Week 6: Visualizing the Bayesian Workflow

Priyanka Verma

24/02/23

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.

The data

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.

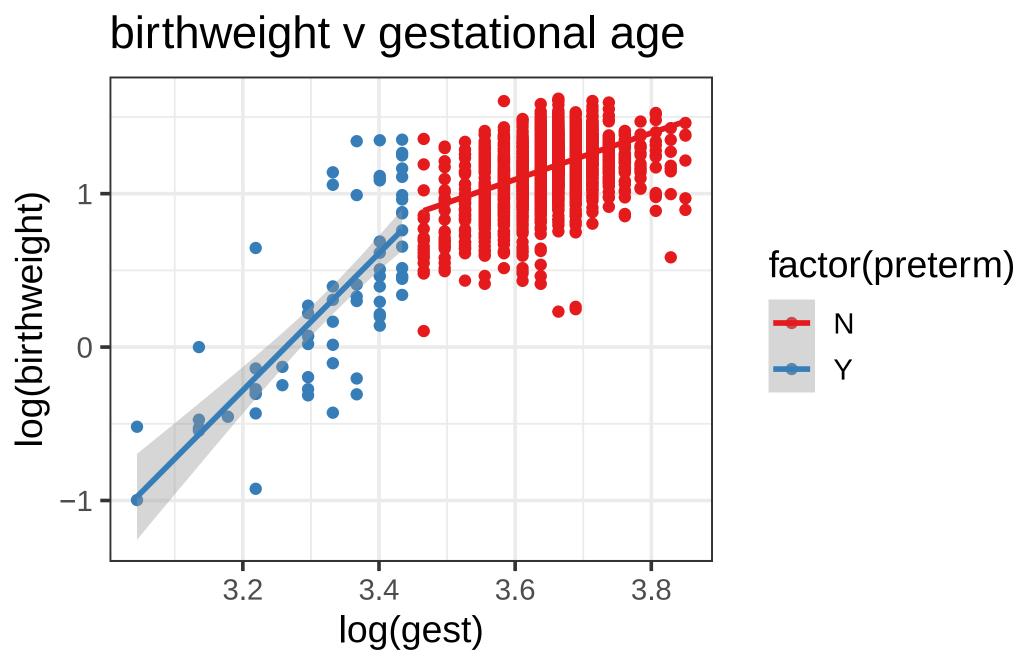
Question 1

- EDA 1
  The mean parameters of birthweight and gestational age for males and females babies are close to each other.

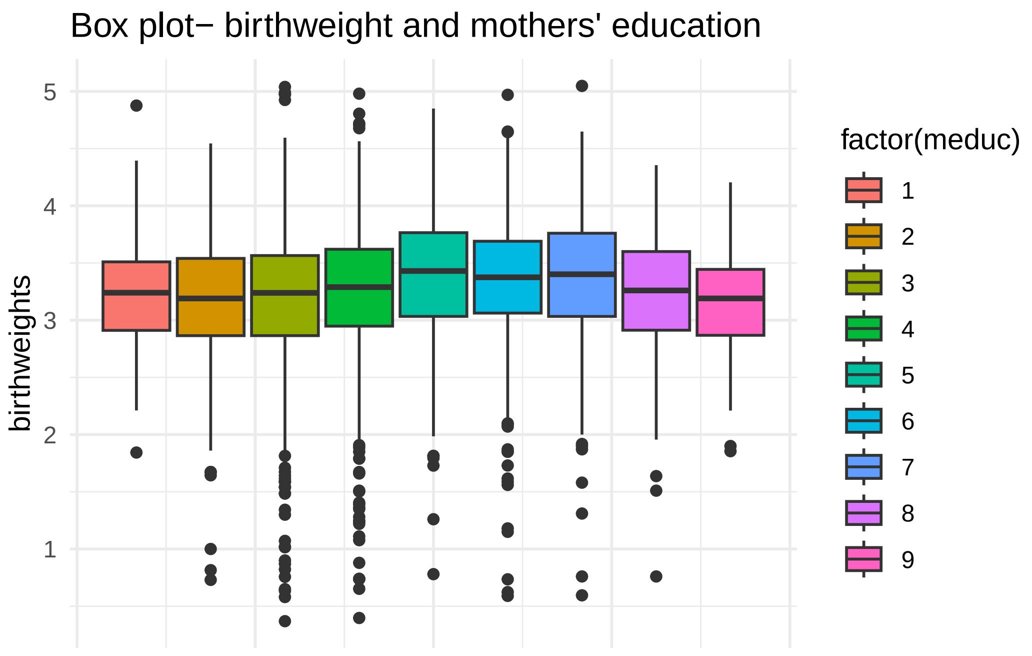
```
# A tibble: 2 x 3
  sex   mean_birthweight mean_gest
  <chr>            <dbl>     <dbl>
1 F                 3.20      38.6
2 M                 3.33      38.6
```

- EDA 2 **The relationship between weight and gestational age varies by whether or not the baby was premature. This evidence suggests a different relationship between the two variables, which lead us to consider interaction terms**

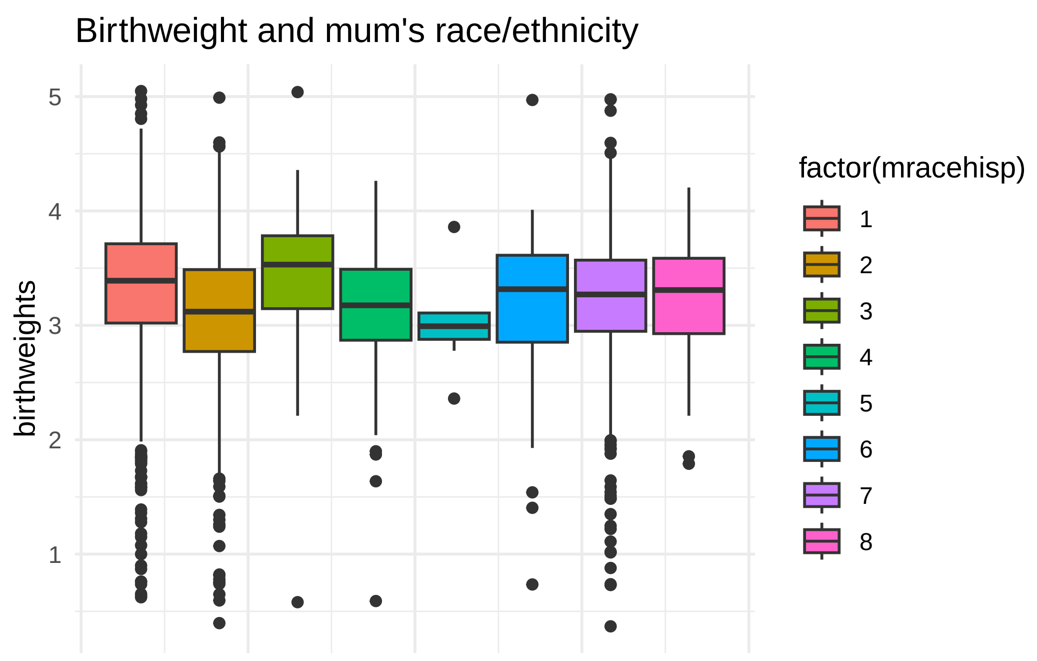
```
ds %>%
  ggplot(aes(log(gest), log(birthweight), color = factor(preterm))) +
  geom_point() + geom_smooth(method = "lm") +
  scale_color_brewer(palette = "Set1") +
  theme_bw(base_size = 14) +
  ggtitle("birthweight v gestational age")
```



- EDA 3 **The median birthweight values are similar for different values of mothers' education. Hence, this is likely to be not a good explanatory varibale in our model.**



- EDA 4 **there appears to be variation in birthweights with mum's race/ethnicity** This is likely because mothers from certain oppressed ethnicities might be less healthy and not have access to nutritive foods during pregnancy, which might cause low birthweight.



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

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

- 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)

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).

Question 2

For Model 1, simulate values of βs and σ based on the priors above. Do 1000 simulations.

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

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

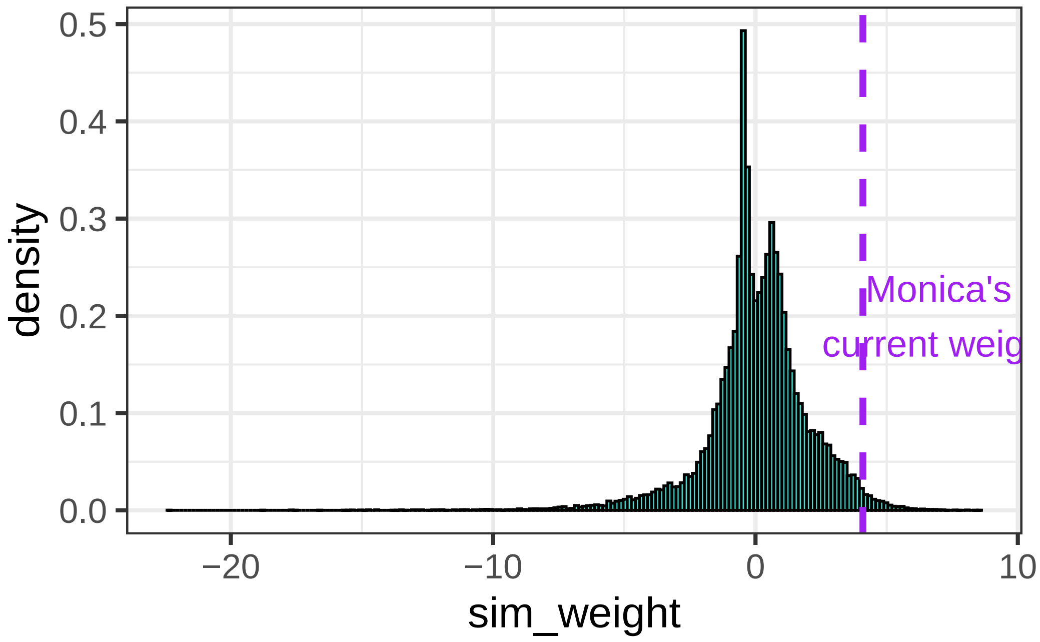
Use these values to simulate (log) birth weights from the likelihood specified in Model 1, based on the set of observed gestational weights.

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

```
set.seed(182)
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){
  this_mu <- beta0[i] + beta1[i]*dsims$log_gest_c
  dsims[paste0(i)] <- this_mu + rnorm(nrow(dsims), 0, sigma[i])
  #dsims[paste0(i)] <- rnorm(nrow(dsims), this_mu, sigma[i])}
```

```
}

dsl <- dsims %>%
  pivot_longer(`1`:`10`, names_to = "sim", values_to = "sim_weight")
```

```
dsl %>%
  ggplot(aes(sim_weight)) + geom_histogram(aes(y = ..density..), bins = 200, fill = "turqu
  geom_vline(xintercept = log(60), color = "purple", lwd = 1.2, lty = 2) +
  theme_bw(base_size = 16) +
  annotate("text", x=7, y=0.2, label= "Monica's\ncurrent weight", color = "purple", size =
```



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)) # normalised covarit
```

```
# 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

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

```
             mean      se_mean          sd      2.5%       25%       50%
beta[1] 1.1626455 8.308921e-05 0.002788610 1.1572315 1.1607747 1.1626489
beta[2] 0.1437272 7.471825e-05 0.002790495 0.1381709 0.1417134 0.1436255
sigma   0.1689026 9.977308e-05 0.001778135 0.1654527 0.1677226 0.1689555
              75%     97.5%     n_eff      Rhat
beta[1] 1.1645489 1.1681262 1126.3833 0.9985003
beta[2] 0.1455693 0.1493035 1394.7915 0.9966201
sigma   0.1700894 0.1721977  317.6162 1.0055867
```

```
#percent change in x (standarised birth --) is 0.14 percent change in y (--)
```

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.

```
beta1 = 1.1626455
beta2 = 0.1437272
est_birthweight = beta1 + beta2*((log(37) - mean(log(ds$gest)))/sd(log(ds$gest)))
print(exp(est_birthweight))
```

```
[1] 2.93641
```

Question 4

$$\log(y_i) \sim N(\beta_0 + \beta_1 \log(x_i) + \beta_2 z_i + \beta_3 \log(x_i) z_i, \sigma^2)$$

- 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)

A stan model to run Model 2, and run it.

```
ds$preterm <- ifelse(ds$preterm=="Y", 1, 0)
ds$log_weight <- log(ds$birthweight)
ds$log_gest_c <- (log(ds$gest) - mean(log(ds$gest)))/sd(log(ds$gest)) # normalised covarit

# put into a list
stan_data2 <- list(N = nrow(ds),
                   log_weight = ds$log_weight,
                   log_gest = ds$log_gest_c,
                   preterm = ds$preterm)
```

```
model2 <- stan(data = stan_data2,
             file = here("code/models/model2.stan"),
             iter = 500,
             seed = 243) #what does the seed do???
```

```
Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
clang -arch arm64 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Libra
In file included from <built-in>:1:
In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/S
In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/R
In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/R
/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen/include/Eigen
namespace Eigen {
^
/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen/include/Eigen
namespace Eigen {
               ^
               ;
In file included from <built-in>:1:
In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/S
In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/R
/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen/include/Eigen
#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 0.000532 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 5.32 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: 1.39 seconds (Warm-up)
Chain 1:                1.095 seconds (Sampling)
Chain 1:                2.485 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
Chain 2:
Chain 2: Gradient evaluation took 0.000358 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 3.58 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: 1.38 seconds (Warm-up)
Chain 2:                1.22 seconds (Sampling)
Chain 2:                2.6 seconds (Total)
Chain 2:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
Chain 3:
Chain 3: Gradient evaluation took 0.000301 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 3.01 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: 1.506 seconds (Warm-up)
Chain 3:                1.118 seconds (Sampling)
Chain 3:                2.624 seconds (Total)
Chain 3:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 0.000404 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 4.04 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: 1.402 seconds (Warm-up)
Chain 4:                1.144 seconds (Sampling)
Chain 4:                2.546 seconds (Total)
Chain 4:
```

```
summary(model2)$summary[c(paste0("beta[", 1:4, "]"), "sigma"),]
```

```
             mean      se_mean          sd       2.5%        25%       50%
beta[1] 1.1695816 8.043482e-05 0.002763499 1.16408193 1.16767119 1.1696482
beta[2] 0.1017462 1.245915e-04 0.003672348 0.09432898 0.09916171 0.1017986
beta[3] 0.5622527 3.201190e-03 0.062700772 0.44239416 0.51717307 0.5639181
beta[4] 0.1982959 6.692580e-04 0.012844289 0.17411041 0.18917477 0.1989054
sigma   0.1612724 7.166526e-05 0.001799254 0.15796862 0.16007825 0.1611842
              75%     97.5%     n_eff      Rhat
beta[1] 1.1714861 1.1748458 1180.4034 0.9992262
beta[2] 0.1041868 0.1088904  868.7818 1.0060038
beta[3] 0.6023427 0.6861491  383.6390 1.0045765
beta[4] 0.2070525 0.2237837  368.3265 1.0045237
sigma   0.1624722 0.1647869  630.3296 1.0032030
```

Question 5

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
```

The results are same for both the models- the calculated model and the uploaded model.

PPCs

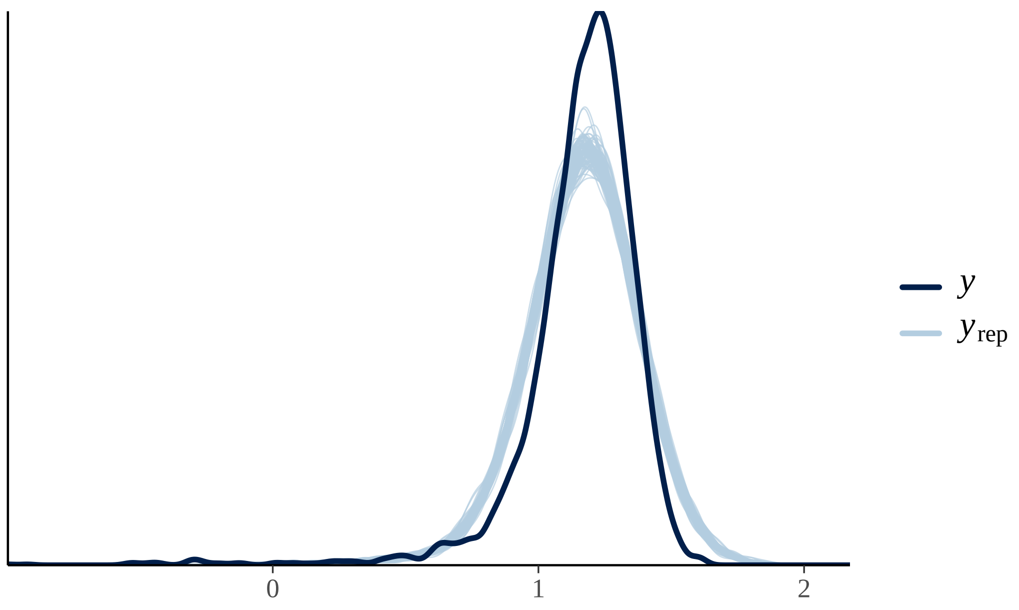
Now we've run two candidate models let's do some posterior predictive checks. The `bayesplot` package has a lot of inbuilt graphing functions to do this. For example, let's plot the distribution of our data (y) against 100 different datasets drawn from the posterior predictive distribution:

```r
set.seed(1856)
y <- ds$log_weight
yrep1 <- extract(mod1)[["log_weight_rep"]]
yrep2 <- extract(mod2)[["log_weight_rep"]]
samp100 <- sample(nrow(yrep1), 100) #sampling 100 random ones
dim(yrep1) # 3842 observations = N . for each of those i have a 1000 posterior predictive
```

```
[1] 1000 3842
```

```r
ppc_dens_overlay(y, yrep1[samp100, ]) + ggtitle("distribution of observed versus predicted
```

distribution of observed versus predicted birthweights

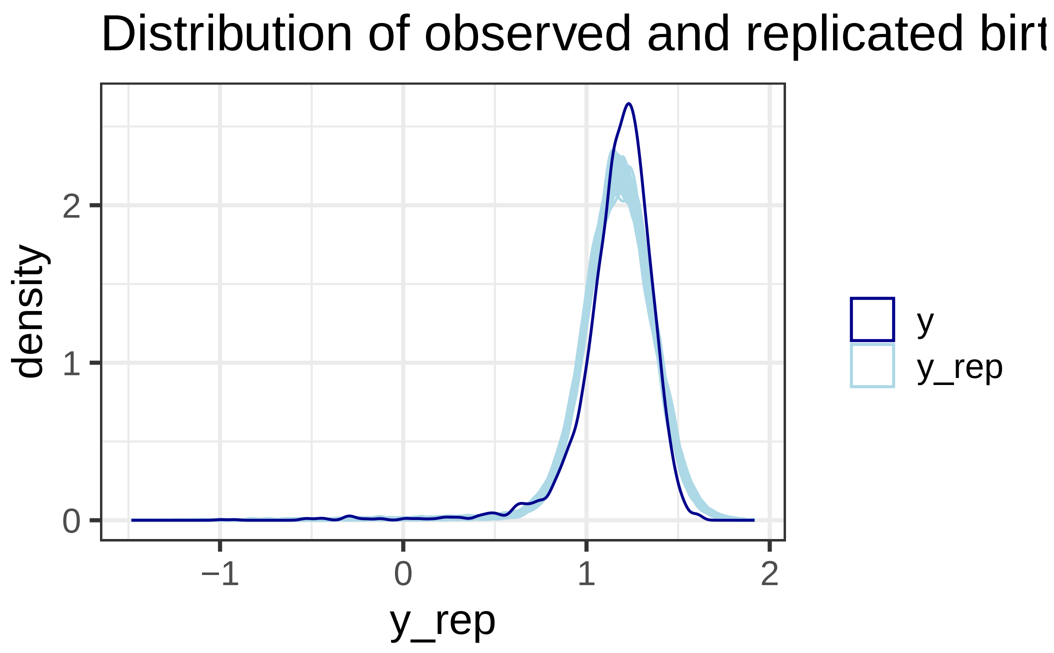


Question 6

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`)

```
yrep2 <- extract(mod2)[["log_weight_rep"]]
N <- nrow(ds)

# first, get into a tibble
rownames(yrep2) <- 1:nrow(yrep2)
dr <- as_tibble(t(yrep2))
dr <- dr %>% bind_cols(i = 1:N, log_weight_obs = log(ds$birthweight))

# turn into long format; easier to plot
dr <- dr %>%
  pivot_longer(-(i:log_weight_obs), names_to = "sim", values_to ="y_rep")

#dim(yrep2) # 3842 observations = N . for each of those i have a 1000 posterior predictive
```

```
# filter to just include 100 draws and plot!
dr %>%
  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(name = "",
                     values = c("y" = "darkblue",
                                "y_rep" = "lightblue")) +
  ggtitle("Distribution of observed and replicated birthweights") +
  theme_bw(base_size = 16)
```

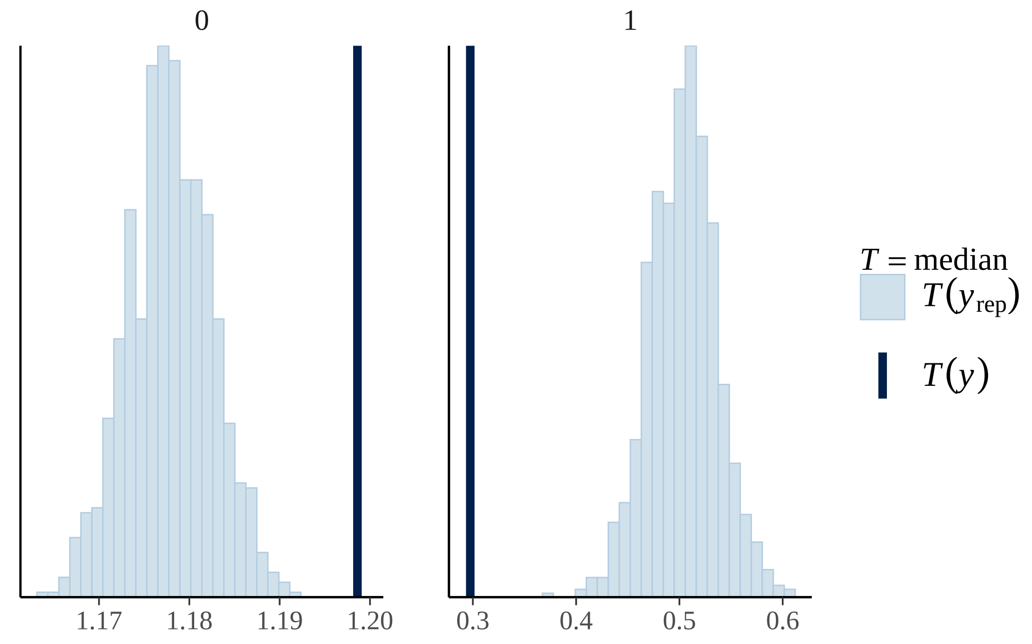


Test statistics

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')
```

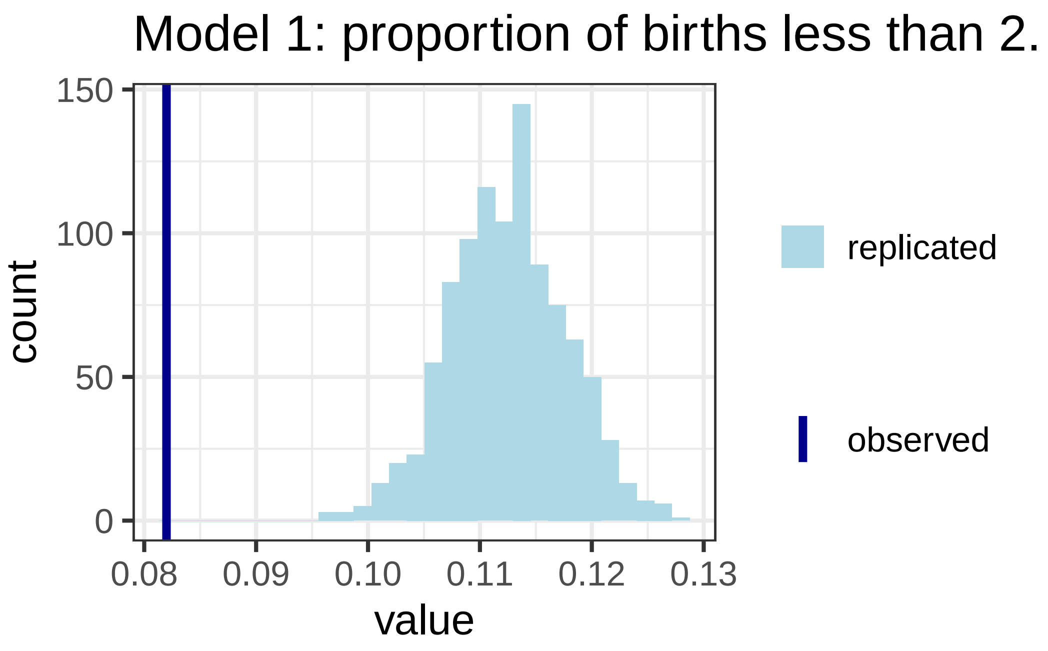


Question 7

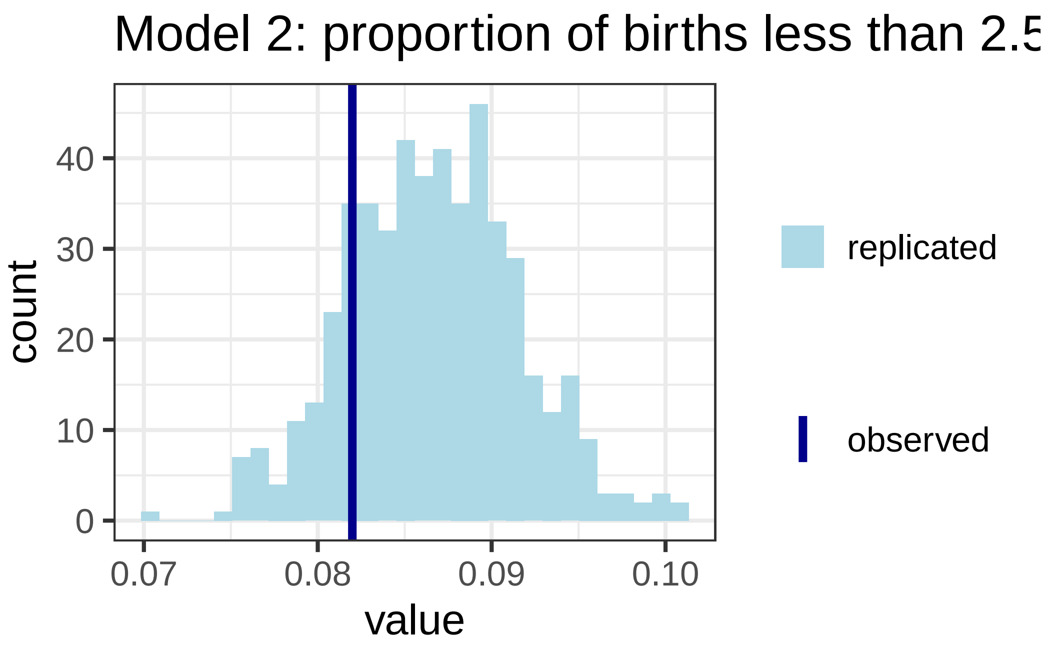
Use a test statistic of the proportion of births under 2.5kg. Calculate the test statistic for the data, and the posterior predictive samples for both models, and plot the comparison (one plot per model).

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

```
ggplot(data = as_tibble(t_y_rep), aes(value)) +
    geom_histogram(aes(fill = "replicated")) +
    geom_vline(aes(xintercept = t_y, color = "observed"), lwd = 1.5) +
  ggtitle("Model 1: proportion of births less than 2.5kg") +
  theme_bw(base_size = 16) +
  scale_color_manual(name = "",
                     values = c("observed" = "darkblue"))+
  scale_fill_manual(name = "",
                     values = c("replicated" = "lightblue"))
```



```
ggplot(data = as_tibble(t_y_rep_2), aes(value)) +
    geom_histogram(aes(fill = "replicated")) +
    geom_vline(aes(xintercept = t_y, color = "observed"), lwd = 1.5) +
  ggtitle("Model 2: proportion of births less than 2.5kg") +
  theme_bw(base_size = 16) +
  scale_color_manual(name = "",
                     values = c("observed" = "darkblue"))+
  scale_fill_manual(name = "",
                     values = c("replicated" = "lightblue"))
```



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   1377.5  72.5
p_loo         9.1   1.3
looic     -2755.1 145.0
------
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.
```

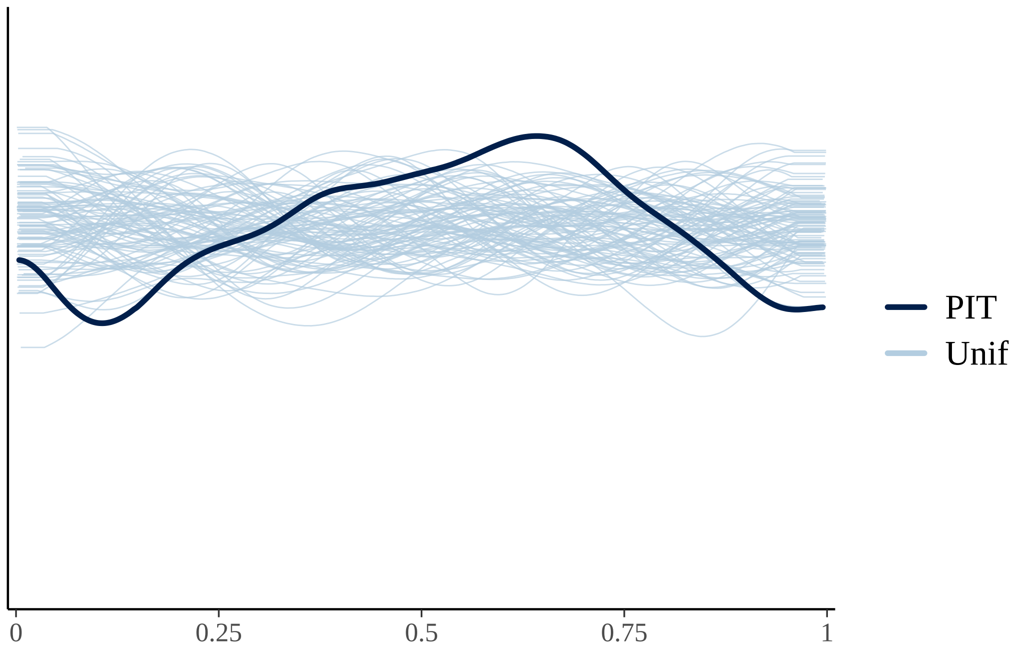
Comparing the two models tells us Model 2 is better:

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

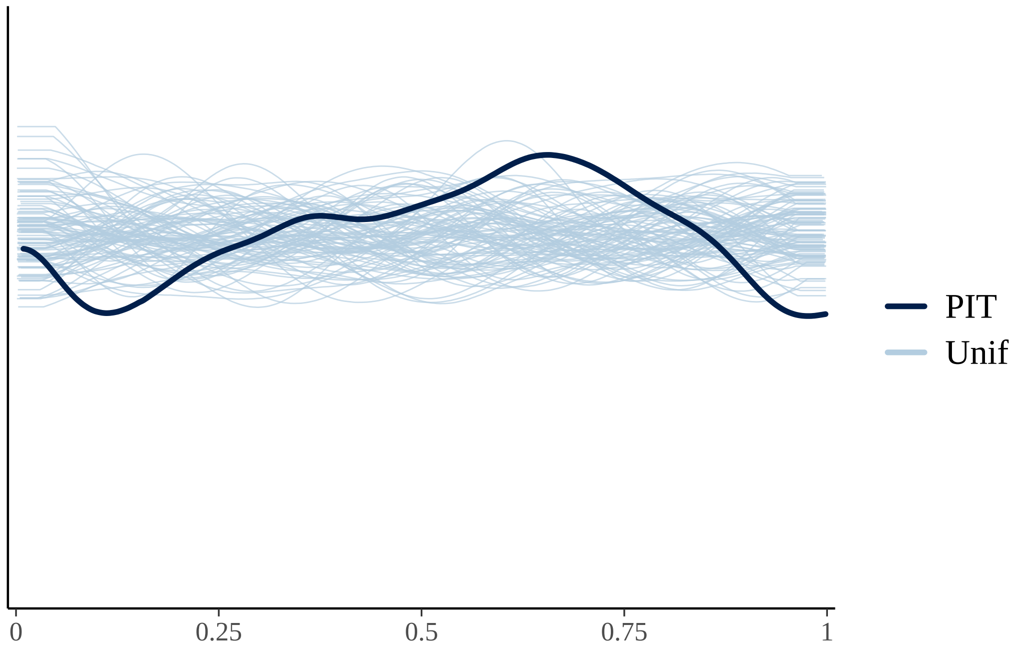
```
       elpd_diff se_diff
model2    0.0       0.0
model1 -175.3      36.2
```

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))
```



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



Bonus question (not required)

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

Question 8

Based on the original dataset, choose one (or more) additional covariates to add to the linear regression model. Run the model in Stan, and compare with Model 2 above on at least 2 posterior predictive checks.

```
stan_data3 <- list(N = nrow(ds),
                   log_weight = ds$log_weight,
                   log_gest = ds$log_gest_c,
                   preterm = ds$preterm,
                   sex = ds$sex_bin)
```

```
model3 <- stan(data = stan_data3,
             file = here("code/models/model3-Q8.stan"),
             iter = 500,
             seed = 243)
```

```
Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
clang -arch arm64 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Libra
In file included from <built-in>:1:
In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/S
In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/R
In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/R
/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen/include/Eigen
namespace Eigen {
^
/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen/include/Eigen
namespace Eigen {
               ^
               ;
In file included from <built-in>:1:
In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/S
In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/R
/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen/include/Eigen
#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 0.000586 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 5.86 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: 1.703 seconds (Warm-up)
Chain 1:                1.464 seconds (Sampling)
Chain 1:                3.167 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
Chain 2:
Chain 2: Gradient evaluation took 0.00038 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 3.8 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: 2.108 seconds (Warm-up)
Chain 2:                1.484 seconds (Sampling)
Chain 2:                3.592 seconds (Total)
Chain 2:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
Chain 3:
Chain 3: Gradient evaluation took 0.000382 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 3.82 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: 2.087 seconds (Warm-up)
Chain 3:                1.784 seconds (Sampling)
Chain 3:                3.871 seconds (Total)
Chain 3:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 0.00038 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 3.8 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: 1.953 seconds (Warm-up)
Chain 4:                1.44 seconds (Sampling)
Chain 4:                3.393 seconds (Total)
Chain 4:
```

```
summary(model3)$summary[c(paste0("beta[", 1:5, "]"), "sigma"),]
```

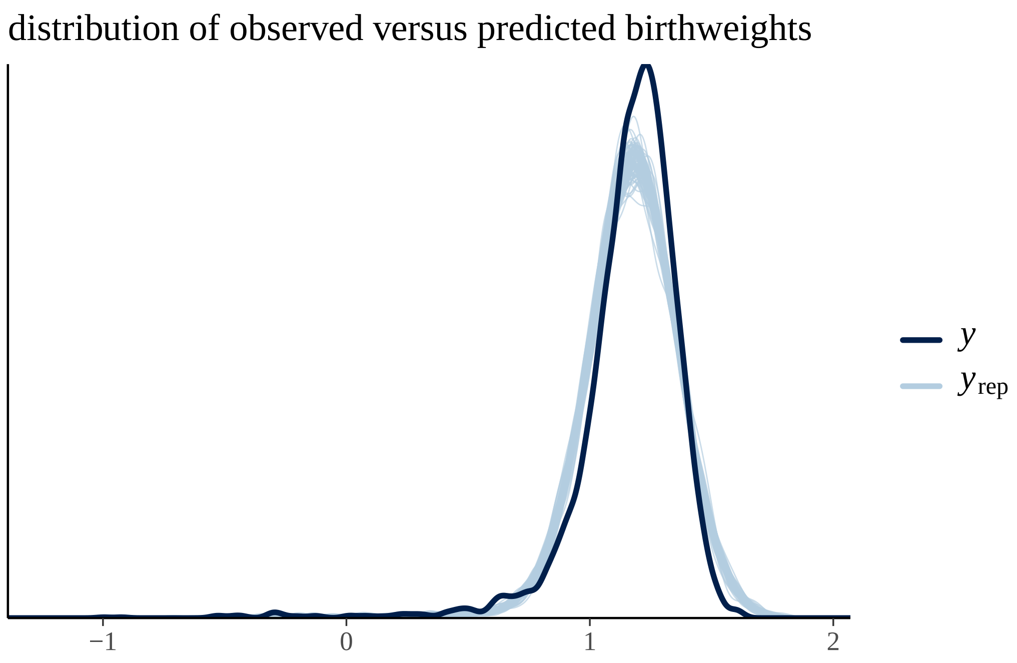
```
              mean      se_mean          sd       2.5%       25%        50%
beta[1] 1.14863087 0.0001274426 0.003545570 1.14170861 1.1462625 1.14854878
beta[2] 0.10253871 0.0001088599 0.003589063 0.09549912 0.1001183 0.10251359
beta[3] 0.54948221 0.0036485470 0.061985535 0.42817851 0.5050605 0.54850831
beta[4] 0.19506220 0.0007276946 0.012672510 0.17126134 0.1859093 0.19515730
beta[5] 0.04189216 0.0001915916 0.004992674 0.03187982 0.0386048 0.04198679
sigma   0.15988300 0.0000697319 0.001818419 0.15659048 0.1585737 0.15982078
               75%      97.5%     n_eff      Rhat
beta[1] 1.15095827 1.15578506  774.0042 1.0010718
beta[2] 0.10501355 0.10958428 1086.9920 0.9999179
beta[3] 0.59072886 0.67131069  288.6296 1.0109358
beta[4] 0.20346964 0.22015458  303.2683 1.0095914
beta[5] 0.04534553 0.05158414  679.0682 1.0043410
sigma   0.16107068 0.16362635  680.0253 1.0015901
```

PPC for Model 3

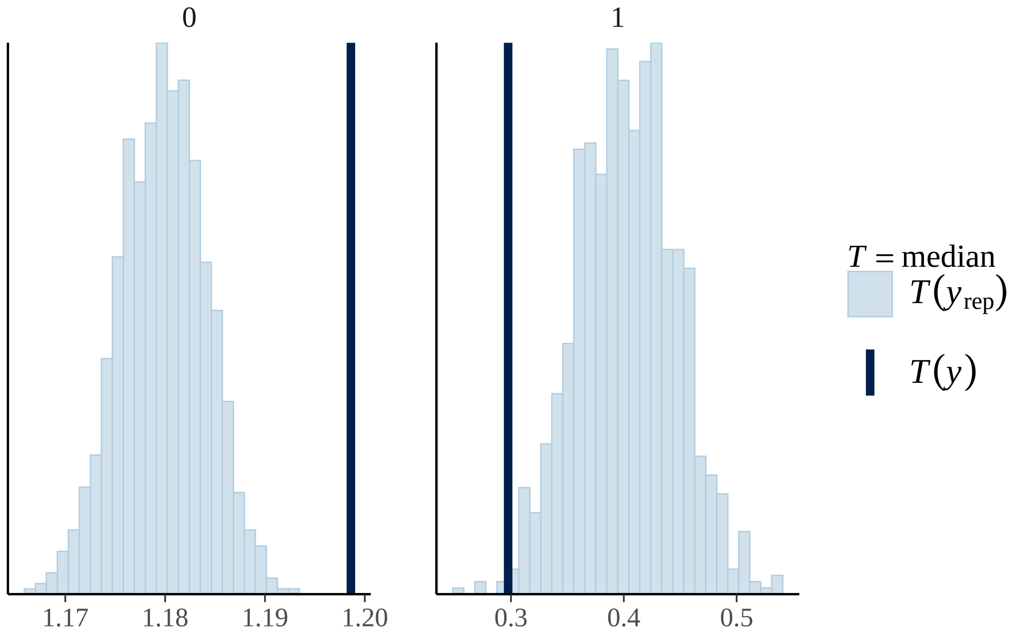
```
set.seed(1856)
y <- ds$log_weight
yrep3 <- extract(model3)[["log_weight_rep"]]
samp100 <- sample(nrow(yrep3), 100) #sampling 100 random ones
dim(yrep3) # 3842 observations = N . for each of those i have a 1000 posterior predictive
```

```
[1] 1000 3842
```

```
ppc_dens_overlay(y, yrep3[samp100, ]) + ggtitle("distribution of observed versus predicted
```



```
ppc_stat_grouped(ds$log_weight, yrep3, group = ds$preterm, stat = 'median')
```



```
loglik3 <- extract(model3)[["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.

```
loo3 <- loo(loglik3, save_psis = TRUE)
loo2 <- loo(loglik2, save_psis = TRUE)
```

Look at the output:

```
loo3
```

```
Computed from 1000 by 3842 log-likelihood matrix

         Estimate    SE
elpd_loo   1584.8  70.3
p_loo        16.1   2.3
looic     -3169.5 140.7
------
```

```
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.
```

```r
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 3 is better:

```r
loo_compare(loo3, loo2)
```

```
       elpd_diff se_diff
model1   0.0       0.0  
model2 -31.9       8.1  
```