Bayesian Methods - Assignment 8

Ryan Durfey

May 29, 2016

1. Revisit comparison of the two groups in section 1 of the workshop

```
library(rjags)
```

```
## Loading required package: coda
```

```
## Linked to JAGS 4.2.0
```

```
## Loaded modules: basemod,bugs
```

```
library(runjags)
library(shinystan)
```

```
## Loading required package: shiny
```

```
##
## This is shinystan version 2.1.0
```

```
library(rstan)
```

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

```
## rstan (Version 2.9.0-3, packaged: 2016-02-11 15:54:41 UTC, GitRev: 05c3d0058b6a)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## rstan_options(auto_write = TRUE)
## options(mc.cores = parallel::detectCores())
```

```
##
## Attaching package: 'rstan'
```

```
## The following object is masked from 'package:runjags':
##
##     extract
```

```
## The following object is masked from 'package:coda':
##
##     traceplot
```

```
source("//Users/rdurfey/R_misc/BayesianMethods/DBDA2Eprograms/DBDA2E-utilities.R")
```

```
##
## ***********************************************************************
## Kruschke, J. K. (2015). Doing Bayesian Data Analysis, Second Edition:
## A Tutorial with R, JAGS, and Stan. Academic Press / Elsevier.
## ***********************************************************************
```

```
myDataFrame = read.csv("//Users/rdurfey/R_misc/BayesianMethods/DBDA2Eprograms/TwoGroupIQ.csv")
y = as.numeric(myDataFrame[,"Score"])
x = as.numeric(as.factor(myDataFrame[,"Group"]))
xLevels = levels(as.factor(myDataFrame[,"Group"]))

Ntotal = length(y)
```

```
# Specify the data in a list, for later shipment to JAGS:
dataList = list(
    y = y ,
    x = x ,
    Ntotal = Ntotal ,
    meanY = mean(y) ,
    sdY = sd(y)
)

# model string to pass to stan
modelString = "
data {
    int<lower=1> Ntotal;
    int x[Ntotal];          //Group variable
    real y[Ntotal];
    real meanY;
    real sdY;
}
transformed data {
    real unifLo;
    real unifHi;
    real normalSigma;
    real expLambda;         //Parameter of prior for nu
    unifLo <- sdY/100;
    unifHi <- sdY*100;
    normalSigma <- sdY*100;
    expLambda<-1/30.0;      //Setting value for expLambda
}
parameters {
    real<lower=0> nu;
    real mu[2];                 //Making 2 groups
    real<lower=0> sigma[2];     //Making 2 groups
}
model {
    sigma ~ uniform(unifLo, unifHi);        //Recall that sigma is a vector of 2 numbers
    mu ~ normal(meanY, normalSigma);        //Recall that mu is a vector of 2 numbers
    nu~exponential(expLambda);      //Exponential prior for nu
    for (i in 1:Ntotal){
        y[i] ~ student_t(nu, mu[x[i]], sigma[x[i]]);          //Student_t distribution for y with nest
    }

}
"


# stan DSO
stanDsoRobust <- stan_model( model_code=modelString )
```

```
# run the MCMC
parameters = c( "mu" , "sigma" , "nu" )     # The parameters to be monitored
burnInSteps = 1000
nChains = 4
thinSteps = 1
numSavedSteps<-5000
```

```
# Get MC sample of posterior:
stanFitRobust <- sampling( object=stanDsoRobust ,
                   data = dataList ,
                   pars = parameters , # optional
                   chains = nChains ,
                   cores=nChains,
                   iter = ( ceiling(numSavedSteps/nChains)*thinSteps
                            +burnInSteps ) ,
                   warmup = burnInSteps ,
                   init = "random" , # optional
                   thin = thinSteps )
```

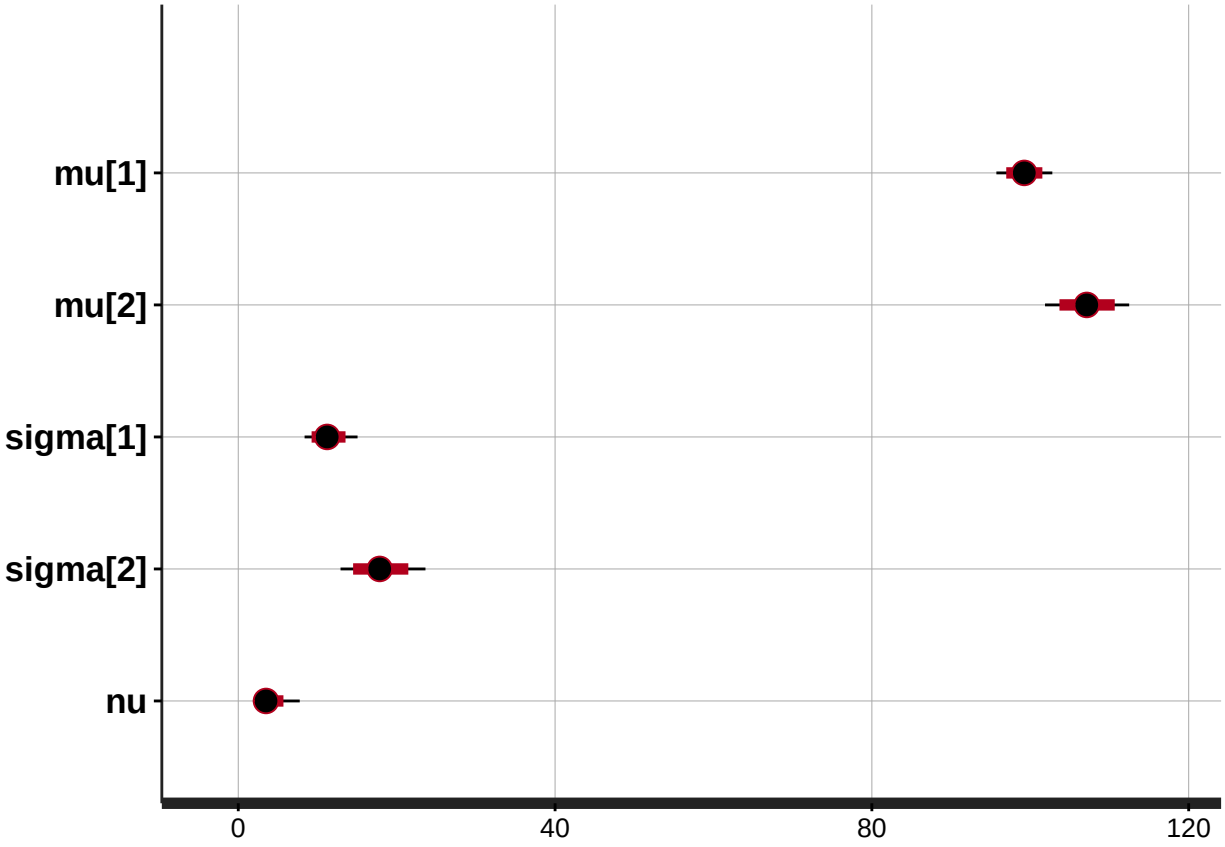
```
print (stanFitRobust)
```

```
## Inference for Stan model: 2e669b24c5def8646456a1ba92b32415.
## 4 chains, each with iter=2250; warmup=1000; thin=1;
## post-warmup draws per chain=1250, total post-warmup draws=5000.
##
##             mean se_mean   sd    2.5%     25%     50%     75%   97.5%
## mu[1]      99.26    0.03 1.78   95.72   98.07   99.24  100.47  102.78
## mu[2]     107.15    0.05 2.70  101.86  105.34  107.13  108.97  112.49
## sigma[1]   11.32    0.03 1.70    8.38   10.11   11.23   12.35   15.06
## sigma[2]   17.94    0.05 2.73   12.91   16.03   17.86   19.71   23.64
## nu          3.83    0.03 1.52    1.96    2.82    3.50    4.45    7.76
## lp__     -451.02    0.04 1.56 -454.85 -451.89 -450.69 -449.85 -448.89
##          n_eff Rhat
## mu[1]     3082    1
## mu[2]     2863    1
## sigma[1]  2690    1
## sigma[2]  2576    1
## nu        2270    1
## lp__      1776    1
##
## Samples were drawn using NUTS(diag_e) at Mon Jun  6 19:17:26 2016.
## 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).
```

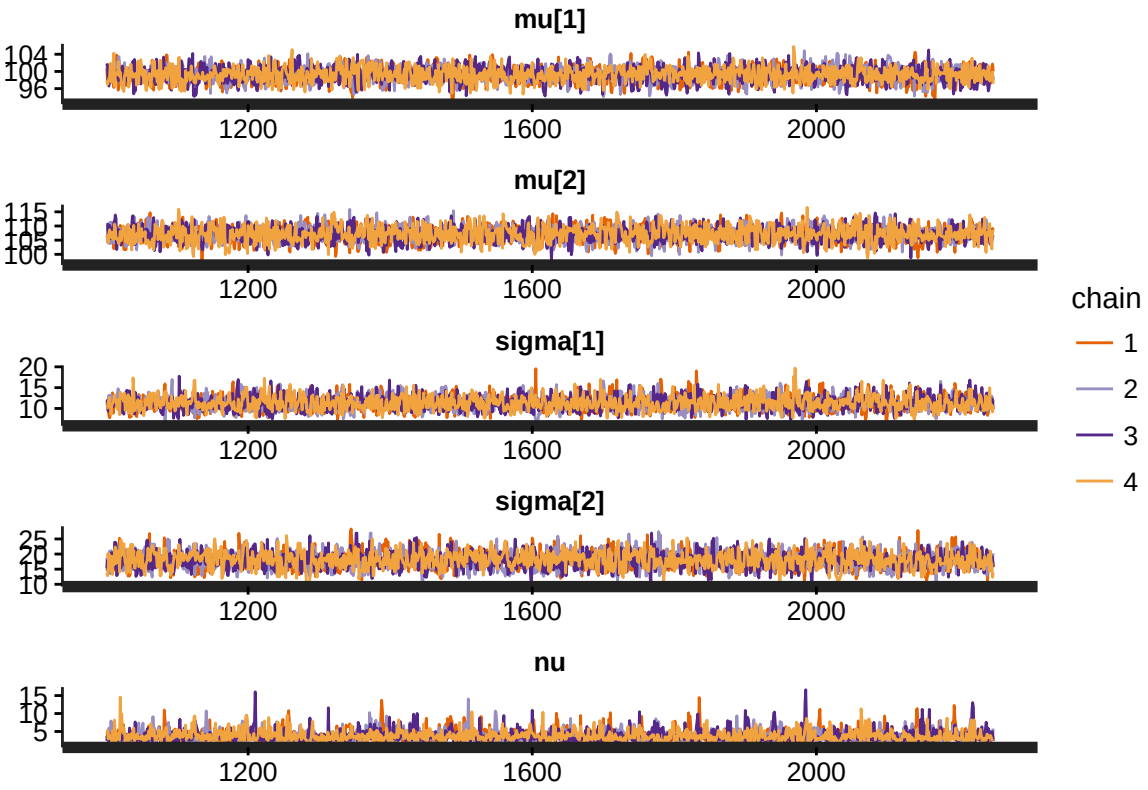
```
plot(stanFitRobust)
```

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

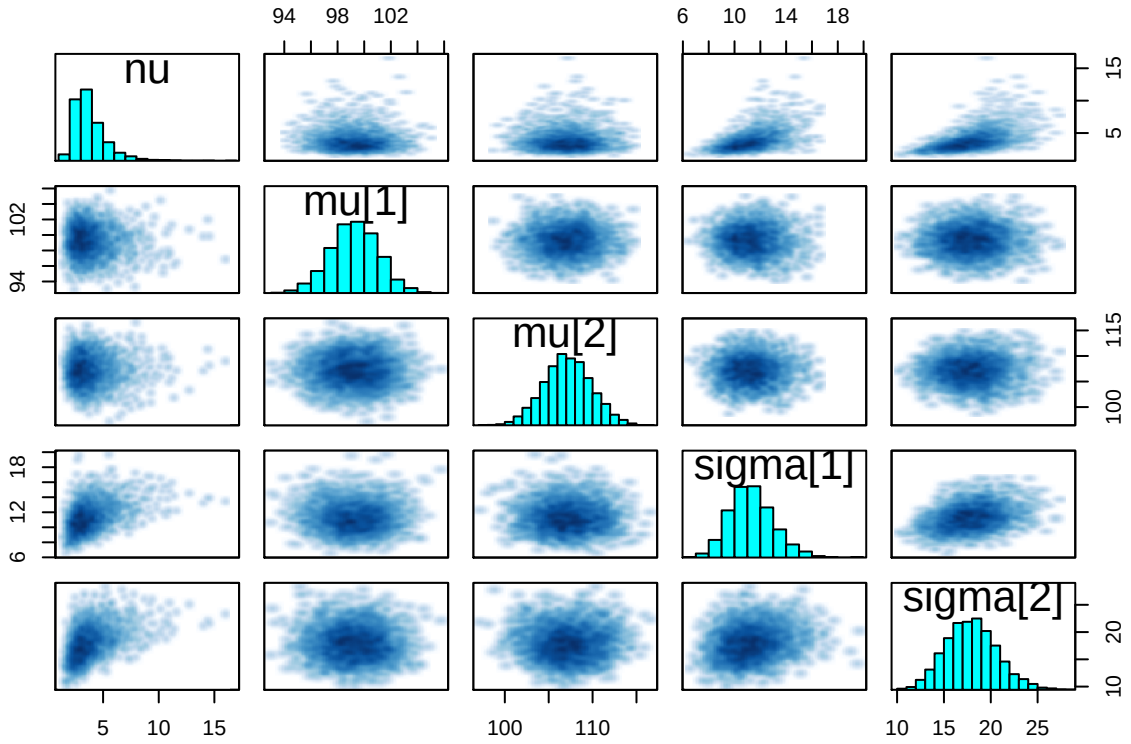
```
## outer_level: 0.95 (95% intervals)
```



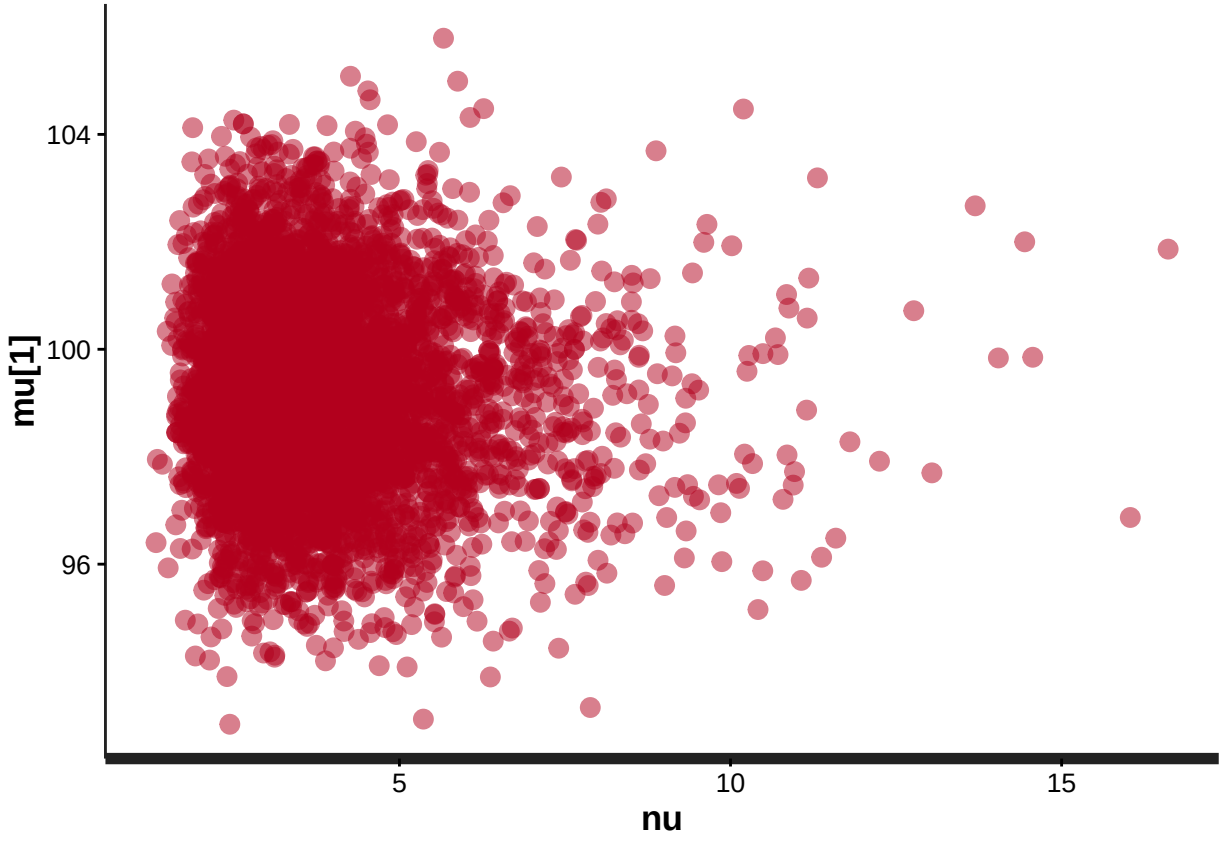

```
rstan::traceplot(stanFitRobust, ncol=1, inc_warmup=F)
```



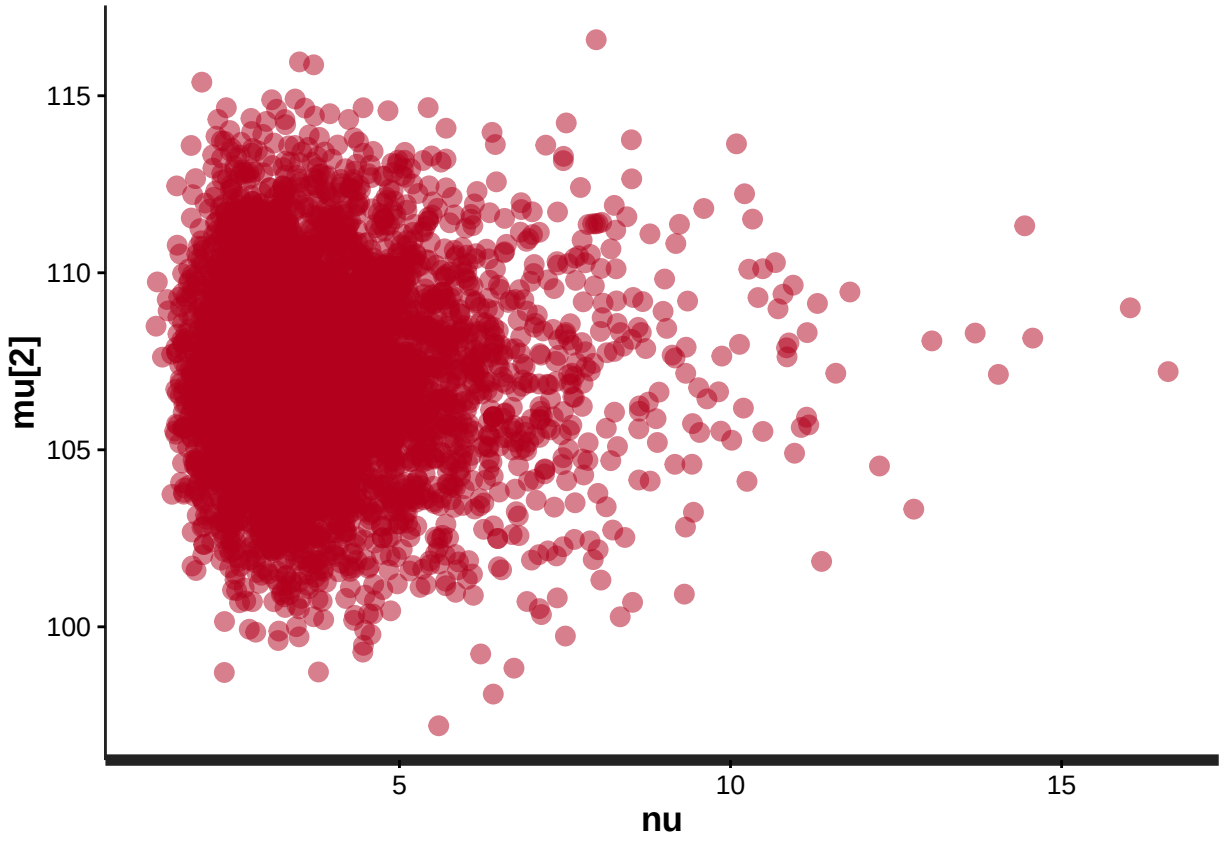
```
pairs(stanFitRobust, pars=c('nu','mu','sigma'))
```



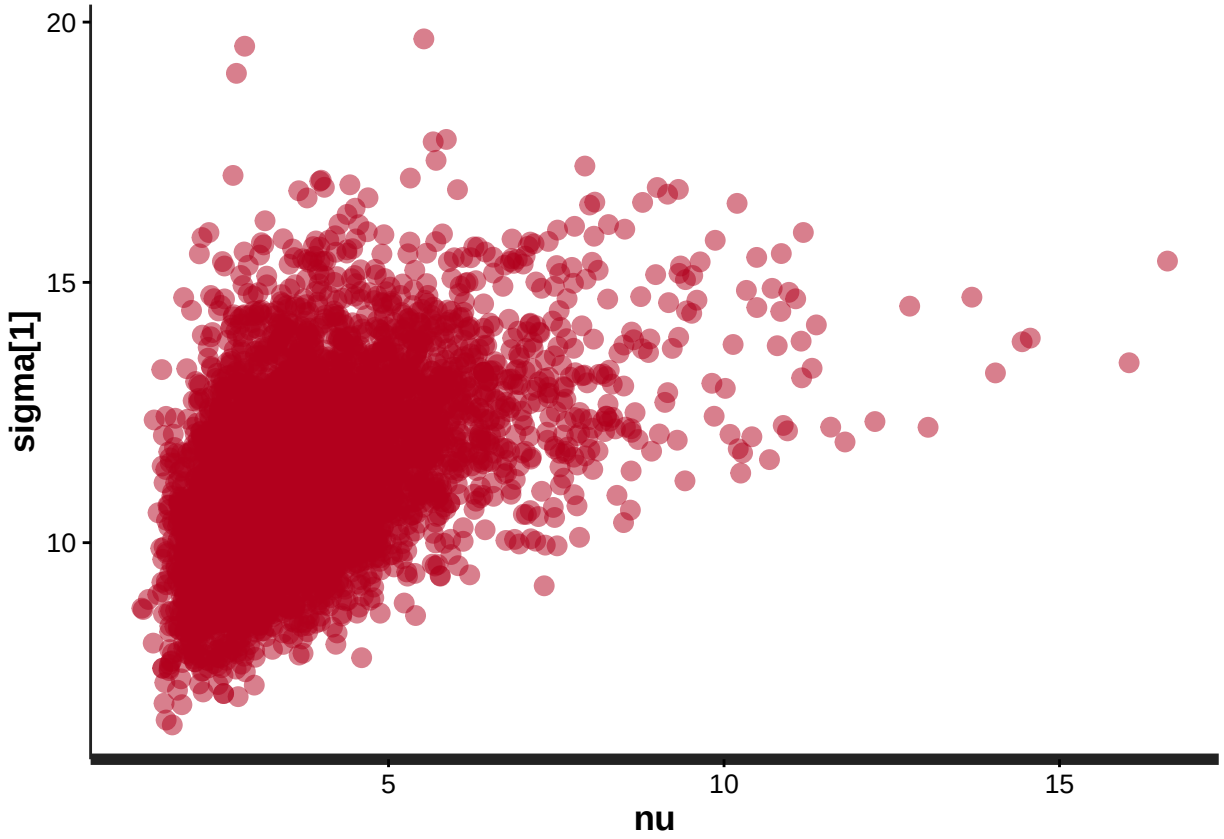

```
stan_scat(stanFitRobust, c('nu','mu[1]'))
```



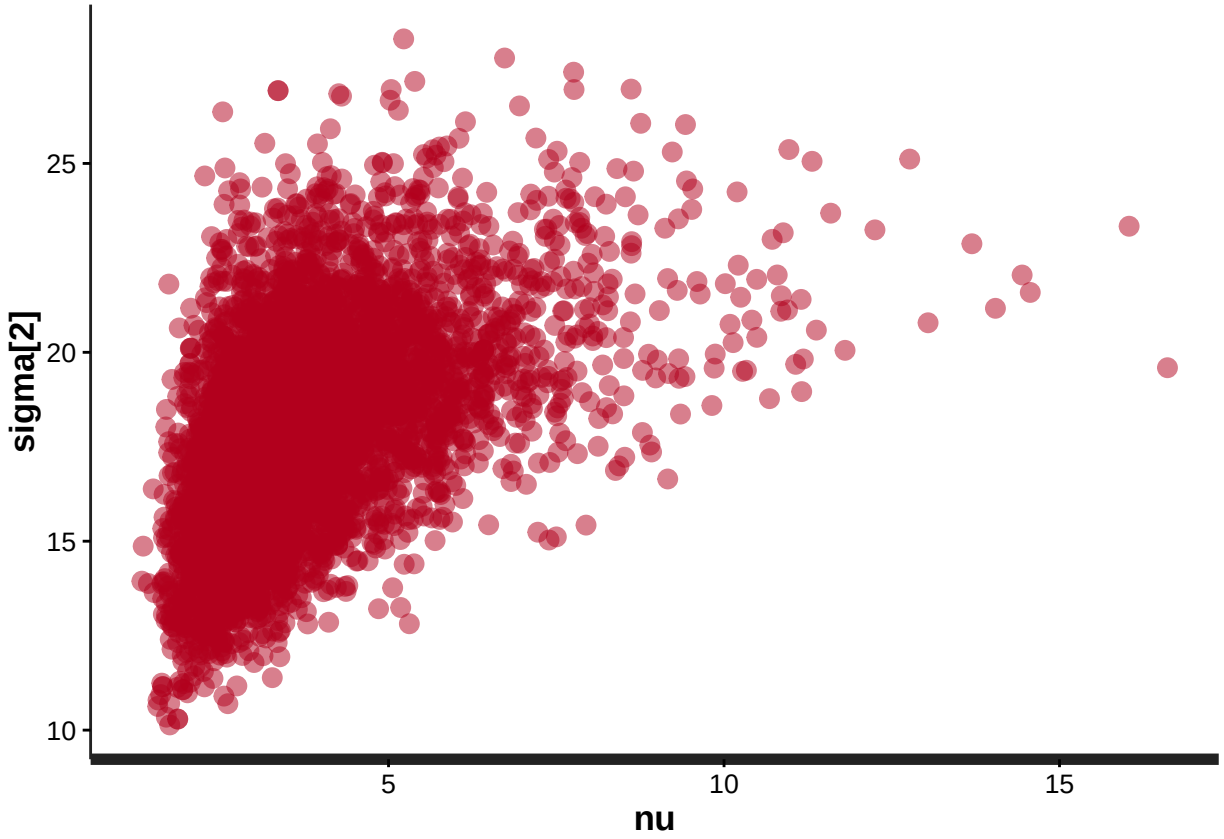
```
stan_scat(stanFitRobust, c('nu','mu[2]'))
```



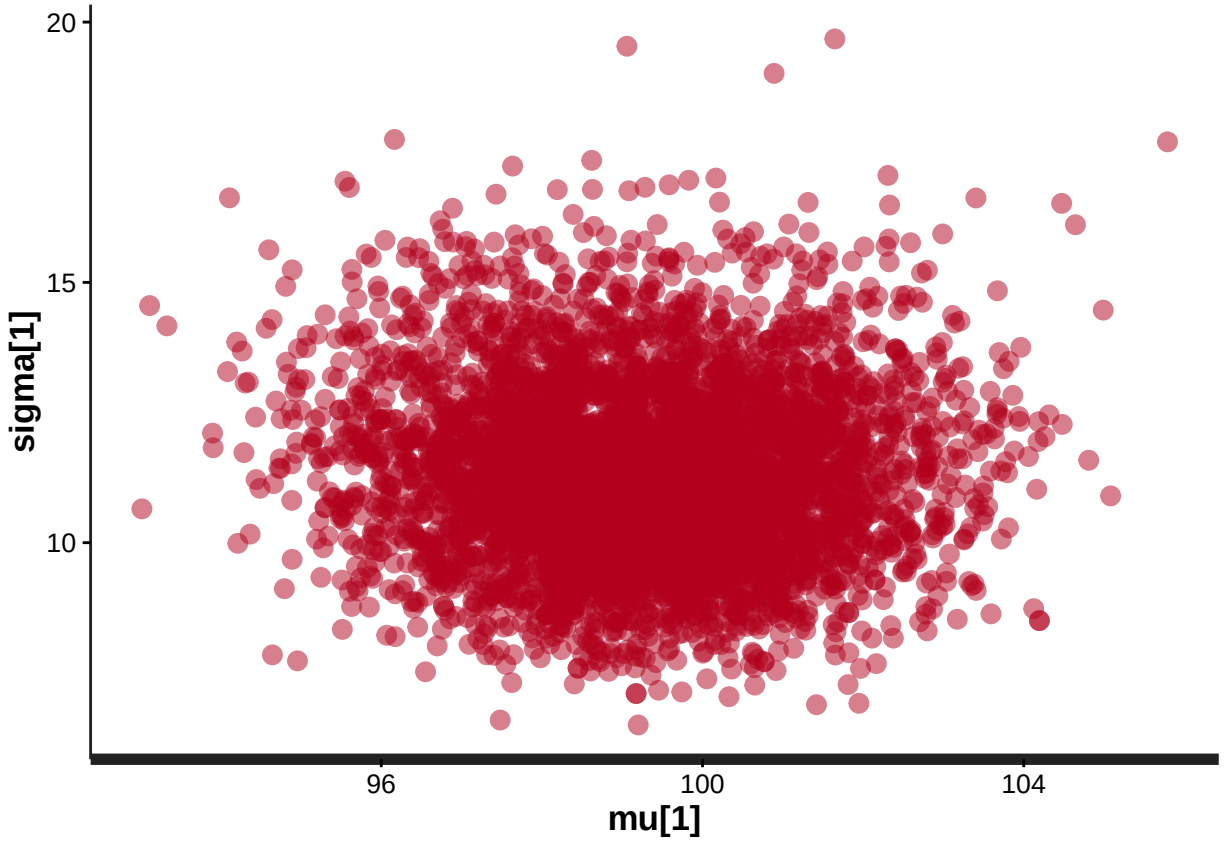

```
stan_scat(stanFitRobust, c('nu','sigma[1]'))
```



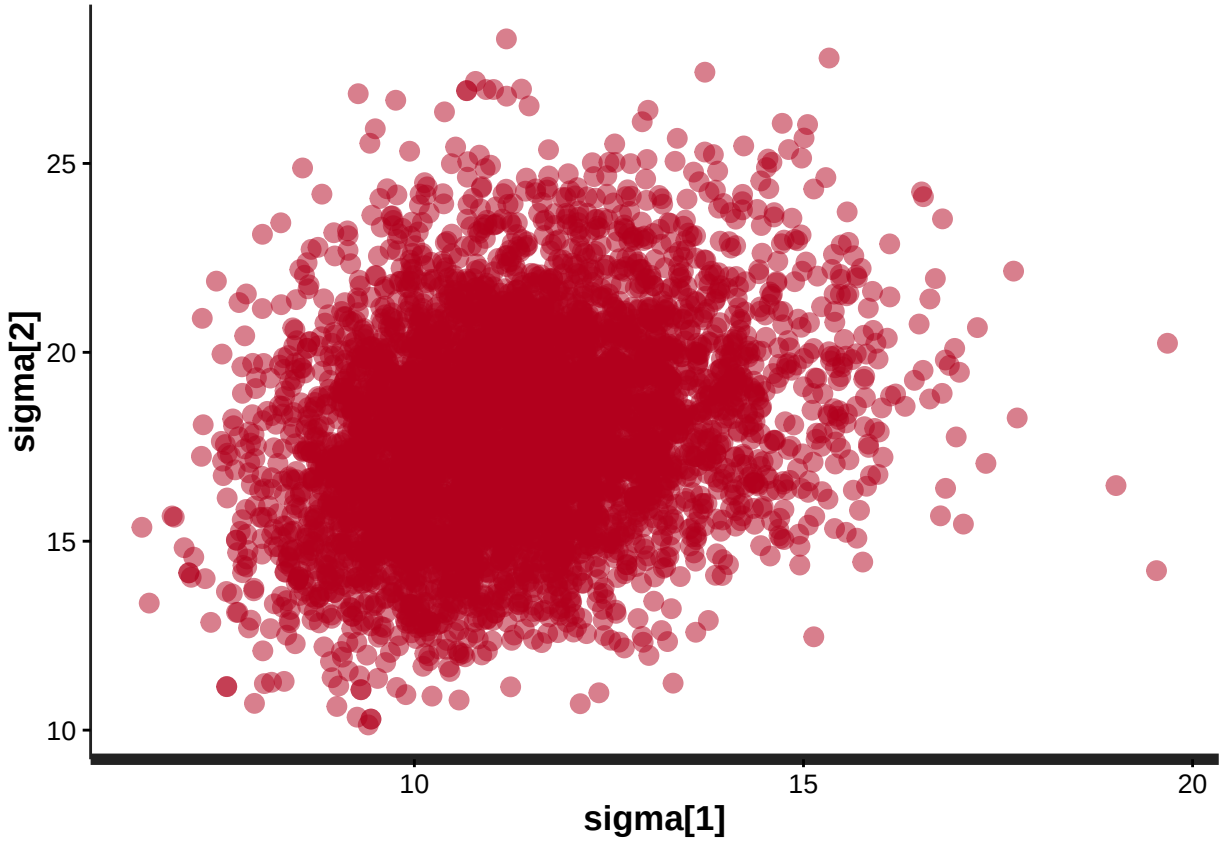

```
stan_scat(stanFitRobust, c('nu','sigma[2]'))
```




```
stan_scat(stanFitRobust, c('mu[1]','sigma[1]'))
```

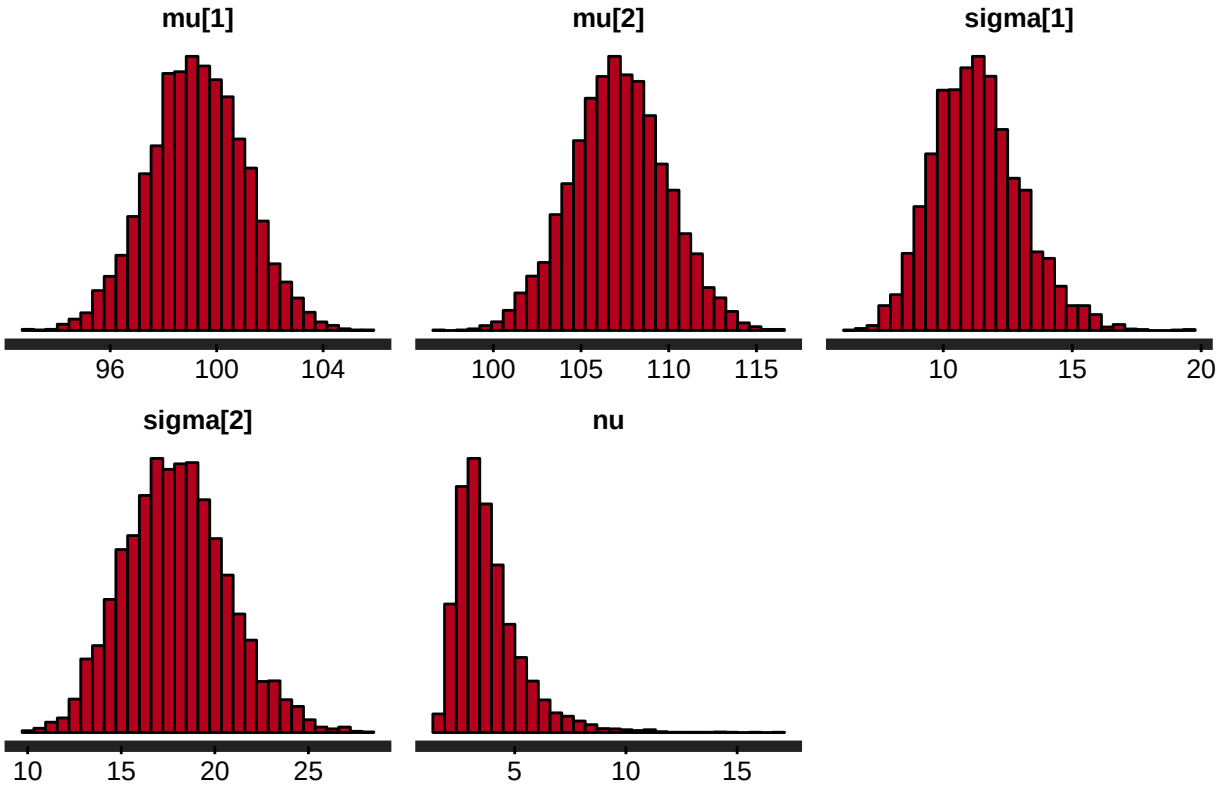



```
stan_scat(stanFitRobust, c('sigma[1]','sigma[2]'))
```

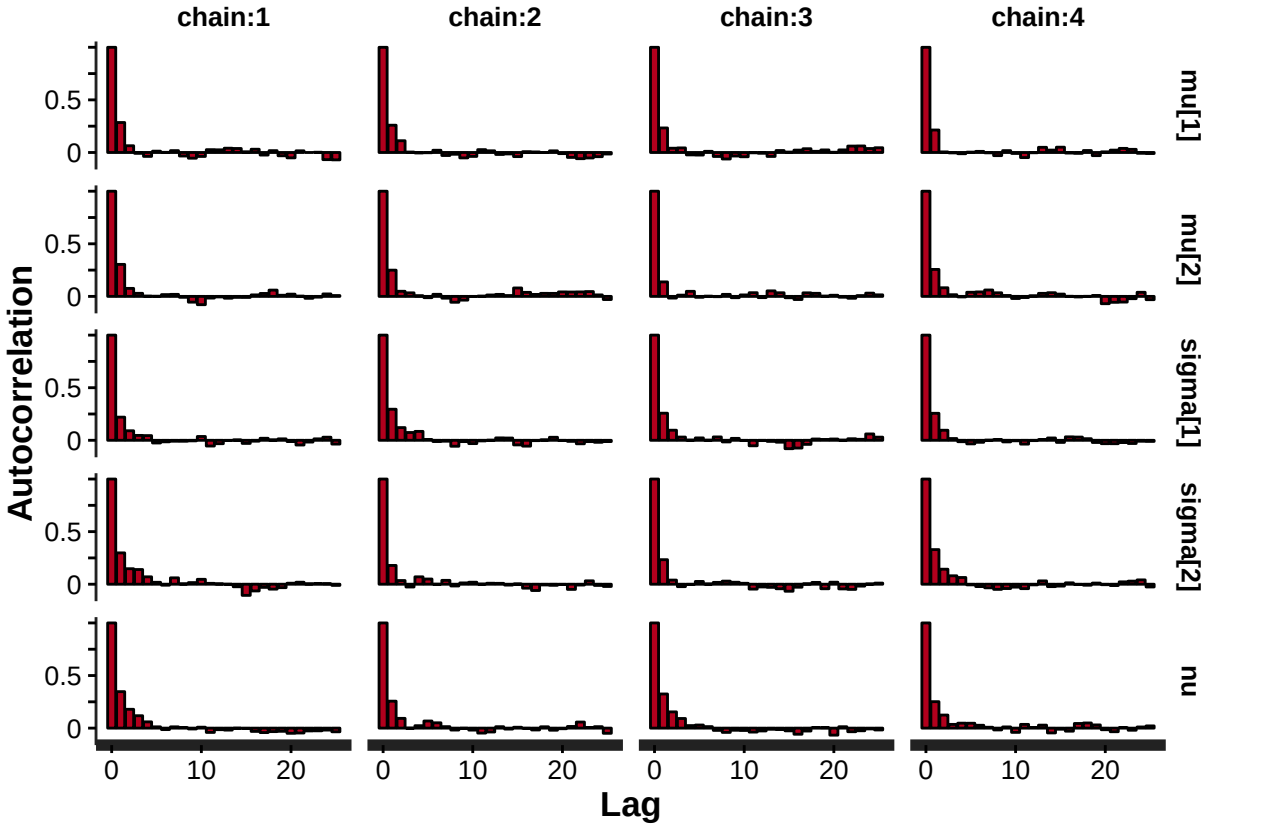



```
stan_hist(stanFitRobust)
```

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

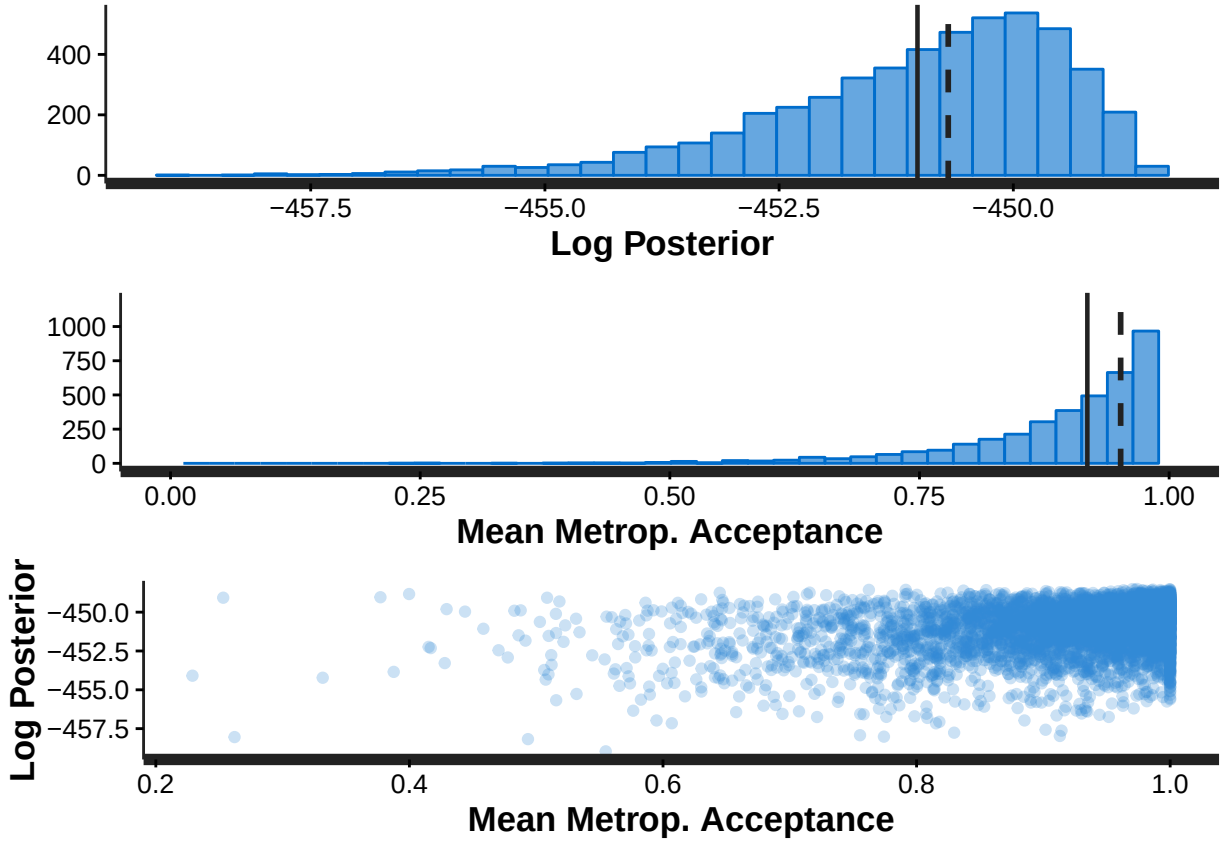



```
stan_ac(stanFitRobust, separate_chains = T)
```

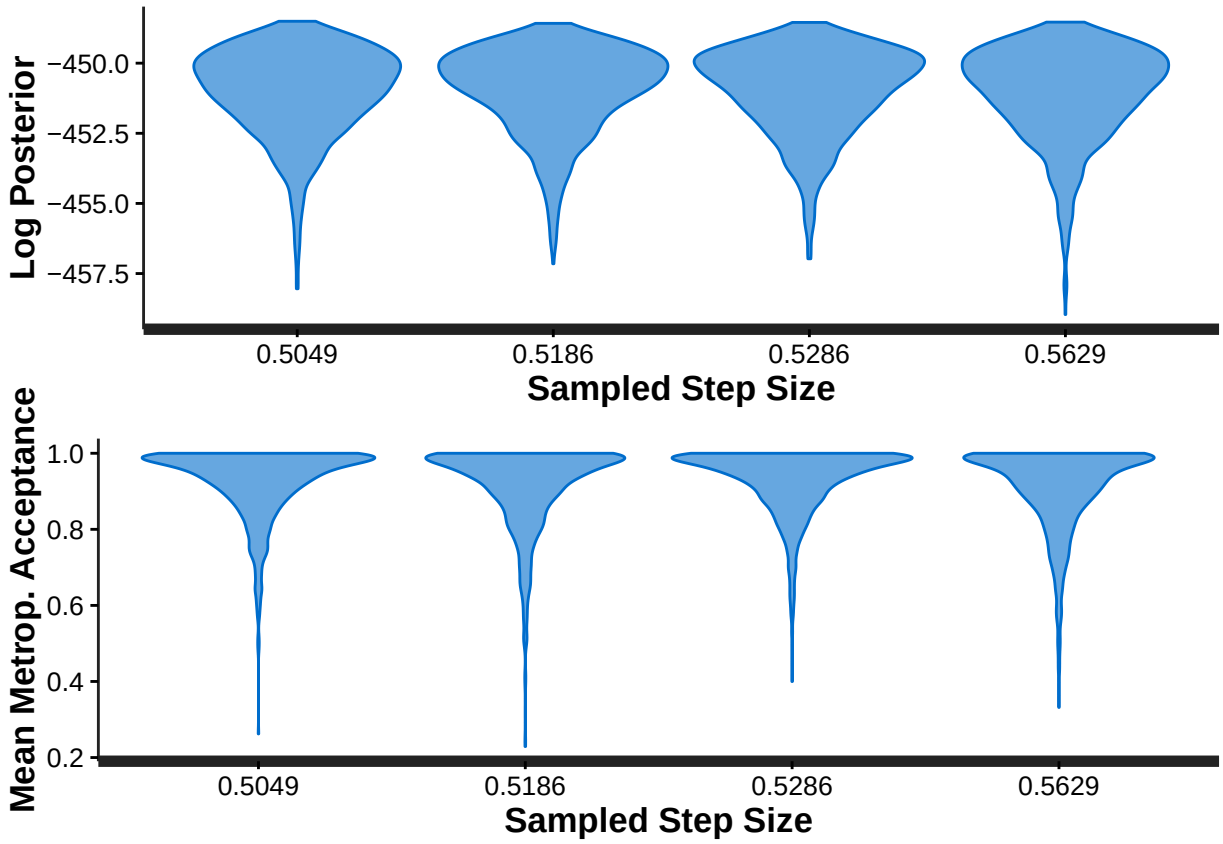


```
stan_diag(stanFitRobust,information = "sample",chain=0)
```

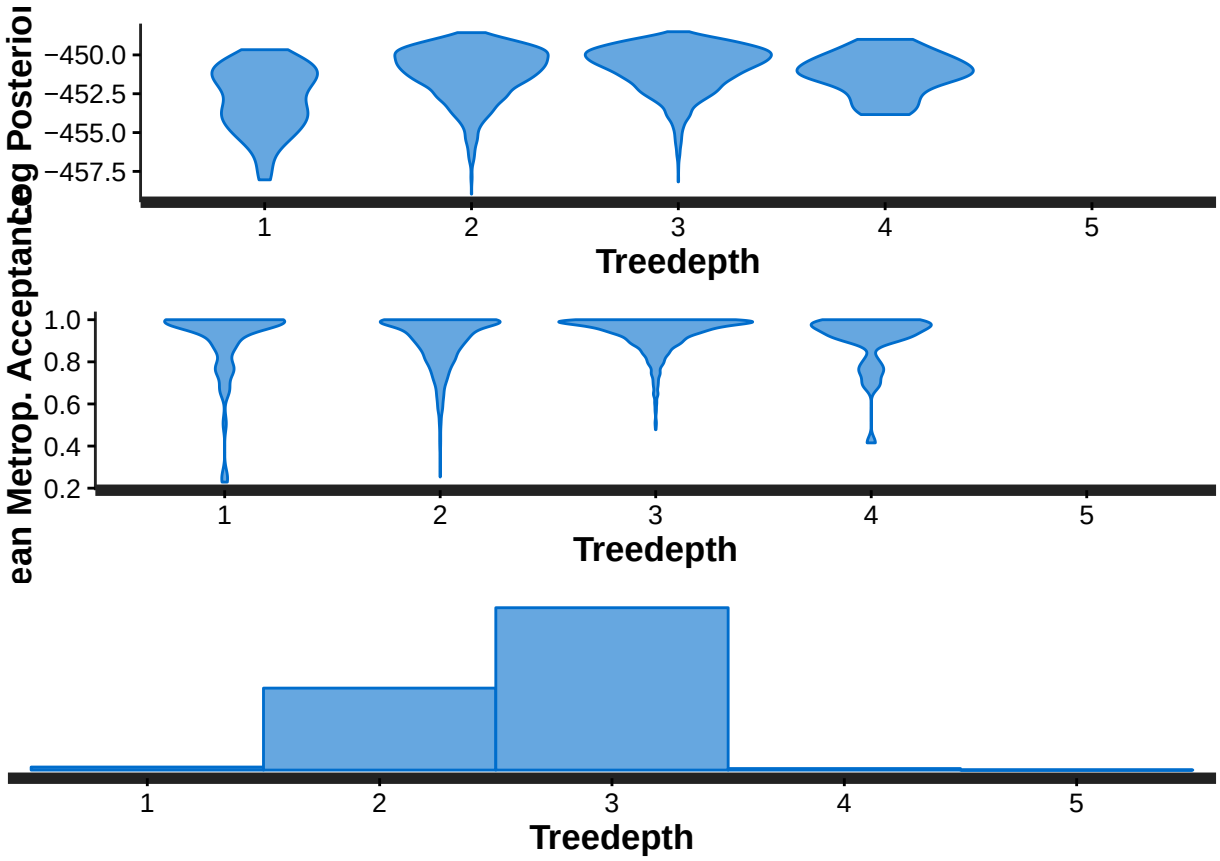
```
## Warning: Removed 1 rows containing missing values (geom_bar).
```



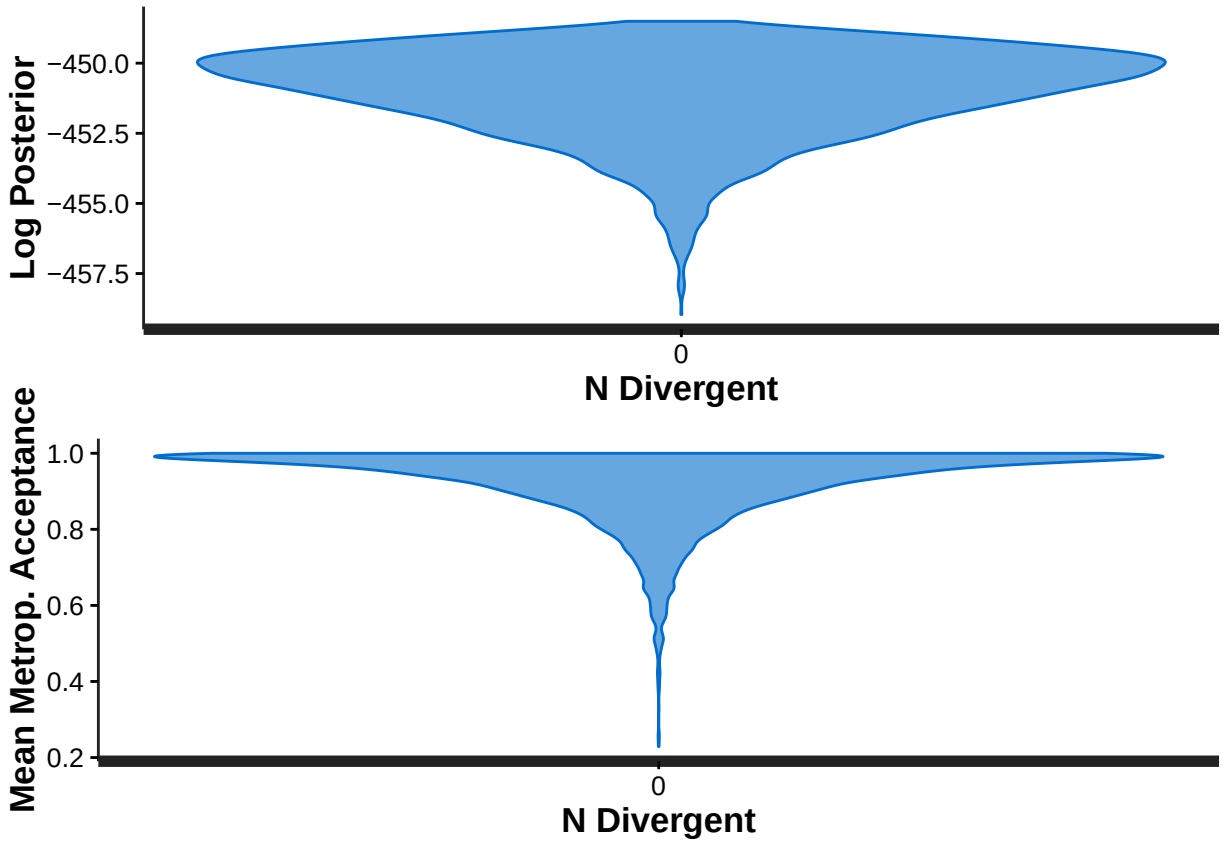

```
stan_diag(stanFitRobust,information = "stepsize",chain = 0)
```




```
stan_diag(stanFitRobust,information = "treedepth",chain = 0)
```




```
stan_diag(stanFitRobust,information = "divergence",chain = 0)
```




```
# launch_shinystan(stanFitRobust)
```

2. Do you think it is necessary to change the structure of the model shown on the diagram at the beginning of section 1?

It's not necessary to change the model structure shown in the diagram because it already does a pretty good job of separating the data groups.

3. Analyze convergence of MCMC, adjust parameters if necessary and rerun the process to obtain the a better quality of MCMC.

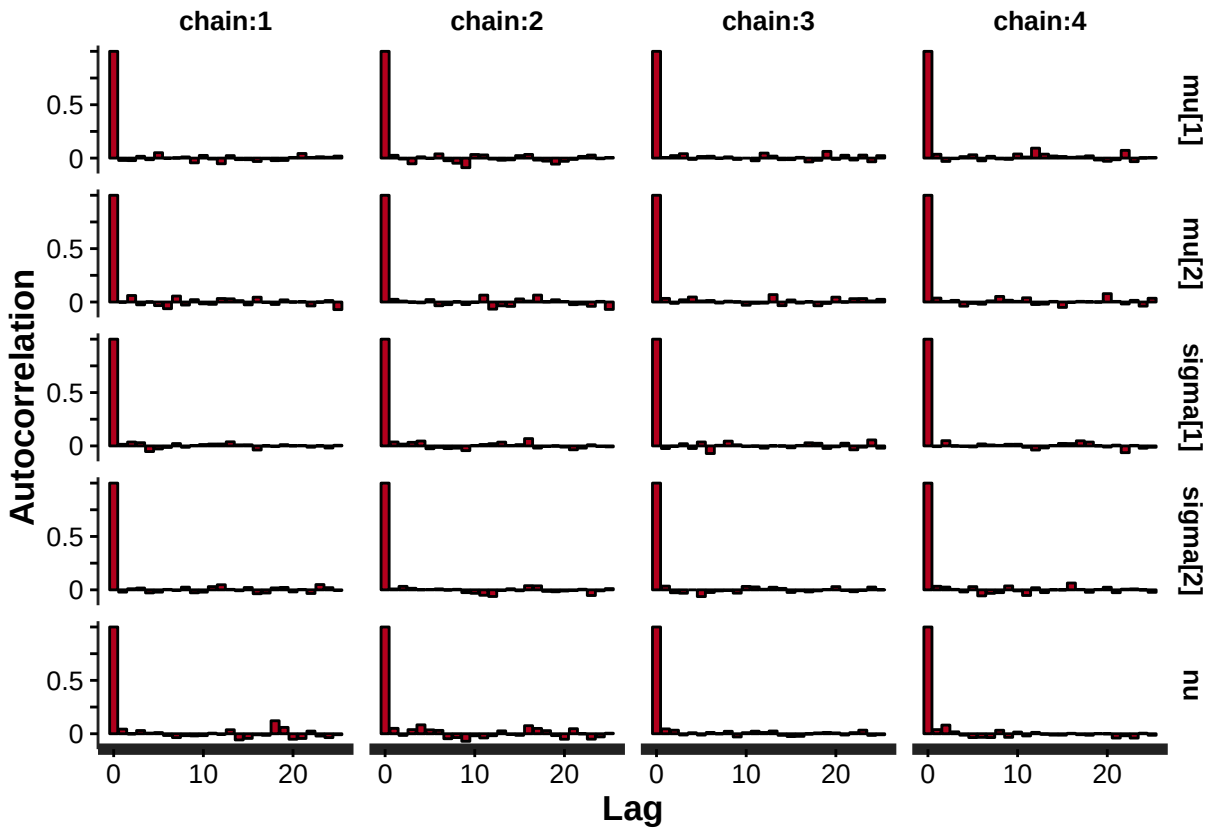
In the autocorrelation plots above, we see some high autocorrelation up to a lag of about 4. Because of that, we can adjust the thin steps from 1 to 4 in order to fix this & obtain a higher quality MCMC.

```
# due to autocorrelation, we can adjust the thin parameter to help

# run the MCMC
parameters = c( "mu" , "sigma" , "nu" )     # The parameters to be monitored
burnInSteps = 1000
nChains = 4
thinSteps = 4
numSavedSteps<-5000
# Get MC sample of posterior:
stanFitRobust2 <- sampling( object=stanDsoRobust ,
```

```
                    data = dataList ,
                    pars = parameters , # optional
                    chains = nChains ,
                    cores=nChains,
                    iter = (ceiling(numSavedSteps/nChains)*thinSteps+burnInSteps ) ,
                    warmup = burnInSteps ,
                    init = "random" , # optional
                    thin = thinSteps )

stan_ac(stanFitRobust2, separate_chains = T)
```



4. Answer the main question of the example discusses in section 1.3: Are the two groups different or not?

```
# frequentist
# t.test(y[x==1],y[x==2], var.equal=F, paired=FALSE)

# bayesian
summary(stanFitRobust)
```

```
## $summary
##                mean     se_mean        sd        2.5%         25%
## mu[1]      99.259764 0.03212849 1.783744   95.721474   98.074313
## mu[2]     107.154216 0.05051488 2.703067  101.863169  105.343016
## sigma[1]   11.321485 0.03269886 1.696022    8.375222   10.108379
## sigma[2]   17.939138 0.05386415 2.733891   12.913674   16.027206
```

```
## nu          3.834397 0.03188990 1.519504    1.956943    2.816108
## lp__     -451.023008 0.03711527 1.563947 -454.848028 -451.891865
##                  50%         75%       97.5%    n_eff     Rhat
## mu[1]      99.243942  100.473347  102.783069 3082.369 1.000085
## mu[2]     107.133338  108.972608  112.487805 2863.352 1.001450
## sigma[1]   11.225741   12.347624   15.060063 2690.282 1.000562
## sigma[2]   17.860545   19.709566   23.636711 2576.100 1.001176
## nu          3.502499    4.445916    7.761839 2270.373 1.000834
## lp__     -450.694621 -449.847320 -448.885115 1775.576 1.001014
##
## $c_summary
## , , chains = chain:1
##
##           stats
## parameter         mean       sd        2.5%         25%         50%
##   mu[1]      99.265237 1.799278   95.705572   98.080628   99.262590
##   mu[2]     107.097700 2.725591  101.557011  105.374211  107.107078
##   sigma[1]   11.322640 1.744750    8.303882   10.116283   11.213709
##   sigma[2]   17.914541 2.770269   13.139479   15.892178   17.739972
##   nu          3.835977 1.587273    1.925741    2.788829    3.490973
##   lp__     -451.095231 1.651169 -455.341310 -452.004854 -450.738118
##           stats
## parameter          75%       97.5%
##   mu[1]     100.480013  102.730246
##   mu[2]     108.971195  112.363098
##   sigma[1]   12.335998   15.351377
##   sigma[2]   19.610347   24.156141
##   nu          4.444373    8.247463
##   lp__     -449.860727 -448.864328
##
## , , chains = chain:2
##
##           stats
## parameter         mean       sd        2.5%         25%         50%
##   mu[1]      99.263277 1.772558   95.783767   98.081628   99.242436
##   mu[2]     107.285020 2.582148  102.413263  105.501872  107.163037
##   sigma[1]   11.240590 1.649428    8.320336   10.062455   11.182720
##   sigma[2]   17.902387 2.761713   12.890689   15.958065   17.787195
##   nu          3.775907 1.415436    1.989304    2.795895    3.467666
##   lp__     -450.977044 1.513686 -454.738069 -451.807421 -450.663747
##           stats
## parameter         75%       97.5%
##   mu[1]     100.46497  102.797716
##   mu[2]     109.02564  112.391137
##   sigma[1]   12.26111   14.724327
##   sigma[2]   19.75352   23.447103
##   nu          4.38125    7.495271
##   lp__     -449.84959 -448.946438
##
## , , chains = chain:3
##
##           stats
## parameter         mean       sd        2.5%         25%         50%
##   mu[1]      99.256998 1.824090   95.526491   98.089894   99.266133
```

```
##   mu[2]     107.062883 2.605525  102.073333  105.240219  107.107377
##   sigma[1]   11.394121 1.709063    8.456376   10.184076   11.279875
##   sigma[2]   17.868960 2.636334   12.963160   16.101700   17.779977
##   nu          3.881727 1.617811    1.965089    2.817124    3.517684
##   lp__     -450.989813 1.526633 -454.587224 -451.896300 -450.643877
##           stats
## parameter          75%       97.5%
##   mu[1]     100.536181  102.796451
##   mu[2]     108.756729  112.337609
##   sigma[1]   12.480581   15.066973
##   sigma[2]   19.506889   23.385062
##   nu          4.486797    7.842138
##   lp__     -449.810590 -448.907795
##
## , , chains = chain:4
##
##           stats
## parameter         mean       sd        2.5%        25%         50%
##   mu[1]      99.253544 1.740075   95.855566   98.01524   99.232338
##   mu[2]     107.171262 2.886135  101.703909  105.19765  107.157222
##   sigma[1]   11.328588 1.677895    8.430565   10.11160   11.199597
##   sigma[2]   18.070664 2.763817   12.827188   16.22715   18.098825
##   nu          3.843978 1.447461    1.961547    2.87114    3.511982
##   lp__     -451.029942 1.559778 -454.775114 -451.90657 -450.780778
##           stats
## parameter          75%       97.5%
##   mu[1]     100.422372  102.691524
##   mu[2]     109.115049  112.938285
##   sigma[1]   12.295939   14.928913
##   sigma[2]   19.882673   23.601600
##   nu          4.429205    7.616689
##   lp__     -449.868192 -448.843781
```

```
dis1<-cbind(Mu=rstan::extract(stanFitRobust,pars="mu[1]")$'mu[1]',
            Sigma=rstan::extract(stanFitRobust,pars="sigma[1]")$'sigma[1]')
dis2<-cbind(Mu=rstan::extract(stanFitRobust,pars="mu[2]")$'mu[2]',
            Sigma=rstan::extract(stanFitRobust,pars="sigma[2]")$'sigma[2]')

(rbind(mean1HDI=HDIofMCMC(dis1[,1]),mean2HDI=HDIofMCMC(dis2[,1])))
```

```
##               [,1]     [,2]
## mean1HDI  95.63127 102.6451
## mean2HDI 101.85795 112.4683
```

```
(rbind(sigma1HDI=HDIofMCMC(dis1[,2]),sigma2HDI=HDIofMCMC(dis2[,2])))
```

```
##                [,1]     [,2]
## sigma1HDI  8.265466 14.83875
## sigma2HDI 12.817464 23.44729
```

```
c(mean(dis1[,2]),mean(dis2[,2])) # mean values
```

```
## [1] 11.32148 17.93914
```

```
c(sd(dis1[,2]),sd(dis2[,2]))     # standard deviations of samples of MCMC standard deviations
```

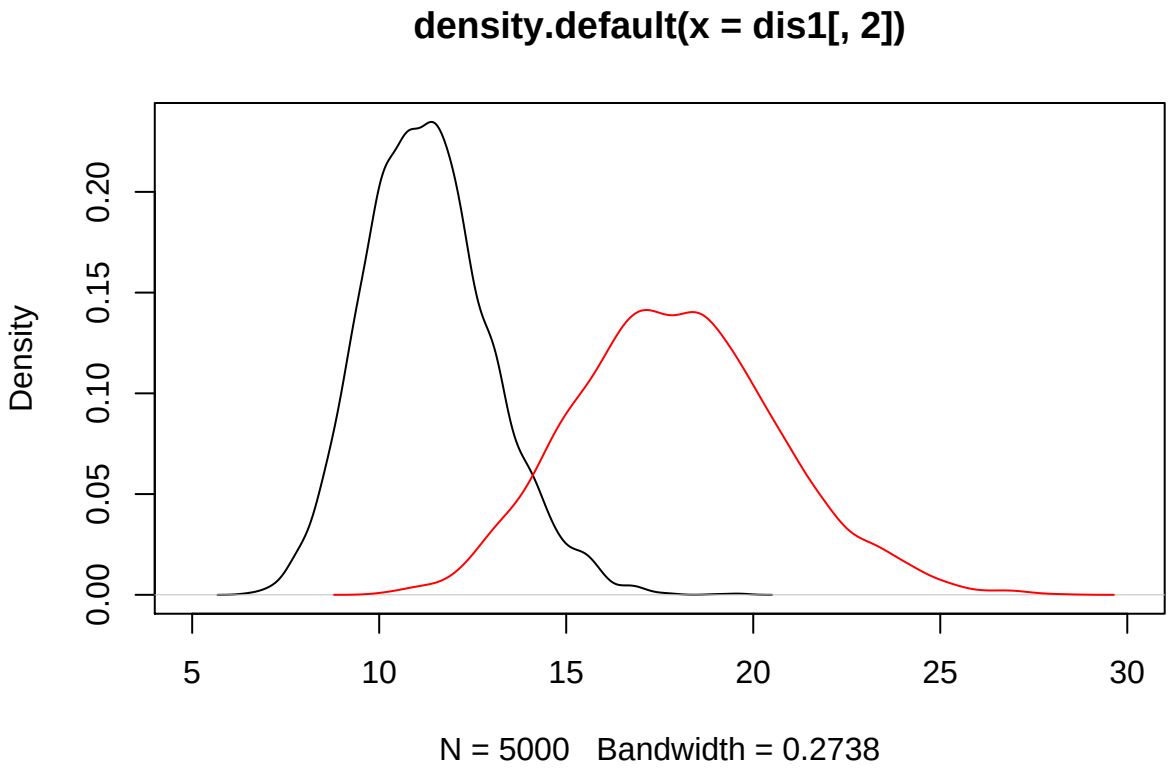
```
## [1] 1.696022 2.733891
```

```r
#Kolmogorov-Smirnov test for posterior distributions of standard deviations
ks.test(dis1[,2],dis2[,2])
```

```
## Warning in ks.test(dis1[, 2], dis2[, 2]): p-value will be approximate in
## the presence of ties
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  dis1[, 2] and dis2[, 2]
## D = 0.8642, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

```r
den<-density(dis2[,2])
plot(density(dis1[,2]),xlim=c(5,30))
lines(den$x,den$y,col="red")
```



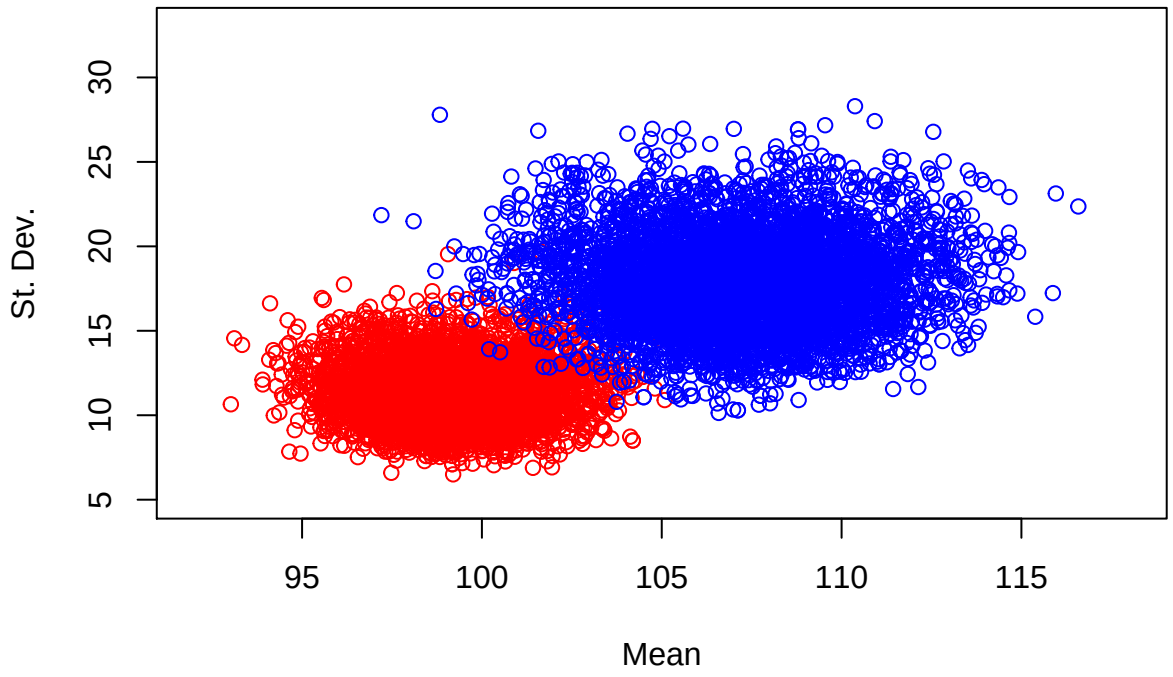

```r
#t-test for means of posterior distributions for standard deviations
t.test(dis1[,2],dis2[,2], var.equal=F, paired=FALSE)
```

```
##
##  Welch Two Sample t-test
##
## data:  dis1[, 2] and dis2[, 2]
## t = -145.45, df = 8350.4, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -6.706842 -6.528464
## sample estimates:
## mean of x mean of y
##  11.32148  17.93914
```

```r
t.test(dis1[,1],dis2[,1], var.equal=F, paired=FALSE)
```

```
##
##  Welch Two Sample t-test
##
## data:  dis1[, 1] and dis2[, 1]
## t = -172.37, df = 8658.8, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -7.984232 -7.804673
## sample estimates:
## mean of x mean of y
##  99.25976 107.15422
```

```r
plot(dis1,xlim=c(92,118),ylim=c(5,33),col="red",xlab="Mean",ylab="St. Dev.")
points(dis2,col="blue")
```



From the plots as well as tests on the extracted Mu and Sigma values, we can conclude that there are indeed two different groups within the data.

5. Find at least 3 strong arguments (different models or methods) proving your opinion based on the simulated chains.

```r
# structure the data into a dataframe with Mu, Sigma, and Group
dat1<-as.data.frame(dis1)
dat1$Group<-0
dat2<-as.data.frame(dis2)
dat2$Group<-1
dat<-rbind(dat1,dat2)
dat$Group<-as.factor(dat$Group)
```

```r
# logistic model
log.mod<-glm(Group ~ Mu + Sigma, data=dat,family = "binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
summary(log.mod)
```

```
##
## Call:
## glm(formula = Group ~ Mu + Sigma, family = "binomial", data = dat)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.5637  -0.0075   0.0000   0.0022   3.1261
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -202.06984   11.02777  -18.32   <2e-16 ***
## Mu             1.76982    0.09946   17.79   <2e-16 ***
## Sigma          1.42685    0.08294   17.20   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 13862.9  on 9999  degrees of freedom
## Residual deviance:   471.8  on 9997  degrees of freedom
## AIC: 477.8
##
## Number of Fisher Scoring iterations: 10
```

```r
# randomForest classification tree
library(randomForest)
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
rf.mod<-randomForest(Group ~ Mu + Sigma, data=dat)
table(dat$Group,predict(rf.mod,type='response'))
```

```
##
##        0    1
##   0 4960   40
##   1   45 4955
```

```r
# SVM
library(e1071)
svm.mod<-svm(Group~Mu+Sigma,data=dat,cost=1,kernel="radial")
table(dat$Group,predict(svm.mod,type='response'))
```

```
##
##        0    1
##   0 4961   39
##   1   40 4960
```

The above methods lend evidence to the existence of groups within the data. The logistic regression model summary shows that both Mu and Sigma are very significant in classifying observations into two groups. We wouldn't expect to see this if there were actually no true distinct groups. The other two methods, random forest and support vector machine models, indicate the same conclusion through their confusion matrices. Both models are very good at correctly placing the data points into two distinct groups. If there were truly no groups in the data, we wouldn't expect this models to accurately sort the data this way. So, it looks like there are indeed two different groups within the data.