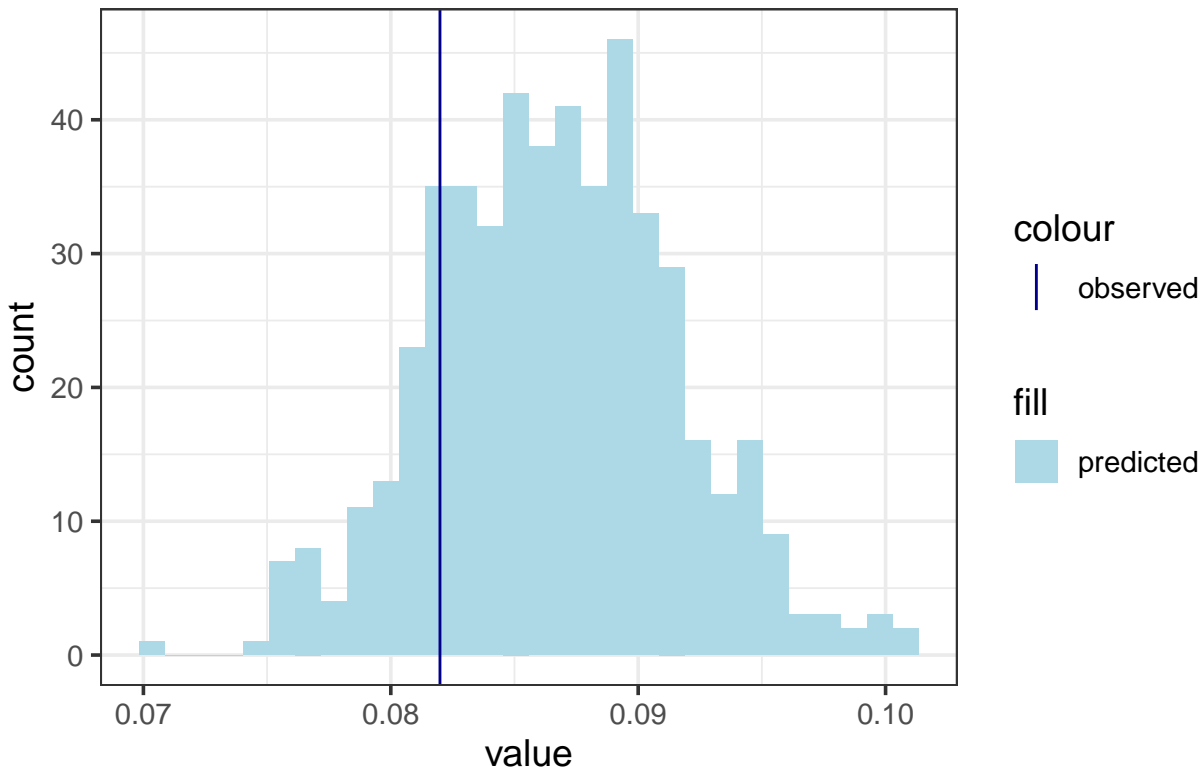
ggplot(data = as_tibble(t_y_rep), aes(value)) +
  geom_histogram(aes(fill = "predicted")) +
  geom_vline(aes(xintercept = t_y, color = "observed")) +
  ggtitle("Model 1: proportion of births less than 2.5kg") +
  theme_bw(base_size = 14) +
  scale_color_manual(values = c("observed" = "darkblue"))+
  scale_fill_manual(values = c("predicted" = "lightblue"))
```

Model 1: proportion of births less than 2.5kg



```
ggplot(data = as_tibble(t_y_rep_2), aes(value)) +  
  geom_histogram(aes(fill = "predicted")) +  
  geom_vline(aes(xintercept = t_y, color = "observed")) +  
  ggtitle("Model 2: proportion of births less than 2.5kg") +  
  theme_bw(base_size = 14) +  
  scale_color_manual(values = c("observed" = "darkblue")) +  
  scale_fill_manual(values = c("predicted" = "lightblue"))
```


Model 2: proportion of births less than 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    1377.1   72.6
```

```
## p_loo          9.8   1.5
## looic         -2754.2 145.3
## -----
## 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.
```

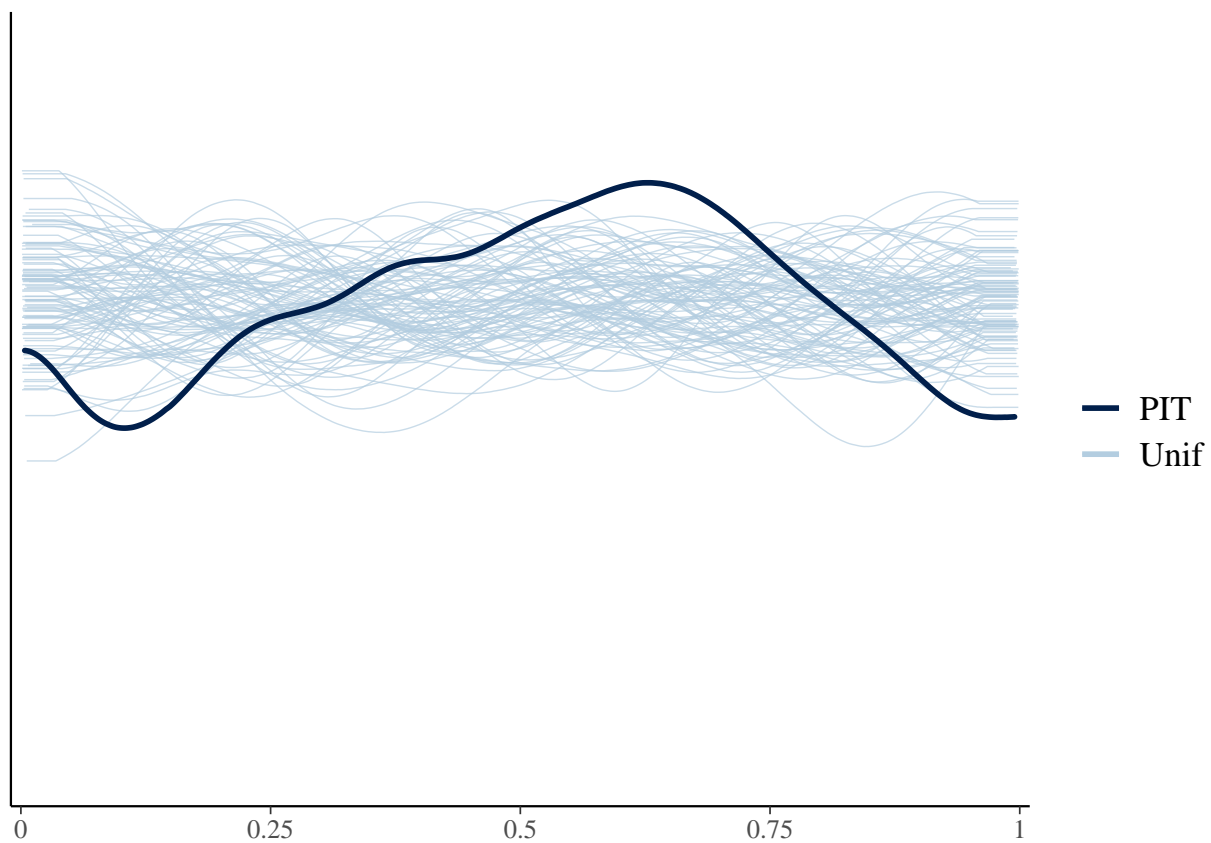
Comparing the two models tells us Model 2 is better:

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

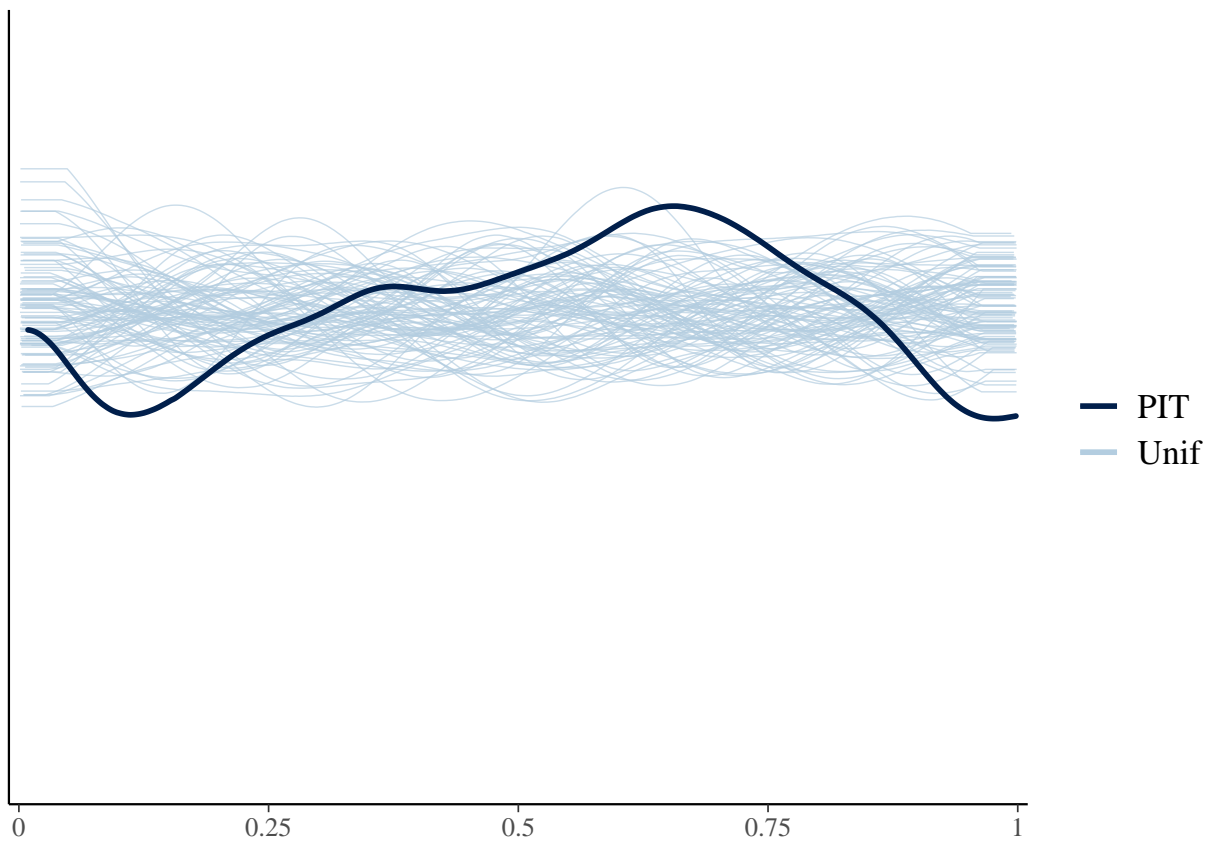
```
##           elpd_diff se_diff
## model2      0.0        0.0
## model1 -175.7       36.4
```

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.