BDA - Assignment 7

Anonymous

Contents

Exercise 1. Linear model: drowning data with Stan

The provided data drowning in the aaltobda package contains the number of people who died from drowning each year in Finland 1980–2019. A statistician is going to fit a linear model with Gaussian residual model to these data using time as the predictor and number of drownings as the target variable (see the related linear model example for the Kilpisjärvi-temperature data in the example Stan codes). She has two objective questions: i) What is the trend of the number of people drowning per year? (We would plot the histogram of the slope of the linear model.) ii) What is the prediction for the year 2020? (We would plot the histogram of the posterior predictive distribution for the number of people drowning at $\tilde{x} = 2020$.) To access the data, use:

```
library(aaltobda)
data("drowning")
```

Corresponding Stan code is provided in Listing 1. However, it is not entirely correct for the problem. First, there are three mistakes. Second, there are no priors defined for the parameters. In Stan, this corresponds to using uniform priors. Your tasks are the following:

a)

Find the three mistakes in the code and fix them. Report the original mistakes and your fixes clearly in your report. Include the full corrected Stan code in your report.
Hint: You may find some of the mistakes in the code using Stan syntax checker. If you copy the Stan code to a file ending .stan and open it in RStudio (you can also choose from RStudio menu File → New File → Stan file to create a new Stan file), the editor will show you some syntax errors. More syntax errors might be detected by clicking 'Check' in the bar just above the Stan file in the RStudio editor. Note that some of the errors in the presented Stan code may not be syntax errors.

The three mistakes in the original Stan code are:

```
"
data {
  int<lower=0> N; // number of data points
  vector[N] x; // observation year
  vector[N] y; // observation number of drowned
  real xpred; // prediction year
}

parameters {
  real alpha;
  real beta;
  real<upper=0> sigma; // <- MISTAKE 1: Should have been real<lower=0> sigma
}

transformed parameters {
  vector[N] mu = alpha + beta*x;
}

model {
  y ~ normal(mu, sigma) // <- MISTAKE 2: No semicolon at the end
}

generated quantities {
  real ypred = normal_rng(mu, sigma); // <- MISTAKE 3: should have been
  // ypred = normal_rng(mu_pred, sigma), where mu_pred = alpha + beta * xpred.
  // The derivation of mu_pred can be added to the transformed parameters.
}
```

```
"
```

The completely fixed Stan code now becomes:

```
"
data {
  int<lower=0> N; // number of data points
  vector[N] x; // observation year
  vector[N] y; // observation number of drowned
  real xpred; // prediction year
}

parameters {
  real alpha;
  real beta;
  real<lower=0> sigma; // <- MISTAKE 1 fixed
}

transformed parameters {
  vector[N] mu = alpha + beta*x;
  real mu_pred = alpha + beta * xpred; // <- The derivation of mu_pred is added.
}

model {
  y ~ normal(mu, sigma); // <- MISTAKE 2 fixed
}

generated quantities {
  real ypred = normal_rng(mu_pred, sigma); // <- MISTAKE 3 fixed
}

"
```

b)

Determine a suitable weakly-informative prior $normal(0;\sigma_\beta)$ for the slope beta. It is very unlikely that the mean number of drownings changes more than 50% in one year. The approximate historical mean yearly number of drownings is 138. Hence, set σ_β so that the following holds for the prior probability for $\beta : Pr(-69 < \beta < 69) = 0.99$. Determine suitable value for σ_β and report the approximate numerical value for it.

The distribution for σ_β is a normal distribution with $\mu_\beta = 0$ and the standard deviation σ_β is unknown. Because it is required that $Pr(-69 < \beta < 69) = 0.99$, it means $\beta = -69$ is the 0.5% quantile and $\beta = 69$ is the 99.5% quantile. Therefore, we need to calculate σ_β such that the cumulative distribution $CDF[0, \sigma_\beta](-69) = 0.005$ and $CDF[0, \sigma_\beta](69) = 99.95$. Since the normal distribution is symmetric around the mean, we only need to find a solution that satisfies one equation, for example $CDF[0, \sigma_\beta](-69) = 0.005$.

To find the standard deviation of a normal distribution with a mean of 0 such that the cumulative distribution function (CDF) of the normal distribution is equal to the 0.005 quantile at x = -69, we can use the qnorm() function to estimate the standard deviation σ_β by minimizing the error between the expected value -69 and the value returned by qnorm().

```
find_sigma_beta <- function(sigmaSeq) {
  beta_quantile_value = -69
  beta_quantile = 0.005
  minError = 1e9

  for (sigma in sigmaSeq){
    newError = abs(qnorm(p = beta_quantile, mean = 0, sd = sigma) - (beta_quantile_value))
    if (newError < minError){
      sigmaMin = sigma
      minError = newError
    }
  }
  sigmaMin

}

sigma <- find_sigma_beta(seq(0,100, length.out = 100000))
print("The optimal beta sigma is: ")
```

```
## [1] "The optimal beta sigma is: "
```

```
print(sigma)
```

```
## [1] 26.78727
```

Therefore, the weakly informative prior for *beta* is $P(\beta) \sim Normal(0, 26.787)$ (answer)

c)

Using the obtained σ_β, add the desired prior in the Stan code. In the report, in a separate section, indicate clearly how you carried out your prior implementation, e.g. "Added line … in block … "

```
"
data {
  int<lower=0> N; // number of data points
  vector[N] x; // observation year
  vector[N] y; // observation number of drowned
  real xpred; // prediction year
}

parameters {
  real alpha;
  real beta;
  real<lower=0> sigma; // <- MISTAKE 1 fixed
}

transformed parameters {
  vector[N] mu = alpha + beta*x;
  real mu_pred = alpha + beta * xpred;
}
```

```
model {
  beta ~ normal(0, 26.787); // Added this line in the block model code
  y ~ normal(mu, sigma);
}

generated quantities {
  real ypred = normal_rng(mu_pred, sigma);
}

"
```

Add the line “beta ~ normal(0, 26.787);” to the block model code

d)

In a similar way, add a weakly informative prior for the intercept alpha and explain how you chose the prior.

Hint! Example resulting plots for the problem, with the fixes and the desired prior applied, are shown in Figure 1. If you want, you can use these plots as a reference for testing if your modified Stan code produces similar results. However, running the inference and comparing the plots is not required.

Because if we take into account very further time (a very long time ago), there are not any people at all, so there shouldn’t be any drownings. This means that the intercept is expected to be 0. Finally, we have the equation for the mean number of drownings per year: $\alpha + \beta x = 138$. Since $\mu_b = 0$, it means that α should be accountable for the variance. So we set $\sigma_\alpha = 138$. The conclusion is $\alpha \sim N(0, 138)$

```
"
data {
  int<lower=0> N; // number of data points
  vector[N] x; // observation year
  vector[N] y; // observation number of drowned
  real xpred; // prediction year
}

parameters {
  real alpha;
  real beta;
  real<lower=0> sigma; // <- MISTAKE 1 fixed
}

transformed parameters {
  vector[N] mu = alpha + beta*x;
  real mu_pred = alpha + beta * xpred;
}

model {
  alpha ~ normal(0, 138);
  beta ~ normal(0, 26.787);
  y ~ normal(mu, sigma);
}

generated quantities {
  real ypred = normal_rng(mu_pred, sigma);
}

"
```

Add the line “alpha ~ normal(0, 138);” to the block model code

Now, we load the model

```
set_cmdstan_path("C:/Users/nguye/Documents/.cmdstan/cmdstan-2.31.0")
```

```
## CmdStan path set to: C:/Users/nguye/Documents/.cmdstan/cmdstan-2.31.0
```

```
file <- file.path("C:/Users/nguye/Desktop/BDA/assignments/assignment 7", "ex1_model_correct.stan")
model <- cmdstan_model(file)
model$compile(quiet = FALSE)
```

```
# Run MCMC using the 'sample' method
N = length(drowning$year) # The number of years
x = drowning$year  # The years (x-axis)
y = drowning$y # The number of drownings (y-axis)
xpred = 2020
drowning_data <- list(N=N, x=x, y=y, xpred=xpred)

iter_warmup <- 2000 # Number of warm up iterations
iter_sampling <- 2000 # Number of sampling iterations
chains <- 2 # Number of MCMC

fit_mcmc <- model$sample(
  data = drowning_data,
  seed = 123,
  iter_warmup = iter_warmup,
  iter_sampling = iter_sampling,
  save_warmup = TRUE, # The warmup iterations is saved for visualization below
  chains = chains,
  parallel_chains = chains
)
```

```
## Running MCMC with 2 parallel chains...
##
## Chain 1 Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 2 Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 4000 [  2%]  (Warmup)
## Chain 2 Iteration:  100 / 4000 [  2%]  (Warmup)
## Chain 1 Iteration:  200 / 4000 [  5%]  (Warmup)
## Chain 2 Iteration:  200 / 4000 [  5%]  (Warmup)
## Chain 2 Iteration:  300 / 4000 [  7%]  (Warmup)
## Chain 1 Iteration:  300 / 4000 [  7%]  (Warmup)
## Chain 2 Iteration:  400 / 4000 [ 10%]  (Warmup)
## Chain 1 Iteration:  400 / 4000 [ 10%]  (Warmup)
## Chain 2 Iteration:  500 / 4000 [ 12%]  (Warmup)
## Chain 1 Iteration:  500 / 4000 [ 12%]  (Warmup)
## Chain 2 Iteration:  600 / 4000 [ 15%]  (Warmup)
## Chain 1 Iteration:  600 / 4000 [ 15%]  (Warmup)
## Chain 2 Iteration:  700 / 4000 [ 17%]  (Warmup)
## Chain 1 Iteration:  700 / 4000 [ 17%]  (Warmup)
## Chain 2 Iteration:  800 / 4000 [ 20%]  (Warmup)
## Chain 1 Iteration:  800 / 4000 [ 20%]  (Warmup)
## Chain 2 Iteration:  900 / 4000 [ 22%]  (Warmup)
## Chain 1 Iteration:  900 / 4000 [ 22%]  (Warmup)
## Chain 2 Iteration: 1000 / 4000 [ 25%]  (Warmup)
## Chain 1 Iteration: 1000 / 4000 [ 25%]  (Warmup)
## Chain 2 Iteration: 1100 / 4000 [ 27%]  (Warmup)
## Chain 1 Iteration: 1100 / 4000 [ 27%]  (Warmup)
## Chain 2 Iteration: 1200 / 4000 [ 30%]  (Warmup)
## Chain 1 Iteration: 1200 / 4000 [ 30%]  (Warmup)
## Chain 2 Iteration: 1300 / 4000 [ 32%]  (Warmup)
## Chain 1 Iteration: 1300 / 4000 [ 32%]  (Warmup)
## Chain 2 Iteration: 1400 / 4000 [ 35%]  (Warmup)
## Chain 1 Iteration: 1400 / 4000 [ 35%]  (Warmup)
## Chain 2 Iteration: 1500 / 4000 [ 37%]  (Warmup)
## Chain 1 Iteration: 1500 / 4000 [ 37%]  (Warmup)
## Chain 2 Iteration: 1600 / 4000 [ 40%]  (Warmup)
## Chain 1 Iteration: 1600 / 4000 [ 40%]  (Warmup)
## Chain 2 Iteration: 1700 / 4000 [ 42%]  (Warmup)
## Chain 1 Iteration: 1700 / 4000 [ 42%]  (Warmup)
## Chain 2 Iteration: 1800 / 4000 [ 45%]  (Warmup)
## Chain 1 Iteration: 1800 / 4000 [ 45%]  (Warmup)
## Chain 2 Iteration: 1900 / 4000 [ 47%]  (Warmup)
## Chain 1 Iteration: 1900 / 4000 [ 47%]  (Warmup)
## Chain 2 Iteration: 2000 / 4000 [ 50%]  (Warmup)
## Chain 2 Iteration: 2001 / 4000 [ 50%]  (Sampling)
## Chain 1 Iteration: 2000 / 4000 [ 50%]  (Warmup)
## Chain 1 Iteration: 2001 / 4000 [ 50%]  (Sampling)
## Chain 2 Iteration: 2100 / 4000 [ 52%]  (Sampling)
## Chain 1 Iteration: 2100 / 4000 [ 52%]  (Sampling)
## Chain 2 Iteration: 2200 / 4000 [ 55%]  (Sampling)
## Chain 1 Iteration: 2200 / 4000 [ 55%]  (Sampling)
## Chain 2 Iteration: 2300 / 4000 [ 57%]  (Sampling)
## Chain 1 Iteration: 2300 / 4000 [ 57%]  (Sampling)
## Chain 2 Iteration: 2400 / 4000 [ 60%]  (Sampling)
## Chain 1 Iteration: 2400 / 4000 [ 60%]  (Sampling)
## Chain 2 Iteration: 2500 / 4000 [ 62%]  (Sampling)
## Chain 1 Iteration: 2500 / 4000 [ 62%]  (Sampling)
## Chain 2 Iteration: 2600 / 4000 [ 65%]  (Sampling)
## Chain 1 Iteration: 2600 / 4000 [ 65%]  (Sampling)
## Chain 2 Iteration: 2700 / 4000 [ 67%]  (Sampling)
## Chain 1 Iteration: 2700 / 4000 [ 67%]  (Sampling)
## Chain 2 Iteration: 2800 / 4000 [ 70%]  (Sampling)
## Chain 1 Iteration: 2800 / 4000 [ 70%]  (Sampling)
## Chain 2 Iteration: 2900 / 4000 [ 72%]  (Sampling)
## Chain 1 Iteration: 2900 / 4000 [ 72%]  (Sampling)
## Chain 2 Iteration: 3000 / 4000 [ 75%]  (Sampling)
## Chain 1 Iteration: 3000 / 4000 [ 75%]  (Sampling)
## Chain 2 Iteration: 3100 / 4000 [ 77%]  (Sampling)
## Chain 1 Iteration: 3100 / 4000 [ 77%]  (Sampling)
## Chain 2 Iteration: 3200 / 4000 [ 80%]  (Sampling)
## Chain 1 Iteration: 3200 / 4000 [ 80%]  (Sampling)
## Chain 2 Iteration: 3300 / 4000 [ 82%]  (Sampling)
## Chain 1 Iteration: 3300 / 4000 [ 82%]  (Sampling)
## Chain 2 Iteration: 3400 / 4000 [ 85%]  (Sampling)
## Chain 1 Iteration: 3400 / 4000 [ 85%]  (Sampling)
## Chain 2 Iteration: 3500 / 4000 [ 87%]  (Sampling)
## Chain 1 Iteration: 3500 / 4000 [ 87%]  (Sampling)
## Chain 2 Iteration: 3600 / 4000 [ 90%]  (Sampling)
## Chain 1 Iteration: 3600 / 4000 [ 90%]  (Sampling)
## Chain 2 Iteration: 3700 / 4000 [ 92%]  (Sampling)
## Chain 1 Iteration: 3700 / 4000 [ 92%]  (Sampling)
## Chain 2 Iteration: 3800 / 4000 [ 95%]  (Sampling)
## Chain 1 Iteration: 3800 / 4000 [ 95%]  (Sampling)
## Chain 2 Iteration: 3900 / 4000 [ 97%]  (Sampling)
## Chain 1 Iteration: 3900 / 4000 [ 97%]  (Sampling)
## Chain 2 Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 2 finished in 12.2 seconds.
## Chain 1 Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 1 finished in 12.5 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 12.3 seconds.
## Total execution time: 12.6 seconds.
```

```
## Warning: 3334 of 4000 (83.0%) transitions hit the maximum treedepth limit of 10.
## See https://mc-stan.org/misc/warnings for details.
```

```
fit_mcmc$summary()
```

```
## # A tibble: 46 x 10
##    variable       mean     median      sd     mad       q5     q95  rhat ess_b~1
##    <chr>         <dbl>      <dbl>   <dbl>   <dbl>    <dbl>   <dbl> <dbl>   <dbl>
##  1 lp__      172.       170.      4.85e+0 6.44e+0  1.65e+2 1.79e+2  2.41    2.44
##  2 alpha      -0.135     -0.160   1.24e+0 1.84e+0 -1.43e+0 1.19e+0  2.80    2.31
##  3 beta        1.00       1.00    6.20e-4 9.19e-4  9.99e-1 1.00e+0  2.82    2.30
##  4 sigma       0.00753    0.00724 1.45e-3 1.44e-3  5.59e-3 1.03e-2  1.67    3.21
##  5 mu[1]    1980.      1980       1.09e-2 1.48e-2  1.98e+3 1.98e+3  2.91    2.29
##  6 mu[2]    1981.      1981       1.04e-2 1.48e-2  1.98e+3 1.98e+3  3.43    2.20
##  7 mu[3]    1982.      1982       1.01e-2 1.48e-2  1.98e+3 1.98e+3  4.41    2.12
##  8 mu[4]    1983.      1983       1.00e-2 1.48e-2  1.98e+3 1.98e+3  6.77    2.05
##  9 mu[5]    1984.      1984       1.00e-2 1.48e-2  1.98e+3 1.98e+3 16.8     2.02
## 10 mu[6]    1985.      1985.      1.00e-2 7.41e-3  1.98e+3 1.99e+3 42.2     2.01
## # ... with 36 more rows, 1 more variable: ess_tail <dbl>, and abbreviated
## #   variable name 1: ess_bulk
```

Exercise 2. Hierarchical model: factory data with Stan

Note! Both Assignment 8 and 9 build upon this part of the assignment, so it is important to get this assignment correct before you start with Assignment 8 and 9. If you need help, ask TAs.

The factory data in the aaltobda package contains quality control measurements from 6 machines in a factory (units of the measurements are irrelevant here). In the data file, each column contains the measurements for a single machine. Quality control measurements are expensive and time-consuming, so only 5 measurements were done for each machine. In addition to the existing machines, we are interested in the quality of another machine (the seventh machine). To read in the data, just use:

```r
library(aaltobda)
data("factory")
```

For this problem, you'll use the following Gaussian models:
- a separate model, in which each machine has its own model
- a pooled model, in which all measurements are combined and there is no distinction between machines
- a hierarchical model, which has a hierarchical structure as described in BDA3 Section 11.6 As in the model described in the book, use the same measurement standard deviation σ for all the groups in the hierarchical model. In the separate model, however, use separate measurement standard deviation σ_j for each group j. You should use weakly informative priors for all your models.

The provided Stan code in Listing 2 given on the next page is an example of the separate model (but with very strange results, why?). This separate model can be summarized mathematically as:

$y_{ij} \sim N(\mu_j, \sigma_j)$
$\mu_j \sim N(0, 1)$
$\sigma_j \sim Inv - \chi^2(10)$

To run Stan for that model, simply use:

```r
library(cmdstanr)
data("factory")
#sm <- cmdstan_model(stan_file = "[path to stan model code]")
#stan_data <- list(
#  y = factory,
#  N = nrow(factory),
#  J = ncol(factory)
#)
#model <- sm$sample(data = stan_data, refresh=1000)
#model
```

Note! These are not the results you would expect to turn in your report. You will need to change the code for the separate model as well. For each of the three models (separate, pooled, hierarchical), your tasks are the following:

a)

Describe the model with mathematical notation (as is done for the separate model above). Also describe in words the difference between the three models.

Let y_{ij} be the measure i of the machine number j, which is between 1 and 6 in the table.
Let μ_j be the measurement mean of machine number j.
Let σ_j be the measurement variance of machine number j.

Separate model

In the separate model, the measurements of each machine come from their respective Gaussian distributions. These distributions are independent of each other between the machines. The separate model assumes that all machines are different, thus they have their own mean μ_j and variance σ_j distribution. Therefore, we cannot infer about any new machine in the separate model

Mathematical notation:
$y_{ij} \sim N(\mu_j, \sigma_j)$
$\mu_j \sim N(0, 50)$
$\sigma_j \sim N(0, 20)$

Pooled model

In the pooled model, the measurements of all machines are thought to come from a single Gaussian distribution, which is shared among all machines. The pooled model assumes that all machines are similar and all of the measurements share the same constant mean μ_j and variance σ_j prior. Therefore, we can not infer data for any new machine from this prior.

Mathematical notation:
$y_i \sim N(\mu, \sigma)$
$\mu \sim N(0, 50)$
$\sigma \sim N(0, 20)$

Hierarchical model

Regarding the hyperparameters, the measurements share a common variance distribution, but hierarchical prior for the mean, where the means of each machine is inherited from the hyperparameter τ. The hierachical model assumes that while the variance of the machines can be explained by coming from the same distribution, it makes the measurement mean more abstract by allowing all of the mean priors to be sampled from a distribution with tunable hyperparameters. This helps achieve the independent aspect of the measurements in the separate model while preserving the common grounds between the machines.

Mathematical notation:
$y_{ij} \sim N(\mu_j, \sigma_j)$
$\sigma_j \sim N(0, 20)$
$\mu_j \sim N(\mu_\tau, \sigma_\tau)$
$\mu_\tau \sim N(0, 10)$
$\sigma_\tau \sim N(0, 10)$

b)

Implement the model in Stan and include the code in the report. Use weakly informative priors for all your models.

Separate model

The chosen weakly informative priors for the separate model is:
$\mu_j \sim N(0, 50)$

$\sigma_j \sim N(0, 20)$

```
"
data {
  int<lower=0> N; // number of measurements per machine
  int<lower=0> K; // number of machines
  array[N] vector[K] y; // An array of vectors containing the table data
}

parameters {
  vector[K] mu;
  vector<lower=0>[K] sigma;
}

model {
  for (k in 1:K) {
    mu[k] ~ normal(0, 50);
    sigma[k] ~ normal(0, 20);
  }
  for (k in 1:K) {
    y[, k] ~ normal(mu[k], sigma[k]);
  }
}

generated quantities {
  array[K] real ypred;
  for (k in 1:K) {
    ypred[k] = normal_rng(mu[k], sigma[k]);
  }
}

"
```

Compiling the separate model

```
set_cmdstan_path("C:/Users/nguye/Documents/.cmdstan/cmdstan-2.31.0")
```

```
## CmdStan path set to: C:/Users/nguye/Documents/.cmdstan/cmdstan-2.31.0
```

```
file_separate <- file.path("C:/Users/nguye/Desktop/BDA/assignments/assignment 7", "ex2_model_separate.s
model_separate <- cmdstan_model(file_separate)
model_separate$compile(quiet = FALSE)
```

Pooled model

The chosen weakly informative priors for the separate model is:
$\mu \sim N(0, 50)$
$\sigma \sim N(0, 20)$

```
"
data {
  int<lower=0> N; // number of measurements per machine
  int<lower=0> K; // number of machines
  array[N] vector[K] y; // An array of vectors containing the table data
}

parameters {
  real mu; // There is only a single mu for all machines
  real<lower=0> sigma; // There is only a single sigma for all machines
}

model {
  // priors
  mu ~ normal(0, 50);
  sigma ~ normal(0, 20);

  // likelihood
  for (k in 1:K){
    y[,k] ~ normal(mu, sigma);
  }
}

generated quantities {
  real ypred = normal_rng(mu, sigma);
}

"
```

Compiling the pooled model

```
set_cmdstan_path("C:/Users/nguye/Documents/.cmdstan/cmdstan-2.31.0")
```

```
## CmdStan path set to: C:/Users/nguye/Documents/.cmdstan/cmdstan-2.31.0
```

```
file_pooled <- file.path("C:/Users/nguye/Desktop/BDA/assignments/assignment 7", "ex2_model_pooled.stan"
model_pooled <- cmdstan_model(file_pooled)
model_pooled$compile(quiet = FALSE)
```

Hierarchical model

The chosen weakly informative priors for the separate model is:
$\sigma \sim N(0, 20)$
$\mu_\tau \sim N(0, 10)$
$\sigma_\tau \sim N(0, 10)$

```
"
data {
  int<lower=0> N; // number of measurements per machine
  int<lower=0> K; // number of machines
  array[N] vector[K] y; // An array of vectors containing the table data
}

parameters {
  vector[K] mu;
  real sigma;
  real mu_tau;
  real sigma_tau;
  real mu_ypred7;
}

model {
  mu_tau ~ normal(0, 10);
  sigma_tau ~ normal(0, 10);
  sigma ~ normal(0, 20);
  mu_ypred7 ~ normal(mu_tau, sigma_tau);
  for (k in 1:K) {
    mu[k] ~ normal(mu_tau, sigma_tau);
  }
  for (k in 1:K) {
    y[, k] ~ normal(mu[k], sigma);
  }
}

generated quantities {
  array[K] real ypred;
  for (k in 1:K) {
    ypred[k] = normal_rng(mu[k] , sigma);
  }
  real ypred7 = normal_rng(mu_ypred7, sigma);
}

"
```

Compiling the hierarchical model

```
set_cmdstan_path("C:/Users/nguye/Documents/.cmdstan/cmdstan-2.31.0")
```

```
## CmdStan path set to: C:/Users/nguye/Documents/.cmdstan/cmdstan-2.31.0
```

```
file_hierarchical <- file.path("C:/Users/nguye/Desktop/BDA/assignments/assignment 7", "ex2_model_hierarc
model_hierarchical <- cmdstan_model(file_hierarchical)
model_hierarchical$compile(quiet = FALSE)
```

c)

Using the model (with weakly informative priors) report, comment on and, if applicable, plot histograms for the following distributions:
i) the posterior distribution of the mean of the quality measurements of the sixth machine.

ii) the predictive distribution for another quality measurement of the sixth machine.
iii) the posterior distribution of the mean of the quality measurements of the seventh machine.

Separate model

Preparing the data

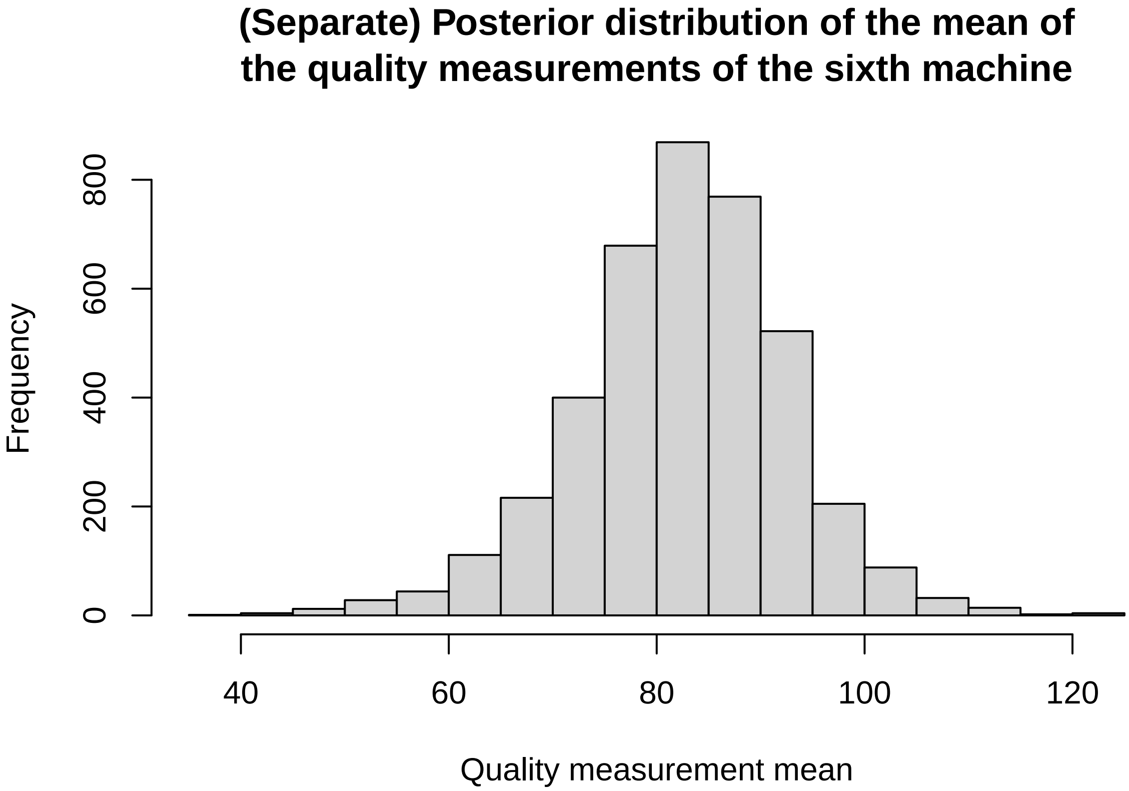
```
stan_data <- list(
  y = factory,
  N = nrow(factory),
  K = ncol(factory)
)
file_separate <- file.path("C:/Users/nguye/Desktop/BDA/assignments/assignment 7", "ex2_model_separate.s
model_separate <- cmdstan_model(file_separate)
model_separate$compile(quiet = FALSE)
result_separate <- model_separate$sample(data = stan_data, refresh=1000)
```

```
## Running MCMC with 4 sequential chains...
##
## Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1 finished in 0.9 seconds.
## Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2 finished in 0.9 seconds.
## Chain 3 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3 finished in 0.8 seconds.
## Chain 4 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4 finished in 0.8 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 0.9 seconds.
## Total execution time: 3.9 seconds.
```

```
#result_separate$summary()
```

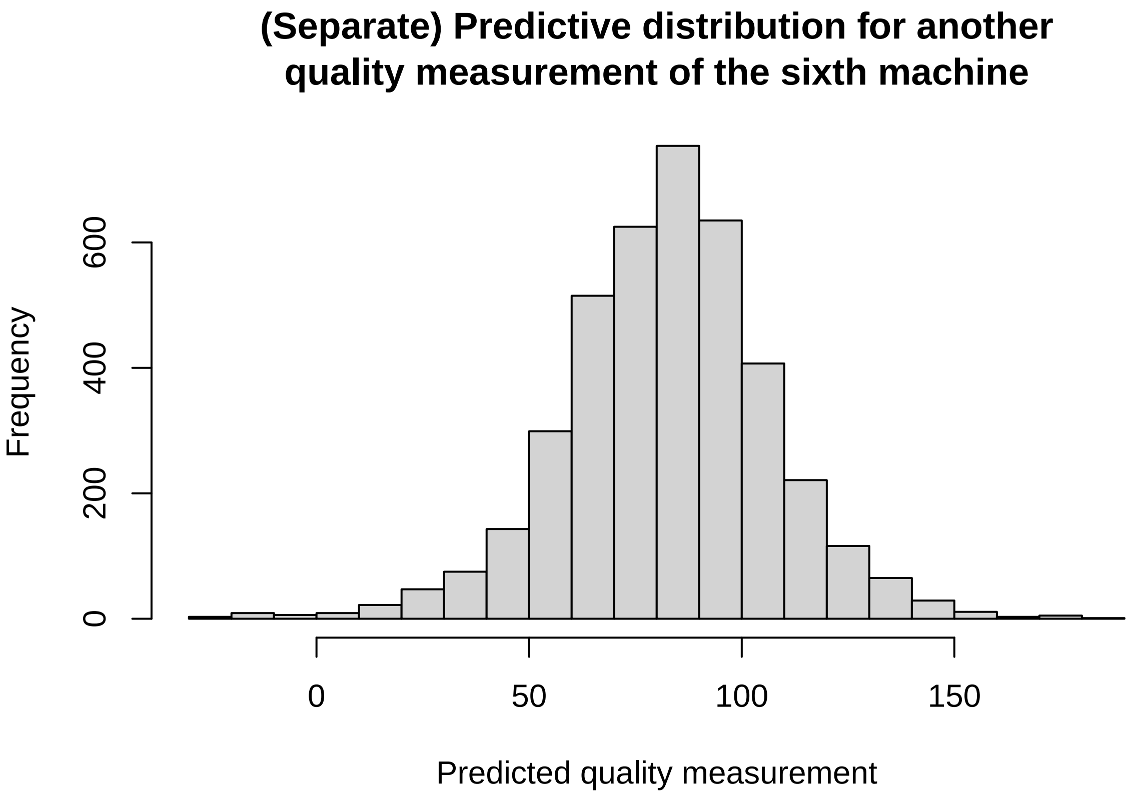
- i) the posterior distribution of the mean of the quality measurements of the sixth machine.

```
hist(result_separate$draws("mu[6]"), main="(Separate) Posterior distribution of the mean of\nthe quality
```



- ii) the predictive distribution for another quality measurement of the sixth machine.

```
hist(result_separate$draws("ypred[6]"), main="(Separate) Predictive distribution for another\nquality me
```



- iii) the posterior distribution of the mean of the quality measurements of the seventh machine.

In the separate model, the machines are independent of each other. Knowing the posterior distribution of the quality measurements of the six machines do not give any knowledge about the seventh machine. Therefore, it is infeasible to plot the posterior distribution of the mean of the quality measurements of the seventh machine in the separate model.

Pooled model

Preparing the data

```
stan_data <- list(
  y = factory,
  N = nrow(factory),
  K = ncol(factory)
)
file_pooled <- file.path("C:/Users/nguye/Desktop/BDA/assignments/assignment 7", "ex2_model_pooled.stan"
model_pooled <- cmdstan_model(file_pooled)
model_pooled$compile(quiet = FALSE)
result_pooled <- model_pooled$sample(data = stan_data, refresh=1000)
```

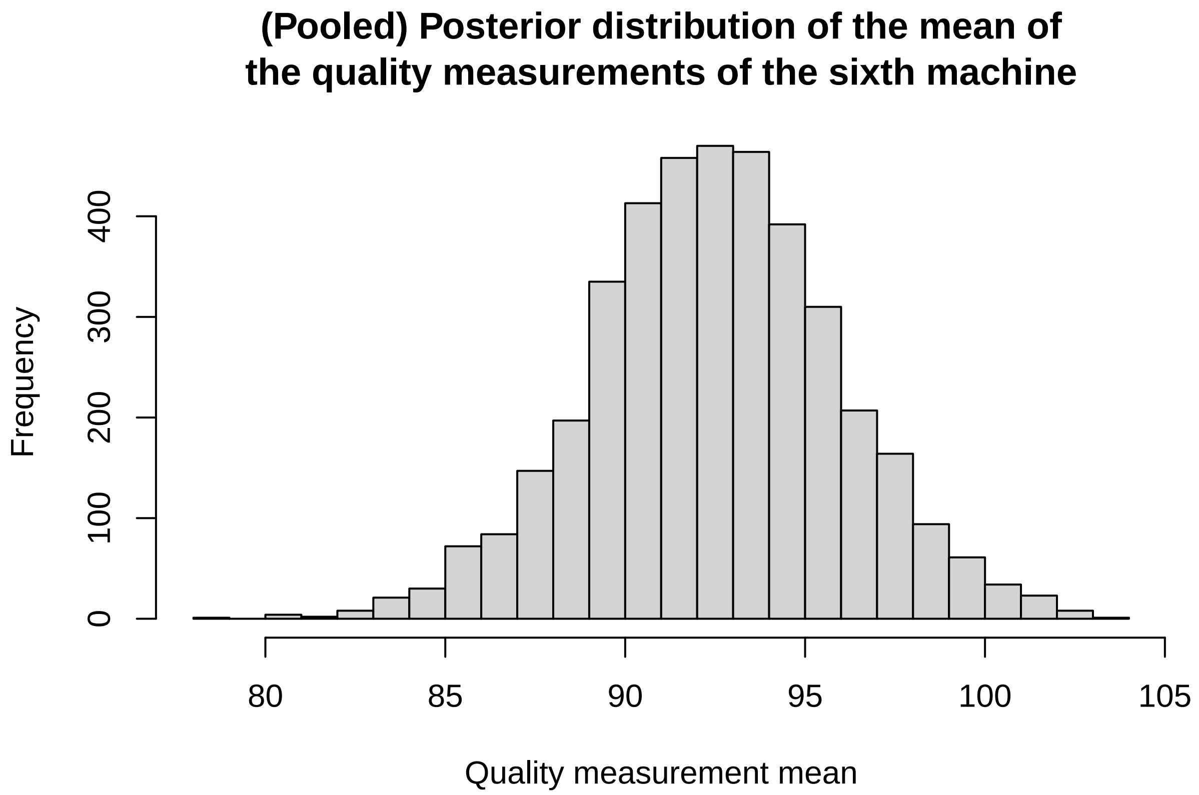
```
## Running MCMC with 4 sequential chains...
##
```

```
## Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1 finished in 0.3 seconds.
## Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2 finished in 0.3 seconds.
## Chain 3 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3 finished in 0.4 seconds.
## Chain 4 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4 finished in 0.3 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 0.3 seconds.
## Total execution time: 1.7 seconds.
```

```
#result_pooled$summary()
```

- i) the posterior distribution of the mean of the quality measurements of the sixth machine.
  A note is that this histogram is shared among all machines

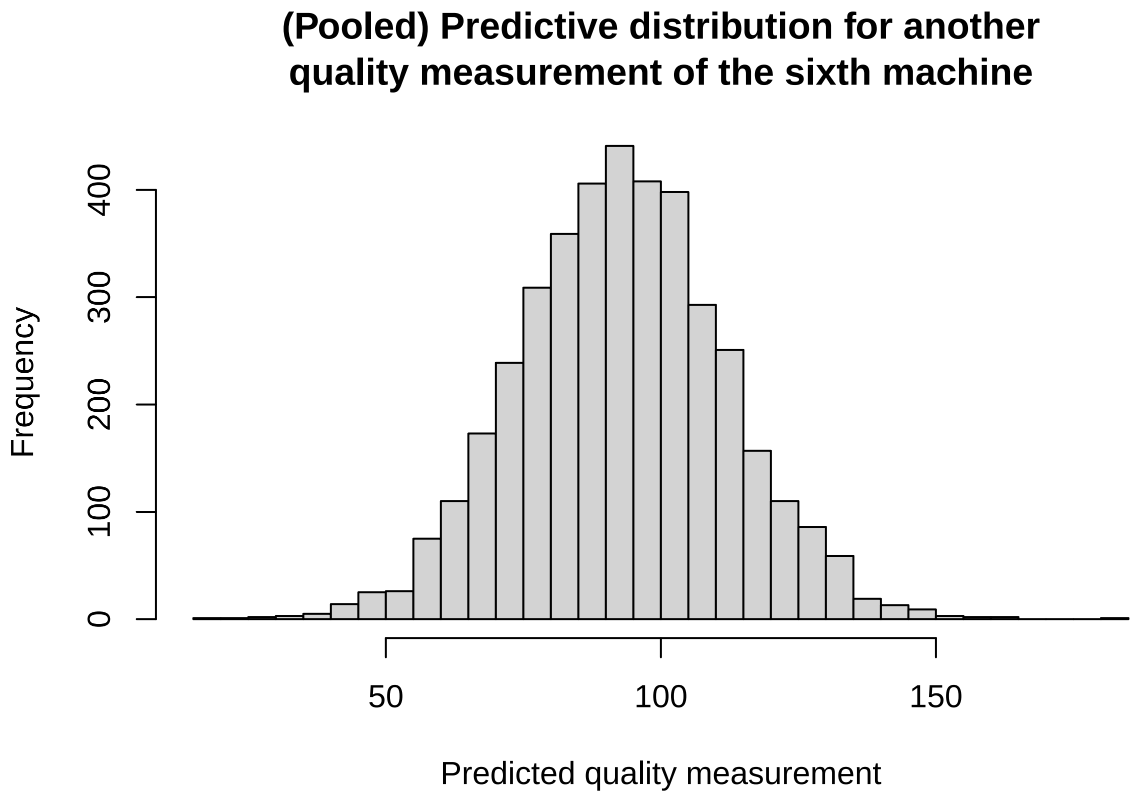
```
hist(result_pooled$draws("mu"), main="(Pooled) Posterior distribution of the mean of\nthe quality measu
```



* ii) the predictive distribution for another quality measurement of the sixth machine.

A note is that this histogram is shared among all machines

```
hist(result_pooled$draws("ypred"), main="(Pooled) Predictive distribution for another\nquality measurem
```



* iii) the posterior distribution of the mean of the quality measurements of the seventh machine.

In the pooled model, all of the quality measurements come from the same distribution, which means there is no difference between the machines. As a result, the posterior distribution of the mean of the quality measurements of the seventh machine is similar to that of other machines, which is already plotted in part (i).

Hierarchical model

Preparing the data

```
stan_data <- list(
  y = factory,
  N = nrow(factory),
  K = ncol(factory)
)
file_hierarchical <- file.path("C:/Users/nguye/Desktop/BDA/assignments/assignment 7", "ex2_model_hierar
model_hierarchical <- cmdstan_model(file_hierarchical)
model_hierarchical$compile(quiet = FALSE)
result_hierarchical <- model_hierarchical$sample(data = stan_data, refresh=1000)
```

```
## Running MCMC with 4 sequential chains...
```

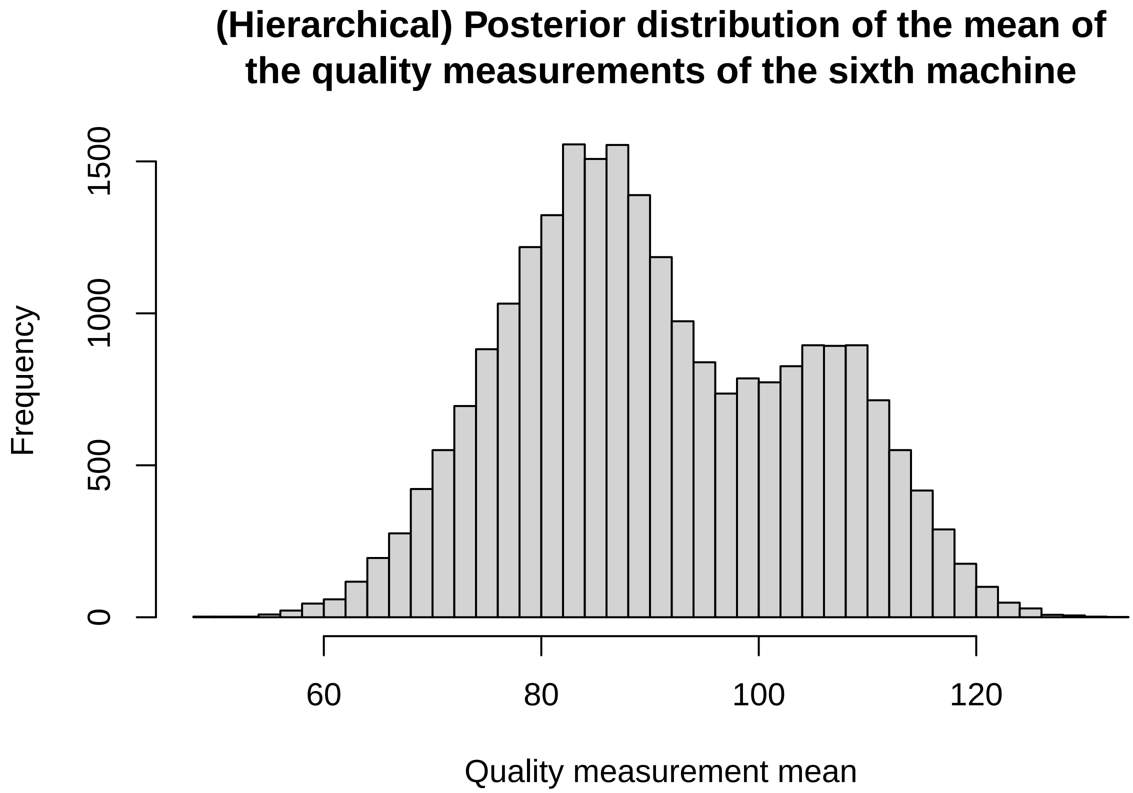
```
## Chain 1 Rejecting initial value:
```

```
## Chain 1   Error evaluating the log probability at the initial value.
## Chain 1 Exception: normal_lpdf: Scale parameter is -0.695185, but must be positive! (in ’C:/Users/ngu
## Chain 1 Exception: normal_lpdf: Scale parameter is -0.695185, but must be positive! (in ’C:/Users/ngu

## Chain 1 Rejecting initial value:

## Chain 1   Error evaluating the log probability at the initial value.

## Chain 1 Exception: normal_lpdf: Scale parameter is -1.77745, but must be positive! (in ’C:/Users/nguy
## Chain 1 Exception: normal_lpdf: Scale parameter is -1.77745, but must be positive! (in ’C:/Users/nguy

## Chain 1 Rejecting initial value:

## Chain 1   Error evaluating the log probability at the initial value.

## Chain 1 Exception: normal_lpdf: Scale parameter is -1.1686, but must be positive! (in ’C:/Users/nguye
## Chain 1 Exception: normal_lpdf: Scale parameter is -1.1686, but must be positive! (in ’C:/Users/nguye

## Chain 1 Rejecting initial value:

## Chain 1   Error evaluating the log probability at the initial value.

## Chain 1 Exception: normal_lpdf: Scale parameter is -0.13843, but must be positive! (in ’C:/Users/nguy
## Chain 1 Exception: normal_lpdf: Scale parameter is -0.13843, but must be positive! (in ’C:/Users/nguy

## Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)

## Chain 1 Informational Message: The current Metropolis proposal is about to be rejected because of the

## Chain 1 Exception: normal_lpdf: Scale parameter is -0.239611, but must be positive! (in ’C:/Users/ngu

## Chain 1 If this warning occurs sporadically, such as for highly constrained variable types like covar

## Chain 1 but if this warning occurs often then your model may be either severely ill-conditioned or mi

## Chain 1

## Chain 1 Informational Message: The current Metropolis proposal is about to be rejected because of the

## Chain 1 Exception: normal_lpdf: Scale parameter is -83.2185, but must be positive! (in ’C:/Users/nguy

## Chain 1 If this warning occurs sporadically, such as for highly constrained variable types like covar

## Chain 1 but if this warning occurs often then your model may be either severely ill-conditioned or mi

## Chain 1

## Chain 1 Informational Message: The current Metropolis proposal is about to be rejected because of the
## Chain 1 Exception: normal_lpdf: Scale parameter is -70.0048, but must be positive! (in ’C:/Users/nguy

## Chain 1 If this warning occurs sporadically, such as for highly constrained variable types like cova

## Chain 1 but if this warning occurs often then your model may be either severely ill-conditioned or m

## Chain 1

## Chain 1 Informational Message: The current Metropolis proposal is about to be rejected because of th

## Chain 1 Exception: normal_lpdf: Scale parameter is -15.8477, but must be positive! (in ’C:/Users/nguy

## Chain 1 If this warning occurs sporadically, such as for highly constrained variable types like cova

## Chain 1 but if this warning occurs often then your model may be either severely ill-conditioned or m

## Chain 1

## Chain 1 Informational Message: The current Metropolis proposal is about to be rejected because of th

## Chain 1 Exception: normal_lpdf: Scale parameter is -8.76515, but must be positive! (in ’C:/Users/nguy

## Chain 1 If this warning occurs sporadically, such as for highly constrained variable types like cova

## Chain 1 but if this warning occurs often then your model may be either severely ill-conditioned or m

## Chain 1

## Chain 1 Iteration: 1000 / 2000 [ 50%]  (Warmup)

## Chain 1 Informational Message: The current Metropolis proposal is about to be rejected because of th

## Chain 1 Exception: normal_lpdf: Scale parameter is -105.64, but must be positive! (in ’C:/Users/nguy

## Chain 1 If this warning occurs sporadically, such as for highly constrained variable types like cova

## Chain 1 but if this warning occurs often then your model may be either severely ill-conditioned or m

## Chain 1

## Chain 1 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1 finished in 0.5 seconds.
## Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2 Iteration: 1000 / 2000 [ 50%]  (Warmup)

## Chain 2 Informational Message: The current Metropolis proposal is about to be rejected because of th
## Chain 2 Exception: normal_lpdf: Scale parameter is -21.9903, but must be positive! (in ’C:/Users/nguy

## Chain 2 If this warning occurs sporadically, such as for highly constrained variable types like covar

## Chain 2 but if this warning occurs often then your model may be either severely ill-conditioned or mi

## Chain 2

## Chain 2 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2 Iteration: 2000 / 2000 [100%]  (Sampling)

## Chain 2 Informational Message: The current Metropolis proposal is about to be rejected because of the

## Chain 2 Exception: normal_lpdf: Scale parameter is -130.462, but must be positive! (in ’C:/Users/nguy

## Chain 2 If this warning occurs sporadically, such as for highly constrained variable types like covar

## Chain 2 but if this warning occurs often then your model may be either severely ill-conditioned or mi

## Chain 2

## Chain 2 finished in 0.5 seconds.

## Chain 3 Rejecting initial value:

## Chain 3   Error evaluating the log probability at the initial value.

## Chain 3 Exception: normal_lpdf: Scale parameter is -1.83258, but must be positive! (in ’C:/Users/nguy
## Chain 3 Exception: normal_lpdf: Scale parameter is -1.83258, but must be positive! (in ’C:/Users/nguy

## Chain 3 Rejecting initial value:

## Chain 3   Error evaluating the log probability at the initial value.

## Chain 3 Exception: normal_lpdf: Scale parameter is -1.17256, but must be positive! (in ’C:/Users/nguy
## Chain 3 Exception: normal_lpdf: Scale parameter is -1.17256, but must be positive! (in ’C:/Users/nguy

## Chain 3 Rejecting initial value:

## Chain 3   Error evaluating the log probability at the initial value.

## Chain 3 Exception: normal_lpdf: Scale parameter is -1.1141, but must be positive! (in ’C:/Users/nguye
## Chain 3 Exception: normal_lpdf: Scale parameter is -1.1141, but must be positive! (in ’C:/Users/nguye

## Chain 3 Rejecting initial value:

## Chain 3   Error evaluating the log probability at the initial value.
## Chain 3 Exception: normal_lpdf: Scale parameter is -0.353553, but must be positive! (in 'C:/Users/ngu
## Chain 3 Exception: normal_lpdf: Scale parameter is -0.353553, but must be positive! (in 'C:/Users/ngu

## Chain 3 Rejecting initial value:

## Chain 3   Error evaluating the log probability at the initial value.

## Chain 3 Exception: normal_lpdf: Scale parameter is -0.624828, but must be positive! (in 'C:/Users/ngu
## Chain 3 Exception: normal_lpdf: Scale parameter is -0.624828, but must be positive! (in 'C:/Users/ngu

## Chain 3 Rejecting initial value:

## Chain 3   Error evaluating the log probability at the initial value.

## Chain 3 Exception: normal_lpdf: Scale parameter is -1.37229, but must be positive! (in 'C:/Users/nguy
## Chain 3 Exception: normal_lpdf: Scale parameter is -1.37229, but must be positive! (in 'C:/Users/nguy

## Chain 3 Rejecting initial value:

## Chain 3   Error evaluating the log probability at the initial value.

## Chain 3 Exception: normal_lpdf: Scale parameter is -0.643179, but must be positive! (in 'C:/Users/ngu
## Chain 3 Exception: normal_lpdf: Scale parameter is -0.643179, but must be positive! (in 'C:/Users/ngu

## Chain 3 Iteration:    1 / 2000 [  0%]  (Warmup)

## Chain 3 Informational Message: The current Metropolis proposal is about to be rejected because of the

## Chain 3 Exception: normal_lpdf: Scale parameter is -0.214355, but must be positive! (in 'C:/Users/ngu

## Chain 3 If this warning occurs sporadically, such as for highly constrained variable types like covar

## Chain 3 but if this warning occurs often then your model may be either severely ill-conditioned or mi

## Chain 3

## Chain 3 Informational Message: The current Metropolis proposal is about to be rejected because of the

## Chain 3 Exception: normal_lpdf: Scale parameter is -238.832, but must be positive! (in 'C:/Users/nguy

## Chain 3 If this warning occurs sporadically, such as for highly constrained variable types like covar

## Chain 3 but if this warning occurs often then your model may be either severely ill-conditioned or mi

## Chain 3

## Chain 3 Informational Message: The current Metropolis proposal is about to be rejected because of the
## Chain 3 Exception: normal_lpdf: Scale parameter is -2.50533, but must be positive! (in ’C:/Users/nguy

## Chain 3 If this warning occurs sporadically, such as for highly constrained variable types like covar

## Chain 3 but if this warning occurs often then your model may be either severely ill-conditioned or mi

## Chain 3

## Chain 3 Informational Message: The current Metropolis proposal is about to be rejected because of the

## Chain 3 Exception: normal_lpdf: Scale parameter is -12.3541, but must be positive! (in ’C:/Users/nguy

## Chain 3 If this warning occurs sporadically, such as for highly constrained variable types like covar

## Chain 3 but if this warning occurs often then your model may be either severely ill-conditioned or mi

## Chain 3

## Chain 3 Iteration: 1000 / 2000 [ 50%]  (Warmup)

## Chain 3 Informational Message: The current Metropolis proposal is about to be rejected because of the

## Chain 3 Exception: normal_lpdf: Scale parameter is -71.6128, but must be positive! (in ’C:/Users/nguy

## Chain 3 If this warning occurs sporadically, such as for highly constrained variable types like covar

## Chain 3 but if this warning occurs often then your model may be either severely ill-conditioned or mi

## Chain 3

## Chain 3 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3 finished in 0.5 seconds.

## Chain 4 Rejecting initial value:

## Chain 4   Error evaluating the log probability at the initial value.

## Chain 4 Exception: normal_lpdf: Scale parameter is -1.08954, but must be positive! (in ’C:/Users/nguy
## Chain 4 Exception: normal_lpdf: Scale parameter is -1.08954, but must be positive! (in ’C:/Users/nguy

## Chain 4 Rejecting initial value:

## Chain 4   Error evaluating the log probability at the initial value.

## Chain 4 Exception: normal_lpdf: Scale parameter is -0.60745, but must be positive! (in ’C:/Users/nguy
## Chain 4 Exception: normal_lpdf: Scale parameter is -0.60745, but must be positive! (in ’C:/Users/nguy
## Chain 4 Rejecting initial value:

## Chain 4   Error evaluating the log probability at the initial value.

## Chain 4 Exception: normal_lpdf: Scale parameter is -1.25162, but must be positive! (in ’C:/Users/nguy
## Chain 4 Exception: normal_lpdf: Scale parameter is -1.25162, but must be positive! (in ’C:/Users/nguy

## Chain 4 Iteration:    1 / 2000 [  0%]  (Warmup)

## Chain 4 Informational Message: The current Metropolis proposal is about to be rejected because of the

## Chain 4 Exception: normal_lpdf: Scale parameter is -30.1425, but must be positive! (in ’C:/Users/nguy

## Chain 4 If this warning occurs sporadically, such as for highly constrained variable types like covar

## Chain 4 but if this warning occurs often then your model may be either severely ill-conditioned or mi

## Chain 4

## Chain 4 Informational Message: The current Metropolis proposal is about to be rejected because of the

## Chain 4 Exception: normal_lpdf: Scale parameter is -1.92249, but must be positive! (in ’C:/Users/nguy

## Chain 4 If this warning occurs sporadically, such as for highly constrained variable types like covar

## Chain 4 but if this warning occurs often then your model may be either severely ill-conditioned or mi

## Chain 4

## Chain 4 Informational Message: The current Metropolis proposal is about to be rejected because of the

## Chain 4 Exception: normal_lpdf: Scale parameter is -0.123652, but must be positive! (in ’C:/Users/ngu

## Chain 4 If this warning occurs sporadically, such as for highly constrained variable types like covar

## Chain 4 but if this warning occurs often then your model may be either severely ill-conditioned or mi

## Chain 4

## Chain 4 Informational Message: The current Metropolis proposal is about to be rejected because of the

## Chain 4 Exception: normal_lpdf: Scale parameter is -60.3159, but must be positive! (in ’C:/Users/nguy

## Chain 4 If this warning occurs sporadically, such as for highly constrained variable types like covar

## Chain 4 but if this warning occurs often then your model may be either severely ill-conditioned or mi

## Chain 4
## Chain 4 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4 Iteration: 1001 / 2000 [ 50%]  (Sampling)

## Chain 4 Informational Message: The current Metropolis proposal is about to be rejected because of the

## Chain 4 Exception: normal_lpdf: Scale parameter is -57.9632, but must be positive! (in 'C:/Users/nguy

## Chain 4 If this warning occurs sporadically, such as for highly constrained variable types like cova

## Chain 4 but if this warning occurs often then your model may be either severely ill-conditioned or mi

## Chain 4

## Chain 4 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4 finished in 0.5 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 0.5 seconds.
## Total execution time: 2.4 seconds.
```

```
#result_hierarchical$summary()
```

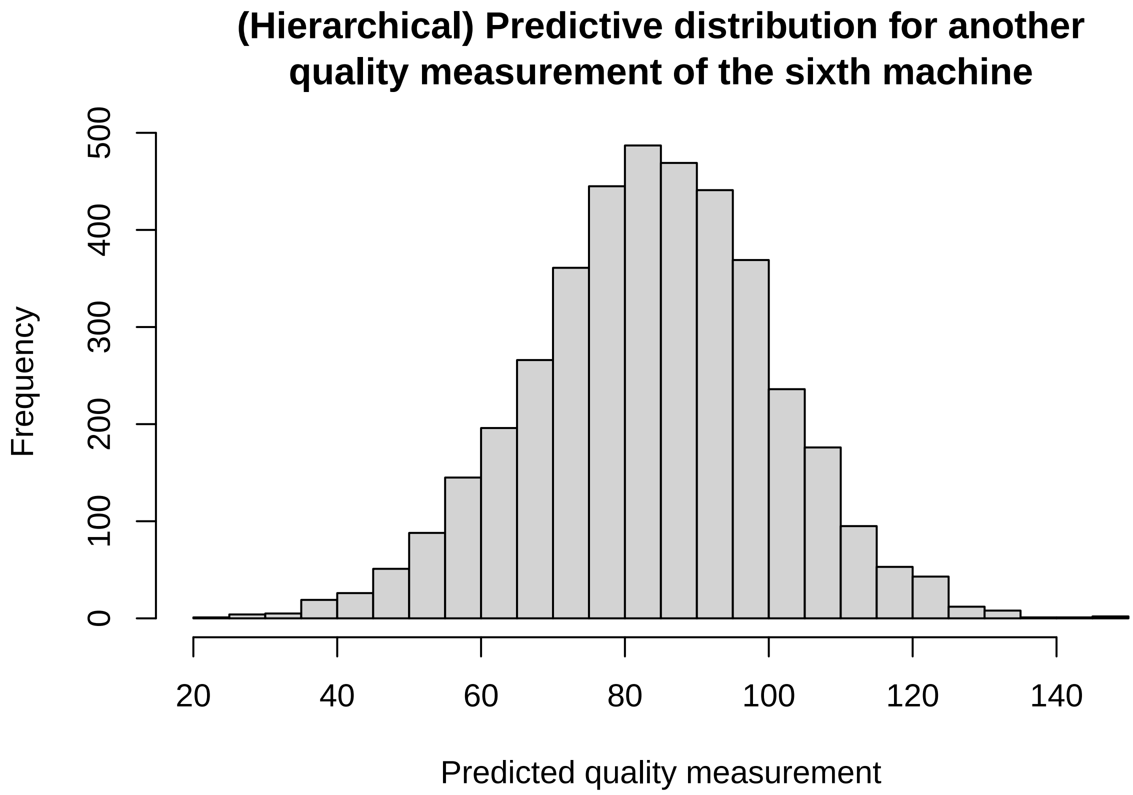
- i) the posterior distribution of the mean of the quality measurements of the sixth machine.

```
hist(result_hierarchical$draws("mu"), main="(Hierarchical) Posterior distribution of the mean of\nthe qu
```



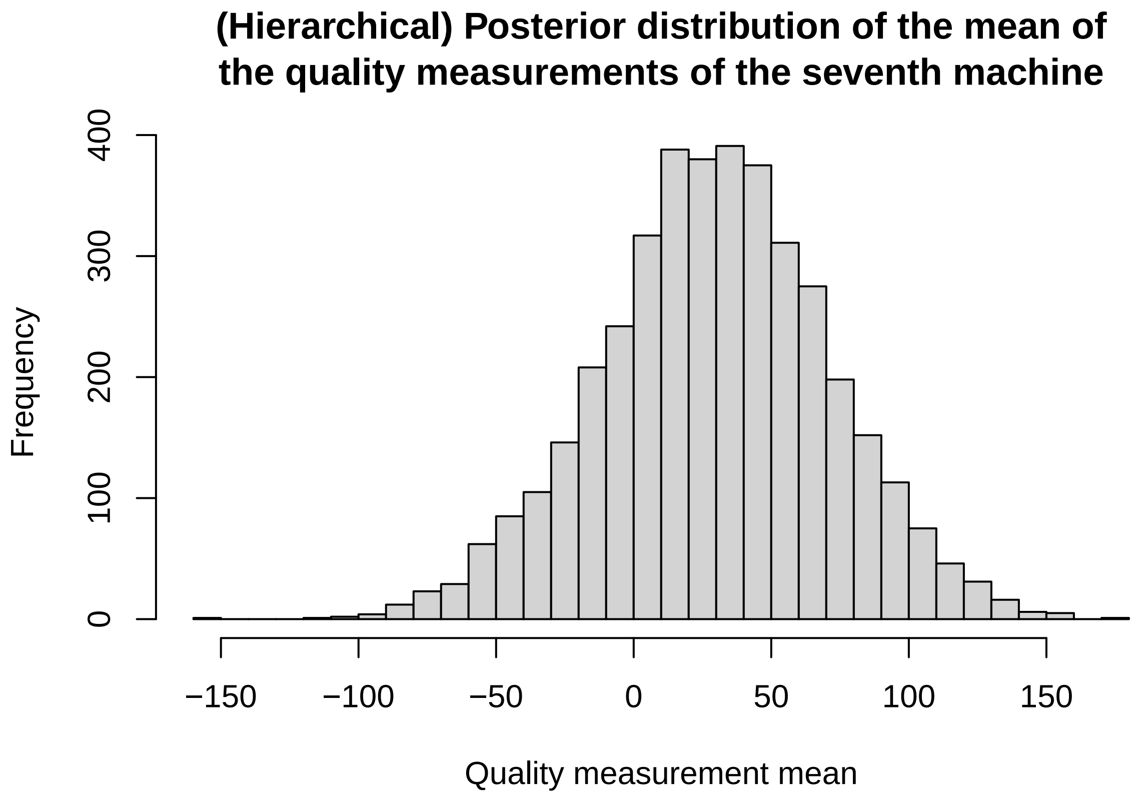
- ii) the predictive distribution for another quality measurement of the sixth machine.

```
hist(result_hierarchical$draws("ypred[6]"), main="(Hierarchical) Predictive distribution for another\nqu
```



- iii) the posterior distribution of the mean of the quality measurements of the seventh machine.

```
hist(result_hierarchical$draws("ypred7"), main="(Hierarchical) Posterior distribution of the mean of\nth
```



d)

Report the posterior expectation for μ_1 with a 90% credible interval but using a normal(0; 10) prior for the μ parameter(s) and a Gamma(1; 1) prior for the σ parameter(s). For the hierarchical model, use the normal(0; 10) and Gamma(1; 1) as hyper-priors. Hint! See the example Stan codes here for the comparison of k groups with and without the hierarchical structure.

The function used to obtain the 90% quantile

```r
ninetyPercentCredibleInterval <- function(dist, model) {
  meanDist <- mean(dist)
  lowerQuantile <- quantile(dist, 0.05)
  upperQuantile <- quantile(dist, 0.95)
  cat("The 90% credible interval of the posterior distribution for mu1 in the",model,"model")
  cat("\nThe mean:", meanDist)
  cat("\nThe 5% lower quantile:", lowerQuantile)
  cat("\nThe 95% upper quantil:", upperQuantile)
}
```

Separate model

```
"
data {
  int<lower=0> N; // number of measurements per machine
  int<lower=0> K; // number of machines
  array[N] vector[K] y; // An array of vectors containing the table data
}

parameters {
  vector[K] mu;
  vector<lower=0>[K] sigma;
}

model {
  for (k in 1:K) {
    mu[k] ~ normal(0, 10);
    sigma[k] ~ gamma(1, 1);
  }
  for (k in 1:K) {
    y[, k] ~ normal(mu[k], sigma[k]);
  }
}

generated quantities {
  array[K] real ypred;
  for (k in 1:K) {
    ypred[k] = normal_rng(mu[k], sigma[k]);
  }
}

"
```

```
stan_data <- list(
  y = factory,
  N = nrow(factory),
  K = ncol(factory)
)
file_separate_partd <- file.path("C:/Users/nguye/Desktop/BDA/assignments/assignment 7", "ex2_model_sepa
model_separate_partd <- cmdstan_model(file_separate_partd)
model_separate_partd$compile(quiet = FALSE)
result_separate_partd <- model_separate_partd$sample(data = stan_data, show_messages=FALSE)
```

```
## Running MCMC with 4 sequential chains...
##
## Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 1 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 1 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 1 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 1 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 1 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 1 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 1 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 1 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 1 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1 finished in 1.8 seconds.
## Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 2 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 2 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 2 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 2 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 2 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 2 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 2 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 2 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 2 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 2 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2 finished in 1.8 seconds.
## Chain 3 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 3 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 3 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 3 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 3 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 3 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 3 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 3 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 3 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 3 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 3 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3 finished in 1.8 seconds.
## Chain 4 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 4 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 4 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 4 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 4 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 4 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 4 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 4 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 4 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 4 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 4 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4 finished in 1.8 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 1.8 seconds.
## Total execution time: 7.7 seconds.
```

```
ninetyPercentCredibleInterval(result_separate_partd$draws("mu[1]"),"separate")
```

```
## The 90% credible interval of the posterior distribution for mu1 in the separate model
## The mean: 50.51723
## The 5% lower quantile: 34.60888
## The 95% upper quantil: 63.96723
```

Pooled model

```
"
data {
  int<lower=0> N; // number of measurements per machine
  int<lower=0> K; // number of machines
  array[N] vector[K] y; // An array of vectors containing the table data
}

parameters {
  real mu; // There is only a single mu for all machines
  real<lower=0> sigma; // There is only a single sigma for all machines
}

model {
  // priors
  mu ~ normal(0, 10);
  sigma ~ gamma(1, 1);

  // likelihood
  for (k in 1:K){
    y[,k] ~ normal(mu, sigma);
  }
}

generated quantities {
  real ypred = normal_rng(mu, sigma);
}

"
```

```
stan_data <- list(
  y = factory,
  N = nrow(factory),
  K = ncol(factory)
)
file_pooled_partd <- file.path("C:/Users/nguye/Desktop/BDA/assignments/assignment 7", "ex2_model_pooled_
model_pooled_partd <- cmdstan_model(file_pooled_partd)
model_pooled_partd$compile(quiet = FALSE)
result_pooled_partd <- model_pooled_partd$sample(data = stan_data, show_messages=FALSE)
```

```
## Running MCMC with 4 sequential chains...
##
## Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 1 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 1 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 1 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 1 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 1 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 1 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 1 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 1 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 1 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1 finished in 0.3 seconds.
## Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 2 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 2 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 2 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 2 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 2 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 2 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 2 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 2 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 2 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 2 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2 finished in 0.3 seconds.
## Chain 3 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 3 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 3 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 3 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 3 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 3 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 3 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 3 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 3 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 3 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 3 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3 finished in 0.3 seconds.
## Chain 4 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 4 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 4 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 4 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 4 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 4 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 4 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 4 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 4 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 4 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 4 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4 finished in 0.3 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 0.3 seconds.
## Total execution time: 1.5 seconds.
```

```r
ninetyPercentCredibleInterval(result_pooled_partd$draws("mu"), "pooled")
```

```
## The 90% credible interval of the posterior distribution for mu1 in the pooled model
## The mean: 85.45027
## The 5% lower quantile: 79.82337
## The 95% upper quantil: 90.33901
```

Hierarchical model

```
"
data {
  int<lower=0> N; // number of measurements per machine
  int<lower=0> K; // number of machines
  array[N] vector[K] y; // An array of vectors containing the table data
}

parameters {
  vector[K] mu;
  real sigma;
  real mu_tau;
  real sigma_tau;
  real mu_ypred7;
}

model {
  mu_tau ~ normal(0, 10);
  sigma_tau ~ gamma(1, 1);
  sigma ~ gamma(1, 1);
  mu_ypred7 ~ normal(mu_tau, sigma_tau);
  for (k in 1:K) {
    mu[k] ~ normal(mu_tau, sigma_tau);
  }
  for (k in 1:K) {
    y[, k] ~ normal(mu[k], sigma);
  }
}

generated quantities {
  array[K] real ypred;
  for (k in 1:K) {
    ypred[k] = normal_rng(mu[k] , sigma);
  }
  real ypred7 = normal_rng(mu_ypred7, sigma);
}

"
```

```
stan_data <- list(
  y = factory,
  N = nrow(factory),
  K = ncol(factory)
)
file_hierarchical_partd <- file.path("C:/Users/nguye/Desktop/BDA/assignments/assignment 7", "ex2_model_
model_hierarchical_partd <- cmdstan_model(file_hierarchical_partd)
model_hierarchical_partd$compile(quiet = FALSE)
result_hierarchical_partd <- model_hierarchical_partd$sample(data = stan_data, show_messages=FALSE)
```

```
## Running MCMC with 4 sequential chains...

## Chain 1 Rejecting initial value:

## Chain 1   Error evaluating the log probability at the initial value.

## Chain 1 Exception: gamma_lpdf: Random variable is -1.55486, but must be positive finite! (in 'C:/Use
## Chain 1 Exception: gamma_lpdf: Random variable is -1.55486, but must be positive finite! (in 'C:/Use

## Chain 1 Rejecting initial value:

## Chain 1   Error evaluating the log probability at the initial value.

## Chain 1 Exception: gamma_lpdf: Random variable is -0.469358, but must be positive finite! (in 'C:/Us
## Chain 1 Exception: gamma_lpdf: Random variable is -0.469358, but must be positive finite! (in 'C:/Us

## Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 1 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 1 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 1 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 1 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 1 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 1 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 1 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 1 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 1 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1 finished in 1.1 seconds.

## Chain 2 Rejecting initial value:

## Chain 2   Error evaluating the log probability at the initial value.

## Chain 2 Exception: gamma_lpdf: Random variable is -0.0826051, but must be positive finite! (in ’C:/U
## Chain 2 Exception: gamma_lpdf: Random variable is -0.0826051, but must be positive finite! (in ’C:/U

## Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 2 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 2 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 2 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 2 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 2 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 2 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 2 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 2 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 2 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 2 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2 finished in 1.0 seconds.

## Chain 3 Rejecting initial value:

## Chain 3   Error evaluating the log probability at the initial value.
## Chain 3 Exception: gamma_lpdf: Random variable is -0.0502089, but must be positive finite! (in 'C:/Us
## Chain 3 Exception: gamma_lpdf: Random variable is -0.0502089, but must be positive finite! (in 'C:/Us

## Chain 3 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 3 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 3 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 3 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 3 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 3 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 3 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 3 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 3 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 3 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 3 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3 finished in 1.2 seconds.
## Chain 4 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 4 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 4 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 4 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 4 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 4 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 4 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 4 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 4 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 4 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 4 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4 finished in 1.0 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 1.1 seconds.
## Total execution time: 4.7 seconds.
## Warning: 32 of 4000 (1.0%) transitions ended with a divergence.
## See https://mc-stan.org/misc/warnings for details.
```

```
ninetyPercentCredibleInterval(result_hierarchical_partd$draws("mu[1]"), "hierarchical")
```

```
## The 90% credible interval of the posterior distribution for mu1 in the hierarchical model
## The mean: 75.39354
## The 5% lower quantile: 66.6644
## The 95% upper quantil: 84.51516
```