PubH 7440 - Homework 4

Jake Wittman

4/1/2020

1a)

```r
# Extract y_reps from output
y_reps <- extract(out_model1, pars = c("y_rep_bike", "y_rep_nonbike",
                                       "lambda_rep_bike", "lambda_rep_nonbike"))
y_rep_bike <- y_reps$y_rep_bike
y_rep_nonbike <- y_reps$y_rep_nonbike

# Calculate P(y_rep > yi | y)
# y_rep - yi
y_diff_bike <- t(y_rep_bike) - bikes$bikes
y_diff_nobike <- t(y_rep_nonbike) - no_bikes$bikes
# If y_rep > yi 1, else 0
y_diff_bike <- ifelse(y_diff_bike > 0, 1, 0)
y_diff_nobike <- ifelse(y_diff_nobike > 0, 1, 0)
# Apply mean along row to get probability that y_rep > yi
apply(y_diff_bike, 1, mean) %>% mean(.)
```

```
## [1] 0.5096067
```

```r
apply(y_diff_nobike, 1, mean) %>% mean(.)
```

```
## [1] 0.4421028
```

The values from the apply function are the average number of times the predicted y_{rep} was larger than the observed. The probability for the bike model is about 0.51, while the probability $y_{rep} > y_i$ for the non-bike route model is about 0.44. These values tell us that our model predictions are about as likely to be larger than our observed values as they are to be smaller than our observed values.

1b)

model	DIC	WAIC
1	109.7139	108.9922
2	323.2877	175.5077
3	112.5688	107.2593

The creators of Stan don't recommend using DIC as a model selection tool; they recommend WAIC or LOO instead. It is not straightforward to produce DIC from Stan output, but it is pretty easy to get WAIC so I calculated both to see how they would compare.

Both WAIC and DIC provide the largest values for model 2, while DIC gives the lowest value for model 1 and WAIC gives the lowest value for model 3. The DIC/WAIC values for models 1 and 3 are fairly close together, however, suggesting that both models 1 and 3 may be adequate for the data.

2a)

```
## Compiling the C++ model
```

```
## Start sampling
```

	Mean	Median	Lower 95% CI	Upper 95% CI
County 1	1.9586	1.9584	1.5253	2.3929
County 2	1.8749	1.8743	1.4139	2.3401
County 3	1.9191	1.9204	1.4557	2.3785
First Floor	-0.3279	-0.3304	-0.9618	0.3051
Sigma	0.8065	0.7970	0.6448	1.0227

2b) The predicited log radon level from a house in Blue Earth county taken on the basement is 1.959 (95% CI: [0.289, 3.635]). The predicted log radon from a house in Blue Earth county taken on the first floor is 1.629 (95% CI: [-0.108, 3.376]).

```
## Compiling the C++ model

## recompiling to avoid crashing R session

## Start sampling

## Output of model 'radon_fit':
##
## Computed from 27000 by 41 log-likelihood matrix
##
##           Estimate  SE
## elpd_waic    -51.7 4.7
## p_waic         4.8 1.1
## waic         103.5 9.4

## Warning: 2 (4.9%) p_waic estimates greater than 0.4. We recommend trying
## loo instead.

##
## Output of model 'radon_fit2':
##
## Computed from 27000 by 41 log-likelihood matrix
##
##           Estimate   SE
## elpd_waic    -51.2  5.0
## p_waic         3.9  1.0
## waic         102.4 10.1

## Warning: 2 (4.9%) p_waic estimates greater than 0.4. We recommend trying
## loo instead.

##
## Model comparisons:
##            elpd_diff se_diff
## radon_fit2  0.0       0.0
## radon_fit  -0.5       1.1
```

2c) BRMS uses Stan on the backend so again, DIC is not available but WAIC is. The model without floor level has a lower WAIC (WAIC difference = 0.5). It is minimal, but suggests that the second model may be a better fit.

2d)

```
## Iteration: 1
## Iteration: 2
## Iteration: 3
## Iteration: 4
## Iteration: 5
## Iteration: 1
## Iteration: 2
## Iteration: 3
## Iteration: 4

## Estimated Bayes factor in favor of bridge1 over bridge2: 18245.10752
```

The Bayes Factor for M2/M1 is very large: 18226. If this is correct, it suggests that the model withot the floor level is a better model conditional on our data.

```
## Compiling the C++ model

## Start sampling

##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: penicillin_yield ~ block + treatment
##    Data: pen_long (Number of observations: 20)
## Samples: 3 chains, each with iter = 10000; warmup = 1000; thin = 1;
##          total post-warmup samples = 27000
##
## Population-Level Effects:
##            Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     90.01      3.09    83.78    96.26 1.00    13639    14887
## block2        -8.97      3.42   -15.68    -2.04 1.00    15699    17385
## block3        -6.97      3.45   -13.86    -0.03 1.00    16328    15678
## block4        -3.98      3.43   -10.91     2.85 1.00    16215    16808
## block5        -9.99      3.44   -16.79    -3.18 1.00    16876    17702
## treatmentB     1.00      3.14    -5.27     7.24 1.00    18701    16608
## treatmentC     4.99      3.14    -1.28    11.19 1.00    18361    17994
## treatmentD     1.97      3.14    -4.32     8.19 1.00    18613    17176
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     4.78      1.09     3.18     7.42 1.00    12101    14078
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
## is a crude measure of effective sample size, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

3a) The pencilin level produced in block 1 and treatment A is 90.05 with unknown units. The only 95% credible intervals that do not include 0 are the distributions for block 2, block3, and block 5. These blocks all have lower estimated penicillin production that block 1, while block 4 seems to produce approximately the same amount of penicillin. The treatment effect 95% credible intervals all overlap 0, suggesting they do not affect penicillin production.

3b) $Penicillin.yield_{ij} \sim N(\alpha_i + \beta_j, \sigma^2)$

$\alpha_i \sim N(0, \tau^2)$

$\beta_j \sim N(0, \eta^2)$

$i = 1, ..., 5$ for the 5 different blocks and $j = 1, ..., 4$ for the 4 treatments. The indicator groups should be fixed at 0 because we are trying to estimate their deviations from the population mean, which is estimated by the intercept.

3c)

```
## Compiling the C++ model
## Start sampling
```

```
## Warning: There were 22 divergent transitions after warmup. Increasing adapt_delta above 0.99 may hel
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

Based on the output of the hierarchical model, the average amount of penicillin yielded in our population is 85.97 [95% CI: 77.93 - 93.98]. The block random effect has a standard deviation of 5.27 [95% CI: 0.59 - 15.68] while the treatment random effect has a standard deviation of 3.39 [95% CI: 0.11 - 14.30].

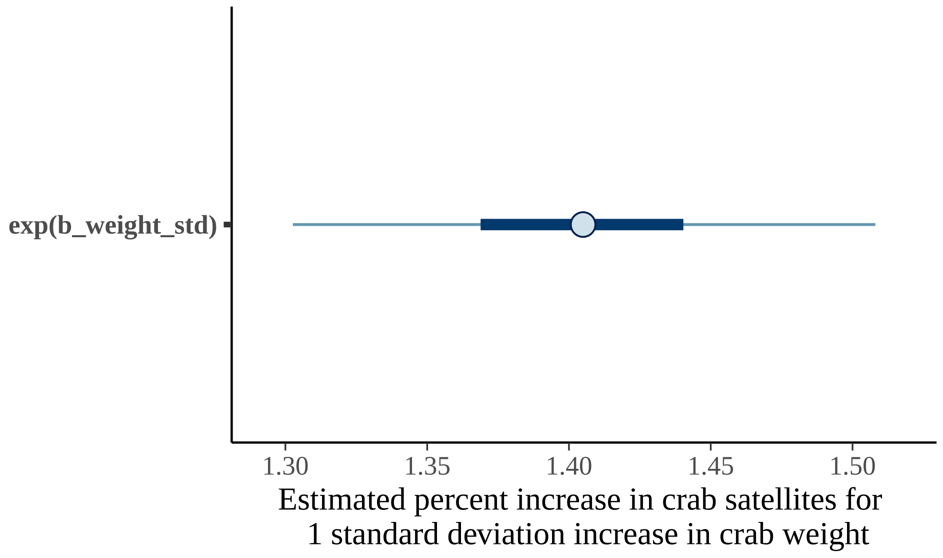
3d) Using BRMS, I am not able to calculate DIC. Instead I will substitute WAIC and loo estimates. The $\Delta WAIC = -0.8$) favoring the hierarchical model, while the $\Delta LOO = -1.1$ favoring the hierarchical model also.

```
## Compiling the C++ model
```

```
## Start sampling
```

```
##  Family: poisson
##   Links: mu = log
## Formula: satell ~ weight_std
##    Data: crabs (Number of observations: 173)
## Samples: 3 chains, each with iter = 5000; warmup = 1000; thin = 1;
##          total post-warmup samples = 12000
##
## Population-Level Effects:
##            Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept      1.01      0.05     0.91     1.10 1.00     6965     7515
## weight_std     0.34      0.04     0.26     0.41 1.00     6662     6555
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
## is a crude measure of effective sample size, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

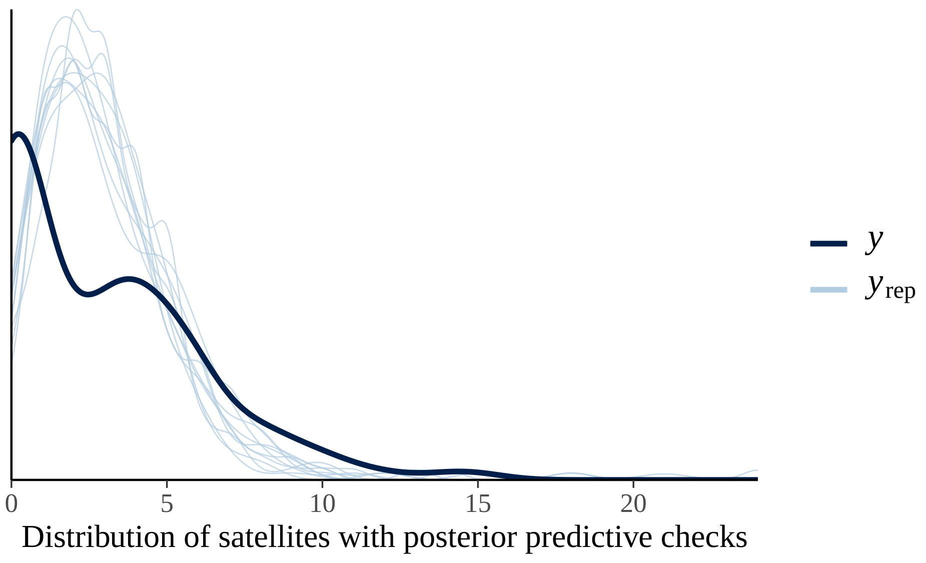
4a)



The median percent increase in crab satellites as weight increases by 1 standard deviation is 1.404 (95% CI: [1.303, 1.508]).

4b)

```
## Using 10 posterior samples for ppc type 'dens_overlay' by default.
```



The predicted y_{rep} values compared to the observed y values suggest there may be some overdispersion in oru data.

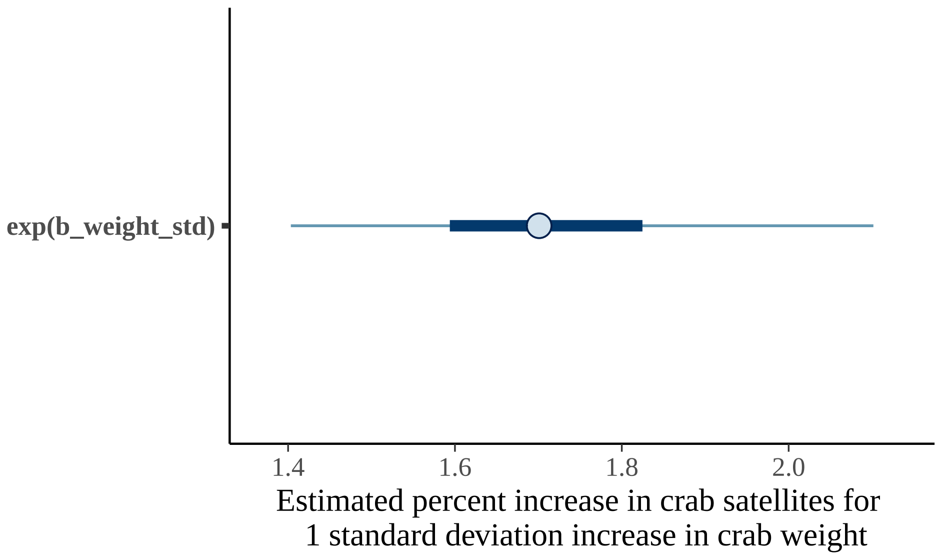
```
## Compiling the C++ model
```

```
## Start sampling
```

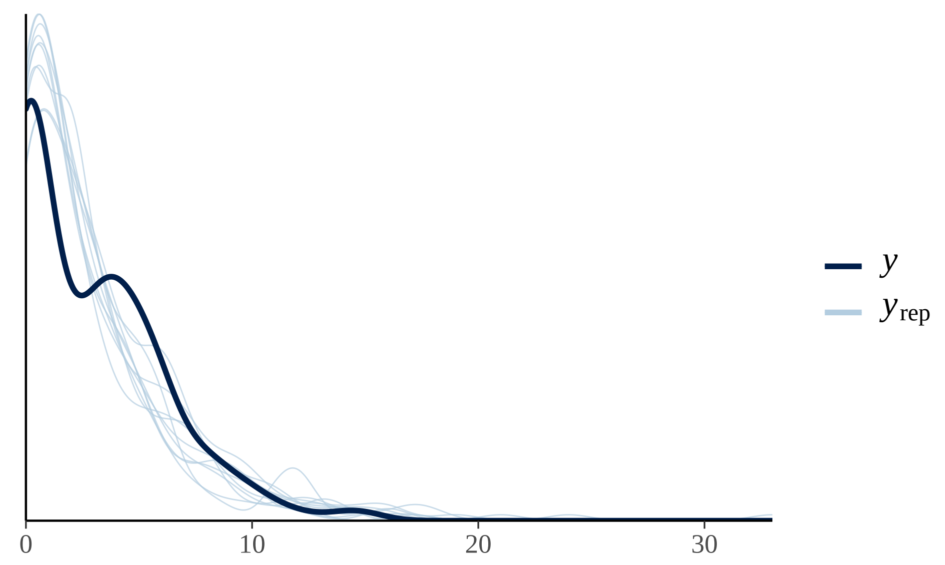
```
## Compiling the C++ model
```

```
## Start sampling
```

4c) The median percent increase in crab satellites as weight increases by 1 standard deviation is 1.706 (95% CI: [1.403, 2.102]).



```
## Using 10 posterior samples for ppc type 'dens_overlay' by default.
```



The posterior predictive checks show the hierarchical model seems to produce values more similar to our observed values than the non-hierarchical model.

4d) WAIC for these two models suggest that the hierarchical model estimating crab satellites is a better fit ($\Delta WAIC = 132.8$). LOO CV also suggests that the hierarchical model is a better fit ($\Delta LOO = 96.4$)

Code Appendix

```r
knitr::opts_chunk$set(echo = FALSE, include = FALSE, fig.width = 5, fig.height = 3)
# Load libraries ----------------------------------------------------------
if (!require(pacman)) install.packages("pacman")
pacman::p_load(tidyverse,
               rstan,
               brms,
               bayesplot,
               loo,
               conflicted,
               here,
               bayesplot)
# Deal with package conflicts
conflict_prefer("extract", "rstan")
conflict_prefer("dirichlet", "brms")
conflict_prefer("exponential", "brms")
conflict_prefer("filter", "stats")
conflict_prefer("lag", "stats")
conflict_prefer("loo", "loo")
conflict_prefer("Position", "ggplot2")
# Q1
# data --------------------------------------------------------------------
intersections <- 1:18
streets <- c(rep(1, 10), rep(0, 8))
bikes <- c(16, 9, 10, 13, 19, 20, 18, 17, 35, 55, 12, 1, 2, 4, 9, 7, 9, 8)

bike <- data.frame(
  intersections = intersections,
  streets = streets,
  bikes = bikes
)

bikes <- bike %>% dplyr::filter(streets == 1)
no_bikes <- bike %>% dplyr::filter(streets == 0)


# Stan model -------------------------------------------------------------
bike_model1 <- rstan::stan_model(file = here::here("homework/hw4.stan"))
# 1a ---------------------------------------------------------------------
set.seed(1)
n_cores <- parallel::detectCores() - 1
out_model1 <- sampling(
  object = bike_model1,
  data = list(y_bike1 = bikes$bikes,
              n_bike = nrow(bikes),
              y_nonbike1 = no_bikes$bikes,
              n_nonbike = nrow(no_bikes),
              y_bike2 = bikes$bikes,
              y_nonbike2 = no_bikes$bikes,
              y_bike3 = bikes$bikes,
              y_nonbike3 = no_bikes$bikes),
  warmup = 5000,
  iter = 20000,
  chains = 3,
  cores = n_cores,
  control = list(adapt_delta = 0.999),
  show_messages = FALSE
)

# Check that chains mixed
#traceplot(out_model1, par = c("alpha_nonbike1", "beta_nonbike1", "alpha_bike1", "beta_bike1"))
#traceplot(out_model1, par = c("lambda_bike2", "lambda_bike3", "lambda_nonbike2", "lambda_nonbike3"))
# Check gelman-rubin
out_model1
# Extract y_reps from output
y_reps <- extract(out_model1, pars = c("y_rep_bike", "y_rep_nonbike",
                                       "lambda_rep_bike", "lambda_rep_nonbike"))
y_rep_bike <- y_reps$y_rep_bike
y_rep_nonbike <- y_reps$y_rep_nonbike

# Calculate P(y_rep > yi | y)
# y_rep - yi
y_diff_bike <- t(y_rep_bike) - bikes$bikes
y_diff_nobike <- t(y_rep_nonbike) - no_bikes$bikes
# If y_rep > yi 1, else 0
y_diff_bike <- ifelse(y_diff_bike > 0, 1, 0)
y_diff_nobike <- ifelse(y_diff_nobike > 0, 1, 0)
# Apply mean along row to get probability that y_rep > yi
apply(y_diff_bike, 1, mean) %>% mean(.)
apply(y_diff_nobike, 1, mean) %>% mean(.)
```

```r
y_all <- c(no_bikes$bikes, bikes$bikes)
postSamples1 <- Reduce(cbind, extract(out_model1,
                                       pars=c("lambda_nonbike1",
                                              "lambda_bike1")))
postSamples2 <- Reduce(cbind, extract(out_model1,
                                       pars=c("lambda_nonbike2",
                                              "lambda_bike2")))

postSamples3 <- Reduce(cbind, extract(out_model1,
                                       pars=c("lambda_nonbike3",
                                              "lambda_bike3")))


dic <- function(data, likelihood, postSamples){

  logLikelihood <- function(theta) sum(likelihood(data, theta, log=T))

  thetaBayes <- colMeans(postSamples)

  pDIC <- 2*(logLikelihood(thetaBayes) - mean(apply(postSamples, 1, logLikelihood) ))
  dic <- -2*logLikelihood(thetaBayes) + 2*pDIC

  return(dic)
}

dic1 <- dic(y_all, dpois, postSamples1)
dic2 <- dic(y_all, dpois, postSamples2)
dic3 <- dic(y_all, dpois, postSamples3)

# Stan doesn't allow users to get DIC values easily, they recommend using WAIC or LOO
# instead. I want to compare WAIC to the DIC
# Extract log_likelihood for each model Log_likelihood is generated explicitly
# in the generated quantities code for stan and then extracted
# I wasn't quite sure how to get the log likelihood for the bike & non-bike
# models together. So I calculated the likelihood separately for each
# and then concatenated the two vectors in stan.
log_lik1 <- extract_log_lik(out_model1, parameter_name = "log_lik1")
log_lik2 <- extract_log_lik(out_model1, parameter_name = "log_lik2")
log_lik3 <- extract_log_lik(out_model1, parameter_name = "log_lik3")

waic1 <- waic(log_lik1)
waic2 <- waic(log_lik2)
waic3 <- waic(log_lik3)

waic_out <- c(waic1$estimates[3, 1], waic2$estimates[3, 1], waic3$estimates[3, 1])
dic_out <- c(dic1, dic2, dic3)

IC_out <- data.frame(
  model = 1:3,
  DIC = dic_out,
  WAIC = waic_out
)
knitr::kable(IC_out)
```

```r
# Q2
# Read in data and clean a bit
radon <- read.csv(here::here("homework/radon.csv"))
radon$county_num <- as.factor(radon$county_num)
radon$log_radon <- log(radon$radon)


# Test out brms
prior <- c(set_prior("normal(0, 1e4)", class = "b"), #prior for betas
           set_prior("inv_gamma(0.01, 0.01)", class = "sigma") # prior for sigmas
           )

radon_fit <- brm(log_radon ~ county_num + first_floor - 1,
                 data = radon,
                 prior = prior,
                 warmup = 1000,
                 iter = 10000,
                 chains = 3,
                 cores = n_cores,
                 seed = 1,
                 save_all_pars = TRUE)

posterior_summary <- as.data.frame(posterior_summary(radon_fit, robust = FALSE))
posterior_median <- posterior_summary(radon_fit, robust = TRUE)[, 1]
posterior_summary$median <- posterior_median
row.names(posterior_summary) <- c("County 1", "County 2", "County 3", "First Floor", "Sigma", "lp")
posterior_summary <- posterior_summary[1:5, c(1, 5, 3, 4)]

#2a

knitr::kable(
  posterior_summary,
  row.names = TRUE,
  col.names = c("Mean", "Median", "Lower 95% CI", "Upper 95% CI"),
  digits = 4
)
predict_dat <- data.frame(county_num = c(1, 1),
                          first_floor = c(0, 1))
predict_radon <- predict(radon_fit, newdata = predict_dat) %>%
  as_tibble() %>%
  bind_cols(predict_dat)

prior <- c(set_prior("normal(0, 1e4)", class = "b"), #prior for betas
           set_prior("inv_gamma(0.01, 0.01)", class = "sigma") # prior for sigmas
           )

radon_fit2 <- brm(log_radon ~ county_num - 1,
                 data = radon,
                 prior = prior,
                 warmup = 1000,
                 iter = 10000,
                 chains = 3,
                 cores = n_cores,
                 seed = 1,
                 save_all_pars = TRUE)

WAIC(radon_fit, radon_fit2)
set.seed(1)
bf_radon <- bayes_factor(radon_fit2, radon_fit)
bf_radon
# Question 3
pen <- read.csv(here::here("homework/penicillin.csv"))
pen_long <- pen %>%
  mutate(block = case_when(block1 == 1 ~ 1,
                           block2 == 1 ~ 2,
                           block3 == 1 ~ 3,
                           block4 == 1 ~ 4,
                           block5 == 1 ~ 5),
         treatment = case_when(treatA == 1 ~ "A",
                               treatB == 1 ~ "B",
                               treatC == 1 ~ "C",
                               treatD == 1 ~ "D")) %>%
  select(penicillin_yield, block, treatment) %>%
  mutate(block = as.factor(block),
         treatment = as.factor(treatment))
priors <- set_prior("normal(0, 1e3)")
pen_fit1 <- brm(penicillin_yield ~ block + treatment, data = pen_long,
                prior = priors,
                warmup = 1000,
                iter = 10000,
                chains = 3,
                cores = n_cores,
                seed = 1,
                save_all_pars = TRUE)

summary(pen_fit1)

priors <- c(set_prior("normal(0, 1e2)",  class = "sd"), #vague prior for the stdev of the random effect
            set_prior("normal(0, 1e3)", class = "Intercept")) # vague prior on the population mean
pen_fit2 <- brm(penicillin_yield ~  (1|block) + (1|treatment) ,
                data = pen_long,
                prior = priors,
                warmup = 5000,
                iter = 20000,
                chains = 3,
                cores = n_cores,
                control = list(adapt_delta = 0.99),
                seed = 1,
                save_all_pars = TRUE)

set.seed(1)
WAIC(pen_fit1, pen_fit2)
loo(pen_fit1, pen_fit2)
# 4
crabs <- read.csv(here::here("homework/crabs.csv"))
# I'm going to standardize weight since it's measured in such large units
```

```r
crabs$weight_centered <- (crabs$weight - mean(crabs$weight))
crabs$weight_std <- (crabs$weight - mean(crabs$weight)) / (sd(crabs$weight))
priors <- c(set_prior("normal (0, 1e6)", class = "b"),
            set_prior("normal (0, 1e6)", class = "Intercept"))

crab_fit1 <- brm(satell ~ weight_std,
                 data = crabs,
                 family = poisson(),
                 prior = priors,
                 chains = 3,
                 warmup = 1000,
                 iter = 5000,
                 cores = n_cores,
                 seed = 1)
summary(crab_fit1)
beepr::beep(6)


mcmc_intervals(crab_fit1,
               pars = "b_weight_std",
               prob_outer = 0.95,
               transformations = "exp") +
  labs(x = "Estimated percent increase in crab satellites for \n 1 standard deviation increase in crab 
pp_check(crab_fit1) +
  labs(x = "Distribution of satellites with posterior predictive checks")
# Add unique identifier for hierarchical model
crabs$uid <- 1:nrow(crabs)

priors <- c(set_prior("normal (0, 1e3)", class = "b"),
            set_prior("normal (0, 1e3)", class = "sd"),
            set_prior("normal (0, 1e3)", class = "Intercept"))
crab_fit2 <- brm(satell ~ weight_std + (1|uid),
                 data = crabs,
                 family = poisson(),
                 prior = priors,
                 warmup = 1000,
                 chains = 3,
                 iter = 10000,
                 cores = n_cores,
                 seed = 1)

# Just curious - want to try a negative binomial model
priors <- c(set_prior("normal (0, 1e2)", class = "b"),
            set_prior("gamma (0.01, 0.01)", class = "shape"),
            set_prior("normal (0, 1e2)", class = "Intercept"))
crab_fit3 <- brm(satell ~ weight_std,
                 data = crabs,
                 family = negbinomial,
                 prior = priors,
                 warmup = 1000,
                 iter = 10000,
                 chains = 3,
                 cores = n_cores,
                 seed = 1)
# Plot posterior interval for regression coefficient
mcmc_intervals(crab_fit2,
               pars = "b_weight_std",
               prob_outer = 0.95,
               transformations = "exp") +
  labs(x = "Estimated percent increase in crab satellites for \n 1 standard deviation increase in crab w

pp_check(crab_fit2)
WAIC(crab_fit1, crab_fit2, crab_fit3)
loo(crab_fit1, crab_fit2, crab_fit3)

# The LOO function suggests that K-fold CV be used instead
library(future)
plan(multiprocess)
kfold_crabs <- kfold(crab_fit1, crab_fit2, crab_fit3, K = 10, chains = 1)
# Stan code for Q1

data {
  // Define data in this block

  //data for bike route intersections
  int<lower=0> n_bike;// sample size for bike route intersections
  int<lower=0> y_bike1[n_bike]; // bike route data model 1
  int<lower=0> y_bike2[n_bike]; //bike route data model 2
  int<lower=0> y_bike3[n_bike]; //bike route data model 3

    //data for non-bike route intersections
  int<lower=0> n_nonbike;// sample size for streets intersections
  int<lower=0> y_nonbike1[n_nonbike]; // nonbike routes data1
  int<lower=0> y_nonbike2[n_nonbike]; // nonbike routes data2
  int<lower=0> y_nonbike3[n_nonbike]; // nonbike routes data3

}


parameters {
  // Define parameters in this block

  // M1 parameters
  // parameters for bike route intersection
  real<lower = 0> lambda_bike1[n_bike]; //lambda for streets
  real<lower = 0> alpha_bike1; // alpha for streets
  real<lower = 0> beta_bike1; // beta for streets

    // parameters for non-bike route intersection
  real<lower = 0> lambda_nonbike1[n_nonbike]; //lambda for streets
  real<lower = 0> alpha_nonbike1; // alpha for streets
  real<lower = 0> beta_nonbike1; // beta for streets

  // M2 parameters
  real<lower = 0> lambda_bike2;
  real<lower = 0> lambda_nonbike2;
  // M3 parameters
  real<lower = 0> lambda_bike3[n_bike];
  real<lower = 0> lambda_nonbike3[n_nonbike];
}

// The model
model {
  // Define model in this block
  // M1
  alpha_bike1 ~ gamma(0.01, 0.01);
  beta_bike1 ~ gamma(0.01, 0.01);
  alpha_nonbike1 ~ gamma(0.01, 0.01);
  beta_nonbike1 ~ gamma(0.01, 0.01);

  for (i in 1:n_bike) {
    lambda_bike1[i] ~ gamma(alpha_bike1, beta_bike1);
    y_bike1[i] ~ poisson(lambda_bike1[i]);
  }

  for (j in 1:n_nonbike) {
    lambda_nonbike1[j] ~ gamma(alpha_nonbike1, beta_nonbike1);
    y_nonbike1[j] ~ poisson(lambda_nonbike1[j]);
  }

  // M2
  lambda_bike2 ~ gamma(0.01, 0.01);
  for (i in 1:n_bike) {
    y_bike2[i] ~ poisson(lambda_bike2);
  }

  lambda_nonbike2 ~ gamma(0.01, 0.01);
  for (j in 1:n_nonbike) {
    y_nonbike2[j] ~ poisson(lambda_nonbike2);
  }

  // M3
  for (i in 1:n_bike) {
    lambda_bike3[i] ~ gamma(0.01, 0.01);
    y_bike3[i] ~ poisson(lambda_bike3[i]);
  }

  for (j in 1:n_nonbike) {
    lambda_nonbike3[j] ~ gamma(0.01, 0.01);
    y_nonbike3[j] ~ poisson(lambda_nonbike3[j]);
  }
}

generated quantities{
  // Define other generated quantities in this block

  // Difference in lambdas
 real lambda_rep_bike[n_bike];
 real y_rep_bike[n_bike];
 real lambda_rep_nonbike[n_nonbike];
 real y_rep_nonbike[n_nonbike];

 // log likelihoods for WAIC
 vector[n_bike] log_lik1_bikes;
 vector[n_nonbike] log_lik1_nonbikes;
 vector[n_bike] log_lik2_bikes;
 vector[n_nonbike] log_lik2_nonbikes;
 vector[n_bike] log_lik3_bikes;
 vector[n_nonbike] log_lik3_nonbikes;
 vector[n_bike + n_nonbike] log_lik1;
 vector[n_bike + n_nonbike] log_lik2;
 vector[n_bike + n_nonbike] log_lik3;

 for (i in 1:n_bike) {
   lambda_rep_bike[i] = gamma_rng(alpha_bike1, beta_bike1);
   y_rep_bike[i] = poisson_rng(lambda_rep_bike[i]);
   log_lik1_bikes[i] = poisson_lpmf(y_bike1[i] | lambda_bike1[i]);
   log_lik2_bikes[i] = poisson_lpmf(y_bike2[i] | lambda_bike2);
   log_lik3_bikes[i] = poisson_lpmf(y_bike3[i] | lambda_bike3[i]);
 }

 for (j in 1:n_nonbike) {
   lambda_rep_nonbike[j] = gamma_rng(alpha_nonbike1, beta_nonbike1);
   y_rep_nonbike[j] = poisson_rng(lambda_rep_nonbike[j]);
   log_lik1_nonbikes[j] = poisson_lpmf(y_nonbike1[j] | lambda_nonbike1[j]);
   log_lik2_nonbikes[j] = poisson_lpmf(y_nonbike2[j] | lambda_nonbike2);
   log_lik3_nonbikes[j] = poisson_lpmf(y_nonbike3[j] | lambda_nonbike3[j]);

 }

// Append the log likelihood vectors together
log_lik1 = append_row(log_lik1_bikes, log_lik1_nonbikes);
log_lik2 = append_row(log_lik2_bikes, log_lik2_nonbikes);
log_lik3 = append_row(log_lik3_bikes, log_lik3_nonbikes);



} // end generated quantities
```