STA365-H3

```r
library(rstanarm)
```

```
## Loading required package: Rcpp

## Registered S3 method overwritten by 'xts':
##   method     from
##   as.zoo.xts zoo

## rstanarm (Version 2.19.3, packaged: 2020-02-11 05:16:41 UTC)

## - Do not expect the default priors to remain the same in future rstanarm versions.

## Thus, R scripts should specify priors explicitly, even if they are just the defaults.

## - For execution on a local, multicore CPU with excess RAM we recommend calling

## options(mc.cores = parallel::detectCores())

## - bayesplot theme set to bayesplot::theme_default()

##    * Does _not_ affect other ggplot2 plots

##    * See ?bayesplot_theme_set for details on theme setting

##
## Attaching package: 'rstanarm'

## The following object is masked from 'package:rstan':
##
##     loo
```

```r
data(radon)
radon$county_int = as.integer(radon$county)
```

Q1

I choose $\sigma, \tau \sim N_+(0, 1^2)$ and $\beta, \gamma_0, \gamma_1 \sim N(0, 2^2)$. As the prior predictive histogram and the orginial histogram for the bootstrap sample are quite similar (both are right-skewed, unimodal and having a relatively close scale), I think the choices are sensible.

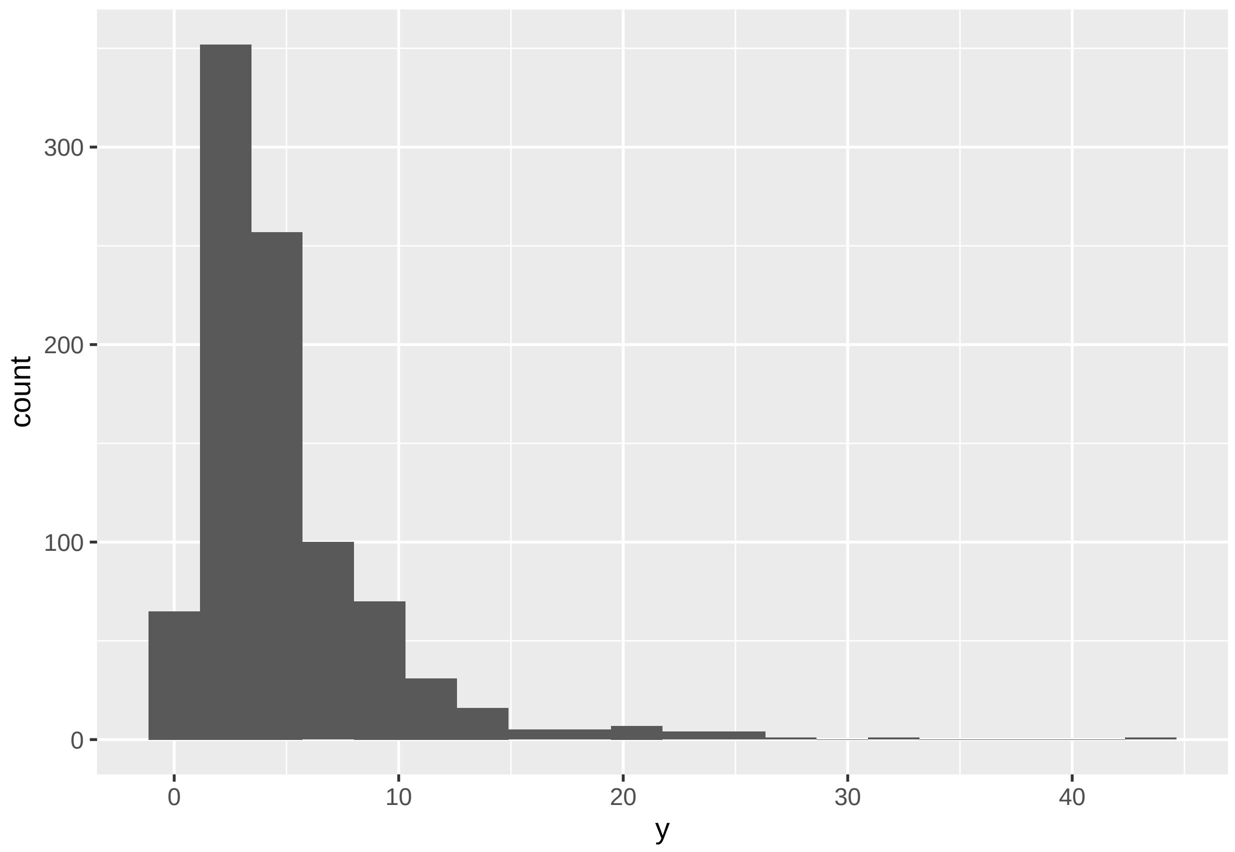
```r
library(viridis)
```

```
## Loading required package: viridisLite
```

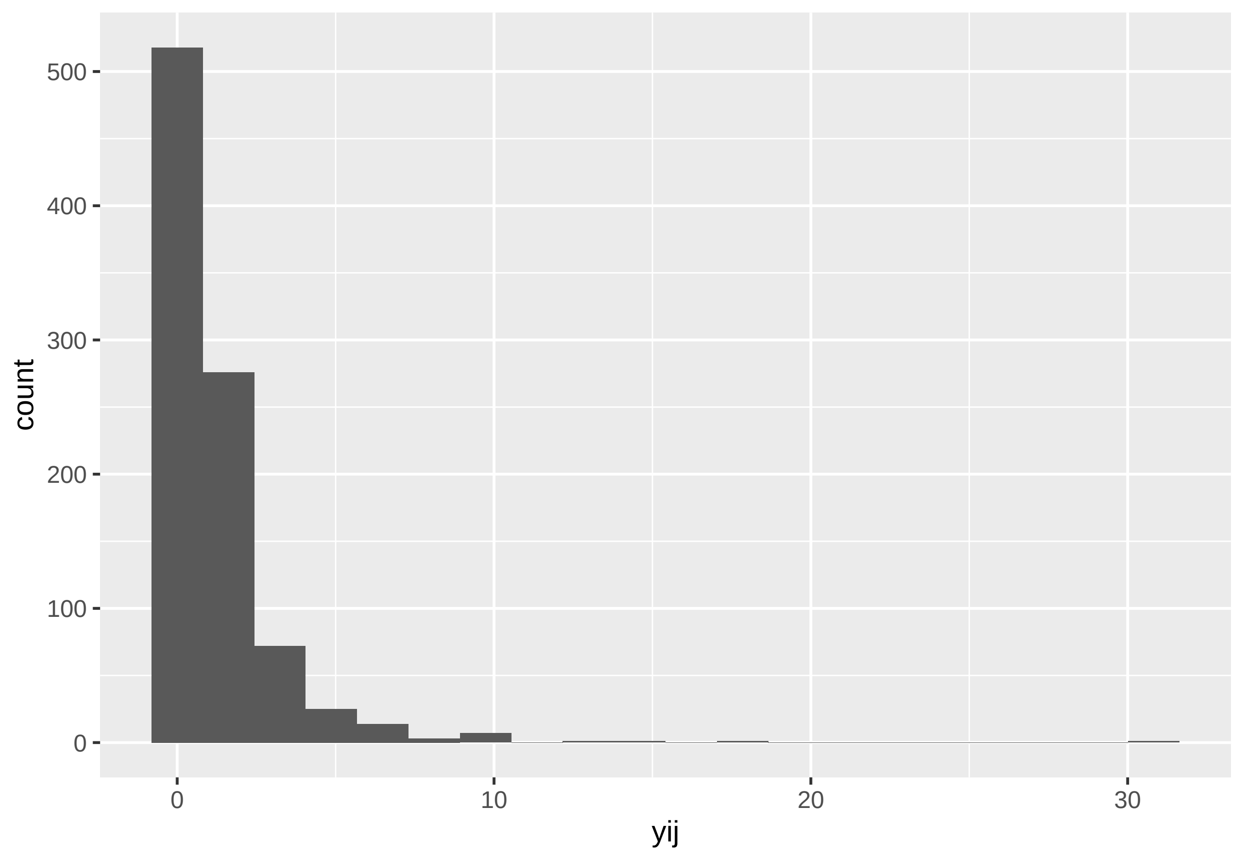
```r
set.seed(0922)
boot_samp <- radon %>% sample_n(size = 919, replace = TRUE)
J <- 85
N <- 919
u_j <- boot_samp %>% group_by(county_int) %>% summarise(mean(log_uranium))
tau <- mean(abs(rnorm(N, 0, sd = 1)))
gamma0_j = rnorm(levels(as.factor(boot_samp$county_int)), 0, sd = 2)
gamma1_j = rnorm(levels(as.factor(boot_samp$county_int)), 0, sd = 2)
u_j <- data.frame(u_j, gamma0_j, gamma1_j)
u_j <- u_j %>% mutate(alpha_j = rnorm(gamma0_j + gamma1_j * mean.log_uranium., sd = tau))
boot_samp <- left_join(boot_samp, u_j, by = "county_int")

dat <- tibble(x = boot_samp$floor,
              county = boot_samp$county_int,
              sigma = rep(mean(abs(rnorm(N, 0, sd = 1))), N),
              beta = rep(mean(rnorm(N, 0, sd = 2)), N),
              alpha_j = boot_samp$alpha_j,
              yij = exp(rnorm(N, mean = alpha_j + beta * x, sd = sigma)),
              y = exp(boot_samp$log_radon),
              level = beta * x + alpha_j
              )

dat %>% ggplot(aes(x = y)) + geom_histogram(bins = 20)
```



```r
dat %>%  ggplot(aes(x = yij)) + geom_histogram(bins = 20)
```



Q2

```
data {
  int<lower = 0> n;
  int<lower = 0> j;
  vector[n] y;
  vector[j] log_u;
  vector[n] x;
  int<lower = 1, upper = j> county[n];
}

parameters {
  real<lower = 0> tau;
  real<lower = 0> sigma;
  real beta;
  real gamma_0;
  real gamma_1;
  vector[j] z;
}

transformed parameters {
  vector[j] mu_j;
  vector[n] mu;
  mu_j = gamma_0 + gamma_1 * log_u + tau * z;

  for (i in 1:n) {
    mu[i] = x[i]*beta + mu_j[county[i]];
  }
}

model {
  y ~ normal(mu, sigma);
  tau ~ normal(0, 1);
  sigma ~ normal(0, 1);
  beta ~ normal(0, 2);
  gamma_0 ~ normal(0, 2);
  gamma_1 ~ normal(0, 2);
  z ~ normal(0, 1);
}
```

```
u <- radon %>% group_by(county_int) %>% summarise(mean_log_u = mean(log_uranium))
u
```

```
## # A tibble: 85 x 2
##    county_int mean_log_u
##         <int>      <dbl>
##  1          1    -0.689
##  2          2    -0.847
##  3          3    -0.113
##  4          4    -0.593
##  5          5    -0.143
##  6          6     0.387
##  7          7     0.272
##  8          8     0.278
##  9          9    -0.332
## 10         10     0.0959
## # ... with 75 more rows
```

```
data <- list(n = 919, j = 85, y = radon$log_radon, log_u = u$mean_log_u, x = radon$floor, county = rado
fit <- sampling(multilevel, data = data)
```

```
##
## SAMPLING FOR MODEL '248c5e10f7926fc38319d72975e4ba95' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 3.948 seconds (Warm-up)
## Chain 1:                1.749 seconds (Sampling)
## Chain 1:                5.697 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '248c5e10f7926fc38319d72975e4ba95' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 3.53 seconds (Warm-up)
## Chain 2:                1.847 seconds (Sampling)
## Chain 2:                5.377 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '248c5e10f7926fc38319d72975e4ba95' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3: 
## Chain 3:  Elapsed Time: 4.162 seconds (Warm-up)
## Chain 3:                1.857 seconds (Sampling)
## Chain 3:                6.019 seconds (Total)
## Chain 3: 
## 
## SAMPLING FOR MODEL '248c5e10f7926fc38319d72975e4ba95' NOW (CHAIN 4).
## Chain 4: 
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4: 
## Chain 4: 
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4: 
## Chain 4:  Elapsed Time: 4.014 seconds (Warm-up)
## Chain 4:                2.127 seconds (Sampling)
## Chain 4:                6.141 seconds (Total)
## Chain 4: 
```

Q3

```r
print(fit, pars = c("beta", "mu_j[84]", "mu_j[47]", "mu_j[2]", "mu_j[81]", "mu_j[37]"))
```

```
## Inference for Stan model: 248c5e10f7926fc38319d72975e4ba95.
## 4 chains, each with iter=2000; warmup=1000; thin=1; 
## post-warmup draws per chain=1000, total post-warmup draws=4000.
## 
##           mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## beta     -0.64       0 0.07 -0.77 -0.68 -0.64 -0.59 -0.50  6643    1
## mu_j[84]  1.51       0 0.12  1.28  1.42  1.50  1.58  1.77  4946    1
## mu_j[47]  1.31       0 0.15  1.00  1.22  1.32  1.41  1.62  7249    1
## mu_j[2]   0.92       0 0.09  0.74  0.85  0.91  0.98  1.09  5083    1
## mu_j[81]  1.74       0 0.16  1.46  1.63  1.72  1.84  2.10  4135    1
## mu_j[37]  0.86       0 0.15  0.56  0.77  0.87  0.97  1.13  4681    1
## 
## Samples were drawn using NUTS(diag_e) at Mon Mar 16 01:31:27 2020.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```r
radon_84 <- radon %>% filter(county_int == 84)
radon_47 <- radon %>% filter(county_int == 47)
radon_2 <- radon %>% filter(county_int == 2)
radon_81 <- radon %>% filter(county_int == 81)
radon_37 <- radon %>% filter(county_int == 37)

lm84 <- lm(log_radon ~ floor, data = radon_84)
summary(lm84)
```

```
##
## Call:
## lm(formula = log_radon ~ floor, data = radon_84)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.8865 -0.1489  0.0000  0.2420  1.0976
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.6750     0.1636  10.235 5.86e-07 ***
## floor        -0.7995     0.5900  -1.355    0.203
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5669 on 11 degrees of freedom
## Multiple R-squared:  0.143,  Adjusted R-squared:  0.06513
## F-statistic: 1.836 on 1 and 11 DF,  p-value: 0.2026
```

```r
lm47 <- lm(log_radon ~ floor, data = radon_47)
summary(lm47)
```

```
##
## Call:
## lm(formula = log_radon ~ floor, data = radon_47)
##
## Residuals:
## ALL 2 residuals are 0: no residual degrees of freedom!
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    1.758         NA      NA       NA
## floor         -2.269         NA      NA       NA
##
## Residual standard error: NaN on 0 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:    NaN
## F-statistic:   NaN on 1 and 0 DF,  p-value: NA
```

```r
lm2 <- lm(log_radon ~ floor, data = radon_2)
summary(lm2)
```

```
##
## Call:
## lm(formula = log_radon ~ floor, data = radon_2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.64629 -0.49927 -0.01724  0.55643  1.15099
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.95314    0.09684   9.842 2.74e-13 ***
## floor       -1.07974    0.40318  -2.678  0.00999 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6779 on 50 degrees of freedom
## Multiple R-squared:  0.1254, Adjusted R-squared:  0.108
## F-statistic: 7.172 on 1 and 50 DF,  p-value: 0.009993
```

```
lm81 <- lm(log_radon ~ floor, data = radon_81)
summary(lm81)
```

```
##
## Call:
## lm(formula = log_radon ~ floor, data = radon_81)
##
## Residuals:
##       1       2       3
##  0.0000  0.2871 -0.2871
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.6319     0.4061   6.481   0.0975 .
## floor        -0.5869     0.4973  -1.180   0.4475
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4061 on 1 degrees of freedom
## Multiple R-squared:  0.582,  Adjusted R-squared:  0.1641
## F-statistic: 1.392 on 1 and 1 DF,  p-value: 0.4475
```

```
lm37 <- lm(log_radon ~ floor, data = radon_37)
summary(lm37)
```

```
##
## Call:
## lm(formula = log_radon ~ floor, data = radon_37)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.9301 -0.4193  0.0000  0.1114  1.5548
##
## Coefficients:
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.41928    0.29160   1.438    0.194
## floor       -0.08281    0.87481  -0.095    0.927
##
## Residual standard error: 0.8248 on 7 degrees of freedom
## Multiple R-squared:  0.001278,   Adjusted R-squared:  -0.1414
## F-statistic: 0.00896 on 1 and 7 DF,  p-value: 0.9272
```

```
floor_0_post <- c(1.51, 1.31, 0.92, 1.75, 0.86)
floor_1_post <- floor_0_post - 0.64
floor_0_no_pooling <- c(1.68, 1.76, 0.95, 2.63, 0.42)
floor_1_no_pooling <- c(0.88, -0.51, -0.13 ,2.05, 0.34)
county <- c(84, 47, 2, 81, 37)
d <- data.frame(county, floor_0_no_pooling, floor_0_post, floor_1_no_pooling, floor_1_post)
knitr::kable(d)
```

county	floor_0_no_pooling	floor_0_post	floor_1_no_pooling	floor_1_post
84	1.68	1.51	0.88	0.87
47	1.76	1.31	-0.51	0.67
2	0.95	0.92	-0.13	0.28
81	2.63	1.75	2.05	1.11
37	0.42	0.86	0.34	0.22

```
sd_0_no_pooling <- sd(floor_0_no_pooling)
sd_1_no_pooling <- sd(floor_1_no_pooling)
sd_0_post <- sd(floor_0_post)
sd_1_post <- sd(floor_1_post)

sd_0_no_pooling
```

```
## [1] 0.8433682
```

```
sd_0_post
```

```
## [1] 0.3808543
```

```
sd_1_no_pooling
```

```
## [1] 0.9982635
```

```
sd_1_post
```

```
## [1] 0.3808543
```

The difference is caused because in no-pooling regression model, we assume that there is no connection at all between the log-radon levels of different floors in different counties, while in part 1 model, we assume there are still some connections between differnt counties. Hence, the estimates for different counties in part 1 model are generally closer to each other, as the standard deviations are lower for both floor levels.