How does age affect the performance of a Formula One driver?

Anonymous

Contents

Introduction **1**

Data **2**

Models **2**

Weakly Prior Choices **3**

Stan Code **4**

Running the Stan Code **9**

Convergence Evaluation **10**

Posterior Predictive Checks **19**

Model Comparison **24**

Predictive Performance Assessment **25**

Prior Sensitivity Analysis **25**

Which age is the best? **26**

Discussion **27**

Conclusions and Self-Reflections **28**

References **28**

Introduction

Formula One (F1) is the most prestigious auto racing league in the world. The number of drivers in this racing class is only 20, and those taking part could be considered "the best of the best." Given the fierce level of competition, every attribute of the driver could be considered important. An F1 driver must be at least 18 years of age, that is, the participating racers are all adults. It is a well-known fact that motor skills and reflexes of adults deteriorate with age. In this report we explore the relationship between the age and performance of F1 drivers through Bayesian analysis. In particular, we consider and compare three models: separate, pooled, and hierarchical. The models incorporate weakly priors, and related discussion of choices and sensitivity analysis is provided for each model separately. We further conduct posterior predictive analysis and finally provide a discussion on the models and potential future research.

Data

Our analysis problem revolves around extracting posterior statistics yielding information about the relationship between a driver's age and performance. An example of such statistic would be the probability that the best age performance-wise lies within a certain range. The raw data is provided by the ("Ergast Developer API" n.d.). For the purposes of our analysis, we conduct an number of preprocessing steps on this dataset. We use only qualifying as the measure of performance and ignore race results in this analysis. In particular, we consider the difference between a driver and his teammate. In F1, a qualifying session consists of three timed periods, called Q1, Q2, and Q3, where 5 drivers are eliminated at the first two periods. From every qualifying for every driver we calculate the difference between a driver and his teammate from the last period they both participated in. To prune out outliers, we ignore cases where this difference is more than 2.5s, as this is rare and likely due to some unusual event unrelated to the driver's capability. We further subtract from this difference the historical mean value of the driver's difference to his teammates during his whole career to get comparable values for each driver that represent how they performed compared to their career average.

The same underlying data has been utilized by various authors. For example, (Nigro n.d.) examines how odds provided by a neural network model fit to a subset of the data fares against officially provided odds. Related to our analysis, the author notes that the age of the winning driver shows a decreasing linear trend. (Jain n.d.) on the other hand utilizes logistic regression with F_1 score to predict whether or not a driver manages to place on podium, i.e., to attain a top-three position. Similar to us, (van Kesteren n.d.) conducts a Bayesian analysis. The author models the skill of drivers considering factors including constructor advantages and seasonal constructor form. In contrast, our analysis is solely based on exploring whether or not a driver's age provides meaningful insight on his performance.

Models

In this section we enumerate and describe the three models we attempt to fit to the data described previously. Let us first define some general prerequisites. Let N denote the size of the dataset. Further, let $\mathcal{I} = \{1, 2, \dots, N\}$ denote the index set of the dataset, and $\mathcal{A} = \{18, 19, \dots, 43\}$ the set of driver ages present in the dataset. Now let $\mathrm{age}(\cdot)$ be a mapping from $\mathcal{I}$ to $\mathcal{A}$ yielding the age of the sample at index $i \in \mathcal{I}$.

Separate

The separate model is formally defined as follows:

$$\begin{aligned} t_i &\sim \mathrm{N}(\mu_{\mathrm{age}(i)}, \sigma_{\mathrm{age}(i)}), \\ \mu_{\mathrm{age}(i)} &\sim \mathrm{N}(0, 1), \\ \sigma_{\mathrm{age}(i)} &\sim \mathrm{N}(0, 1). \end{aligned}$$

The separate model effectively assumes the defining characteristics of the distribution for t_i are different for each age group.

Pooled

The pooled model is formally defined as follows:

$$\begin{aligned}
t_i &\sim \mathrm{N}(\mu, \sigma), \\
\mu &\sim \mathrm{N}(0, 1), \\
\sigma &\sim \mathrm{N}(0, 1).
\end{aligned}$$

The pooled model simply assumes the characteristics of the distribution generating t_i are independent of driver age.

Hierarchical

The hierarchical model is defined as follows:

$$\begin{aligned}
t_i &\sim \mathrm{N}(\mu_{\mathrm{age}(i)}, \sigma), \\
\mu_{\mathrm{age}(i)} &\sim \mathrm{N}(\mu_{\mathrm{unknown}}, \tau), \\
\mu_{\mathrm{unknown}} &\sim \mathrm{N}(0, 1), \\
\tau &\sim \mathrm{N}(0, 1), \\
\sigma &\sim \mathrm{N}(0, 1).
\end{aligned}$$

In the hierarchical model, the variance of the distribution describing t_i is assumed the same for each age group. However, the mean of this distribution is assumed to be dependent on age. The parameters of the distribution yielding the mean of a particular age group are further sampled from the hyperpriors described above.

Separate with additional parameter

Additionally, we consider following separate model with additional parameter corresponding to teammate of the driver:

$$\begin{aligned}
t_i &\sim \mathrm{N}(\mu_{\mathrm{age}(i)} + \alpha_{\mathrm{teammate(i)}}, \sigma_{\mathrm{age}(i)}), \\
\mu_{\mathrm{age}(i)} &\sim \mathrm{N}(0, 1), \\
\sigma_{\mathrm{age}(i)} &\sim \mathrm{N}(0, 1).
\end{aligned}$$

Here, the α parameter is meant to capture the effect that teammate's level has to the expected difference for driver of certain age. If driver has a very strong teammate, his qualifying differences to the teammate are expected to be larger than usual and vice versa. We also briefly tested pooled and hierarchical model with the additional parameter but didn't see it as necessary to include the full analysis of those models to this report for since their comparison to each other is very similar to the comparison of the models without the parameter, and the comparison between one model without the parameter and one with is enough to see the effect of the parameter. Also, as mentioned in the convergence evaluation section, we were not able to achieve satisfying convergence with the hierarchical model with the additional parameter.

Weakly Prior Choices

In this kind of setting, we make a common assumption that the posterior distribution for t_i is of normal form. We refer to the data for prior information about the mean and variance of this distribution. After preprocessing, we have that the time differences between drivers lie within the interval $[-2.5, 2.5]$, which gives rise to a weakly informative prior distribution for the mean of the distribution describing t_i as $\mathrm{N}(0,1)$. Furthermore, it is evident that the differences between teammates is seldom over one second, which gives rise to a prior distribution for the variance of the distribution describing t_i also as $\mathrm{N}(0,1)$. It turns out that the prior distributions are conjugate for the posterior distribution. For the α parameter we assigned prior $\mathrm{N}(0,0.1)$. It is densinely distributed around zero for the α to not have too big effect and fail to fit the other parameters prope because of it.

Stan Code

Separate model:

```r
cat(readLines('../stan/separate_model.stan'), sep = '\n')
```

```
## // The input data N tells how many age pools we have
## // Total length tells how many rows in the input data
## // time contains the time differences to teammate
## // age contains age of driver at the time of the quali where the difference happened, where each row
## // corrseponds to to the row of time
## // model index is age transformed to corresponding model index (26 ages = 26 models)
## // rep ages contains the ages (models) for which replicated dataset is generated for predictive chec
## // rep_length is the length of the replicated dataset
## //
##
## functions {
##    // check if vector contains value
##    // if(r_in(3,{1,2,3,4})) will evaluate as 1
##   int r_in(int pos,int[] pos_var) {
##     for (p in 1:(size(pos_var))) {
##        if (pos_var[p]==pos) {
##           return 1;
##        }
##     }
##     return 0;
##   }
## }
##
## data {
##   int<lower=0> N;
##   int total_length;
##   real time[total_length];
##   int age[total_length];
##   int model_index[total_length];
##   int rep_ages[3];
##   int rep_length;
## }
##
## parameters {
##   real mu[N];
##   real<lower=0> sigma[N];
##
## }
##
```

```
##
##
## model {
##   mu ~ normal(0,1);        //prior
##   sigma ~ normal(0,1);     //prior
##   for(j in 1:total_length){
##     time[j] ~ normal(mu[model_index[j]], sigma[model_index[j]]);
##
##   }
## }
##
##
##
## generated quantities{
##   real yrep[rep_length];
##   real log_lik[total_length];
##   {
##   int  rep_index = 1;
##     for(j in 1:total_length){
##       if(r_in(age[j], rep_ages)) {
##         yrep[rep_index] = normal_rng(mu[model_index[j]], sigma[model_index[j]]);
##         rep_index += 1;
##       }
##     }
##   }
##
##   for(j in 1:total_length){
##     log_lik[j] =  normal_lpdf(time[j] | mu[model_index[j]], sigma[model_index[j]]);
##   }
## }
```

Pooled model:

```
cat(readLines('../stan/pooled_model.stan'), sep = '\n')
```

```
## // The input data N tells how many age pools we have
## // Total length tells how many rows in the input data
## // time contains the time differences to teammate
## // age contains age of driver at the time of the quali where the difference happened, where each row
## // corrseponds to to the row of time
## // rep ages contains the ages (models) for which replicated dataset is generated for predictive check
## // rep_length is the length of the replicated dataset
## //
## functions {
##    // check if vector contains value
##    // if(r_in(3,{1,2,3,4})) will evaluate as 1
##   int r_in(int pos,int[] pos_var) {
##     for (p in 1:(size(pos_var))) {
##        if (pos_var[p]==pos) {
##           return 1;
##        }
##     }
##     return 0;
##   }
## }
##
## data {
##   int<lower=0> N;
##   int total_length;
##   real time[total_length];
##   int age[total_length];
##   int rep_ages[3];
##   int rep_length;
## }
##
##
## parameters {
##   real mu;
##   real<lower=0> sigma;
## }
##
## model {
##   mu ~ normal(0,1);        //prior
##   sigma ~ normal(0,1);     //prior
##   time ~ normal(mu, sigma);
## }
##
## generated quantities{
##   real yrep[rep_length];
##   real log_lik[total_length];
##   {
##   int  rep_index = 1;
##     for(j in 1:total_length){
##       if(r_in(age[j], rep_ages)) {
##         yrep[rep_index] = normal_rng(mu, sigma);
##         rep_index += 1;
##       }
##     }
##   }
##   for(j in 1:total_length){
##     log_lik[j] =  normal_lpdf(time[j] | mu, sigma);
##   }
## }
```

Hierarchical model:

```
cat(readLines('../stan/hierarchical_model.stan'), sep = '\n')
```

```
## // The input data N tells how many age pools we have
## // Total length tells how many rows in the input data
## // time contains the time differences to teammate
## // age contains age of driver at the time of the quali where the difference happened, where each row
## // corrseponds to to the row of time
## // model index is age transformed to corresponding model index (26 ages = 26 models)
## // rep ages contains the ages (models) for which replicated dataset is generated for predictive chec
## // rep_length is the length of the replicated dataset
## //
##
## functions {
##    // if(r_in(3,{1,2,3,4})) will evaluate as 1
##   int r_in(int pos,int[] pos_var) {
##     for (p in 1:(size(pos_var))) {
##        if (pos_var[p]==pos) {
##           return 1;
##        }
##     }
##     return 0;
##   }
## }
##
## data {
##   int<lower=0> N;
##   int total_length;
##   real time[total_length];
##   int age[total_length];
##   int model_index[total_length];
##   int rep_ages[3];
##   int rep_length;
## }
##
##
## parameters {
##   real mu[N];
##   real<lower=0> sigma[N];
##   real<lower=0> tau;
##   real mu_mean;
##   real sigma_mean;
##   real<lower=0> sigma_tau;
## }
##
## model {
##   mu_mean ~ normal(0,1);    //hyperprior
##   tau ~ normal(0,1);       //hyperprior
##   mu ~ normal(mu_mean, tau);
##   sigma_mean ~ normal(0,1); //hyperprior
##   sigma_tau ~ normal(0,1); //hyperprior
##   sigma ~ normal(sigma_mean, sigma_tau);
##   for(j in 1:total_length){
##     time[j] ~ normal(mu[model_index[j]], sigma[model_index[j]]);
##   }
## }
##
## generated quantities{
##   real yrep[rep_length];
##   real log_lik[total_length];
##   {
##   int  rep_index = 1;
##     for(j in 1:total_length){
##       if(r_in(age[j], rep_ages)) {
##         yrep[rep_index] = normal_rng(mu[model_index[j]], sigma[model_index[j]]);
##         rep_index += 1;
##       }
##     }
##   }
##   for(j in 1:total_length){
##     log_lik[j] = normal_lpdf(time[j] | mu[model_index[j]], sigma[model_index[j]]);
##   }
## }
```

Separate model with additional parameter:

```r
cat(readLines('../stan/separate_model_ids.stan'), sep = '\n')
```

```
## // The input data N tells how many age pools we have
## // Total length tells how many rows in the input data
## // time contains the time differences to teammate
## // age contains age of driver at the time of the quali where the difference happened, where each row
## // corrseponds to to the row of time
## // model index is age transformed to corresponding model index (26 ages = 26 models)
## // rep ages contains the ages (models) for which replicated dataset is generated for predictive chec
## // rep_length is the length of the replicated dataset
## // driver_id contains driver_id's of teammates where each row corresponds to row of time
## // driver_count is the amount of drivers in the dataset
## //
##
## functions {
##    // if(r_in(3,{1,2,3,4})) will evaluate as 1
##   int r_in(int pos,int[] pos_var) {
##     for (p in 1:(size(pos_var))) {
##        if (pos_var[p]==pos) {
##           return 1;
##        }
##     }
##     return 0;
##   }
## }
##
## data {
##   int<lower=0> N;
##   int total_length;
##   real time[total_length];
##   int age[total_length];
##   int model_index[total_length];
##   int rep_ages[3];
##   int rep_length;
##   int driver_id[total_length];
##   int driver_count;
## }
##
##
## parameters {
##   real mu[N];
##   real<lower=0> sigma[N];
##   real a[driver_count];
## }
##
##
## model {
##   mu ~ normal(0,1);        //prior
##   sigma ~ normal(0,1);     //prior
##   a ~ normal(0,0.1);       //prior
##   for(j in 1:total_length){
##     time[j] ~ normal(mu[model_index[j]] + a[driver_id[j]], sigma[model_index[j]]);
##   }
## }
##
## generated quantities{
##   real yrep[rep_length];
##   real yrep_id[rep_length];      //replicated dataset where id is assumed to be known
##   real log_lik[total_length];
##   real log_lik_id[total_length]; //log likelihood where teammate is assumed to be known
##   {
##   int  rep_index = 1;
##     for(j in 1:total_length){
##       if(r_in(age[j], rep_ages)) {
##         yrep[rep_index] = normal_rng(mu[model_index[j]], sigma[model_index[j]]);
##         rep_index += 1;
##       }
##     }
##   }
##   {
##   int  rep_index = 1;
##     for(j in 1:total_length){
##       if(r_in(age[j], rep_ages)) {
##         yrep_id[rep_index] = normal_rng(mu[model_index[j]] + a[driver_id[j]], sigma[model_index[j]])
##         rep_index += 1;
##       }
##     }
##   }
##   for (j in 1:total_length){
##     log_lik[j] = normal_lpdf(time[j] | mu[model_index[j]], sigma[model_index[j]]);
## }
##   for (j in 1:total_length){
##     log_lik_id[j] = normal_lpdf(time[j] | mu[model_index[j]] + a[driver_id[j]], sigma[model_index[j]
##  }
## }
```

Running the Stan Code

```r
data = read.csv(file ="../../data/quali_differences_processed.csv")
data$teammateId = mapvalues(data$teammateId, unique(data$teammateId),
                            1:length(unique(data$teammateId)))
data$model_index = data$age-18+1
```

```
options(mc.cores = parallel::detectCores())
rstan_options(auto_write=TRUE)

ages = data$age
N = length(unique(ages))
y_rep_ages = c(19, 27, 41) #ages to generate replicated datasets for model evaluation
y_rep_length = sum(data$age %in% y_rep_ages)
y = data[data$age %in% y_rep_ages,]$difference

y_rep_groups = vector(mode="integer", length=y_rep_length)
rep_index = 1
for(j in 1:nrow(data)){
  if(ages[j] %in% y_rep_ages) {
    y_rep_groups[rep_index] = ages[j]
    rep_index = rep_index + 1
  }
}

stan_data <- list(N = N, total_length = nrow(data), time = data$difference,
                  age = ages, min_age = min(ages), model_index=data$model_index,
                  rep_ages = y_rep_ages, rep_length = y_rep_length)
stan_data_id <- list(N = N, total_length = nrow(data), time = data$difference,
                     age = ages, min_age = min(ages), model_index=data$model_index,
                     rep_ages = y_rep_ages, rep_length = y_rep_length,
                     driver_id = data$teammateId,
                     driver_count = length(unique(data$teammateId)))
```

```
fit_separate <- stan(file = "../stan/separate_model.stan", data = stan_data)
fit_hierarchical <- stan(file = "../stan/hierarchical_model.stan", data = stan_data)
fit_pooled <- stan(file = "../stan/pooled_model.stan", data = stan_data)

fit_separate_id <- stan(file = "../stan/separate_model_ids.stan", data = stan_data_id,
                        iter=6000)
```

Convergence Evaluation

According to (Vehtari et al. 2021), in the context of MCMC sampling, it is prudent to run at least four chains, which Stan performs by default. The authors further guide to accept the sample on the condition that $\widehat{R} < 1.05$, which we observe here.

Fit for each model showing $\widehat{R}$ and ESS:

```
#Separate
print(fit_separate, pars=c('mu', 'sigma'))
```

```
## Inference for Stan model: separate_model.
## 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% n_eff Rhat
## mu[1]        0.07       0 0.20 -0.31 -0.05  0.08  0.20  0.47  6270    1
## mu[2]      0.46       0 0.08  0.30  0.41  0.46  0.52  0.63  5166    1
## mu[3]     -0.01       0 0.05 -0.11 -0.04 -0.01  0.03  0.10  6769    1
## mu[4]      0.00       0 0.03 -0.07 -0.02  0.00  0.02  0.07  5384    1
## mu[5]      0.01       0 0.03 -0.04  0.00  0.01  0.03  0.07  6656    1
## mu[6]      0.00       0 0.02 -0.04 -0.01  0.00  0.02  0.05  5942    1
## mu[7]     -0.09       0 0.03 -0.14 -0.10 -0.09 -0.07 -0.04  5663    1
## mu[8]     -0.06       0 0.03 -0.11 -0.08 -0.06 -0.04 -0.01  6214    1
## mu[9]     -0.08       0 0.02 -0.12 -0.09 -0.08 -0.06 -0.03  5845    1
## mu[10]    -0.16       0 0.03 -0.21 -0.18 -0.16 -0.14 -0.11  6173    1
## mu[11]    -0.05       0 0.03 -0.10 -0.06 -0.05 -0.03  0.00  5656    1
## mu[12]     0.00       0 0.03 -0.05 -0.02  0.00  0.02  0.05  6469    1
## mu[13]    -0.01       0 0.03 -0.06 -0.03 -0.01  0.01  0.04  5644    1
## mu[14]     0.02       0 0.03 -0.03  0.00  0.02  0.04  0.08  6885    1
## mu[15]     0.19       0 0.03  0.12  0.17  0.19  0.21  0.25  5833    1
## mu[16]     0.08       0 0.03  0.02  0.06  0.08  0.10  0.15  5558    1
## mu[17]     0.07       0 0.04  0.00  0.05  0.07  0.09  0.14  6088    1
## mu[18]     0.13       0 0.03  0.06  0.11  0.13  0.15  0.20  6479    1
## mu[19]     0.01       0 0.04 -0.07 -0.02  0.01  0.04  0.09  4638    1
## mu[20]     0.15       0 0.04  0.07  0.12  0.15  0.18  0.23  6492    1
## mu[21]     0.14       0 0.06  0.02  0.10  0.14  0.18  0.26  5671    1
## mu[22]     0.12       0 0.05  0.01  0.08  0.12  0.15  0.23  6136    1
## mu[23]     0.07       0 0.09 -0.10  0.01  0.07  0.13  0.23  6126    1
## mu[24]     0.04       0 0.09 -0.13 -0.02  0.04  0.10  0.22  5296    1
## mu[25]     0.45       0 0.12  0.21  0.37  0.45  0.53  0.70  5990    1
## mu[26]     0.31       0 0.15  0.00  0.21  0.31  0.41  0.60  5937    1
## sigma[1]   0.78       0 0.15  0.55  0.68  0.76  0.86  1.12  4887    1
## sigma[2]   0.59       0 0.06  0.48  0.54  0.58  0.63  0.72  5131    1
## sigma[3]   0.57       0 0.04  0.50  0.54  0.57  0.59  0.64  7116    1
## sigma[4]   0.56       0 0.02  0.52  0.54  0.56  0.58  0.61  6514    1
## sigma[5]   0.59       0 0.02  0.55  0.58  0.59  0.60  0.63  7333    1
## sigma[6]   0.59       0 0.02  0.56  0.58  0.59  0.60  0.62  7807    1
## sigma[7]   0.64       0 0.02  0.61  0.63  0.64  0.65  0.68  6356    1
## sigma[8]   0.65       0 0.02  0.62  0.64  0.65  0.66  0.69  6234    1
## sigma[9]   0.63       0 0.02  0.60  0.62  0.63  0.64  0.66  7003    1
## sigma[10]  0.63       0 0.02  0.59  0.62  0.63  0.64  0.66  6480    1
## sigma[11]  0.67       0 0.02  0.63  0.66  0.67  0.68  0.70  7203    1
## sigma[12]  0.61       0 0.02  0.57  0.59  0.61  0.62  0.65  6037    1
## sigma[13]  0.64       0 0.02  0.60  0.62  0.64  0.65  0.67  6331    1
## sigma[14]  0.67       0 0.02  0.63  0.65  0.67  0.68  0.71  6639    1
## sigma[15]  0.61       0 0.02  0.57  0.59  0.61  0.62  0.66  5685    1
## sigma[16]  0.63       0 0.03  0.58  0.61  0.62  0.64  0.68  7265    1
## sigma[17]  0.65       0 0.03  0.60  0.63  0.65  0.67  0.70  6669    1
## sigma[18]  0.63       0 0.03  0.59  0.61  0.63  0.65  0.68  7316    1
## sigma[19]  0.67       0 0.03  0.61  0.65  0.67  0.68  0.73  5130    1
## sigma[20]  0.54       0 0.03  0.49  0.52  0.54  0.56  0.60  5287    1
## sigma[21]  0.54       0 0.04  0.46  0.51  0.54  0.57  0.64  5702    1
## sigma[22]  0.39       0 0.04  0.32  0.36  0.39  0.42  0.48  5893    1
## sigma[23]  0.54       0 0.06  0.44  0.50  0.53  0.58  0.68  5150    1
## sigma[24]  0.67       0 0.06  0.56  0.62  0.66  0.71  0.81  5292    1
## sigma[25]  0.74       0 0.09  0.59  0.68  0.73  0.80  0.94  5807    1
## sigma[26]  0.64       0 0.12  0.46  0.56  0.62  0.71  0.91  4378    1
##
## Samples were drawn using NUTS(diag_e) at Mon Dec 06 22:15:07 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).
```

```r
#hierarchical
print(fit_hierarchical, pars=c('mu', 'sigma'))
```

```
## Inference for Stan model: hierarchical_model.
## 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% n_eff Rhat
## mu[1]      0.06       0 0.10 -0.13 -0.01  0.06  0.12  0.25  6649    1
## mu[2]      0.32       0 0.08  0.16  0.26  0.32  0.37  0.48  4632    1
## mu[3]      0.00       0 0.05 -0.09 -0.03  0.00  0.03  0.10  7074    1
## mu[4]      0.00       0 0.03 -0.06 -0.02  0.00  0.03  0.07  7744    1
## mu[5]      0.02       0 0.03 -0.04  0.00  0.02  0.03  0.07  7801    1
## mu[6]      0.00       0 0.02 -0.04 -0.01  0.00  0.02  0.05  6631    1
## mu[7]     -0.08       0 0.03 -0.13 -0.10 -0.08 -0.06 -0.03  6674    1
## mu[8]     -0.06       0 0.02 -0.10 -0.07 -0.06 -0.04 -0.01  6270    1
## mu[9]     -0.07       0 0.02 -0.11 -0.09 -0.07 -0.06 -0.03  7258    1
## mu[10]    -0.15       0 0.03 -0.20 -0.17 -0.15 -0.13 -0.10  6242    1
## mu[11]    -0.04       0 0.03 -0.09 -0.06 -0.04 -0.02  0.01  7214    1
## mu[12]     0.00       0 0.02 -0.05 -0.02  0.00  0.02  0.05  7669    1
## mu[13]     0.00       0 0.03 -0.05 -0.02 -0.01  0.01  0.04  7857    1
## mu[14]     0.02       0 0.03 -0.03  0.00  0.02  0.04  0.08  6905    1
## mu[15]     0.18       0 0.03  0.12  0.16  0.18  0.20  0.24  6702    1
## mu[16]     0.08       0 0.03  0.01  0.06  0.08  0.10  0.15  6846    1
## mu[17]     0.07       0 0.03  0.00  0.05  0.07  0.09  0.14  7411    1
## mu[18]     0.12       0 0.03  0.06  0.10  0.12  0.14  0.19  6465    1
## mu[19]     0.01       0 0.04 -0.06 -0.01  0.02  0.04  0.09  6786    1
## mu[20]     0.14       0 0.04  0.06  0.11  0.14  0.17  0.22  7259    1
## mu[21]     0.12       0 0.06  0.00  0.08  0.12  0.16  0.23  7224    1
## mu[22]     0.10       0 0.06 -0.02  0.06  0.10  0.14  0.23  7011    1
## mu[23]     0.06       0 0.07 -0.08  0.01  0.06  0.12  0.21  8031    1
## mu[24]     0.05       0 0.07 -0.09  0.00  0.05  0.09  0.18  7620    1
## mu[25]     0.28       0 0.09  0.11  0.22  0.28  0.34  0.46  4969    1
## mu[26]     0.16       0 0.10 -0.02  0.10  0.16  0.23  0.36  5976    1
## sigma[1]   0.63       0 0.04  0.56  0.61  0.63  0.65  0.71  4679    1
## sigma[2]   0.61       0 0.04  0.54  0.59  0.61  0.63  0.68  4776    1
## sigma[3]   0.59       0 0.03  0.53  0.57  0.59  0.61  0.65  4285    1
## sigma[4]   0.58       0 0.02  0.53  0.56  0.58  0.59  0.62  3505    1
## sigma[5]   0.60       0 0.02  0.56  0.58  0.60  0.61  0.63  5201    1
## sigma[6]   0.59       0 0.02  0.56  0.58  0.59  0.60  0.62  5121    1
## sigma[7]   0.64       0 0.02  0.61  0.63  0.64  0.65  0.67  6869    1
## sigma[8]   0.64       0 0.02  0.61  0.63  0.64  0.66  0.68  5199    1
## sigma[9]   0.63       0 0.01  0.60  0.62  0.63  0.64  0.66  7379    1
## sigma[10]  0.63       0 0.02  0.60  0.61  0.63  0.64  0.66  7921    1
## sigma[11]  0.66       0 0.02  0.63  0.65  0.66  0.67  0.69  4088    1
## sigma[12]  0.61       0 0.02  0.58  0.60  0.61  0.62  0.64  6375    1
## sigma[13]  0.63       0 0.02  0.60  0.62  0.63  0.64  0.67  7010    1
## sigma[14]  0.65       0 0.02  0.62  0.64  0.65  0.67  0.69  4736    1
## sigma[15]  0.61       0 0.02  0.58  0.60  0.61  0.62  0.65  6636    1
## sigma[16]  0.62       0 0.02  0.58  0.61  0.62  0.64  0.67  5633    1
## sigma[17]  0.64       0 0.02  0.60  0.62  0.64  0.65  0.68  7330    1
## sigma[18]  0.63       0 0.02  0.59  0.61  0.63  0.64  0.67  5876    1
## sigma[19]  0.65       0 0.02  0.61  0.63  0.65  0.66  0.69  4717    1
## sigma[20]  0.57       0 0.03  0.51  0.55  0.57  0.59  0.63  2864    1
## sigma[21]  0.59       0 0.03  0.52  0.56  0.59  0.61  0.65  2926    1
## sigma[22]  0.54       0 0.05  0.43  0.51  0.55  0.58  0.63  1706    1
## sigma[23]  0.59       0 0.04  0.51  0.57  0.60  0.62  0.66  4271    1
## sigma[24]  0.63       0 0.03  0.56  0.61  0.63  0.65  0.70  6064    1
## sigma[25]  0.65       0 0.04  0.58  0.62  0.64  0.67  0.73  4635    1
## sigma[26]  0.62       0 0.04  0.54  0.60  0.62  0.64  0.69  5151    1
##
## Samples were drawn using NUTS(diag_e) at Mon Dec 06 22:16:21 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).
```

```
#pooled
print(fit_pooled, pars=c('mu', 'sigma'))
```

```
## Inference for Stan model: pooled_model.
## 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% n_eff Rhat
## mu    0.00       0 0.01 -0.01 0.00 0.00 0.00  0.01  3963    1
## sigma 0.63       0 0.00  0.62 0.63 0.63 0.63  0.64  3748    1
##
## Samples were drawn using NUTS(diag_e) at Mon Dec 06 22:17:14 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).
```

```
#Separate_id (separate with additional parameter)
print(fit_separate_id, pars=c('mu', 'sigma', 'a'))
```

```
## Inference for Stan model: separate_model_ids.
## 4 chains, each with iter=6000; warmup=3000; thin=1;
## post-warmup draws per chain=3000, total post-warmup draws=12000.
##
##            mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## mu[1]     -0.06       0 0.20 -0.46 -0.19 -0.05  0.07  0.35 20127    1
## mu[2]      0.39       0 0.09  0.21  0.33  0.39  0.45  0.57 22497    1
## mu[3]     -0.03       0 0.06 -0.14 -0.07 -0.03  0.01  0.08 20080    1
## mu[4]     -0.07       0 0.04 -0.15 -0.10 -0.08 -0.05  0.00 16973    1
## mu[5]     -0.06       0 0.03 -0.12 -0.08 -0.06 -0.04  0.00 17101    1
## mu[6]     -0.05       0 0.03 -0.10 -0.07 -0.05 -0.04  0.00 15931    1
## mu[7]     -0.14       0 0.03 -0.20 -0.16 -0.14 -0.12 -0.08 16788    1
## mu[8]     -0.14       0 0.03 -0.20 -0.16 -0.14 -0.13 -0.09 16259    1
## mu[9]     -0.15       0 0.03 -0.20 -0.17 -0.15 -0.13 -0.10 15609    1
## mu[10]    -0.22       0 0.03 -0.28 -0.24 -0.22 -0.20 -0.17 18174    1
## mu[11]    -0.11       0 0.03 -0.17 -0.13 -0.11 -0.09 -0.06 16262    1
## mu[12]    -0.06       0 0.03 -0.12 -0.08 -0.06 -0.04 -0.01 16174    1
## mu[13]    -0.10       0 0.03 -0.16 -0.12 -0.10 -0.08 -0.05 15882    1
## mu[14]    -0.07       0 0.03 -0.13 -0.09 -0.07 -0.05 -0.01 18251    1
## mu[15]      0.06       0 0.03 -0.01  0.03  0.06  0.08  0.12 18336    1
## mu[16]     -0.02       0 0.04 -0.09 -0.04 -0.02  0.01  0.06 20056    1
## mu[17]      0.04       0 0.04 -0.04  0.01  0.04  0.06  0.11 18743    1
## mu[18]      0.05       0 0.04 -0.02  0.03  0.05  0.08  0.13 18751    1
## mu[19]     -0.05       0 0.04 -0.14 -0.08 -0.05 -0.03  0.03 18911    1
## mu[20]      0.09       0 0.04  0.01  0.06  0.09  0.12  0.18 17210    1
## mu[21]      0.06       0 0.07 -0.07  0.01  0.06  0.10  0.19 21382    1
## mu[22]      0.03       0 0.06 -0.09 -0.01  0.03  0.07  0.15 18328    1
## mu[23]      0.06       0 0.09 -0.12 -0.01  0.06  0.12  0.25 20751    1
## mu[24]      0.05       0 0.09 -0.12  0.00  0.05  0.11  0.22 23036    1
## mu[25]      0.38       0 0.13  0.13  0.30  0.38  0.46  0.63 21597    1
## mu[26]      0.14       0 0.15 -0.16  0.04  0.14  0.24  0.44 20456    1
## sigma[1]    0.79       0 0.15  0.55  0.68  0.76  0.87  1.15 15397    1
## sigma[2]    0.60       0 0.06  0.49  0.55  0.59  0.64  0.74 21418    1
## sigma[3]    0.56       0 0.04  0.49  0.53  0.56  0.58  0.63 28025    1
## sigma[4]    0.55       0 0.02  0.51  0.53  0.55  0.56  0.60 25641    1
## sigma[5]    0.57       0 0.02  0.54  0.56  0.57  0.58  0.61 26695    1
## sigma[6]    0.58       0 0.02  0.55  0.57  0.58  0.59  0.61 27743    1
## sigma[7]    0.63       0 0.02  0.60  0.62  0.63  0.64  0.67 27212    1
## sigma[8]    0.64       0 0.02  0.60  0.63  0.64  0.65  0.67 30179    1
## sigma[9]    0.61       0 0.02  0.58  0.60  0.61  0.62  0.64 26943    1
## sigma[10]   0.60       0 0.02  0.57  0.59  0.60  0.62  0.64 30060    1
## sigma[11]   0.65       0 0.02  0.62  0.64  0.65  0.66  0.69 26603    1
## sigma[12]   0.59       0 0.02  0.56  0.58  0.59  0.60  0.62 28290    1
## sigma[13]   0.62       0 0.02  0.58  0.60  0.62  0.63  0.66 29305    1
## sigma[14]   0.63       0 0.02  0.59  0.62  0.63  0.64  0.67 25328    1
## sigma[15]   0.59       0 0.02  0.55  0.58  0.59  0.61  0.64 27364    1
## sigma[16]   0.61       0 0.02  0.56  0.59  0.61  0.62  0.66 28041    1
## sigma[17]   0.65       0 0.03  0.60  0.63  0.65  0.67  0.70 27489    1
## sigma[18]   0.61       0 0.02  0.56  0.59  0.61  0.63  0.66 27496    1
## sigma[19]   0.64       0 0.03  0.59  0.62  0.64  0.66  0.70 26643    1
## sigma[20]   0.52       0 0.03  0.47  0.50  0.52  0.54  0.58 23645    1
## sigma[21]   0.55       0 0.05  0.47  0.52  0.55  0.58  0.65 22945    1
## sigma[22]   0.38       0 0.04  0.31  0.35  0.37  0.40  0.46 23891    1
## sigma[23]   0.53       0 0.06  0.43  0.49  0.53  0.57  0.67 23032    1
## sigma[24]   0.58       0 0.06  0.48  0.54  0.58  0.62  0.71 21340    1
## sigma[25]   0.72       0 0.09  0.57  0.65  0.71  0.77  0.92 21871    1
## sigma[26]   0.64       0 0.11  0.46  0.56  0.62  0.70  0.90 18737    1
## a[1]        0.07       0 0.05 -0.03  0.04  0.07  0.11  0.18 23466    1
## a[2]        0.11       0 0.05  0.02  0.08  0.11  0.14  0.20 25521    1
## a[3]        0.26       0 0.04  0.19  0.24  0.26  0.29  0.33 21833    1
## a[4]        0.12       0 0.04  0.05  0.10  0.12  0.14  0.19 17978    1
## a[5]        0.12       0 0.05  0.01  0.08  0.12  0.15  0.22 25570    1
## a[6]        0.02       0 0.06 -0.09 -0.01  0.02  0.06  0.13 27622    1
## a[7]       -0.09       0 0.07 -0.23 -0.14 -0.09 -0.04  0.05 27858    1
## a[8]        0.12       0 0.04  0.04  0.09  0.12  0.15  0.20 22257    1
## a[9]        0.20       0 0.04  0.11  0.17  0.20  0.23  0.28 22609    1
## a[10]      -0.05       0 0.08 -0.19 -0.10 -0.05  0.00  0.10 25881    1
## a[11]       0.10       0 0.04  0.03  0.08  0.10  0.13  0.18 20880    1
## a[12]      -0.03       0 0.08 -0.19 -0.09 -0.04  0.02  0.12 30277    1
## a[13]       0.09       0 0.04  0.01  0.07  0.09  0.12  0.17 22183    1
## a[14]       0.17       0 0.05  0.09  0.14  0.17  0.20  0.26 18529    1
## a[15]      -0.03       0 0.05 -0.12 -0.06 -0.03  0.00  0.06 23402    1
## a[16]       0.03       0 0.04 -0.04  0.00  0.03  0.06  0.11 19970    1
## a[17]       0.14        0 0.05  0.04  0.10  0.14  0.17  0.23 23788    1
## a[18]       0.17        0 0.04  0.10  0.15  0.17  0.20  0.24 16908    1
## a[19]       0.13        0 0.05  0.03  0.09  0.13  0.16  0.22 24553    1
## a[20]       0.06        0 0.08 -0.10  0.01  0.06  0.11  0.21 28986    1
## a[21]       0.21        0 0.04  0.14  0.18  0.21  0.23  0.28 19235    1
## a[22]      -0.05        0 0.06 -0.17 -0.09 -0.05 -0.01  0.07 26157    1
## a[23]       0.06        0 0.05 -0.04  0.02  0.06  0.09  0.16 23467    1
## a[24]      -0.09        0 0.07 -0.23 -0.14 -0.09 -0.04  0.05 26630    1
## a[25]       0.01        0 0.06 -0.10 -0.03  0.01  0.05  0.12 25281    1
## a[26]      -0.05        0 0.08 -0.21 -0.10 -0.05  0.00  0.10 29949    1
## a[27]       0.12        0 0.07 -0.01  0.08  0.12  0.17  0.26 30206    1
## a[28]      -0.03        0 0.10 -0.22 -0.09 -0.03  0.04  0.16 32772    1
## a[29]      -0.15        0 0.08 -0.31 -0.21 -0.15 -0.10  0.00 27641    1
## a[30]       0.34        0 0.04  0.26  0.31  0.34  0.37  0.43 21735    1
## a[31]       0.10        0 0.06 -0.01  0.07  0.10  0.14  0.22 26006    1
## a[32]      -0.14        0 0.07 -0.28 -0.18 -0.14 -0.09  0.00 29641    1
## a[33]       0.03        0 0.08 -0.12 -0.02  0.03  0.08  0.18 29215    1
## a[34]       0.01        0 0.06 -0.11 -0.03  0.01  0.05  0.14 27919    1
## a[35]      -0.10        0 0.10 -0.29 -0.17 -0.10 -0.03  0.09 28522    1
## a[36]      -0.06        0 0.09 -0.25 -0.13 -0.06  0.00  0.12 27838    1
## a[37]       0.03        0 0.07 -0.10 -0.02  0.03  0.07  0.15 27263    1
## a[38]       0.00        0 0.09 -0.17 -0.06  0.00  0.07  0.18 31581    1
## a[39]      -0.22        0 0.07 -0.36 -0.27 -0.22 -0.17 -0.07 22757    1
## a[40]       0.00        0 0.09 -0.17 -0.06  0.00  0.06  0.17 30025    1
## a[41]      -0.11        0 0.09 -0.28 -0.17 -0.11 -0.05  0.06 28017    1
## a[42]      -0.18        0 0.08 -0.34 -0.23 -0.18 -0.13 -0.02 28040    1
## a[43]       0.03        0 0.06 -0.09 -0.01  0.03  0.07  0.14 24438    1
## a[44]      -0.07        0 0.09 -0.24 -0.13 -0.07 -0.02  0.10 29870    1
## a[45]       0.01        0 0.09 -0.15 -0.04  0.01  0.07  0.18 30287    1
## a[46]       0.01        0 0.08 -0.15 -0.04  0.01  0.07  0.18 29676    1
## a[47]       0.08        0 0.06 -0.03  0.04  0.08  0.12  0.19 25225    1
## a[48]      -0.02        0 0.08 -0.16 -0.07 -0.02  0.03  0.13 26071    1
## a[49]      -0.18        0 0.09 -0.35 -0.24 -0.18 -0.13 -0.02 25651    1
## a[50]      -0.06        0 0.09 -0.23 -0.12 -0.06 -0.01  0.10 27531    1
## a[51]      -0.06        0 0.07 -0.19 -0.10 -0.06 -0.01  0.07 25786    1
## a[52]      -0.04        0 0.10 -0.24 -0.11 -0.04  0.02  0.15 29838    1
## a[53]       0.01        0 0.10 -0.19 -0.06  0.01  0.07  0.20 32029    1
## a[54]       0.04        0 0.06 -0.08  0.00  0.04  0.08  0.16 28050    1
## a[55]       0.09        0 0.07 -0.05  0.04  0.09  0.13  0.22 27044    1
## a[56]      -0.04        0 0.10 -0.23 -0.10 -0.04  0.02  0.14 26757    1
## a[57]       0.12        0 0.06  0.00  0.08  0.12  0.17  0.25 28106    1
## a[58]       0.00        0 0.06 -0.12 -0.04  0.00  0.04  0.12 26319    1
## a[59]      -0.01        0 0.10 -0.21 -0.08 -0.01  0.06  0.18 27272    1
## a[60]      -0.15        0 0.06 -0.27 -0.19 -0.15 -0.10 -0.02 28238    1
## a[61]      -0.14        0 0.07 -0.27 -0.18 -0.14 -0.09  0.00 26449    1
## a[62]      -0.06        0 0.10 -0.25 -0.12 -0.06  0.01  0.13 31697    1
## a[63]      -0.04        0 0.09 -0.21 -0.10 -0.04  0.02  0.14 28133    1
## a[64]       0.13        0 0.06  0.01  0.09  0.13  0.17  0.26 29046    1
## a[65]      -0.08        0 0.09 -0.25 -0.14 -0.08 -0.02  0.09 28298    1
## a[66]       0.00        0 0.09 -0.18 -0.06  0.00  0.06  0.18 27004    1
## a[67]       0.02        0 0.08 -0.13 -0.03  0.02  0.08  0.18 30800    1
## a[68]       0.02        0 0.10 -0.17 -0.05  0.02  0.09  0.21 30379    1
## a[69]      -0.18        0 0.09 -0.36 -0.24 -0.18 -0.12 -0.01 30663    1
## a[70]      -0.22        0 0.09 -0.39 -0.28 -0.23 -0.17 -0.05 28933    1
## a[71]      -0.11        0 0.09 -0.27 -0.16 -0.11 -0.05  0.06 29573    1
## a[72]      -0.04        0 0.09 -0.22 -0.11 -0.04  0.02  0.13 27205    1
## a[73]      -0.16        0 0.09 -0.34 -0.23 -0.17 -0.10  0.01 29546    1
## a[74]       0.04        0 0.07 -0.10 -0.01  0.04  0.08  0.17 26635    1
## a[75]      -0.04        0 0.07 -0.17 -0.08 -0.04  0.01  0.09 25050    1
## a[76]       0.00        0 0.08 -0.16 -0.06  0.00  0.05  0.15 28123    1
## a[77]      -0.08        0 0.10 -0.27 -0.14 -0.08 -0.01  0.12 28952    1
## a[78]       0.00        0 0.10 -0.19 -0.06  0.00  0.06  0.18 28725    1
## a[79]       0.02        0 0.07 -0.13 -0.03  0.02  0.06  0.16 26536    1
## a[80]      -0.01        0 0.08 -0.17 -0.06 -0.01  0.05  0.16 31768    1
## a[81]       0.02        0 0.08 -0.14 -0.03  0.02  0.08  0.19 27568    1
## a[82]      -0.11        0 0.08 -0.26 -0.16 -0.11 -0.05  0.05 29433    1
## a[83]      -0.14        0 0.09 -0.32 -0.20 -0.14 -0.08  0.04 29654    1
## a[84]      -0.08        0 0.09 -0.27 -0.15 -0.08 -0.02  0.10 28884    1
## a[85]       0.00        0 0.08 -0.15 -0.05  0.00  0.06  0.16 24687    1
## a[86]       0.03        0 0.09 -0.14 -0.03  0.03  0.09  0.21 26953    1
## a[87]      -0.12        0 0.09 -0.30 -0.18 -0.12 -0.06  0.06 30189    1
## a[88]      -0.04        0 0.09 -0.21 -0.10 -0.04  0.02  0.13 25980    1
## a[89]       0.02        0 0.09 -0.16 -0.05  0.01  0.08  0.20 28680    1
## a[90]      -0.02        0 0.09 -0.19 -0.08 -0.02  0.04  0.15 28765    1
## a[91]      -0.16        0 0.09 -0.34 -0.22 -0.16 -0.10  0.01 27425    1
## a[92]      -0.04        0 0.09 -0.22 -0.10 -0.04  0.02  0.14 28530    1
## a[93]      -0.08        0 0.10 -0.27 -0.15 -0.08 -0.01  0.11 34289    1
## a[94]      -0.02        0 0.10 -0.21 -0.09 -0.02  0.04  0.16 28837    1
## a[95]       0.07        0 0.10 -0.11  0.01  0.07  0.14  0.26 30354    1
## a[96]       0.05        0 0.09 -0.12 -0.01  0.05  0.11  0.22 30395    1
## a[97]      -0.04        0 0.09 -0.21 -0.10 -0.04  0.02  0.14 28524    1
## a[98]      -0.02        0 0.09 -0.20 -0.08 -0.02  0.03  0.14 27817    1
## a[99]       0.00        0 0.09 -0.18 -0.06  0.00  0.05  0.16 33106    1
## a[100]      0.03        0 0.09 -0.13 -0.03  0.03  0.09  0.20 30533    1
## a[101]     -0.01        0 0.10 -0.20 -0.08 -0.01  0.06  0.18 29164    1
## a[102]      0.03        0 0.09 -0.15 -0.03  0.03  0.09  0.20 30340    1
## a[103]     -0.07        0 0.09 -0.24 -0.13 -0.07  0.00  0.12 29207    1
## a[104]     -0.08        0 0.10 -0.26 -0.14 -0.08 -0.01  0.11 32135    1
## a[105]     -0.04        0 0.09 -0.22 -0.10 -0.04  0.02  0.13 31223    1
## a[106]      0.04        0 0.09 -0.14 -0.02  0.04  0.10  0.22 29222    1
## a[107]      0.01        0 0.10 -0.18 -0.06  0.01  0.07  0.20 29355    1
## a[108]      0.02        0 0.10 -0.17 -0.05  0.02  0.09  0.21 29132    1
## a[109]     -0.05        0 0.10 -0.25 -0.12 -0.05  0.01  0.14 27654    1
## a[110]      0.00        0 0.10 -0.18 -0.06  0.00  0.07  0.20 27321    1
## a[111]     -0.02        0 0.10 -0.21 -0.09 -0.02  0.05  0.18 30382    1
## a[112]     -0.04        0 0.10 -0.23 -0.11 -0.04  0.03  0.15 28855    1
## a[113]     -0.01        0 0.06 -0.14 -0.05 -0.01  0.03  0.12 24471    1
## a[114]      0.03        0 0.07 -0.10 -0.01  0.03  0.08  0.16 26585    1
## a[115]      0.13        0 0.04  0.04  0.10  0.13  0.16  0.22 23480    1
## a[116]      0.11        0 0.05  0.00  0.07  0.11  0.14  0.21 21849    1
## a[117]     -0.12        0 0.07 -0.25 -0.17 -0.12 -0.08  0.00 26456    1
## a[118]      0.17        0 0.04  0.08  0.14  0.17  0.20  0.25 24106    1
## a[119]     -0.09        0 0.09 -0.26 -0.15 -0.09 -0.04  0.08 30020    1
## a[120]     -0.06        0 0.09 -0.23 -0.12 -0.06  0.00  0.12 33859    1
## a[121]     -0.11        0 0.07 -0.25 -0.16 -0.11 -0.06  0.02 28118    1
## a[122]     -0.12        0 0.10 -0.31 -0.18 -0.12 -0.06  0.07 28257    1
## a[123]      0.06        0 0.04 -0.02  0.03  0.06  0.08  0.14 22378    1
## a[124]     -0.02        0 0.05 -0.12 -0.05 -0.02  0.02  0.09 22965    1
## a[125]    -0.13       0 0.08 -0.28 -0.18 -0.13 -0.07  0.03 29323    1
## a[126]     0.10       0 0.06 -0.03  0.06  0.10  0.14  0.23 25356    1
## a[127]     0.14       0 0.04  0.06  0.12  0.14  0.17  0.23 21165    1
## a[128]    -0.07       0 0.06 -0.19 -0.11 -0.07 -0.02  0.06 24523    1
## a[129]     0.01       0 0.07 -0.13 -0.04  0.01  0.06  0.16 27722    1
## a[130]    -0.05       0 0.06 -0.18 -0.09 -0.05 -0.01  0.08 25808    1
## a[131]     0.02       0 0.04 -0.06 -0.01  0.02  0.05  0.11 20122    1
## a[132]     0.06       0 0.07 -0.08  0.01  0.06  0.11  0.21 28981    1
## a[133]     0.03       0 0.07 -0.12 -0.02  0.03  0.08  0.17 26057    1
## a[134]    -0.07       0 0.08 -0.23 -0.12 -0.07 -0.01  0.09 31506    1
## a[135]    -0.05       0 0.05 -0.15 -0.09 -0.05 -0.02  0.05 24581    1
## a[136]     0.02       0 0.05 -0.09 -0.02  0.02  0.06  0.13 25402    1
## a[137]    -0.04       0 0.05 -0.14 -0.08 -0.04  0.00  0.06 23776    1
## a[138]     0.02       0 0.10 -0.17 -0.04  0.02  0.09  0.21 28675    1
## a[139]     0.03       0 0.08 -0.13 -0.02  0.03  0.09  0.20 28394    1
## a[140]     0.25       0 0.05  0.16  0.22  0.25  0.28  0.34 21195    1
## a[141]     0.13       0 0.05  0.03  0.10  0.13  0.16  0.23 22329    1
## a[142]    -0.10       0 0.07 -0.24 -0.14 -0.10 -0.05  0.05 28968    1
## a[143]    -0.01       0 0.09 -0.17 -0.06 -0.01  0.05  0.17 28338    1
## a[144]     0.04       0 0.10 -0.15 -0.03  0.04  0.10  0.23 32062    1
## a[145]    -0.05       0 0.07 -0.20 -0.10 -0.05  0.00  0.10 27923    1
## a[146]     0.05       0 0.07 -0.09  0.00  0.05  0.10  0.19 26554    1
## a[147]    -0.03       0 0.09 -0.20 -0.09 -0.03  0.03  0.14 29587    1
## a[148]    -0.06       0 0.07 -0.20 -0.11 -0.06 -0.02  0.07 23792    1
## a[149]     0.05       0 0.06 -0.07  0.00  0.05  0.09  0.16 25272    1
## a[150]    -0.10       0 0.06 -0.21 -0.14 -0.10 -0.07  0.01 22853    1
## a[151]    -0.03       0 0.07 -0.17 -0.08 -0.03  0.02  0.11 21223    1
## a[152]     0.04       0 0.06 -0.07  0.00  0.04  0.08  0.16 26503    1
## a[153]     0.03       0 0.08 -0.13 -0.03  0.03  0.08  0.18 27891    1
## a[154]    -0.04       0 0.08 -0.19 -0.09 -0.04  0.02  0.12 26833    1
## a[155]     0.22       0 0.06  0.11  0.18  0.22  0.26  0.34 27737    1
## a[156]    -0.11       0 0.07 -0.26 -0.16 -0.11 -0.06  0.04 25743    1
## a[157]     0.14       0 0.06  0.01  0.10  0.14  0.18  0.27 25814    1
## a[158]     0.19       0 0.07  0.06  0.14  0.19  0.23  0.31 27519    1
## a[159]     0.05       0 0.07 -0.09  0.00  0.05  0.09  0.19 26353    1
## a[160]    -0.02       0 0.10 -0.21 -0.08 -0.02  0.05  0.17 30559    1
## a[161]    -0.01       0 0.10 -0.20 -0.07 -0.01  0.05  0.18 27548    1
## a[162]    -0.11       0 0.08 -0.27 -0.16 -0.11 -0.05  0.05 27099    1
## a[163]     0.02       0 0.08 -0.14 -0.04  0.02  0.07  0.18 29179    1
## a[164]     0.02       0 0.08 -0.14 -0.04  0.02  0.07  0.18 27755    1
##
## Samples were drawn using NUTS(diag_e) at Mon Dec 06 22:19:18 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).
```

As we can see $\widehat{R}$ are all < 1.05 indicating the chains have converged. Considering the length of the dataset $n = 8552$, also ESS (n_eff) look good for every estimate. Divergent transitions and tree depths for each model can be seen here:

```
#separate
summary(do.call(rbind, get_sampler_params(fit_separate, inc_warmup = FALSE)))
```

```
##  accept_stat__      stepsize__      treedepth__  n_leapfrog__  divergent__
##  Min.   :0.2724   Min.   :0.4834   Min.   :3    Min.   :7     Min.   :0
##  1st Qu.:0.7909   1st Qu.:0.4876   1st Qu.:3    1st Qu.:7     1st Qu.:0
##  Median :0.8978   Median :0.4975   Median :3    Median :7     Median :0
##  Mean   :0.8660   Mean   :0.4968   Mean   :3    Mean   :7     Mean   :0
##  3rd Qu.:0.9698   3rd Qu.:0.5068   3rd Qu.:3    3rd Qu.:7     3rd Qu.:0
##  Max.   :1.0000   Max.   :0.5090   Max.   :3    Max.   :7     Max.   :0
##     energy__
##  Min.   :242.5
##  1st Qu.:259.7
##  Median :264.5
##  Mean   :264.8
##  3rd Qu.:269.5
##  Max.   :294.5
```

```
#hierarchical
summary(do.call(rbind, get_sampler_params(fit_hierarchical, inc_warmup = FALSE)))
```

```
##  accept_stat__        stepsize__      treedepth__     n_leapfrog__
##  Min.   :0.0000218   Min.   :0.2919   Min.   :3.000   Min.   : 7.00
##  1st Qu.:0.8258985   1st Qu.:0.3211   1st Qu.:4.000   1st Qu.:15.00
##  Median :0.9231538   Median :0.3319   Median :4.000   Median :15.00
##  Mean   :0.8748737   Mean   :0.3457   Mean   :3.792   Mean   :13.81
##  3rd Qu.:0.9752603   3rd Qu.:0.3565   3rd Qu.:4.000   3rd Qu.:15.00
##  Max.   :1.0000000   Max.   :0.4271   Max.   :4.000   Max.   :31.00
##   divergent__    energy__
##  Min.   :0    Min.   :118.7
##  1st Qu.:0    1st Qu.:151.7
##  Median :0    Median :157.2
##  Mean   :0    Mean   :157.5
##  3rd Qu.:0    3rd Qu.:163.2
##  Max.   :0    Max.   :192.5
```

```
#pooled
summary(do.call(rbind, get_sampler_params(fit_pooled, inc_warmup = FALSE)))
```

```
##  accept_stat__      stepsize__      treedepth__     n_leapfrog__   divergent__
##  Min.   :0.2705   Min.   :0.7651   Min.   :1.000   Min.   :1.00   Min.   :0
##  1st Qu.:0.8741   1st Qu.:0.7978   1st Qu.:2.000   1st Qu.:3.00   1st Qu.:0
##  Median :0.9543   Median :0.8136   Median :2.000   Median :3.00   Median :0
##  Mean   :0.9141   Mean   :0.8241   Mean   :1.864   Mean   :3.55   Mean   :0
##  3rd Qu.:0.9951   3rd Qu.:0.8399   3rd Qu.:2.000   3rd Qu.:3.00   3rd Qu.:0
##  Max.   :1.0000   Max.   :0.9040   Max.   :3.000   Max.   :7.00   Max.   :0
##     energy__
##  Min.   :328.0
##  1st Qu.:328.9
##  Median :329.6
##  Mean   :329.9
##  3rd Qu.:330.6
##  Max.   :338.3
```

```
#separate with additional parameter
summary(do.call(rbind, get_sampler_params(fit_separate_id, inc_warmup = FALSE)))
```

```
##  accept_stat__      stepsize__      treedepth__     n_leapfrog__    divergent__
##  Min.   :0.3570   Min.   :0.3212   Min.   :3.000   Min.   : 7.00   Min.   :0
##  1st Qu.:0.7699   1st Qu.:0.3261   1st Qu.:4.000   1st Qu.:15.00   1st Qu.:0
##  Median :0.8852   Median :0.3516   Median :4.000   Median :15.00   Median :0
##  Mean   :0.8536   Mean   :0.3542   Mean   :3.994   Mean   :14.98   Mean   :0
##  3rd Qu.:0.9640   3rd Qu.:0.3797   3rd Qu.:4.000   3rd Qu.:15.00   3rd Qu.:0
##  Max.   :1.0000   Max.   :0.3925   Max.   :5.000   Max.   :47.00   Max.   :0
##     energy__
##  Min.   :180.7
##  1st Qu.:219.5
##  Median :229.5
##  Mean   :229.9
##  3rd Qu.:239.6
##  Max.   :295.7
```

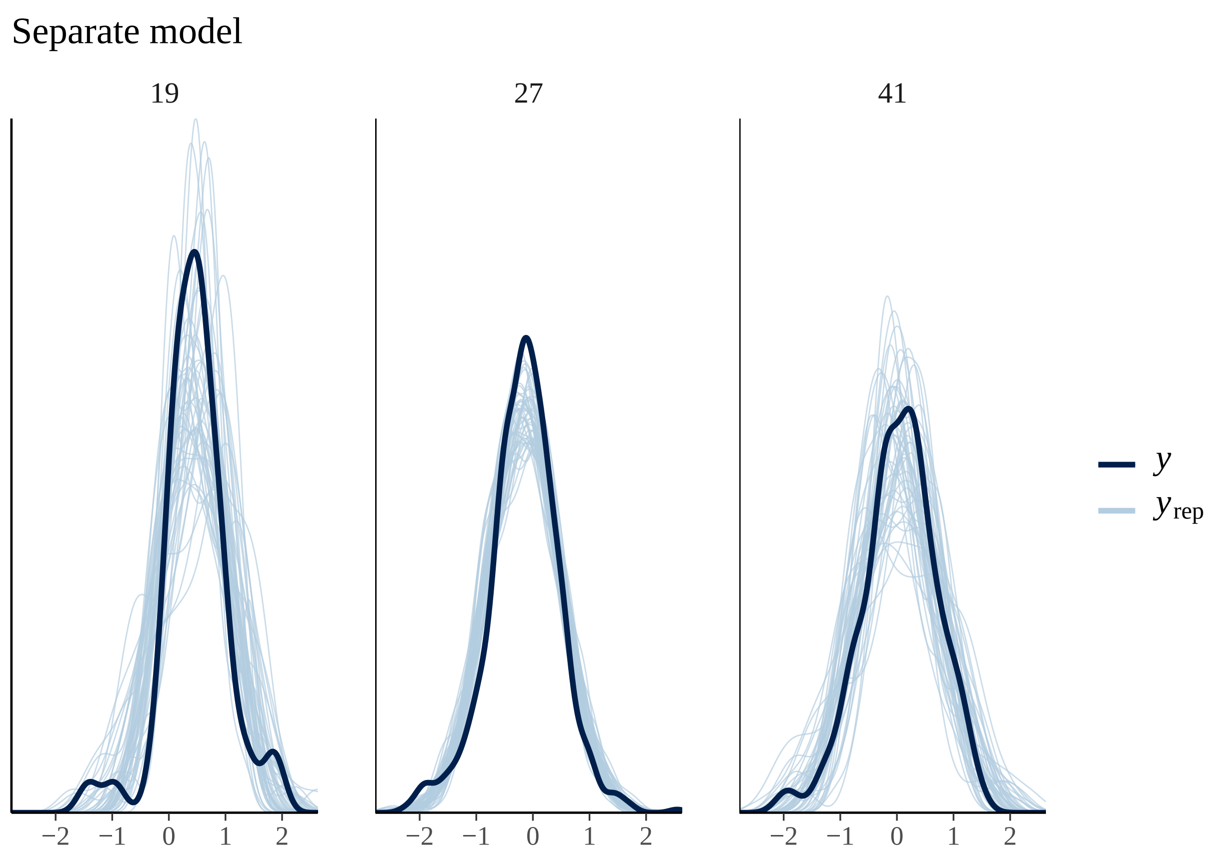
As we can see, there are no divergent transitions and tree depth is reasonable since it's maximum value is easily under 10 for each model.

Initially Stan warned about few divergent transitions and low ESS for separate model with additional parameter, so we increased the chain length to 6000 which fixed the problem. We also attempted to fit the hierarchical model with additional parameter but it had several problems with fitting including divergent transitions, big $\widehat{R}$ and low ESS which we were not able to fix by increasing chain length and increasing the adapt_delta parameter for Stan, indicating that it's not possible to fit the model at least with our computational resources and within a reasonable amount of time.

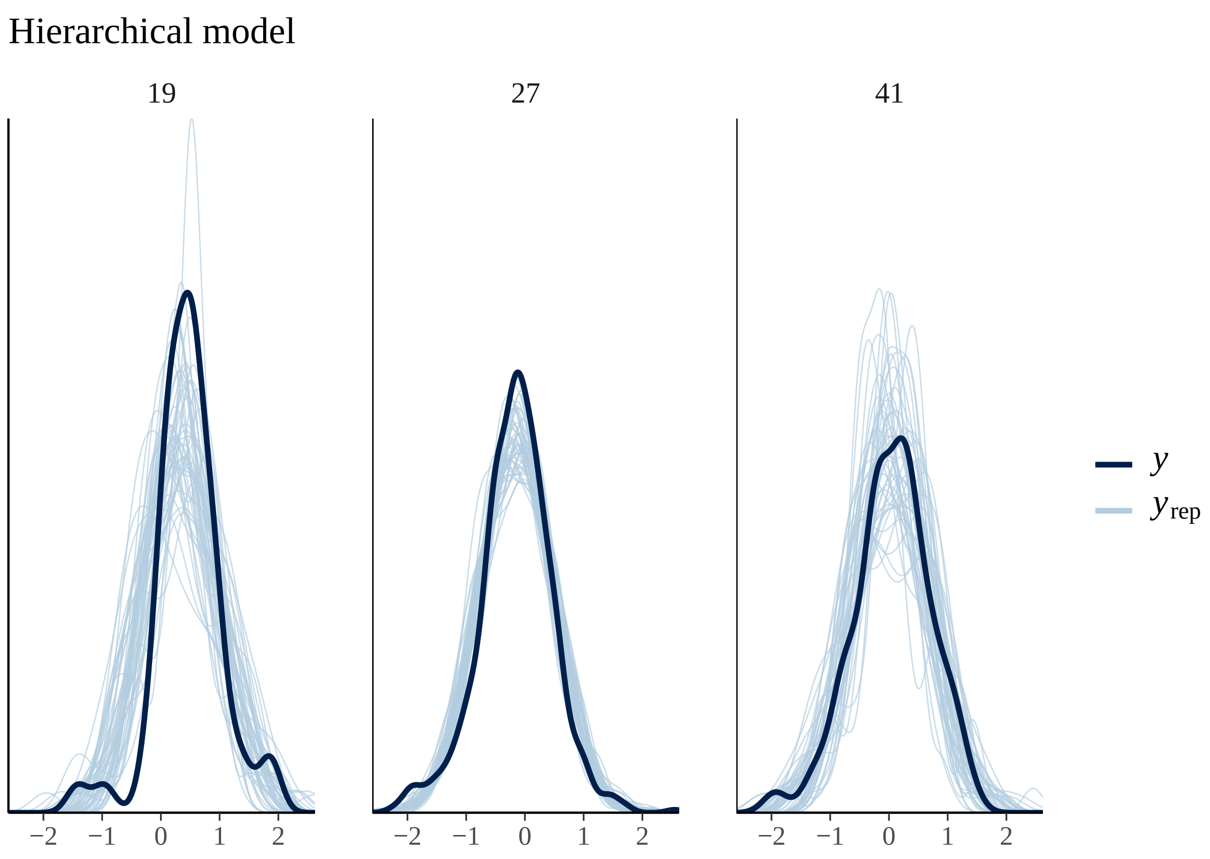
Posterior Predictive Checks

Here we plot density plots for replicated dataset y_{rep} together with the observed data y for three different ages, 19, 27 and 41. The age pools 19 and 41 are very sparse, containting only 48 and 57 samples, respectively, while age 27 contains 594 samples.

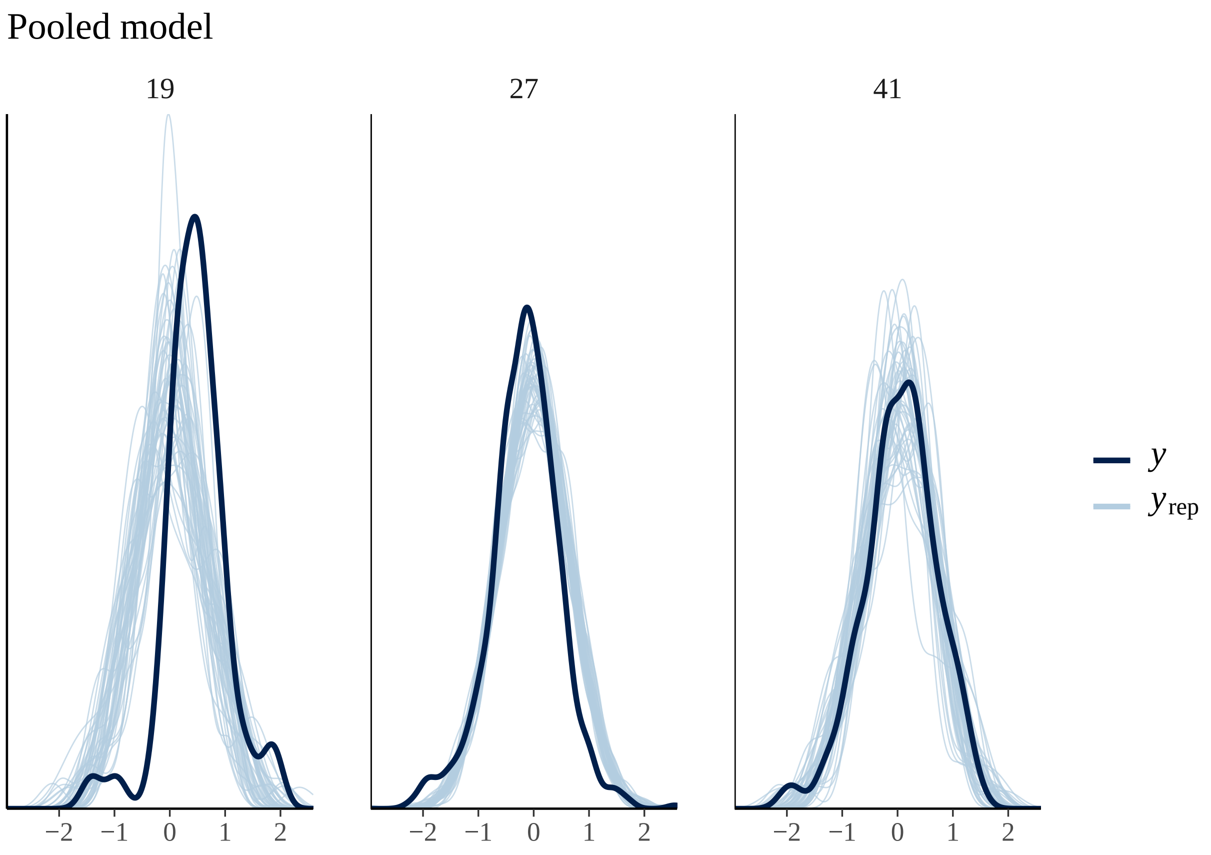
```
yrep_separate <- extract(fit_separate)$yrep
yrep_hierarchical <- extract(fit_hierarchical)$yrep
yrep_pooled <- extract(fit_pooled)$yrep
yrep_separate_id <- extract(fit_separate_id, pars='yrep')$yrep
yrep_separate_id_known <- extract(fit_separate_id, pars='yrep_id')$yrep_id

ppc_dens_overlay_grouped(y, yrep_separate[1:50,], y_rep_groups) + ggtitle('Separate model')
```



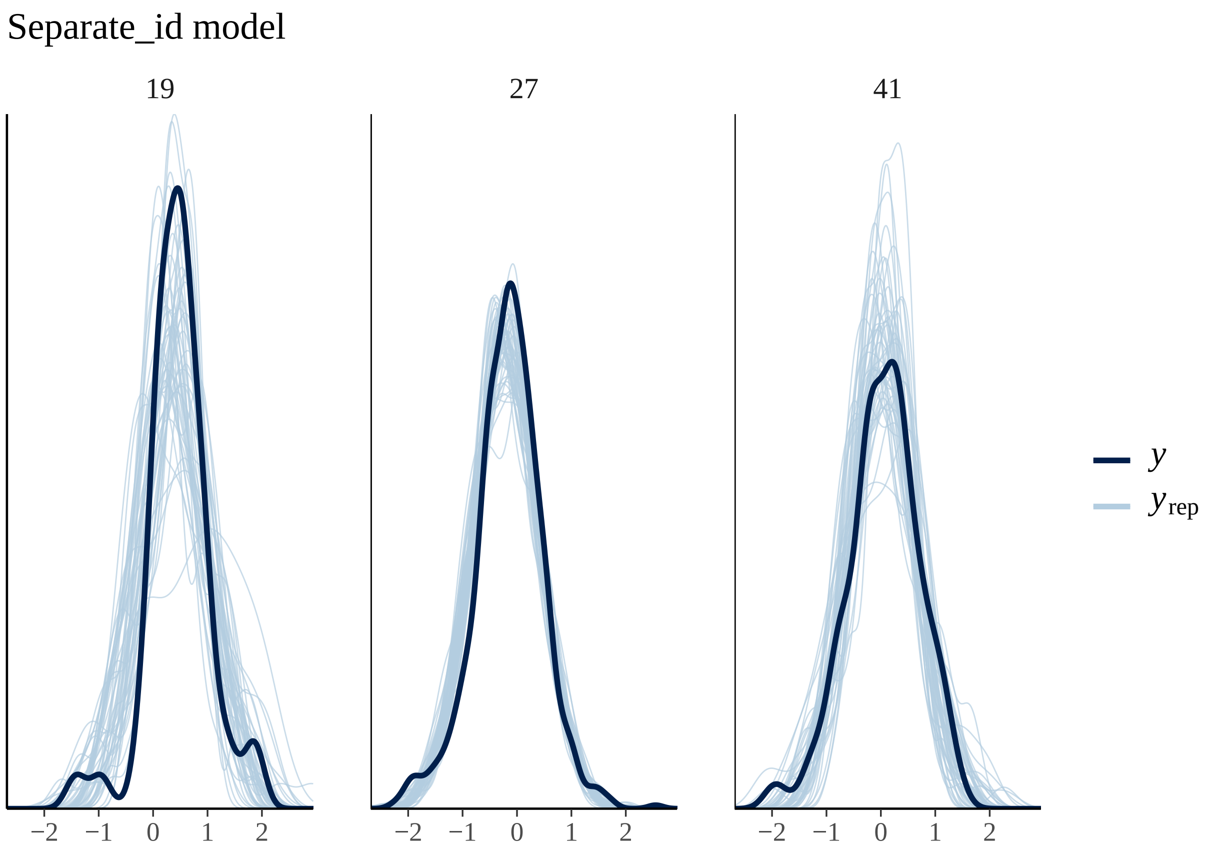
```
ppc_dens_overlay_grouped(y, yrep_hierarchical[1:50,], y_rep_groups) + ggtitle('Hierarchical model')
```



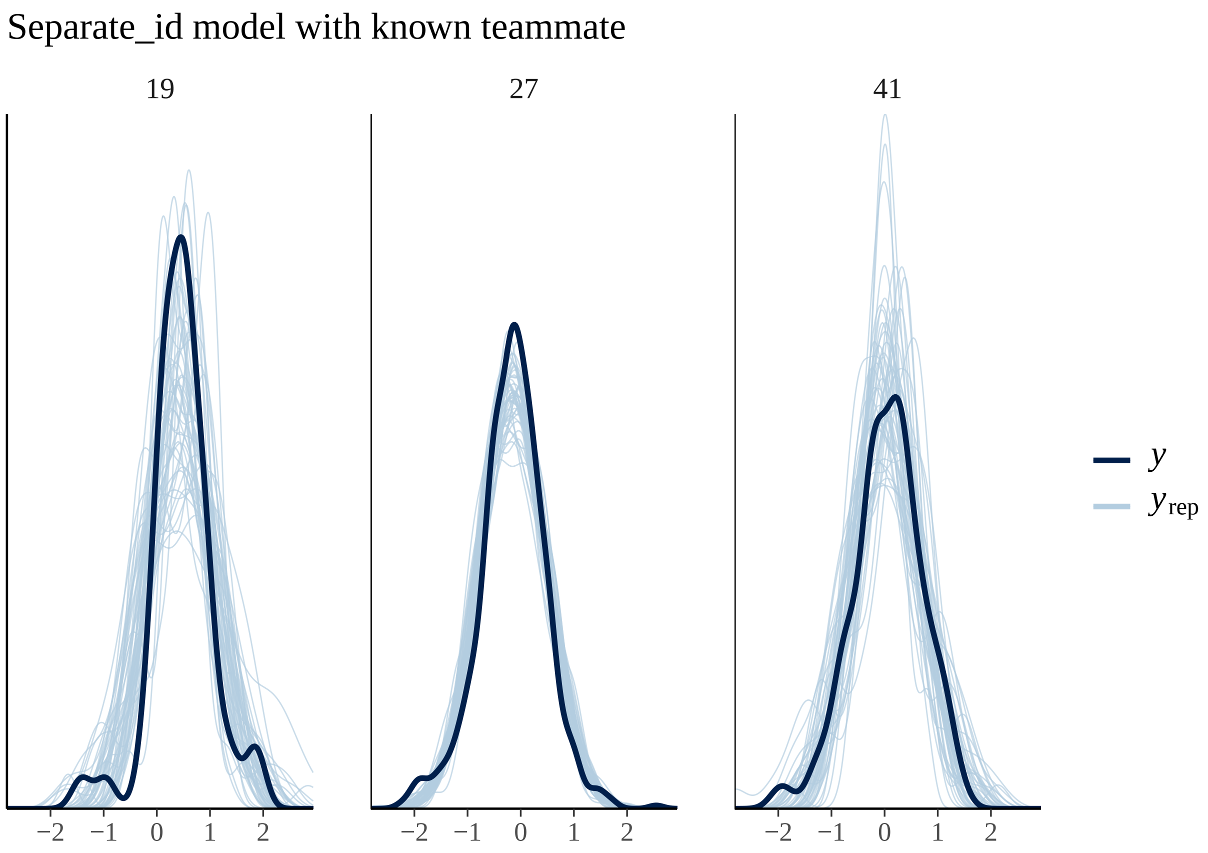
```
ppc_dens_overlay_grouped(y, yrep_pooled[1:50,], y_rep_groups) + ggtitle('Pooled model')
```



```
ppc_dens_overlay_grouped(y, yrep_separate_id[1:50,], y_rep_groups) + ggtitle('Separate_id model')
```



```
ppc_dens_overlay_grouped(y, yrep_separate_id_known[1:50,], y_rep_groups) +
  ggtitle('Separate_id model with known teammate')
```



Here, separate_id plot uses the fit of separate model with additional parameter but assumes teammate is not known, meaning it does not use the additional parameter when creating the replicated dataset. Ideally, the additional parameter should still have helped it to find the mean and variance parameters closer to the real values. Separate_id with known teammate then uses the parameter when generating the replicated meaning it assumes known teammate and therefore should have better performance than the others since it has additional information compared to others.

Based on these plots, hierarchical and separate model seem to have fit reasonably well to the data, while pooled seems to have pretty clear problems especially in age 19. Separate_id with known teammate also seems to have fit slightly better than all the others which have no teammate information.

We examined plots for all the other ages too but didn't include everything to this report, since most were similar to the ones showed here: separate_id with known teammate is best, hierarchical and separate fit reasonably well, pooled has problems with some ages. This is expected since we expect age to have some effect to average results but pooled assumes same distribution for every age. Though, even if it didn't have any effect, it's still possible that the different age pools have differing distributions just by chance, especially as some of them are very sparse, in which case the hierarchical and separate model would fit better anyway. This would be a case of overfit if the reality is that age does not have any effect.

Model Comparison

There was already some comparisons between the graphical predictive checks in the previous sections, but here we compare the loo-cv values for each model.

```
loo_mat_separate <- loo(extract_log_lik(fit_separate))
loo_mat_hierarchical <- loo(extract_log_lik(fit_hierarchical))
loo_mat_pooled <- loo(extract_log_lik(fit_pooled))
loo_mat_separate_id <- loo(extract_log_lik(fit_separate_id))
loo_mat_separate_id_known <- loo(extract(fit_separate_id, pars='log_lik_id')$log_lik_id)

loo_compare(loo_mat_hierarchical, loo_mat_pooled, loo_mat_separate, loo_mat_separate_id,
            loo_mat_separate_id_known)
```

```
##        elpd_diff se_diff
## model5    0.0       0.0 
## model1 -196.1      17.3 
## model3 -200.1      16.7 
## model2 -274.1      23.5 
## model4 -286.5      20.1 
```

For clarification, the outputs correspond to following models:

model1: hierarchical model model2: pooled model *model3: separate model* model4: separate model with additional parameter *model5: separate model with additional parameter with known teammate

Here the separate model with additional parameter and teammate assumed known is quite clearly the best. However, as mentioned before, it is expected as it has additional information compared to other models and hence is not really straightly comparable to others. As we can see, the same model with teammate assumed known has the worse elpd value, meaning it was not succesfull in learning the mean and variance parameters better with the help of the additional parameter. In conclusion, if we assume unknown teammate, hierarchical model seems to be the best, with separate model very close second. Pooled model is again worse, as expected.

Predictive Performance Assessment

Predictive performance is not applicable in this analysis since we are not trying to make a model to predict anything. Instead, we are simply trying to fit models to the data that would reveal properties of the data. Properties in this case meaning whether different age groups seem to have performed differently and if so, which have performed the best and with what probability.

Prior Sensitivity Analysis

We fitted the models with priors for μ and σ both having variance of 10 instead of 1. Additionally, the parameter α has variance of 1 instead of 0.1. We compare the loo-cv values of these fits to the original fits.

```
fit_separate_dprior <- stan(file = "../stan/separate_model_dprior.stan", data = stan_data)
fit_hierarchical_dprior <- stan(file = "../stan/hierarchical_model_dprior.stan", data = stan_data)
fit_pooled_dprior <- stan(file = "../stan/pooled_model_dprior.stan", data = stan_data)
fit_separate_id_dprior <- stan(file = "../stan/separate_model_ids_dprior.stan", data = stan_data_id)

loo_mat_separate_dprior <- loo(extract_log_lik(fit_separate_dprior))
loo_mat_hierarchical_dprior <- loo(extract_log_lik(fit_hierarchical_dprior))
loo_mat_pooled_dprior <- loo(extract_log_lik(fit_pooled_dprior))
loo_mat_separate_id_dprior <- loo(extract_log_lik(fit_separate_id_dprior))
```

```
loo_compare(loo_mat_separate, loo_mat_separate_dprior)
```

```
##        elpd_diff se_diff
## model1  0.0       0.0
## model2 -0.4       0.3
```

```
loo_compare(loo_mat_hierarchical, loo_mat_hierarchical_dprior)
```

```
##        elpd_diff se_diff
## model1  0.0       0.0
## model2 -0.2       0.2
```

```
loo_compare(loo_mat_pooled, loo_mat_pooled_dprior)
```

```
##        elpd_diff se_diff
## model1  0.0       0.0
## model2 -0.1       0.0
```

```
loo_compare(loo_mat_separate_id, loo_mat_separate_id_dprior)
```

```
##        elpd_diff se_diff
## model1    0.0       0.0
## model2 -580.0      21.9
```

Based on this comparisons, the priors don't have significant effect on the posteriors apart from the model with additional parameter. This is expected in the sense that we have relatively many datapoints (8552). The model with additional parameter, is quite significantly worse. It was already examined before that adding the additional parameter worsened the model's performance in the general case where teammate is not known. Apparently when given the additional parameter more room to fit the data more by making the prior less informative, it makes the model even worse in the general case.

Which age is the best?

We use the hierarchical model to assess which ages seem the best since it was the best model assuming teammate is not known based on loo-cv values. We calculate the probability that the μ parameter of an age is the smallest (the more negative the difference the better the performance is) of all ages for each age.

```
mu_hierarchical <- extract(fit_hierarchical, pars='mu')$mu

num_ages = length(unique(ages))
probs_separate_id = vector(mode= 'numeric', length = num_ages)
probs_hierarchical = vector(mode= 'numeric', length = num_ages)

for (i in 1:num_ages) {
  mu1 = mu_hierarchical[,i]
  prob = 1
  for (j in 1:num_ages) {
    if (j != i) {
      mu2 = mu_hierarchical[,j]
      prob = prob * (sum(mu2>mu1)/length(mu1))
    }
  }
  probs_hierarchical[i] = prob
}

probs_df <- data.frame(probs_hierarchical/sum(probs_hierarchical), sort(unique(ages)))
colnames(probs_df) <-c('prob', 'age')
probs_df
```

```
##            prob age
## 1  1.869537e-12  18
## 2  0.000000e+00  19
## 3  1.688508e-09  20
## 4  5.152883e-12  21
## 5  0.000000e+00  22
## 6  0.000000e+00  23
## 7  8.234137e-03  24
## 8  9.495011e-05  25
## 9  1.835825e-03  26
## 10 9.898296e-01  27
## 11 5.468102e-06  28
## 12 0.000000e+00  29
## 13 0.000000e+00  30
## 14 0.000000e+00  31
## 15 0.000000e+00  32
## 16 0.000000e+00  33
## 17 0.000000e+00  34
## 18 0.000000e+00  35
## 19 3.772130e-13  36
## 20 0.000000e+00  37
## 21 0.000000e+00  38
## 22 1.962711e-23  39
## 23 6.120846e-15  40
## 24 1.304255e-13  41
## 25 0.000000e+00  42
## 26 5.912783e-27  43
```

In the above dataframe we can see the probabilites for each age being the best. Based on it, age 27 is the best.

Discussion

The dataset still has some dependency to the manufacturer of the car, which could be further investigated to improve the models. Furthermore the dataset was unequally distributed by the age of the river, thus both ends of the age distributions might not fully reflect the age of the driver effect on performance but rather about few individual drivers performance.

Conclusions and Self-Reflections

The report shows that the hierarchical separate and separate with driver id all capture the all the distributions well. The pooled model is having some issues to the ends of the models, which could be because the younger and older drivers have bigger difference compared to the other aged drivers.

We further deepened our understanding with the bayesian analysis process and how to use these skill to a real data set.

References

"Ergast Developer API." n.d. Accessed November 30, 2021. http://ergast.com/mrd/.

Jain, Adit. n.d. "Predicting the Results for the World's Most Expensive Sport." Accessed December 1, 2021. https://medium.com/@ajleo899/predicting-the-results-for-the-worlds-most-expensive-sport-102e7bc97b25.

Nigro, Veronica. n.d. "Formula 1 Race Predictor." Accessed November 30, 2021. https://towardsdatascience.com/formula-1-race-predictor-5d4bfae887da.

van Kesteren, Erik-Jan. n.d. "Bayesian Analysis of Formula One Race Results." Accessed December 1, 2021. https://github.com/vankesteren/f1model.

Vehtari, Aki, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner. 2021. "Rank-Normalization, Folding, and Localization: An Improved r^ for Assessing Convergence of MCMC (with Discussion)." *Bayesian Analysis* 16 (2). https://doi.org/10.1214/20-ba1221.