BDA - Assignment 7

Anonymous

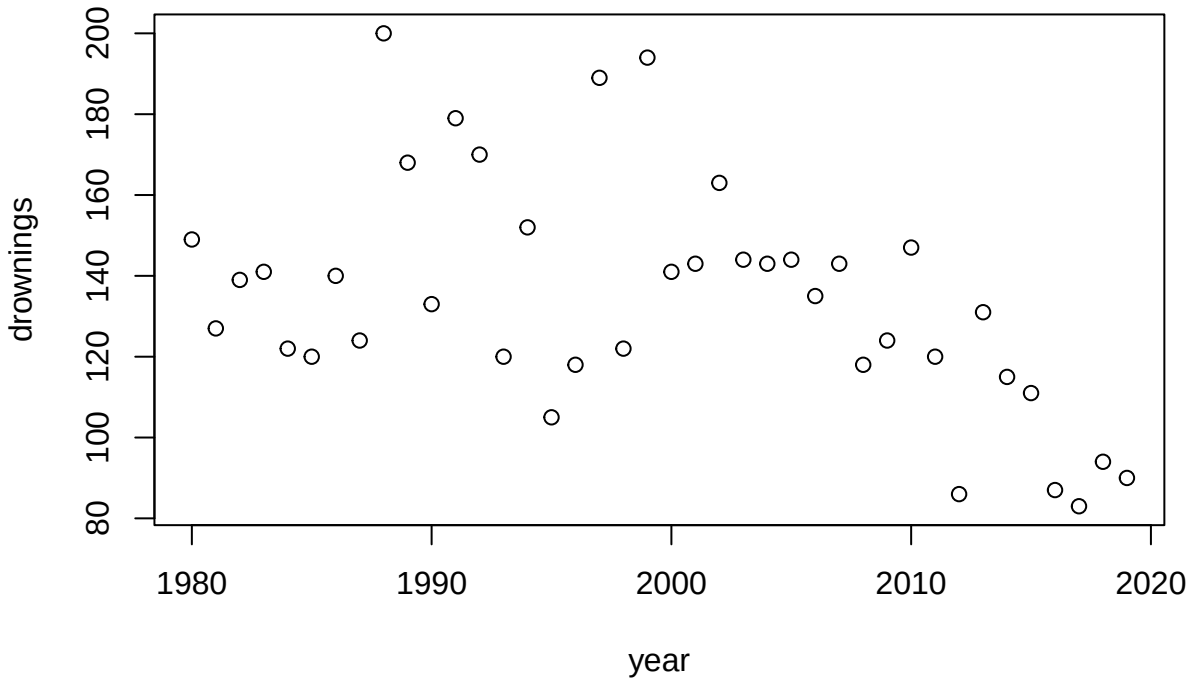
Contents


```
# Loading libraries and data
library(cmdstanr)
options(mc.cores = parallel::detectCores()) # Use all core available
library(posterior)
library(ggplot2)
library(loo)
library(tidyr)
library(dplyr)
library(ggdist)
library(rprojroot)
library(bayesplot)
library(aaltobda)
options(pillar.neg=FALSE)
library(gridExtra)
theme_set(bayesplot::theme_default(base_family = "sans"))
library(shinystan)
library(extraDistr)
data("drowning")
data("factory")
seed <- 5
set.seed(seed)
```

1. Linear model: drowning data with Stan

Plotting the data

```
plot(drowning)
```



a)

Corrections in:

- parameters: sigma must be non-negative, upper changed to lower
- model: added line break ; at the end of y
- generated quantities: changed ypred formulation from using previous mu to use alpha, beta and xpred as the mean.

```
writeLines(readLines("Correct_drowning_model.stan"))
```

```
## 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; // upper to lower
## }
```

```
##
## transformed parameters {
## vector[N] mu = alpha + beta*x;
## }
##
## model {//
## y ~ normal(mu, sigma); // added ;
## }
##
## generated quantities {
## real ypred = normal_rng(alpha + beta * xpred, sigma); // changed mu to alpha + beta * xpred
## }
```

b)

Finding the σ_β was done through trial and error with the following function:

```
# Shows the 0.99 confidence interval. Results were found by manually changing the stdev_b
# until the values were close to -69 and 69
stdev_b <- 26.78
c(qnorm(0.995, 0, sd = stdev_b, lower.tail = FALSE),
  qnorm(0.995, 0, sd = stdev_b, lower.tail = TRUE))
```

```
## [1] -68.98071  68.98071
```

$$\sigma_\beta \approx 26.78$$

c)

Adding $\beta \sim N(0, \sigma_\beta)$ prior to the model: Added line 6 "psbeta". σ_β can be added as a parameter when calling the model. Added line 20 "beta ~ normal(0, psbeta)" to model the prior distribution of $Beta \sim N(0, \sigma_\beta)$

```
writeLines(readLines("drowning_beta_prior.stan"))
```

```
## 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
## real psbeta; // prior std for beta
## }
##
## parameters {
## real alpha;
## real beta; // slope
## real<lower=0> sigma;
## }
##
## transformed parameters {
## vector[N] mu = alpha + beta*x;
```

```
## }
##
## model {
## beta ~ normal(0, psbeta); // prior
## y ~ normal(mu, sigma);
## }
##
## generated quantities {
## real ypred = normal_rng(alpha + beta * xpred, sigma);
## }
```

d)

Add α prior to the model with adjustable priors. Added prior $\alpha \sim N(\mu_\alpha, \sigma_\alpha)$ to the model: Added line 7 "psalpha" for σ_α Added line 8 "pmualpha" for μ_α Added line 23 "alpha ~ normal(pmualpha, psalpha)" as the prior

```
writeLines(readLines("drowning_fullmodel.stan"))
```

```
## 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
## real psbeta; // prior std for beta
## real psalpha; // prior std for alpha
## real pmualpha; // prior mean for alpha
## }
##
## parameters {
## real alpha; // intercept
## real beta; // slope
## real<lower=0> sigma;
## }
##
## transformed parameters {
## vector[N] mu = alpha + beta*x;
## }
##
## model {
## beta ~ normal(0, psbeta); // prior beta
## alpha ~ normal(pmualpha, psalpha); // prior alpha
## y ~ normal(mu, sigma);
## }
##
## generated quantities {
## real ypred = normal_rng(alpha + beta * xpred, sigma);
## }
```

When selecting a prior for alpha, we want to keep it weakly informative to provide some information, but not to skew the results too much. Therefore we need to choose the prior values carefully.

For the mean of the prior normal distribution for α, a centered approach is taken, i.e. the mean number of drownings is chosen.

In Cmdstanr Kilpisjärvi temperatures demo, the temperature changes were less than 10 degrees, and the prior was $\sigma_\alpha = 100$. Therefore, the prior is kept an order of magnitude higher than the largest changes. From the earlier plot, it can be seen that the largest differences in drownings were less than 120. Therefore, in order to keep the prior weakly informative $\sigma_\alpha = 1200$

```
mualpha <- mean(drowning$drownings)
stdev_a <- 1200
mualpha # confirms the historical approximate in d)
```

```
## [1] 134.35
```

Modeling

```
drowning_data <- list(N=nrow(drowning), x = drowning$year, y = drowning$drownings,
                      xpred = 2020, psbeta = stdev_b, pmualpha=mualpha, psalpha=stdev_a)
```

```
mod_lin <- cmdstan_model(stan_file = "drowning_fullmodel.stan")
fit_lin <- mod_lin$sample(data = drowning_data, seed = seed, refresh=1000)
```

```
head(fit_lin$summary(),5)
```

```
## # A tibble: 5 x 10
##   variable     mean   median      sd     mad      q5      q95  rhat ess_bulk
##   <chr>       <dbl>    <dbl>   <dbl>   <dbl>   <dbl>    <dbl> <dbl>    <dbl>
## 1 lp__     -148.    -148.      1.24    0.987 -150.    -147.     1.00    1139.
## 2 alpha    1847.    1853.    633.    628.     774.    2875.     1.00     799.
## 3 beta       -0.857   -0.860   0.317   0.314   -1.37    -0.319  1.00     799.
## 4 sigma      26.4     26.2     3.03    2.98    21.9     31.8    1.00    1167.
## 5 mu[1]     151.     151.      7.38    7.24   139.     163.     1.00    1240.
## # ... with 1 more variable: ess_tail <dbl>
```

$\mu_\beta = -0.857$

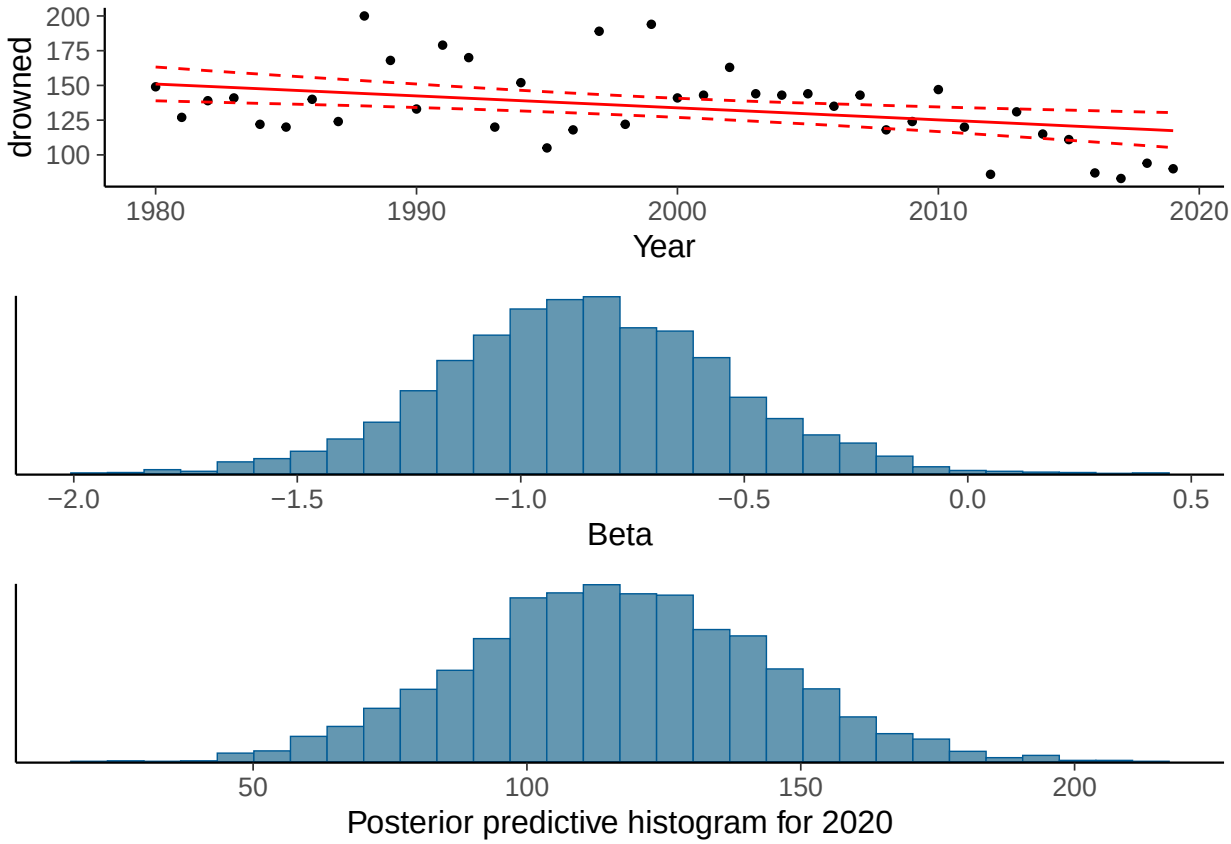
Visualizing i) and ii)

```
draws_lin <- as_draws_df(fit_lin$draws())
mu <- draws_lin %>%
  as_draws_df() %>%
  as_tibble() %>%
  select(starts_with("mu")) %>%
  apply(2, quantile, c(0.05, 0.5, 0.95)) %>%
  t() %>%
  data.frame(x = drowning_data$x, .)  %>%
  gather(pct, y, -x)

pfit <- ggplot() +
  geom_point(aes(x, y), data = data.frame(drowning_data), size = 1) +
  geom_line(aes(x, y, linetype = pct), data = mu, color = 'red') +
```

```
  scale_linetype_manual(values = c(2,1,2)) +
  labs(y = 'drowned', x= "Year") +
  guides(linetype = "none")
phist_b <- mcmc_hist(draws_lin, pars = c('beta')) + xlab("Beta")
phist_y <- mcmc_hist(draws_lin, pars = c('ypred')) +
  xlab("Posterior predictive histogram for 2020")
grid.arrange(pfit, phist_b, phist_y, nrow = 3)
```

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



The plots look very similar to the examples.

2. Hierarchical model: factory data with Stan

```
show(factory)
```

```
##   V1  V2  V3  V4  V5  V6
## 1 83 117 101 105  79  57
## 2 92 109  93 119  97  92
## 3 92 114  92 116 103 104
## 4 46 104  86 102  79  77
## 5 67  87  67 116  92 100
```

Example separate model:

$$
\begin{aligned}
y_{ji} &\sim N(\mu_i, \sigma_j) \\
\mu_i &\sim N(0, 1) \\
\sigma_j &\sim Inv - \chi^2(10)
\end{aligned}
$$

The example priors for the separate model are quite bad. The μ_i prior is extremely informative with a $\sigma_\mu = 1$, with the mean quite far from the results gathered. These factors likely dominate the result.

Distribution of σ_j prior has the same problems, as can be seen from the following charts.

Due to the poor priors in the example, new priors are chosen.

```
sapply(factory, mean)
```

```
##    V1    V2    V3    V4    V5    V6
##  76.0 106.2  87.8 111.6  90.0  86.0
```

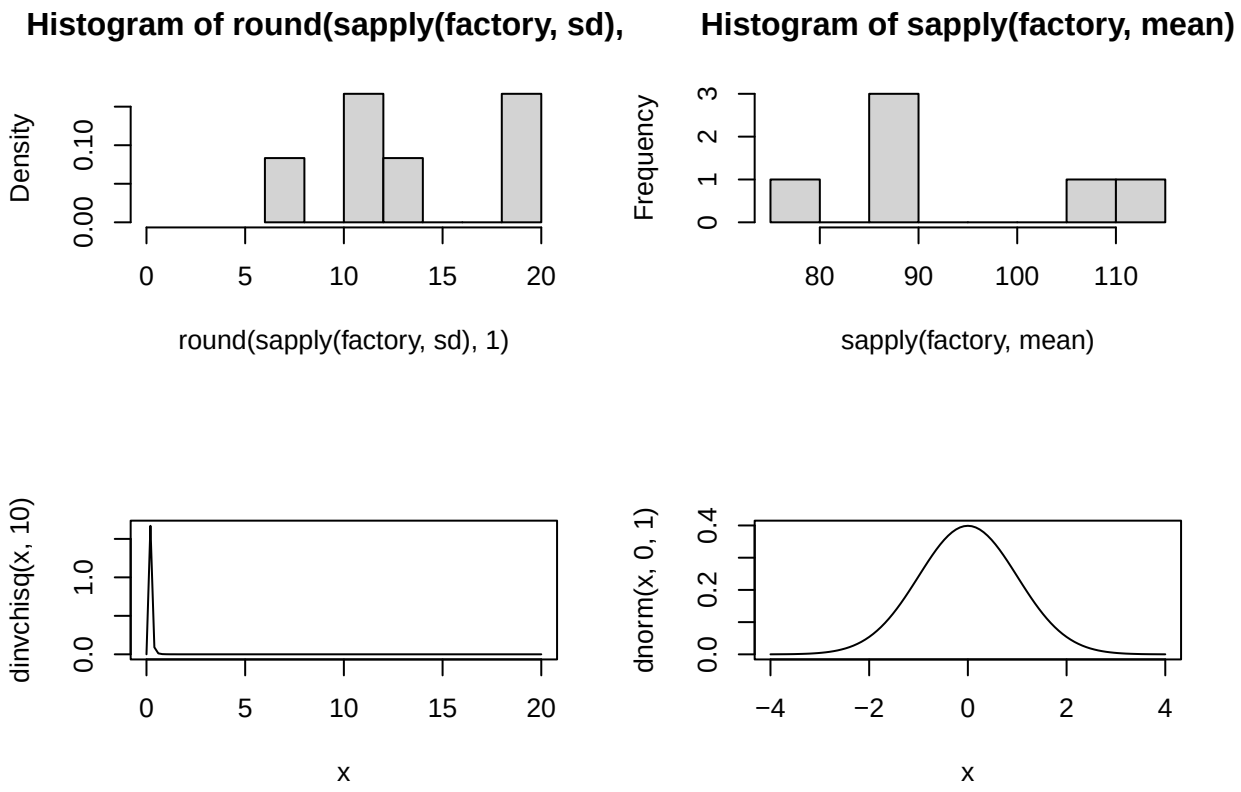
```
round(sapply(factory, sd),1)
```

```
##   V1   V2   V3   V4   V5   V6
## 19.6 11.8 12.8  7.6 10.8 19.2
```

Showcasing distributions of results and priors, priors are extremely informative and do not reflect data:

```
par(mfrow=c(2,2))
hist(round(sapply(factory, sd),1), freq = F, breaks = 6, xlim = c(0,20))
hist(sapply(factory, mean), breaks = 6)

curve(dinvchisq(x, 10), from = 0, to = 20)
curve(dnorm(x, 0, 1), from = -4, to = 4)
```



a)

About prior selection: As there was little previous data available, the priors were chosen rather arbitrarily, but with the goal of being weakly informative. The priors' mean μ distributions were selected to be roughly in the neighborhood of the original measurements, and the standard deviations around triple of the measured standard deviations. μ was restricted to be positive, as it is unlikely that the machines can create an object with a negative property.

Lognormal distribution was used for σ priors, as it restricts the values to be positive, while most of the probability mass is closer to zero.

While this type of prior selection may be bad practice in some cases, it is used to give reasonable priors in absence of knowledge of better methods.

Separate models

$$\begin{aligned} y_{ji} &\sim N(\mu_i, \sigma_j) \\ \mu_i &\sim N(80, 60) \\ \sigma_j &\sim LogNormal(1, 10) \end{aligned}$$

The separate model treats all machines as independent from each other. In other words, the data generating process (DGP) is different for all machines, even if they share the same priors.

Pooled model

$$
\begin{aligned}
y_{ij} &\sim N(\mu, \sigma) \\
\mu &\sim N(80, 60) \\
\sigma &\sim LogNormal(1, 10)
\end{aligned}
$$

Pooled model considers that all data comes from the same DGP, regardless of the machine.

Hierarchical model:

$$
\begin{aligned}
y_{ji} &\sim N(\mu_j, \sigma) \\
\mu_j &\sim N(\mu_0, \sigma_0) \\
\mu_0 &\sim N(80, 60) \\
\sigma_0 &\sim N(0, 10) \\
\sigma &\sim LogNormal(1, 10)
\end{aligned}
$$

Hierarchical model considers that the DGPs for each machine are different, but come from a common distribution. The distribution of parameter μ_j follows the distribution dictated by the hyperparameters μ_0 and σ_0. The variance σ is assumed to be same for all machines.

b)

The pieces of Stan code are based on the CmdStanR demos of the course.

Separate model code:

```
writeLines(readLines("factory_sep.stan"))
```

```
## data {
##   int<lower=0> N; // Number of data points
##   int<lower=0> J; // Number of groups
##   array[N] int<lower=1, upper=J> x; // discrete group indicators
##   vector[N] y; // real valued observations
## }
## parameters {
##   vector[J] mu; // group means
##   vector<lower=0>[J] sigma; // stds
## }
## model {
##   mu ~ normal(80,100);
##   sigma ~ lognormal(1, 10);
##   y ~ normal(mu[x], sigma[J]);
## }
## generated quantities {
##   vector[J] ypred;
##   for (i in 1:J) {
##     ypred[i] = normal_rng(mu[i], sigma[i]); // Predictive distribution for n:th machine
##   }
## }
```

Pooled model code:

```
writeLines(readLines("factory_pool.stan"))
```

```
##
## data {
##   int<lower=0> N; // Number of data points
##   int<lower=0> J; // Number of groups
##   vector[N] y; // real valued observations
## }
## parameters {
##   real mu; // mean
##   real<lower=0> sigma; // std
## }
## model {
##   mu ~ normal(80,100);
##   sigma ~ lognormal(1, 10);
##   y ~ normal(mu, sigma);
## }
## generated quantities{
##   real ypred;
##   ypred = normal_rng(mu, sigma); // Predictive distribution for all machines
## }
```

Hierarchical model code:

```
writeLines(readLines("factory_hier.stan"))
```

```
## data {
##   int<lower=0> N; // Number of data points
##   int<lower=0> J; // Number of groups
##   array[N] int<lower=1, upper=J> x; // discrete group indicators
##   vector[N] y; // real valued observations
## }
## parameters {
##   real mu0; // prior mean
##   real<lower=0> sigma0; // prior std constrained to be positive
##   vector[J] mu; // group means
##   real<lower=0> sigma; // common std constrained to be positive
## }
## model {
##   mu0 ~ normal(80,60);
##   sigma0 ~ normal(0,10);
##   mu ~ normal(mu0,sigma0);
##   sigma ~ lognormal(1, 10);
##   y ~ normal(mu[x], sigma);
## }
## generated quantities {
##   vector[J] ypred;
##   for (i in 1:J) {
##     ypred[i] = normal_rng(mu[i], sigma); // Predictive distribution for n:th machine
##   }
##   real mupred = normal_rng(mu0,sigma0);
## }
```

```r
# Loading data
stan_data <- list(
  N = 6 * nrow(factory),
  J = 6,
  x = rep(1:6, nrow(factory)),
  y = c(t(factory))
)
```

```r
#, results = 'hide', warning=FALSE, message=FALSE}
factory_sep <- cmdstan_model(stan_file = "factory_sep.stan")
fit_sep <- factory_sep$sample(data = stan_data, refresh=1000)
```

```
## Running MCMC with 4 chains, at most 16 in parallel...
##
## Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4 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 0.2 seconds.
## 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: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3 finished in 0.3 seconds.
## 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 0.2 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 0.3 seconds.
## Total execution time: 0.4 seconds.
```

```r
draws_sep <- as_draws_df(fit_sep$draws())
fit_sep$summary()
```

```
## # A tibble: 19 x 10
##    varia~1     mean  median      sd   mad       q5     q95  rhat ess_b~2 ess_t~3
##    <chr>      <dbl>   <dbl>   <dbl> <dbl>    <dbl>   <dbl> <dbl>   <dbl>   <dbl>
##  1 lp__    -1.04e 2 -103.   2.60e 0  2.47 -1.09e+2 -1.00e2  1.00   1592.   2296.
##  2 mu[1]    7.61e 1   76.0  6.68e 0  6.28  6.52e+1  8.72e1  1.00   8800.   3042.
##  3 mu[2]    1.06e 2  106.   6.59e 0  6.64  9.56e+1  1.17e2  1.00   7628.   3023.
##  4 mu[3]    8.77e 1   87.8  6.59e 0  6.27  7.70e+1  9.83e1  1.00   7094.   2970.
##  5 mu[4]    1.12e 2  112.   6.67e 0  6.60  1.01e+2  1.23e2  1.00   7614.   3044.
##  6 mu[5]    8.99e 1   89.8  6.72e 0  6.85  7.89e+1  1.01e2  1.00   8282.   2613.
##  7 mu[6]    8.60e 1   86.0  6.68e 0  6.76  7.52e+1  9.67e1  1.00   7520.   2341.
```

```
##  8 sigma[~  5.08e14    3.06 3.21e16  4.53  2.53e-7  5.27e7  1.00   8926.   3117.
##  9 sigma[~  1.88e12    2.61 8.62e13  3.87  1.71e-7  2.82e7  1.00  10402.   2883.
## 10 sigma[~  1.61e11    1.98 4.71e12  2.93  1.21e-7  4.55e7  1.00   8383.   2720.
## 11 sigma[~  1.95e10    2.50 5.37e11  3.70  1.45e-7  4.78e7  1.00   9850.   3189.
## 12 sigma[~  1.26e13    3.13 5.00e14  4.64  1.99e-7  5.42e7  1.00   8691.   2562.
## 13 sigma[~  1.48e 1   14.5  2.23e 0  2.10  1.16e+1  1.89e1  1.00   4445.   3362.
## 14 ypred[~  7.58e14   76.0  4.80e16 15.1  -8.97e+5  4.91e5  1.00   3609.   3465.
## 15 ypred[~ -3.26e12  106.   2.23e14 15.2  -6.77e+5  4.96e5  1.00   3980.   3632.
## 16 ypred[~  7.62e10   87.7  6.41e12 14.1  -4.37e+5  8.48e5  1.00   3791.   2925.
## 17 ypred[~ -6.27e 9  112.   3.65e11 14.7  -4.65e+5  1.49e6  1.00   4114.   3127.
## 18 ypred[~  3.18e12   89.7  3.16e14 14.6  -7.55e+5  4.22e5  1.00   3978.   3375.
## 19 ypred[~  8.58e 1   85.6  1.64e 1 15.7   5.91e+1  1.13e2  1.00   4418.   3887.
## # ... with abbreviated variable names 1: variable, 2: ess_bulk, 3: ess_tail
```

```
fit_sep$cmdstan_diagnose()
```

```
## Processing csv files: C:/Users/oksan/AppData/Local/Temp/Rtmps7jVM5/factory_sep-202301111036-1-339243
##
## Checking sampler transitions treedepth.
## Treedepth satisfactory for all transitions.
##
## Checking sampler transitions for divergences.
## No divergent transitions found.
##
## Checking E-BFMI - sampler transitions HMC potential energy.
## E-BFMI satisfactory.
##
## Effective sample size satisfactory.
##
## Split R-hat values satisfactory all parameters.
##
## Processing complete, no problems detected.
```

```
#, results = 'hide', warning=FALSE, message=FALSE}
factory_pool <- cmdstan_model(stan_file = "factory_pool.stan")
fit_pool <- factory_pool$sample(data = stan_data, refresh=1000)
```

```
## Running MCMC with 4 chains, at most 16 in parallel...
##
## 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 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 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 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 1 finished in 0.1 seconds.
## Chain 2 finished in 0.1 seconds.
## Chain 3 finished in 0.1 seconds.
## Chain 4 finished in 0.1 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 0.1 seconds.
## Total execution time: 0.4 seconds.
```

```
fit_sep$summary()
```

```
## # A tibble: 19 x 10
##    varia~1     mean  median      sd   mad       q5     q95  rhat ess_b~2 ess_t~3
##    <chr>      <dbl>   <dbl>   <dbl> <dbl>    <dbl>   <dbl> <dbl>   <dbl>   <dbl>
##  1 lp__    -1.04e 2 -103.   2.60e 0  2.47 -1.09e+2 -1.00e2  1.00   1592.   2296.
##  2 mu[1]    7.61e 1   76.0  6.68e 0  6.28  6.52e+1  8.72e1  1.00   8800.   3042.
##  3 mu[2]    1.06e 2  106.   6.59e 0  6.64  9.56e+1  1.17e2  1.00   7628.   3023.
##  4 mu[3]    8.77e 1   87.8  6.59e 0  6.27  7.70e+1  9.83e1  1.00   7094.   2970.
##  5 mu[4]    1.12e 2  112.   6.67e 0  6.60  1.01e+2  1.23e2  1.00   7614.   3044.
##  6 mu[5]    8.99e 1   89.8  6.72e 0  6.85  7.89e+1  1.01e2  1.00   8282.   2613.
##  7 mu[6]    8.60e 1   86.0  6.68e 0  6.76  7.52e+1  9.67e1  1.00   7520.   2341.
##  8 sigma[~  5.08e14    3.06 3.21e16  4.53  2.53e-7  5.27e7  1.00   8926.   3117.
##  9 sigma[~  1.88e12    2.61 8.62e13  3.87  1.71e-7  2.82e7  1.00  10402.   2883.
## 10 sigma[~  1.61e11    1.98 4.71e12  2.93  1.21e-7  4.55e7  1.00   8383.   2720.
## 11 sigma[~  1.95e10    2.50 5.37e11  3.70  1.45e-7  4.78e7  1.00   9850.   3189.
## 12 sigma[~  1.26e13    3.13 5.00e14  4.64  1.99e-7  5.42e7  1.00   8691.   2562.
## 13 sigma[~  1.48e 1   14.5  2.23e 0  2.10  1.16e+1  1.89e1  1.00   4445.   3362.
## 14 ypred[~  7.58e14   76.0  4.80e16 15.1  -8.97e+5  4.91e5  1.00   3609.   3465.
## 15 ypred[~ -3.26e12  106.   2.23e14 15.2  -6.77e+5  4.96e5  1.00   3980.   3632.
## 16 ypred[~  7.62e10   87.7  6.41e12 14.1  -4.37e+5  8.48e5  1.00   3791.   2925.
## 17 ypred[~ -6.27e 9  112.   3.65e11 14.7  -4.65e+5  1.49e6  1.00   4114.   3127.
## 18 ypred[~  3.18e12   89.7  3.16e14 14.6  -7.55e+5  4.22e5  1.00   3978.   3375.
## 19 ypred[~  8.58e 1   85.6  1.64e 1 15.7   5.91e+1  1.13e2  1.00   4418.   3887.
## # ... with abbreviated variable names 1: variable, 2: ess_bulk, 3: ess_tail
```

```
fit_sep$cmdstan_diagnose()
```

```
## Processing csv files: C:/Users/oksan/AppData/Local/Temp/Rtmps7jVM5/factory_sep-202301111036-1-339243
##
## Checking sampler transitions treedepth.
## Treedepth satisfactory for all transitions.
##
## Checking sampler transitions for divergences.
## No divergent transitions found.
##
## Checking E-BFMI - sampler transitions HMC potential energy.
## E-BFMI satisfactory.
##
## Effective sample size satisfactory.
##
## Split R-hat values satisfactory all parameters.
```

```
##
## Processing complete, no problems detected.
```

```
draws_pool <- as_draws_df(fit_pool$draws())
```

```
#, results = 'hide', warning=FALSE, message=FALSE}
factory_hier <- cmdstan_model(stan_file = "factory_hier.stan")
fit_hier <- factory_hier$sample(data = stan_data, refresh=1000)
```

```
## Running MCMC with 4 chains, at most 16 in parallel...
##
## Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4 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 0.2 seconds.
## 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 0.2 seconds.
## 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 0.2 seconds.
## 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 0.2 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 0.2 seconds.
## Total execution time: 0.5 seconds.

## Warning: 153 of 4000 (4.0%) transitions ended with a divergence.
## See https://mc-stan.org/misc/warnings for details.
```

```
fit_sep$summary()
```

```
## # A tibble: 19 x 10
##    varia~1     mean  median      sd   mad       q5     q95  rhat ess_b~2 ess_t~3
##    <chr>      <dbl>   <dbl>   <dbl> <dbl>    <dbl>   <dbl> <dbl>   <dbl>   <dbl>
##  1 lp__    -1.04e 2 -103.   2.60e 0  2.47 -1.09e+2 -1.00e2  1.00   1592.   2296.
##  2 mu[1]    7.61e 1   76.0  6.68e 0  6.28  6.52e+1  8.72e1  1.00   8800.   3042.
##  3 mu[2]    1.06e 2  106.   6.59e 0  6.64  9.56e+1  1.17e2  1.00   7628.   3023.
##  4 mu[3]    8.77e 1   87.8  6.59e 0  6.27  7.70e+1  9.83e1  1.00   7094.   2970.
##  5 mu[4]    1.12e 2  112.   6.67e 0  6.60  1.01e+2  1.23e2  1.00   7614.   3044.
##  6 mu[5]    8.99e 1   89.8  6.72e 0  6.85  7.89e+1  1.01e2  1.00   8282.   2613.
##  7 mu[6]    8.60e 1   86.0  6.68e 0  6.76  7.52e+1  9.67e1  1.00   7520.   2341.
##  8 sigma[~  5.08e14    3.06 3.21e16  4.53  2.53e-7  5.27e7  1.00   8926.   3117.
```

```
##  9 sigma[~  1.88e12    2.61 8.62e13  3.87  1.71e-7  2.82e7  1.00  10402.   2883.
## 10 sigma[~  1.61e11    1.98 4.71e12  2.93  1.21e-7  4.55e7  1.00   8383.   2720.
## 11 sigma[~  1.95e10    2.50 5.37e11  3.70  1.45e-7  4.78e7  1.00   9850.   3189.
## 12 sigma[~  1.26e13    3.13 5.00e14  4.64  1.99e-7  5.42e7  1.00   8691.   2562.
## 13 sigma[~  1.48e 1   14.5  2.23e 0  2.10  1.16e+1  1.89e1  1.00   4445.   3362.
## 14 ypred[~  7.58e14   76.0  4.80e16 15.1  -8.97e+5  4.91e5  1.00   3609.   3465.
## 15 ypred[~ -3.26e12  106.   2.23e14 15.2  -6.77e+5  4.96e5  1.00   3980.   3632.
## 16 ypred[~  7.62e10   87.7  6.41e12 14.1  -4.37e+5  8.48e5  1.00   3791.   2925.
## 17 ypred[~ -6.27e 9  112.   3.65e11 14.7  -4.65e+5  1.49e6  1.00   4114.   3127.
## 18 ypred[~  3.18e12   89.7  3.16e14 14.6  -7.55e+5  4.22e5  1.00   3978.   3375.
## 19 ypred[~  8.58e 1   85.6  1.64e 1 15.7   5.91e+1  1.13e2  1.00   4418.   3887.
## # ... with abbreviated variable names 1: variable, 2: ess_bulk, 3: ess_tail
```

```
fit_sep$cmdstan_diagnose()
```

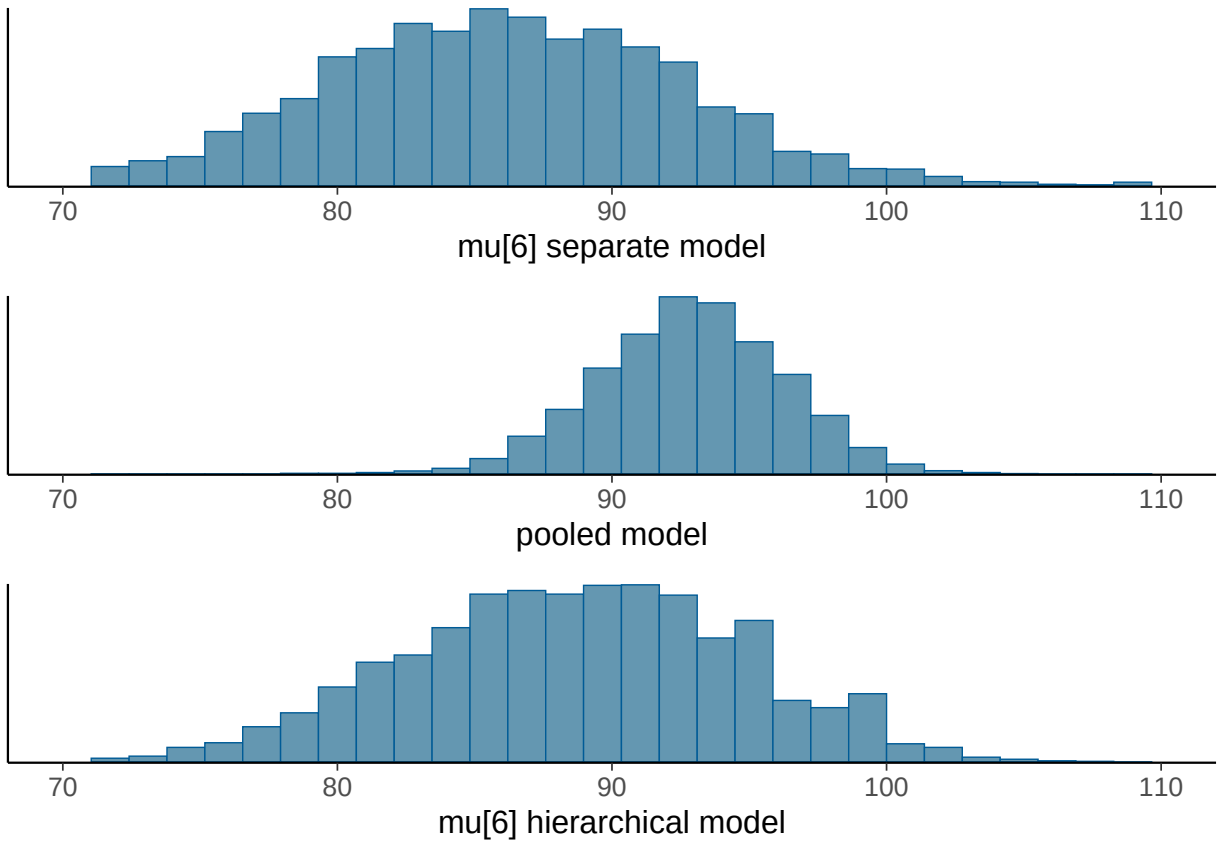
```
## Processing csv files: C:/Users/oksan/AppData/Local/Temp/Rtmps7jVM5/factory_sep-202301111036-1-339243
##
## Checking sampler transitions treedepth.
## Treedepth satisfactory for all transitions.
##
## Checking sampler transitions for divergences.
## No divergent transitions found.
##
## Checking E-BFMI - sampler transitions HMC potential energy.
## E-BFMI satisfactory.
##
## Effective sample size satisfactory.
##
## Split R-hat values satisfactory all parameters.
##
## Processing complete, no problems detected.
```

```
draws_hier <- as_draws_df(fit_hier$draws())
```

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

```
sep_mu6 <- mcmc_hist(draws_sep,
                     pars = c('mu[6]'))+xlim(c(70,110))+labs(x="mu[6] separate model")
pool_mu6 <- mcmc_hist(draws_pool,
                      pars = c('mu'))+xlim(c(70,110))+labs(x="pooled model")
hier_mu6 <- mcmc_hist(draws_hier,
                      pars = c('mu[6]'))+xlim(c(70,110))+labs(x="mu[6] hierarchical model")
grid.arrange(sep_mu6, pool_mu6, hier_mu6, nrow = 3)
```

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

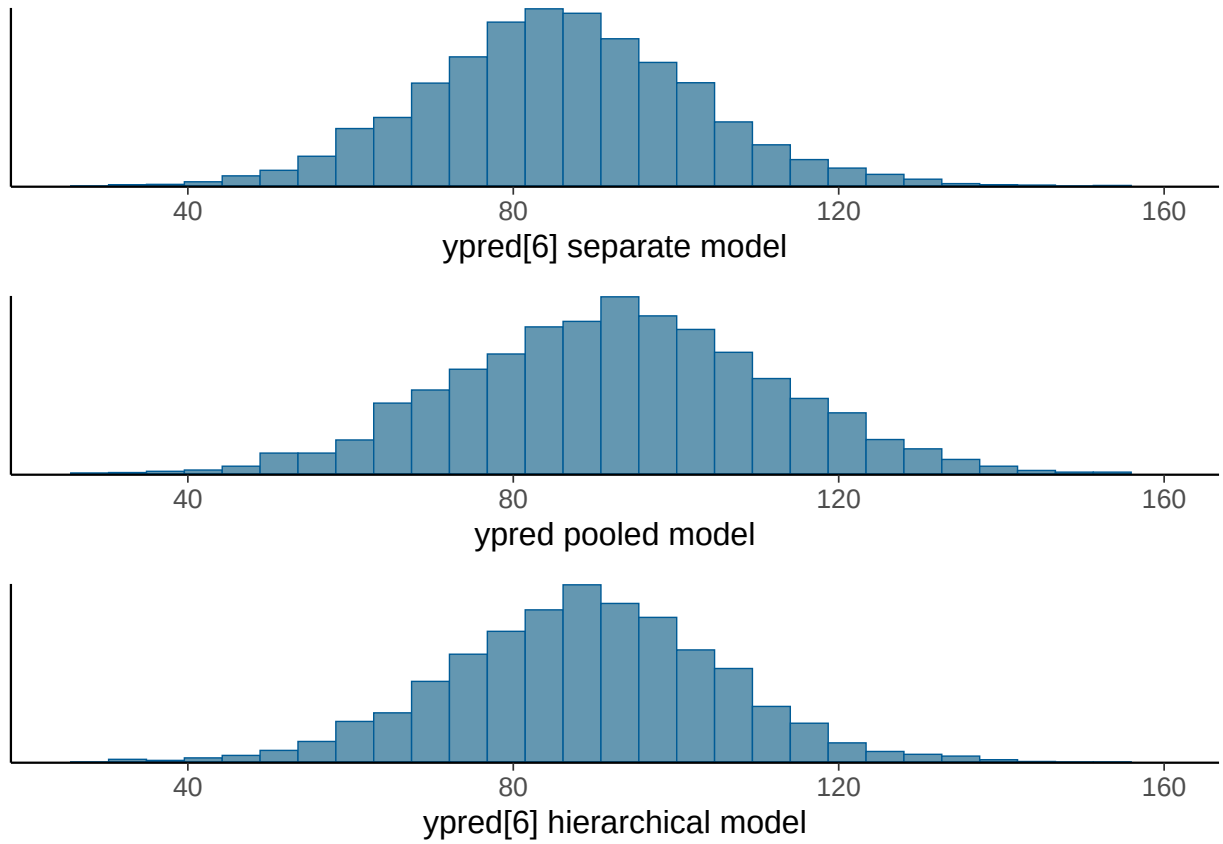


All of the models produced rather different distributions, but within reason. The pooled model is unsurprisingly the narrowest, as it has the most datapoints (all observations for every machine), reducing the effect of the prior. As the other models have less datapoints, their models have more uncertainty, which can be seen as in the larger width of mu[6] predictions. The separate model is the widest, as it has the least datapoints (only the observations for the 6th machine).

```
sep_sigma6 <- mcmc_hist(draws_sep, pars = c('ypred[6]')
                        ) + xlim(c(25,160)) + labs(x="ypred[6] separate model")
pool_sigma6 <- mcmc_hist(draws_pool, pars = c('ypred')
                         ) + xlim(c(25,160)) + labs(x="ypred pooled model")
hier_sigma6 <- mcmc_hist(draws_hier, pars = c('ypred[6]')
                         ) + xlim(c(25,160)) + labs(x="ypred[6] hierarchical model")

grid.arrange(sep_sigma6, pool_sigma6, hier_sigma6, nrow = 3)
```

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



The y_pred results seem very rather similar, with slight differences in the mean.

iii) the posterior distribution of the mean of the quality measurements of the seventh

machine.

For the separate model, predicting the results of a seventh machine does not seem beneficial. As the DGP of a single machine is assumed to be independent of other machines, and there is no information of the seventh machine and therefore extremely little can be said about the distributions of the seventh machine. One option is to model the μ distribution directly with a previously mentioned prior distribution, but the distribution is so wide that it is of little use and purely based on assumptions.

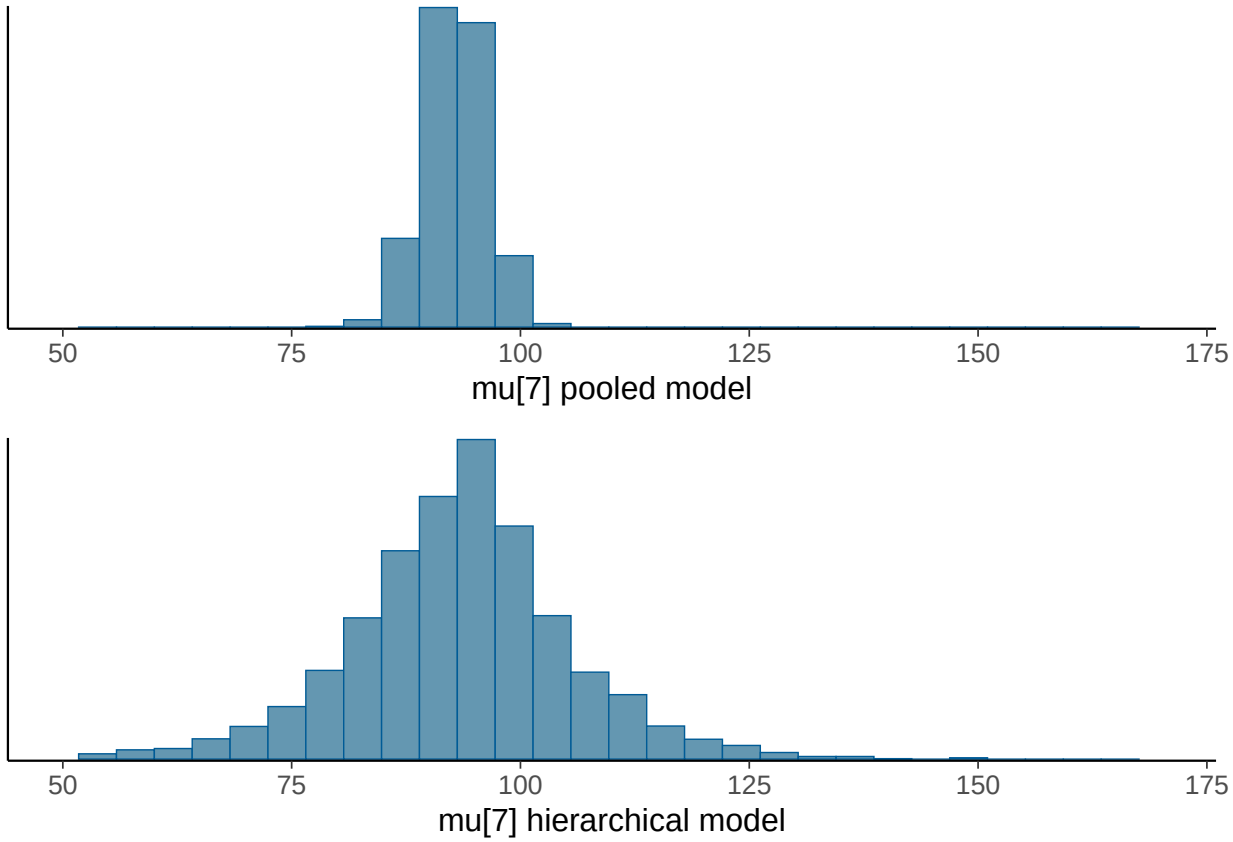
For the pooled model, as it assumes that all measurements come from the same distribution, the mean distribution is the same as it is for all machines.

In the hierarchical model, it comes from the prior $\mu_0 \sim N(80, 60)$.

```
pool_mu7 <- mcmc_hist(draws_pool, pars = c('mu')
                      ) + xlim(c(50,170)) + labs(x="mu[7] pooled model")
hier_mu7 <- mcmc_hist(draws_hier, pars = c('mupred')
                      ) + xlim(c(50,170)) + labs(x="mu[7] hierarchical model")

grid.arrange(pool_mu7, hier_mu7, nrow = 2)
```

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



The mu7 distribution looks quite different in the hierarchical model compared to the pooled model, with much wider tails. This is expected, as they have similar priors, but the hierarchical model adds uncertainty by as it takes into consideration the effect of an individual machine.

d)

Separate model:

$$
\begin{aligned}
y_{ji} &\sim N(\mu_i, \sigma_j) \\
\mu_i &\sim N(0, 10) \\
\sigma_j &\sim Gamma(1, 1)
\end{aligned}
$$

```
writeLines(readLines("factory_sep_d.stan"))
```

```
## data {
##   int<lower=0> N; // Number of data points
##   int<lower=0> J; // Number of groups
##   array[N] int<lower=1, upper=J> x; // discrete group indicators
##   vector[N] y; // real valued observations
## }
## parameters {
##   vector[J] mu; // group means
##   vector<lower=0>[J] sigma; // stds
```

```
## }
## model {
##   mu ~ normal(0,10);
##   sigma ~ gamma(1,1);
##   y ~ normal(mu[x], sigma[J]);
## }
```

Pooled model:

$$
\begin{aligned}
y_{ji} &\sim N(\mu, \sigma) \\
\mu &\sim N(0, 10) \\
\sigma &\sim Gamma(1, 1)
\end{aligned}
$$

```
writeLines(readLines("factory_pool_d.stan"))
```

```
##
## data {
##   int<lower=0> N; // Number of data points
##   int<lower=0> J; // Number of groups
##   vector[N] y; // real valued observations
## }
## parameters {
##   real mu; // mean
##   real<lower=0> sigma; // std
## }
## model {
##   mu ~ normal(0,10);
##   sigma ~ gamma(1,1);
##   y ~ normal(mu, sigma);
## }
```

Hierarchical model:

$$
\begin{aligned}
y_{ji} &\sim N(\mu_j, \sigma) \\
\mu_j &\sim N(\mu_0, \sigma_0) \\
\mu_0 &\sim N(0, 10) \\
\sigma_0 &\sim Gamma(1, 1) \\
\sigma &\sim Gamma(1, 1)
\end{aligned}
$$

```
writeLines(readLines("factory_hier_d.stan"))
```

```
##
## data {
##   int<lower=0> N; // Number of data points
##   int<lower=0> J; // Number of groups
##   array[N] int<lower=1, upper=J> x; // discrete group indicators
##   vector[N] y; // real valued observations
## }
```

```
## parameters {
##   real mu0; // prior mean
##   real<lower=0> sigma0; // prior std constrained to be positive
##   vector[J] mu; // group means
##   real<lower=0> sigma; // common std constrained to be positive
## }
## model {
##   mu0 ~ normal(0,10);
##   sigma0 ~ gamma(1,1);
##   mu ~ normal(mu0,sigma0);
##   sigma ~ gamma(1,1);
##   y ~ normal(mu[x], sigma);
## }
## generated quantities{
##   real mupred = normal_rng(mu0,sigma0);
## }
```

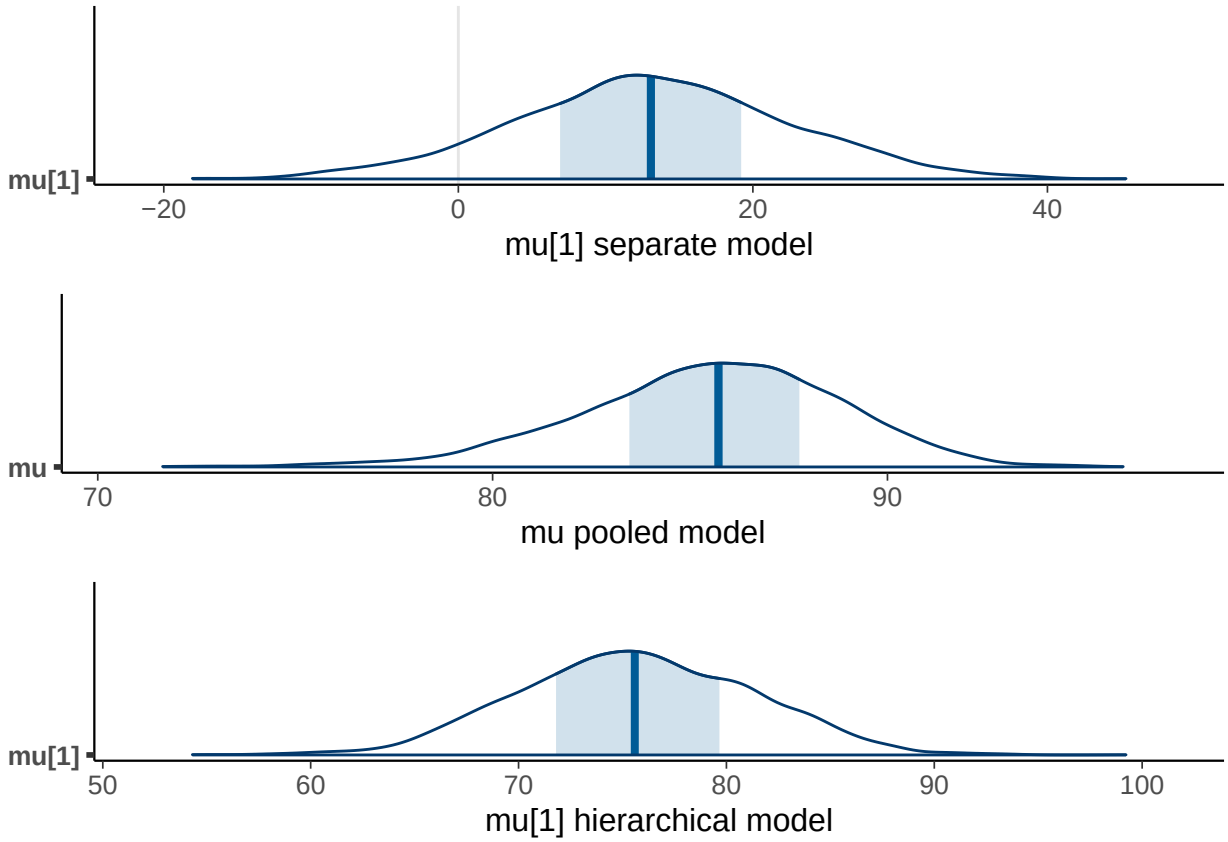
```
factory_sep <- cmdstan_model(stan_file = "factory_sep_d.stan")
fit_sep <- factory_sep$sample(data = stan_data, refresh=1000)
draws_sep <- as_draws_df(fit_sep$draws())
```

```
factory_pool <- cmdstan_model(stan_file = "factory_pool_d.stan")
fit_pool <- factory_pool$sample(data = stan_data, refresh=1000)
draws_pool <- as_draws_df(fit_pool$draws())
```

```
factory_hier <- cmdstan_model(stan_file = "factory_hier_d.stan")
fit_hier <- factory_hier$sample(data = stan_data, refresh=1000)
draws_hier <- as_draws_df(fit_hier$draws())
```

Posterior expectation for μ_1 with a 90% credible interval can be seen from the following chart:

```
sep_mu1 <- mcmc_areas(draws_sep, pars = c('mu[1]')) + labs(x="mu[1] separate model")
pool_mu1 <- mcmc_areas(draws_pool, pars = c('mu')) + labs(x="mu pooled model")
hier_mu1 <- mcmc_areas(draws_hier, pars = c('mu[1]')) + labs(x="mu[1] hierarchical model")
grid.arrange(sep_mu1, pool_mu1, hier_mu1, nrow = 3)
```


mu[1]
−20
0
20
40
mu[1] separate model
mu
70
80
90
mu pooled model
mu[1]
50
60
70
80
90
100
mu[1] hierarchical model