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)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0     v purrr   1.0.1
## v tibble  3.1.8     v dplyr   1.1.0
## v tidyr   1.2.1     v stringr 1.5.0
## v readr   2.1.3     v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(here)
```

```
## here() starts at C:/Users/nigel/OneDrive/School/First Year Masters/STA2201/Labs
```

```
# for bayes stuff
library(rstan)
```

```
## Loading required package: StanHeaders
##
## rstan version 2.26.15 (Stan version 2.26.1)
##
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
## For within-chain threading using ‘reduce_sum()‘ or ‘map_rect()‘ Stan functions,
## change ‘threads_per_chain‘ option:
## rstan_options(threads_per_chain = 1)
##
## Do not specify '-march=native' in 'LOCAL_CPPFLAGS' or a Makevars file
##
## Attaching package: 'rstan'
##
## The following object is masked from 'package:tidyr':
##
##     extract
```

```
library(bayesplot)
```

```
## This is bayesplot version 1.10.0
## - Online documentation and vignettes at mc-stan.org/bayesplot
## - bayesplot theme set to bayesplot::theme_default()
##    * Does _not_ affect other ggplot2 plots
##    * See ?bayesplot_theme_set for details on theme setting
```

```
library(loo)
```

```
## This is loo version 2.5.1
## - Online documentation and vignettes at mc-stan.org/loo
## - As of v2.0.0 loo defaults to 1 core but we recommend using as many as possible. Use the 'cores' ar
## - Windows 10 users: loo may be very slow if 'mc.cores' is set in your .Rprofile file (see https://gi
##
## Attaching package: 'loo'
##
## The following object is masked from 'package:rstan':
##
##     loo
```

```
library(tidybayes)

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
- `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

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.

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

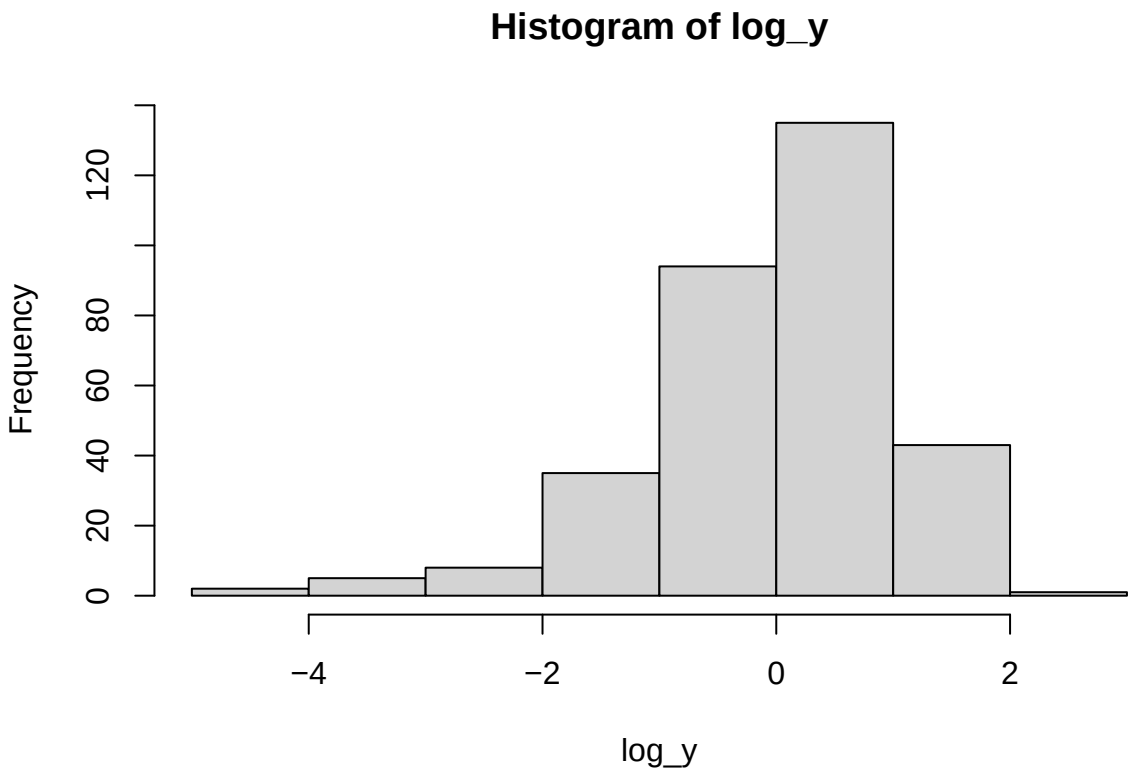
For Model 1, simulate values of βs and σ based on the priors above. Do 1000 simulations. Use these values to simulate (log) birth weights from the likelihood specified in Model 1, based on the set of observed gestational weights. **Remember the gestational weights should be centered and standardized**.

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

```
beta_1 <- rnorm(1000, 0, 1)
beta_2 <- rnorm(1000, 0, 1)
sigma <- rnorm(1000, 0, 1)
sigma <- sigma[sample(which(sigma>0))]

x_1 <- (ds$gest - mean(ds$gest))/sd(ds$gest)
log_y <- log(rnorm(1000, mean = beta_1 + beta_2*log(x_1), sd = sigma))

hist(log_y)
```



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 = 500,
             seed = 243)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000294 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 2.94 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: 0.497 seconds (Warm-up)
## Chain 1:                0.401 seconds (Sampling)
## Chain 1:                0.898 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.000156 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 1.56 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: 0.506 seconds (Warm-up)
## Chain 2:                0.448 seconds (Sampling)
## Chain 2:                0.954 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0.00014 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 1.4 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: 0.472 seconds (Warm-up)
## Chain 3:                0.405 seconds (Sampling)
## Chain 3:                0.877 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0.000141 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 1.41 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: 0.481 seconds (Warm-up)
## Chain 4:                0.406 seconds (Sampling)
## Chain 4:                0.887 seconds (Total)
## Chain 4:
```

```
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 birth weight of a baby who was born at a gestational age of 37 weeks.

```
posterior_samples_1 <- extract(mod1)
beta_11 <- posterior_samples_1[["beta"]][,1]
beta_12 <- posterior_samples_1[["beta"]][,2]
sigma_1 <- posterior_samples_1[["sigma"]]

x_new <- 37

beta_11_hat <- median(beta_11)
beta_12_hat <- median(beta_12)

pred_1 <- exp(beta_11_hat + beta_12_hat*log(x_new))
pred_1
```

```
## [1] 5.372529
```

Question 4

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

```
z0 = as.numeric(as.factor(ds$preterm)) - (1+numeric(length(ds$preterm)))

my_data <- list(N = nrow(ds),
                log_weight = ds$log_weight,
                log_gest = ds$log_gest_c,
                preterm = z0)
```

```
my_mod_2 <- stan(data = my_data,
                 file = "my_model_2.stan",
                 iter = 500)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000765 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 7.65 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: 51.52 seconds (Warm-up)
## Chain 1:                106.958 seconds (Sampling)
## Chain 1:                158.478 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.000487 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 4.87 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: 49.696 seconds (Warm-up)
## Chain 2:                98.426 seconds (Sampling)
## Chain 2:                148.122 seconds (Total)
```

```
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0.000553 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 5.53 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: 47.019 seconds (Warm-up)
## Chain 3:                116.524 seconds (Sampling)
## Chain 3:                163.543 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0.000581 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 5.81 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: 58.221 seconds (Warm-up)
## Chain 4:                132.412 seconds (Sampling)
## Chain 4:                190.633 seconds (Total)
## Chain 4:
```

Question 5

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

```
load("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
```

```
summary(my_mod_2)
```

```
## $summary
##                mean      se_mean          sd         2.5%          25%
## beta1  1.168302e+00 1.020957e-04 0.002971074    1.1625449    1.1662994
## beta2 -6.449544e-03 4.945521e-02 0.729754449   -1.4158599   -0.4901246
## beta3 -3.184399e-01 1.252012e-03 0.026377671   -0.3701563   -0.3355469
## beta4  1.224866e-01 4.944876e-02 0.729799720   -1.2841826   -0.3503431
## sigma  4.072071e-01 9.315957e-05 0.002308403    0.4026811    0.4056710
## lp__   4.980739e+03 8.492534e-02 1.552392782 4977.1205133 4979.8246527
##                50%          75%        97.5%    n_eff      Rhat
## beta1    1.1683522    1.1703335    1.1739704 846.8607 0.9992593
## beta2    0.0114571    0.4662853    1.3983260 217.7356 1.0129714
## beta3   -0.3172816   -0.3009439   -0.2652538 443.8699 1.0115592
## beta4    0.1036870    0.6039430    1.5349222 217.8194 1.0129674
## sigma    0.4072128    0.4086702    0.4119309 614.0001 1.0013966
## lp__  4981.0721121 4981.8745647 4982.9716623 334.1402 1.0071280
##
## $c_summary
## , , chains = chain:1
##
##          stats
## parameter          mean          sd         2.5%          25%          50%
##     beta1    1.16818570 0.002982566    1.1622307    1.1661329    1.16837067
##     beta2    0.09004853 0.751069822   -1.2883644   -0.4183690    0.15378921
##     beta3   -0.31847826 0.026132501   -0.3676368   -0.3351943   -0.31815625
##     beta4    0.02559513 0.751470484   -1.2991535   -0.4448063   -0.03554961
##     sigma    0.40719552 0.002468207    0.4027023    0.4054058    0.40709363
##     lp__  4980.70395718 1.512964179 4977.4257574 4979.9758271 4980.99520155
##          stats
## parameter          75%        97.5%
##     beta1    1.1701661    1.1735927
##     beta2    0.5622588    1.4122671
```

```
##     beta3   -0.2998643   -0.2665455
##     beta4    0.5361949    1.4062788
##     sigma    0.4088251    0.4125998
##     lp__  4981.8102043 4982.9960831
##
## , , chains = chain:2
##
##          stats
## parameter          mean          sd         2.5%          25%           50%
##     beta1  1.168159e+00 0.002997490    1.1629927    1.1659200    1.16823963
##     beta2  5.010631e-03 0.692151026   -1.1993678   -0.4484233   -0.02886277
##     beta3 -3.127290e-01 0.023925868   -0.3586765   -0.3271891   -0.31289610
##     beta4  1.118385e-01 0.691861327   -1.1120629   -0.3053990    0.14690430
##     sigma  4.070786e-01 0.001946688    0.4035475    0.4056148    0.40705543
##     lp__   4.980913e+03 1.479695510 4977.3571034 4980.0278015 4981.24443888
##          stats
## parameter          75%        97.5%
##     beta1    1.1701772    1.1740219
##     beta2    0.4249088    1.2311864
##     beta3   -0.2990378   -0.2626359
##     beta4    0.5644484    1.3075371
##     sigma    0.4083881    0.4109662
##     lp__  4982.0670737 4983.0456570
##
## , , chains = chain:3
##
##          stats
## parameter          mean          sd         2.5%          25%           50%
##     beta1    1.16831786 0.003076280    1.1624409    1.1661861    1.16832194
##     beta2    0.03172603 0.659552558   -1.1977340   -0.4610615    0.09610040
##     beta3   -0.31918125 0.027979992   -0.3699807   -0.3385285   -0.31849647
##     beta4    0.08436631 0.659617373   -1.0657071   -0.3709300    0.02173374
##     sigma    0.40743427 0.002185753    0.4030610    0.4059976    0.40733151
##     lp__  4980.78373398 1.424731191 4977.2433647 4979.8628761 4981.10790789
##          stats
## parameter          75%        97.5%
##     beta1    1.1705415    1.1740788
##     beta2    0.4865104    1.1787167
##     beta3   -0.3005715   -0.2620903
##     beta4    0.5759400    1.3160658
##     sigma    0.4088489    0.4118652
##     lp__  4981.7435965 4982.9312377
##
## , , chains = chain:4
##
##          stats
## parameter         mean          sd         2.5%          25%          50%
##     beta1    1.1685459 0.002824417    1.1633167    1.1667013    1.1684375
##     beta2   -0.1525834 0.791207094   -1.7261798   -0.6348380   -0.1227727
##     beta3   -0.3233711 0.026373496   -0.3720111   -0.3425817   -0.3235292
##     beta4    0.2681466 0.791247055   -1.3528062   -0.1640806    0.2400451
##     sigma    0.4071200 0.002577554    0.4017711    0.4056673    0.4073458
##     lp__  4980.5552829 1.758674374 4976.2272630 4979.5943342 4980.9389880
##          stats
```

```
## parameter          75%        97.5%
##     beta1    1.1703256    1.1740117
##     beta2    0.2810719    1.4700629
##     beta3   -0.3034304   -0.2733121
##     beta4    0.7506320    1.8422322
##     sigma    0.4088971    0.4117827
##     lp__  4981.8690686 4982.9386247
```

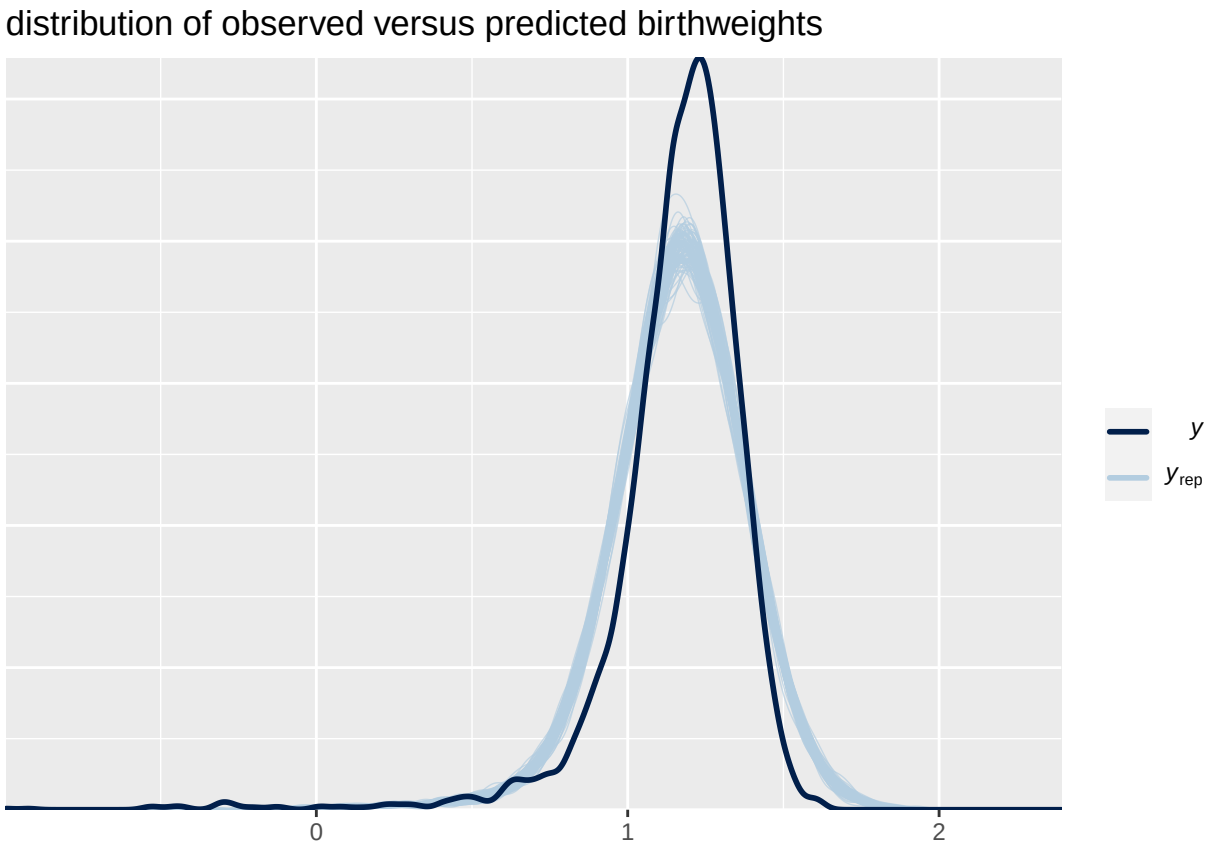
PPCs

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"]]
dim(yrep1)
```

```
## [1] 1000 3842
```

```
samp100 <- sample(nrow(yrep1), 100)
ppc_dens_overlay(y, yrep1[samp100, ])  + ggtitle("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`)

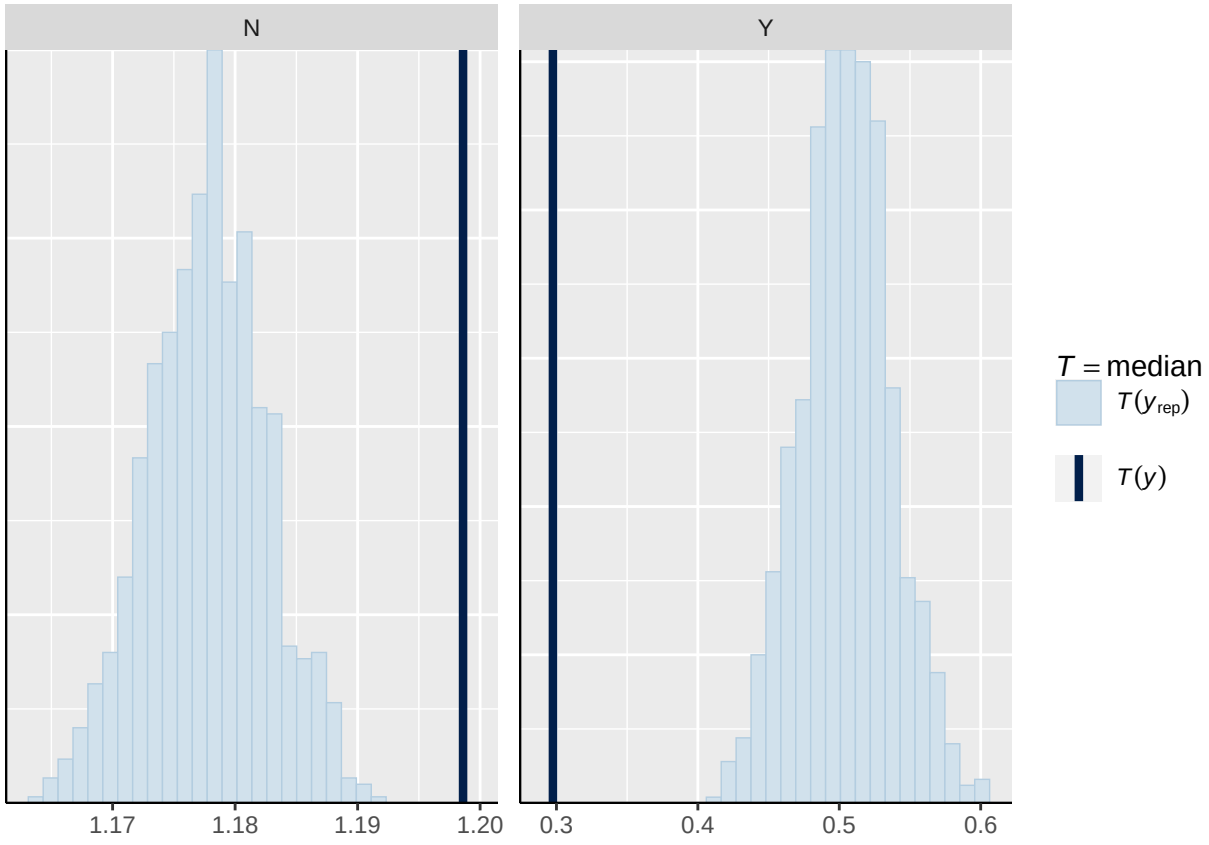
Test statistics

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')
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

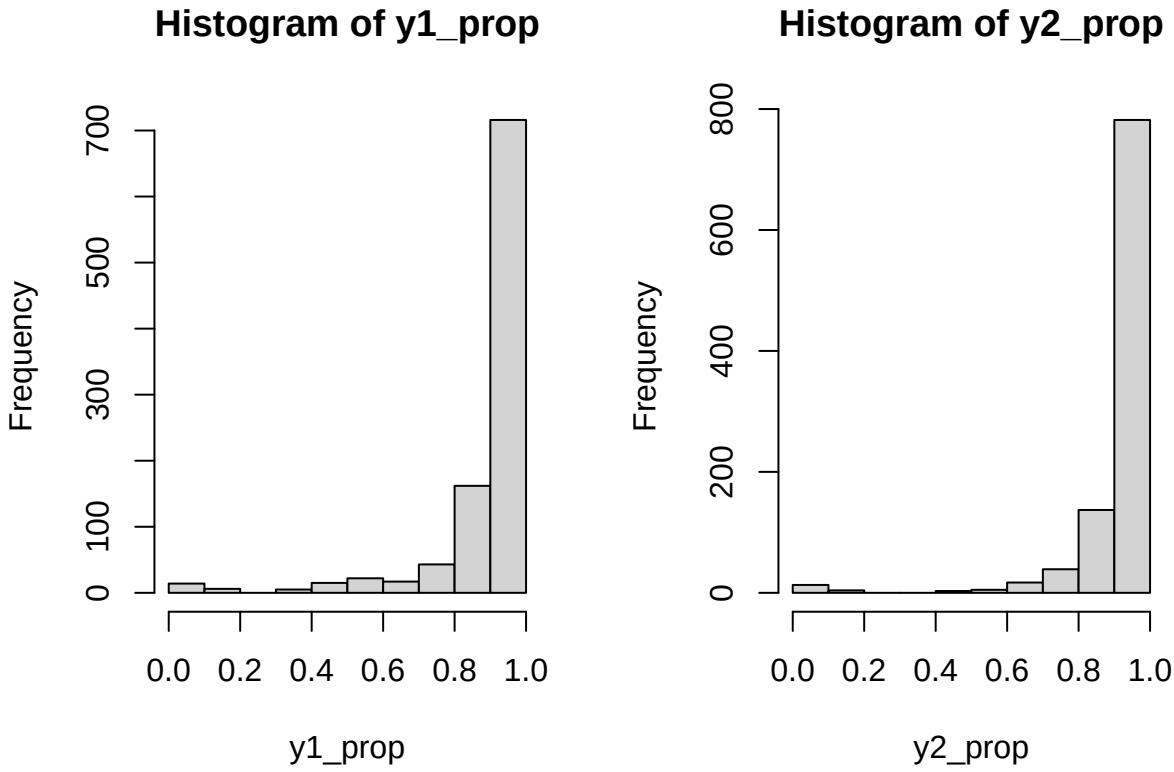


Question 7

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).

```
y_prop <- length(which(y > log(2.5)))/length(y)

y1_prop <- numeric(0)
y2_prop <- numeric(0)
for (i in 1:1000){
  y1_prop[i]= length(which(yrep1[,i]>log(2.5)))/length(yrep1[,i])
  y2_prop[i]= length(which(yrep2[,i]>log(2.5)))/length(yrep2[,i])
}
par(mfrow=c(1,2))
hist(y1_prop)
hist(y2_prop)
```



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   1377.0  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 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:

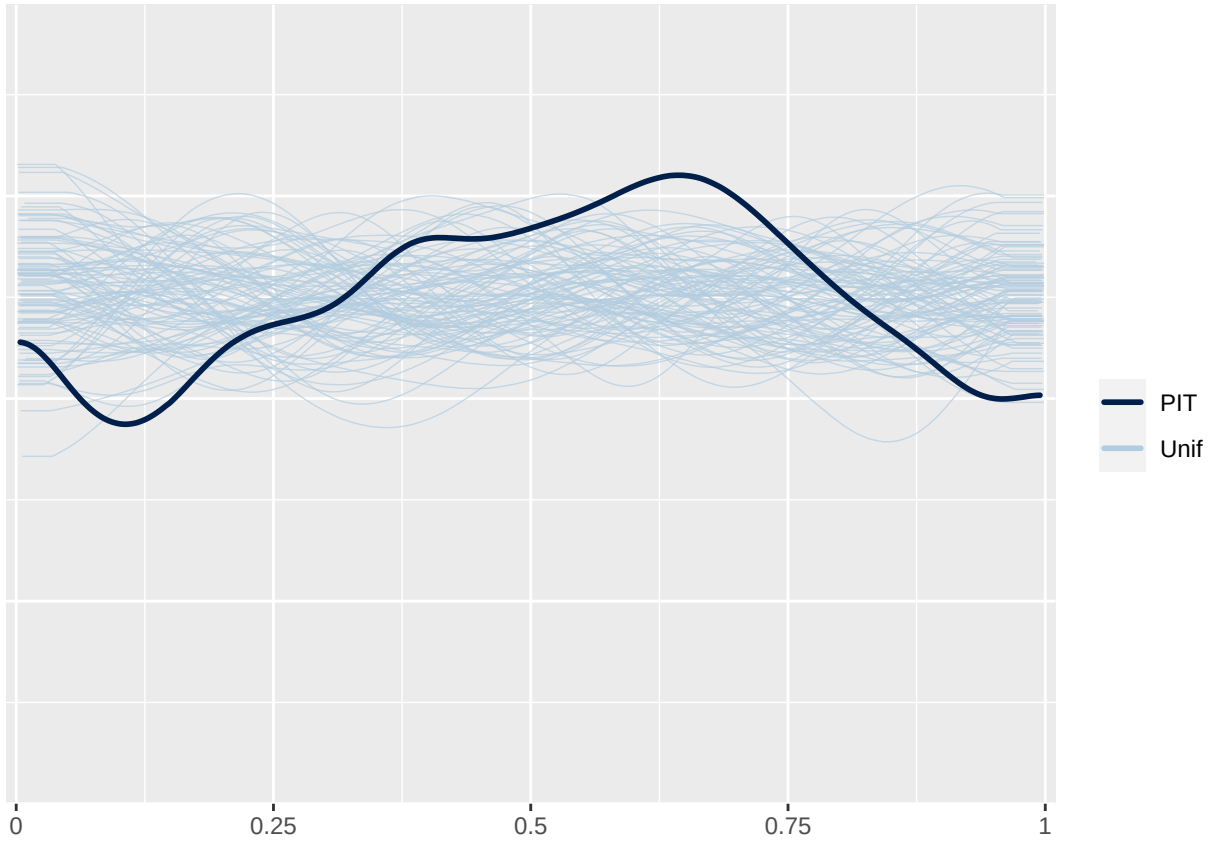
```
loo_compare(loo1, loo2)
```

```
##        elpd_diff se_diff
## model2    0.0       0.0
## model1 -175.8      36.2
```

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

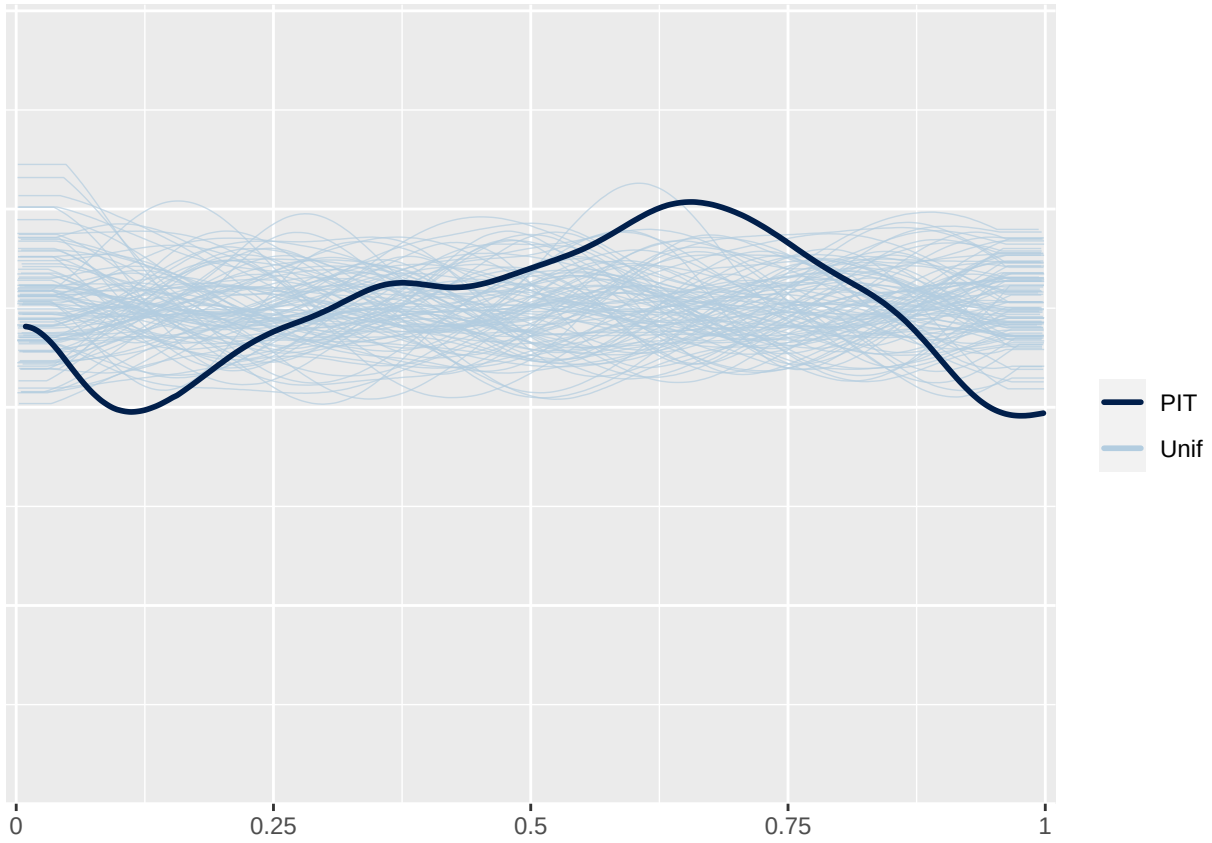
```
ppc_loo_pit_overlay(yrep = yrep1, y = y, lw = weights(loo1$psis_object))
```

```
## NOTE: The kernel density estimate assumes continuous observations and is not optimal for discrete ob
```




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

```
## NOTE: The kernel density estimate assumes continuous observations and is not optimal for discrete ob
```



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.