BEST with Stan

In this notebook, I show the Bayesian model behind the BESTmcmc using the Bayesian modelling package stan

Use the battery life data from class 4.

```
library(readr)
batterydata_r <- read_csv("~/Google Drive crowston@syr.edu/Courses/IST 772 Crowston/Week 4/batterydata.
```

```
## Parsed with column specification:
## cols(
##   Obs = col_double(),
##   Battery = col_double(),
##   Time = col_double()
## )
```

BESTmcmc

```
#install.packages("BEST")
library(BEST)
```

```
## Loading required package: HDInterval
```

```
mcmcsteps <- 3000   # how many iterations of the sampling to run, so we do the same for all approaches

bestOut <- BESTmcmc(batterydata_r$Time[batterydata_r$Battery == 1],
                    batterydata_r$Time[batterydata_r$Battery == 2],
                    numSavedSteps=mcmcsteps)
```

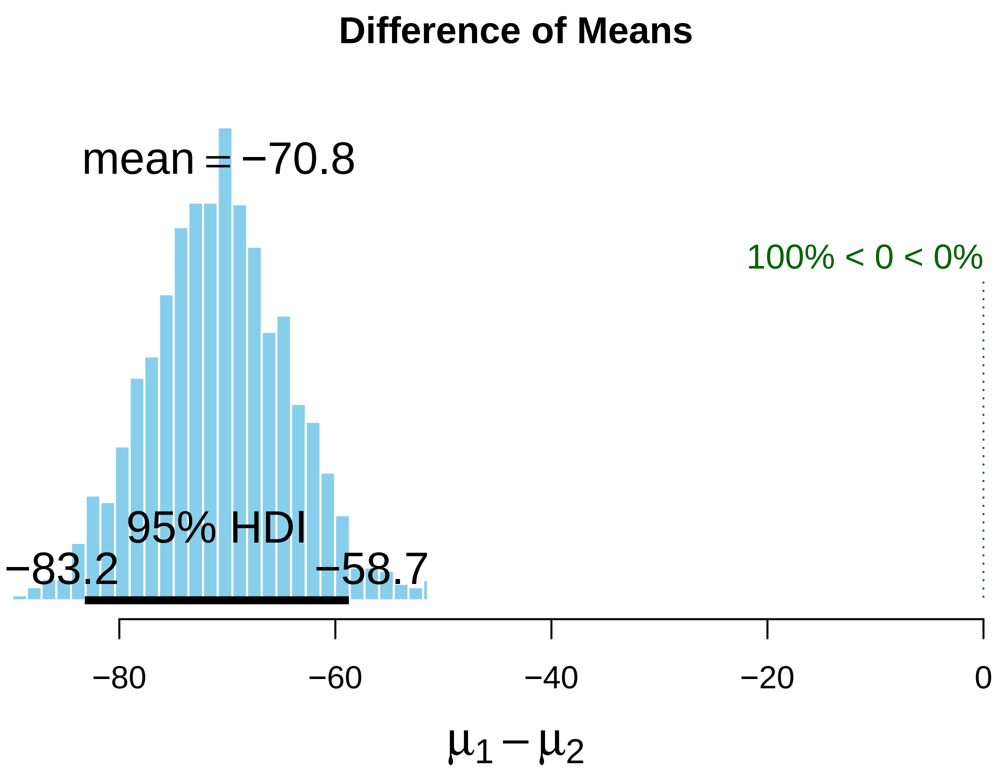
```
## Waiting for parallel processing to complete...
```

```
## done.
```

```
summary(bestOut)
```

```
##              mean  median    mode HDI%   HDIlo   HDIup compVal %>compVal
## mu1       1323.92 1323.88 1324.13   95 1311.17 1335.59
## mu2       1394.71 1394.70 1394.69   95 1393.88 1395.52
## muDiff     -70.79  -70.78  -70.53   95  -83.19  -58.74       0         0
## sigma1      58.89   58.69   58.94   95   50.02   67.41
## sigma2       3.82    3.80    3.81   95    3.23    4.44
## sigmaDiff   55.08   54.90   55.12   95   46.29   63.63       0       100
## nu          58.28   50.33   33.75   95    8.11  128.65
## log10nu      1.69    1.70    1.74   95    1.20    2.20
## effSz       -1.71   -1.71   -1.72   95   -2.11   -1.32       0         0
```

```
plot(bestOut)
```



A stan model to implement the BEST model

Stan is a general purpose Bayesian modelling package. The code below implements in stan the model for BEST. The model is compiled to a code module, which takes a minute or two.

```
// The data section describes the data you're trying to fit, in this case, N values in two groups
data {
  int<lower=1> N;                              // sample size (note: putting bounds
                                               //    provides simple data check)
  vector[N] y;                                 // response
  int<lower=1, upper=2> groupID[N];            // group ID
}

// The transformed data section describes any pre-processing we want to do to our data
transformed data{
  real meany;                                  // mean of y; see mu prior
  real sdy;                                    // sd of y; see mu prior

  meany = mean(y);
  sdy = sd(y);
}

// The parameters section describes the parameters of the distribution that we're estimating
parameters {
  vector[2] mu;                                // estimated group means and sd
  vector<lower=0>[2] sigma;                    // Kruschke puts upper bound as well; ignored here
  real<lower=0, upper=100> nu;                 // df for t distribution
}
```

```
// The model section describes the priors on the parameters and how the data are
//   derived from the parameters (the likelihood)
model {
  // priors, chosen to match BESTmcmc
  mu ~ normal(meany, 1000*sdy);
  sigma ~ uniform(sdy/1000, sdy*1000);
  nu ~ exponential(1.0/29);                    // BESTmcmc has exponential(1/29) + 1
                                               //     but stan didn't accept that

  // likelihood
  for (n in 1:N){
    y[n] ~ student_t(nu, mu[groupID[n]], sigma[groupID[n]]);
  }
}

// The generated quantities section describes post-processing on the parameters
generated quantities {
  real muDiff;                                 // mean difference
  real CohensD;                                // effect size; see footnote 1 in Kruschke paper

  muDiff = mu[1] - mu[2];
  CohensD = muDiff / sqrt(sum(sigma)/2);
}
```

Put the data to be passed to stan in a list

```
fit_data <- list(N=length(batterydata_r$Time), y=batterydata_r$Time, groupID=batterydata_r$Battery)
```

Draw samples of the parameters (uses MCMC). The default is to run 4 chains with random starting points. The chains can run in parallel if you have a multi-core CPU.

```
#install.packages("rstan")
library(rstan)
```

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

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

```
## rstan (Version 2.19.3, 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)
```

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

stan_fit <- sampling(best_model, data=fit_data, iter=mcmcsteps)
```

Look at the results. The summary reports summary statistics for the sampled parameters altogether and for each chain. Because the MCMC process results in samples that aren't completely independent, n_eff reports the effective sample size, which should be large. Rhat will be close to 1 if the sampling converged; if it isn't there's a problem. The samples from the different chains should be similar; if not, there is a problem with convergence.

```
summary(stan_fit)
```

```
## $summary
```

```
##                 mean     se_mean         sd        2.5%         25%        50%
## mu[1]    1324.076220 0.080197135  6.5293242 1311.218641 1319.699507 1324.09084
## mu[2]    1394.712303 0.005180851  0.4319791 1393.862903 1394.425160 1394.71105
## sigma[1]   58.818679 0.055187843  4.6080038   50.709898   55.608890   58.53277
## sigma[2]    3.799869 0.003755277  0.3124722    3.240546    3.587178    3.78208
## nu         48.134849 0.329491893 21.9389979   13.579588   30.727635   45.16683
## muDiff    -70.636082 0.080694082  6.5481064  -83.488697  -75.000404  -70.61881
## CohensD   -12.648827 0.014967674  1.2508885  -15.035078  -13.502809  -12.66810
## lp__     -606.054207 0.032236391  1.6608859 -610.159514 -606.919756 -605.70694
##                 75%       97.5%    n_eff      Rhat
## mu[1]    1328.47413 1337.022193 6628.553 0.9997464
## mu[2]    1394.99777 1395.563480 6952.216 0.9995905
## sigma[1]   61.66202   68.874234 6971.702 0.9998959
## sigma[2]    3.99029    4.472526 6923.707 0.9996624
## nu         63.41364   94.574625 4433.473 0.9997225
## muDiff    -66.27143  -57.664371 6584.883 0.9997425
## CohensD   -11.80669  -10.145956 6984.391 0.9998392
## lp__     -604.80768 -603.899627 2654.525 1.0019085
##
## $c_summary
## , , chains = chain:1
##
##           stats
## parameter         mean         sd        2.5%        25%         50%
##   mu[1]    1323.941894  6.6520150 1310.914355 1319.46926 1323.892786
##   mu[2]    1394.708116  0.4467800 1393.855780 1394.40279 1394.707934
##   sigma[1]   58.828305  4.5622856   50.922945   55.52519   58.561500
##   sigma[2]    3.797287  0.3128381    3.237881    3.58438    3.782243
##   nu         47.811725 21.6226720   12.738347   30.73600   44.710726
##   muDiff    -70.766222  6.6524042  -83.689543  -75.21646  -70.867400
##   CohensD   -12.672387  1.2812692  -15.104924  -13.55954  -12.723371
##   lp__     -606.086747  1.6593249 -610.063441 -606.97882 -605.754713
##           stats
## parameter          75%       97.5%
##   mu[1]    1328.509353 1336.799778
##   mu[2]    1394.994158 1395.625820
##   sigma[1]   61.682713   68.633469
##   sigma[2]    3.986295    4.487786
##   nu         62.086673   94.582923
##   muDiff    -66.307778  -58.008394
##   CohensD   -11.758797  -10.117129
##   lp__     -604.798873 -603.909868
##
## , , chains = chain:2
##
##           stats
## parameter         mean         sd        2.5%         25%         50%
##   mu[1]    1324.247965  6.3378900 1312.023246 1319.819194 1324.104589
##   mu[2]    1394.709324  0.4145817 1393.885200 1394.426511 1394.702541
##   sigma[1]   58.755326  4.4489154   50.809038   55.602588   58.654097
##   sigma[2]    3.794545  0.3101471    3.214035    3.586387    3.778027
##   nu         48.138843 21.3808001   14.771372   31.029529   45.891172
##   muDiff    -70.461359  6.3904142  -82.685602  -74.877008  -70.605498
##   CohensD   -12.624315  1.2353654  -14.982918  -13.473840  -12.625231
```

```
##   lp__      -605.916374  1.5496286 -609.686350 -606.795823 -605.576237
##            stats
## parameter         75%       97.5%
##   mu[1]     1328.33272 1337.024152
##   mu[2]     1394.99093 1395.524204
##   sigma[1]    61.48513   68.439825
##   sigma[2]     3.98948    4.466112
##   nu          62.66900   93.825085
##   muDiff     -66.45777  -57.665047
##   CohensD    -11.81732  -10.100773
##   lp__      -604.70466 -603.921945
##
## , , chains = chain:3
##
##            stats
## parameter          mean         sd        2.5%         25%         50%
##   mu[1]     1324.119463  6.5748442 1311.302450 1319.624679 1324.474721
##   mu[2]     1394.722569  0.4278058 1393.867304 1394.437399 1394.725288
##   sigma[1]    58.926008  4.7737738   50.536190   55.645543   58.554972
##   sigma[2]     3.805023  0.3326129    3.202483    3.581246    3.785354
##   nu          48.308258 22.1870141   13.924541   30.451566   45.343343
##   muDiff     -70.603105  6.5752624  -83.363633  -75.086905  -70.246784
##   CohensD    -12.632485  1.2526141  -15.018057  -13.466181  -12.631558
##   lp__      -606.163858  1.7481265 -610.800617 -606.989338 -605.798886
##            stats
## parameter          75%       97.5%
##   mu[1]     1328.495903 1337.375005
##   mu[2]     1395.010494 1395.549953
##   sigma[1]    61.834309   69.416543
##   sigma[2]     4.006466    4.518185
##   nu          64.917014   93.772038
##   muDiff     -66.203930  -57.279520
##   CohensD    -11.809055  -10.164891
##   lp__      -604.929200 -603.932818
##
## , , chains = chain:4
##
##            stats
## parameter         mean         sd       2.5%        25%         50%        75%
##   mu[1]     1323.99556  6.5506440 1310.92841 1319.85294 1323.968112 1328.54239
##   mu[2]     1394.70920  0.4383464 1393.81931 1394.42168 1394.705006 1394.99531
##   sigma[1]    58.76508  4.6435512   50.71482   55.69852   58.267529   61.55204
##   sigma[2]     3.80262  0.2932430    3.28780    3.60461    3.784059    3.98612
##   nu          48.28057 22.5642152   13.23343   30.53499   44.915060   64.60571
##   muDiff     -70.71364  6.5738913  -83.93361  -74.85102  -70.735103  -66.20530
##   CohensD    -12.66612  1.2342916  -14.98452  -13.47344  -12.700772  -11.84606
##   lp__      -606.04985  1.6723978 -610.20887 -606.93079 -605.687016 -604.80769
##            stats
## parameter        97.5%
##   mu[1]     1336.865293
##   mu[2]     1395.572719
##   sigma[1]    68.975805
##   sigma[2]     4.411974
##   nu          95.131004
```

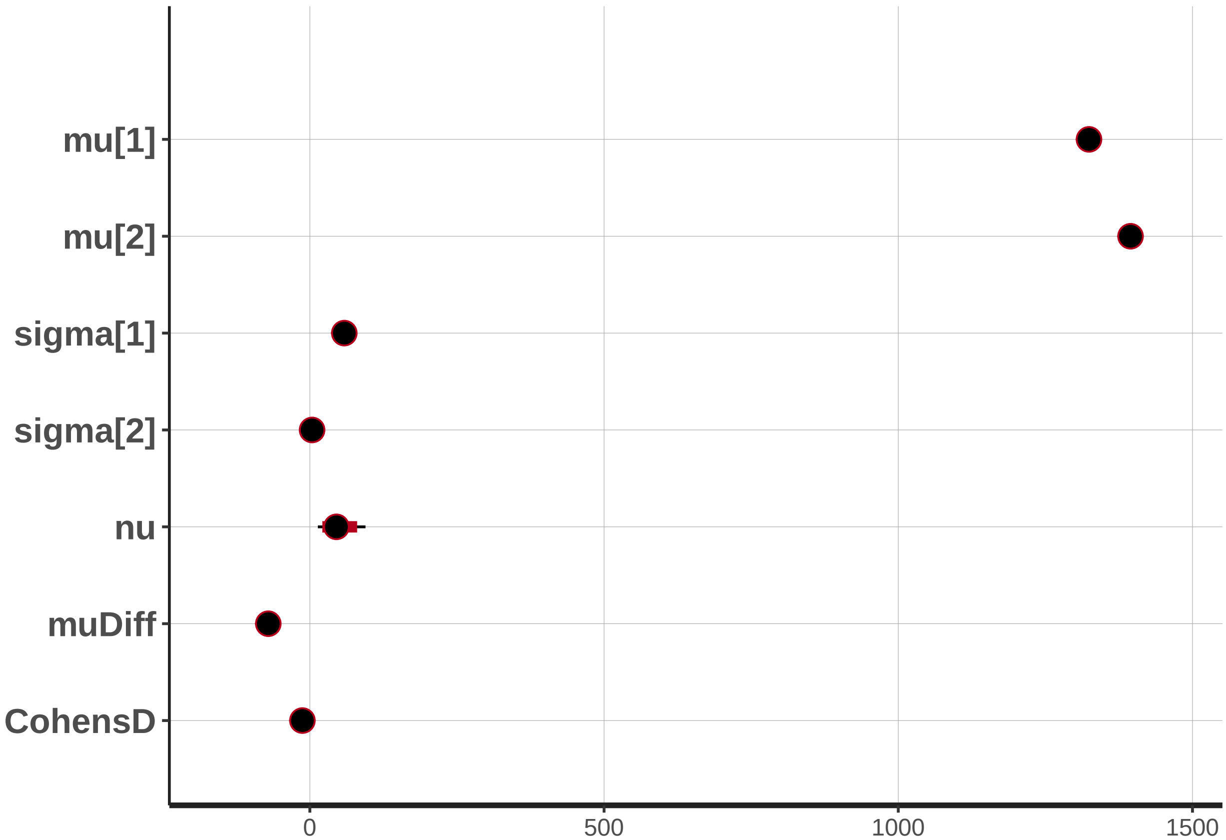
```
##    muDiff     -57.772745
##    CohensD    -10.271616
##    lp__      -603.839657
```

Plot the results.

```r
plot(stan_fit)
```

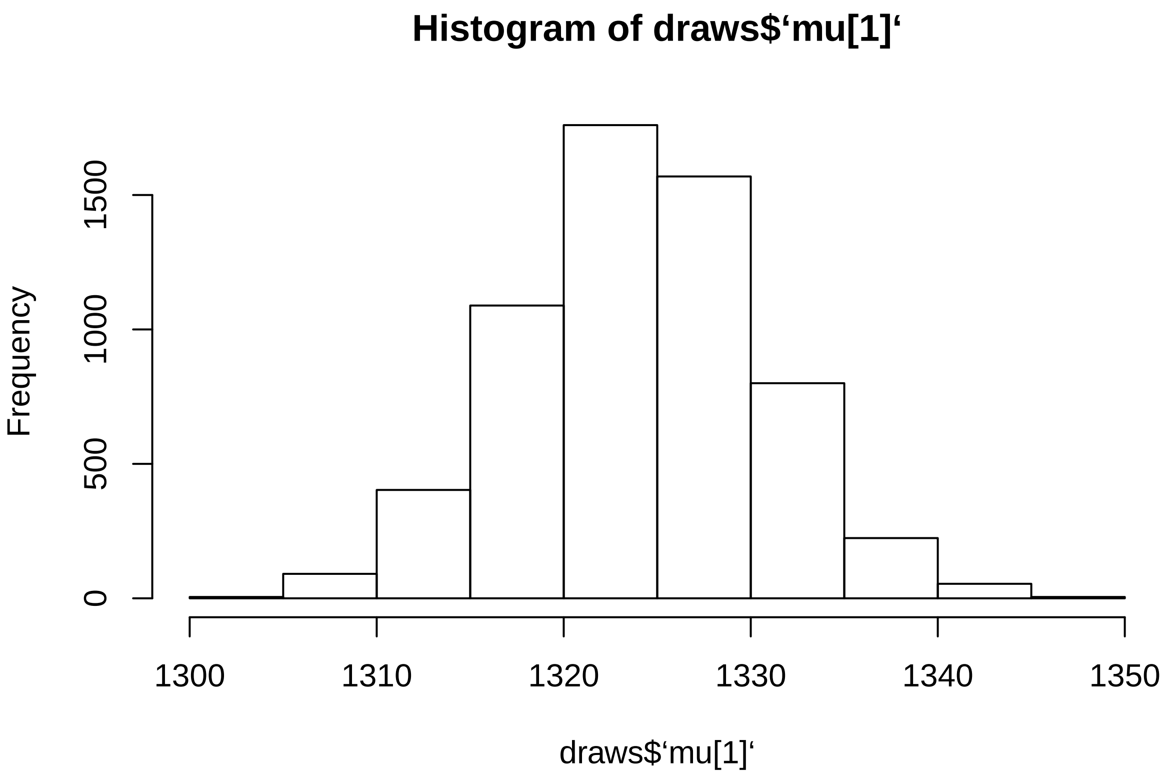
```
## ci_level: 0.8 (80% intervals)
```

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

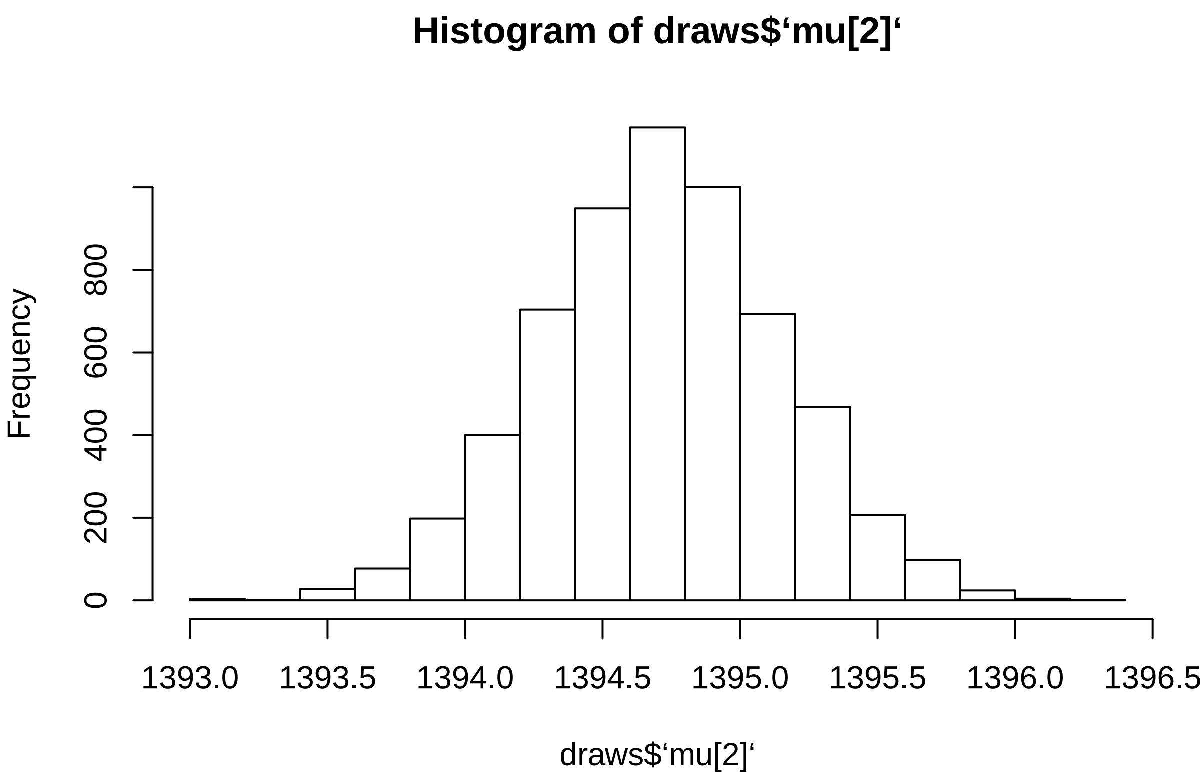


We can retrieve the sampled values for individual parameters, e.g., to plot them.

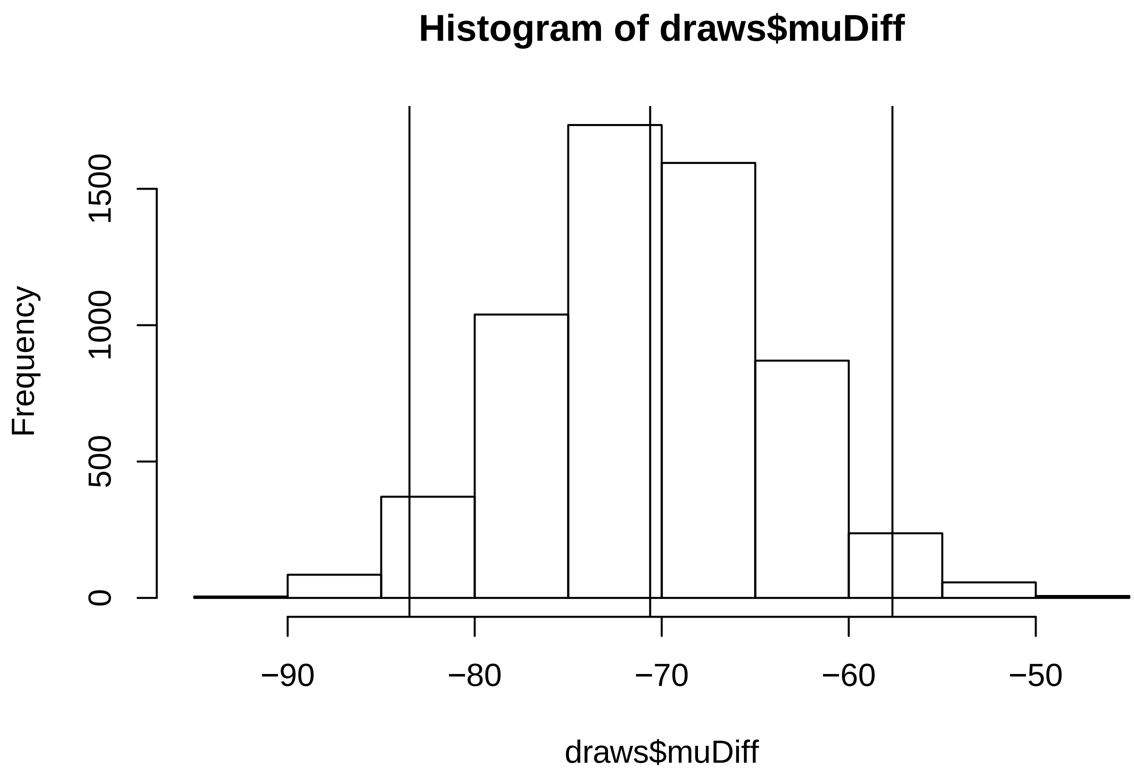
```r
draws<-as.data.frame(stan_fit)
hist(draws$`mu[1]`)
```




```r
hist(draws$`mu[2]`)
```




```r
hist(draws$muDiff)
abline(v=quantile(draws$muDiff, c(0.025, 0.5, 0.975)))
```



Bayesian comparison of two groups using brms

Stan is powerful and flexible but also hard to use. The brms package is a wrapper around stan for some common models (it also takes a while to run). A t-test can be expressed as a regression (coming in class 8) on a binary variable. For these examples, I didn't match the priors used in BESTmcmc. (NB: for some reason, only when knitting, the brm command prints a warning, but it still works.)

```
#install.packages("brms")
library(brms)
```

```
## Loading required package: Rcpp
```

```
## Loading 'brms' package (version 2.11.1). Useful instructions
## can be found by typing help('brms'). A more detailed introduction
## to the package is available through vignette('brms_overview').
```

```
##
## Attaching package: 'brms'
```

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

```
brm_fit<-brm(Time ~ Battery, data=batterydata_r,
             iter=mcmcsteps, cores=4, file="battery_bms")
```

Look at the results.

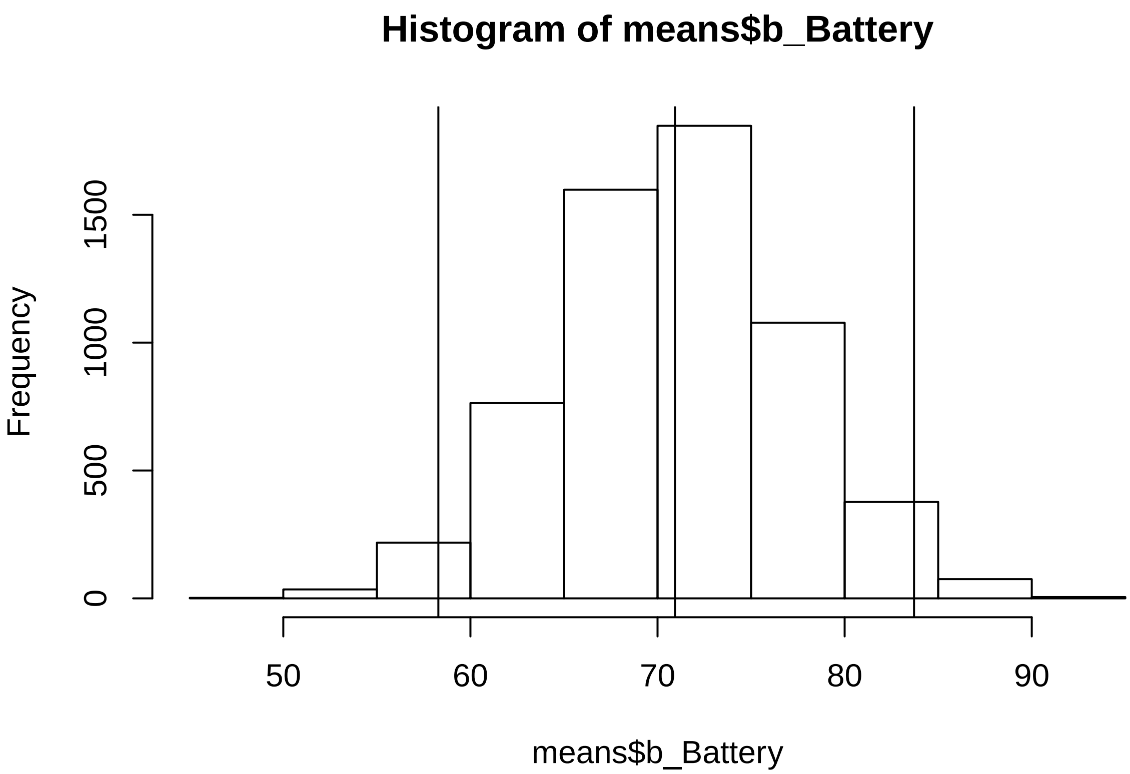
```
summary(brm_fit)
```

```
##  Family: gaussian
```

```
##   Links: mu = identity; sigma = identity
## Formula: Time ~ Battery
##    Data: batterydata_r (Number of observations: 172)
## Samples: 4 chains, each with iter = 3000; warmup = 1500; thin = 1;
##          total post-warmup samples = 6000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept  1253.58      9.94  1233.31  1273.37 1.00     5936     4515
## Battery      70.93      6.32    58.29    83.71 1.00     6038     4730
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma    41.25      2.26    37.04    45.91 1.00     6179     4469
##
## Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

The coefficient on Battery is the difference in Time between the two kinds of batteries. We can retrieve and plots the sampled values as follows:

```r
means<-posterior_samples(brm_fit, "^b_")      # retrieve the samples for the b weights
hist(means$b_Battery)
abline(v=quantile(means$b_Battery, c(0.025, 0.5, 0.975)))
```



Note that because Battery is 1 or 2, the predicted means of the two groups are the intercept plus 1 or 2 times the difference, i.e.,

```r
mean(means$b_Intercept + means$b_Battery)      # when Battery == 1
```

```
## [1] 1324.509
```

```r
mean(means$b_Intercept + means$b_Battery * 2) # when Battery == 2
```

```
## [1] 1395.437
```

Unequal sigmas

The previous model assumed a common standard deviation for the two groups, which we know is not the case. We can model different SDs in the two groups and a t distribution for the data as follows:

```r
brm_fit_uneq<-brm(bf(Time ~ Battery, sigma ~ Battery), family=student,
                  data=batterydata_r, iter=mcmcsteps, cores=4, file='battery_brm',
                  control=list(max_treedepth=20))       # added because there was a warning about it on
```

Look at the results.

```r
summary(brm_fit_uneq)
```

```
## Warning: The model has not converged (some Rhats are > 1.05). Do not analyse the results!
## We recommend running more iterations and/or setting stronger priors.
```

```
##  Family: student
##   Links: mu = identity; sigma = log; nu = identity
## Formula: Time ~ Battery
##          sigma ~ Battery
##    Data: batterydata_r (Number of observations: 172)
## Samples: 4 chains, each with iter = 3000; warmup = 1500; thin = 1;
##          total post-warmup samples = 6000
##
## Population-Level Effects:
##                 Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept        1258.94     14.15  1227.34  1283.59 1.08       32       98
## sigma_Intercept     7.00      0.41     6.47     7.95 1.27       11       28
## Battery            68.14      7.00    56.15    83.99 1.06       42       83
## sigma_Battery      -2.96      0.42    -3.85    -2.53 1.28       11       29
##
## Family Specific Parameters:
##    Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## nu    26.38     19.54     1.45    68.46 1.28       11       68
##
## Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

As before, the coefficient for Battery is the predicted difference. However, the sigma values are the log of the standard deviations of the groups, so we need to use exp to obtain the estimate. And again, because Battery is coded as 1 or 2, we need to include the b weight as well.

```r
sigmas<-exp(posterior_samples(brm_fit_uneq, "^b_sigma_")) # retrieve the samples for the b weights
mean(sigmas$b_sigma_Intercept*sigmas$b_sigma_Battery)     # when Battery == 1
```

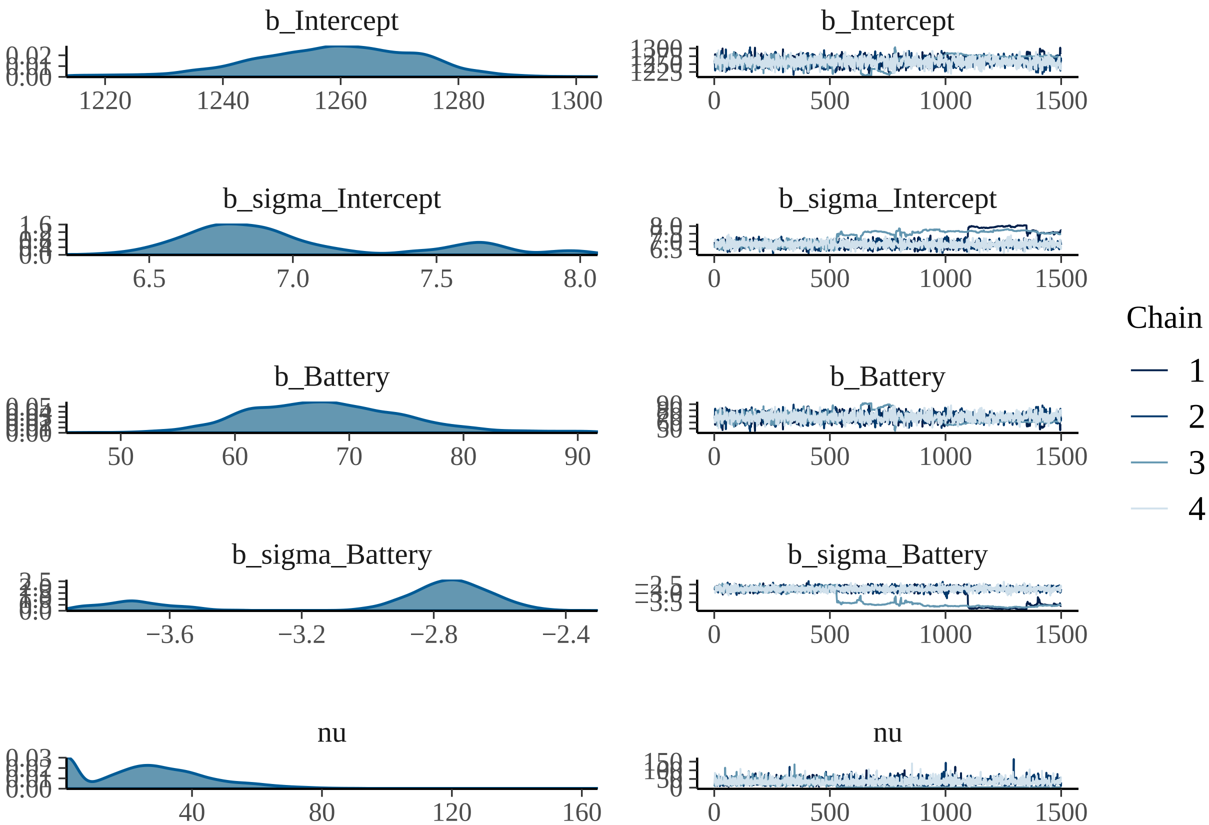
```
## [1] 57.01247
```

```r
mean(sigmas$b_sigma_Intercept*sigmas$b_sigma_Battery^2)   # when Battery == 2
```

```
## [1] 3.185306
```

We can plot the samples.

```r
plot(brm_fit_uneq)
```



A non-parametric comparison of two groups

By the way, if you find that your data don't meet the assumptions of the t test, specifically, normally-distributed data, another alternative is to use a non-parametric test that doesn't make that assumption, e.g., the Mann Whitney U Test aka the Wilcoxon rank sum test.

```r
wilcox.test(batterydata_r$Time[batterydata_r$Battery == 1],
            batterydata_r$Time[batterydata_r$Battery == 2])
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  batterydata_r$Time[batterydata_r$Battery == 1] and batterydata_r$Time[batterydata_r$Battery ==
## W = 378.5, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0
```