Lab Exercise - Bayes

Luis Correia - Student No. 1006508566

February 26th 2020

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

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
library(ggplot2)
library(GGally)
library(matrixStats)

ds <- read_rds("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 (body mass index)

- `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)
```

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.

In order to analyse the data we will produce some scatter plots of some pairs variables of interest and additional plots to elucidate further relations that might be interest to investigate.

Main variables include the following pairs and cross-analysis:

- Paired Scatterplots: Mom Age, Mom BMI, Gestational Age, Birth Weight, Education, Race
- Groups: Race, Preterm Status, Education, Sex
- Categories: Race, Sex, Preterm Status on boxplots

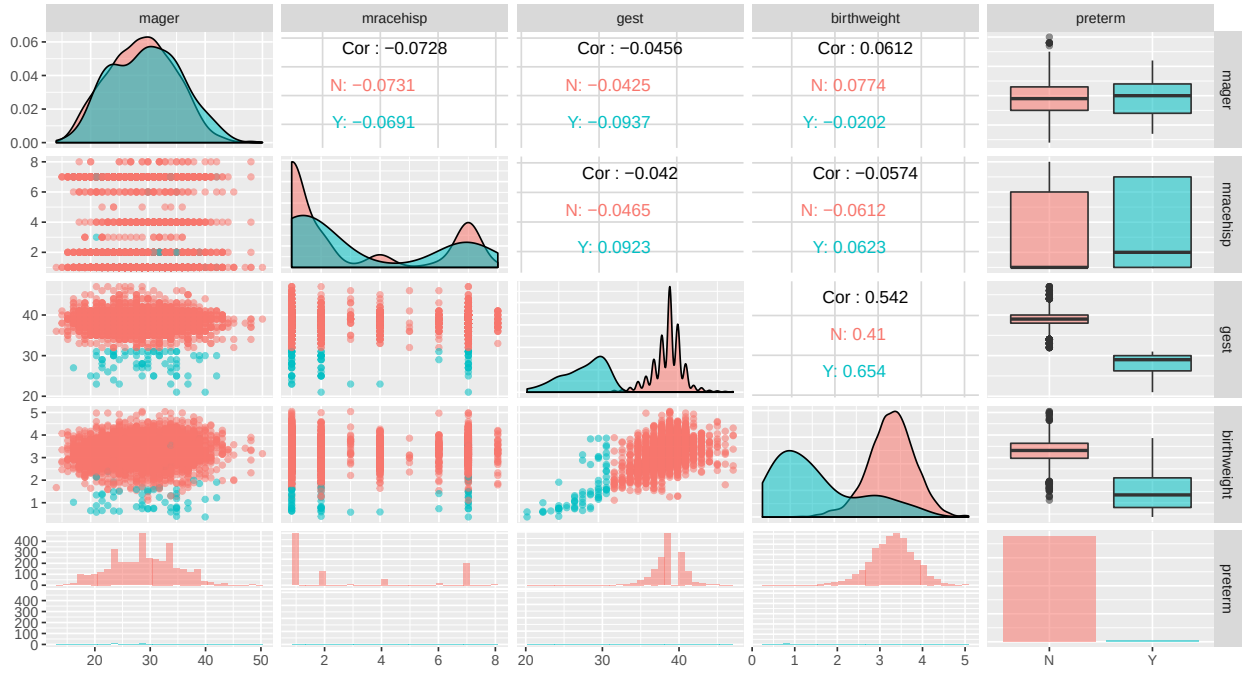
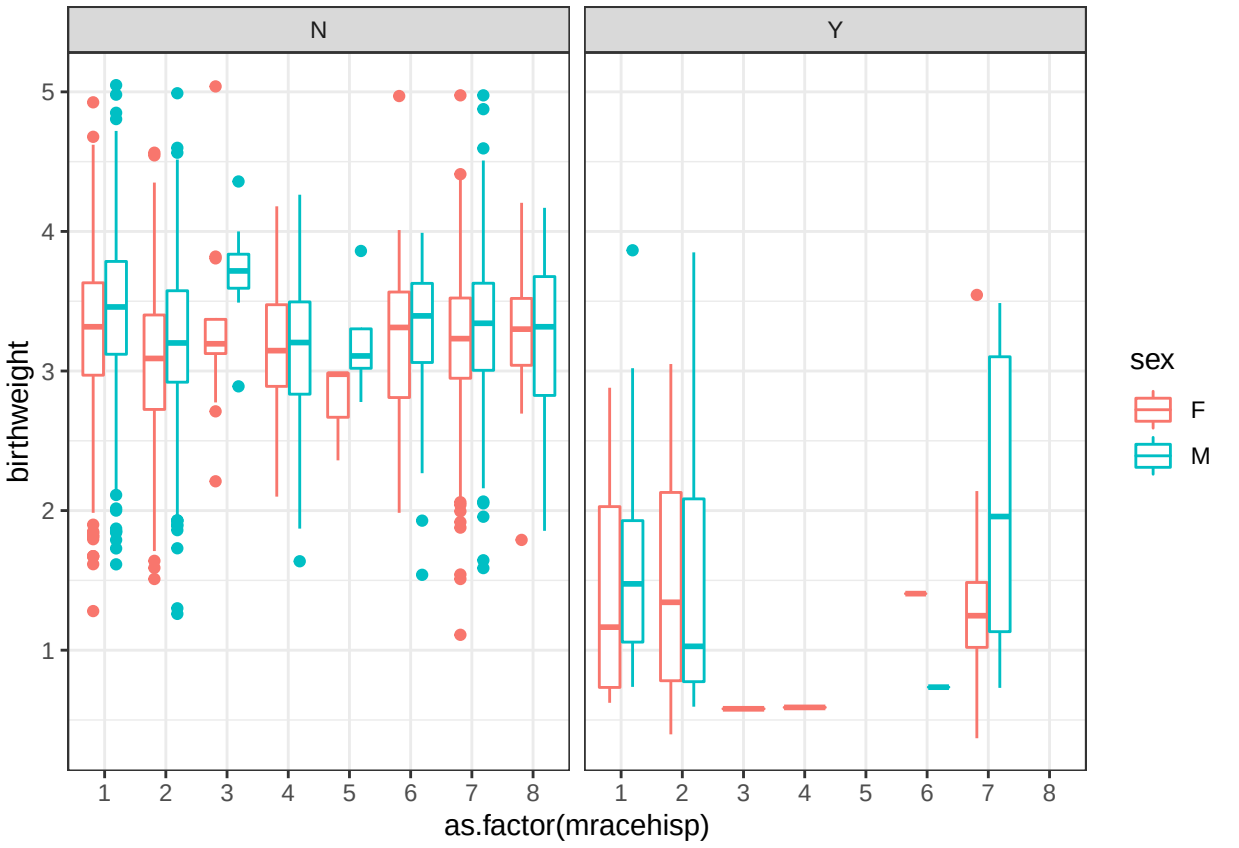


Figure 1: Scatterplots of Variables of interest - groups: Preterm

```
# Additional Boxplots to visualize differences considering categorical variables
ds %>%
  ggplot(aes(as.factor(mracehisp), birthweight))+
  facet_wrap(~preterm) +
  geom_boxplot(aes(colour=sex)) +
  theme_bw()
```



Preliminary findings:

- From Scatterplots, we can observe some possible influences between variables and the variable of interest `birthweight`. Gestational duration which have a positive influence. We noted that `preterm = Yes` babies present lower `birthweight` than group `preterm = No`, suggesting this variable can contribute as an additional predictor in our model;
- Another characteristic that seems to play significant influence is the `preterm` status, where babies under 32 weeks of gestational age presents aparently lower weight than if they were not preterm. This can be observed from boxplot above.
- Other covatiates such as `race` could be further investigated but doesn't appear to play a fundamental role.

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)

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

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

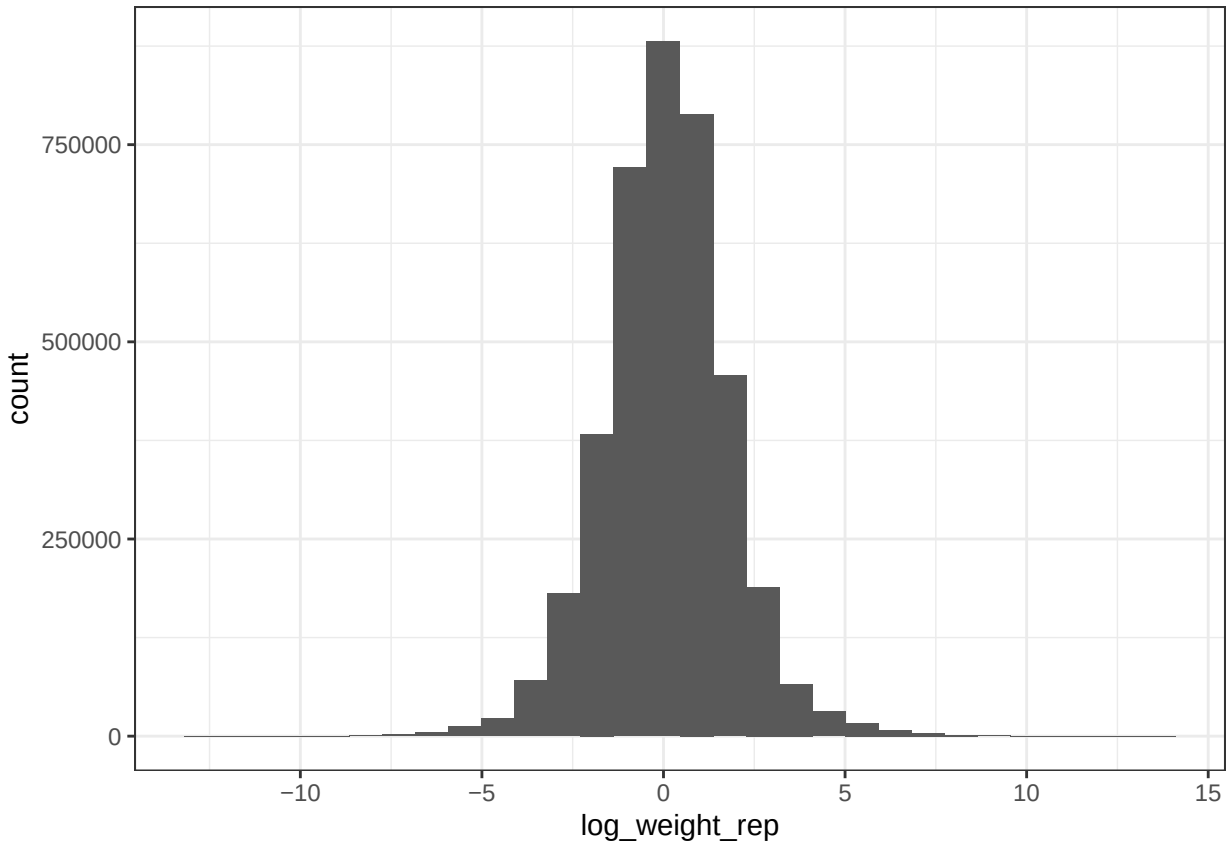
where the plus means positive values only i.e. Half Normal.

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. 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 weeks should be centered and standardized**.

```
set.seed(182)
nsims <- 1000
sigma <- abs(rnorm(mean = 0, sd = 1, nsims))
beta0 <- rnorm(mean = 0, sd = 1, nsims)
beta1 <- rnorm(mean = 0, sd = 1, nsims)

# a tibble to store simulations
# we will calculate likelihood based on observed set of (log, centered, standarized) gestational length
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$log_gest_c[i]
  dsims[paste0(i)] <- rnorm(mean = this_mu, sd = sigma[i], length(ds$gest))
}

dstp <- dsims %>%
  pivot_longer(`1`:paste0(nsims), names_to = "sim", values_to = "log_weight_rep")

dstp %>%
  ggplot(aes(log_weight_rep))+
  geom_histogram()+
  theme_bw()
```



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 = "simple_weight.stan",
             iter = 250,
             seed = 243)
```

```
##
## SAMPLING FOR MODEL 'simple_weight' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.358 seconds (Warm-up)
## Chain 1:                0.27 seconds (Sampling)
## Chain 1:                0.628 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: 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: 0.346 seconds (Warm-up)
## Chain 2:                0.265 seconds (Sampling)
## Chain 2:                0.611 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: 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: 0.324 seconds (Warm-up)
## Chain 3:                0.278 seconds (Sampling)
## Chain 3:                0.602 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: 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: 0.27 seconds (Warm-up)
## Chain 4:                0.288 seconds (Sampling)
## Chain 4:                0.558 seconds (Total)
## Chain 4:
```

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

```
##              mean      se_mean          sd      2.5%       25%       50%
## beta[1] 1.1625947 9.950197e-05 0.002658321 1.1574088 1.1608640 1.1625474
## beta[2] 0.1438209 9.829892e-05 0.002655021 0.1391351 0.1418923 0.1438726
## sigma   0.1689788 2.120098e-04 0.002118172 0.1647000 0.1675429 0.1687997
##               75%     97.5%     n_eff      Rhat
## beta[1] 1.1644051 1.1675475 713.75871 0.9943006
## beta[2] 0.1455711 0.1491824 729.52203 1.0038659
## sigma   0.1703457 0.1738574  99.81836 1.0229139
```

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 = "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 0.001 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: 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: 1.41 seconds (Warm-up)
## Chain 1:                1.917 seconds (Sampling)
## Chain 1:                3.327 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'simple_weight_preterm_int' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.001 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 10 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: 1.69 seconds (Warm-up)
## Chain 2:                1.453 seconds (Sampling)
## Chain 2:                3.143 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'simple_weight_preterm_int' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0.001 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 10 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: 1.643 seconds (Warm-up)
## Chain 3:                2.255 seconds (Sampling)
## Chain 3:                3.898 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: 1.738 seconds (Warm-up)
## Chain 4:                2.954 seconds (Sampling)
## Chain 4:                4.692 seconds (Total)
## Chain 4:
```

```
# My order is - intercept, log_gest, preterm, interaction
summary(mod2)$summary[c("beta[1]", "beta[2]", "gamma[1]", "gamma[2]", "sigma"),]
```

```
##               mean      se_mean          sd       2.5%        25%       50%
## beta[1]  1.1695294 9.719726e-05 0.002705732 1.16415137 1.16775349 1.1695605
## beta[2]  0.1018708 1.686418e-04 0.003569473 0.09502386 0.09938769 0.1018038
## gamma[1] 0.5589914 8.292839e-03 0.064514523 0.42394500 0.51814537 0.5606322
## gamma[2] 0.1978094 1.645173e-03 0.013489043 0.17126853 0.18890484 0.1980472
## sigma    0.1611081 1.164977e-04 0.001750955 0.15804497 0.15983706 0.1610556
##                75%     97.5%     n_eff      Rhat
## beta[1]  1.1712265 1.1748947 774.92813 0.9971959
## beta[2]  0.1044569 0.1085821 447.99996 1.0051916
## gamma[1] 0.6026402 0.6948666  60.52133 1.0513522
## gamma[2] 0.2070215 0.2242600  67.22630 1.0452493
## sigma    0.1624550 0.1645481 225.89972 1.0211297
```

Question 4

For reference I have uploaded some model 2 results. Check your results are similar. Note: order of my covariates is: intercept, preterm, log_gest, interaction.

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

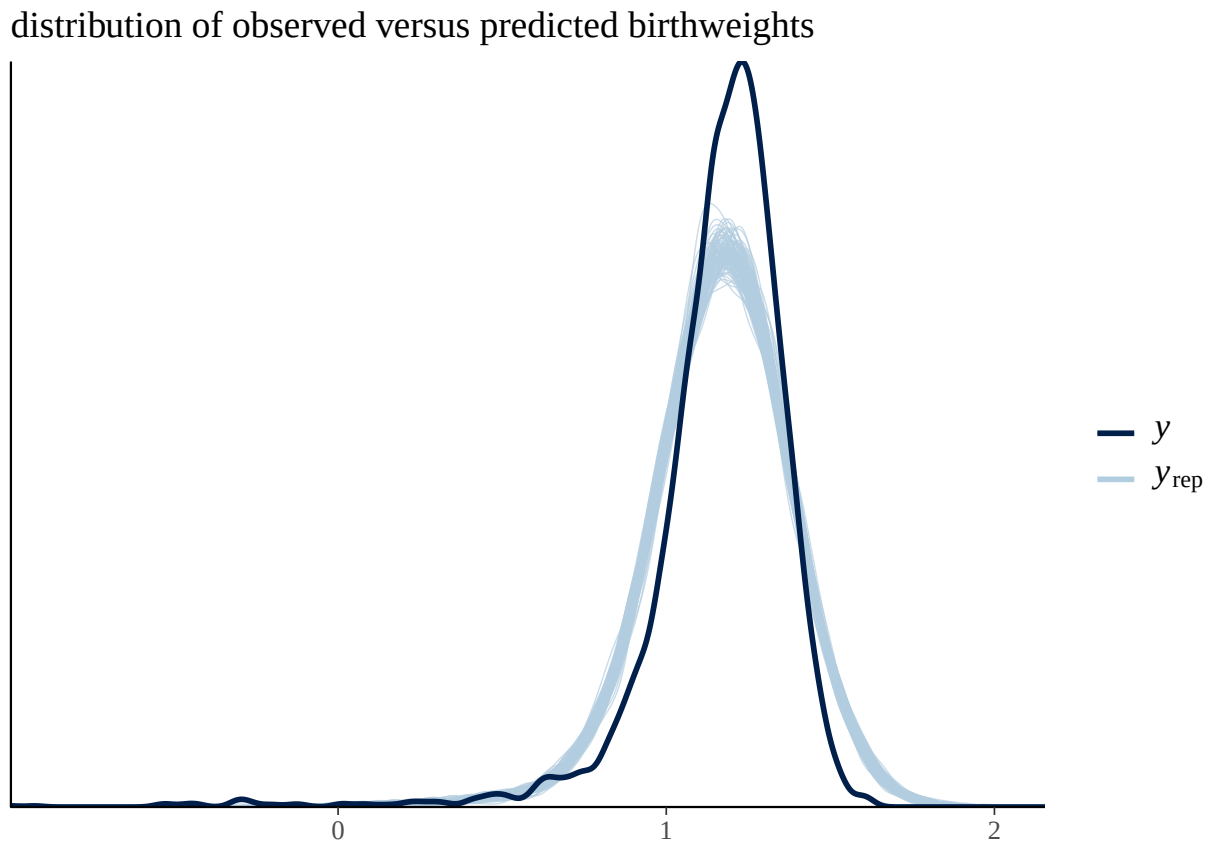
```
##              mean      se_mean          sd       2.5%        25%       50%
## beta[1] 1.1697241 1.385590e-04 0.002742186 1.16453578 1.16767109 1.1699278
## beta[2] 0.5563133 5.835253e-03 0.058054991 0.43745504 0.51708255 0.5561553
## beta[3] 0.1020960 1.481816e-04 0.003669476 0.09459462 0.09997153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

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:

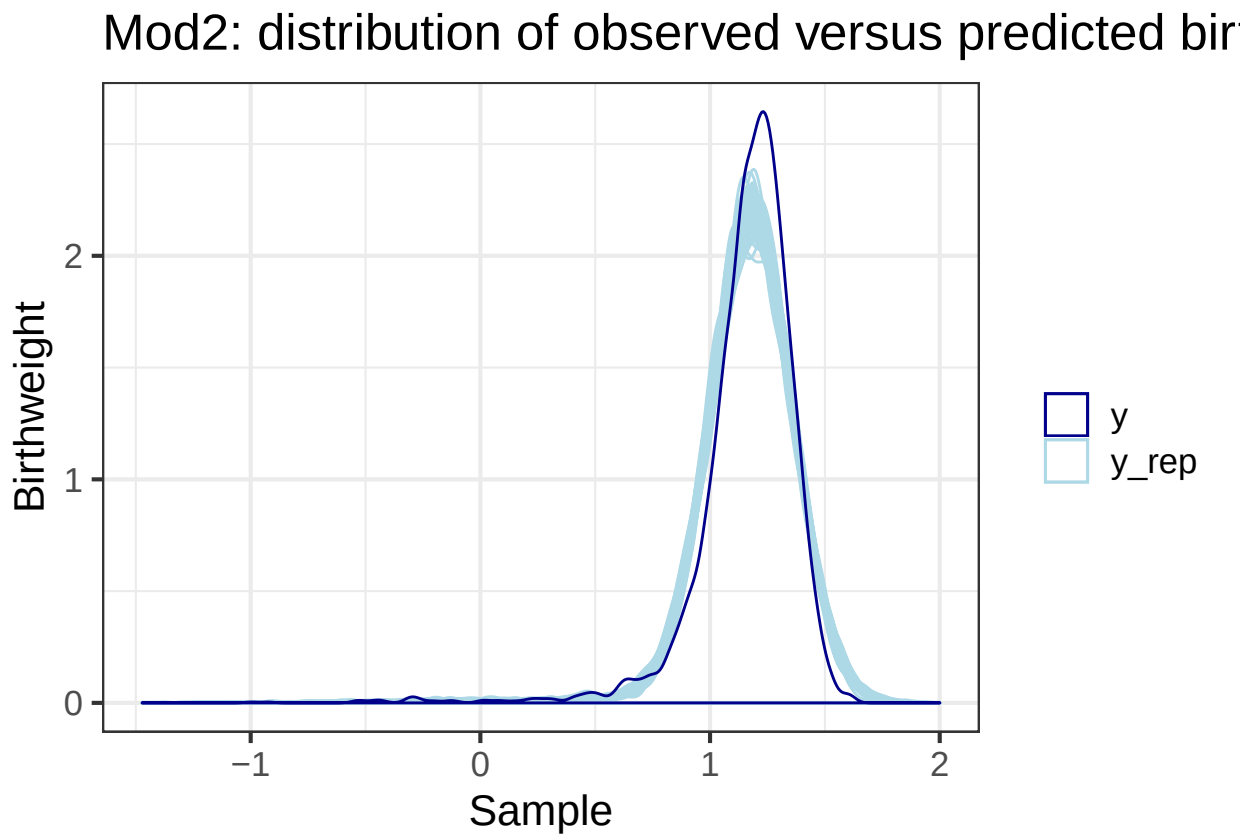
```
set.seed(1856)
y <- ds$log_weight
yrep1 <- extract(mod1)[["log_weight_rep"]]
yrep2 <- extract(mod2)[["log_weight_rep"]] # will need mod2 for later
samp100 <- sample(nrow(yrep1), 100)
ppc_dens_overlay(y, yrep1[samp100, ])  + ggtitle("distribution of observed versus predicted birthweights
```



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`)

```r
N <- length(ds$birthweight)

set.seed(1856)
samp100 <- sample(nrow(yrep2), 100)

#Doing the same by just constructing other dataframe
dpMod2 <- as.tibble(data.frame(sim = 1:100, y_rep = yrep2[samp100, ]))

colnames(dpMod2) <- c("sim",1:N)

dpMod2 <- dpMod2 %>%
  pivot_longer(`1`:paste0(N), names_to = "pt", values_to = "y_rep")

# plot densities for 100 samples
dpMod2 %>%
  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")) +
  labs(x = "Sample",
       y = "Birthweight") +
  ggtitle("Mod2: distribution of observed versus predicted birthweights") +
  theme_bw(base_size = 16)
```

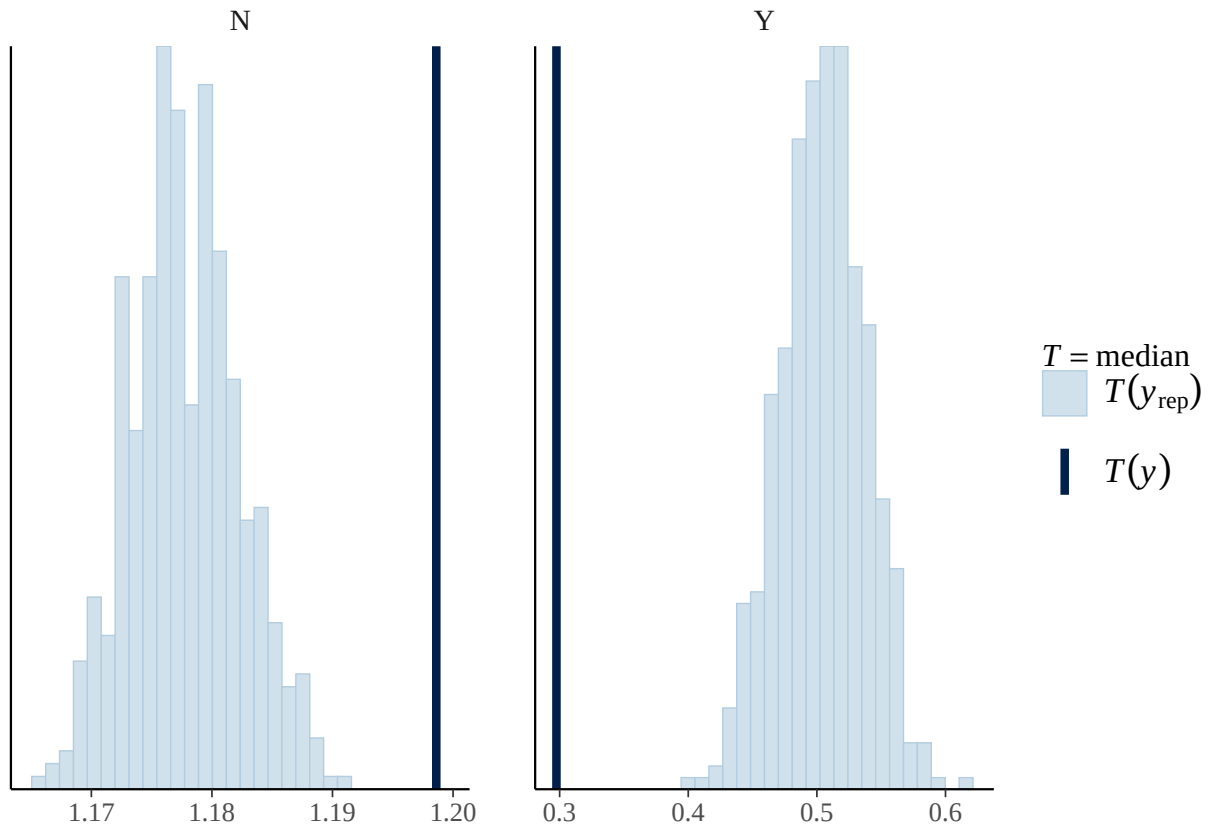


Test statistics

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')
```

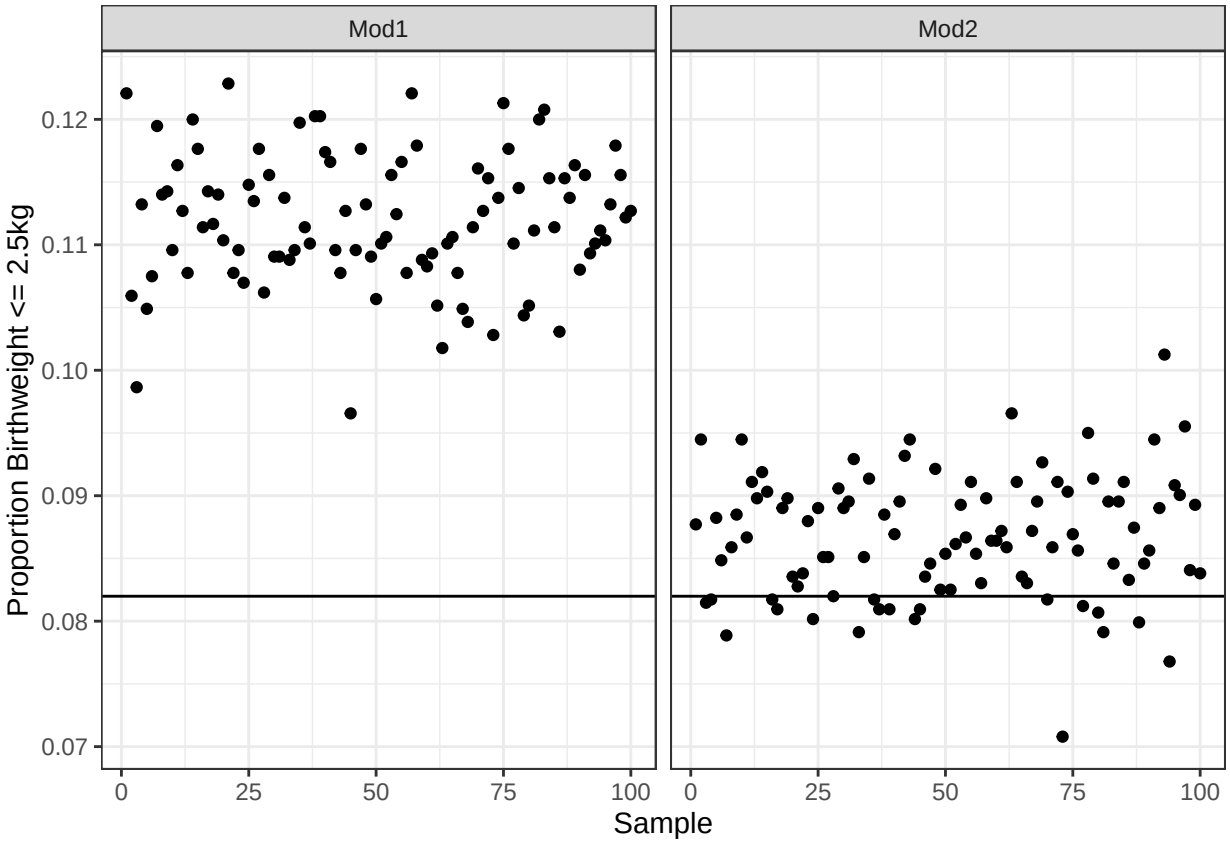


Question 6

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).

```
N <- length(ds$birthweight)
S <- 100
wBlM1 <- vector(length = N)
wBlM2 <- vector(length = N)


wBl <- (ds %>%
  filter(birthweight<=2.5) %>%
  summarise(c = n()))$c/N

# Calculate proportion of babies under 2.5 for a sample from Model 1
sampS1 <- sample(nrow(yrep1), S)

for (i in 1:S)
  wBlM1[i] <- count(exp(yrep1[sampS1,1:N])[i,]<=2.5)/N

# Calculate proportion of babies under 2.5 for a sample from Model 1
sampS2 <- sample(nrow(yrep2), S)

for (i in 1:S)
  wBlM2[i] <- count(exp(yrep2[sampS2,1:N])[i,]<=2.5)/N
```

```
# Generate data-frame to plot the curves and compare with real
dpr <- as.tibble(rbind(data.frame(id = 1:S, prop = wBlM1[1:S], model = rep("Mod1", S)),
                       data.frame(id = 1:S, prop = wBlM2[1:S], model = rep("Mod2", S))))

# Plot the graph to compare samples from each model and the observed statistic
p <- dpr %>%
  ggplot(mapping = aes(x = id, y = prop)) +
  facet_wrap(~model) +
  geom_point() +
  labs(x = "Sample",
       y = "Proportion Birthweight <= 2.5kg") +
  theme_bw()

p + geom_hline(yintercept = wBl)
```



As per this simulation, we can verify that Model 2 projects with more accuracy the statistics of proportion of babies under 2.5kg - the horizontal line represents the observer proportion.

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 500 by 3842 log-likelihood matrix
##
##          Estimate    SE
## elpd_loo   1377.4  72.4
## p_loo         9.5   1.4
## looic     -2754.8 144.9
## ------
## 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.
```

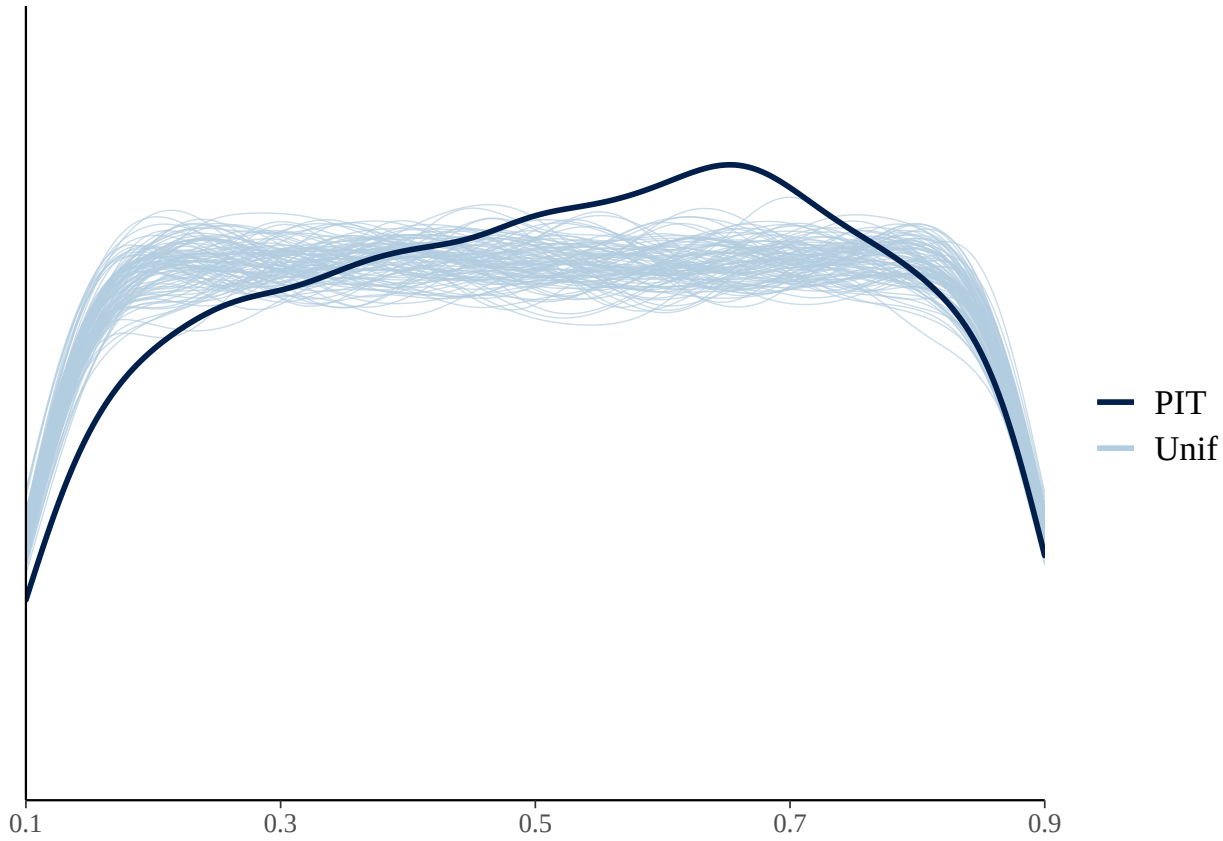
Comparing the two models tells us Model 2 is better:

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

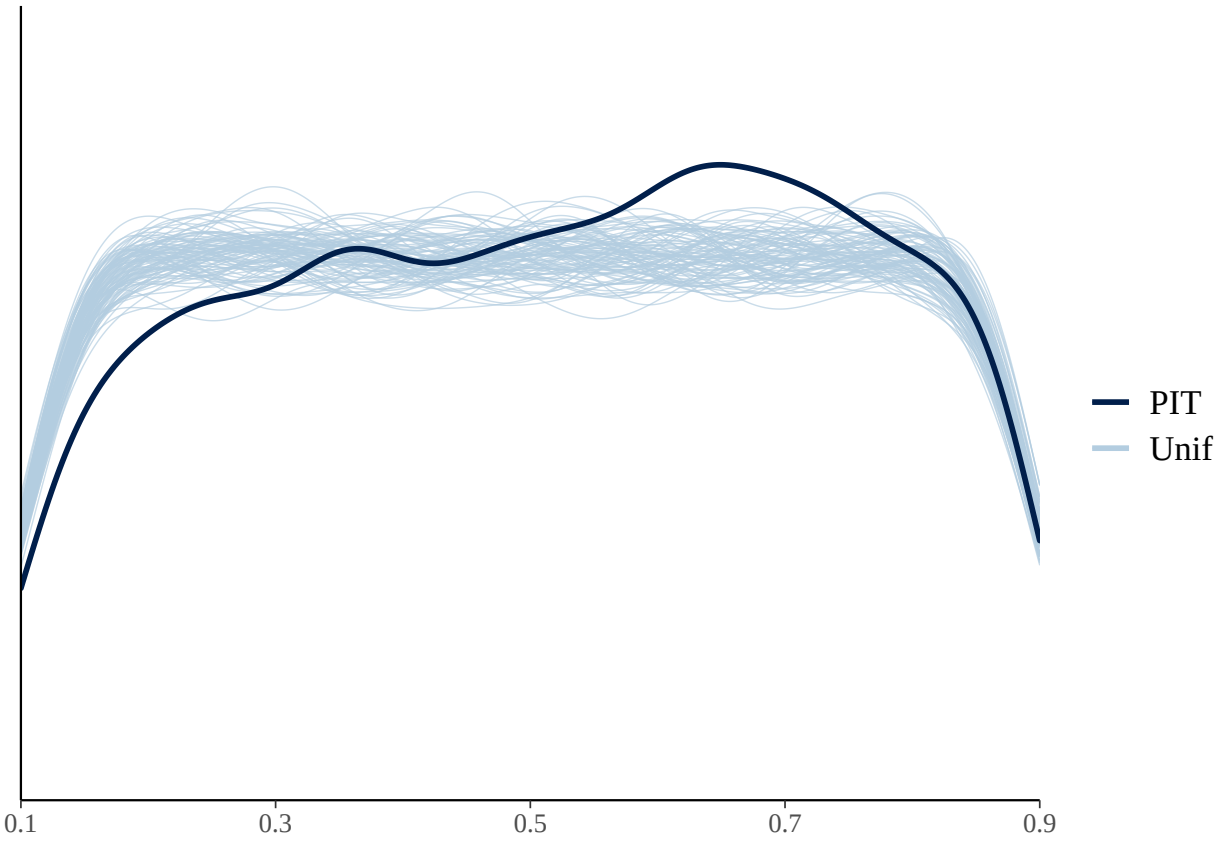
```
## elpd_diff        se
##     175.4      36.2
```

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))
```




```
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