Week 6: Visualizing the Bayesian Workflow

19/02/23

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

Read it in, along with all our packages.

```
library(tidyverse)
library(here)
# for bayes stuff
library(rstan)
library(bayesplot)
library(loo)
library(tidybayes)

ds <- readRDS("D:\\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
```

```
ds
```

```
# A tibble: 3,864 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
 7    25         1     1  26.9 F           39  3.06 Y
 8    30         1     6  32.5 M           39  3.34 Y
 9    32         1     6  38.7 F           39  4.11 Y
10    30         1     7  25.6 M           40  3.91 Y
# ... with 3,854 more rows
```

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.

```
ds1 <- ds %>%
  rename(birthweight = dbwt, gest = combgest) %>%
  mutate(preterm = ifelse(gest<32, "Y", "N")) %>%
  filter(ilive=="Y",gest< 99, birthweight<9.999)
```

```
class(ds1$preterm)
```

```
[1] "character"
```

Question 1

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.

```
library(skimr)
library(janitor)
library(ggplot2)
skim(ds1)
```

Table 1: Data summary

Name	ds1
Number of rows	3842
Number of columns	9
_____	
Column type frequency:	
character	3
numeric	6
_____	
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
sex	0	1	1	1	0	2	0
ilive	0	1	1	1	0	1	0
preterm	0	1	1	1	0	2	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
mager	0	1	28.93	5.84	14.00	25.00	29.0	33.00	50.00	
mracehisp	0	1	2.88	2.51	1.00	1.00	1.0	6.00	8.00	
meduc	0	1	4.41	1.81	1.00	3.00	4.0	6.00	9.00	
bmi	0	1	29.13	13.53	13.60	22.30	25.8	31.30	99.90	
gest	0	1	38.59	2.40	21.00	38.00	39.0	40.00	47.00	
birthweight	0	1	3.26	0.58	0.37	2.95	3.3	3.63	5.05	

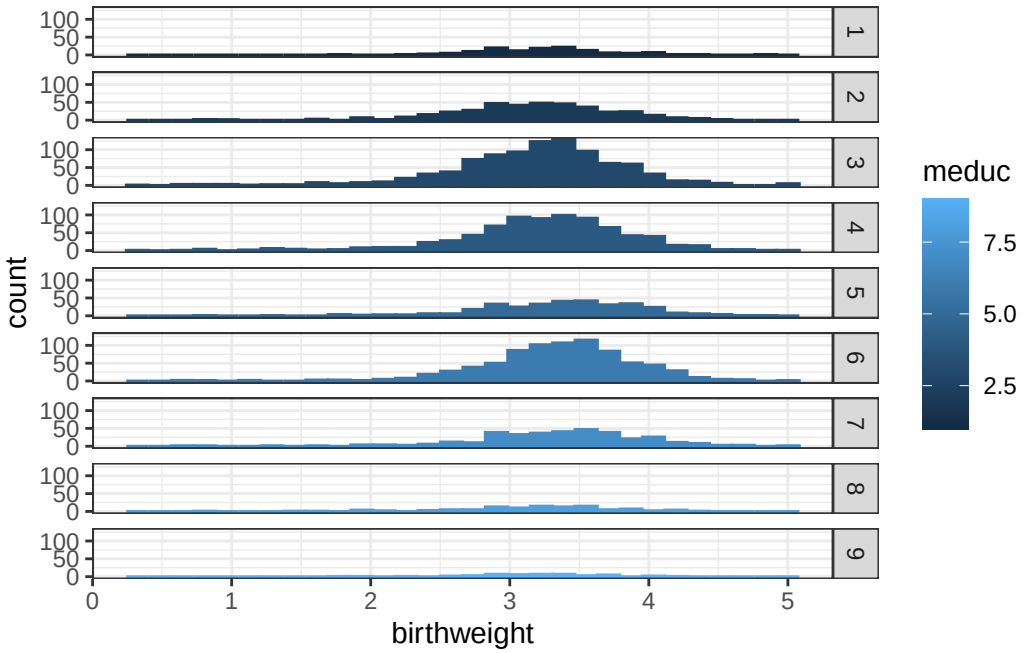
```
ds1 |>get_dupes()
```

```
# A tibble: 0 x 10
# ... with 10 variables: mager <dbl>, mracehisp <dbl>, meduc <dbl>, bmi <dbl>,
#   sex <chr>, gest <dbl>, birthweight <dbl>, ilive <chr>, preterm <chr>,
#   dupe_count <int>
```

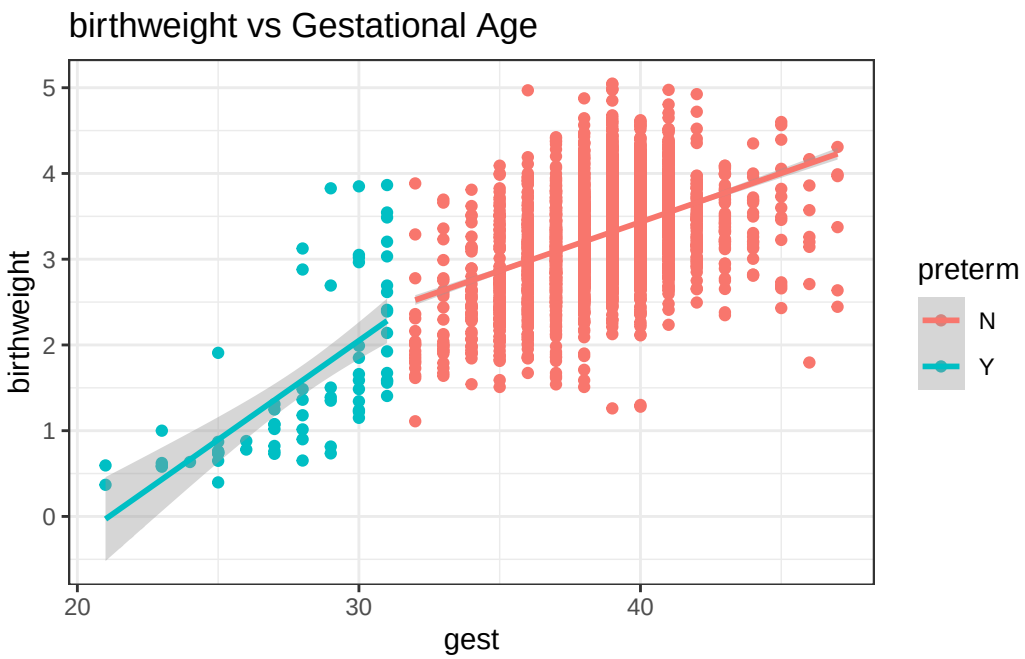
```
ds1<-ds1 |> distinct()
summary(ds1)
```

```
     mager         mracehisp         meduc            bmi       
 Min.   :14.00   Min.   :1.000   Min.   :1.000   Min.   :13.60  
 1st Qu.:25.00   1st Qu.:1.000   1st Qu.:3.000   1st Qu.:22.30  
 Median :29.00   Median :1.000   Median :4.000   Median :25.80  
 Mean   :28.93   Mean   :2.876   Mean   :4.406   Mean   :29.13  
 3rd Qu.:33.00   3rd Qu.:6.000   3rd Qu.:6.000   3rd Qu.:31.30  
 Max.   :50.00   Max.   :8.000   Max.   :9.000   Max.   :99.90  
     sex                 gest        birthweight       ilive          
 Length:3842        Min.   :21.00   Min.   :0.369   Length:3842       
 Class :character   1st Qu.:38.00   1st Qu.:2.948   Class :character  
 Mode  :character   Median :39.00   Median :3.300   Mode  :character  
                    Mean   :38.59   Mean   :3.265                     
                    3rd Qu.:40.00   3rd Qu.:3.629                     
                    Max.   :47.00   Max.   :5.048                     
   preterm         
 Length:3842        
 Class :character   
 Mode  :character   
```

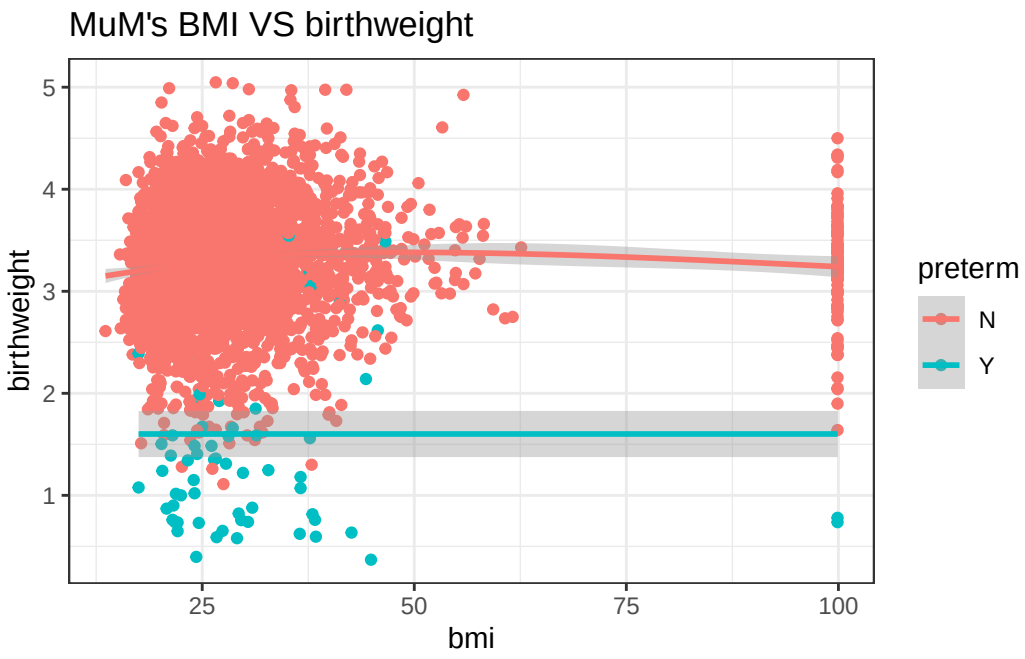
```
ds1|> ggplot(aes(x=birthweight,fill=meduc, color=meduc)) +geom_histogram( position="identi
```




```
ds1 |> ggplot(aes(x = gest, y = birthweight, color = preterm, group = preterm))+geom_point
```




```
ds1 |> ggplot(aes(x=bmi,y=birthweight,color = preterm))+geom_point()+theme_bw()+geom_smoot
```



From above 3 plots,we could see that the higher education mum did not have more weight babies,their distributions of babies' weights were almost similar.

The babies' weights hold a positive relationship with gestational age/time,especially before and after the preterm the both slopes should be different. The cutoff(30-32 weeks) for preterm is a bit arbitrary So there is a big uncertainty in the preterm slope line.

The mums' BMI(healthy state) did have no impact on their babies' weights.The BMI between 50-100 had an even distribution of birth weights.But the data was clearly non-reliable because the females with 100 BMI should not exist.

I thought that The gest VS. birthweight was a graph type that's appropriate to the data type.

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)

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. Use these values to simulate (log) birth weights from the likelihood specified in Model 1, based on the set of observed gestational weights. **Remember the gestational weights should be centered and standardized**.

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

```
ds1$log_weight <- log(ds1$birthweight)
ds1$log_gest_c <- (log(ds1$gest) - mean(log(ds1$gest)))/sd(log(ds1$gest))
ds1 <- ds1 |> mutate(preterm=ifelse(gest<32, 1, 0))
class(ds1$preterm)
```

```
[1] "numeric"
```

```
set.seed(1000)
n_sims <- 1000
sigma <- abs(rnorm(n_sims,0,1))
beta0 <- rnorm(n_sims,0,1)
beta1 <- rnorm(n_sims,0,1)

dsims <- tibble(log_gest_c = ds1$log_gest_c)

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

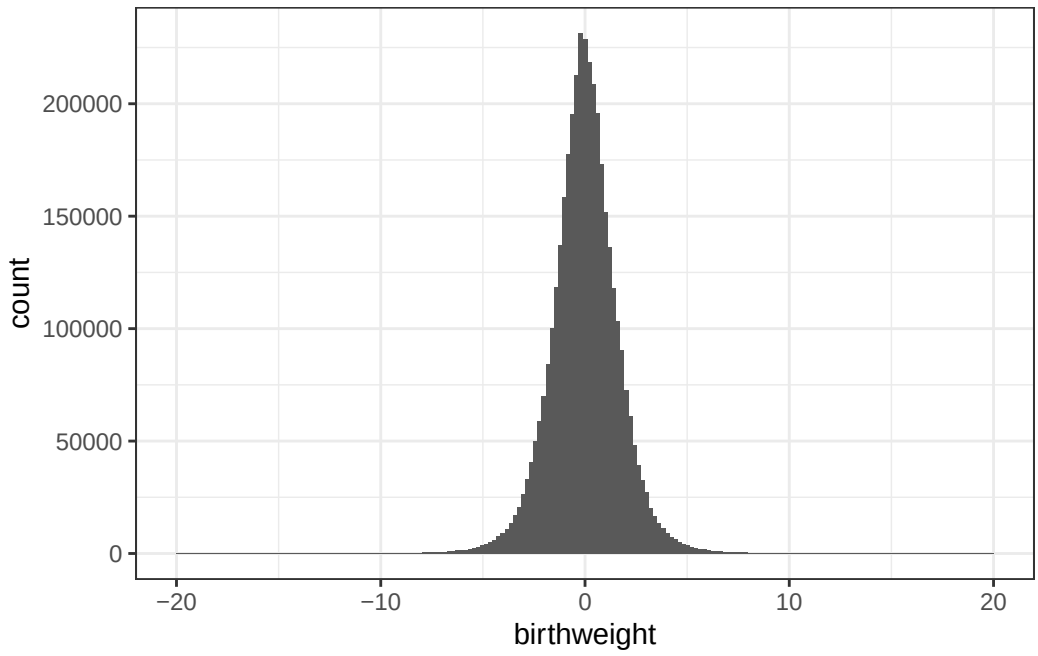
```
tibble [3,842 x 1,001] (S3: tbl_df/tbl/data.frame)
 $ log_gest_c: num [1:3842] 0.188 0.565 0.932 -1.002 -1.852 ...
 $ 1         : num [1:3842] 2.193 1.981 3.526 0.932 -0.327 ...
 $ 2         : num [1:3842] -1.826 -2.302 0.983 0.319 0.456 ...
 $ 3         : num [1:3842] 0.6362 0.0587 -0.4228 2.3099 3.474 ...
```

```
$ 4         : num [1:3842] 0.8898 1.3101 1.6278 0.0939 -2.1953 ...
$ 5         : num [1:3842] -1.5458 -1.0824 0.0816 -2.2777 -1.0697 ...
$ 6         : num [1:3842] 0.0843 -0.377 0.0984 0.1054 -0.0247 ...
$ 7         : num [1:3842] -0.115 -0.259 -1.272 2.307 2.785 ...
$ 8         : num [1:3842] 2.612 1.189 1.198 0.3 0.928 ...
$ 9         : num [1:3842] -0.1918 -0.289 -0.36 0.0646 0.2401 ...
$ 10        : num [1:3842] -1.64 -1.35 -1.9 1.96 5.16 ...
$ 11        : num [1:3842] 2.697 2.189 2.928 1.458 0.863 ...
$ 12        : num [1:3842] 0.7592 0.0584 -0.364 -0.372 0.4084 ...
$ 13        : num [1:3842] -0.4396 -0.0228 0.352 -1.7879 -2.5549 ...
$ 14        : num [1:3842] -1.921 -2.002 -2.525 -0.887 -0.411 ...
$ 15        : num [1:3842] -2.051 -0.208 -0.422 -0.472 1.217 ...
$ 16        : num [1:3842] 0.171 0.201 -0.142 0.894 0.959 ...
$ 17        : num [1:3842] 1.51 1.33 1.28 2.05 2.69 ...
$ 18        : num [1:3842] -0.761 -0.601 -0.451 -1.092 -1.332 ...
$ 19        : num [1:3842] -2.16 0.693 1.036 -3.085 -4.353 ...
$ 20        : num [1:3842] 1.0031 0.8097 1.1008 0.0771 0.0506 ...
$ 21        : num [1:3842] 0.671 2.684 0.67 3.388 9.576 ...
$ 22        : num [1:3842] 0.0282 -1.8972 0.7628 0.6317 -1.378 ...
$ 23        : num [1:3842] 2.45 1.6 1.81 1.62 2.98 ...
$ 24        : num [1:3842] 0.651 0.676 1.378 -2.237 -2.257 ...
$ 25        : num [1:3842] -0.4008 -0.1452 -0.1702 0.0759 -0.3646 ...
$ 26        : num [1:3842] -1.037 -1.719 -0.376 -1.913 -0.664 ...
$ 27        : num [1:3842] 4.13 1.64 -0.14 -3.09 -6.17 ...
$ 28        : num [1:3842] -0.536 0.119 1.262 -2.257 -4.092 ...
$ 29        : num [1:3842] -1.79 -1.17 -1.53 0.39 -0.25 ...
$ 30        : num [1:3842] 1.846 -0.312 -0.816 -2.551 -1.335 ...
$ 31        : num [1:3842] -1.181 -0.892 -1.21 -0.251 0.254 ...
$ 32        : num [1:3842] -1.7912 -2.1469 -0.7516 0.0344 0.1302 ...
$ 33        : num [1:3842] 1.32 2.23 2.33 1.89 1.24 ...
$ 34        : num [1:3842] -0.53 0.453 0.301 -1.634 -3.121 ...
$ 35        : num [1:3842] -3.407 0.115 -0.235 -2.612 0.152 ...
$ 36        : num [1:3842] 0.448 0.449 0.386 1.122 1.554 ...
$ 37        : num [1:3842] -0.533 -0.291 -2.058 1.628 3.665 ...
$ 38        : num [1:3842] -0.229 -1.552 -1.367 0.23 1.505 ...
$ 39        : num [1:3842] -0.421 1.084 0.74 -1.052 -2.159 ...
$ 40        : num [1:3842] -2.9 -2.09 1.98 -3.22 -1.04 ...
$ 41        : num [1:3842] 0.072 1.547 0.281 1.783 2.282 ...
$ 42        : num [1:3842] 1.065 1.179 0.631 1.475 1.257 ...
$ 43        : num [1:3842] -0.981 -0.614 -0.257 -2.343 -3.099 ...
$ 44        : num [1:3842] -0.0727 -0.2635 -0.3381 0.2347 0.6005 ...
$ 45        : num [1:3842] -0.2943 -0.2153 -0.3989 -0.0117 -0.0614 ...
$ 46        : num [1:3842] 0.1639 4.9587 0.0275 -2.8269 0.6974 ...
```

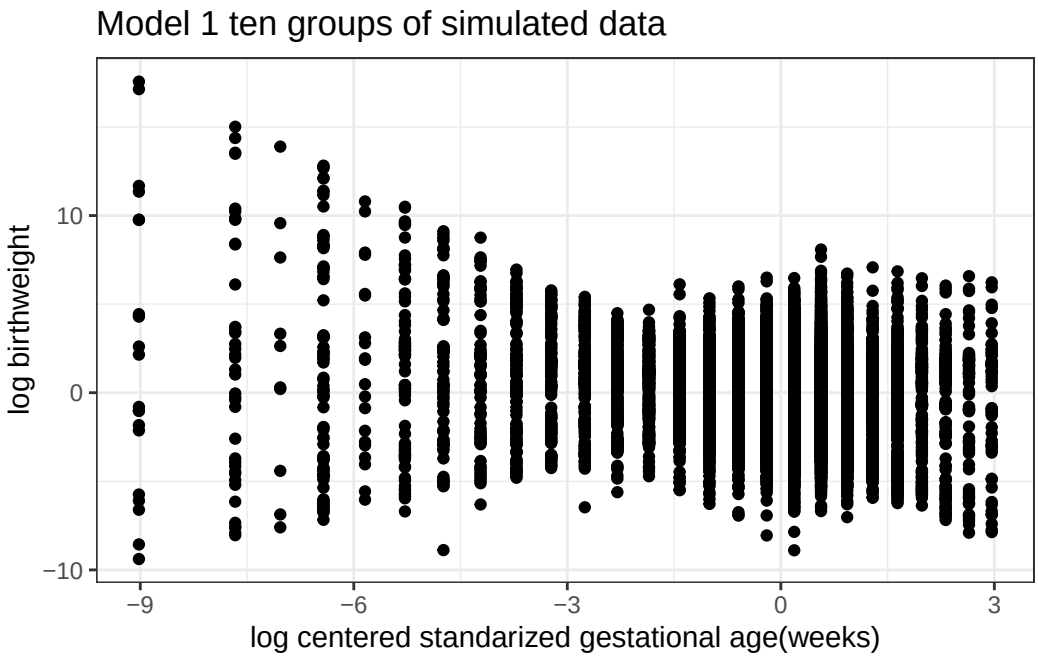
```
$ 47        : num [1:3842] 0.092 0.835 1.227 -1.568 -2.894 ...
$ 48        : num [1:3842] -0.564 -0.65 -0.617 -0.293 -0.289 ...
$ 49        : num [1:3842] -0.381 -0.524 -1.643 -0.495 -4.051 ...
$ 50        : num [1:3842] 2.9891 2.3682 0.8706 1.2867 -0.0493 ...
$ 51        : num [1:3842] -2.2295 -2.6527 -2.3944 0.0977 1.4098 ...
$ 52        : num [1:3842] -0.157 1.345 3.151 0.138 -1.499 ...
$ 53        : num [1:3842] -0.6608 -0.0916 0.1708 -0.8238 -0.7515 ...
$ 54        : num [1:3842] 0.0877 2.5264 -0.844 2.7038 3.0597 ...
$ 55        : num [1:3842] 1.667 1.168 0.913 3.533 4.584 ...
$ 56        : num [1:3842] 0.111 -0.133 0.577 -1.277 -0.594 ...
$ 57        : num [1:3842] -0.139 0.946 1.428 -3.451 -2.755 ...
$ 58        : num [1:3842] 1.861 1.531 2.014 1.277 0.174 ...
$ 59        : num [1:3842] -0.0141 2.7792 3.2531 2.2147 -0.6856 ...
$ 60        : num [1:3842] 0.268 -0.706 -1.648 0.55 2.524 ...
$ 61        : num [1:3842] 0.029 0.778 0.852 -2.217 -3.575 ...
$ 62        : num [1:3842] -1.47 -1.61 -1.91 -1.63 -1.2 ...
$ 63        : num [1:3842] 2.287 2.13 0.383 0.476 4.186 ...
$ 64        : num [1:3842] -1.395 -0.892 -1.517 -2.332 0.178 ...
$ 65        : num [1:3842] 1.2561 0.1488 -0.2407 -0.0622 -0.0216 ...
$ 66        : num [1:3842] -0.291 0.0166 0.5904 -0.9982 -0.8935 ...
$ 67        : num [1:3842] -2.992 -1.59 -3.194 0.906 3.236 ...
$ 68        : num [1:3842] -1.89 -3.568 -1.053 0.355 1.566 ...
$ 69        : num [1:3842] -2.345 -0.878 -0.545 -1.586 -1.883 ...
$ 70        : num [1:3842] 0.317 -0.161 -0.577 1.755 2.785 ...
$ 71        : num [1:3842] -1.763 -0.976 -0.996 3.052 1.399 ...
$ 72        : num [1:3842] -0.461 -0.381 0.522 1.265 0.538 ...
$ 73        : num [1:3842] 1.77 2.26 1.8 2.14 1.62 ...
$ 74        : num [1:3842] 0.868 -1.999 0.372 1.304 0.287 ...
$ 75        : num [1:3842] -1.026 -1.51 -1.917 0.113 1.348 ...
$ 76        : num [1:3842] 0.479 0.232 0.572 0.187 -0.169 ...
$ 77        : num [1:3842] 0.549 0.685 0.526 -0.737 -0.918 ...
$ 78        : num [1:3842] -2.5784 -2.8152 -1.6099 -0.0688 0.0897 ...
$ 79        : num [1:3842] -2.423 0.0722 -0.5583 -1.3946 0.9029 ...
$ 80        : num [1:3842] 0.475 1.401 2.028 -0.921 -2.454 ...
$ 81        : num [1:3842] -0.0781 1.0247 1.2843 1.625 -0.2001 ...
$ 82        : num [1:3842] 0.166 -0.486 -0.689 1.052 1.479 ...
$ 83        : num [1:3842] 0.5874 0.201 0.2509 -0.464 0.0462 ...
$ 84        : num [1:3842] 2.542 1.404 1.947 0.267 -0.911 ...
$ 85        : num [1:3842] -1.783 -0.288 -1.504 -2.389 -2.662 ...
$ 86        : num [1:3842] 1.17 0.877 0.392 2.19 3.193 ...
$ 87        : num [1:3842] 0.4234 0.4595 0.0361 0.8869 1.1438 ...
$ 88        : num [1:3842] 2.555 1.98 -0.234 1.852 1.461 ...
$ 89        : num [1:3842] 0.387 3.051 1.474 -0.665 0.614 ...
```

```
$ 90        : num [1:3842] 2.4805 -3.0944 2.0956 -0.0487 -3.794 ...
$ 91        : num [1:3842] 3.1829 1.1281 3.5916 0.0249 0.8487 ...
$ 92        : num [1:3842] 1.5964 0.0691 -2.5906 -6.0327 -2.9555 ...
$ 93        : num [1:3842] -0.754 -1.61 -2.081 -1.066 -0.678 ...
$ 94        : num [1:3842] -0.566 -1.182 -2.019 2.196 4.133 ...
$ 95        : num [1:3842] -1.51 -0.943 -0.109 -3.476 -4.885 ...
$ 96        : num [1:3842] 0.147 0.256 1.705 -1.237 -3.552 ...
$ 97        : num [1:3842] -3.27 -2.28 -2.53 -3.25 -3.44 ...
$ 98        : num [1:3842] -1.28 -1.5 -3.76 -2.68 -2.57 ...
 [list output truncated]
```

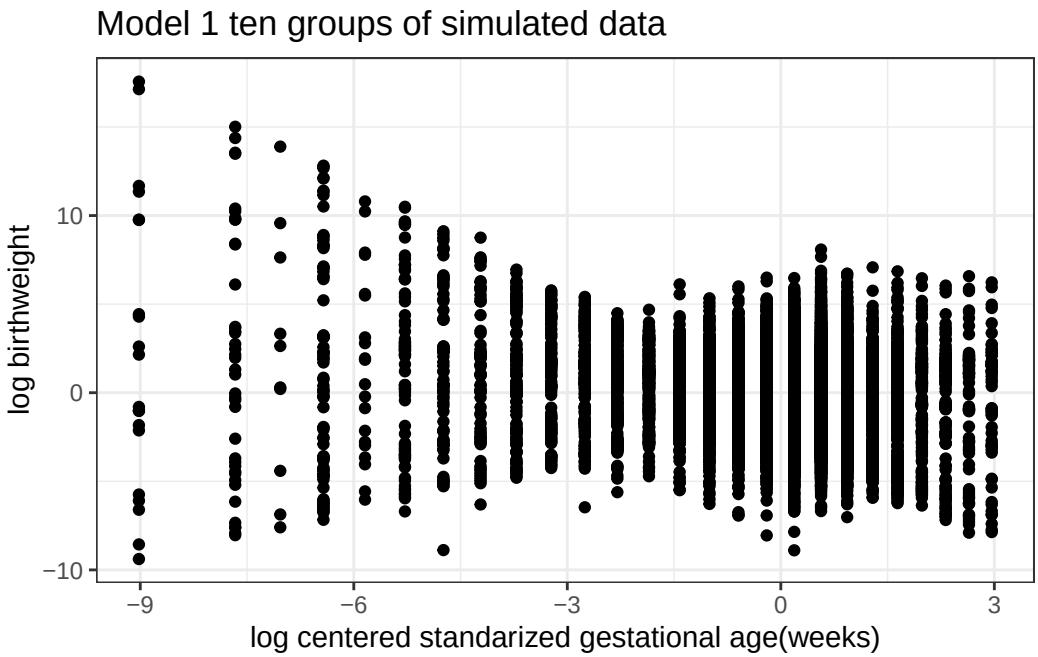
```
sims_plot <- dsims |> select(-log_gest_c) |> pivot_longer(cols = everything())|>
  ggplot(aes(x = value)) + geom_histogram(bins = 200) + theme_bw() + xlab("birthweight")+x
sims_plot
```




```
samp10 <- sample(ncol(dsims|>select(-log_gest_c)), 10)
a<-as.matrix(dsims[,samp10])
df<-as.data.frame(cbind(a,dsims$log_gest_c))

df|> ggplot()+ geom_point(aes(x=df[,11],y=df[,1]))+ geom_point(aes(x=df[,11],y=df[,2]))+ge
```




```
ddf<-df|> select(-V11)|>pivot_longer(cols = everything())
ddf<-cbind(names=rep(df$V11, each=10),ddf)
ddf|>
ggplot(aes(x=names,y=value))+geom_point()+theme_bw()+xlab("log centered standarized gestat
```




```
#samp10_plot<-ggplot()+geom_point(data=df,aes(x=df[,11],y=df[,1]))

#for(i in 2:10){samp10_plot<- samp10_plot+geom_point(data=df,aes(x=df[,11],y=df[,i]))}

#samp10_plot+theme_bw() +xlab("log centered standarized gestational age(weeks)")+ ylab(" l
```

The distribution of ten groups of simulated data was so weird, its model seemed to be not so fittable.

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.

```
ds1$log_weight <- log(ds1$birthweight)
ds1$log_gest_c <- (log(ds1$gest) - mean(log(ds1$gest)))/sd(log(ds1$gest))

# put into a list
stan_data <- list(N = nrow(ds1),
                  log_weight = ds1$log_weight,
```

```
                    log_gest = ds1$log_gest_c)
str(ds1)
```

```
tibble [3,842 x 11] (S3: tbl_df/tbl/data.frame)
 $ mager      : num [1:3842] 16 25 27 26 28 31 25 30 32 30 ...
 $ mracehisp  : num [1:3842] 2 7 2 1 7 7 1 1 1 1 ...
 $ meduc      : num [1:3842] 2 2 3 3 2 3 1 6 6 7 ...
 $ bmi        : num [1:3842] 23 43.6 19.5 21.5 40.6 29.3 26.9 32.5 38.7 25.6 ...
 $ sex        : chr [1:3842] "M" "M" "F" "F" ...
 $ gest       : num [1:3842] 39 40 41 36 34 35 39 39 39 40 ...
 $ birthweight: num [1:3842] 3.18 4.14 3.17 3.4 2.71 ...
 $ ilive      : chr [1:3842] "Y" "Y" "Y" "Y" ...
 $ preterm    : num [1:3842] 0 0 0 0 0 0 0 0 0 0 ...
 $ log_weight : num [1:3842] 1.157 1.42 1.155 1.224 0.997 ...
 $ log_gest_c : num [1:3842] 0.188 0.565 0.932 -1.002 -1.852 ...
```

Now fit the model

```
mod1 <- rstan::stan(data = stan_data,
             file = here::here("D:\\simple_weight.stan"),
             iter = 500,
             seed = 243)
```

```
SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 0.000628 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 6.28 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: 0.6 seconds (Warm-up)
Chain 1:                0.497 seconds (Sampling)
Chain 1:                1.097 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
Chain 2:
Chain 2: Gradient evaluation took 0.000191 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 1.91 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: 0.576 seconds (Warm-up)
Chain 2:                0.533 seconds (Sampling)
Chain 2:                1.109 seconds (Total)
Chain 2:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
Chain 3:
Chain 3: Gradient evaluation took 0.000188 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 1.88 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: 0.58 seconds (Warm-up)
Chain 3:                0.52 seconds (Sampling)
Chain 3:                1.1 seconds (Total)
Chain 3:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 0.000385 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 3.85 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: 0.584 seconds (Warm-up)
Chain 4:                0.476 seconds (Sampling)
Chain 4:                1.06 seconds (Total)
Chain 4:
```

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

```
             mean      se_mean          sd      2.5%       25%       50%
beta[1] 1.1624783 8.160385e-05 0.002856578 1.1570200 1.1604786 1.1625011
beta[2] 0.1437529 8.295075e-05 0.002912236 0.1381284 0.1416970 0.1436747
sigma   0.1690330 1.113724e-04 0.001902828 0.1652694 0.1677842 0.1690763
              75%     97.5%     n_eff      Rhat
beta[1] 1.1644669 1.1681028 1225.3801 0.9978044
beta[2] 0.1456716 0.1495180 1232.5721 0.9998714
sigma   0.1702528 0.1727953  291.9066 1.0146111
```

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.

```
mean(log(ds1$gest))
```

```
[1] 3.650894
```

```
exp(mean(log(ds1$gest)))
```

```
[1] 38.50906
```

```
sd(log(ds1$gest))
```

```
[1] 0.06723322
```

log_weight= beta[1] + beta[2] * log_gest_c =1.1624783+0.1437529*(log(37)-3.650894)/0.06723322=1.077005

the log birth weight range within(1.077005-0.1690330,1.077005+0.1690330)=(0.907972,1.246038)

An estimate of the expected birthweight of a baby who was born at a gestational age of 37 weeks was 2.935873 kg and its range (95%CI) within

(2.479289kg,3.476542kg).

The next part is my coding to compute the expected birthweight of a baby:

```
library(tidyverse)
library(rstan)
library(bayesplot)
```

```
library(tidybayes)

mod1|> spread_draws(beta[k])|>
  mutate(expect_weight_EST=exp(beta[1]+beta[2]*(log(37)-mean(log(ds1$gest)))/sd(log(ds1$ge
  median_qi(expect_weight_EST)
```

```
# A tibble: 2 x 7
      k expect_weight_EST .lower .upper .width .point .interval
  <int>             <dbl>  <dbl>  <dbl>  <dbl> <chr>  <chr>
1     1              1.61   1.61   1.61   0.95 median qi
2     2              1.06   1.06   1.06   0.95 median qi
```

```
cat("an estimate of the expected birthweight of a baby who was born \
    at a gestational age of 37 weeks =\n ",round(exp(1.060765),3),"kg", "\n")
```

```
an estimate of the expected birthweight of a baby who was born
    at a gestational age of 37 weeks =
  2.889 kg
```

An estimate of the expected birthweight of a baby who was born at a gestational age of 37 weeks = 2.889 kg.

Question 4

Write a stan model to run Model 2, and run it.

```
# put into a list
stan_data1 <- list(N = nrow(ds1),
                   log_weight = ds1$log_weight,
                   log_gest = ds1$log_gest_c,
                   preterm=ds1$preterm)
mod3 <- rstan::stan(data = stan_data1,
              file = here::here("D:\\simple_weight1.stan"),
              iter = 500,
              seed=243)
```

```
SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
```

```
Chain 1:
Chain 1: Gradient evaluation took 0.001147 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 11.47 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: 2.362 seconds (Warm-up)
Chain 1:                2.345 seconds (Sampling)
Chain 1:                4.707 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
Chain 2:
Chain 2: Gradient evaluation took 0.000666 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 6.66 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: 2.581 seconds (Warm-up)
Chain 2:                2.131 seconds (Sampling)
Chain 2:                4.712 seconds (Total)
Chain 2:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
Chain 3:
Chain 3: Gradient evaluation took 0.000472 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 4.72 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: 3.011 seconds (Warm-up)
Chain 3:                2.011 seconds (Sampling)
Chain 3:                5.022 seconds (Total)
Chain 3:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 0.000473 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 4.73 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: 2.332 seconds (Warm-up)
Chain 4:                2.025 seconds (Sampling)
Chain 4:                4.357 seconds (Total)
Chain 4:
```

```
summary(mod3)$summary[c("beta[1]", "beta[2]", "beta[3]", "beta[4]","sigma"),]
```

```
             mean      se_mean          sd       2.5%        25%       50%
beta[1] 1.1696329 8.021297e-05 0.002705139 1.16410383 1.16791913 1.1695478
beta[2] 0.1018545 1.111662e-04 0.003424916 0.09508969 0.09961365 0.1019319
beta[3] 0.5620695 3.406560e-03 0.062560942 0.43112646 0.52265217 0.5614275
beta[4] 0.1982641 6.964438e-04 0.012807594 0.17144797 0.18979854 0.1986269
sigma   0.1611971 8.785429e-05 0.001825790 0.15774991 0.15994557 0.1611909
              75%     97.5%     n_eff     Rhat
beta[1] 1.1714725 1.1748162 1137.3388 1.000638
beta[2] 0.1040358 0.1087724  949.1923 1.002232
beta[3] 0.6039584 0.6839901  337.2675 1.015352
beta[4] 0.2062635 0.2232005  338.1917 1.013325
sigma   0.1623667 0.1649513  431.8927 1.004553
```

Question 5

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

```
load(here::here("D:\\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
```

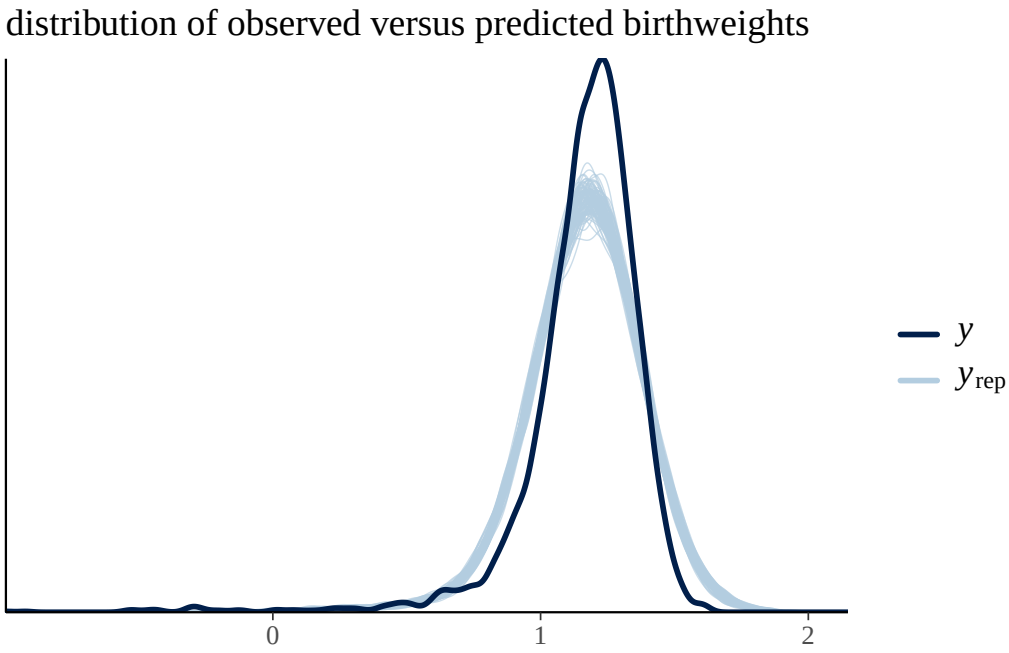
```
sigma   0.1610727 9.950037e-05 0.001782004 0.15784213 0.15978020 0.1610734
              75%     97.5%     n_eff      Rhat
beta[1] 1.1716235 1.1750167 391.67359 1.0115970
beta[2] 0.5990427 0.6554967  98.98279 1.0088166
beta[3] 0.1044230 0.1093843 613.22428 0.9978156
beta[4] 0.2064079 0.2182454 121.59685 1.0056875
sigma   0.1623019 0.1646189 320.75100 1.0104805
```

Both results are similar, just beta[2]and beta[3] exchange order because prof. arranged the formula coefficients order which was different from mine.

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(100)
y <- ds1$log_weight
yrep1 <- extract(mod1)[["log_weight_rep"]]
yrep2 <- extract(mod3)[["log_weight_rep"]]
samp100 <- sample(nrow(yrep1), 100)
ppc_dens_overlay(y, yrep1[samp100, ])  + ggtitle("distribution of observed versus predicte
```

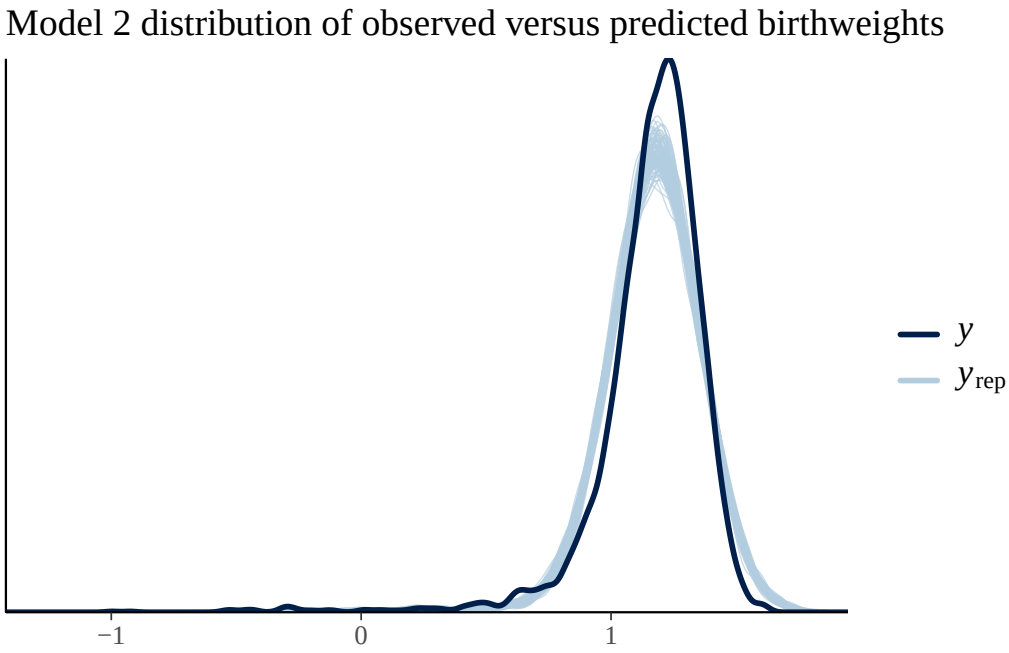


Question 6

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`)

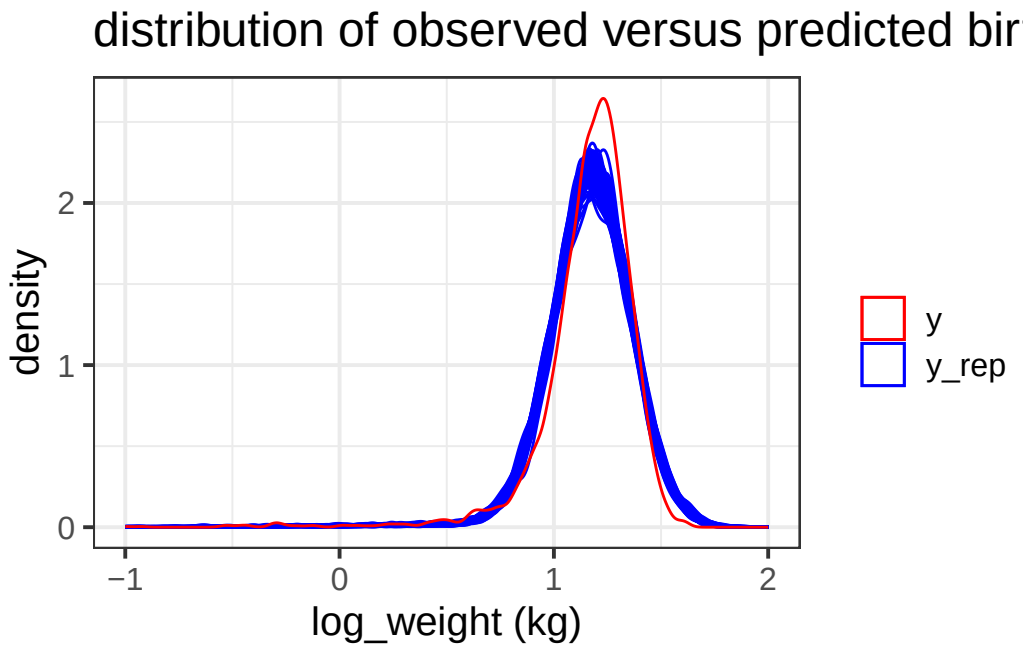
The following plot was amde with build-in function of the bayes plot:

```
set.seed(100)
y <- ds1$log_weight
yrep2 <- extract(mod3)[["log_weight_rep"]]
samp100_1 <- sample(nrow(yrep2), 100)
ppc_dens_overlay(y, yrep2[samp100_1, ])  + ggtitle("Model 2 distribution of observed versu
```



The following figure was plotted by myself just with "geom_density":

```
library(tidyverse)
library(ggplot2)

samp100_2 <- sample(nrow(yrep2), 100)

df1 <- as.tibble(t(yrep2[samp100_2, ]))

df1 <- df1 |> cbind(i = 1:3842, log_weight_obs = ds1$log_weight)|>
pivot_longer(`V1`:`V100`, names_to = "simus", values_to = "log_weight_rep")

set.seed(100)
df1 |>ggplot(aes(log_weight_rep, group = simus)) +
geom_density(alpha = 0.4, aes(color = "y_rep")) +
geom_density(data = ds1 |> mutate(simus = "V1"),alpha = 15,
aes(x = log(birthweight), col = "y")) +
scale_color_manual(name = "",values = c("y" = "red",
"y_rep" = "blue")) +
xlab("log_weight (kg)") + ggtitle("distribution of observed versus predicted birthweights"
theme_bw(base_size = 15)+scale_x_continuous(limits = c(-1, 2))
```



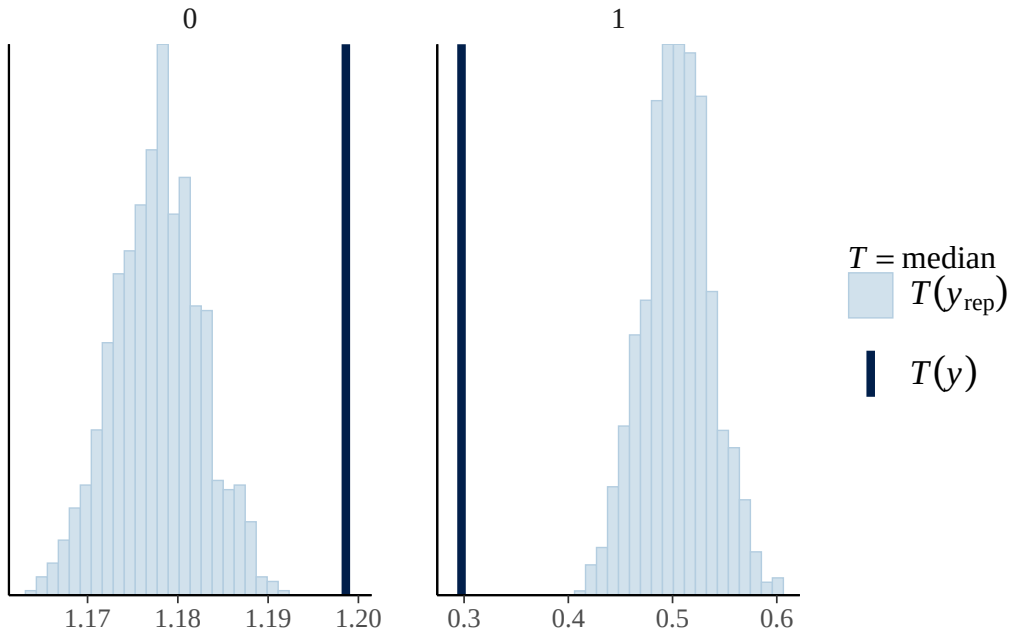

```
#ggplot(ds1,aes(x=log_weight))+ggplot(ds1,aes(x=yrep2[samp100, ])) + geom_density(alpha =
#yrep2[samp100,1 ]
#dfyrep2<-as.data.frame(yrep2[samp100, ]
#ggplot(yrep2[samp100, ],aes(x=yrep2[samp100, ])) + geom_density(alpha = 0.2) + theme_bw()
```

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

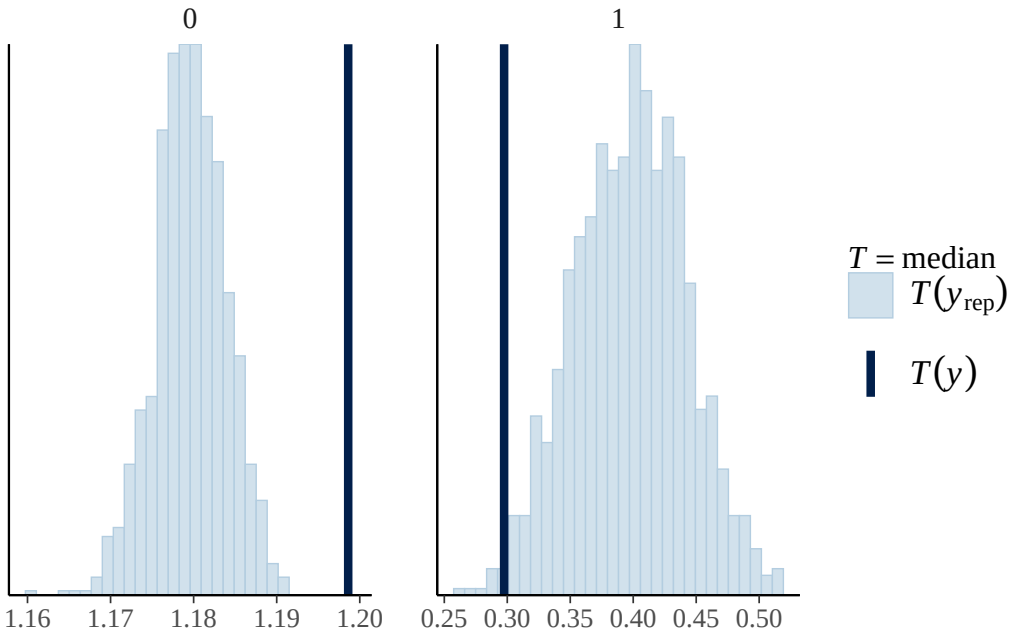
```
ppc_stat_grouped(ds1$log_weight, yrep1, group = ds1$preterm, stat = 'median')
```



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).

```
ppc_stat_grouped(ds1$log_weight, yrep2, group = ds1$preterm, stat = 'median')
```




```
ds2<-ds1|>mutate(less2.5=ifelse(birthweight<2.5,1,0))
ds2|>summarise(propt_less2.5=mean(less2.5))
```

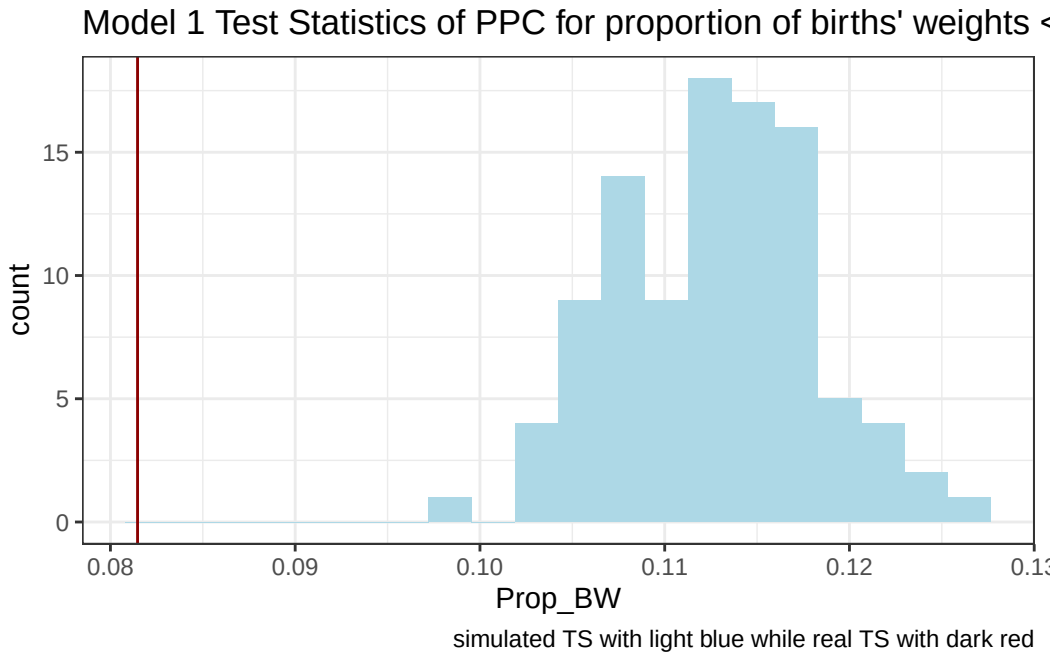
```
# A tibble: 1 x 1
  propt_less2.5
          <dbl>
1        0.0815
```

The proportion of births' weights for both models under 2.5kg is 0.08146799.

Model1

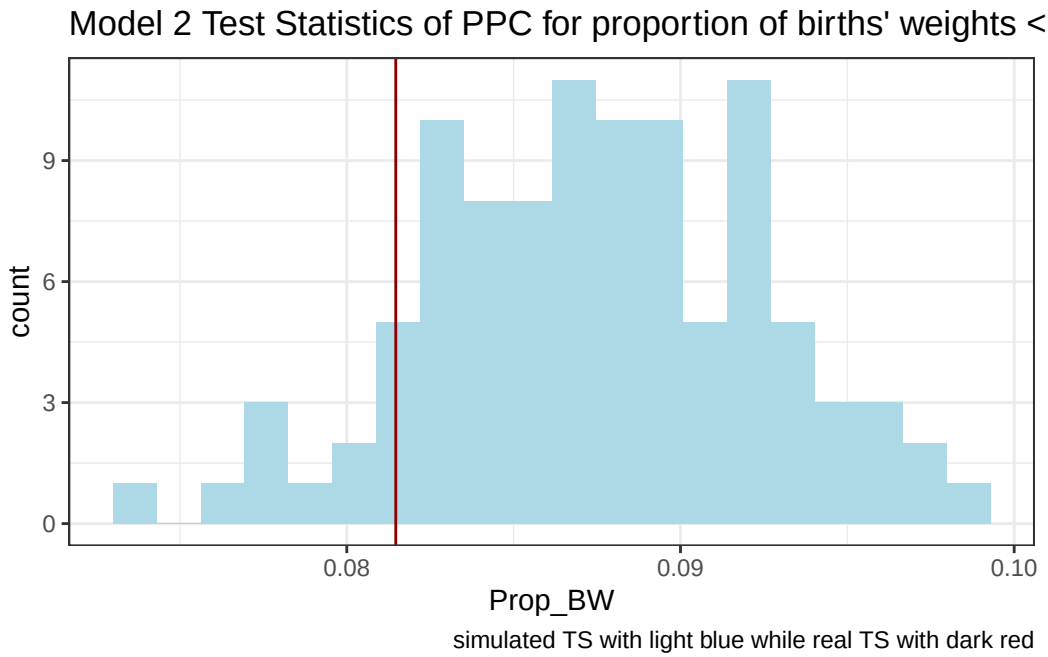
```
y <- ds1$log_weight
yrep1 <- extract(mod1)[["log_weight_rep"]]
df <- as.tibble(t(yrep1[samp100, ]))

df <- df |> cbind(i = 1:3842, log_weight_obs = ds1$log_weight)|>
pivot_longer(`V1`:`V100`, names_to = "simus", values_to = "log_weight_rep")

test_stat_obs <- mean(ds1$birthweight < 2.5)
test_stat_rep <- df |> group_by(simus) |>
summarize(Prop_BW = mean(exp(log_weight_rep) < 2.5))
```

```
test_stat_rep |>
ggplot(aes(x = Prop_BW)) +
geom_histogram(bins = 20, fill = "lightblue") +
geom_vline(xintercept = test_stat_obs, color = "darkred") +theme_bw() +
labs(caption = "simulated TS with light blue while real TS with dark red",
title = "Model 1 Test Statistics of PPC for proportion of births' weights < 2.5kg")
```



Model2

```
test_stat_obs <- mean(ds1$birthweight < 2.5)
test_stat_rep <- df1 |> group_by(simus) |>
summarize(Prop_BW = mean(exp(log_weight_rep) < 2.5))
test_stat_rep |>
ggplot(aes(x = Prop_BW)) +
geom_histogram(bins = 20, fill = "lightblue") +
geom_vline(xintercept = test_stat_obs, color = "darkred") +theme_bw() +
labs(caption = "simulated TS with light blue while real TS with dark red",
title = "Model 2 Test Statistics of PPC for proportion of births' weights < 2.5kg")
```



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(mod3)[["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   1377.0  72.4
p_loo         9.8   1.4
looic     -2754.1 144.8
------
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 1000 by 3842 log-likelihood matrix

         Estimate    SE
elpd_loo   1552.7  69.8
p_loo        15.3   2.3
looic     -3105.4 139.7
------
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.
```

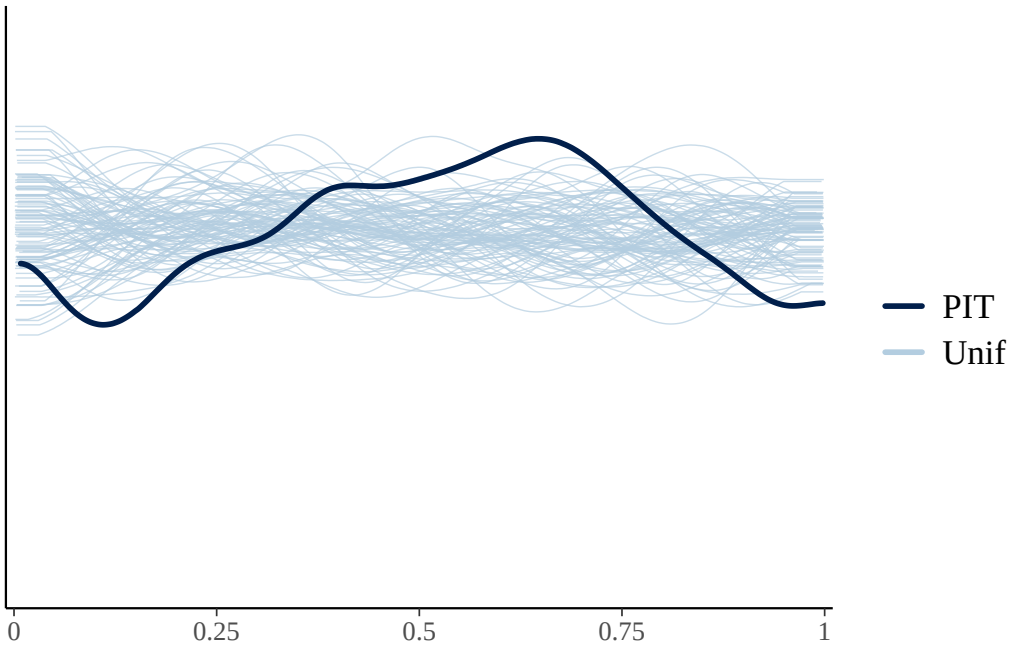
Comparing the two models tells us Model 2 is better:

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

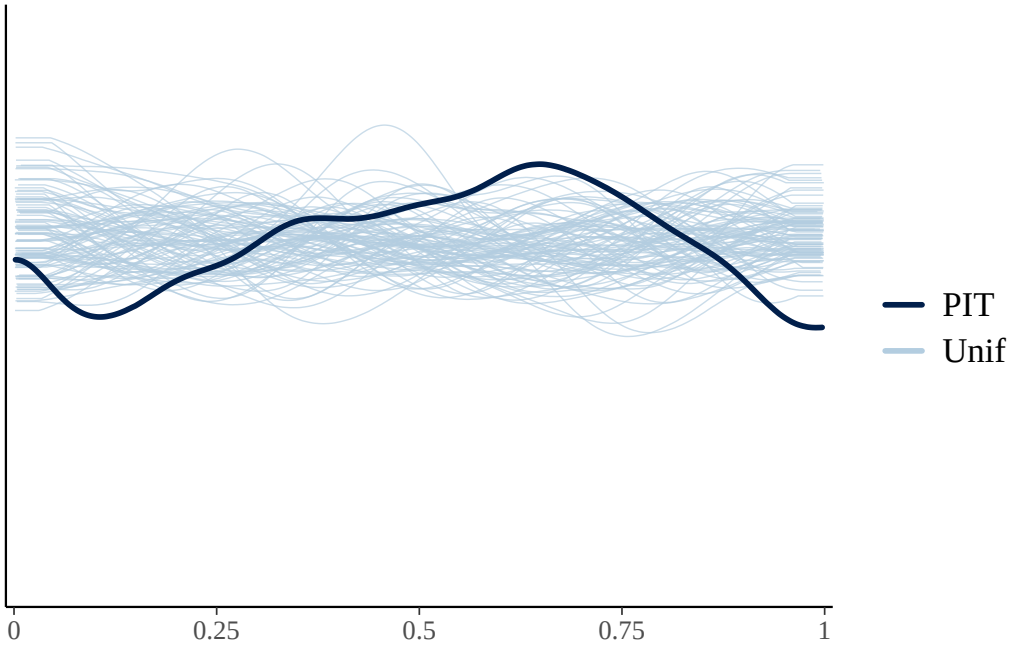
```
       elpd_diff se_diff
model2    0.0       0.0
model1 -175.6      36.3
```

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

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




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



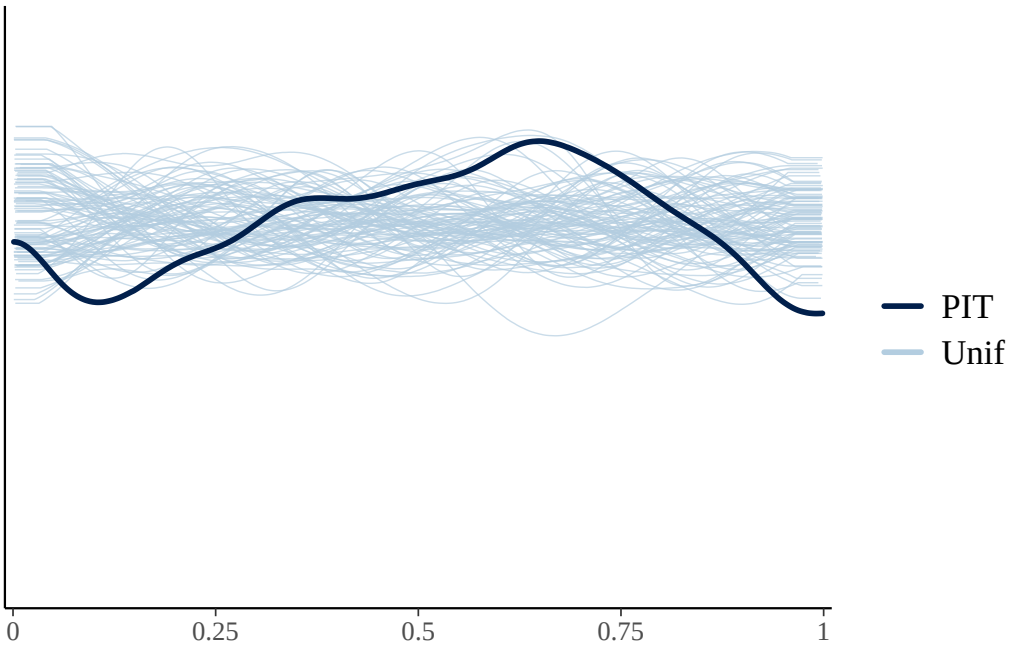
Bonus question (not required)

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

```
set.seed(1000)
y <- ds1$log_weight
yrep3 <- extract(mod3)[["log_weight_rep"]]
loglik3 <- extract(mod3)[["log_lik"]]
loo3 <- loo(loglik3, save_psis = TRUE)
loo3
```

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

         Estimate    SE
elpd_loo   1552.7  69.8
p_loo        15.3   2.3
looic     -3105.4 139.7
------
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.
```

```
ppc_loo_pit_overlay(yrep = yrep3, y = y, lw = weights(loo3$psis_object))
```



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.

I want to add more 2 variables like `mager`- mum's age and `sex`-baby's sex to check if these factors influenced the births' weights.

```
ds2 <- ds1|> mutate(sex=ifelse(sex=="F",0,1))

# put into a list
stan_data2 <- list(N = nrow(ds2),
                   log_weight = ds2$log_weight,
                   log_gest = ds2$log_gest_c,
                   preterm=ds2$preterm,
                   mager=ds2$mager,
                   sex=ds2$sex
                   )
mod4 <- rstan::stan(data = stan_data2,
              file = here::here("D:\\simple_weight2.stan"),
              iter = 500,
```

```
                seed=243)
```

```
SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 0.002231 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 22.31 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: 13.418 seconds (Warm-up)
Chain 1:                35.526 seconds (Sampling)
Chain 1:                48.944 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
Chain 2:
Chain 2: Gradient evaluation took 0.000959 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 9.59 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: 22.218 seconds (Warm-up)
Chain 2:                24.281 seconds (Sampling)
Chain 2:                46.499 seconds (Total)
Chain 2:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
Chain 3:
Chain 3: Gradient evaluation took 0.000937 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 9.37 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: 32.889 seconds (Warm-up)
Chain 3:                22.733 seconds (Sampling)
Chain 3:                55.622 seconds (Total)
Chain 3:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 0.000778 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 7.78 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: 15.491 seconds (Warm-up)
Chain 4:                24.061 seconds (Sampling)
Chain 4:                39.552 seconds (Total)
Chain 4:
```

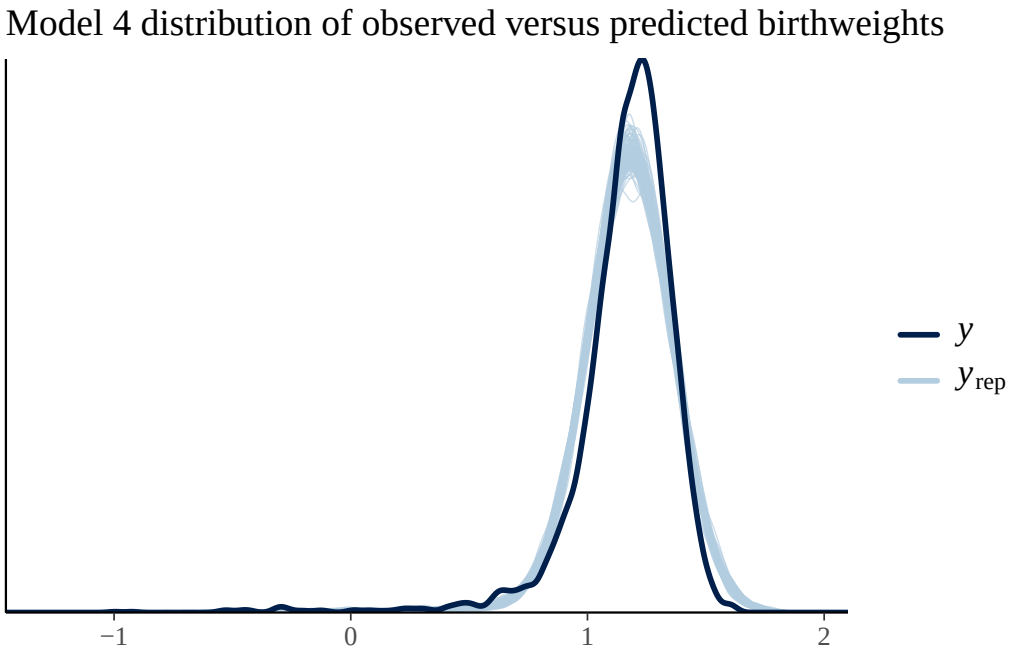
```
summary(mod4)$summary[c("beta[1]", "beta[2]", "beta[3]", "beta[4]","beta[5]","beta[6]","si
```

```
              mean      se_mean           sd        2.5%         25%
beta[1] 1.074780842 4.385909e-04 0.0128954220 1.050373963 1.066144727
beta[2] 0.103429296 1.216776e-04 0.0032232403 0.097289902 0.101224626
beta[3] 0.562082182 5.525572e-03 0.0597840748 0.456584327 0.516529423
beta[4] 0.197177869 1.074114e-03 0.0123031159 0.174410602 0.188425313
beta[5] 0.002544497 1.418151e-05 0.0004324991 0.001710406 0.002238634
beta[6] 0.042022105 2.128076e-04 0.0050501140 0.031906533 0.038486442
sigma   0.159143579 5.095985e-05 0.0017484908 0.155974255 0.157938811
               50%         75%       97.5%     n_eff      Rhat
beta[1] 1.074389655 1.084508713 1.099424358  864.4739 1.0005652
beta[2] 0.103296843 0.105711151 0.109634370  701.7204 1.0003529
beta[3] 0.561152789 0.606269241 0.678971441  117.0622 1.0102749
beta[4] 0.196714646 0.206013430 0.220674462  131.1987 1.0073207
beta[5] 0.002566759 0.002828506 0.003369655  930.0916 1.0002478
beta[6] 0.042285389 0.045608717 0.051050192  563.1552 1.0003400
sigma   0.159090192 0.160359661 0.162711772 1177.2544 0.9989525
```

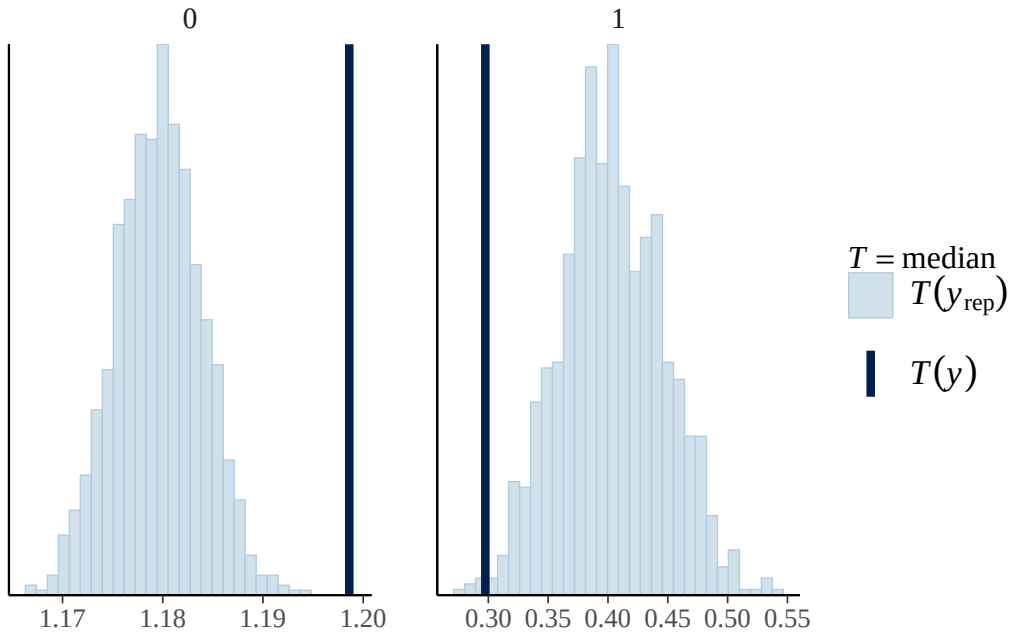
From the summary we could see that beta[5]=0.002544497 and beta[6]=0.042022105, their values were small, so the mums' age and babies' gender had trivial impact on the births' weights.

PPC of both model2 and model4

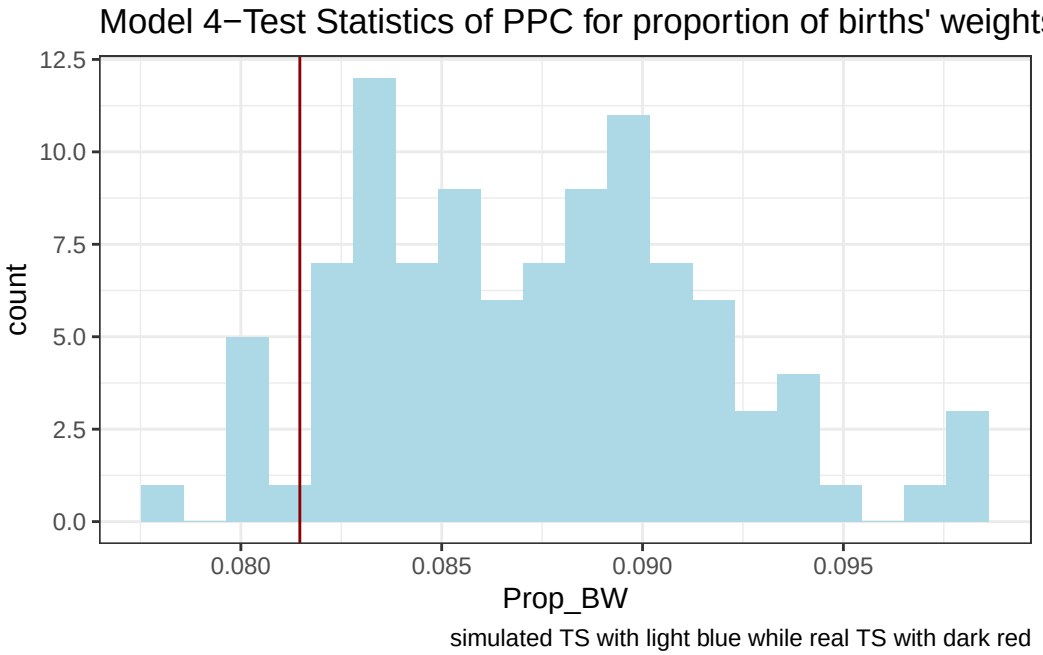
```
set.seed(100)
y4 <- ds2$log_weight
yrep4 <- extract(mod4)[["log_weight_rep"]]
samp100_4 <- sample(nrow(yrep4), 100)
ppc_dens_overlay(y4, yrep4[samp100_4, ])  + ggtitle("Model 4 distribution of observed vers
```




```
ppc_stat_grouped(ds2$log_weight, yrep4, group = ds2$preterm, stat = 'median')
```




```
samp100_4 <- sample(nrow(yrep4), 100)

df2 <- as.tibble(t(yrep4[samp100_2, ]))

df2 <- df2 |> cbind(i = 1:3842, log_weight_obs = ds2$log_weight)|>
pivot_longer(`V1`:`V100`, names_to = "simus", values_to = "log_weight_rep")
test_stat_obs <- mean(ds2$birthweight < 2.5)
test_stat_rep <- df2 |> group_by(simus) |>
summarize(Prop_BW = mean(exp(log_weight_rep) < 2.5))
test_stat_rep |>
ggplot(aes(x = Prop_BW)) +
geom_histogram(bins = 20, fill = "lightblue") +
geom_vline(xintercept = test_stat_obs, color = "darkred") +theme_bw() +
labs(caption = "simulated TS with light blue while real TS with dark red",
title = "Model 4-Test Statistics of PPC for proportion of births' weights < 2.5 kg")
```



Via checking both,the model4 did not more improvements than the model2.