Assignment_1_methods_3

Tilde Sloth

10/12/2022

Assignment 1 - Language development in autistic and neurotypical children

Quick recap

Autism Spectrum Disorder is often related to language impairment. However, this phenomenon has rarely been empirically traced in detail: i) relying on actual naturalistic language production, ii) over extended periods of time.

We therefore videotaped circa 30 kids with ASD and circa 30 comparison kids (matched by linguistic performance at visit 1) for ca. 30 minutes of naturalistic interactions with a parent. We repeated the data collection 6 times per kid, with 4 months between each visit. We transcribed the data and counted: i) the amount of words that each kid uses in each video. Same for the parent. ii) the amount of unique words that each kid uses in each video. Same for the parent. iii) the amount of morphemes per utterance (Mean Length of Utterance) displayed by each child in each video. Same for the parent.

This data is in the file you prepared in the previous class, but you can also find it here:https://www.dropbox.com/s/d6eerv6cl6eksf3/data_clean.csv?dl=0 (https://www.dropbox.com/s/d6eerv6cl6eksf3/data_clean.csv?dl=0)

```
pacman::p_load(tidyverse, dplyr, brms, bayesplot, rstanarm, msm)
d <- read_csv("CleanedChild_Data.csv")
```

The structure of the assignment

We will be spending a few weeks with this assignment. In particular, we will:

Part 1) simulate data in order to better understand the model we need to build, and to better understand how much data we would have to collect to run a meaningful study (precision analysis)

Part 2) analyze our empirical data and interpret the inferential results

Part 3) use your model to predict the linguistic trajectory of new children and assess the performance of the model based on that.

As you work through these parts, you will have to produce a written document (separated from the code) answering the following questions:

Q1 - Briefly describe your simulation process, its goals, and what you have learned from the simulation. Add at least a plot showcasing the results of the simulation. Make a special note on sample size considerations: how much data do you think you will need? what else could you do to increase the precision of your estimates?

Q2 - Briefly describe the empirical data and how they compare to what you learned from the simulation (what can you learn from them?). Briefly describe your model(s) and model quality. Report the findings: how does development differ between autistic and neurotypical children (N.B. remember to report both population and individual level findings)? which additional factors should be included in the model? Add at least one plot showcasing your findings.

Q3 - Given the model(s) from Q2, how well do they predict the data? Discuss both in terms of absolute error in training vs testing; and in terms of characterizing the new kids' language development as typical or in need of support.

Below you can find more detailed instructions for each part of the assignment.

Part 1 - Simulating data

Before we even think of analyzing the data, we should make sure we understand the problem, and we plan the analysis. To do so, we need to simulate data and analyze the simulated data (where we know the ground truth).

In particular, let's imagine we have n autistic and n neurotypical children. We are simulating their average utterance length (Mean Length of Utterance or MLU) in terms of words, starting at Visit 1 and all the way to Visit 6. In other words, we need to define a few parameters: - average MLU for ASD (population mean) at Visit 1 and average individual deviation from that (population standard deviation) - average MLU for TD (population mean) at Visit 1 and average individual deviation from that (population standard deviation) - average change in MLU by visit for ASD (population mean) and average individual deviation from that (population standard deviation) - average change in MLU by visit for TD (population mean) and average individual deviation from that (population standard deviation) - an error term. Errors could be due to measurement, sampling, all sorts of noise.

Note that this makes a few assumptions: population means are exact values; change by visit is linear (the same between visit 1 and 2 as between visit 5 and 6). This is fine for the exercise. In real life research, you might want to vary the parameter values much more, relax those assumptions and assess how these things impact your inference.

We go through the literature and we settle for some values for these parameters: - average MLU for ASD and TD: 1.5 (remember the populations are matched for linguistic ability at first visit) - average individual variability in initial MLU for ASD 0.5; for TD 0.3 (remember ASD tends to be more heterogeneous) - average change in MLU for ASD: 0.4; for TD 0.6 (ASD is supposed to develop less) - average individual variability in change for ASD 0.4; for TD 0.2 (remember ASD tends to be more heterogeneous) - error is identified as 0.2

This would mean that on average the difference between ASD and TD participants is 0 at visit 1, 0.2 at visit 2, 0.4 at visit 3, 0.6 at visit 4, 0.8 at visit 5 and 1 at visit 6.

With these values in mind, simulate data, plot the data (to check everything is alright); and set up an analysis pipeline. Remember the usual bayesian workflow: - define the formula - define the prior - prior predictive checks - fit the model - model quality checks: traceplots, divergences, rhat, effective samples - model quality checks: posterior predictive checks, prior-posterior update checks - model comparison

Once the pipeline is in place, loop through different sample sizes to assess how much data you would need to collect. N.B. for inspiration on how to set this up, check the tutorials by Kurz that are linked in the syllabus.

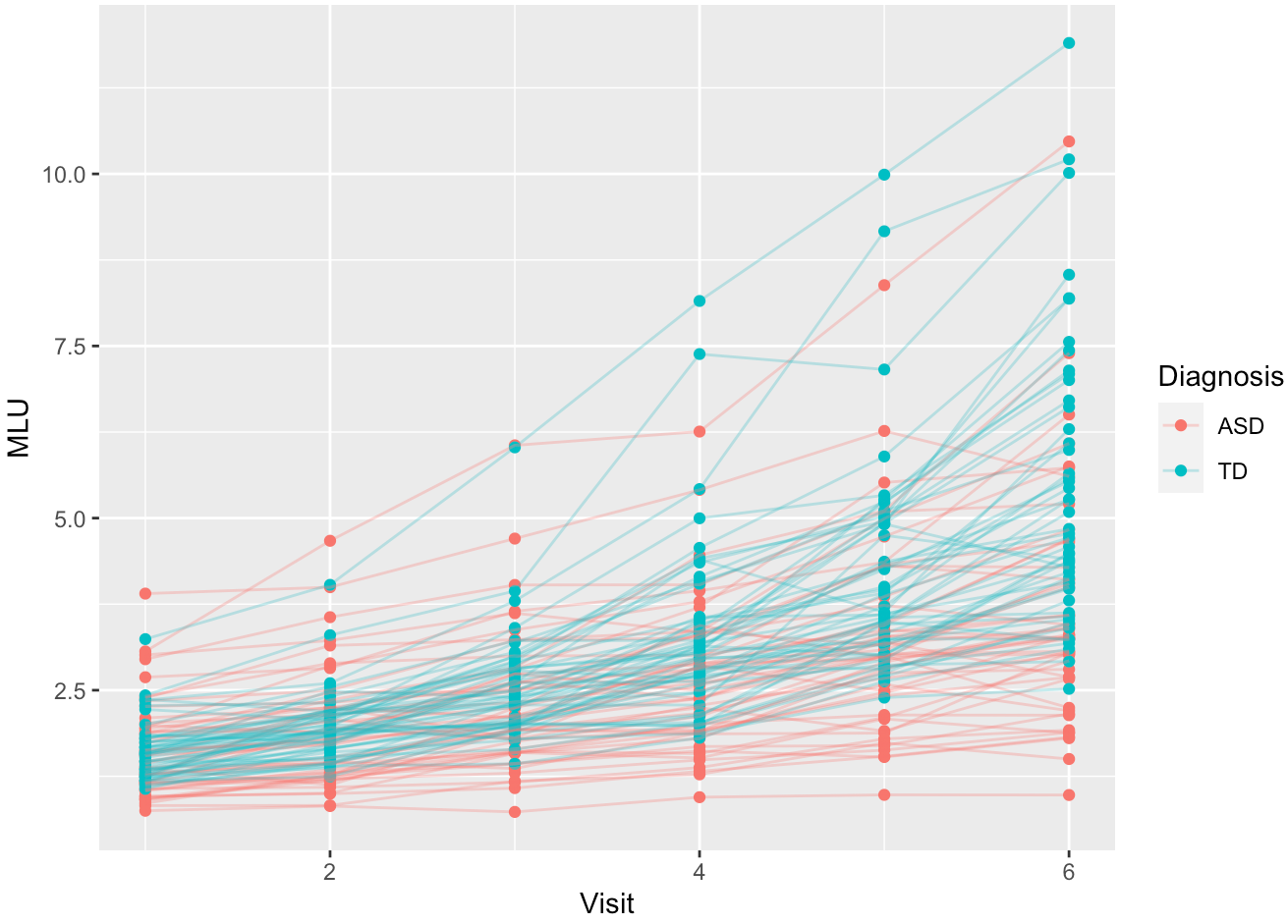
BONUS questions for Part 1: what if the difference between ASD and TD was 0? how big of a sample size would you need? What about different effect sizes, and different error terms?

Data simulation (NV & TS)

```
#Simulation function
simulation <- function(seed, n){
#Defining the different parameters for the model
mu_asd <- log(1.5)
mu_td <- log(1.5)
sigma_asd <- log(1.5)-log(1.5-0.5)
sigma_td <- log(1.5)-log(1.5-0.3)
#Now we define the values that are relative to the previous parameters
mu_slope_asd <- mu_asd*0.4
mu_slope_td <- mu_td*0.6
sigma_slope_asd <- mu_slope_asd*0.4
sigma_slope_td <- mu_slope_td*0.2
error <- 0.1

#Setting the seed
set.seed(seed)
#Converting it into a data frame
parameters <- tibble(Diagnosis = rep(c("ASD", "TD"), each = ceiling(n/2))) %>% mutate
(
Condition = ifelse(Diagnosis == "ASD", 0, 1),
Intercept = ifelse(Diagnosis == "ASD",
rnorm(n, mu_asd, sigma_asd),
rnorm(n, mu_td, sigma_td)),
Slope = ifelse(Diagnosis == "ASD",
rnorm(n, mu_slope_asd, sigma_slope_asd),
rnorm(n, mu_slope_td, sigma_slope_td)),
Kid = seq(n))

grid <- tibble(
  expand_grid(Kid = seq(n),
              Visit = seq(6))
)

df <- merge(parameters, grid)

for (i in seq(nrow(df))){
  df$MLU[i] <- exp(rnorm(1, df$Intercept[i] + df$Slope[i] * (df$Visit[i] - 1), erro
r))
}
return(df)
}

SimD <- simulation(123, 100)
```

Plot data (TS)

```
SimD %>%
  ggplot(aes(x = Visit, y = MLU, color = Diagnosis, group = Kid)) + geom_point() + ge
om_line(alpha = 0.3)
```



Analysis pipeline (TS & NV)

```
#define the formula

MLUformula <- bf(MLU ~ 0 + Diagnosis + Diagnosis:Visit + (1+Visit|Kid))

MLUformula_1 <- bf(MLU ~ 0 + Diagnosis + Diagnosis:Visit)
```

```
#define the prior
get_prior(MLUformula,
          data = SimD)
```

```
##                 prior class               coef group resp dpar nlpar lb ub
##                (flat)     b                                               
##                (flat)     b       DiagnosisASD                            
##                (flat)     b DiagnosisASD:Visit                            
##                (flat)     b        DiagnosisTD                            
##                (flat)     b  DiagnosisTD:Visit                            
##                lkj(1)   cor                                               
##                lkj(1)   cor                      Kid                      
##  student_t(3, 0, 2.5)    sd                                              0
##  student_t(3, 0, 2.5)    sd                      Kid                     0
##  student_t(3, 0, 2.5)    sd          Intercept   Kid                     0
##  student_t(3, 0, 2.5)    sd              Visit   Kid                     0
##  student_t(3, 0, 2.5) sigma                                              0
##        source
##       default
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##       default
##  (vectorized)
##       default
##  (vectorized)
##  (vectorized)
##  (vectorized)
##       default
```

```
get_prior(MLUformula_1,
          data = SimD)
```

```
##                 prior class               coef group resp dpar nlpar lb ub
##                (flat)     b                                               
##                (flat)     b       DiagnosisASD                            
##                (flat)     b DiagnosisASD:Visit                            
##                (flat)     b        DiagnosisTD                            
##                (flat)     b  DiagnosisTD:Visit                            
##  student_t(3, 0, 2.5) sigma                                              0
##        source
##       default
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##       default
```

```
MLU_prior <- c(
  prior(normal(0, 0.2), class = b),
  prior(normal(0.2, 0.2), class = b, coef = "DiagnosisASD"),
  prior(normal(0.2, 0.2), class = b, coef = "DiagnosisTD"),
  prior(normal(0, 0.3), class = sd, coef = Intercept, group = Kid),
  prior(normal(0, 0.05), class = sd, coef = Visit, group = Kid),
  prior(normal(0.1,0.1),class=sigma))

MLU_prior_1 <- c(
  prior(normal(0, 0.2), class = b),
  prior(normal(0.2, 0.2), class = b, coef = "DiagnosisASD"),
  prior(normal(0.2, 0.2), class = b, coef = "DiagnosisTD"),
  prior(normal(0.1,0.1), class = sigma))
```

We set the priors based of the information given in the description. We choose to make similar priors for the two groups ASD and TD, since we want the data to work hard to prove a difference. In that way the priors are quite uninformed but they still make sure that we don't expect extreme outliers. ___

```
#prior predictive checks
MLU_model_prior <-
  brm(
    MLUformula,
    data = SimD,
    family = lognormal,
    prior = MLU_prior,
    sample_prior = "only",
    iter = 5000,
    warmup = 1000,
    cores = 2,
    chains = 2,
    backend = "cmdstanr",
    threads = threading(2),
    control = list(
      adapt_delta = 0.999,
      max_treedepth = 20))
```

```
## Running MCMC with 2 parallel chains, with 2 thread(s) per chain...
##
## Chain 1 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 2 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 2 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 1 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 1 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 1 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 2 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 1 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 1 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 2 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 2 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 1 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 1 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 2 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 2 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 1 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1 Iteration: 1001 / 5000 [ 20%]  (Sampling)
## Chain 1 Iteration: 1100 / 5000 [ 22%]  (Sampling)
## Chain 2 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 2 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 2 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 1 Iteration: 1200 / 5000 [ 24%]  (Sampling)
## Chain 1 Iteration: 1300 / 5000 [ 26%]  (Sampling)
## Chain 2 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 2 Iteration: 1001 / 5000 [ 20%]  (Sampling)
## Chain 1 Iteration: 1400 / 5000 [ 28%]  (Sampling)
## Chain 2 Iteration: 1100 / 5000 [ 22%]  (Sampling)
## Chain 1 Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 1 Iteration: 1600 / 5000 [ 32%]  (Sampling)
## Chain 2 Iteration: 1200 / 5000 [ 24%]  (Sampling)
## Chain 1 Iteration: 1700 / 5000 [ 34%]  (Sampling)
## Chain 1 Iteration: 1800 / 5000 [ 36%]  (Sampling)
## Chain 2 Iteration: 1300 / 5000 [ 26%]  (Sampling)
## Chain 1 Iteration: 1900 / 5000 [ 38%]  (Sampling)
## Chain 1 Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 2 Iteration: 1400 / 5000 [ 28%]  (Sampling)
## Chain 1 Iteration: 2100 / 5000 [ 42%]  (Sampling)
## Chain 2 Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 1 Iteration: 2200 / 5000 [ 44%]  (Sampling)
## Chain 1 Iteration: 2300 / 5000 [ 46%]  (Sampling)
## Chain 2 Iteration: 1600 / 5000 [ 32%]  (Sampling)
## Chain 1 Iteration: 2400 / 5000 [ 48%]  (Sampling)
## Chain 1 Iteration: 2500 / 5000 [ 50%]  (Sampling)
## Chain 2 Iteration: 1700 / 5000 [ 34%]  (Sampling)
## Chain 1 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 1 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 2 Iteration: 1800 / 5000 [ 36%]  (Sampling)
```

```
## Chain 1 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 2 Iteration: 1900 / 5000 [ 38%]  (Sampling)
## Chain 1 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 1 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 2 Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 1 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 1 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 2 Iteration: 2100 / 5000 [ 42%]  (Sampling)
## Chain 1 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 1 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 2 Iteration: 2200 / 5000 [ 44%]  (Sampling)
## Chain 1 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 2 Iteration: 2300 / 5000 [ 46%]  (Sampling)
## Chain 1 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 1 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 2 Iteration: 2400 / 5000 [ 48%]  (Sampling)
## Chain 1 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 2 Iteration: 2500 / 5000 [ 50%]  (Sampling)
## Chain 1 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 1 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 1 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 1 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 2 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 1 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 1 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 2 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 1 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 2 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 1 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 1 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 2 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 1 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 1 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 2 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 1 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 2 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 1 finished in 2.9 seconds.
## Chain 2 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 2 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 2 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 2 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 2 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 2 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 2 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 2 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 2 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 2 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 2 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 2 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 2 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 2 Iteration: 4500 / 5000 [ 90%]  (Sampling)
```

```
## Chain 2 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 2 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 2 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 2 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 2 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 2 finished in 4.9 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 3.9 seconds.
## Total execution time: 5.0 seconds.
```

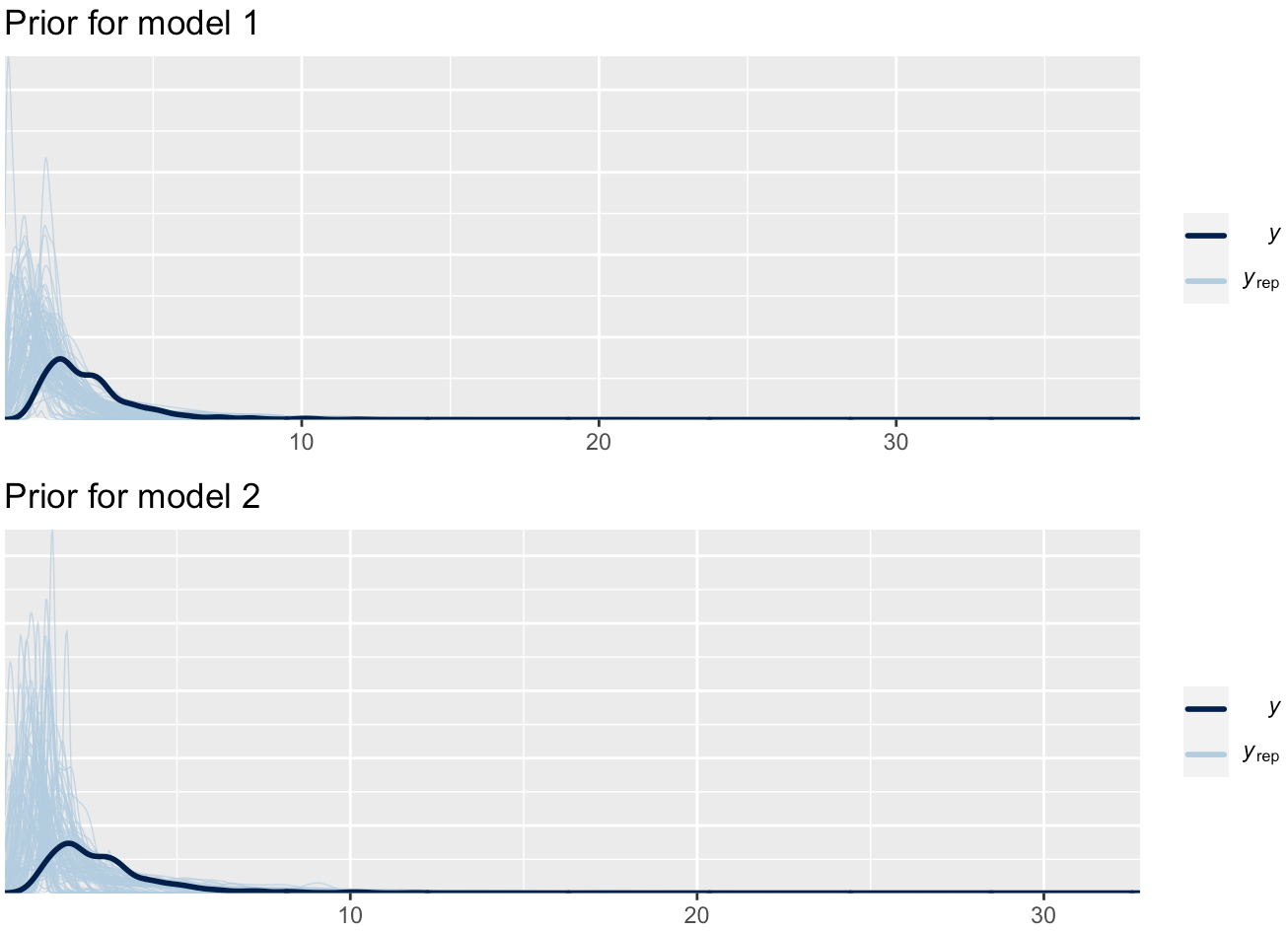
```
ppp1 <- pp_check(MLU_model_prior, ndraws=100) + labs(title="Prior for model 1")

MLU_model_prior_1 <-
  brm(
    MLUformula_1,
    data = SimD,
    save_pars = save_pars(all = TRUE),
    family = lognormal,
    prior = MLU_prior_1,
    sample_prior = "only",
    iter = 5000,
    warmup = 1000,
    cores = 2,
    chains = 2,
    backend = "cmdstanr",
    threads = threading(2),
    control = list(
      adapt_delta = 0.999,
      max_treedepth = 20))
```

```
## Running MCMC with 2 parallel chains, with 2 thread(s) per chain...
##
## Chain 1 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 1 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 1 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 1 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 1 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 1 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 1 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 1 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 1 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1 Iteration: 1001 / 5000 [ 20%]  (Sampling)
## Chain 1 Iteration: 1100 / 5000 [ 22%]  (Sampling)
## Chain 1 Iteration: 1200 / 5000 [ 24%]  (Sampling)
## Chain 1 Iteration: 1300 / 5000 [ 26%]  (Sampling)
## Chain 1 Iteration: 1400 / 5000 [ 28%]  (Sampling)
## Chain 1 Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 1 Iteration: 1600 / 5000 [ 32%]  (Sampling)
## Chain 1 Iteration: 1700 / 5000 [ 34%]  (Sampling)
## Chain 1 Iteration: 1800 / 5000 [ 36%]  (Sampling)
## Chain 1 Iteration: 1900 / 5000 [ 38%]  (Sampling)
## Chain 1 Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 1 Iteration: 2100 / 5000 [ 42%]  (Sampling)
## Chain 1 Iteration: 2200 / 5000 [ 44%]  (Sampling)
## Chain 1 Iteration: 2300 / 5000 [ 46%]  (Sampling)
## Chain 1 Iteration: 2400 / 5000 [ 48%]  (Sampling)
## Chain 1 Iteration: 2500 / 5000 [ 50%]  (Sampling)
## Chain 1 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 1 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 1 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 1 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 1 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 1 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 1 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 1 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 1 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 1 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 2 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 2 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 2 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 2 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 2 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 2 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 2 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 2 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 2 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 2 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 2 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 2 Iteration: 1001 / 5000 [ 20%]  (Sampling)
```

```
## Chain 2 Iteration: 1100 / 5000 [ 22%]  (Sampling)
## Chain 2 Iteration: 1200 / 5000 [ 24%]  (Sampling)
## Chain 2 Iteration: 1300 / 5000 [ 26%]  (Sampling)
## Chain 2 Iteration: 1400 / 5000 [ 28%]  (Sampling)
## Chain 2 Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 2 Iteration: 1600 / 5000 [ 32%]  (Sampling)
## Chain 2 Iteration: 1700 / 5000 [ 34%]  (Sampling)
## Chain 2 Iteration: 1800 / 5000 [ 36%]  (Sampling)
## Chain 2 Iteration: 1900 / 5000 [ 38%]  (Sampling)
## Chain 2 Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 2 Iteration: 2100 / 5000 [ 42%]  (Sampling)
## Chain 2 Iteration: 2200 / 5000 [ 44%]  (Sampling)
## Chain 2 Iteration: 2300 / 5000 [ 46%]  (Sampling)
## Chain 2 Iteration: 2400 / 5000 [ 48%]  (Sampling)
## Chain 2 Iteration: 2500 / 5000 [ 50%]  (Sampling)
## Chain 2 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 2 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 2 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 2 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 2 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 2 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 2 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 2 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 1 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 1 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 1 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 1 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 1 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 1 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 1 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 1 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 1 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 1 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 1 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 1 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 1 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 1 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 1 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 2 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 2 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 2 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 2 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 2 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 2 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 2 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 2 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 2 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 2 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 2 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 2 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 2 Iteration: 4600 / 5000 [ 92%]  (Sampling)
```

```
## Chain 2 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 2 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 2 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 2 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 1 finished in 0.2 seconds.
## Chain 2 finished in 0.2 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 0.2 seconds.
## Total execution time: 0.2 seconds.
```

```
ppp2 <- pp_check(MLU_model_prior_1, ndraws=100) + labs(title="Prior for model 2")
gridExtra::grid.arrange(ppp1, ppp2)
```



```
#fit the model
MLU_model <-
  brm(
    MLUformula,
    data = SimD,
    save_pars = save_pars(all = TRUE),
    family = lognormal,
    prior = MLU_prior,
    sample_prior = T,
    iter = 5000,
    warmup = 1000,
    cores = 8,
    chains = 2,
    backend = "cmdstanr",
    threads = threading(2),
    control = list(
      adapt_delta = 0.999,
      max_treedepth = 20))
```

```
## Running MCMC with 2 chains, at most 8 in parallel, with 2 thread(s) per chain...
##
## Chain 1 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 2 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 2 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 2 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 1 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 2 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 2 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 2 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 2 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 1 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 2 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 1 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 2 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 1 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 2 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 1 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 1 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 2 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 2 Iteration: 1001 / 5000 [ 20%]  (Sampling)
## Chain 1 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 1 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1 Iteration: 1001 / 5000 [ 20%]  (Sampling)
## Chain 2 Iteration: 1100 / 5000 [ 22%]  (Sampling)
## Chain 1 Iteration: 1100 / 5000 [ 22%]  (Sampling)
## Chain 1 Iteration: 1200 / 5000 [ 24%]  (Sampling)
## Chain 2 Iteration: 1200 / 5000 [ 24%]  (Sampling)
## Chain 1 Iteration: 1300 / 5000 [ 26%]  (Sampling)
## Chain 1 Iteration: 1400 / 5000 [ 28%]  (Sampling)
## Chain 2 Iteration: 1300 / 5000 [ 26%]  (Sampling)
## Chain 1 Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 1 Iteration: 1600 / 5000 [ 32%]  (Sampling)
## Chain 2 Iteration: 1400 / 5000 [ 28%]  (Sampling)
## Chain 1 Iteration: 1700 / 5000 [ 34%]  (Sampling)
## Chain 1 Iteration: 1800 / 5000 [ 36%]  (Sampling)
## Chain 2 Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 1 Iteration: 1900 / 5000 [ 38%]  (Sampling)
## Chain 1 Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 1 Iteration: 2100 / 5000 [ 42%]  (Sampling)
## Chain 2 Iteration: 1600 / 5000 [ 32%]  (Sampling)
## Chain 1 Iteration: 2200 / 5000 [ 44%]  (Sampling)
## Chain 1 Iteration: 2300 / 5000 [ 46%]  (Sampling)
## Chain 2 Iteration: 1700 / 5000 [ 34%]  (Sampling)
## Chain 1 Iteration: 2400 / 5000 [ 48%]  (Sampling)
## Chain 1 Iteration: 2500 / 5000 [ 50%]  (Sampling)
## Chain 2 Iteration: 1800 / 5000 [ 36%]  (Sampling)
## Chain 1 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 1 Iteration: 2700 / 5000 [ 54%]  (Sampling)
```

```
## Chain 2 Iteration: 1900 / 5000 [ 38%]  (Sampling)
## Chain 1 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 1 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 2 Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 1 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 1 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 2 Iteration: 2100 / 5000 [ 42%]  (Sampling)
## Chain 1 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 1 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 2 Iteration: 2200 / 5000 [ 44%]  (Sampling)
## Chain 1 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 1 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 2 Iteration: 2300 / 5000 [ 46%]  (Sampling)
## Chain 1 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 1 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 2 Iteration: 2400 / 5000 [ 48%]  (Sampling)
## Chain 1 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 1 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 2 Iteration: 2500 / 5000 [ 50%]  (Sampling)
## Chain 1 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 1 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 2 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 1 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 1 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 2 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 1 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 1 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 2 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 1 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 1 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 2 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 1 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 1 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 2 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 1 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 1 finished in 92.2 seconds.
## Chain 2 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 2 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 2 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 2 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 2 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 2 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 2 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 2 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 2 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 2 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 2 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 2 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 2 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 2 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 2 Iteration: 4500 / 5000 [ 90%]  (Sampling)
```

```
## Chain 2 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 2 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 2 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 2 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 2 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 2 finished in 146.1 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 119.1 seconds.
## Total execution time: 146.1 seconds.
```

```
MLU_model_1 <-
  brm(
    MLUformula_1,
    data = SimD,
    save_pars = save_pars(all = TRUE),
    family = lognormal,
    prior = MLU_prior_1,
    sample_prior = T,
    iter = 5000,
    warmup = 1000,
    cores = 2,
    chains = 2,
    backend = "cmdstanr",
    threads = threading(2),
    control = list(
      adapt_delta = 0.999,
      max_treedepth = 20))
```

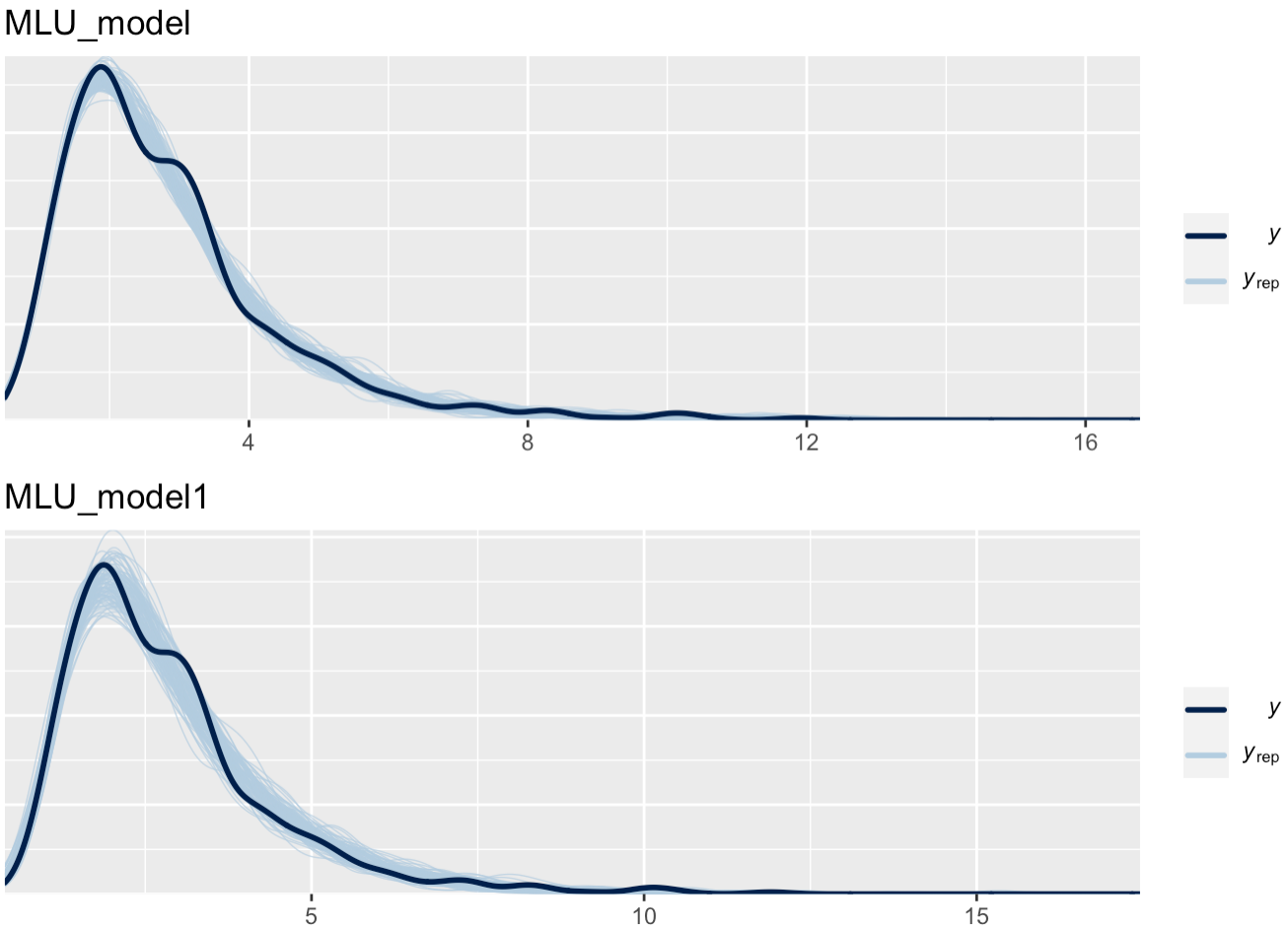
```
## Running MCMC with 2 parallel chains, with 2 thread(s) per chain...
##
## Chain 1 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 2 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 2 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 1 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 2 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 1 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 2 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 2 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 1 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 1 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 2 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 2 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 2 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 1 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 2 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 1 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 2 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 1 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 2 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 2 Iteration: 1001 / 5000 [ 20%]  (Sampling)
## Chain 1 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1 Iteration: 1001 / 5000 [ 20%]  (Sampling)
## Chain 2 Iteration: 1100 / 5000 [ 22%]  (Sampling)
## Chain 1 Iteration: 1100 / 5000 [ 22%]  (Sampling)
## Chain 2 Iteration: 1200 / 5000 [ 24%]  (Sampling)
## Chain 1 Iteration: 1200 / 5000 [ 24%]  (Sampling)
## Chain 2 Iteration: 1300 / 5000 [ 26%]  (Sampling)
## Chain 2 Iteration: 1400 / 5000 [ 28%]  (Sampling)
## Chain 1 Iteration: 1300 / 5000 [ 26%]  (Sampling)
## Chain 2 Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 1 Iteration: 1400 / 5000 [ 28%]  (Sampling)
## Chain 2 Iteration: 1600 / 5000 [ 32%]  (Sampling)
## Chain 1 Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 2 Iteration: 1700 / 5000 [ 34%]  (Sampling)
## Chain 2 Iteration: 1800 / 5000 [ 36%]  (Sampling)
## Chain 1 Iteration: 1600 / 5000 [ 32%]  (Sampling)
## Chain 2 Iteration: 1900 / 5000 [ 38%]  (Sampling)
## Chain 1 Iteration: 1700 / 5000 [ 34%]  (Sampling)
## Chain 2 Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 1 Iteration: 1800 / 5000 [ 36%]  (Sampling)
## Chain 2 Iteration: 2100 / 5000 [ 42%]  (Sampling)
## Chain 1 Iteration: 1900 / 5000 [ 38%]  (Sampling)
## Chain 2 Iteration: 2200 / 5000 [ 44%]  (Sampling)
## Chain 2 Iteration: 2300 / 5000 [ 46%]  (Sampling)
## Chain 1 Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 1 Iteration: 2100 / 5000 [ 42%]  (Sampling)
## Chain 2 Iteration: 2400 / 5000 [ 48%]  (Sampling)
```

```
## Chain 1 Iteration: 2200 / 5000 [ 44%]  (Sampling)
## Chain 2 Iteration: 2500 / 5000 [ 50%]  (Sampling)
## Chain 1 Iteration: 2300 / 5000 [ 46%]  (Sampling)
## Chain 2 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 1 Iteration: 2400 / 5000 [ 48%]  (Sampling)
## Chain 2 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 2 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 1 Iteration: 2500 / 5000 [ 50%]  (Sampling)
## Chain 2 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 1 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 2 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 1 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 2 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 1 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 2 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 2 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 1 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 2 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 1 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 2 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 1 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 2 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 2 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 1 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 2 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 1 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 2 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 1 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 2 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 1 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 2 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 1 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 2 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 1 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 2 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 2 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 1 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 2 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 2 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 1 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 2 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 2 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 1 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 2 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 1 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 1 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 2 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 2 finished in 4.4 seconds.
## Chain 1 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 1 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 1 Iteration: 4500 / 5000 [ 90%]  (Sampling)
```

```
## Chain 1 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 1 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 1 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 1 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 1 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 1 finished in 5.1 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 4.7 seconds.
## Total execution time: 5.2 seconds.
```

Model comparison (NV)

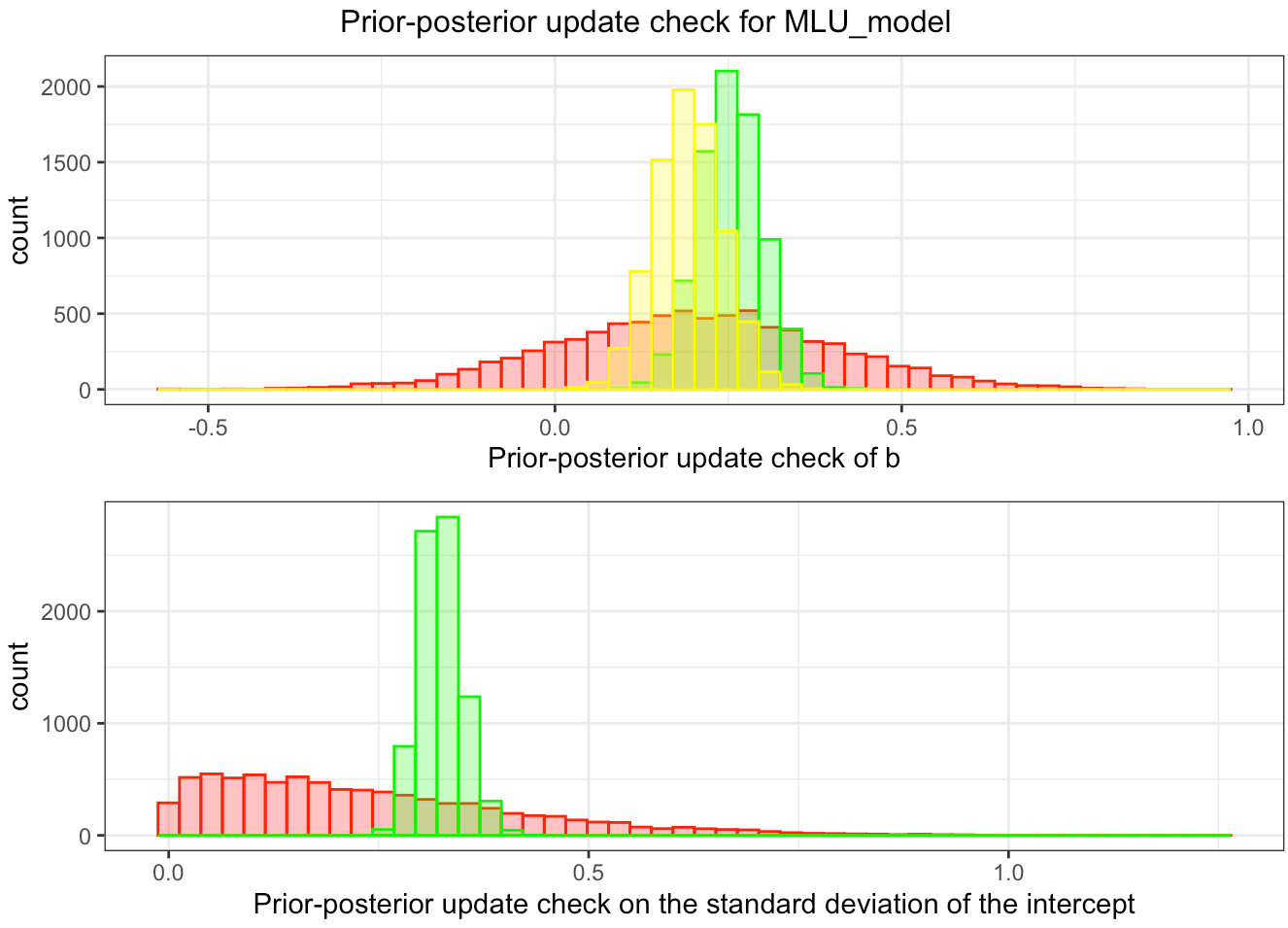
```
#model quality checks: posterior predictive checks
pp1 <- pp_check(MLU_model, ndraws=100) + labs(title = "MLU_model")
pp2 <- pp_check(MLU_model_1, ndraws=100)+ labs(title = "MLU_model1")
gridExtra::grid.arrange(pp1, pp2)
```



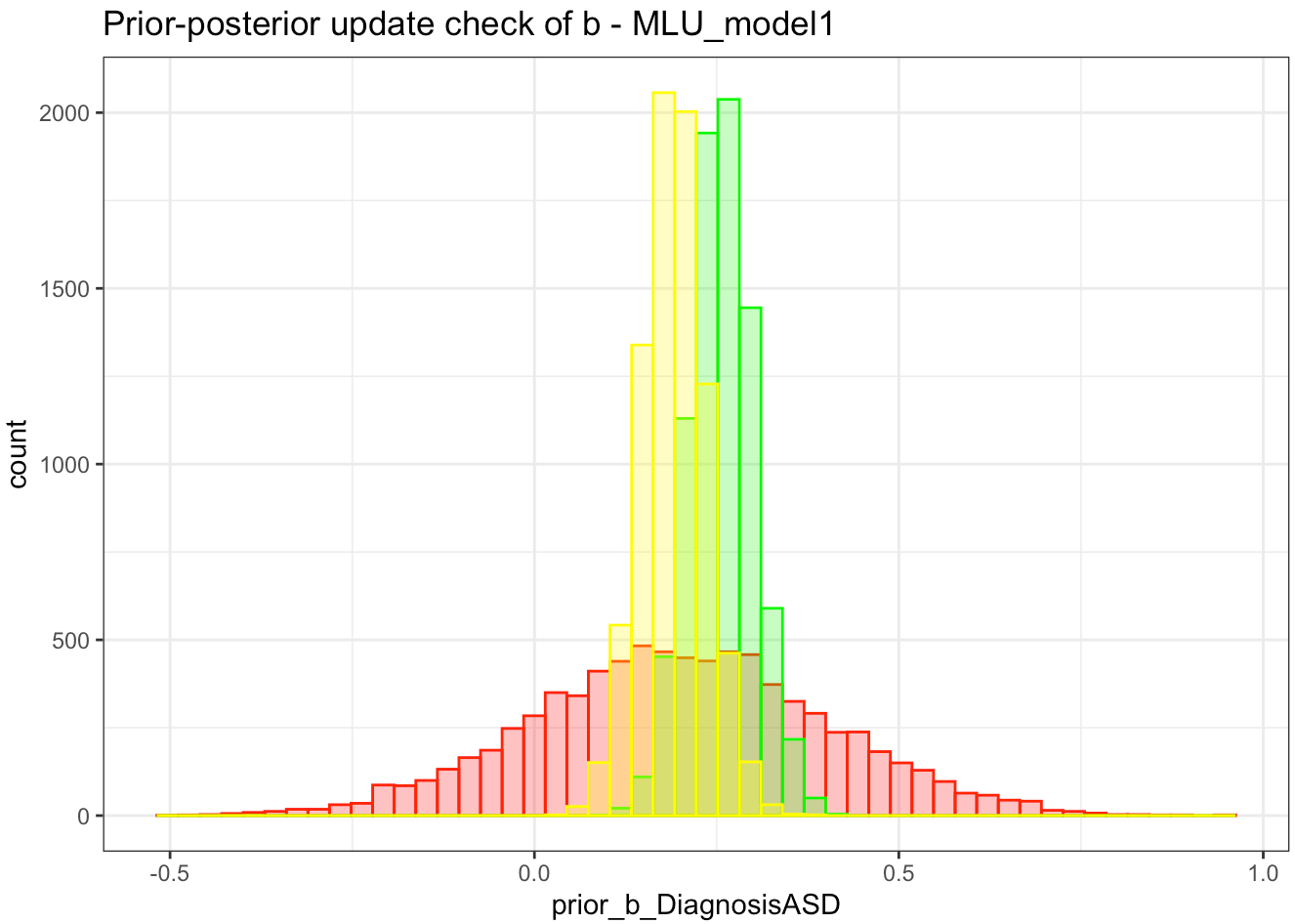
From the plot we can infer that MLU_model has the best performance

Model Quality checks (TS)

```
#model quality checks: prior-posterior update checks
posterior <- as_draws_df(MLU_model)
posterior2 <- as_draws_df(MLU_model_1)

p1 <- ggplot(posterior) + geom_histogram(aes(prior_b_DiagnosisASD), fill = "red", col
or = "red", alpha = 0.3, bins = 50) + geom_histogram(aes(b_DiagnosisASD), fill = "gre
en", color = "green", alpha = 0.3, bins = 50) + geom_histogram(aes(b_DiagnosisTD), fi
ll = "yellow", color = "yellow", alpha = 0.3, bins = 50) + theme_bw() + xlab("Prior-p
osterior update check of b")

p2 <- ggplot(posterior) + geom_histogram(aes(prior_sd_Kid__Intercept), fill = "red",
color = "red", alpha = 0.3, bins = 50) + geom_histogram(aes(sd_Kid__Intercept), fill
= "green", color = "green", alpha = 0.3, bins = 50) + theme_bw() + xlab("Prior-poster
ior update check on the standard deviation of the intercept")

p1_2 <- ggplot(posterior2) + geom_histogram(aes(prior_b_DiagnosisASD), fill = "red",
color = "red", alpha = 0.3, bins = 50) + geom_histogram(aes(b_DiagnosisASD), fill = "
green", color = "green", alpha = 0.3, bins = 50) + geom_histogram(aes(b_DiagnosisTD),
fill = "yellow", color = "yellow", alpha = 0.3, bins = 50) + theme_bw() + labs(title
="Prior-posterior update check of b - MLU_model1")

gridExtra::grid.arrange(grobs =list(p1,p2),top = "Prior-posterior update check for ML
U_model")
```



```
p1_2
```



K-fold cross validation (NV)

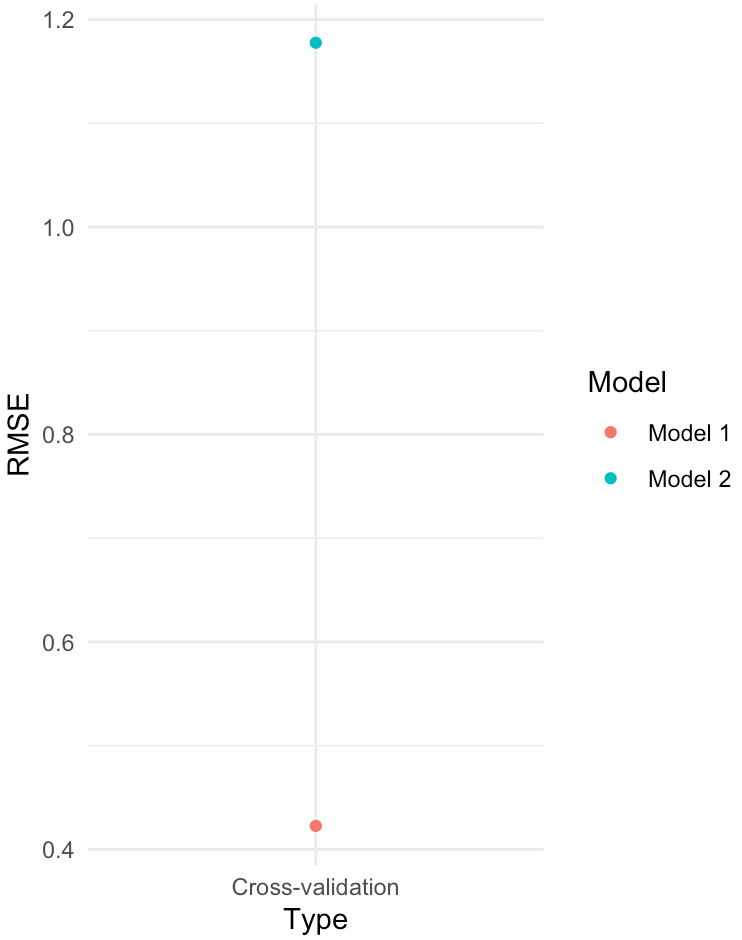
```
#k-fold cross validation for the two models
kfold_m1 <- kfold (MLU_model, folds = "stratified", group = "Diagnosis", K = 5, save_
fits = TRUE, cores = 4)
```

```
## Running MCMC with 2 sequential chains, with 2 thread(s) per chain...
##
## Chain 1 finished in 109.3 seconds.
## Chain 2 finished in 67.0 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 88.1 seconds.
## Total execution time: 176.5 seconds.
##
## Running MCMC with 2 sequential chains, with 2 thread(s) per chain...
##
## Chain 1 finished in 109.8 seconds.
## Chain 2 finished in 109.4 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 109.6 seconds.
## Total execution time: 219.4 seconds.
##
## Running MCMC with 2 sequential chains, with 2 thread(s) per chain...
##
## Chain 1 finished in 64.6 seconds.
## Chain 2 finished in 52.0 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 58.3 seconds.
## Total execution time: 116.8 seconds.
##
## Running MCMC with 2 sequential chains, with 2 thread(s) per chain...
##
## Chain 1 finished in 114.2 seconds.
## Chain 2 finished in 111.9 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 113.1 seconds.
## Total execution time: 226.2 seconds.
##
## Running MCMC with 2 sequential chains, with 2 thread(s) per chain...
##
## Chain 1 finished in 111.1 seconds.
## Chain 2 finished in 67.3 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 89.2 seconds.
## Total execution time: 178.5 seconds.
```

```
kfold_m2 <- kfold (MLU_model_1, folds = "stratified", group = "Diagnosis", K = 5, sav
e_fits = TRUE, cores = 4)
```

```
## Running MCMC with 2 sequential chains, with 2 thread(s) per chain...
##
## Chain 1 finished in 5.2 seconds.
## Chain 2 finished in 3.3 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 4.3 seconds.
## Total execution time: 8.7 seconds.
##
## Running MCMC with 2 sequential chains, with 2 thread(s) per chain...
##
## Chain 1 finished in 4.9 seconds.
## Chain 2 finished in 2.1 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 3.5 seconds.
## Total execution time: 7.2 seconds.
##
## Running MCMC with 2 sequential chains, with 2 thread(s) per chain...
##
## Chain 1 finished in 5.0 seconds.
## Chain 2 finished in 4.8 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 4.9 seconds.
## Total execution time: 9.9 seconds.
##
## Running MCMC with 2 sequential chains, with 2 thread(s) per chain...
##
## Chain 1 finished in 5.4 seconds.
## Chain 2 finished in 4.2 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 4.8 seconds.
## Total execution time: 9.8 seconds.
##
## Running MCMC with 2 sequential chains, with 2 thread(s) per chain...
##
## Chain 1 finished in 3.2 seconds.
## Chain 2 finished in 2.1 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 2.7 seconds.
## Total execution time: 5.5 seconds.
```

```
#Defining estimator RMSE
rmse <- function(y, yrep){
  yrep_mean <- colMeans(yrep)
  sqrt(mean((yrep_mean-y)^2))
}

#Split into y and yrep
kfold_pred_m1 <- kfold_predict(kfold_m1)
kfold_pred_m2 <- kfold_predict(kfold_m2)

#Computing the RMSE of the difference between the observed responses (y), and the kfo
ld predicted responses (yrep)
RMSE_m1 <- rmse(kfold_pred_m1$y, kfold_pred_m1$yrep)
RMSE_m2 <- rmse(kfold_pred_m2$y, kfold_pred_m2$yrep)

#Making a data frame containing the two cross-validations
RMSE_df <- tribble(~Model, ~Type, ~RMSE,
            "Model 1", "Cross-validation", RMSE_m1,
            "Model 2", "Cross-validation", RMSE_m2)
#Plotting it
RMSE_df %>%
  ggplot(aes(y = RMSE, x = Type, color = Model))+
  geom_point() + theme_minimal()
```



POWER ANALYSIS (NV & TS)

```
#n_sim <- 100
#s_100 <- tibble(seed = 1:n_sim) %>%
    #mutate(d = map(seed, simulation, n = 50)) %>%
    #mutate(fit = map2(d, seed, ~update(MLU_model, newdata = .x, seed = .y, iter = 10
00)))

#head(s_100)
```

```
#parameters <-
    #s_100 %>%
        #mutate(DiagnosisTD = map(fit, ~ fixef(.) %>%
        #data.frame() %>%
        #rownames_to_column("parameter"))) %>%
    #unnest(DiagnosisTD)

#parameters %>%
    #select(-d, -fit) %>%
    #filter(parameter == "DiagnosisTD") %>%
    #head()

#parameters %>%
    #filter(parameter == "DiagnosisTD") %>%
    #ggplot(aes(x = seed, y = Estimate, ymin = Q2.5, ymax = Q97.5)) +
    #geom_hline(yintercept=c(0,.5), color = "white") +
    #geom_pointrange(fatten = 1/2) +
    #labs(x = "seed (i.e., simulation index)",
    #y = expression(beta[1]))


#parameters %>%
    #filter(parameter == "DiagnosisTD") %>%
    #mutate(check = ifelse(Q2.5 > 0, 1, 0)) %>%
    #summarise(power = mean(check))
```

A good power value is 0.8 or above. With n = 100 the power of MLU_model is 0.83 ___

Part 2 - Strong in the Bayesian ken, you are now ready to analyse the actual data

2.1) - Describe your sample (n, age, gender, clinical and cognitive features of the two groups) and critically assess whether the groups (ASD and TD) are balanced. Briefly discuss whether the data is enough given the simulations in part 1.

Participant information (LL)

```
"### Participant information ###"
```

```
## [1] "### Participant information ###"
```

```
# paste n() unique participants
paste('Children participating:', length(unique(d$Child.ID)))
```

```
## [1] "Children participating: 61"
```

```
#Splitting dataframes into ASD and TD:
ASD_D <- d %>% filter(Diagnosis == "ASD" ) %>%  filter(Visit == 1)
TD_D <- d %>% filter(Diagnosis == "TD" ) %>% filter(Visit == 1)



#n() unique participants of both groups
paste("Of the 61 participants,",length(unique(ASD_D$Child.ID)),"participants have AS
D, and",length(unique(TD_D$Child.ID)),"participants is typically developing")
```

```
## [1] "Of the 61 participants, 29 participants have ASD, and 32 participants is typi
cally developing"
```

```
"### Gender ### "
```

```
## [1] "### Gender ### "
```

```
options(dplyr.summarise.inform = FALSE)
gender<- d %>% group_by(Diagnosis,Gender) %>% filter(Visit==1) %>% summarize(n())
paste("Of the 29 ASD participants",gender[1,3], "participants are female and",gender
[2,3],"are male")
```

```
## [1] "Of the 29 ASD participants 4 participants are female and 25 are male"
```

```
paste("Of the 32 TD participants",gender[3,3], "participants are female and",gender
[4,3],"are male")
```

```
## [1] "Of the 32 TD participants 6 participants are female and 26 are male"
```

```
"### Age ### "
```

```
## [1] "### Age ### "
```

```
# mean age and standard deviation of ASD Children
paste("The mean age (in months) of the ASD Children is",round(mean(ASD_D$Age,na.rm =
T),digits = 2),"with a standard deviation of",round(sd(ASD_D$Age,na.rm = T),2))
```

```
## [1] "The mean age (in months) of the ASD Children is 33 with a standard deviation
of 5.59"
```

```
# mean age and standard deviation of TD Children
paste("The mean age (in months) of the TD Children is",round(mean(TD_D$Age,na.rm =
T),digits = 2),"with a standard deviation of",round(sd(TD_D$Age,na.rm = T),2))
```

```
## [1] "The mean age (in months) of the TD Children is 20.37 with a standard deviatio
n of 1.49"
```

```
"### Non_Verbal IQ ### "
```

```
## [1] "### Non_Verbal IQ ### "
```

```
# mean nonverbal IQ and standard deviation of ASD Children
paste("The mean non-verbal IQ of the ASD Children is",round(mean(ASD_D$NonVerbalIQ,n
a.rm = T),digits = 2),"with a standard deviation of",round(sd(ASD_D$NonVerbalIQ,na.rm
= T),2))
```

```
## [1] "The mean non-verbal IQ of the ASD Children is 26.9 with a standard deviation
of 5.68"
```

```
# mean nonverbal IQ and standard deviation of TD Children
paste("The mean non-verbal IQ of the TD Children is",round(mean(TD_D$NonVerbalIQ,na.r
m = T),digits = 2),"with a standard deviation of",round(sd(TD_D$NonVerbalIQ,na.rm =
T),2))
```

```
## [1] "The mean non-verbal IQ of the TD Children is 26 with a standard deviation of
3.34"
```

```
"### Verbal IQ ### "
```

```
## [1] "### Verbal IQ ### "
```

```
# mean verbal IQ and standard deviation of ASD Children
paste("The mean verbal IQ of the ASD Children is",round(mean(ASD_D$VerbalIQ,na.rm =
T),digits = 2),"with a standard deviation of",round(sd(ASD_D$VerbalIQ,na.rm = T),2))
```

```
## [1] "The mean verbal IQ of the ASD Children is 17.31 with a standard deviation of
7.45"
```

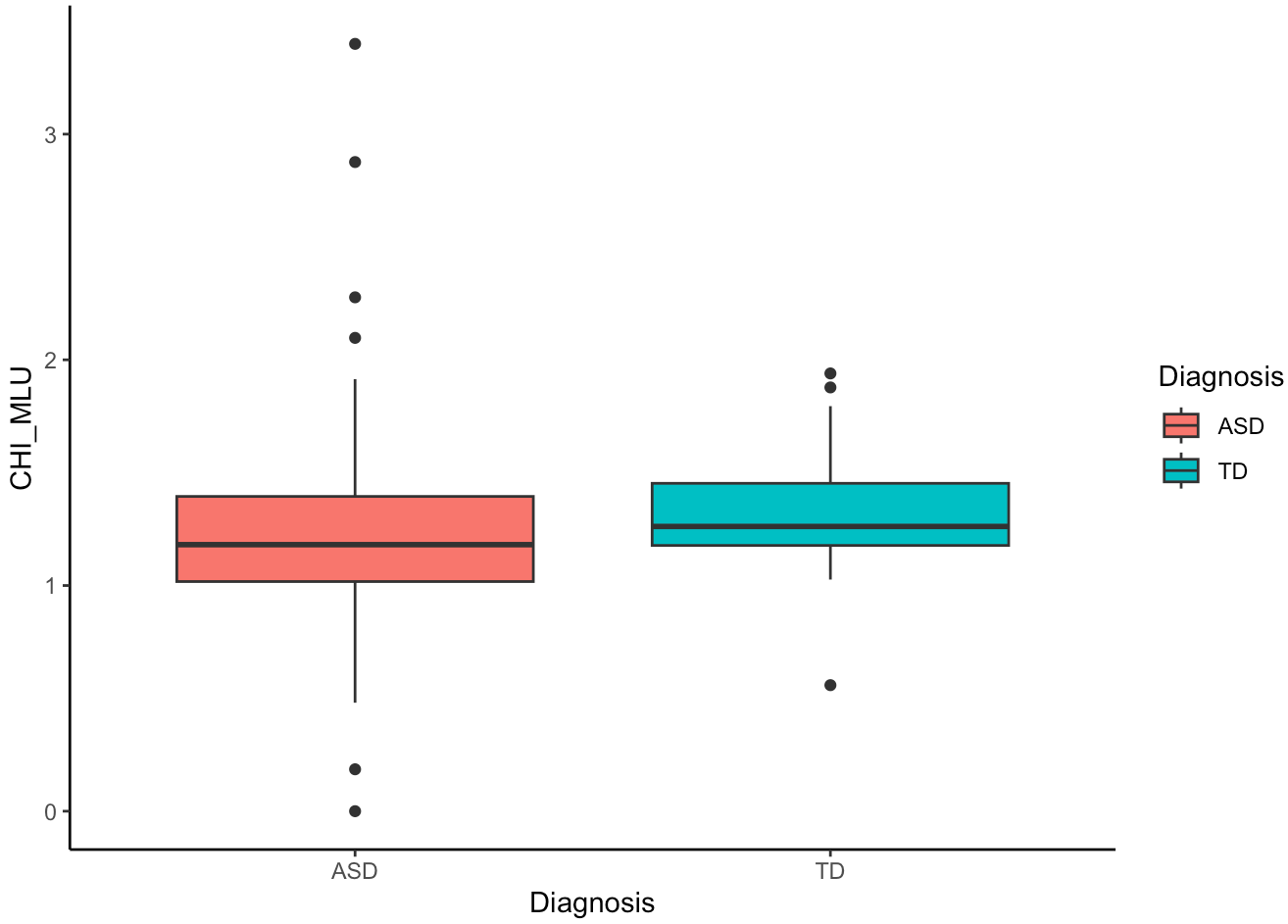
```
# mean verbal IQ and standard deviation of TD Children
paste("The mean verbal IQ of the TD Children is",round(mean(TD_D$VerbalIQ,na.rm = T),
digits = 2),"with a standard deviation of",round(sd(TD_D$VerbalIQ,na.rm = T),2))
```

```
## [1] "The mean verbal IQ of the TD Children is 20.22 with a standard deviation of
5.13"
```

```
rm(ASD_D, TD_D)
```

Finding outliers (EL & LL)

```
d %>%
  filter(Visit==1) %>%
  ggplot(aes(x=Diagnosis, y=CHI_MLU, fill=Diagnosis)) +
  geom_boxplot() +
  theme_classic()
```



Generally the sample is very balanced across the different parameters. ASD children differ from TD children with slightly higher standard deviations (around +2) in age, non-verbal IQ and verbal indicating a larger spread in variables for ASD compared to TD. Most prominently, the mean age of ASD is rather larger than age of TD.

___

2.2) - Describe linguistic development (in terms of MLU over time) in TD and ASD children (as a function of group). Discuss the difference (if any) between the two groups.

Modeling on empirical data (EL)

```
# as MLU_model performed the best on the simulation we are going to use that on the r
eal data to model the interaction between Diagnosis and Time/Visit

d_22 <- d %>%
  select(Child.ID, CHI_MLU, Visit, Diagnosis) %>%
  rename(
    Kid = Child.ID,
    MLU = CHI_MLU
  )

MLU_model_real_22 <-
  brm(
    MLUformula,
    data = d_22,
    save_pars = save_pars(all = TRUE),
    family = gaussian,
    prior = MLU_prior,
    #refit = "on_change",
    sample_prior = T,
    iter = 5000,
    warmup = 1000,
    cores = 2,
    chains = 2,
    backend = "cmdstanr",
    threads = threading(2),
    control = list(
      adapt_delta = 0.999,
      max_treedepth = 20))
```

```
## Running MCMC with 2 parallel chains, with 2 thread(s) per chain...
##
## Chain 1 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 2 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 2 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 1 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 2 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 2 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 1 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 2 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 2 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 2 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 1 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 2 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 2 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 1 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 2 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 1 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 1 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 1 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 2 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 2 Iteration: 1001 / 5000 [ 20%]  (Sampling)
## Chain 1 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1 Iteration: 1001 / 5000 [ 20%]  (Sampling)
## Chain 1 Iteration: 1100 / 5000 [ 22%]  (Sampling)
## Chain 1 Iteration: 1200 / 5000 [ 24%]  (Sampling)
## Chain 2 Iteration: 1100 / 5000 [ 22%]  (Sampling)
## Chain 1 Iteration: 1300 / 5000 [ 26%]  (Sampling)
## Chain 1 Iteration: 1400 / 5000 [ 28%]  (Sampling)
## Chain 1 Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 1 Iteration: 1600 / 5000 [ 32%]  (Sampling)
## Chain 2 Iteration: 1200 / 5000 [ 24%]  (Sampling)
## Chain 1 Iteration: 1700 / 5000 [ 34%]  (Sampling)
## Chain 1 Iteration: 1800 / 5000 [ 36%]  (Sampling)
## Chain 1 Iteration: 1900 / 5000 [ 38%]  (Sampling)
## Chain 1 Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 2 Iteration: 1300 / 5000 [ 26%]  (Sampling)
## Chain 1 Iteration: 2100 / 5000 [ 42%]  (Sampling)
## Chain 1 Iteration: 2200 / 5000 [ 44%]  (Sampling)
## Chain 1 Iteration: 2300 / 5000 [ 46%]  (Sampling)
## Chain 1 Iteration: 2400 / 5000 [ 48%]  (Sampling)
## Chain 2 Iteration: 1400 / 5000 [ 28%]  (Sampling)
## Chain 1 Iteration: 2500 / 5000 [ 50%]  (Sampling)
## Chain 1 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 1 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 2 Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 1 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 1 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 1 Iteration: 3000 / 5000 [ 60%]  (Sampling)
```

```
## Chain 1 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 2 Iteration: 1600 / 5000 [ 32%]  (Sampling)
## Chain 1 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 1 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 1 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 1 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 2 Iteration: 1700 / 5000 [ 34%]  (Sampling)
## Chain 1 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 1 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 1 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 1 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 2 Iteration: 1800 / 5000 [ 36%]  (Sampling)
## Chain 1 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 1 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 1 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 1 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 2 Iteration: 1900 / 5000 [ 38%]  (Sampling)
## Chain 1 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 1 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 1 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 2 Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 1 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 1 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 1 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 1 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 1 finished in 27.5 seconds.
## Chain 2 Iteration: 2100 / 5000 [ 42%]  (Sampling)
## Chain 2 Iteration: 2200 / 5000 [ 44%]  (Sampling)
## Chain 2 Iteration: 2300 / 5000 [ 46%]  (Sampling)
## Chain 2 Iteration: 2400 / 5000 [ 48%]  (Sampling)
## Chain 2 Iteration: 2500 / 5000 [ 50%]  (Sampling)
## Chain 2 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 2 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 2 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 2 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 2 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 2 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 2 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 2 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 2 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 2 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 2 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 2 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 2 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 2 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 2 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 2 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 2 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 2 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 2 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 2 Iteration: 4500 / 5000 [ 90%]  (Sampling)
```

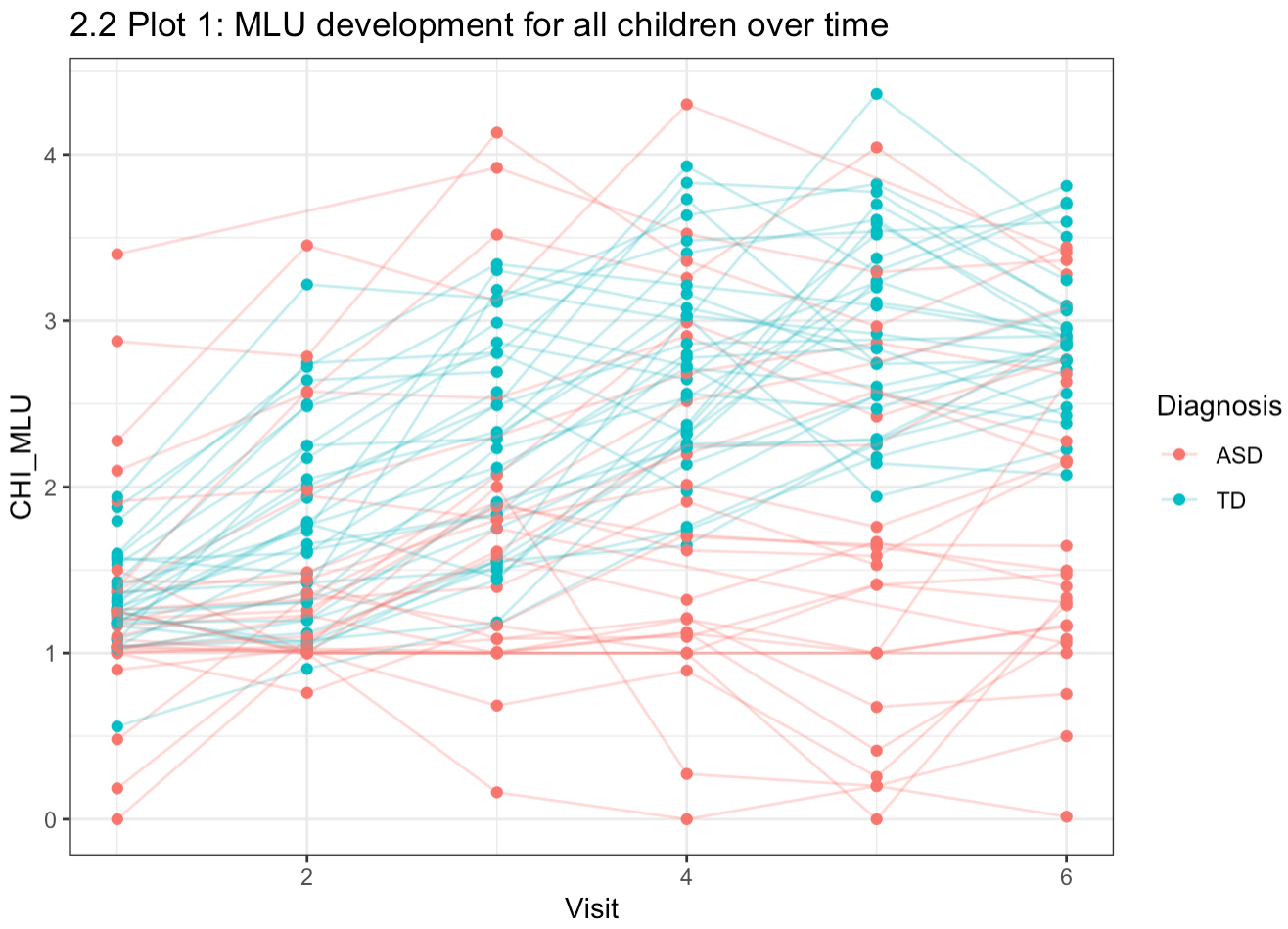
```
## Chain 2 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 2 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 2 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 2 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 2 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 2 finished in 63.8 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 45.6 seconds.
## Total execution time: 63.8 seconds.
```

MLU development (LL)

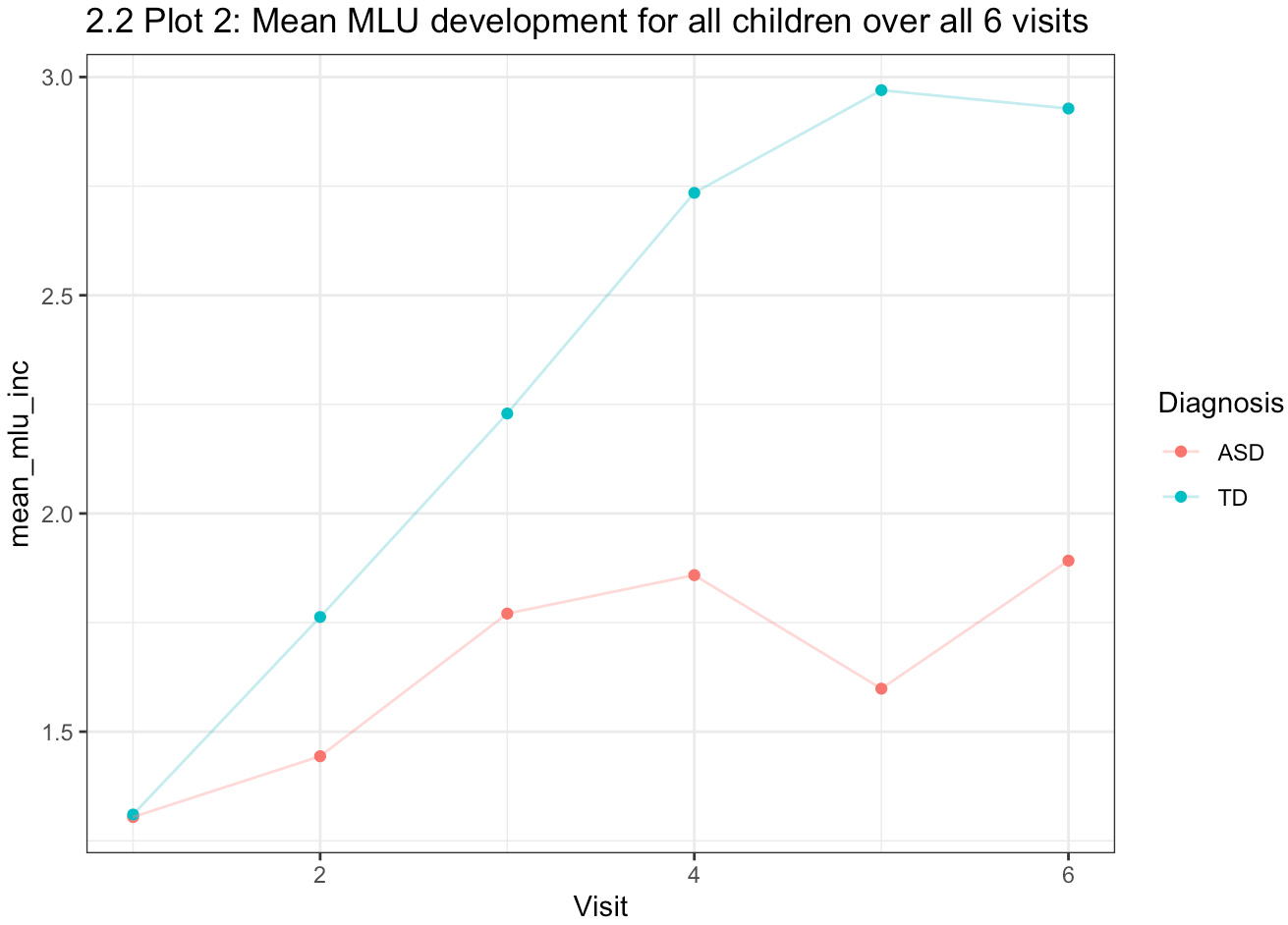
```
summary(MLU_model_real_22) #Summary for the best simulation formula made on the real data
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: MLU ~ 0 + Diagnosis + Diagnosis:Visit + (1 + Visit | Kid)
##    Data: d_22 (Number of observations: 352)
##   Draws: 2 chains, each with iter = 5000; warmup = 1000; thin = 1;
##          total post-warmup draws = 8000
##
## Group-Level Effects:
## ~Kid (Number of levels: 61)
##                     Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS
## sd(Intercept)           0.59      0.08     0.44     0.75 1.00     3775
## sd(Visit)               0.10      0.02     0.06     0.13 1.00     2499
## cor(Intercept,Visit)   -0.07      0.23    -0.46     0.44 1.00     2920
##                     Tail_ESS
## sd(Intercept)           4580
## sd(Visit)               2712
## cor(Intercept,Visit)    2807
##
## Population-Level Effects:
##                   Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## DiagnosisASD          0.98      0.12     0.74     1.20 1.00     5764     5729
## DiagnosisTD           0.85      0.11     0.62     1.05 1.00     5715     5730
## DiagnosisASD:Visit    0.12      0.03     0.07     0.18 1.00     7728     6249
## DiagnosisTD:Visit     0.37      0.02     0.32     0.42 1.00     8959     6015
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     0.40      0.02     0.37     0.44 1.00     4039     5059
##
## Draws were sampled using sample(hmc). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

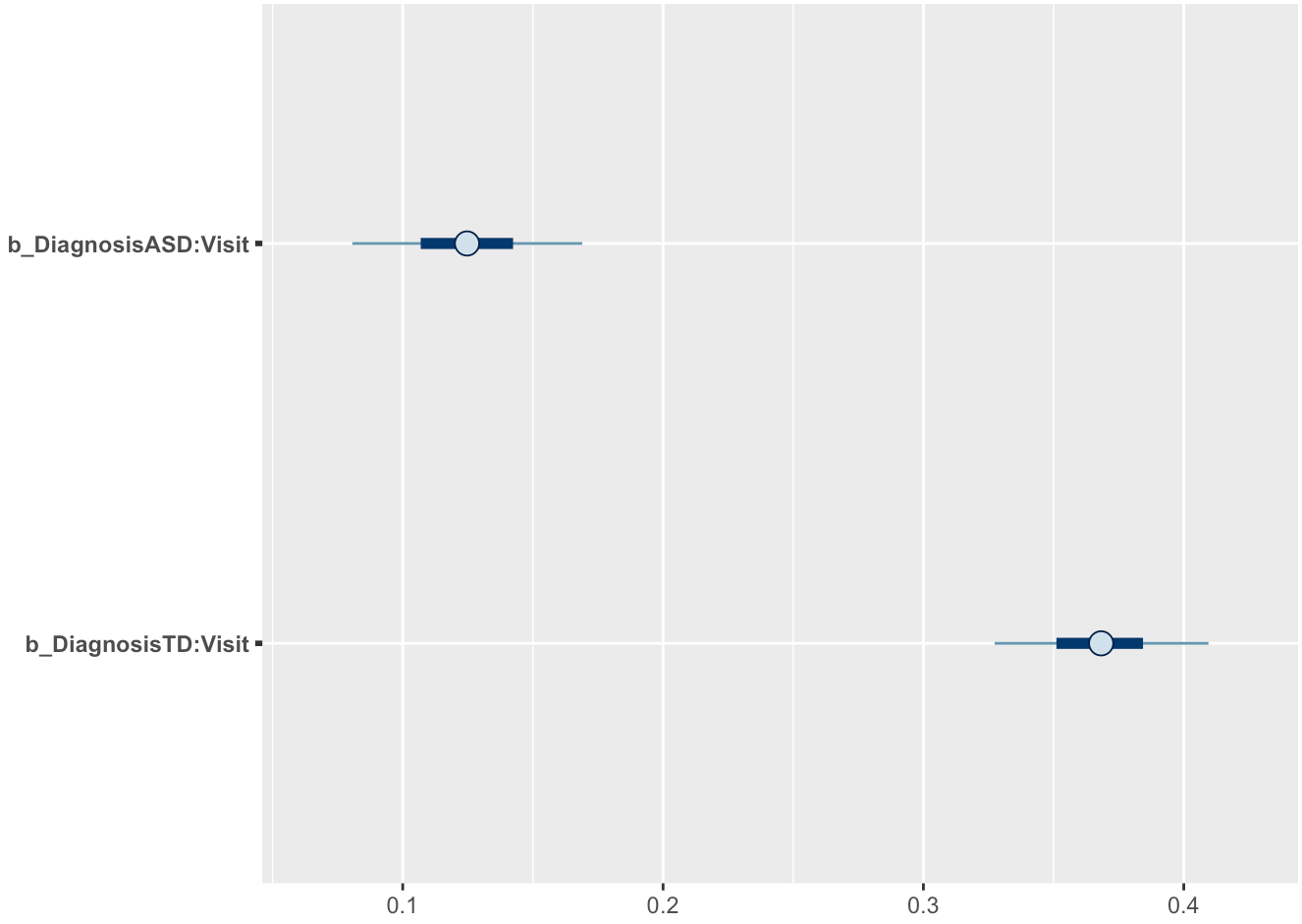
```
# Plotting the MLU development for all children over time
d %>%
  ggplot(aes(x = Visit, y = CHI_MLU, color = Diagnosis, group = Child.ID)) + geom_poi
nt() + geom_line(alpha = 0.3) +
  theme_bw() +
  ggtitle("2.2 Plot 1: MLU development for all children over time")
```




```
# Plotting the mean MLU development for both groups of children over time
d %>% group_by(Diagnosis, Visit) %>% summarise(mean_mlu_inc = mean(CHI_MLU) ) %>%
  ggplot(aes(x = Visit, y = mean_mlu_inc, color = Diagnosis)) + geom_point() + geom_l
ine(alpha = 0.3) +
  theme_bw() +
  ggtitle("2.2 Plot 2: Mean MLU development for all children over all 6 visits")
```




```
mcmc_intervals(MLU_model_real_22, pars = c("b_DiagnosisASD:Visit", "b_DiagnosisTD:Vis
it"))
```



___ *Differences between lingustic development of MLU between ASD and TD over time*

The model summary shows that the increase in MLU is generally less over time (per visit) in ASD children compared to TD children. In ASD children the MLU increases by 0.16 per visit while in TD children MLU increases 0.40 per visit.

This tendency is also reflected in plot 1 of the original data where MLU development over time generally steadily increases for TD children from visit 1 to 6. However, the MLU development in ASD children is a lot more varied. Here there is some children having larger decrements in MLU throughout the visits. Furthermore, other ASD children exhibit a very high baseline MLU at visit 1 and some have big increments towards higher MLU than TD children.

One could then suspect the variance of the diagnosisASD:Visit estimates to be larger than that of the TD children. If one looks at the mcmc_plot, this difference is true, but still the difference is not too critical. We can also rule out that this is due to differences in priors of MLU as they are kept the same for each group of children.

Taking a look at plot 2 there is 2 different general trajectories for each group of children where ASD children has a lower has a lower increase in MLU pr Visit but also their development stagnates already at visit 3 compared to visit 5 for TD children. It is important to note the inconsistency of ASD children MLU development trajectories as previously mentioned making plot 2 ASD mean trajectory less reliable due to individual ___

2.3) - Describe individual differences in linguistic development: do all kids follow the same path? Are all kids reflected by td for their group?

Individual MLU development (EL)

```
p <- c(0.25, 0.5, 0.75) # set quantiles lower, median, upper

p_names <- map_chr(p, ~paste0(.x*100, "%")) # saving the names for coloumns

p_funs <- map(p, ~partial(quantile, probs = .x, na.rm = TRUE)) %>%
  set_names(nm = p_names) #

# creating a df with quantiles where lower is 20% - mid is 50% - upper is 80%
d_22_quantiles <- d_22 %>%
  group_by(Diagnosis, Visit) %>%
  summarize_at(vars(MLU), funs(!!!p_funs)) %>%
  rename(
    lower = 3,
    mid = 4,
    upper = 5
  )
```

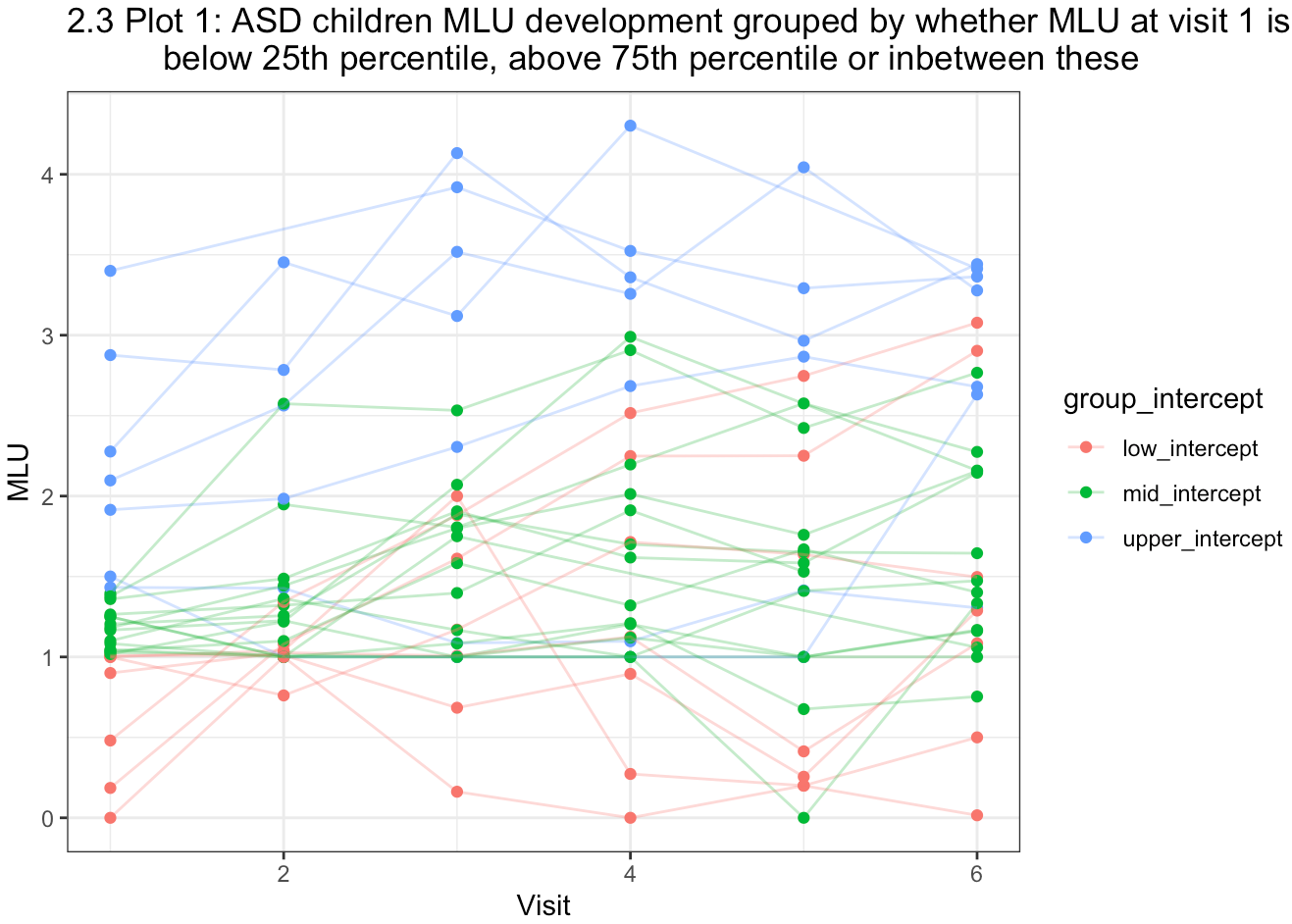
```
## Warning: `funs()` was deprecated in dplyr 0.8.0.
## ℹ Please use a list of either functions or lambdas:
##
## # Simple named list: list(mean = mean, median = median)
##
## # Auto named with `tibble::lst()`: tibble::lst(mean, median)
##
## # Using lambdas list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
```

```
d_22_w_quantiles <- left_join(d_22, d_22_quantiles) # joining the summarised quantile
s with children id's
```

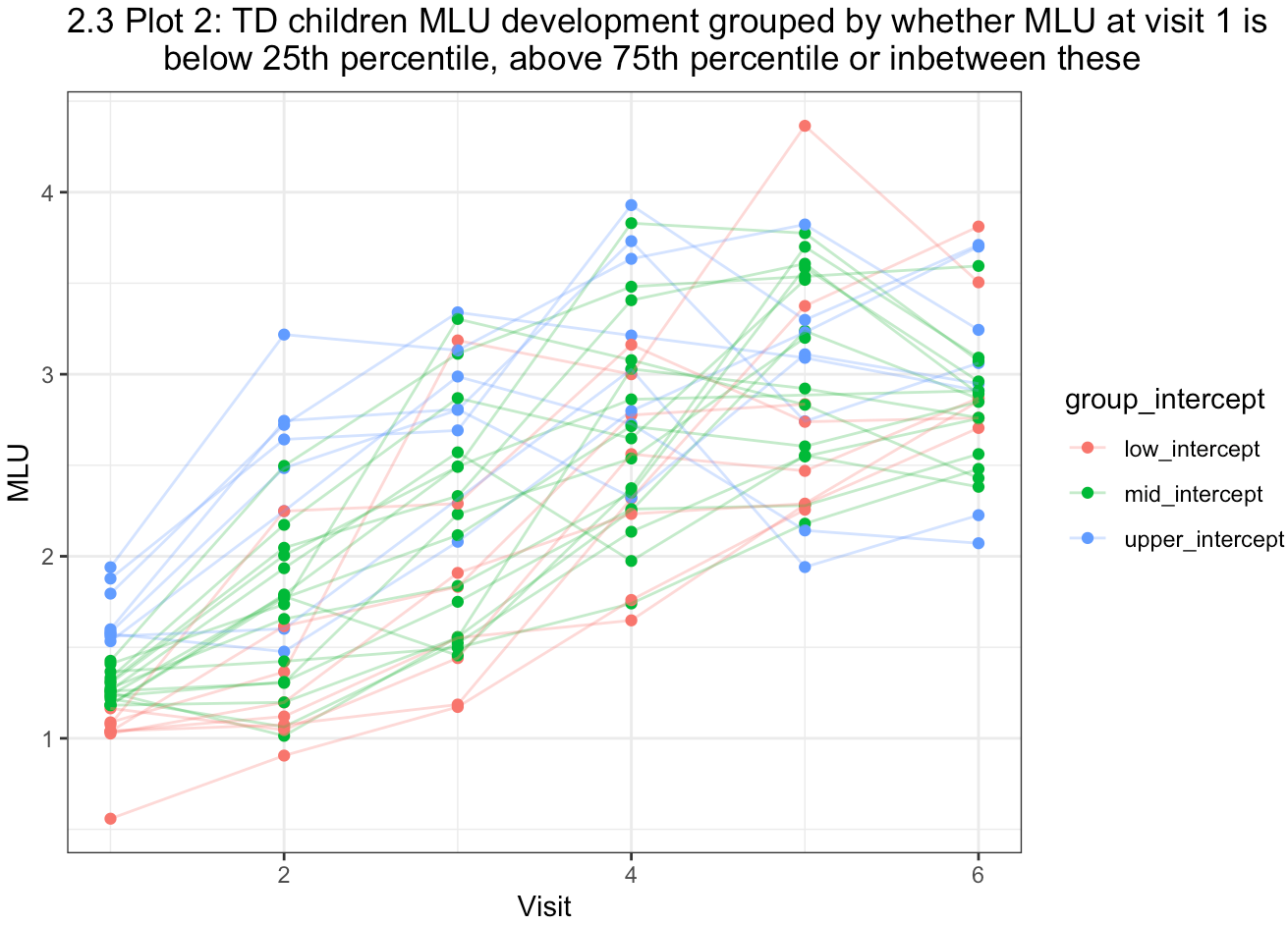
(LL)

```
d_22_w_quantiles <- d_22_w_quantiles %>%
  mutate(
         group_intercept=
    case_when( # creating a new column with low, mid and upper intercept for each MLU
at visit 1 conditioned by quantiles
    (Visit==1) & (MLU < lower) ~ "low_intercept",
    (Visit==1) & (lower <= MLU) & (MLU <= upper) ~ "mid_intercept",
    (Visit==1) & (upper < MLU)  ~ "upper_intercept",
  ) ) %>% group_by(Kid) %>%
    mutate(group_intercept= first(na.omit(group_intercept))) # taking the group_inter
cept for visit 1 and duplicating for each child
```

```
d_22_w_quantiles %>% filter(Diagnosis=="ASD") %>%
  ggplot(aes(x = Visit, y = MLU, color = group_intercept, group = Kid)) + geom_point
() + geom_line(alpha = 0.3) +
  theme_bw() +
  ggtitle("2.3 Plot 1: ASD children MLU development grouped by whether MLU at visit 1
is
          below 25th percentile, above 75th percentile or inbetween these")
```




```
d_22_w_quantiles %>% filter(Diagnosis=="TD") %>%
  ggplot(aes(x = Visit, y = MLU, color = group_intercept, group = Kid)) + geom_point
() + geom_line(alpha = 0.3) +
  theme_bw() +
  ggtitle("2.3 Plot 2: TD children MLU development grouped by whether MLU at visit 1
is
          below 25th percentile, above 75th percentile or inbetween these")
```



___ All the kids follow a general trend of increasing their mean length of utterance pr visit. Their linguistic development is therefore a positive one with some variablity per visit. The ASD kids generally have a slower increase of linguistic development than TD kids. There are a few outliers in the ASD group, as the variance in their visit effect has a greater SD (which we can see in plot 2.3 1). This means that even though the ASD kids generally still increase their linguistic abilities, the ASD kids may be more prone to be affected by day-to-day variations such as stimulation and mood. An ASD kid may therefore become non-verbal during a single visit, and then return to their predictied linguistic abilities.

Some of the ASD kids also have a higher linguistic baseline compared to TD kids. However, these kids are outliers.

2.4) - Include additional predictors in your model of language development (N.B. not other indexes of child language: types and tokens, that'd be cheating). Identify the best model, by conceptual reasoning, model comparison or a mix. Report the model you choose (and name its competitors, if any) and discuss why it's the best model.

Model with additional predictors (LL & EL)

```
MLUformula_real_data <- bf(CHI_MLU ~ Diagnosis + Visit + Age)

MLU_model_real_datax <-
  brm(
    MLUformula_real_data,
    data = d,
    save_pars = save_pars(all = TRUE),
    family = gaussian,
    #refit = "on_change",
    sample_prior = T,
    iter = 5000,
    warmup = 1000,
    cores = 8,
    chains = 2,
    backend = "cmdstanr",
    threads = threading(2),
    control = list(
      adapt_delta = 0.999,
      max_treedepth = 20))
```

```
## Warning: Rows containing NAs were excluded from the model.
```

```
## Running MCMC with 2 chains, at most 8 in parallel, with 2 thread(s) per chain...
##
## Chain 1 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 2 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 2 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 1 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 1 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 1 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 2 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 2 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 2 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 1 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 1 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 2 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 2 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 2 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 2 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 2 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 1 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 1 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 1 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1 Iteration: 1001 / 5000 [ 20%]  (Sampling)
## Chain 2 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 2 Iteration: 1001 / 5000 [ 20%]  (Sampling)
## Chain 2 Iteration: 1100 / 5000 [ 22%]  (Sampling)
## Chain 1 Iteration: 1100 / 5000 [ 22%]  (Sampling)
## Chain 2 Iteration: 1200 / 5000 [ 24%]  (Sampling)
## Chain 1 Iteration: 1200 / 5000 [ 24%]  (Sampling)
## Chain 1 Iteration: 1300 / 5000 [ 26%]  (Sampling)
## Chain 2 Iteration: 1300 / 5000 [ 26%]  (Sampling)
## Chain 1 Iteration: 1400 / 5000 [ 28%]  (Sampling)
## Chain 1 Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 2 Iteration: 1400 / 5000 [ 28%]  (Sampling)
## Chain 2 Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 1 Iteration: 1600 / 5000 [ 32%]  (Sampling)
## Chain 1 Iteration: 1700 / 5000 [ 34%]  (Sampling)
## Chain 1 Iteration: 1800 / 5000 [ 36%]  (Sampling)
## Chain 2 Iteration: 1600 / 5000 [ 32%]  (Sampling)
## Chain 1 Iteration: 1900 / 5000 [ 38%]  (Sampling)
## Chain 1 Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 2 Iteration: 1700 / 5000 [ 34%]  (Sampling)
## Chain 2 Iteration: 1800 / 5000 [ 36%]  (Sampling)
## Chain 1 Iteration: 2100 / 5000 [ 42%]  (Sampling)
## Chain 1 Iteration: 2200 / 5000 [ 44%]  (Sampling)
## Chain 2 Iteration: 1900 / 5000 [ 38%]  (Sampling)
## Chain 1 Iteration: 2300 / 5000 [ 46%]  (Sampling)
## Chain 1 Iteration: 2400 / 5000 [ 48%]  (Sampling)
## Chain 2 Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 1 Iteration: 2500 / 5000 [ 50%]  (Sampling)
```

```
## Chain 1 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 2 Iteration: 2100 / 5000 [ 42%]  (Sampling)
## Chain 2 Iteration: 2200 / 5000 [ 44%]  (Sampling)
## Chain 1 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 1 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 1 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 2 Iteration: 2300 / 5000 [ 46%]  (Sampling)
## Chain 2 Iteration: 2400 / 5000 [ 48%]  (Sampling)
## Chain 1 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 1 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 2 Iteration: 2500 / 5000 [ 50%]  (Sampling)
## Chain 1 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 1 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 2 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 2 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 1 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 1 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 1 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 2 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 2 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 1 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 1 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 2 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 1 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 1 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 2 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 2 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 1 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 1 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 2 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 1 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 1 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 1 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 2 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 2 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 1 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 1 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 1 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 2 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 1 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 1 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 2 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 2 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 1 finished in 2.3 seconds.
## Chain 2 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 2 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 2 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 2 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 2 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 2 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 2 Iteration: 4500 / 5000 [ 90%]  (Sampling)
```

```
## Chain 2 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 2 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 2 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 2 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 2 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 2 finished in 2.7 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 2.5 seconds.
## Total execution time: 2.8 seconds.
```

```
MLUformula_real_data_1 <- bf(CHI_MLU ~ Diagnosis + Visit + Age + Socialization)

MLU_model_real_data_1x <-
  brm(
    MLUformula_real_data_1,
    data = d,
    save_pars = save_pars(all = TRUE),
    family = gaussian,
    #refit = "on_change",
    sample_prior = T,
    iter = 5000,
    warmup = 1000,
    cores = 8,
    chains = 2,
    backend = "cmdstanr",
    threads = threading(2),
    control = list(
      adapt_delta = 0.999,
      max_treedepth = 20))
```

```
## Warning: Rows containing NAs were excluded from the model.
```

```
## Running MCMC with 2 chains, at most 8 in parallel, with 2 thread(s) per chain...
##
## Chain 1 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 2 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 2 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 1 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 1 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 1 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 2 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 1 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 1 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 2 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 2 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 1 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 1 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 2 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 2 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 1 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1 Iteration: 1001 / 5000 [ 20%]  (Sampling)
## Chain 1 Iteration: 1100 / 5000 [ 22%]  (Sampling)
## Chain 2 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 2 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 2 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 1 Iteration: 1200 / 5000 [ 24%]  (Sampling)
## Chain 1 Iteration: 1300 / 5000 [ 26%]  (Sampling)
## Chain 2 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 2 Iteration: 1001 / 5000 [ 20%]  (Sampling)
## Chain 1 Iteration: 1400 / 5000 [ 28%]  (Sampling)
## Chain 1 Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 2 Iteration: 1100 / 5000 [ 22%]  (Sampling)
## Chain 1 Iteration: 1600 / 5000 [ 32%]  (Sampling)
## Chain 1 Iteration: 1700 / 5000 [ 34%]  (Sampling)
## Chain 2 Iteration: 1200 / 5000 [ 24%]  (Sampling)
## Chain 2 Iteration: 1300 / 5000 [ 26%]  (Sampling)
## Chain 1 Iteration: 1800 / 5000 [ 36%]  (Sampling)
## Chain 1 Iteration: 1900 / 5000 [ 38%]  (Sampling)
## Chain 1 Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 2 Iteration: 1400 / 5000 [ 28%]  (Sampling)
## Chain 1 Iteration: 2100 / 5000 [ 42%]  (Sampling)
## Chain 1 Iteration: 2200 / 5000 [ 44%]  (Sampling)
## Chain 2 Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 1 Iteration: 2300 / 5000 [ 46%]  (Sampling)
## Chain 1 Iteration: 2400 / 5000 [ 48%]  (Sampling)
## Chain 2 Iteration: 1600 / 5000 [ 32%]  (Sampling)
## Chain 1 Iteration: 2500 / 5000 [ 50%]  (Sampling)
## Chain 1 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 2 Iteration: 1700 / 5000 [ 34%]  (Sampling)
## Chain 1 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 1 Iteration: 2800 / 5000 [ 56%]  (Sampling)
```

```
## Chain 2 Iteration: 1800 / 5000 [ 36%]  (Sampling)
## Chain 1 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 1 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 1 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 2 Iteration: 1900 / 5000 [ 38%]  (Sampling)
## Chain 1 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 1 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 2 Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 2 Iteration: 2100 / 5000 [ 42%]  (Sampling)
## Chain 1 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 1 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 2 Iteration: 2200 / 5000 [ 44%]  (Sampling)
## Chain 1 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 1 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 2 Iteration: 2300 / 5000 [ 46%]  (Sampling)
## Chain 1 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 1 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 2 Iteration: 2400 / 5000 [ 48%]  (Sampling)
## Chain 1 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 1 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 2 Iteration: 2500 / 5000 [ 50%]  (Sampling)
## Chain 1 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 1 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 2 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 1 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 1 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 2 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 1 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 1 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 2 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 1 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 1 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 2 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 1 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 2 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 2 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 1 finished in 2.5 seconds.
## Chain 2 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 2 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 2 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 2 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 2 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 2 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 2 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 2 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 2 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 2 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 2 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 2 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 2 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 2 Iteration: 4500 / 5000 [ 90%]  (Sampling)
```

```
## Chain 2 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 2 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 2 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 2 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 2 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 2 finished in 3.4 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 2.9 seconds.
## Total execution time: 3.5 seconds.
```

(LL)

```
loo1 <- loo(MLU_model_real_datax)
loo2 <- loo(MLU_model_real_data_1x)

comp <- loo_compare(loo1, loo2)

print(comp, simplify = FALSE, digits = 3)
```

```
##                         elpd_diff se_diff  elpd_loo se_elpd_loo p_loo
## MLU_model_real_data_1x    0.000     0.000 -389.981   13.730       6.165
## MLU_model_real_datax    -15.666     5.995 -405.647   15.225       5.246
##                         se_p_loo looic    se_looic
## MLU_model_real_data_1x    0.623  779.961   27.460
## MLU_model_real_datax      0.608  811.294   30.451
```

K-fold cross validation for additional models (NV)

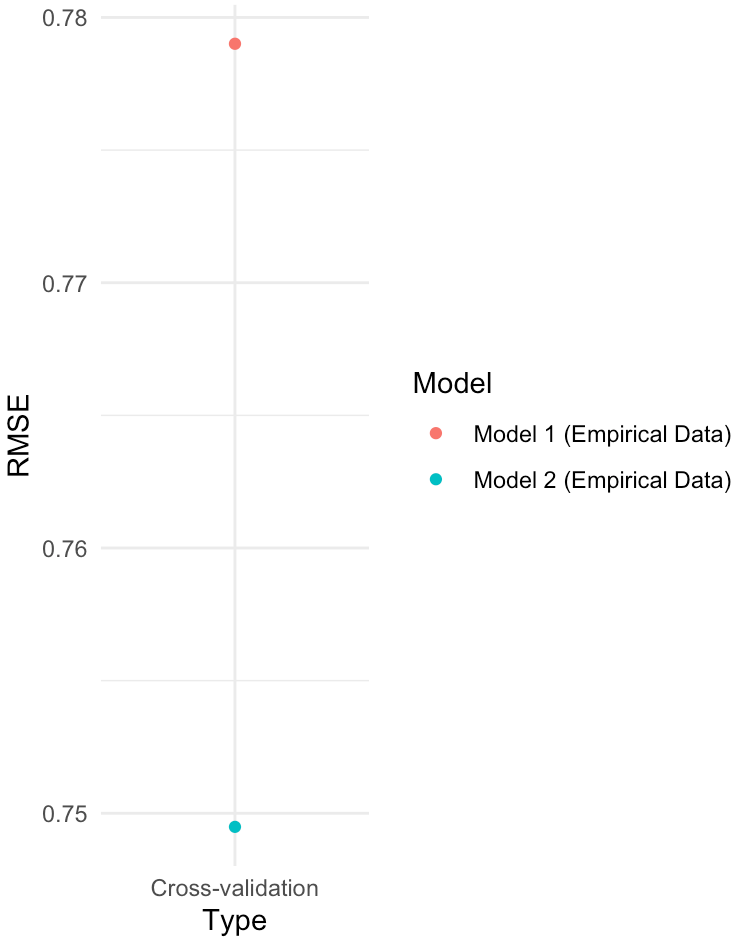
```
#k-fold cross validation for the two models
kfold_real_m1 <- kfold (MLU_model_real_datax, folds = "stratified", group = "Diagnosi
s", K = 5, save_fits = TRUE, cores = 4)
```

```
## Running MCMC with 2 sequential chains, with 2 thread(s) per chain...
##
## Chain 1 finished in 1.2 seconds.
## Chain 2 finished in 1.7 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 1.5 seconds.
## Total execution time: 3.0 seconds.
##
## Running MCMC with 2 sequential chains, with 2 thread(s) per chain...
##
## Chain 1 finished in 2.0 seconds.
## Chain 2 finished in 1.1 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 1.5 seconds.
## Total execution time: 3.2 seconds.
##
## Running MCMC with 2 sequential chains, with 2 thread(s) per chain...
##
## Chain 1 finished in 1.0 seconds.
## Chain 2 finished in 1.3 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 1.2 seconds.
## Total execution time: 2.4 seconds.
##
## Running MCMC with 2 sequential chains, with 2 thread(s) per chain...
##
## Chain 1 finished in 1.0 seconds.
## Chain 2 finished in 1.5 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 1.3 seconds.
## Total execution time: 2.6 seconds.
##
## Running MCMC with 2 sequential chains, with 2 thread(s) per chain...
##
## Chain 1 finished in 1.6 seconds.
## Chain 2 finished in 1.0 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 1.3 seconds.
## Total execution time: 2.7 seconds.
```

```
kfold_real_m2 <- kfold (MLU_model_real_data_1x, folds = "stratified", group = "Diagno
sis", K = 5, save_fits = TRUE, cores = 4)
```

```
## Running MCMC with 2 sequential chains, with 2 thread(s) per chain...
##
## Chain 1 finished in 2.3 seconds.
## Chain 2 finished in 2.1 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 2.2 seconds.
## Total execution time: 4.5 seconds.
##
## Running MCMC with 2 sequential chains, with 2 thread(s) per chain...
##
## Chain 1 finished in 2.2 seconds.
## Chain 2 finished in 2.2 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 2.2 seconds.
## Total execution time: 4.4 seconds.
##
## Running MCMC with 2 sequential chains, with 2 thread(s) per chain...
##
## Chain 1 finished in 2.3 seconds.
## Chain 2 finished in 1.4 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 1.9 seconds.
## Total execution time: 3.9 seconds.
##
## Running MCMC with 2 sequential chains, with 2 thread(s) per chain...
##
## Chain 1 finished in 1.6 seconds.
## Chain 2 finished in 1.1 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 1.3 seconds.
## Total execution time: 2.7 seconds.
##
## Running MCMC with 2 sequential chains, with 2 thread(s) per chain...
##
## Chain 1 finished in 2.2 seconds.
## Chain 2 finished in 2.1 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 2.2 seconds.
## Total execution time: 4.5 seconds.
```

```
#Defining estimator RMSE
rmse <- function(y, yrep){
  yrep_mean <- colMeans(yrep)
  sqrt(mean((yrep_mean-y)^2))
}

#Split into y and yrep
#Kfold
kfold_pred_real_m1 <- kfold_predict(kfold_real_m1)
kfold_pred_real_m2 <- kfold_predict(kfold_real_m2)

#Computing the RMSE of the difference between the observed responses (y), and the kfo
ld predicted responses (yrep)
RMSE_real_kfold_m1 <- rmse(kfold_pred_real_m1$y, kfold_pred_real_m1$yrep)
RMSE_real_kfold_m2 <- rmse(kfold_pred_real_m2$y, kfold_pred_real_m2$yrep)

#Making a data frame containing the two cross-validations
RMSE_real_df <- tribble(~Model, ~Type, ~RMSE,
            "Model 1 (Empirical Data)", "Cross-validation", RMSE_real_kfold_m1,
            "Model 2 (Empirical Data)", "Cross-validation", RMSE_real_kfold_m2)
#Plotting it
RMSE_real_df %>%
  ggplot(aes(y = RMSE, x = Type, color = Model))+
  geom_point() + theme_minimal()
```



Comparison analysis We have chosen to compare two models with additional predictors: A model using age and a model using age and socialization

Theoretical comparison: The child's socialization in secondary settings should have an effect on the childs word accusition.

A model including socialization as a predictor should therefore be better at predicting MLU pr visit than ethnicity.

Leave one out comparison Using the loo() and loo_compare() function the socialization model turns out to have a greater ELDP as the theory would predict. *The model utilizing socialization therefore seems to be the best model based on model comparison and conceptual reasoning.*

K-fold Cross validation This conclusion is also supported by a K-fold cross validation, where the socialization model has the lowest RMSE. ___