BDA - Assignment 8

21/11/2021

Contents

Seprate model **1**
1. 1
2 4
3. 5
4. 5

Pooled model **5**
1. 5
2. 6
4. 7

Hierarchical **8**
1. 8
2. 9
3. 10

5. **10**

Seprate model

1.

We copy the model from assignment 7 and modify it to add the log likelihood in the "generated quantaties" block.

```
data {
 int<lower=0> N;
 int<lower=0> J;
 vector[J] y[N];
 real mean_mu_prior;
 real<lower=0> mean_sigma_prior;
 real<lower=0> sigma_prior;
}

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

model {
 //priors
 for (j in 1:J){
  mu[j] ~ normal(mean_mu_prior, mean_sigma_prior);
```

```
  sigma[j] ~ inv_chi_square(sigma_prior);
 }
 //likelihoods
 for (j in 1:J)
  y[,j] ~ normal(mu[j], sigma[j]);
 }

generated quantities {
 //for the first machine
 real ypred;
 vector[J] log_lik[N];
 ypred = normal_rng(mu[6], sigma[6]);
 for (j in 1:J){
  for (n in 1:N){
      log_lik[n,j] = normal_lpdf(y[n,j] | mu[j], sigma[j]);
   }}
}
```

```
mean_mu_prior = 100
mean_sigma_prior = 10
sigma_prior = 10
separate_data <- list(
  y = factory,
  N = nrow(factory),
  J = ncol(factory),
 mean_mu_prior = mean_mu_prior,
 mean_sigma_prior = mean_sigma_prior,
 sigma_prior = sigma_prior
)
```

```
fit_separate = sampling(separatemodel,
  data = separate_data,            # named list of data
  chains = 4,             # number of Markov chains
  warmup = 1000,          # number of warmup iterations per chain
  iter = 2000,            # total number of iterations per chain
  cores = 1,              # number of cores (could use one per chain)
  refresh = 0             # no progress shown
  )
```

```
print(fit_separate)
```

```
## Inference for Stan model: c220590bd7aa17b9434e50848db1ac01.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##                mean se_mean    sd    2.5%     25%     50%     75%   97.5%
## mu[1]         83.45    0.10  6.08   72.69   79.24   82.97   87.19   96.45
## mu[2]        105.45    0.05  3.61   98.07  103.16  105.56  107.79  112.35
## mu[3]         89.63    0.05  3.99   82.27   87.07   89.37   91.91   98.23
## mu[4]        110.91    0.04  2.53  105.77  109.33  110.95  112.55  115.96
## mu[5]         91.03    0.05  3.32   84.59   88.80   90.96   93.11   97.71
## mu[6]         89.93    0.08  5.47   79.52   86.35   89.78   93.34  101.47
## sigma[1]      15.28    0.07  4.45    9.27   12.26   14.41   17.31   26.17
## sigma[2]       8.58    0.04  2.34    5.45    6.97    8.19    9.67   14.10
## sigma[3]       9.43    0.04  2.60    5.93    7.60    8.93   10.61   15.68
```

```
## sigma[4]        5.61    0.03  1.57    3.52    4.53    5.30    6.31    9.50
## sigma[5]        7.87    0.03  2.10    4.97    6.42    7.49    8.93   12.90
## sigma[6]       14.14    0.06  3.81    8.87   11.48   13.41   15.99   23.67
## ypred          89.76    0.25 15.44   59.29   79.99   89.53   99.28  121.23
## log_lik[1,1]   -3.69    0.01  0.27   -4.29   -3.86   -3.66   -3.50   -3.21
## log_lik[1,2]   -4.18    0.01  0.66   -5.82   -4.53   -4.07   -3.71   -3.23
## log_lik[1,3]   -4.15    0.01  0.67   -5.87   -4.45   -3.99   -3.69   -3.28
## log_lik[1,4]   -3.39    0.01  0.54   -4.76   -3.68   -3.29   -3.00   -2.64
## log_lik[1,5]   -4.39    0.01  0.78   -6.32   -4.81   -4.24   -3.83   -3.29
## log_lik[1,6]   -6.76    0.02  1.47  -10.48   -7.53   -6.51   -5.68   -4.75
## log_lik[2,1]   -3.93    0.00  0.31   -4.66   -4.09   -3.89   -3.72   -3.45
## log_lik[2,2]   -3.22    0.01  0.31   -3.90   -3.40   -3.19   -3.00   -2.71
## log_lik[2,3]   -3.31    0.01  0.28   -3.95   -3.47   -3.28   -3.12   -2.83
## log_lik[2,4]   -3.94    0.01  0.78   -5.80   -4.34   -3.79   -3.36   -2.87
## log_lik[2,5]   -3.40    0.01  0.39   -4.34   -3.60   -3.34   -3.13   -2.82
## log_lik[2,6]   -3.63    0.00  0.26   -4.17   -3.80   -3.61   -3.45   -3.19
## log_lik[3,1]   -3.93    0.00  0.31   -4.66   -4.09   -3.89   -3.72   -3.45
## log_lik[3,2]   -3.70    0.01  0.47   -4.82   -3.95   -3.61   -3.37   -2.98
## log_lik[3,3]   -3.27    0.01  0.27   -3.88   -3.43   -3.24   -3.07   -2.80
## log_lik[3,4]   -3.19    0.01  0.47   -4.30   -3.43   -3.11   -2.86   -2.52
## log_lik[3,5]   -4.45    0.01  0.88   -6.64   -4.88   -4.27   -3.81   -3.31
## log_lik[3,6]   -4.25    0.01  0.50   -5.53   -4.45   -4.14   -3.91   -3.58
## log_lik[4,1]   -7.16    0.02  1.51  -10.99   -7.94   -6.90   -6.06   -5.05
## log_lik[4,2]   -3.14    0.01  0.27   -3.73   -3.31   -3.12   -2.95   -2.68
## log_lik[4,3]   -3.29    0.01  0.31   -4.02   -3.47   -3.25   -3.06   -2.78
## log_lik[4,4]   -4.26    0.01  0.95   -6.56   -4.78   -4.05   -3.54   -2.97
## log_lik[4,5]   -4.39    0.01  0.78   -6.32   -4.81   -4.24   -3.83   -3.29
## log_lik[4,6]   -4.07    0.01  0.40   -5.02   -4.30   -4.03   -3.79   -3.43
## log_lik[5,1]   -4.31    0.01  0.45   -5.34   -4.57   -4.26   -3.99   -3.58
## log_lik[5,2]   -5.92    0.02  1.53   -9.77   -6.69   -5.62   -4.77   -3.88
## log_lik[5,3]   -6.57    0.02  1.59  -10.40   -7.44   -6.29   -5.42   -4.33
## log_lik[5,4]   -3.19    0.01  0.47   -4.30   -3.43   -3.11   -2.86   -2.52
## log_lik[5,5]   -3.05    0.01  0.27   -3.65   -3.22   -3.02   -2.86   -2.59
## log_lik[5,6]   -3.95    0.01  0.36   -4.79   -4.12   -3.90   -3.71   -3.41
## lp__         -167.75    0.06  2.66 -173.73 -169.32 -167.31 -165.83 -163.63
##              n_eff Rhat
## mu[1]         4054    1
## mu[2]         5044    1
## mu[3]         5487    1
## mu[4]         4248    1
## mu[5]         4530    1
## mu[6]         5012    1
## sigma[1]      3573    1
## sigma[2]      3439    1
## sigma[3]      3969    1
## sigma[4]      3301    1
## sigma[5]      3749    1
## sigma[6]      4039    1
## ypred         3912    1
## log_lik[1,1]  2846    1
## log_lik[1,2]  7538    1
## log_lik[1,3]  6503    1
## log_lik[1,4]  6745    1
## log_lik[1,5]  6781    1
```

```
## log_lik[1,6]  7190    1
## log_lik[2,1]  4606    1
## log_lik[2,2]  3076    1
## log_lik[2,3]  3056    1
## log_lik[2,4]  6262    1
## log_lik[2,5]  5016    1
## log_lik[2,6]  3429    1
## log_lik[3,1]  4606    1
## log_lik[3,2]  6185    1
## log_lik[3,3]  2815    1
## log_lik[3,4]  4251    1
## log_lik[3,5]  5897    1
## log_lik[3,6]  5327    1
## log_lik[4,1]  8015    1
## log_lik[4,2]  2377    1
## log_lik[4,3]  3215    1
## log_lik[4,4]  6763    1
## log_lik[4,5]  6781    1
## log_lik[4,6]  4137    1
## log_lik[5,1]  4416    1
## log_lik[5,2]  6409    1
## log_lik[5,3]  6114    1
## log_lik[5,4]  4251    1
## log_lik[5,5]  2825    1
## log_lik[5,6]  4927    1
## lp__          1723    1
##
## Samples were drawn using NUTS(diag_e) at Wed Nov 17 20:48:14 2021.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

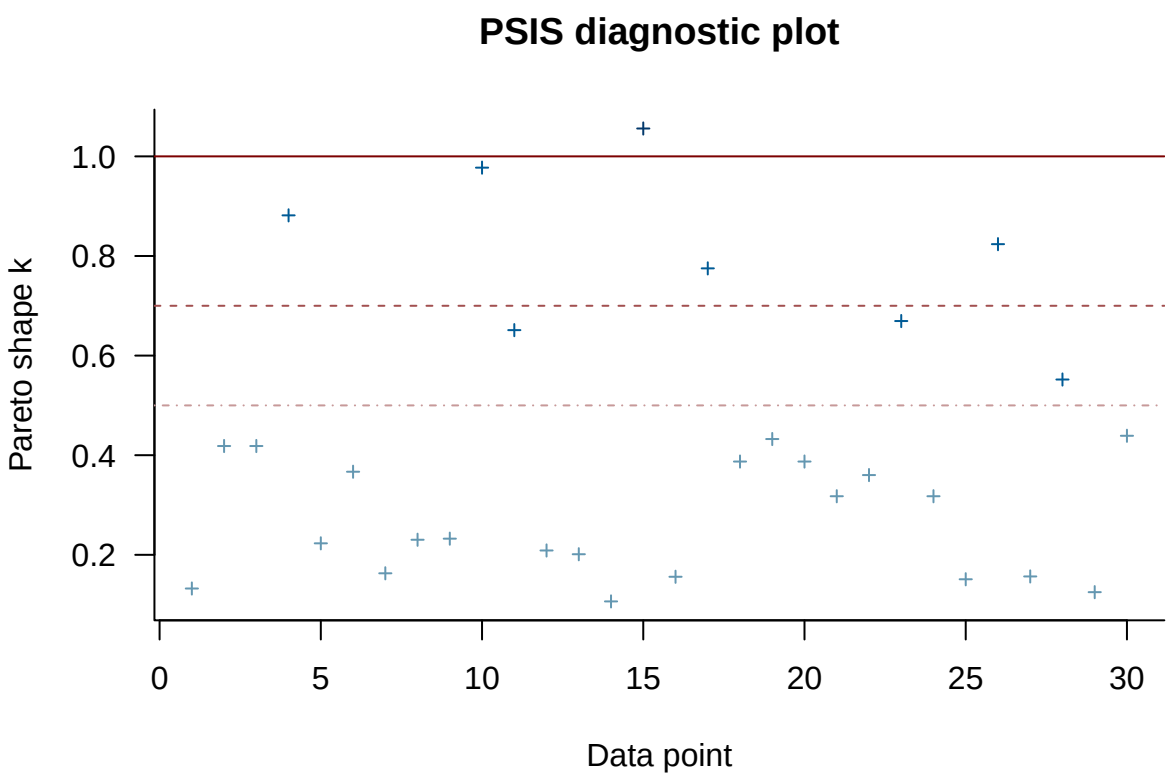
2

```
log_like_separate = extract_log_lik(fit_separate, merge_chains = FALSE)
r_eff <- relative_eff(exp(log_like_separate), cores = 4)
loo_separate <- loo(log_like_separate, r_eff = r_eff, cores = 4)
p_loo_separate <- loo_separate$p_loo
print(loo_separate)
```

```
##
## Computed from 4000 by 30 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -137.8  9.8
## p_loo        19.6  5.3
## looic       275.7 19.7
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     22    73.3%   746
##  (0.5, 0.7]   (ok)        3    10.0%   320
```

```
##    (0.7, 1]   (bad)       4    13.3%   15
##    (1, Inf)   (very bad)  1     3.3%   10
## See help('pareto-k-diagnostic') for details.
```

```
plot(loo_separate)
```



3.

We can the effective number of paramters, p_{eff} using the "p_loo" value from the above table which is $p_{loo-cv} =$19.6253022

4.

From the table above we can see that the $\widehat{k}$ values are mostly good, but the are some very bad values as well. Hence, the model is unreliable.

Pooled model

1.

```
data {
 int<lower=0> N;
 int<lower=0> J;
 vector[N*J] y;
 real mean_mu_prior;
 real<lower=0> mean_sigma_prior;
 real<lower=0> sigma_prior;
```

```
}

parameters {
 real mu;
 real <lower=0> sigma;
}

model {
 //prior
  mu ~ normal(mean_mu_prior, mean_sigma_prior);
  sigma ~ inv_chi_square(sigma_prior);
 //likelihoods
  y ~ normal(mu, sigma);
}

generated quantities {
 real ypred;
 vector[N*J] log_lik;
 ypred = normal_rng(mu, sigma);
  for (n in 1:(N*J)){
      log_lik[n] = normal_lpdf(y[n] | mu, sigma);
   }
}
```

```
mean_mu_prior = 100
mean_sigma_prior = 10
sigma_prior = 10
pooled_data <- list(
  y = unlist(factory),
  N = nrow(factory),
  J = ncol(factory),
 mean_mu_prior = mean_mu_prior,
 mean_sigma_prior = mean_sigma_prior,
 sigma_prior = sigma_prior
)
```

```
fit_pooled = sampling(pooledmodel,
  data = pooled_data,            # named list of data
  chains = 4,             # number of Markov chains
  warmup = 1000,          # number of warmup iterations per chain
  iter = 2000,            # total number of iterations per chain
  cores = 1,              # number of cores (could use one per chain)
  refresh = 0             # no progress shown
  )
```

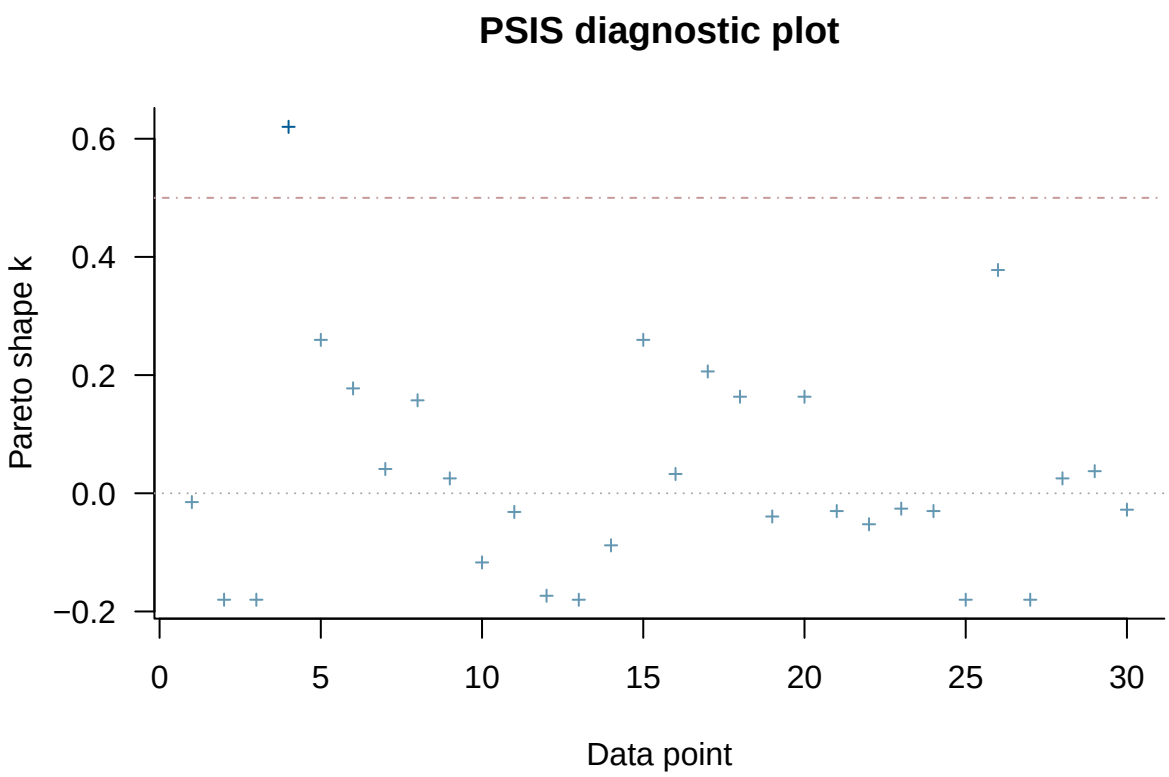
2.

```
log_like_pooled = extract_log_lik(fit_pooled, merge_chains = FALSE)
r_eff <- relative_eff(exp(log_like_pooled), cores = 4)
loo_pooled <- loo(log_like_pooled, r_eff = r_eff, cores = 4)
p_loo_pooled <- loo_pooled$p_loo
print(loo_pooled)
```

```
##
## Computed from 4000 by 30 log-likelihood matrix
```

```
##
##          Estimate   SE
## elpd_loo   -131.1  5.3
## p_loo         2.3  1.0
## looic       262.2 10.5
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     29    96.7%   1419
##  (0.5, 0.7]   (ok)        1     3.3%   276
##    (0.7, 1]   (bad)       0     0.0%   <NA>
##    (1, Inf)   (very bad)  0     0.0%   <NA>
##
## All Pareto k estimates are ok (k < 0.7).
## See help('pareto-k-diagnostic') for details.
```

```
plot(loo_pooled)
```



3. As in the separate model, we can the value using the "p_loo" value from the above table which is $p_{loo-cv} =$2.3327397

4.

From the above table we can see that all the k values are good (at the time of writing). The value is hence reliable.

Hierarchical

1.

```
data {
 int<lower=0> N;
 int<lower=0> J;
 vector[J] y[N];
 real mean_mu_prior;
 real<lower=0> mean_sigma_prior;
 real<lower=0> sigma_prior;
}

parameters {
 real mu;
 real <lower=0> sigma;
 real <lower=0> tau;
 vector[J] mu_i;
}

model {
  //hyperpriors
  mu ~ normal(mean_mu_prior, mean_sigma_prior);
  tau ~ inv_chi_square(sigma_prior);

 //prior
  for(j in 1:J){
    mu_i[j] ~ normal(mu, tau);
  }
  sigma ~ inv_chi_square(tau);

 //likelihoods
 for(j in 1:J){
   y[,j] ~ normal(mu_i[j], sigma);
 }

}

generated quantities {
 real ypred;
 real ypred_7;
 vector[J] log_lik[N];
 ypred = normal_rng(mu_i[6], sigma);
 ypred_7 = normal_rng(mu, sigma);
 for (j in 1:J){
  for (n in 1:N){
      log_lik[n,j] = normal_lpdf(y[n,j] | mu_i[j], sigma);
   }}
}
```

```
mean_mu_prior = 100
mean_sigma_prior = 10
sigma_prior = 10
hierarchical_data <- list(
  y = factory,
```

```
  N = nrow(factory),
  J = ncol(factory),
 mean_mu_prior = mean_mu_prior,
 mean_sigma_prior = mean_sigma_prior,
 sigma_prior = sigma_prior
)
```

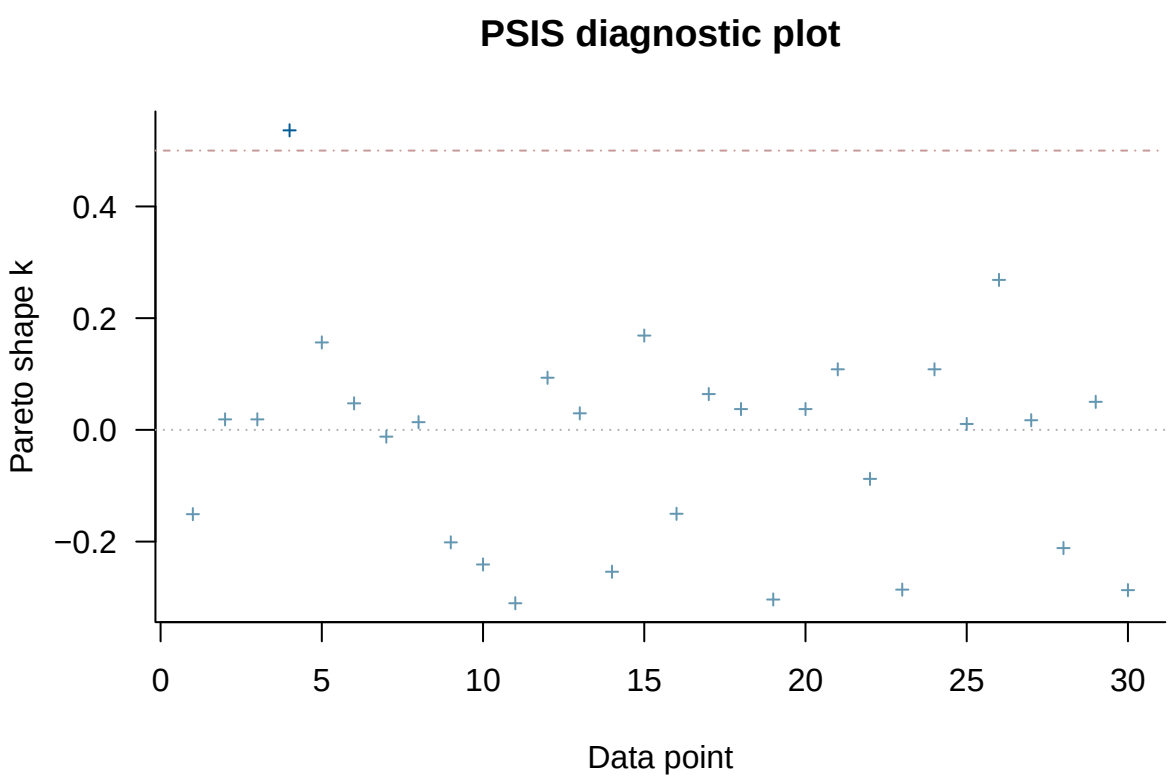
```
fit_hierarchical = sampling(hierarchicalmodel,
  data = hierarchical_data,            # named list of data
  chains = 4,             # number of Markov chains
  warmup = 1000,          # number of warmup iterations per chain
  iter = 2000,            # total number of iterations per chain
  cores = 1,              # number of cores (could use one per chain)
  refresh = 0             # no progress shown
  )
```

2.

```
log_like_hierarchical = extract_log_lik(fit_hierarchical, merge_chains = FALSE)
r_eff <- relative_eff(exp(log_like_hierarchical), cores = 4)
loo_hierarchical <- loo(log_like_hierarchical, r_eff = r_eff, cores = 4)
print(loo_hierarchical)
```

```
##
## Computed from 4000 by 30 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo   -130.9 4.6
## p_loo         2.1 0.9
## looic       261.9 9.2
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     29    96.7%   225
##  (0.5, 0.7]   (ok)        1     3.3%   69
##    (0.7, 1]   (bad)       0     0.0%   <NA>
##    (1, Inf)   (very bad)  0     0.0%   <NA>
##
## All Pareto k estimates are ok (k < 0.7).
## See help('pareto-k-diagnostic') for details.
```

```
plot(loo_hierarchical)
```




```
p_loo_hierarchical <- loo_hierarchical$p_loo
```

3.

As in the separate model, we can the value using the "p_loo" value from the above table which is $p_{loo-cv} =$2.0666404. ## 4. From the above table we can see that the $\widehat{k}$ are mostly good with 1 ok value, so in overall they are all at least ok. Hence, the value is somewhat reliable.

5.

```
loo_compare(loo_separate,loo_pooled,loo_hierarchical)
```

```
##        elpd_diff se_diff
## model3  0.0       0.0
## model2 -0.2       0.7
## model1 -6.9       7.5
```

From the results we can see that the difference in the hierarchical and pooled model are marginal, if any at all, while the separate receives mmuch worse score then the two other. However, since the standard errors are so large, it's hard to be sure of the result.