Untitled

2023-04-08

```r
set.seed(1)
```

```r
library(rstan)
```

```
## Loading required package: StanHeaders
```

```
## Loading required package: ggplot2
```

```
## rstan (Version 2.21.8, GitRev: 2e1f913d3ca3)
```

```
## 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)
```

```r
library(Rlab)
```

```
## Rlab 4.0 attached.
```

```
##
## Attaching package: 'Rlab'
```

```
## The following objects are masked from 'package:stats':
##
##     dexp, dgamma, dweibull, pexp, pgamma, pweibull, qexp, qgamma,
##     qweibull, rexp, rgamma, rweibull
```

```
## The following object is masked from 'package:datasets':
##
##     precip
```

```r
train <- read.csv("train_new.csv")
test <- read.csv("test_new.csv")
#train
```

```r
data_train <- train[-c(1,4,11)]
data_test <- test[-c(1,3,10)]
```

```r
#data_train["SibSp"] = ceiling(data_train["SibSp"]/10)
#data_train["Parch"] = ceiling(data_train["Parch"]/10)
data_train["Sex"] = (data_train$Sex=="male")*1
#data_test["SibSp"] = ceiling(data_test["SibSp"]/10)
#data_test["Parch"] = ceiling(data_test["Parch"]/10)
data_test["Sex"] = (data_test$Sex=="male")*1
data_train["Age"] = (data_train$Age-mean(data_train$Age))/sqrt(var(data_train$Age))
data_test["Age"] = (data_test$Age-mean(data_test$Age))/sqrt(var(data_test$Age))
data_train["family"] = data_train$Parch+data_train$SibSp
data_test["family"] = data_test$Parch+data_test$SibSp
data_train["family"] = (data_train$family-mean(data_train$family))/sqrt(var(data_train$family))
data_test["family"] = (data_test$family-mean(data_test$family))/sqrt(var(data_test$family))
```

```r
data_stan <- list(
  N = 889,
  n = 417,
  vive_train = data_train$Survived,
  class1_train = (data_train$Pclass==1)*1,
  class1_test = (data_test$Pclass==1)*1,
  class2_train = (data_train$Pclass==2)*1,
  class2_test = (data_test$Pclass==2)*1,
  sex_train = data_train$Sex,
  sex_test = data_test$Sex,
  #young_train = (data_train$Age<10)*1,
  #young_test = (data_test$Age<10)*1,
  #old_train = (data_train$Age>60)*1,
  #old_test = (data_test$Age>60)*1
  age_train = data_train$Age,
  age_test = data_test$Age
  #family_train = data_train$family,
  #family_test = data_test$family
)
fit <- stan(file = "first.stan", data = data_stan, iter = 1000, chain = 4)
```

```
## Trying to compile a simple C file

## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -arch arm64 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Library/Framew
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/StanHeade
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen
## /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen/include/Eigen/src/Cor
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen/include/Eigen/src/Cor
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/StanHeade
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen
## /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen/include/Eigen/Core:96
```

```
## #include <complex>
##          ^~~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL 'first' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000218 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 2.18 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.82597 seconds (Warm-up)
## Chain 1:                0.873329 seconds (Sampling)
## Chain 1:                1.6993 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'first' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.000144 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 1.44 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 2: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 2: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 2: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 2: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 2: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 2: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 2: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.766714 seconds (Warm-up)
## Chain 2:                0.745872 seconds (Sampling)
## Chain 2:                1.51259 seconds (Total)
## Chain 2:
```

```
##
## SAMPLING FOR MODEL 'first' 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 / 1000 [  0%]  (Warmup)
## Chain 3: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 3: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 3: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 3: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 3: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 3: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 3: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 3: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 3: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 3: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 3: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.855654 seconds (Warm-up)
## Chain 3:                0.757776 seconds (Sampling)
## Chain 3:                1.61343 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'first' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0.000144 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 1.44 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 4: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 4: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 4: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 4: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 4: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 4: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 4: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 4: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 4: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 4: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 4: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.793997 seconds (Warm-up)
## Chain 4:                0.801365 seconds (Sampling)
## Chain 4:                1.59536 seconds (Total)
## Chain 4:
```

```r
#print(fit)
cor(data_train[,-c(1,8)])
```

```
##              Pclass         Sex         Age       SibSp       Parch       Fare
## Pclass  1.00000000  0.12774090 -0.36528727  0.08165562  0.01682449 -0.5481933
## Sex     0.12774090  1.00000000  0.07262202 -0.11634817 -0.24750798 -0.1799575
## Age    -0.36528727  0.07262202  1.00000000 -0.25298934 -0.16628774  0.1082175
## SibSp   0.08165562 -0.11634817 -0.25298934  1.00000000  0.41454164  0.1608869
## Parch   0.01682449 -0.24750798 -0.16628774  0.41454164  1.00000000  0.2175320
## Fare   -0.54819329 -0.17995753  0.10821751  0.16088685  0.21753204  1.0000000
## family  0.06422053 -0.20319145 -0.25600999  0.89065367  0.78298776  0.2186582
##              family
## Pclass  0.06422053
## Sex    -0.20319145
## Age    -0.25600999
## SibSp   0.89065367
## Parch   0.78298776
## Fare    0.21865817
## family  1.00000000
```

```
#plot(fit, pars = c("pred"))
fit_ss <- extract(fit, pars = "pred", permuted = TRUE)$pred
```

```
pred_stan <- 1:417
for(i in 1:417) {
  #pred_stan[i] <- mean(rbern(1,fit_ss))
  pred_stan[i] <- mean(fit_ss[,i])
  pred_stan[i] <- (pred_stan[i] > 0.5) * 1
  #pred_stan[i] <- rbern(1,pred_stan[i])
}
```

```
freit <- glm(Survived ~ Pclass+Sex+Age,family=binomial(link='logit'),data=data_train)
summary(freit)
```

```
##
## Call:
## glm(formula = Survived ~ Pclass + Sex + Age, family = binomial(link = "logit"),
##     data = data_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6523  -0.6482  -0.4363   0.6269   2.4471
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.77737    0.32699  11.552  < 2e-16 ***
## Pclass      -1.18690    0.12077  -9.828  < 2e-16 ***
## Sex         -2.61053    0.18663 -13.988  < 2e-16 ***
## Age         -0.44723    0.09714  -4.604 4.14e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1182.82  on 888  degrees of freedom
## Residual deviance:  804.34  on 885  degrees of freedom
```

```
## AIC: 812.34
##
## Number of Fisher Scoring iterations: 5
```

```r
pred <- predict(freit, data_test, type="response")
pred <- (pred > 0.5)*1
```

```r
result <- read.csv("submission.csv")
#result
```

```r
real <- c(result[1:152,]$Survived,result[154:418,]$Survived)
mean(pred == real)
```

```
## [1] 0.7673861
```

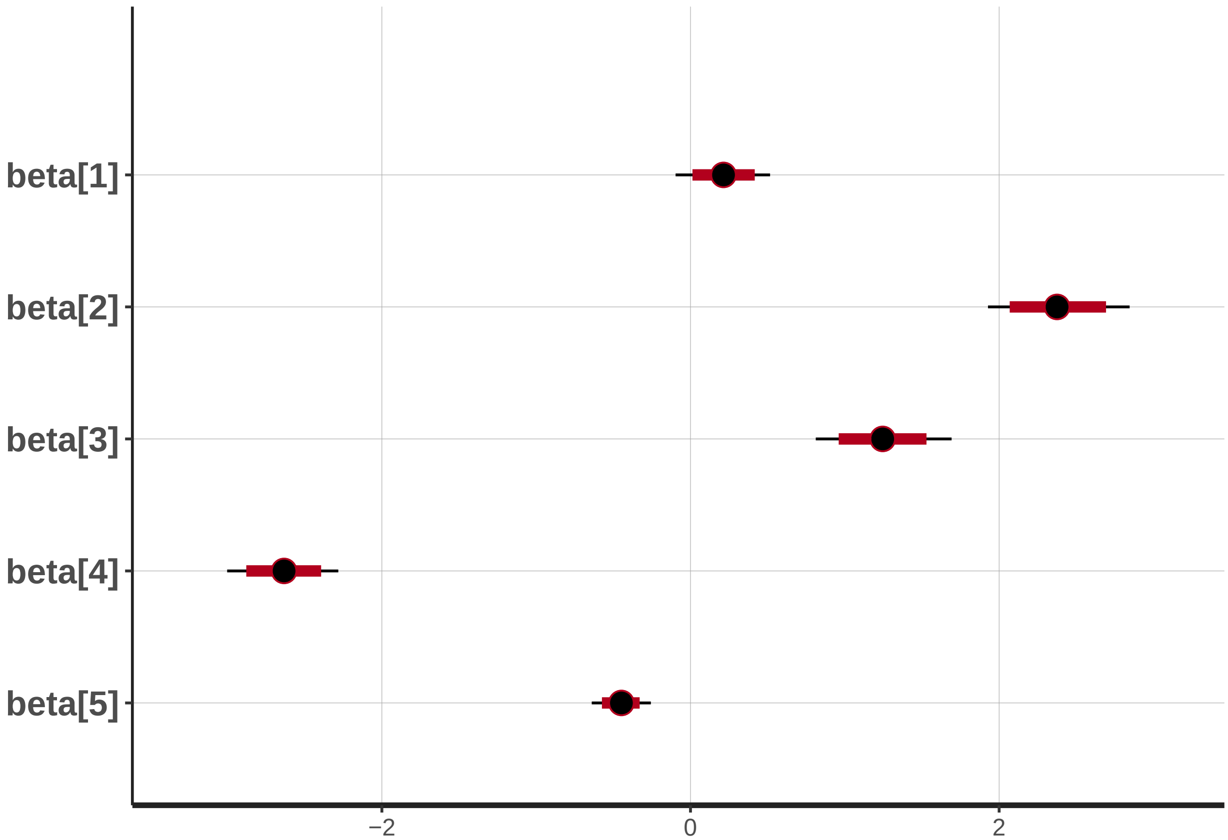
```r
mean(pred_stan == real)
```

```
## [1] 0.7697842
```

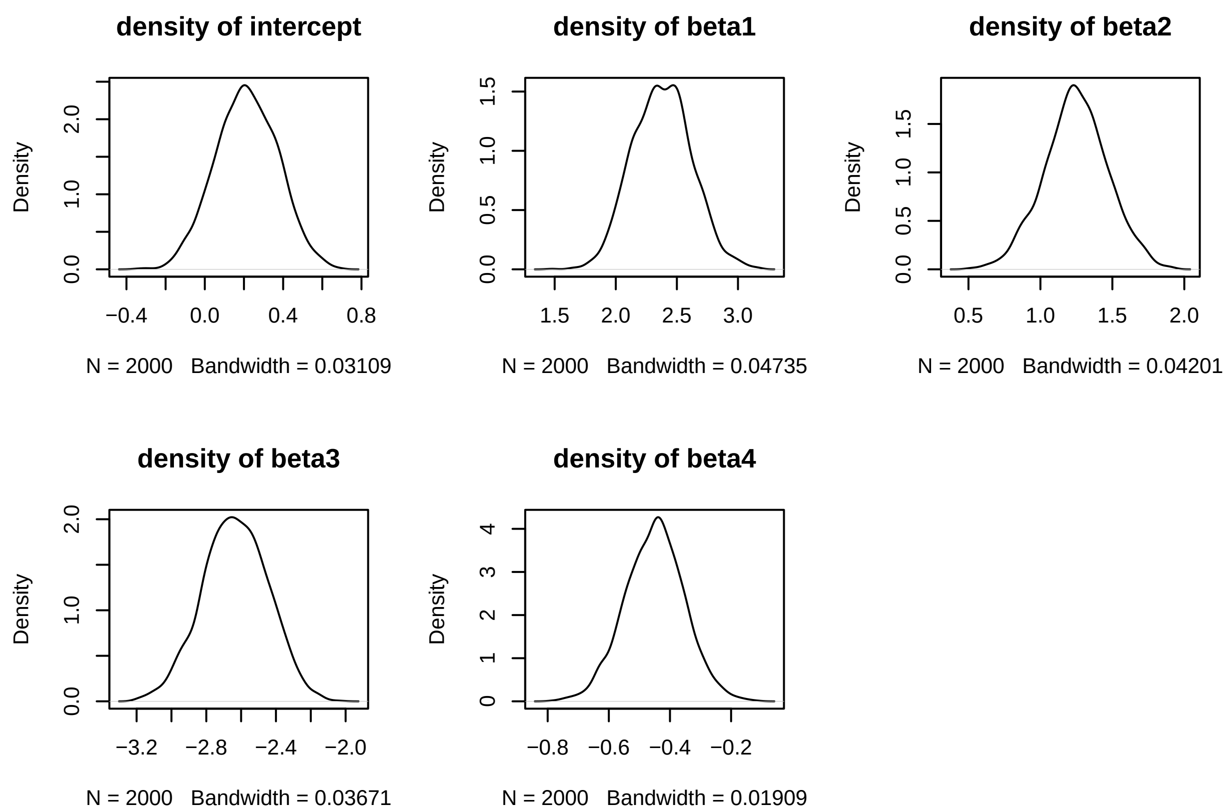
```r
plot(fit, pars = "beta", ylab = "bbbbb", main = "gggg")
```

```
## ci_level: 0.8 (80% intervals)
```

```
## outer_level: 0.95 (95% intervals)
```



```r
betas <- extract(fit, pars = "beta", permuted = TRUE)$beta
par(mfrow=c(2,3))
plot(density(betas[,1]), main="density of intercept")
plot(density(betas[,2]), main="density of beta1")
plot(density(betas[,3]), main="density of beta2")
plot(density(betas[,4]), main="density of beta3")
plot(density(betas[,5]), main="density of beta4")
#plot(density(betas[,6]))
```



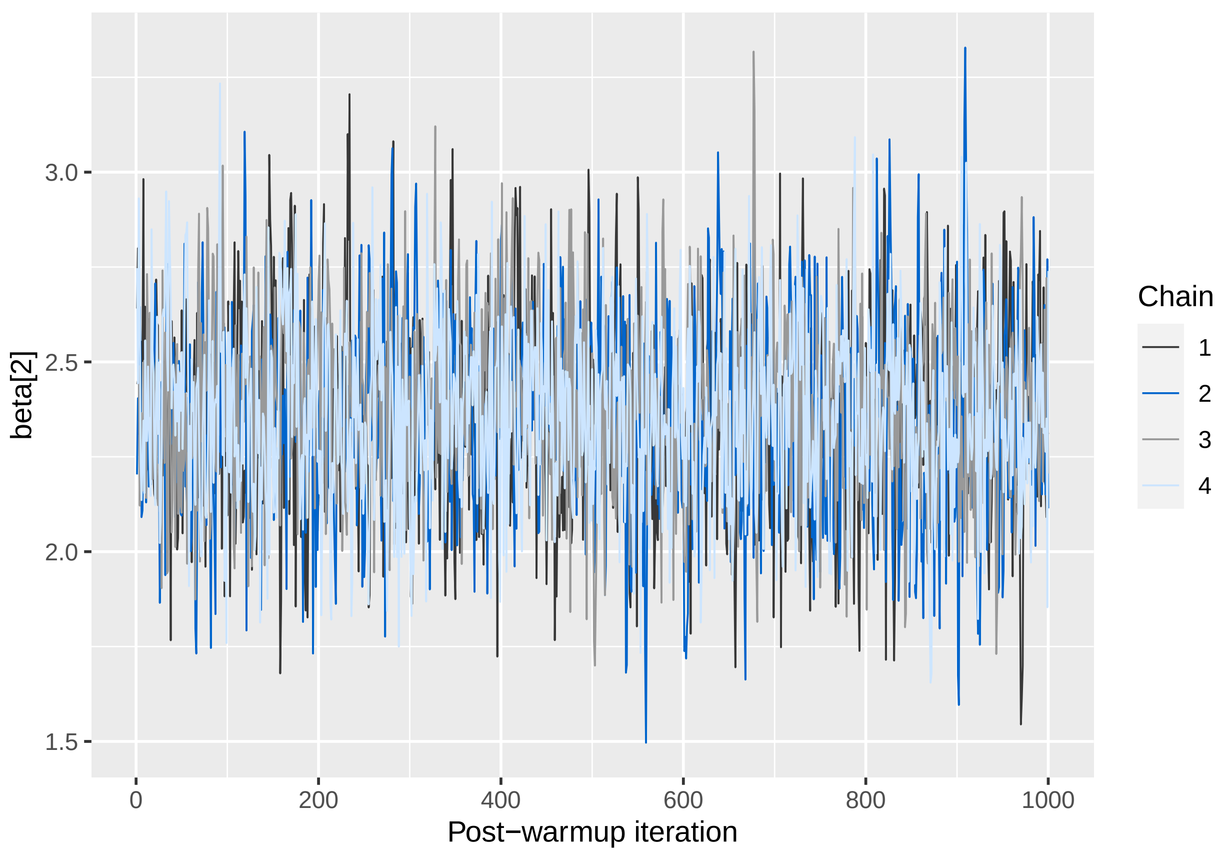

```r
library("bayesplot")
```

```
## 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
```

```r
library("ggplot2")
library("rstan")
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
```

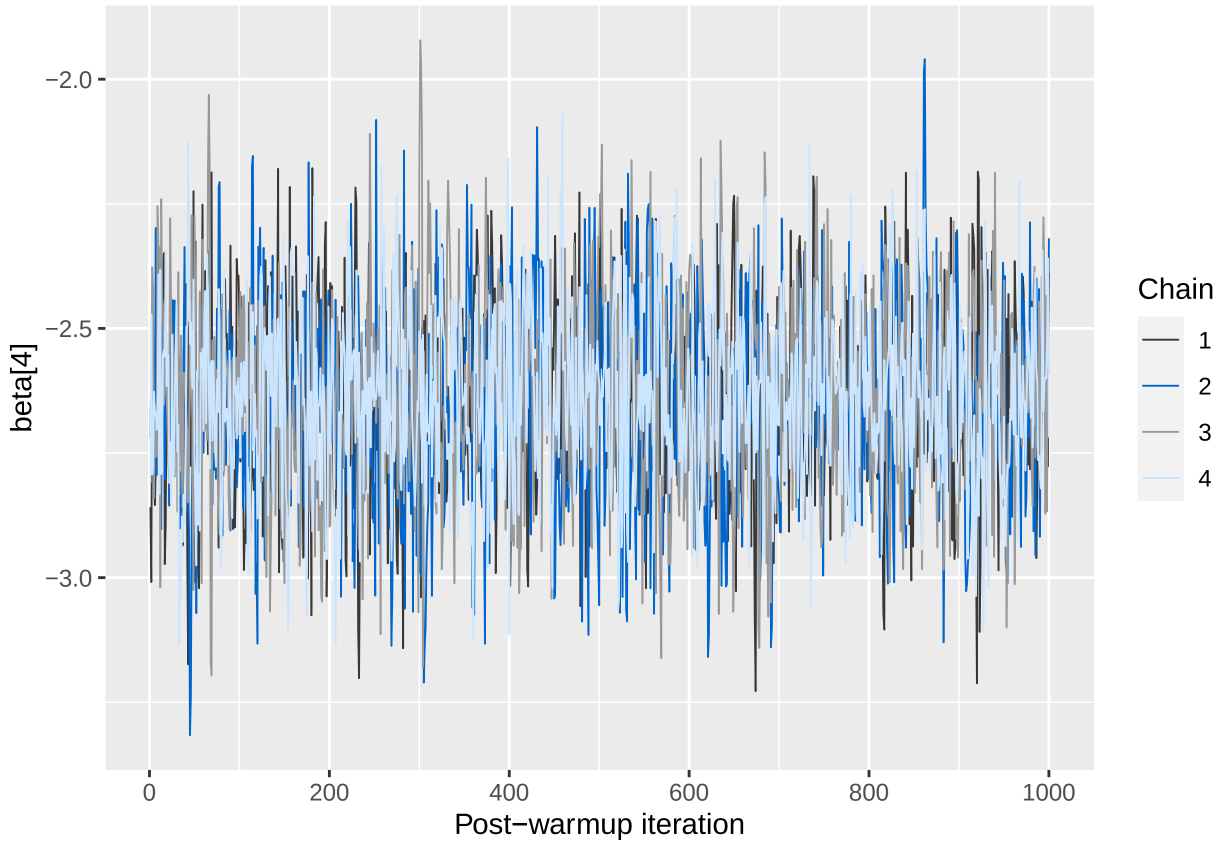
```r
schools_mod_cp <- stan_model("first.stan")
fit_cp <- sampling(schools_mod_cp, data = data_stan, seed = 1, control = list(adapt_delta = 0.9))
# Extract posterior draws for later use
posterior_cp <- as.array(fit_cp)
np_cp <- nuts_params(fit_cp)
color_scheme_set("mix-brightblue-gray")
mcmc_trace(posterior_cp, pars = "beta[2]", np = np_cp) +
  xlab("Post-warmup iteration")
```

```
## No divergences to plot.
```




```r
mcmc_trace(posterior_cp, pars = "beta[4]", np = np_cp) +
  xlab("Post-warmup iteration")
```

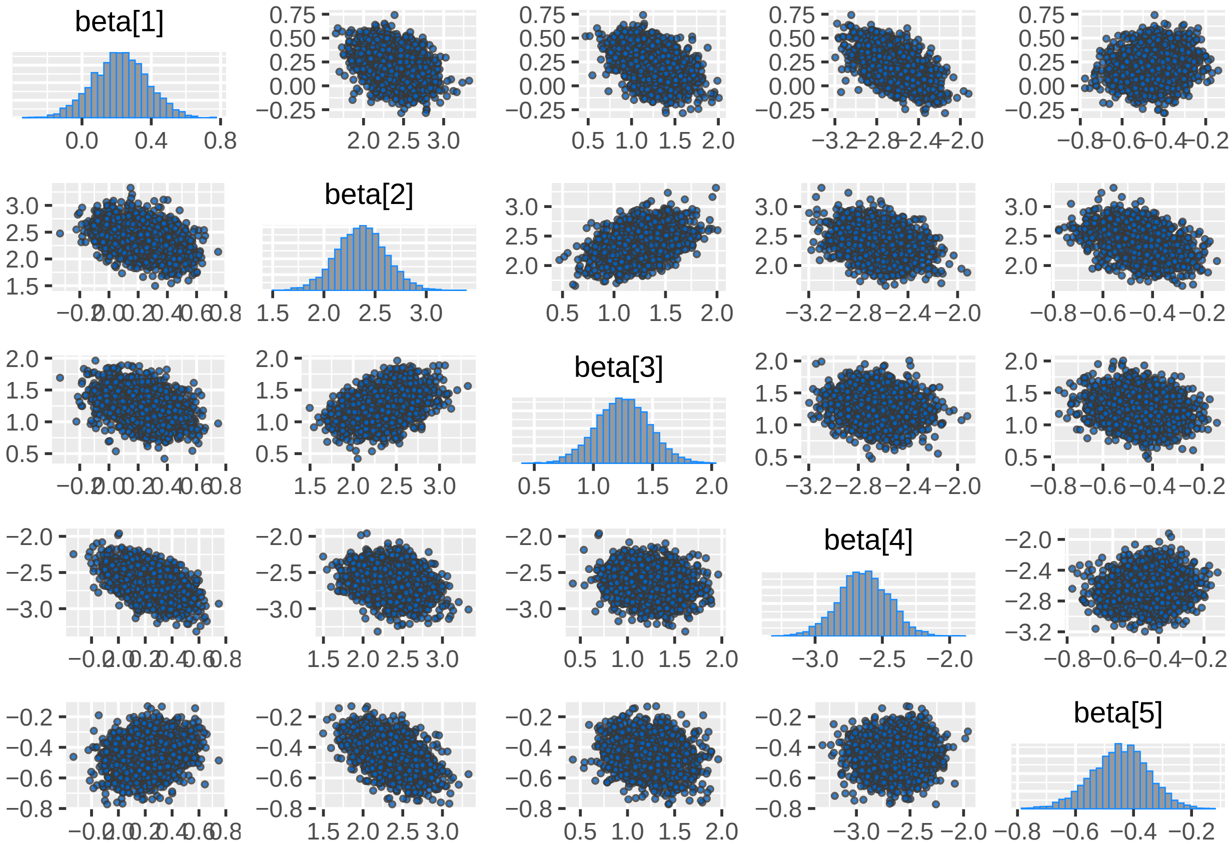
```
## No divergences to plot.
```




```r
available_mcmc(pattern = "_nuts_")
```

```
## bayesplot MCMC module:
## (matching pattern '_nuts_')
##   mcmc_nuts_acceptance
##   mcmc_nuts_divergence
##   mcmc_nuts_energy
##   mcmc_nuts_stepsize
##   mcmc_nuts_treedepth
```

```r
lp_cp <- log_posterior(fit_cp)

mcmc_pairs(posterior_cp, np = np_cp,
           pars = c("beta[1]", "beta[2]","beta[3]","beta[4]","beta[5]"),
           off_diag_args = list(size = 0.75))
```


beta[1]
beta[2]
beta[3]
beta[4]
beta[5]