Assignment 3: Machine learning

3 main parts: 1. Create a skeptical and an informed simulation, based on the meta-analysis. 2. Build and test our machine learning pipeline on the simulated data. 3. Apply the pipeline to the empirical data.

Report: - Describe your machine learning pipeline. Produce a diagram of it to guide the reader (e.g. see Rybner et al 2022 Vocal markers of autism: Assessing the generalizability of ML models), and describe the different parts: data budgeting, data preprocessing, model choice and training, assessment of performance. - Briefly justify and describe your use of simulated data, and results from the pipeline on them. - Describe results from applying the ML pipeline to the empirical data and what can we learn from them.

```
pacman::p_load(tidyverse, dplyr, tidybayes, ggplot2, ggridges, plyr, brms, cmdstanr, gridExtra, readxl, tidymodel
s, knitr, dotwhisker, DALEX, DALEXtra, tidytext, reshape, stringr)
```

Part I - Simulating data

Use the meta-analysis reported in Parola et al (2020), create a simulated dataset with 100 matched pairs of schizophrenia and controls, each participant producing 10 repeated measures (10 trials with their speech recorded). for each of these "recordings" (data points) produce 10 acoustic measures: 6 from the meta-analysis, 4 with just random noise. Do the same for a baseline data set including only 10 noise variables. Tip: see the slides for the code.

Taken from Parola et al (2020).

Values I use for InformedEffectMean: *-0.55* - for pitch variability; *-1.26* - proportion of spoken time; *-0.75* - speech rate; *1.89* - duration of pauses; *0.25* - pitch mean; *0.05* - number of pauses.

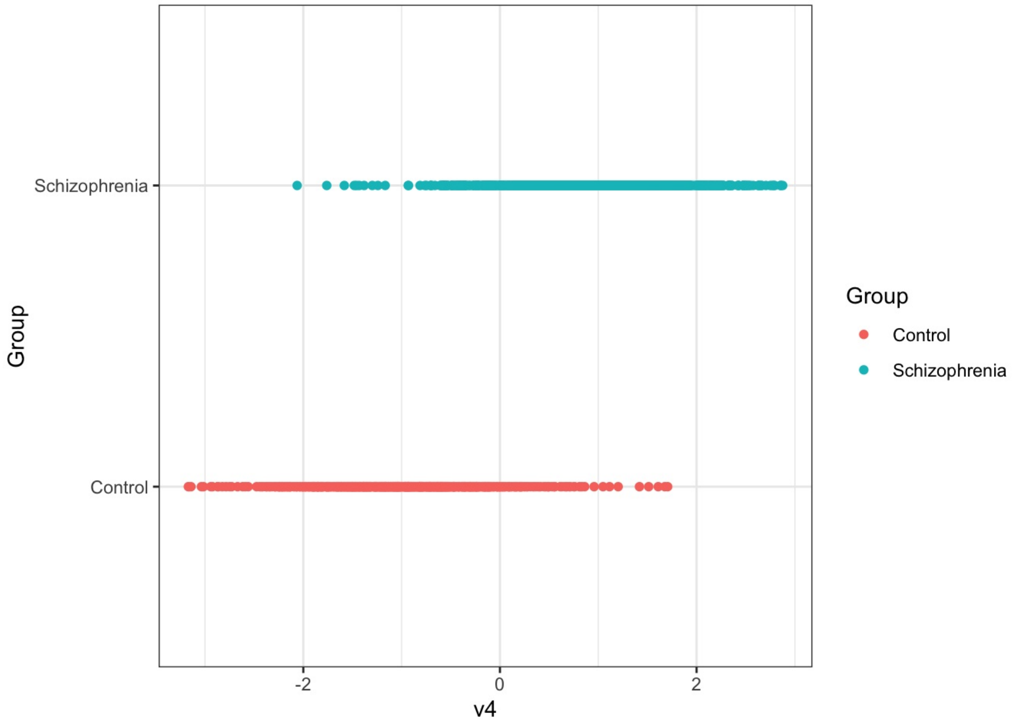
```
#Population size
n <- 100
trials <- 10

#Acoustic measures:
InformedEffectMean <- c(-1.26, -0.55, -0.75, 1.89, 0.25, 0.05, 0, 0, 0, 0)
SkepticEffectMean <- rep(0, 10)

#Individual variability from population, across trials and measurement error.
IndividualSD <- 1
TrialSD <- 0.5
Error <- 0.2
```

Generating tibble:

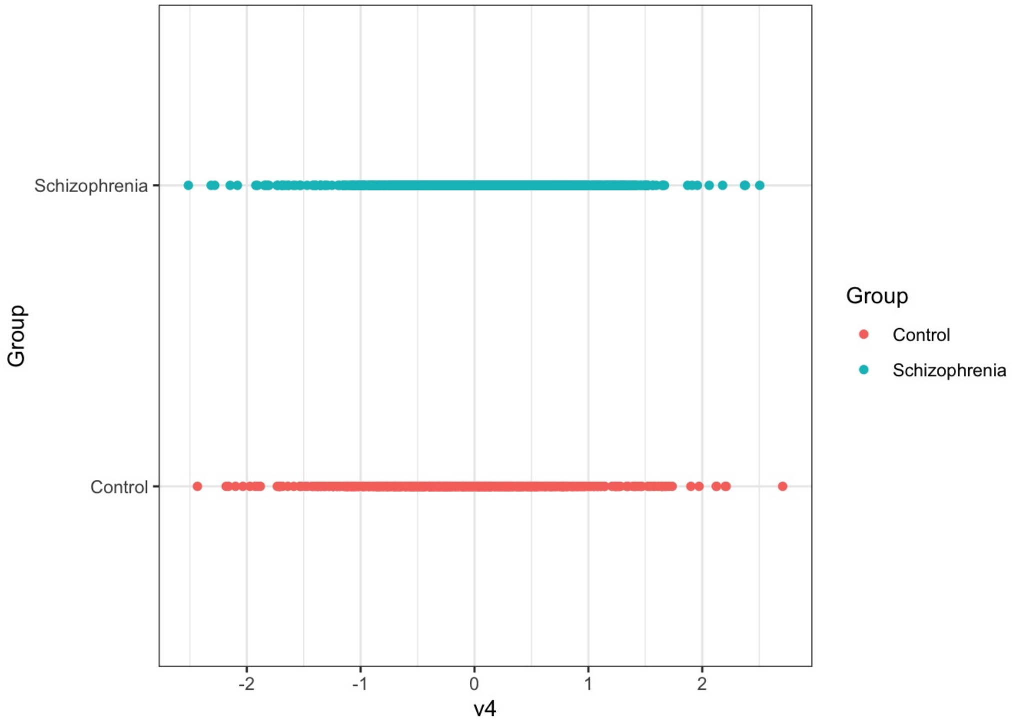
```
#For each pair of participants we need to identify the true effect size for each variable. Generating the effect,
the true difference for each pair for each variable.
for (i in seq(10)) {
  temp_informed <- tibble( #Informed data set
    ID = seq(n),
    TrueEffect = rnorm(n, InformedEffectMean[i], IndividualSD),
    Variable = paste0("v", i))

  temp_skeptic <- tibble( #Controlled data set
    ID = seq(n),
    TrueEffect = rnorm(n, SkepticEffectMean[i], IndividualSD),
    Variable = paste0("v", i))
  if (i == 1) {
    d_informed_true <- temp_informed
    d_skeptic_true <- temp_skeptic
  } else {
    d_informed_true <- rbind(d_informed_true, temp_informed )
    d_skeptic_true <- rbind(d_skeptic_true, temp_skeptic)
  }
}
```

Creating a tibble with one row per trial:

```r
d_trial <- tibble(expand_grid(ID = seq(n), Trial = seq(trials), Group = c("Schizophrenia", "Control")))


d_informed <- merge(d_informed_true, d_trial) #add all true effects by variables.
d_skeptic <- merge(d_skeptic_true, d_trial)

for (i in seq(nrow(d_informed))) { #loop each row, identify the measurement.
  d_informed$measurement[i] <- ifelse(d_informed$Group[i] == "Schizophrenia",
                                    rnorm(1, rnorm(1, d_informed$TrueEffect[i]/2, TrialSD), Error),
                                    rnorm(1, rnorm(1, (-d_informed$TrueEffect[i])/2, TrialSD), Error))
  d_skeptic$measurement[i] <- ifelse(d_skeptic$Group[i] == "Schizophrenia",
                                    rnorm(1, rnorm(1, d_skeptic$TrueEffect[i]/2, TrialSD), Error),
                                    rnorm(1, rnorm(1, (-d_skeptic$TrueEffect[i])/2, TrialSD), Error))
}

#Per each trial/speech recording we have one value of each variable.
d_informed_wide <- d_informed %>%
  mutate(TrueEffect = NULL) %>%
  pivot_wider(names_from = Variable, values_from = measurement)

d_skeptic_wide <- d_skeptic %>%
  mutate(TrueEffect = NULL) %>%
  pivot_wider(names_from = Variable, values_from = measurement)

ggplot(d_informed_wide, aes(x = v4, y = Group, color = Group)) + geom_point() + theme_bw() #plotted pitch ranges
of each group.
```




```r
ggplot(d_skeptic_wide, aes(x = v4, y = Group, color = Group)) + geom_point() + theme_bw()
```



Visualizing each of the variables from the two data sets:

```
plot_informed_vars <- ggplot(d_informed, aes(x = measurement, color = Group, fill = Group)) +
  geom_density(alpha = 0.5) +
  facet_wrap(~Variable) +
  ggtitle("Plot of informed data")

plot_skeptic_vars <- ggplot(d_skeptic, aes(x = measurement, color = Group, fill = Group)) +
  geom_density(alpha = 0.5) +
  facet_wrap(~Variable) +
  ggtitle("Plot of skeptic data")
plot_informed_vars
plot_skeptic_vars
```

Part II - ML pipeline on simulated data

On the two simulated datasets (separately) build a machine learning pipeline: i) create a data budget (e.g. balanced training and test sets); ii) pre-process the data (e.g. scaling the features); iii) fit and assess a classification algorithm on the training data (e.g. Bayesian multilevel logistic regression); iv) assess performance on the test set; v) discuss whether performance is as expected and feature importance is as expected.

Data budgeting:

```
TestID <- sample(seq(n), 20) #sampling 20 pairs from a sequence of 1 to 100

train_informed <- d_informed_wide %>% #train data set is the subset where the pair ID is not in the test set.
  subset(!(ID %in% TestID))

test_informed <- d_informed_wide %>%
  subset(ID %in% TestID) # test set is the one that has id in the test set.

#-----------------------------------------------------------------
train_skeptic <- d_skeptic_wide %>% #train data set is the subset where the pair ID is not in the test set.
  subset(!(ID %in% TestID))

test_skeptic <- d_skeptic_wide %>%
  subset(ID %in% TestID)
```

Data pre-processing (TIDYMODELS):

```
rec_informed <- train_informed %>%
  recipe(Group ~ . ) %>% # defines the outcome
  step_scale(all_numeric() ) %>% # scales numeric predictors
  step_center(all_numeric() ) %>% # center numeric predictors
  prep(training = train_informed, retain = TRUE)

rec_skeptic <- train_skeptic %>%
  recipe(Group ~ . ) %>% # defines the outcome
  step_scale(all_numeric() ) %>% # scales numeric predictors
  step_center(all_numeric() ) %>% # center numeric predictors
  prep(training = train_skeptic, retain = TRUE)


#Apply recipe to train and test
train_informed_s <- juice(rec_informed)
test_informed_s <- bake(rec_informed, new_data = test_informed, all_predictors()) %>%
  mutate(Group = test_informed$Group)

train_skeptic_s <- juice(rec_skeptic)
test_skeptic_s <- bake(rec_skeptic, new_data = test_skeptic, all_predictors()) %>%
  mutate(Group = test_skeptic$Group)
```

Setting up the models:

```
#Fixed effects:
p_range_f_1 <- bf(Group ~ 1 + v2 + v1 + v3 + v4 + v5 + v6 + v7 + v8 + v9 + v10)
get_prior(p_range_f_1, train_informed_s, family = bernoulli)
```

```
##                 prior     class coef group resp dpar nlpar lb ub       source
##                (flat)         b                                       default
##                (flat)         b   v1                             (vectorized)
##                (flat)         b  v10                             (vectorized)
##                (flat)         b   v2                             (vectorized)
##                (flat)         b   v3                             (vectorized)
##                (flat)         b   v4                             (vectorized)
##                (flat)         b   v5                             (vectorized)
##                (flat)         b   v6                             (vectorized)
##                (flat)         b   v7                             (vectorized)
##                (flat)         b   v8                             (vectorized)
##                (flat)         b   v9                             (vectorized)
##  student_t(3, 0, 2.5) Intercept                                       default
```

```
#Priors for fixed effect model:
p_range_p <- c(
  prior(normal(0, 1), class = Intercept),
  prior(normal(0, 0.3), class = b)
)

#Varying intercepts:
p_range_f_2 <- bf(Group ~ 1 + v2 + v1 + v3 + v4 + v5 + v6 + v7 + v8 + v9 + v10 + (1|ID))
get_prior(p_range_f_2, train_informed_s, family = bernoulli)
```

```
##                 prior     class      coef group resp dpar nlpar lb ub
##                (flat)         b
##                (flat)         b        v1
##                (flat)         b       v10
##                (flat)         b        v2
##                (flat)         b        v3
##                (flat)         b        v4
##                (flat)         b        v5
##                (flat)         b        v6
##                (flat)         b        v7
##                (flat)         b        v8
##                (flat)         b        v9
##  student_t(3, 0, 2.5) Intercept
##  student_t(3, 0, 2.5)        sd                                  0
##  student_t(3, 0, 2.5)        sd              ID                  0
##  student_t(3, 0, 2.5)        sd Intercept    ID                  0
##        source
##       default
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##       default
##       default
##  (vectorized)
##  (vectorized)
```

```
#Priors for varying intercept models:
p_range_p_2 <- c(
  prior(normal(0, 1), class = Intercept),
  prior(normal(0, 0.3), class = b),
  prior(normal(0, 0.3), class = sd)
)

#Varying slopes with priors specified above
p_range_f_3 <- bf(Group ~ 1 + v2 + v1 + v3 + v4 + v5 + v6 + v7 + v8 + v9 + v10 + (1 + v2 + v1 + v3 + v4 + v5 + v6
+ v7 + v8 + v9 + v10|ID))
get_prior(p_range_f_3, train_informed_s, family = bernoulli)
```

```
##                 prior     class      coef group resp dpar nlpar lb ub
##                (flat)         b                                      
##                (flat)         b        v1                            
##                (flat)         b       v10                            
##                (flat)         b        v2                            
##                (flat)         b        v3                            
##                (flat)         b        v4                            
##                (flat)         b        v5                            
##                (flat)         b        v6                            
##                (flat)         b        v7                            
##                (flat)         b        v8                            
##                (flat)         b        v9                            
##                lkj(1)       cor                                      
##                lkj(1)       cor              ID                      
##  student_t(3, 0, 2.5) Intercept                                      
##  student_t(3, 0, 2.5)        sd                                  0   
##  student_t(3, 0, 2.5)        sd              ID                  0   
##  student_t(3, 0, 2.5)        sd Intercept    ID                  0   
##  student_t(3, 0, 2.5)        sd        v1    ID                  0   
##  student_t(3, 0, 2.5)        sd       v10    ID                  0   
##  student_t(3, 0, 2.5)        sd        v2    ID                  0   
##  student_t(3, 0, 2.5)        sd        v3    ID                  0   
##  student_t(3, 0, 2.5)        sd        v4    ID                  0   
##  student_t(3, 0, 2.5)        sd        v5    ID                  0   
##  student_t(3, 0, 2.5)        sd        v6    ID                  0   
##  student_t(3, 0, 2.5)        sd        v7    ID                  0   
##  student_t(3, 0, 2.5)        sd        v8    ID                  0   
##  student_t(3, 0, 2.5)        sd        v9    ID                  0   
##        source
##       default
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##       default
##  (vectorized)
##       default
##       default
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
```

Informed models

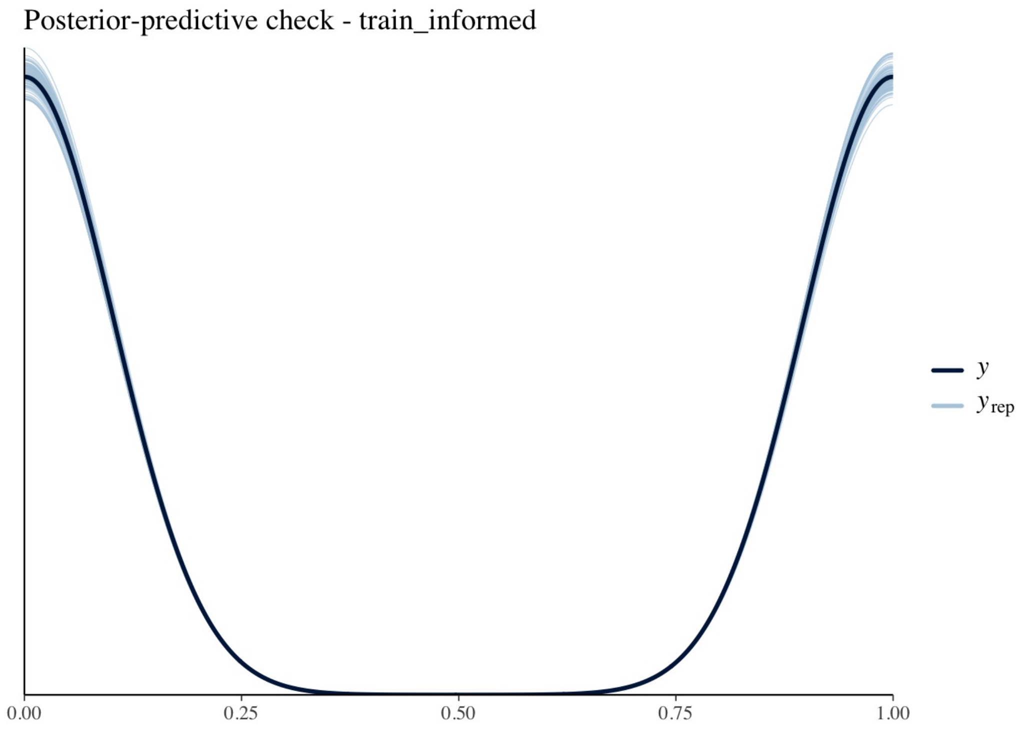
Informed: fixed effects

```
pitch_m_informed <- brm(
  p_range_f_1,
  data = train_informed_s,
  family = bernoulli,
  prior = p_range_p,
  sample_prior = T,
  backend = "cmdstanr",
  threads = threading(2),
  chains = 2,
  core = 2,
  control = list(adapt_delt = 0.99, max_treedepth = 20))
```

```
## Start sampling
```

```
## Running MCMC with 2 parallel chains, with 2 thread(s) per chain...
##
## Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 1 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 2 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 1 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 1 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 2 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 1 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 1 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 2 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 2 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 2 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 2 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 2 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 1 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 2 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 1 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 1 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 1 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1 finished in 0.9 seconds.
## Chain 2 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 2 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 2 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2 finished in 1.1 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 1.0 seconds.
## Total execution time: 1.3 seconds.
```

```
pp_check(pitch_m_informed, ndraws=100) + labs(title = "Posterior-predictive check - train_informed")
```

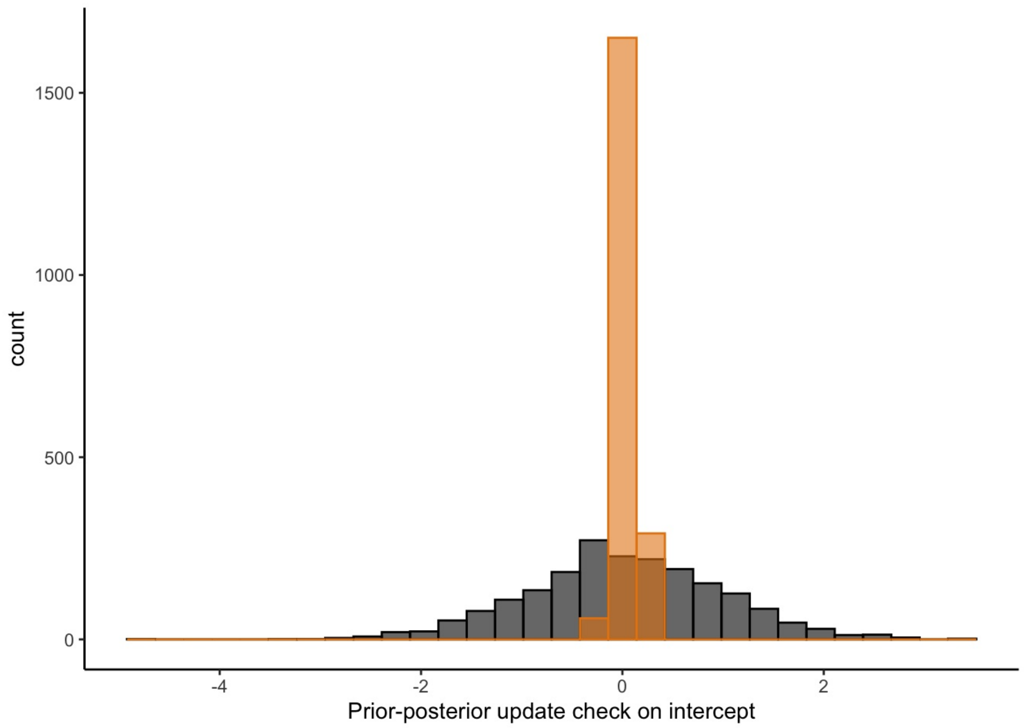



```
informed_draws <- as_draws_df(pitch_m_informed)
#variables(informed_draws)
```

Prior-predictive checks: Fixed effect model

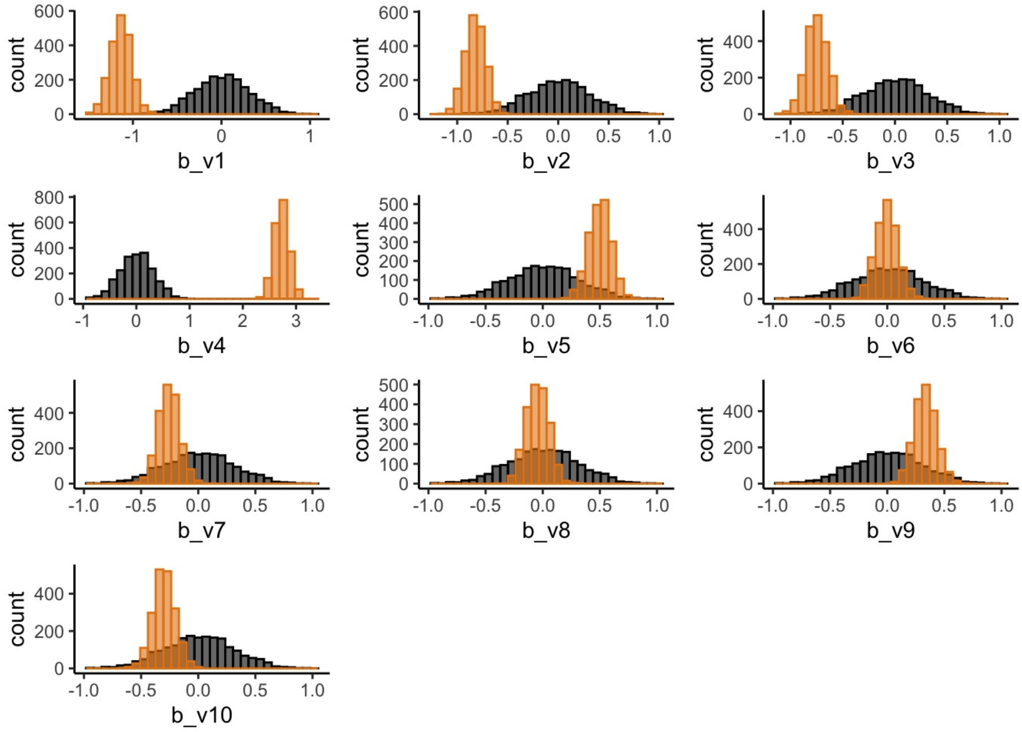
```
informed_pp_intercept <- ggplot(informed_draws) +
  geom_histogram(aes(prior_Intercept), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_Intercept), fill="#E68613", color="#E68613",alpha=0.6) + #Posterior
  theme_classic() +
   xlab("Prior-posterior update check on intercept")
informed_pp_intercept
```

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



```r
informed_pp_b_v2 <- ggplot(informed_draws) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v2), fill="#E68613", color="#E68613",alpha=0.6) + #Posterior of v2
  theme_classic() +
   xlab("b_v2")

informed_pp_b_v1 <- ggplot(informed_draws) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v1), fill="#E68613", color="#E68613",alpha=0.6) + #Posterior of v2
  theme_classic() +
   xlab("b_v1")

informed_pp_b_v3 <- ggplot(informed_draws) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v3), fill="#E68613", color="#E68613",alpha=0.6) + #Posterior of v2
  theme_classic() +
   xlab("b_v3")

informed_pp_b_v4 <- ggplot(informed_draws) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v4), fill="#E68613", color="#E68613",alpha=0.6) + #Posterior of v2
  theme_classic() +
   xlab("b_v4")

informed_pp_b_v5 <- ggplot(informed_draws) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v5), fill="#E68613", color="#E68613",alpha=0.6) + #Posterior of v2
  theme_classic() +
   xlab("b_v5")

informed_pp_b_v6 <- ggplot(informed_draws) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v6), fill="#E68613", color="#E68613",alpha=0.6) + #Posterior of v2
  theme_classic() +
   xlab("b_v6")

informed_pp_b_v7 <- ggplot(informed_draws) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v7), fill="#E68613", color="#E68613",alpha=0.6) + #Posterior of v2
  theme_classic() +
   xlab("b_v7")

informed_pp_b_v8 <- ggplot(informed_draws) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v8), fill="#E68613", color="#E68613",alpha=0.6) + #Posterior of v2
  theme_classic() +
   xlab("b_v8")

informed_pp_b_v9 <- ggplot(informed_draws) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v9), fill="#E68613", color="#E68613",alpha=0.6) + #Posterior of v2
  theme_classic() +
   xlab("b_v9")

informed_pp_b_v10 <- ggplot(informed_draws) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v10), fill="#E68613", color="#E68613",alpha=0.6) + #Posterior of v2
  theme_classic() +
   xlab("b_v10")

grid.arrange(informed_pp_b_v1, informed_pp_b_v2, informed_pp_b_v3, informed_pp_b_v4, informed_pp_b_v5, informed_p
p_b_v6, informed_pp_b_v7, informed_pp_b_v8, informed_pp_b_v9, informed_pp_b_v10)
```

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




```
#ggsave("prior-post.jpeg", plot = p1_10, path = "/Users/justina/Desktop/Desktop - Justina’s MacBook Pro/Aarhus_Un
i/Semester_3/Methods 3/Assignment-3")
```

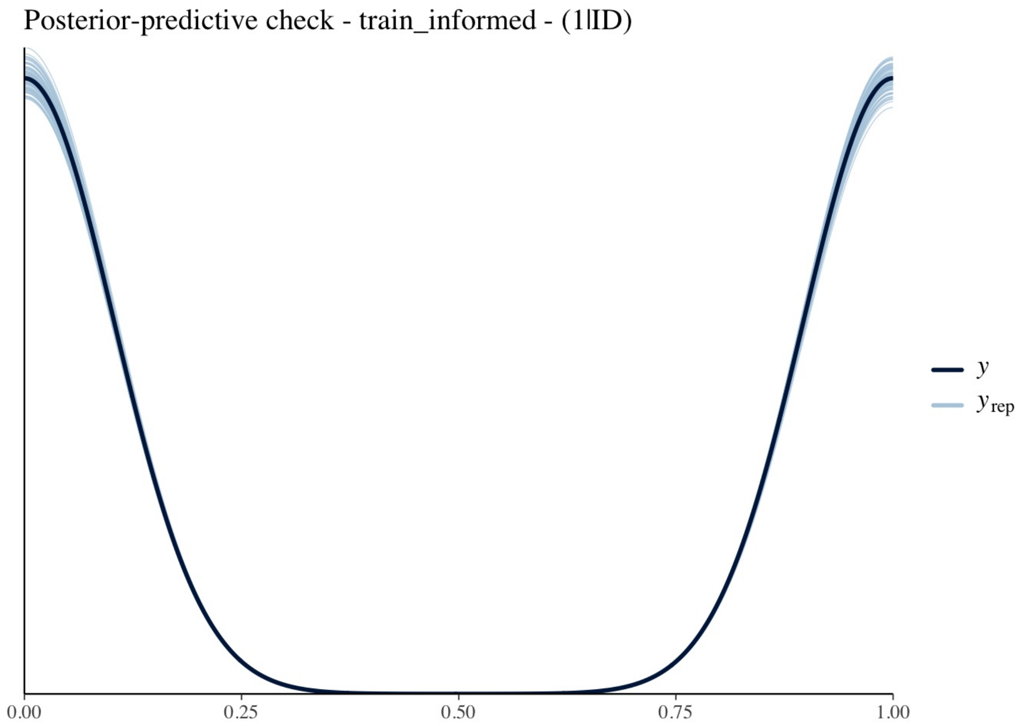
Informed: varying intercepts

```
pitch_m_informed_3 <- brm(
  p_range_f_2,
  data = train_informed_s,
  family = bernoulli,
  prior = p_range_p,
  sample_prior = T,
  backend = "cmdstanr",
  threads = threading(2),
  chains = 2,
  core = 2,
  control = list(adapt_delt = 0.99, max_treedepth = 20))
```

```
## Start sampling
```

```
## Running MCMC with 2 parallel chains, with 2 thread(s) per chain...
##
## Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 2 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 1 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 2 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 1 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 2 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 1 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 2 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 1 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 2 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 1 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 2 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 1 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 2 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 1 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 2 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 1 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 2 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 1 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 2 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1 finished in 5.5 seconds.
## Chain 2 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2 finished in 5.5 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 5.5 seconds.
## Total execution time: 5.6 seconds.
```

```
pp_check(pitch_m_informed_3, ndraws=100) + labs(title = "Posterior-predictive check - train_informed - (1|ID)")
```




```
informed_draws_2 <- as_draws_df(pitch_m_informed_3)
#colnames(informed_draws_2)
```

Prior-predictive checks: Varying intercept model

```
informed_pp_sd_2 <- ggplot(informed_draws_2) +
  geom_histogram(aes(prior_sd_ID), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(sd_ID__Intercept), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("Prior-posterior update check on intercept")


informed_pp_intercept_2 <- ggplot(informed_draws_2) +
  geom_histogram(aes(prior_Intercept), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_Intercept), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("Prior-posterior update check on intercept")


informed_pp_b_v2_2 <- ggplot(informed_draws_2) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v2), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("b_v2")

informed_pp_b_v1_2 <- ggplot(informed_draws_2) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v1), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("b_v1")

informed_pp_b_v3_2 <- ggplot(informed_draws_2) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v3), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("b_v3")

informed_pp_b_v4_2 <- ggplot(informed_draws_2) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v4), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("b_v4")

informed_pp_b_v5_2 <- ggplot(informed_draws_2) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v5), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("b_v5")

informed_pp_b_v6_2 <- ggplot(informed_draws_2) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
```

```
  geom_histogram(aes(b_v6), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("b_v6")

informed_pp_b_v7_2 <- ggplot(informed_draws_2) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v7), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("b_v7")

informed_pp_b_v8_2 <- ggplot(informed_draws_2) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v8), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("b_v8")

informed_pp_b_v9_2 <- ggplot(informed_draws_2) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v9), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("b_v9")

informed_pp_b_v10_2 <- ggplot(informed_draws_2) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v10), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("b_v10")

#p1_10_m2 <- grid.arrange(informed_pp_b_v1_2, informed_pp_b_v2_2, informed_pp_b_v3_2, informed_pp_b_v4_2, informe
d_pp_b_v5_2, informed_pp_b_v6_2, informed_pp_b_v7_2, informed_pp_b_v8_2, informed_pp_b_v9_2, informed_pp_b_v10_2)

#ggsave("prior-post-m2.jpeg", plot = p1_10_m2, path = "/Users/justina/Desktop/Desktop - Justina’s MacBook Pro/Aar
hus_Uni/Semester_3/Methods 3/Assignment-3")
```

Informed: random slopes

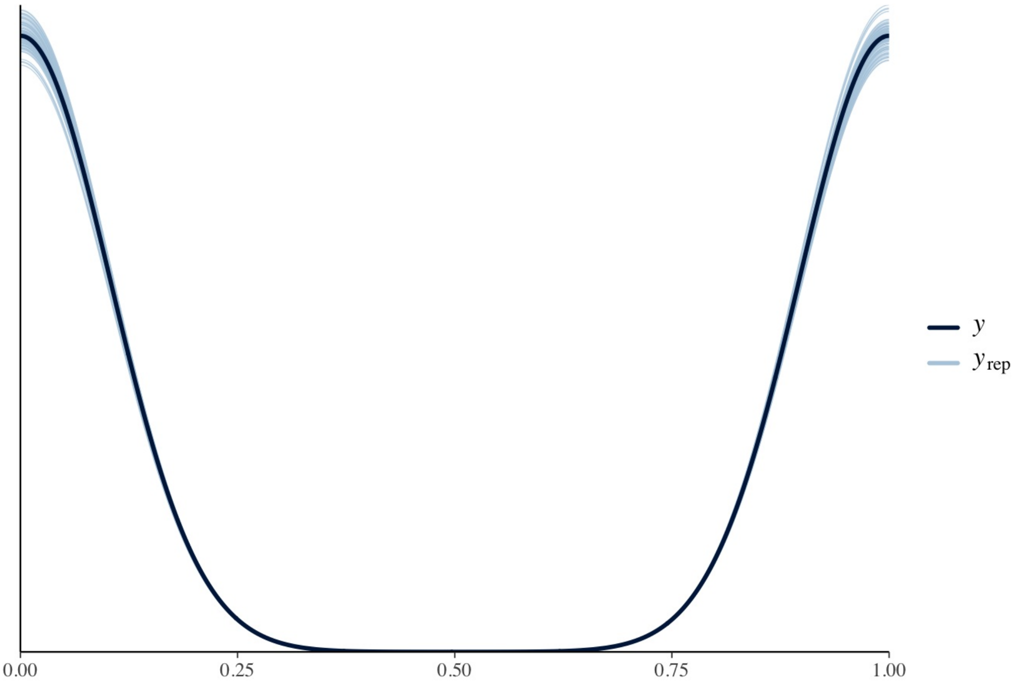
```
pitch_m_informed_2 <- brm(
  p_range_f_2,
  data = train_informed_s,
  family = bernoulli,
  prior = p_range_p,
  sample_prior = T,
  backend = "cmdstanr",
  threads = threading(2),
  chains = 2,
  core = 2,
  control = list(adapt_delt = 0.99, max_treedepth = 20))
```

```
## Start sampling
```

```
## Running MCMC with 2 parallel chains, with 2 thread(s) per chain...
##
## Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 2 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 2 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 2 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 1 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 2 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 1 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 2 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 1 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 2 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 1 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 2 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 1 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 2 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 1 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 2 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 1 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 2 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 1 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 2 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1 finished in 5.4 seconds.
## Chain 2 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2 finished in 5.5 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 5.5 seconds.
## Total execution time: 5.6 seconds.
```

```
pp_check(pitch_m_informed_2, ndraws=100) + labs(title = "Posterior-predictive check - train_informed - random slo
pes")
```

Posterior-predictive check - train_informed - random slopes

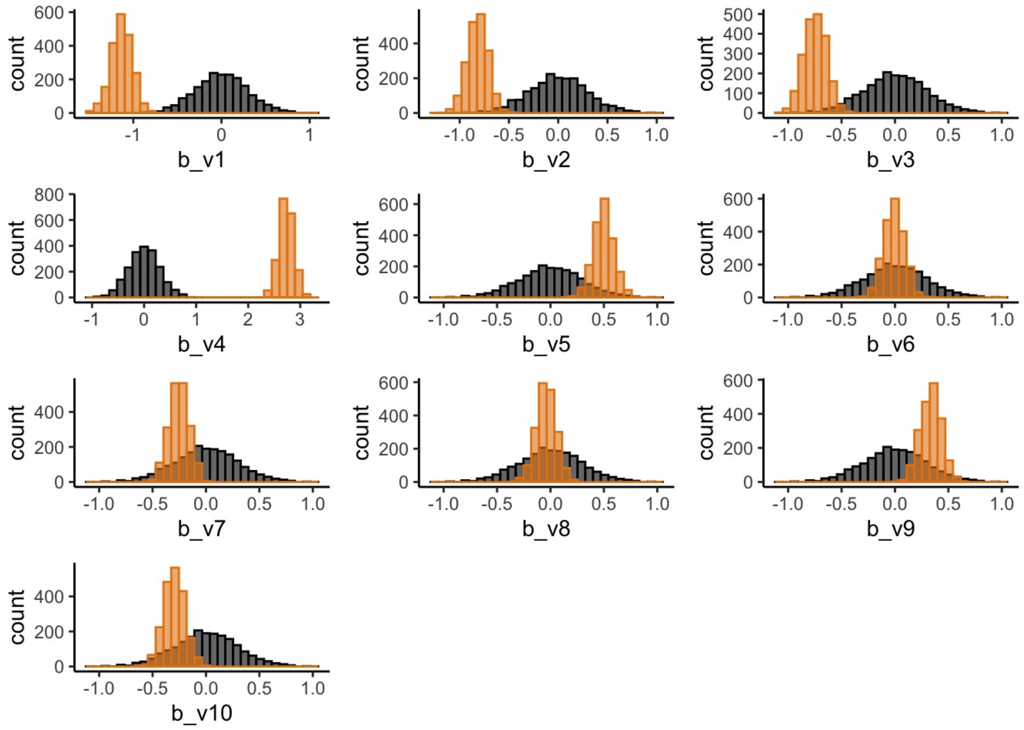



```
informed_draws_3 <- as_draws_df(pitch_m_informed_2)
#colnames(informed_draws_3)
```

Prior-predictive checks: Varying slope model

```
informed_pp_sd_3 <- ggplot(informed_draws_3) +
  geom_histogram(aes(prior_sd_ID), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(sd_ID__Intercept), fill="#E68613", color="#E68613",alpha=0.6) + #Posterior
  theme_classic() +
   xlab("Prior-posterior update check on intercept")


informed_pp_intercept_3 <- ggplot(informed_draws_3) +
  geom_histogram(aes(prior_Intercept), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_Intercept), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("Prior-posterior update check on intercept")


informed_pp_b_v2_3 <- ggplot(informed_draws_3) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v2), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("b_v2")

informed_pp_b_v1_3 <- ggplot(informed_draws_3) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v1), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("b_v1")

informed_pp_b_v3_3 <- ggplot(informed_draws_3) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v3), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("b_v3")

informed_pp_b_v4_3 <- ggplot(informed_draws_3) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v4), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("b_v4")

informed_pp_b_v5_3 <- ggplot(informed_draws_3) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v5), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("b_v5")

informed_pp_b_v6_3 <- ggplot(informed_draws_3) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v6), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("b_v6")

informed_pp_b_v7_3 <- ggplot(informed_draws_3) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v7), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("b_v7")

informed_pp_b_v8_3 <- ggplot(informed_draws_3) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v8), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("b_v8")

informed_pp_b_v9_3 <- ggplot(informed_draws_3) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v9), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("b_v9")

informed_pp_b_v10_3 <- ggplot(informed_draws_3) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v10), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("b_v10")

grid.arrange(informed_pp_b_v1_3, informed_pp_b_v2_3, informed_pp_b_v3_3, informed_pp_b_v4_3, informed_pp_b_v5_3,
informed_pp_b_v6_3, informed_pp_b_v7_3, informed_pp_b_v8_3, informed_pp_b_v9_3, informed_pp_b_v10_3)
```

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




```
#ggsave("prior-post-m3.jpeg", plot = p1_10_m3, path = "/Users/justina/Desktop/Desktop - Justina’s MacBook Pro/Aar
hus_Uni/Semester_3/Methods 3/Assignment-3")
```

Skeptic models

Skeptic: fixed effects

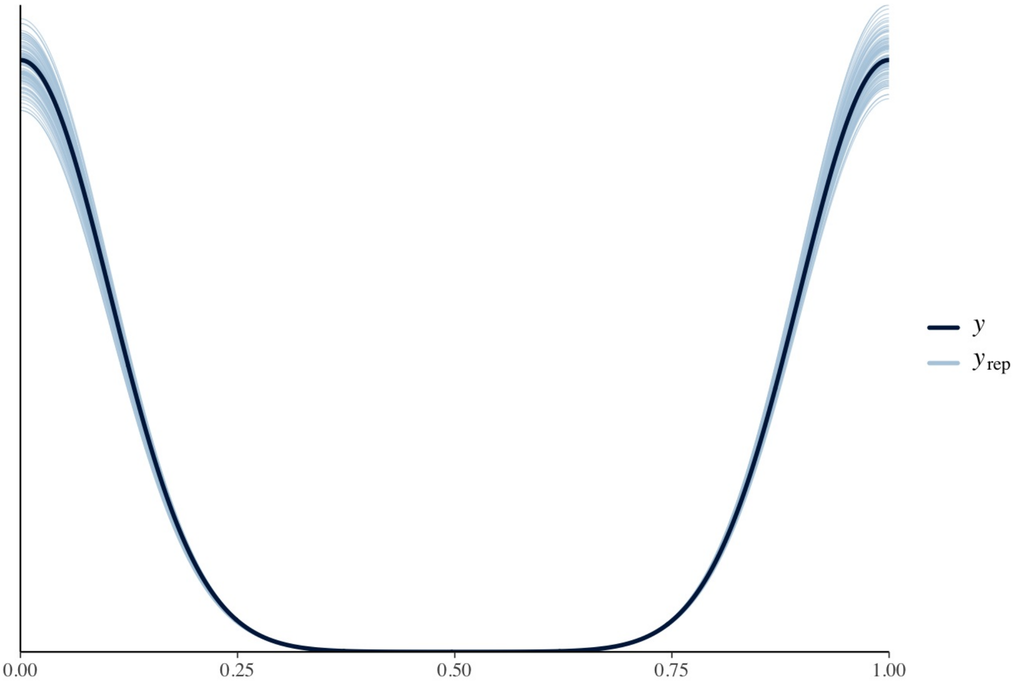
```
pitch_m_skeptic <- brm(
  p_range_f_1,
  data = train_skeptic_s,
  family = bernoulli,
  prior = p_range_p,
  sample_prior = T,
  backend = "cmdstanr",
  threads = threading(2),
  chains = 2,
  core = 2,
  control = list(adapt_delt = 0.99, max_treedepth = 20))
```

```
## Start sampling
```

```
## Running MCMC with 2 parallel chains, with 2 thread(s) per chain...
## 
## Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup) 
## Chain 1 Iteration:  100 / 2000 [  5%]  (Warmup) 
## Chain 1 Iteration:  200 / 2000 [ 10%]  (Warmup) 
## Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup) 
## Chain 2 Iteration:  100 / 2000 [  5%]  (Warmup) 
## Chain 1 Iteration:  300 / 2000 [ 15%]  (Warmup) 
## Chain 1 Iteration:  400 / 2000 [ 20%]  (Warmup) 
## Chain 2 Iteration:  200 / 2000 [ 10%]  (Warmup) 
## Chain 2 Iteration:  300 / 2000 [ 15%]  (Warmup) 
## Chain 2 Iteration:  400 / 2000 [ 20%]  (Warmup) 
## Chain 1 Iteration:  500 / 2000 [ 25%]  (Warmup) 
## Chain 1 Iteration:  600 / 2000 [ 30%]  (Warmup) 
## Chain 2 Iteration:  500 / 2000 [ 25%]  (Warmup) 
## Chain 1 Iteration:  700 / 2000 [ 35%]  (Warmup) 
## Chain 1 Iteration:  800 / 2000 [ 40%]  (Warmup) 
## Chain 1 Iteration:  900 / 2000 [ 45%]  (Warmup) 
## Chain 2 Iteration:  600 / 2000 [ 30%]  (Warmup) 
## Chain 2 Iteration:  700 / 2000 [ 35%]  (Warmup) 
## Chain 2 Iteration:  800 / 2000 [ 40%]  (Warmup) 
## Chain 1 Iteration: 1000 / 2000 [ 50%]  (Warmup) 
## Chain 1 Iteration: 1001 / 2000 [ 50%]  (Sampling) 
## Chain 1 Iteration: 1100 / 2000 [ 55%]  (Sampling) 
## Chain 2 Iteration:  900 / 2000 [ 45%]  (Warmup) 
## Chain 2 Iteration: 1000 / 2000 [ 50%]  (Warmup) 
## Chain 2 Iteration: 1001 / 2000 [ 50%]  (Sampling) 
## Chain 1 Iteration: 1200 / 2000 [ 60%]  (Sampling) 
## Chain 1 Iteration: 1300 / 2000 [ 65%]  (Sampling) 
## Chain 2 Iteration: 1100 / 2000 [ 55%]  (Sampling) 
## Chain 2 Iteration: 1200 / 2000 [ 60%]  (Sampling) 
## Chain 1 Iteration: 1400 / 2000 [ 70%]  (Sampling) 
## Chain 1 Iteration: 1500 / 2000 [ 75%]  (Sampling) 
## Chain 2 Iteration: 1300 / 2000 [ 65%]  (Sampling) 
## Chain 2 Iteration: 1400 / 2000 [ 70%]  (Sampling) 
## Chain 1 Iteration: 1600 / 2000 [ 80%]  (Sampling) 
## Chain 1 Iteration: 1700 / 2000 [ 85%]  (Sampling) 
## Chain 1 Iteration: 1800 / 2000 [ 90%]  (Sampling) 
## Chain 2 Iteration: 1500 / 2000 [ 75%]  (Sampling) 
## Chain 2 Iteration: 1600 / 2000 [ 80%]  (Sampling) 
## Chain 1 Iteration: 1900 / 2000 [ 95%]  (Sampling) 
## Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling) 
## Chain 2 Iteration: 1700 / 2000 [ 85%]  (Sampling) 
## Chain 2 Iteration: 1800 / 2000 [ 90%]  (Sampling) 
## Chain 1 finished in 1.0 seconds.
## Chain 2 Iteration: 1900 / 2000 [ 95%]  (Sampling) 
## Chain 2 Iteration: 2000 / 2000 [100%]  (Sampling) 
## Chain 2 finished in 1.1 seconds.
## 
## Both chains finished successfully.
## Mean chain execution time: 1.0 seconds.
## Total execution time: 1.2 seconds.
```

```
pp_check(pitch_m_skeptic, ndraws=100) + labs(title = "Posterior-predictive check - train_skeptic")
```

Posterior-predictive check - train_skeptic




```
skeptic_draws <- as_draws_df(pitch_m_skeptic)
#variables(informed_draws)

skeptic_pp_intercept <- ggplot(skeptic_draws) +
  geom_histogram(aes(prior_Intercept), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_Intercept), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("Prior-posterior update check on intercept")


skeptic_pp_b_v2 <- ggplot(skeptic_draws) +
  geom_histogram(aes(prior_b), fill="black", color="black",alpha=0.6,) +
  geom_histogram(aes(b_v2), fill="#E68613", color="#E68613",alpha=0.6) +
  theme_classic() +
   xlab("Prior-posterior update check on b_v2")
```

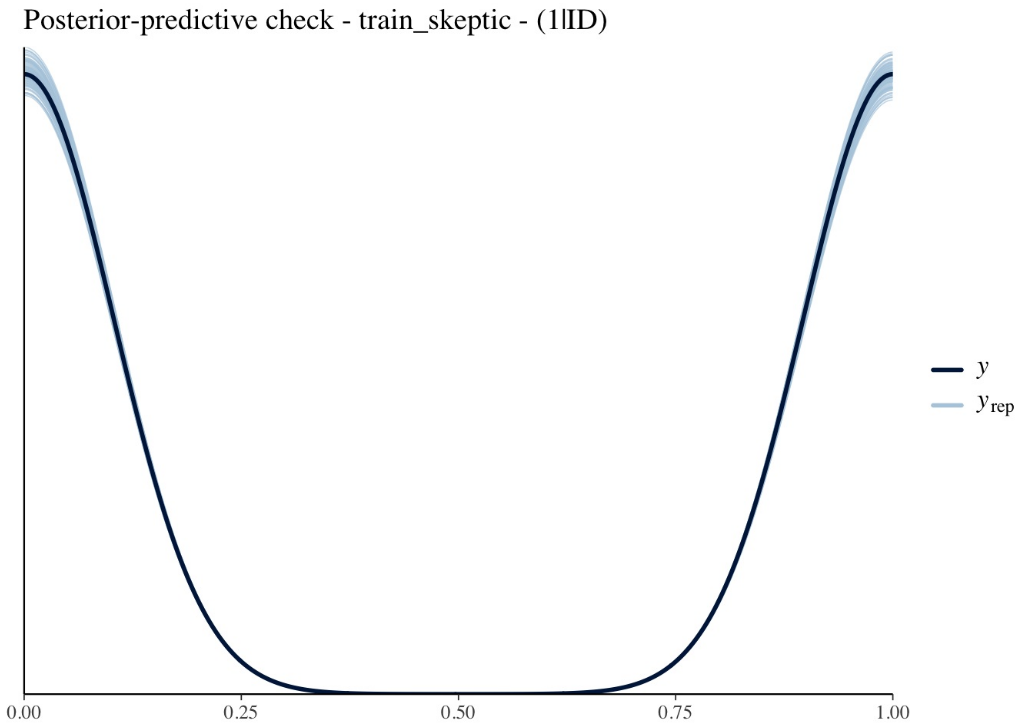
Skeptic: varying intercepts

```
pitch_m_skeptic_2 <- brm(
  p_range_f_2,
  data = train_skeptic_s,
  family = bernoulli,
  prior = p_range_p_2,
  sample_prior = T,
  backend = "cmdstanr",
  threads = threading(2),
  chains = 2,
  core = 2,
  control = list(adapt_delt = 0.99, max_treedepth = 20))
```

```
## Start sampling
```

```
## Running MCMC with 2 parallel chains, with 2 thread(s) per chain...
##
## Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 2 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 1 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 1 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 2 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 2 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 2 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 1 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 2 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 1 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 2 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 1 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 2 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 1 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 2 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 1 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 2 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 1 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 2 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1 finished in 6.3 seconds.
## Chain 2 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 2 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2 finished in 6.8 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 6.6 seconds.
## Total execution time: 6.9 seconds.
```

```
pp_check(pitch_m_informed_2, ndraws=100) + labs(title = "Posterior-predictive check - train_skeptic - (1|ID)")
```



Skeptic: varying slopes

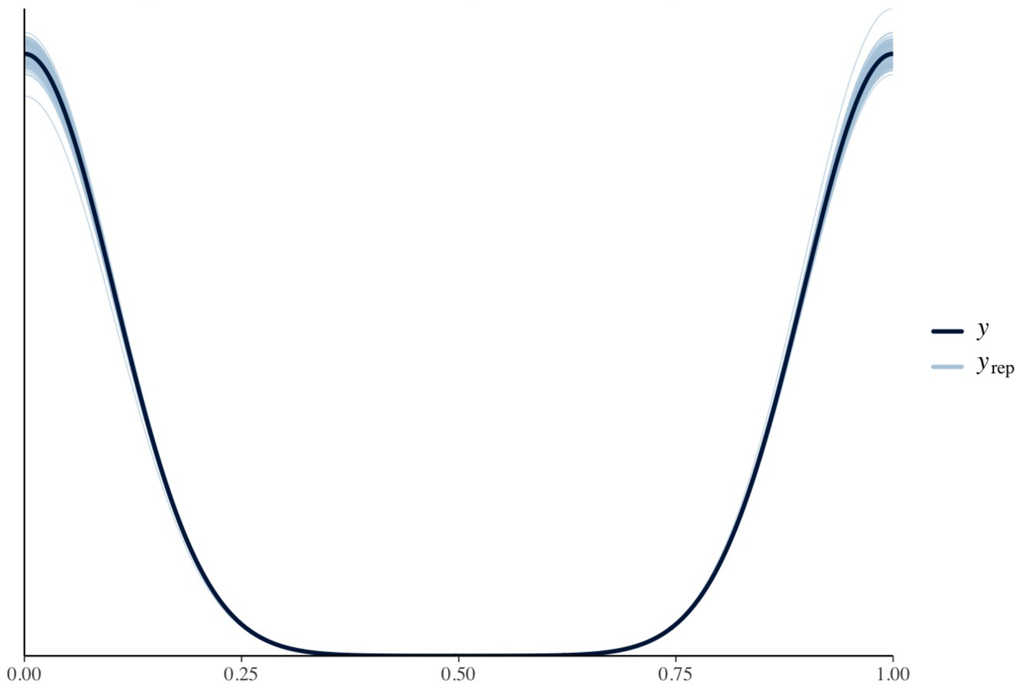
```
pitch_m_skeptic_3 <- brm(
  p_range_f_3,
  data = train_skeptic_s,
  family = bernoulli,
  prior = p_range_p_2,
  sample_prior = T,
  backend = "cmdstanr",
  threads = threading(2),
  chains = 2,
  core = 2,
  control = list(adapt_delt = 0.99, max_treedepth = 20))
```

```
## Start sampling
```

```
## Running MCMC with 2 parallel chains, with 2 thread(s) per chain...
##
## Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 2 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 1 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 2 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 1 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 2 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 1 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 2 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 1 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 2 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 1 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 2 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 1 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 2 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 1 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 2 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 1 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 2 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 1 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 2 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1 finished in 118.8 seconds.
## Chain 2 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 2 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2 finished in 126.4 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 122.6 seconds.
## Total execution time: 126.5 seconds.
```

```
pp_check(pitch_m_informed_3, ndraws=100) + labs(title = "Posterior-predictive check - train_skeptic - random slop
es")
```

Posterior-predictive check - train_skeptic - random slopes



Calculating performance of the models: INFORMED

```
#Informed
train_informed_s$PredictionsPerc0 <- predict(pitch_m_informed)[, 1]
train_informed_s$Predictions0[train_informed_s$PredictionsPerc0 > 0.5] <- "Schizophrenia"
```

```
## Warning: Unknown or uninitialised column: `Predictions0`.
```

```
train_informed_s$Predictions0[train_informed_s$PredictionsPerc0 <= 0.5] <- "Control"

train_informed_s$PredictionsPerc1 <- predict(pitch_m_informed_3)[, 1]
train_informed_s$Predictions1[train_informed_s$PredictionsPerc1 > 0.5] <- "Schizophrenia"
```

```
## Warning: Unknown or uninitialised column: `Predictions1`.
```

```
train_informed_s$Predictions1[train_informed_s$PredictionsPerc1 <= 0.5] <- "Control"

train_informed_s$PredictionsPerc2 <- predict(pitch_m_informed_2)[, 1]
train_informed_s$Predictions2[train_informed_s$PredictionsPerc2 > 0.5] <- "Schizophrenia"
```

```
## Warning: Unknown or uninitialised column: `Predictions2`.
```

```
train_informed_s$Predictions2[train_informed_s$PredictionsPerc2 <= 0.5] <- "Control"

train_informed_s <- train_informed_s %>%
  mutate(
    Group  = as.factor(Group),
    Predictions0 = as.factor(Predictions0),
    Predictions1 = as.factor(Predictions1),
    Predictions2 = as.factor(Predictions2)
  )

test_informed_s$PredictionsPerc0 <- predict(pitch_m_informed, newdata = test_informed_s, allow_new_levels = T)[,
1]
test_informed_s$Predictions0[test_informed_s$PredictionsPerc0 > 0.5] <- "Schizophrenia"
```

```
## Warning: Unknown or uninitialised column: `Predictions0`.
```

```
test_informed_s$Predictions0[test_informed_s$PredictionsPerc0 <= 0.5] <- "Control"

test_informed_s$PredictionsPerc1 <- predict(pitch_m_informed_3, newdata = test_informed_s, allow_new_levels = T)[
, 1]
test_informed_s$Predictions1[test_informed_s$PredictionsPerc1 > 0.5] <- "Schizophrenia"
```

```
## Warning: Unknown or uninitialised column: `Predictions1`.
```

```
test_informed_s$Predictions1[test_informed_s$PredictionsPerc1 <= 0.5] <- "Control"

test_informed_s$PredictionsPerc2 <- predict(pitch_m_informed_2, newdata = test_informed_s, allow_new_levels = T)[
, 1]
test_informed_s$Predictions2[test_informed_s$PredictionsPerc2 > 0.5] <- "Schizophrenia"
```

```
## Warning: Unknown or uninitialised column: `Predictions2`.
```

```
test_informed_s$Predictions2[test_informed_s$PredictionsPerc2 <= 0.5] <- "Control"

test_informed_s <- test_informed_s %>%
  mutate(
    Group  = as.factor(Group),
    Predictions0 = as.factor(Predictions0),
    Predictions1 = as.factor(Predictions1),
    Predictions2 = as.factor(Predictions2)
  )
```

SKEPTIC

```
train_skeptic_s$PredictionsPerc0 <- predict(pitch_m_skeptic)[, 1]
train_skeptic_s$Predictions0[train_skeptic_s$PredictionsPerc0 > 0.5] <- "Schizophrenia"
```

```
## Warning: Unknown or uninitialised column: `Predictions0`.
```

```
train_skeptic_s$Predictions0[train_skeptic_s$PredictionsPerc0 <= 0.5] <- "Control"

train_skeptic_s$PredictionsPerc1 <- predict(pitch_m_skeptic_2)[, 1]
train_skeptic_s$Predictions1[train_skeptic_s$PredictionsPerc1 > 0.5] <- "Schizophrenia"
```

```
## Warning: Unknown or uninitialised column: `Predictions1`.
```

```
train_skeptic_s$Predictions1[train_skeptic_s$PredictionsPerc1 <= 0.5] <- "Control"

train_skeptic_s$PredictionsPerc2 <- predict(pitch_m_skeptic_3)[, 1]
train_skeptic_s$Predictions2[train_skeptic_s$PredictionsPerc2 > 0.5] <- "Schizophrenia"
```

```
## Warning: Unknown or uninitialised column: `Predictions2`.
```

```
train_skeptic_s$Predictions2[train_skeptic_s$PredictionsPerc2 <= 0.5] <- "Control"

train_skeptic_s <- train_skeptic_s %>%
  mutate(
    Group  = as.factor(Group),
    Predictions0 = as.factor(Predictions0),
    Predictions1 = as.factor(Predictions1),
    Predictions2 = as.factor(Predictions2)
  )

test_skeptic_s$PredictionsPerc0 <- predict(pitch_m_skeptic, newdata = test_skeptic_s, allow_new_levels = T)[, 1]
test_skeptic_s$Predictions0[test_skeptic_s$PredictionsPerc0 > 0.5] <- "Schizophrenia"
```

```
## Warning: Unknown or uninitialised column: `Predictions0`.
```

```
test_skeptic_s$Predictions0[test_skeptic_s$PredictionsPerc0 <= 0.5] <- "Control"

test_skeptic_s$PredictionsPerc1 <- predict(pitch_m_skeptic_2, newdata = test_skeptic_s, allow_new_levels = T)[, 1
]
test_skeptic_s$Predictions1[test_skeptic_s$PredictionsPerc1 > 0.5] <- "Schizophrenia"
```

```
## Warning: Unknown or uninitialised column: `Predictions1`.
```

```
test_skeptic_s$Predictions1[test_skeptic_s$PredictionsPerc1 <= 0.5] <- "Control"

test_skeptic_s$PredictionsPerc2 <- predict(pitch_m_skeptic_3, newdata = test_skeptic_s, allow_new_levels = T)[, 1
]
test_skeptic_s$Predictions2[test_skeptic_s$PredictionsPerc2 > 0.5] <- "Schizophrenia"
```

```
## Warning: Unknown or uninitialised column: `Predictions2`.
```

```
test_skeptic_s$Predictions2[test_skeptic_s$PredictionsPerc2 <= 0.5] <- "Control"

test_skeptic_s <- test_skeptic_s %>%
  mutate(
    Group  = as.factor(Group),
    Predictions0 = as.factor(Predictions0),
    Predictions1 = as.factor(Predictions1),
    Predictions2 = as.factor(Predictions2)
  )
```

Assessing average performance:

```
#INFORMED: TRAINING DATA
conf_mat(train_informed_s, truth = Group, estimate = Predictions0, dnn = c("Prediction", "Truth"))
```

```
##                Truth
## Prediction      Control Schizophrenia
##   Control           756            45
##   Schizophrenia      44           755
```

```
metrics(train_informed_s, truth = Group, estimate = Predictions0) %>%
  knitr::kable()
```

.metric	.estimator	.estimate
accuracy	binary	0.944375
kap	binary	0.888750

```
conf_mat(train_informed_s, truth = Group, estimate = Predictions1, dnn = c("Prediction", "Truth"))
```

```
##                Truth
## Prediction      Control Schizophrenia
##   Control           756            44
##   Schizophrenia      44           756
```

```
metrics(train_informed_s, truth = Group, estimate = Predictions1) %>%
  knitr::kable()
```

.metric	.estimator	.estimate
accuracy	binary	0.945
kap	binary	0.890

```
conf_mat(train_informed_s, truth = Group, estimate = Predictions2, dnn = c("Prediction", "Truth"))
```

```
##                Truth
## Prediction      Control Schizophrenia
##   Control           756            45
##   Schizophrenia      44           755
```

```
metrics(train_informed_s, truth = Group, estimate = Predictions2) %>%
  knitr::kable()
```

.metric	.estimator	.estimate
accuracy	binary	0.944375
kap	binary	0.888750

```
#INFORMED: TEST DATA
conf_mat(test_informed_s, truth = Group, estimate = Predictions0, dnn = c("Prediction", "Truth"))
```

```
##                Truth
## Prediction      Control Schizophrenia
##   Control           188             9
##   Schizophrenia      12           191
```

```
metrics(test_informed_s, truth = Group, estimate = Predictions0) %>%
  knitr::kable()
```

.metric	.estimator	.estimate
accuracy	binary	0.9475
kap	binary	0.8950

```
conf_mat(test_informed_s, truth = Group, estimate = Predictions1, dnn = c("Prediction", "Truth"))
```

```
##                Truth
## Prediction      Control Schizophrenia
##   Control           188             9
##   Schizophrenia      12           191
```

```
metrics(test_informed_s, truth = Group, estimate = Predictions1) %>%
  knitr::kable()
```

.metric	.estimator	.estimate
accuracy	binary	0.9475
kap	binary	0.8950

```
conf_mat(test_informed_s, truth = Group, estimate = Predictions2, dnn = c("Prediction", "Truth"))
```

```
##                Truth
## Prediction      Control Schizophrenia
##   Control           189             8
##   Schizophrenia      11           192
```

```
metrics(test_informed_s, truth = Group, estimate = Predictions2) %>%
  knitr::kable()
```

.metric	.estimator	.estimate
accuracy	binary	0.9525
kap	binary	0.9050

```
#SKEPTIC: TRAINING DATA
conf_mat(train_skeptic_s, truth = Group, estimate = Predictions0, dnn = c("Prediction", "Truth"))
```

```
##                Truth
## Prediction      Control Schizophrenia
##   Control           491           334
##   Schizophrenia     309           466
```

```
metrics(train_skeptic_s, truth = Group, estimate = Predictions0) %>%
  knitr::kable()
```

.metric	.estimator	.estimate
accuracy	binary	0.598125
kap	binary	0.196250

```
conf_mat(train_skeptic_s, truth = Group, estimate = Predictions1, dnn = c("Prediction", "Truth"))
```

```
##                Truth
## Prediction      Control Schizophrenia
##   Control           491           332
##   Schizophrenia     309           468
```

```
metrics(train_skeptic_s, truth = Group, estimate = Predictions1) %>%
  knitr::kable()
```

.metric	.estimator	.estimate
accuracy	binary	0.599375
kap	binary	0.198750

```
conf_mat(train_skeptic_s, truth = Group, estimate = Predictions2, dnn = c("Prediction", "Truth"))
```

```
##                Truth
## Prediction      Control Schizophrenia
##   Control           795             2
##   Schizophrenia       5           798
```

```
metrics(train_skeptic_s, truth = Group, estimate = Predictions2) %>%
  knitr::kable()
```

.metric	.estimator	.estimate
accuracy	binary	0.995625
kap	binary	0.991250

```
#SKEPTIC: TEST DATA
conf_mat(test_skeptic_s, truth = Group, estimate = Predictions0, dnn = c("Prediction", "Truth"))
```

```
##                Truth
## Prediction      Control Schizophrenia
##   Control            95           112
##   Schizophrenia     105            88
```

```
metrics(test_skeptic_s, truth = Group, estimate = Predictions0) %>%
  knitr::kable()
```

.metric	.estimator	.estimate
accuracy	binary	0.4575
kap	binary	-0.0850

```
conf_mat(test_skeptic_s, truth = Group, estimate = Predictions1, dnn = c("Prediction", "Truth"))
```

```
##                Truth
## Prediction      Control Schizophrenia
##   Control           100           117
##   Schizophrenia     100            83
```

```
metrics(test_skeptic_s, truth = Group, estimate = Predictions1) %>%
  knitr::kable()
```

.metric	.estimator	.estimate
accuracy	binary	0.4575
kap	binary	-0.0850

```
conf_mat(test_skeptic_s, truth = Group, estimate = Predictions2, dnn = c("Prediction", "Truth"))
```

```
##                Truth
## Prediction      Control Schizophrenia
##   Control           107           107
##   Schizophrenia      93            93
```

```
metrics(test_skeptic_s, truth = Group, estimate = Predictions2) %>%
  knitr::kable()
```

.metric	.estimator	.estimate
accuracy	binary	0.5
kap	binary	0.0

```
PerformanceProb <- tibble(expand_grid(
  Sample = seq(2000),
  Model = c("FixedEffects", "VaryingIntercept", "VaryingSlope"),
  Setup = c("Informed", "Skeptic"),
  Type = c("Training", "Test")
))

#Informed: fixed effects
train0 <- inv_logit_scaled(posterior_linpred(pitch_m_informed, summary = F))
test0 <- inv_logit_scaled(posterior_linpred(pitch_m_informed, summary = F,
                                            newdata = test_informed_s, allow_new_levels = T))

for (i in seq(2000)) {
  train_informed_s$Predictions0 <- as.factor(ifelse(train0[i,]> 0.5, "Schizophrenia", "Control"))
  test_informed_s$Predictions0 <- as.factor(ifelse(test0[i,]> 0.5, "Schizophrenia", "Control"))

  PerformanceProb$Accuracy[PerformanceProb$Sample == i & PerformanceProb$Model == "FixedEffects" &
                             PerformanceProb$Setup == "Informed" & PerformanceProb$Type == "Training"] <- accurac
y(train_informed_s, truth = Group, estimate = Predictions0)[, ".estimate"]

    PerformanceProb$Accuracy[PerformanceProb$Sample == i & PerformanceProb$Model == "FixedEffects" &
                             PerformanceProb$Setup == "Informed" & PerformanceProb$Type == "Test"] <- accuracy(te
st_informed_s, truth = Group, estimate = Predictions0)[, ".estimate"]

}
```

```
## Warning: Unknown or uninitialised column: `Accuracy`.
```

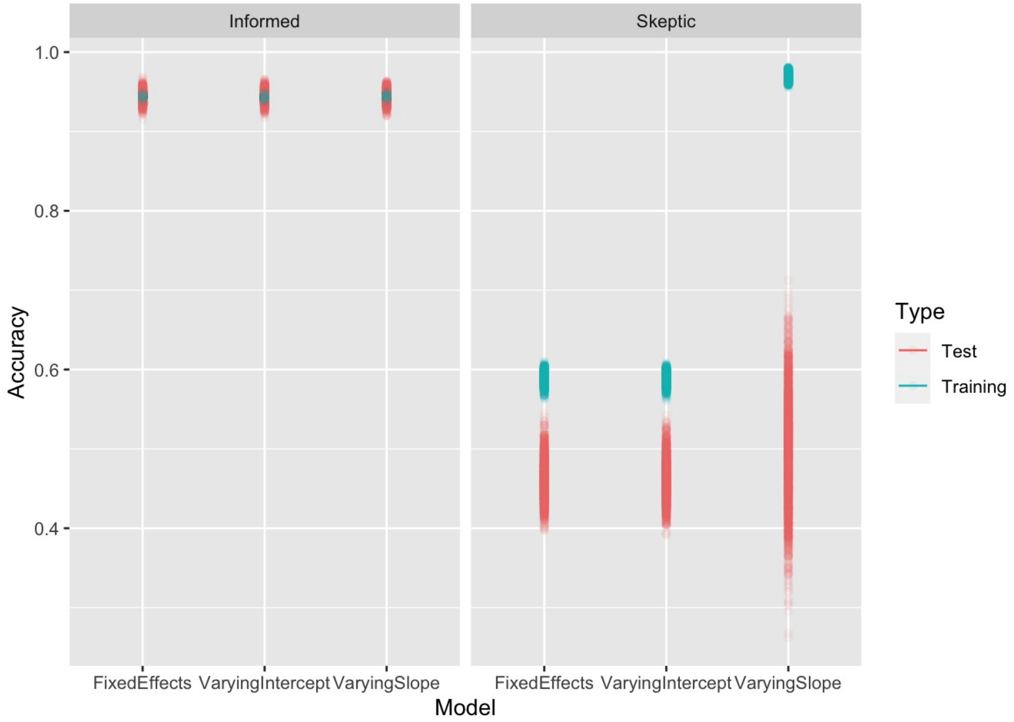
```
#Informed: varying intercepts
train1 <- inv_logit_scaled(posterior_linpred(pitch_m_informed_3, summary = F))
test1 <- inv_logit_scaled(posterior_linpred(pitch_m_informed_3, summary = F,
                                            newdata = test_informed_s, allow_new_levels = T))

for (i in seq(2000)) {
  train_informed_s$Predictions1 <- as.factor(ifelse(train1[i,]> 0.5, "Schizophrenia", "Control"))
  test_informed_s$Predictions1 <- as.factor(ifelse(test1[i,]> 0.5, "Schizophrenia", "Control"))

  PerformanceProb$Accuracy[PerformanceProb$Sample == i & PerformanceProb$Model == "VaryingIntercept" &
                             PerformanceProb$Setup == "Informed" & PerformanceProb$Type == "Training"] <- accurac
y(train_informed_s, truth = Group, estimate = Predictions1)[, ".estimate"]

    PerformanceProb$Accuracy[PerformanceProb$Sample == i & PerformanceProb$Model == "VaryingIntercept" &
                             PerformanceProb$Setup == "Informed" & PerformanceProb$Type == "Test"] <- accuracy(te
st_informed_s, truth = Group, estimate = Predictions1)[, ".estimate"]

}

#Informed: varying slopes
train2 <- inv_logit_scaled(posterior_linpred(pitch_m_informed_2, summary = F))
test2 <- inv_logit_scaled(posterior_linpred(pitch_m_informed_2, summary = F,
                                            newdata = test_informed_s, allow_new_levels = T))

for (i in seq(2000)) {
  train_informed_s$Predictions2 <- as.factor(ifelse(train2[i,]> 0.5, "Schizophrenia", "Control"))
  test_informed_s$Predictions2 <- as.factor(ifelse(test2[i,]> 0.5, "Schizophrenia", "Control"))

  PerformanceProb$Accuracy[PerformanceProb$Sample == i & PerformanceProb$Model == "VaryingSlope" &
                             PerformanceProb$Setup == "Informed" & PerformanceProb$Type == "Training"] <- accurac
y(train_informed_s, truth = Group, estimate = Predictions2)[, ".estimate"]

    PerformanceProb$Accuracy[PerformanceProb$Sample == i & PerformanceProb$Model == "VaryingSlope" &
                             PerformanceProb$Setup == "Informed" & PerformanceProb$Type == "Test"] <- accuracy(te
st_informed_s, truth = Group, estimate = Predictions2)[, ".estimate"]

}
```

```r
#Skeptic: fixed effects

train0_s <- inv_logit_scaled(posterior_linpred(pitch_m_skeptic, summary = F))
test0_s <- inv_logit_scaled(posterior_linpred(pitch_m_skeptic, summary = F,
                                            newdata = test_skeptic_s, allow_new_levels = T))

for (i in seq(2000)) {
  train_skeptic_s$Predictions0 <- as.factor(ifelse(train0_s[i,]> 0.5, "Schizophrenia", "Control"))
  test_skeptic_s$Predictions0 <- as.factor(ifelse(test0_s[i,]> 0.5, "Schizophrenia", "Control"))

  PerformanceProb$Accuracy[PerformanceProb$Sample == i & PerformanceProb$Model == "FixedEffects" &
                           PerformanceProb$Setup == "Skeptic" & PerformanceProb$Type == "Training"] <- accuracy
(train_skeptic_s, truth = Group, estimate = Predictions0)[, ".estimate"]

    PerformanceProb$Accuracy[PerformanceProb$Sample == i & PerformanceProb$Model == "FixedEffects" &
                           PerformanceProb$Setup == "Skeptic" & PerformanceProb$Type == "Test"] <- accuracy(tes
t_skeptic_s, truth = Group, estimate = Predictions0)[, ".estimate"]

}
#Skeptic: varying intercepts
train1_s <- inv_logit_scaled(posterior_linpred(pitch_m_skeptic_2, summary = F))
test1_s <- inv_logit_scaled(posterior_linpred(pitch_m_skeptic_2, summary = F,
                                            newdata = test_skeptic_s, allow_new_levels = T))

for (i in seq(2000)) {
  train_skeptic_s$Predictions1 <- as.factor(ifelse(train1_s[i,]> 0.5, "Schizophrenia", "Control"))
  test_skeptic_s$Predictions1 <- as.factor(ifelse(test1_s[i,]> 0.5, "Schizophrenia", "Control"))

  PerformanceProb$Accuracy[PerformanceProb$Sample == i & PerformanceProb$Model == "VaryingIntercept" &
                           PerformanceProb$Setup == "Skeptic" & PerformanceProb$Type == "Training"] <- accuracy
(train_skeptic_s, truth = Group, estimate = Predictions1)[, ".estimate"]

    PerformanceProb$Accuracy[PerformanceProb$Sample == i & PerformanceProb$Model == "VaryingIntercept" &
                           PerformanceProb$Setup == "Skeptic" & PerformanceProb$Type == "Test"] <- accuracy(tes
t_skeptic_s, truth = Group, estimate = Predictions1)[, ".estimate"]
}

#Skeptic: varying slopes
train2_s <- inv_logit_scaled(posterior_linpred(pitch_m_skeptic_3, summary = F))
test2_s <- inv_logit_scaled(posterior_linpred(pitch_m_skeptic_3, summary = F,
                                            newdata = test_skeptic_s, allow_new_levels = T))

for (i in seq(2000)) {
  train_skeptic_s$Predictions2 <- as.factor(ifelse(train2_s[i,]> 0.5, "Schizophrenia", "Control"))
  test_skeptic_s$Predictions2 <- as.factor(ifelse(test2_s[i,]> 0.5, "Schizophrenia", "Control"))

  PerformanceProb$Accuracy[PerformanceProb$Sample == i & PerformanceProb$Model == "VaryingSlope" &
                           PerformanceProb$Setup == "Skeptic" & PerformanceProb$Type == "Training"] <- accuracy
(train_skeptic_s, truth = Group, estimate = Predictions2)[, ".estimate"]

    PerformanceProb$Accuracy[PerformanceProb$Sample == i & PerformanceProb$Model == "VaryingSlope" &
                           PerformanceProb$Setup == "Skeptic" & PerformanceProb$Type == "Test"] <- accuracy(tes
t_skeptic_s, truth = Group, estimate = Predictions2)[, ".estimate"]
}
```

Plot of accuracy in classification:

```r
PerformanceProb <- PerformanceProb %>%
  mutate(
    Model = as.factor(Model),
    Type = as.factor(Type),
    Setup = as.factor(Setup),
    Accuracy= as.numeric(Accuracy)
  )

performance_1 <- ggplot(data=PerformanceProb, aes(x=Model, y=Accuracy, color = Type)) +
  geom_point(alpha = 0.05) +
  facet_wrap(~Setup) +
  stat_summary(geom = "line", fun = mean)

performance_1
```

```
## `geom_line()`: Each group consists of only one observation.
## ℹ Do you need to adjust the group aesthetic?
## `geom_line()`: Each group consists of only one observation.
## ℹ Do you need to adjust the group aesthetic?
```



Assessing impact of the priors:

```
#What is the impact of the priors?
#Construct the sequence of sds to loop through for the slope
priSD <- seq(0.1, 1.5, length.out = 15) #defining prior confidence
priorsN <- p_range_p_2 #using mine

#Create empty variables to store output of the loop:
PerformanceProb_p <- tibble(expand_grid(
  Sample = seq(4000),
  Prior = priSD,
  Setup = c("Informed", "Skeptic"),
  Type = c("Training", "Test")
))
```

```r
for (i in 1:length(priSD)) {
  print(i)
  priorsN[2,] <- set_prior(paste0("normal(0, ", priSD[i], ")"), class = "b")
#model Fixed effects
  model_for_loop <- update(
    pitch_m_informed,
    prior = priorsN,
    sample_prior = T,
    backend = "cmdstanr",
    chains = 2,
    cores = 2,
    iter = 4e3,
    threads = threading(2),
    control = list(adapt_delta = 0.9, max_treedepth = 20)
  )

    model_sk_for_loop <- update(
    pitch_m_skeptic,
    prior = priorsN,
    sample_prior = T,
    backend = "cmdstanr",
    chains = 2,
    cores = 2,
    iter = 4e3,
    threads = threading(2),
    control = list(adapt_delta = 0.9, max_treedepth = 20)
  )


  train_inf <- inv_logit_scaled(posterior_linpred(model_for_loop, summary = F))
  test_inf <- inv_logit_scaled(posterior_linpred(model_for_loop, summary = F,
                                          newdata = test_informed_s, allow_new_levels = T))

  train_sk <- inv_logit_scaled(posterior_linpred(model_sk_for_loop, summary = F))
  test_sk <- inv_logit_scaled(posterior_linpred(model_sk_for_loop, summary = F,
                                          newdata = test_skeptic_s, allow_new_levels = T))

  for (j in seq(4000)) {
    train_informed_s$Predictions <- as.factor(ifelse(train_inf[j,] > 0.5, "Schizophrenia", "Control"))
    test_informed_s$Predictions <- as.factor(ifelse(test_inf[j,] > 0.5, "Schizophrenia", "Control"))

    train_skeptic_s$Predictions <- as.factor(ifelse(train_sk[j,] > 0.5, "Schizophrenia", "Control"))
    test_skeptic_s$Predictions <- as.factor(ifelse(test_sk[j,] > 0.5, "Schizophrenia", "Control"))

    PerformanceProb_p$Accuracy[PerformanceProb_p$Sample == j & PerformanceProb_p$Prior == priSD[i] &
                              PerformanceProb_p$Setup == "Informed" & PerformanceProb_p$Type == "Training"] <- ac
curacy(train_informed_s,  truth = Group, estimate = Predictions)[, ".estimate"]

    PerformanceProb_p$Accuracy[PerformanceProb_p$Sample == j & PerformanceProb_p$Prior == priSD[i] &
                              PerformanceProb_p$Setup == "Informed" & PerformanceProb_p$Type == "Test"] <- accura
cy(test_informed_s, truth = Group, estimate = Predictions)[, ".estimate"]

    PerformanceProb_p$Accuracy[PerformanceProb_p$Sample == j & PerformanceProb_p$Prior == priSD[i] &
                             PerformanceProb_p$Setup == "Skeptic" & PerformanceProb_p$Type == "Training"] <- accu
racy(train_skeptic_s, truth = Group, estimate = Predictions)[, ".estimate"]

    PerformanceProb_p$Accuracy[PerformanceProb_p$Sample == j & PerformanceProb_p$Prior == priSD[i] &
                             PerformanceProb_p$Setup == "Skeptic" & PerformanceProb_p$Type == "Test"] <- accuracy
(test_skeptic_s, truth = Group, estimate = Predictions)[, ".estimate"]
  }
}
```

```
priSD <- seq(0.1, 1.5, length.out = 15) #defining prior confidence
priorsN <- p_range_p_2 #using mine

#Create empty variables to store output of the loop:
PerformanceProb_p_3 <- tibble(expand_grid(
  Sample = seq(4000),
  Prior = priSD,
  Setup = c("Informed", "Skeptic"),
  Type = c("Training", "Test")
))

for (i in 1:length(priSD)) {
  print(i)
  priorsN[2,] <- set_prior(paste0("normal(0, ", priSD[i], ")"), class = "b")
#model Varying slope
  model_for_loop <- update(
    pitch_m_informed_2, #model with varying slope
    prior = priorsN,
    sample_prior = T,
    backend = "cmdstanr",
    chains = 2,
    cores = 2,
    iter = 4e3,
    threads = threading(2),
    control = list(adapt_delta = 0.9, max_treedepth = 20)
  )

    model_sk_for_loop <- update(
    pitch_m_skeptic_3, #model with varying slope
    prior = priorsN,
    sample_prior = T,
    backend = "cmdstanr",
    chains = 2,
    cores = 2,
    iter = 4e3,
    threads = threading(2),
    control = list(adapt_delta = 0.9, max_treedepth = 20)
  )


  train_inf_3 <- inv_logit_scaled(posterior_linpred(model_for_loop, summary = F))
  test_inf_3 <- inv_logit_scaled(posterior_linpred(model_for_loop, summary = F,
                                            newdata = test_informed_s, allow_new_levels = T))

  train_sk_3 <- inv_logit_scaled(posterior_linpred(model_sk_for_loop, summary = F))
  test_sk_3 <- inv_logit_scaled(posterior_linpred(model_sk_for_loop, summary = F,
                                            newdata = test_skeptic_s, allow_new_levels = T))

  for (j in seq(4000)) {
    train_informed_s$Predictions_3 <- as.factor(ifelse(train_inf_3[j,] > 0.5, "Schizophrenia", "Control"))
    test_informed_s$Predictions_3 <- as.factor(ifelse(test_inf_3[j,] > 0.5, "Schizophrenia", "Control"))

    train_skeptic_s$Predictions_3 <- as.factor(ifelse(train_sk_3[j,] > 0.5, "Schizophrenia", "Control"))
    test_skeptic_s$Predictions_3 <- as.factor(ifelse(test_sk_3[j,] > 0.5, "Schizophrenia", "Control"))

    PerformanceProb_p_3$Accuracy[PerformanceProb_p_3$Sample == j & PerformanceProb_p_3$Prior == priSD[i] &
                              PerformanceProb_p_3$Setup == "Informed" & PerformanceProb_p_3$Type == "Training"] <
- accuracy(train_informed_s,  truth = Group, estimate = Predictions_3)[, ".estimate"]

    PerformanceProb_p_3$Accuracy[PerformanceProb_p_3$Sample == j & PerformanceProb_p_3$Prior == priSD[i] &
                              PerformanceProb_p_3$Setup == "Informed" & PerformanceProb_p_3$Type == "Test"] <- ac
curacy(test_informed_s, truth = Group, estimate = Predictions_3)[, ".estimate"]

    PerformanceProb_p_3$Accuracy[PerformanceProb_p_3$Sample == j & PerformanceProb_p_3$Prior == priSD[i] &
                             PerformanceProb_p_3$Setup == "Skeptic" & PerformanceProb_p_3$Type == "Training"] <-
accuracy(train_skeptic_s, truth = Group, estimate = Predictions_3)[, ".estimate"]

    PerformanceProb_p_3$Accuracy[PerformanceProb_p_3$Sample == j & PerformanceProb_p_3$Prior == priSD[i] &
                             PerformanceProb_p_3$Setup == "Skeptic" & PerformanceProb_p_3$Type == "Test"] <- accu
racy(test_skeptic_s, truth = Group, estimate = Predictions_3)[, ".estimate"]
  }
}
```

Plot of priors (Fixed effects model):

```
PerformanceProb_p <- PerformanceProb_p %>%
  mutate(
    Type = as.factor(Type),
    Setup = as.factor(Setup),
    Prior = as.numeric(Prior),
    Accuracy= as.numeric(Accuracy)
  )

ggplot(data=PerformanceProb_p, aes(x=Prior, y=Accuracy, color = Type)) +
  geom_point(alpha = 0.05) +
  facet_wrap(~Setup) +
  geom_hline(yintercept = 0.5) +
  ggtitle("Analysis of accuracy")
```

Plot of priors (Varying slope model):

```
PerformanceProb_p_3 <- PerformanceProb_p_3 %>%
  mutate(
    Type = as.factor(Type),
    Setup = as.factor(Setup),
    Prior = as.numeric(Prior),
    Accuracy= as.numeric(Accuracy)
  )

ggplot(data=PerformanceProb_p_3, aes(x=Prior, y=Accuracy, color = Type)) +
  geom_point(alpha = 0.05) +
  facet_wrap(~Setup) +
  geom_hline(yintercept = 0.5) +
  ggtitle("Sensitivity analysis  (model with varying slopes)")
```

Feature importance:

```
library(reshape)
#Informed:
#Model with fixed intercepts

#Converting into a tibble, changing the format so I could plot it as overlapping densities.
f_in <- as_draws_df(pitch_m_informed)

ggplot(f_in) +
  geom_density(aes(x = b_v1), colour = "green", fill = "green", alpha = 0.2) +
  geom_density(aes(x = b_v2), colour = "blue", fill = "blue", alpha = 0.2) +
  geom_density(aes(x = b_v3), colour = "red", fill = "red", alpha = 0.2) +
  geom_density(aes(x = b_v4), colour = "yellow", fill = "yellow", alpha = 0.2) +
  geom_density(aes(x = b_v5), colour = "black", fill = "black", alpha = 0.2) +
  geom_density(aes(x = b_v6), colour = "violet", fill = "violet", alpha = 0.2) +
  geom_density(aes(x = b_v7), colour = "grey", fill = "grey", alpha = 0.2) +
  geom_density(aes(x = b_v8), colour = "chocolate", fill = "chocolate", alpha = 0.2) +
  geom_density(aes(x = b_v9), colour = "darkgreen", fill = "darkgreen", alpha = 0.2) +
  geom_density(aes(x = b_v10), colour = "orange", fill = "orange", alpha = 0.2) +
  xlab("Variables v1-v10") + ggtitle("Fixed effects")

p_v2 <- ggplot(f_in, aes(x=b_v2)) +
  geom_vline(xintercept = -0.55, colour="red", linetype = "longdash") +
  geom_density() + xlab("Pitch variability: v2")

p_v1 <- ggplot(f_in, aes(x=b_v1)) +
  geom_vline(xintercept = -1.26, colour="red", linetype = "longdash") +
  geom_density() + xlab("Proportion of spoken time: v1")

p_v3 <- ggplot(f_in, aes(x=b_v3)) +
  geom_vline(xintercept = -0.75, colour="red", linetype = "longdash") +
  geom_density() + xlab("Speech rate: v3")

p_v4 <- ggplot(f_in, aes(x=b_v4)) +
  geom_vline(xintercept = 1.89, colour="red", linetype = "longdash") +
  geom_density() + xlab("Duration of pauses: v4")

p_v5 <- ggplot(f_in, aes(x=b_v5)) +
  geom_vline(xintercept = 0.25, colour="red", linetype = "longdash") +
  geom_density() + xlab("Pitch mean: v5")

p_v6 <- ggplot(f_in, aes(x=b_v6)) +
  geom_vline(xintercept = 0.05, colour="red", linetype = "longdash") +
  geom_density() + xlab("Number of pauses: v6")

p_v7 <- ggplot(f_in, aes(x=b_v7)) +
```

```
  geom_vline(xintercept = 0, colour="red", linetype = "longdash") +
  geom_density()

p_v8 <- ggplot(f_in, aes(x=b_v8)) +
  geom_vline(xintercept = 0, colour="red", linetype = "longdash") +
  geom_density()

p_v9 <- ggplot(f_in, aes(x=b_v9)) +
  geom_vline(xintercept = 0, colour="red", linetype = "longdash") +
  geom_density()

p_v10 <- ggplot(f_in, aes(x=b_v10)) +
  geom_vline(xintercept = 0, colour="red", linetype = "longdash") +
  geom_density()

#Plot of density for each  simulated variable
grid.arrange(p_v1, p_v2, p_v3, p_v4, p_v5, p_v6, p_v7, p_v8, p_v9, p_v10)
#According to this model, with fixed predictors, the features of v1 (proportion of spoken time), v3 (Speech rate)
and v4 (Duration of pauses) are used the most.


#Model with Varying intercept:
v_int <- as_draws_df(pitch_m_informed_3)
ggplot(v_int) +
  geom_density(aes(x = b_v1), colour = "green", fill = "green", alpha = 0.2) +
  geom_density(aes(x = b_v2), colour = "blue", fill = "blue", alpha = 0.2) +
  geom_density(aes(x = b_v3), colour = "red", fill = "red", alpha = 0.2) +
  geom_density(aes(x = b_v4), colour = "yellow", fill = "yellow", alpha = 0.2) +
  geom_density(aes(x = b_v5), colour = "black", fill = "black", alpha = 0.2) +
  geom_density(aes(x = b_v6), colour = "violet", fill = "violet", alpha = 0.2) +
  geom_density(aes(x = b_v7), colour = "grey", fill = "grey", alpha = 0.2) +
  geom_density(aes(x = b_v8), colour = "chocolate", fill = "chocolate", alpha = 0.2) +
  geom_density(aes(x = b_v9), colour = "darkgreen", fill = "darkgreen", alpha = 0.2) +
  geom_density(aes(x = b_v10), colour = "orange", fill = "orange", alpha = 0.2) +
  xlab("Variables v1-v10") + ggtitle("Varying intercept")

p_v1_v <- ggplot(v_int, aes(x=b_v1)) +
  geom_vline(xintercept = -1.26, colour="red", linetype = "longdash") +
  geom_density() + xlab("Proportion of spoken time: v1")

p_v2_v <- ggplot(v_int, aes(x=b_v2)) +
  geom_vline(xintercept = -0.55, colour="red", linetype = "longdash") +
  geom_density() + xlab("Pitch variability: v2")

p_v3_v <- ggplot(v_int, aes(x=b_v3)) +
  geom_vline(xintercept = -0.75, colour="red", linetype = "longdash") +
  geom_density() + xlab("Speech rate: v3")

p_v4_v <- ggplot(v_int, aes(x=b_v4)) +
  geom_vline(xintercept = 1.89, colour="red", linetype = "longdash") +
  geom_density() + xlab("Duration of pauses: v4")

p_v5_v <- ggplot(v_int, aes(x=b_v5)) +
  geom_vline(xintercept = 0.25, colour="red", linetype = "longdash") +
  geom_density() + xlab("Pitch mean: v5")

p_v6_v <- ggplot(v_int, aes(x=b_v6)) +
  geom_vline(xintercept = 0.05, colour="red", linetype = "longdash") +
  geom_density() + xlab("Number of pauses: v6")

p_v7_v <- ggplot(v_int, aes(x=b_v7)) +
  geom_vline(xintercept = 0, colour="red", linetype = "longdash") +
  geom_density()

p_v8_v <- ggplot(v_int, aes(x=b_v8)) +
  geom_vline(xintercept = 0, colour="red", linetype = "longdash") +
  geom_density()

p_v9_v <- ggplot(v_int, aes(x=b_v9)) +
  geom_vline(xintercept = 0, colour="red", linetype = "longdash") +
  geom_density()

p_v10_v <- ggplot(v_int, aes(x=b_v10)) +
  geom_vline(xintercept = 0, colour="red", linetype = "longdash") +
  geom_density()

grid.arrange(p_v1_v, p_v2_v, p_v3_v, p_v4_v, p_v5_v, p_v6_v, p_v7_v, p_v8_v, p_v9_v, p_v10_v)
#According to this model, with varying intercept, the features of v1 (proportion of spoken time), v3 (Speech rate
), v5 (Pitch mean) and v6(number of pauses) are used the most.
```

```
#Model with Varying slopes:
v_sl <- as_draws_df(pitch_m_informed_2)

ggplot(v_sl) +
  geom_density(aes(x = b_v1), colour = "green", fill = "green", alpha = 0.2) +
  geom_density(aes(x = b_v2), colour = "blue", fill = "blue", alpha = 0.2) +
  geom_density(aes(x = b_v3), colour = "red", fill = "red", alpha = 0.2) +
  geom_density(aes(x = b_v4), colour = "yellow", fill = "yellow", alpha = 0.2) +
  geom_density(aes(x = b_v5), colour = "black", fill = "black", alpha = 0.2) +
  geom_density(aes(x = b_v6), colour = "violet", fill = "violet", alpha = 0.2) +
  geom_density(aes(x = b_v7), colour = "grey", fill = "grey", alpha = 0.2) +
  geom_density(aes(x = b_v8), colour = "chocolate", fill = "chocolate", alpha = 0.2) +
  geom_density(aes(x = b_v9), colour = "darkgreen", fill = "darkgreen", alpha = 0.2) +
  geom_density(aes(x = b_v10), colour = "orange", fill = "orange", alpha = 0.2) +
  xlab("Variables v1-v10") + ggtitle("Varying slope")

p_v1_s <- ggplot(v_sl, aes(x=b_v1)) +
  geom_vline(xintercept = -1.26, colour="red", linetype = "longdash") +
  geom_density() + xlab("Proportion of spoken time: v1")

p_v2_s <- ggplot(v_sl, aes(x=b_v2)) +
  geom_vline(xintercept = -0.55, colour="red", linetype = "longdash") +
  geom_density() + xlab("Pitch variability: v2")

p_v3_s <- ggplot(v_sl, aes(x=b_v3)) +
  geom_vline(xintercept = -0.75, colour="red", linetype = "longdash") +
  geom_density() + xlab("Speech rate: v3")

p_v4_s <- ggplot(v_sl, aes(x=b_v4)) +
  geom_vline(xintercept = 1.89, colour="red", linetype = "longdash") +
  geom_density() + xlab("Duration of pauses: v4")

p_v5_s <- ggplot(v_sl, aes(x=b_v5)) +
  geom_vline(xintercept = 0.25, colour="red", linetype = "longdash") +
  geom_density() + xlab("Pitch mean: v5")

p_v6_s <- ggplot(v_sl, aes(x=b_v6)) +
  geom_vline(xintercept = 0.05, colour="red", linetype = "longdash") +
  geom_density() + xlab("Number of pauses: v6")

p_v7_s <- ggplot(v_sl, aes(x=b_v7)) +
  geom_vline(xintercept = 0, colour="red", linetype = "longdash") +
  geom_density()

p_v8_s <- ggplot(v_sl, aes(x=b_v8)) +
  geom_vline(xintercept = 0, colour="red", linetype = "longdash") +
  geom_density()

p_v9_s <- ggplot(v_sl, aes(x=b_v9)) +
  geom_vline(xintercept = 0, colour="red", linetype = "longdash") +
  geom_density()

p_v10_s <- ggplot(v_sl, aes(x=b_v10)) +
  geom_vline(xintercept = 0, colour="red", linetype = "longdash") +
  geom_density()

grid.arrange(p_v1_s, p_v2_s, p_v3_s, p_v4_s, p_v5_s, p_v6_s, p_v7_s, p_v8_s, p_v9_s, p_v10_s)
#According to this model, with varying intercept, the features of v1 (proportion of spoken time), v3 (Speech rate
), v5 (Pitch mean) and v6(number of pauses) are used the most.
```

Another way of assessing feature importance:

```
#Tidymodels
train_informed_scaled <- train_informed_s[3:13] #Excluding unnecessary columns

d_inf <- train_informed_scaled %>%
  mutate(ID = NULL, Trial = NULL, Preds = NULL, Predictions = NULL, v1_s = NULL)

LogisticRegression_inf_SIM <- logistic_reg() %>%
  set_mode("classification") %>%
  set_engine("glm") %>%
  fit(Group ~ . , data = d_inf)

explainer_lm <- explain_tidymodels( #for the logistic regression
  LogisticRegression_inf_SIM,
  data = train_informed_scaled,
  y = as.numeric(train_informed_scaled$Group) - 1,
  label = "logReg",
  verbose = FALSE
)

explainer_lm %>% model_parts() %>% plot(show_boxplots = FALSE) + ggtitle("Feature importance ", "")


model_profile_lm1 <- model_profile(explainer_lm, type = "partial",
                                   variables = c("v1", "v2", "v3", "v4","v5", "v6","v7", "v8","v9", "v10"))
plot(model_profile_lm1, variables = c("v1", "v2", "v3", "v4","v5", "v6")) + ggtitle("Partial dependence profile "
, "")
plot(model_profile_lm1, variables = c("v7", "v8", "v9", "v10")) + ggtitle("Partial dependence profile ", "")
```

Part III - Applying the ML pipeline to empirical data

Download the empirical dataset from brightspace and apply your ML pipeline to the new data, adjusting where needed. Warning: in the simulated dataset we only had 10 features, now you have many more! Such is the life of the ML practitioner. Consider the impact a higher number of features will have on your ML inference, and decide whether you need to cut down the number of features before running the pipeline (or alternatively expand the pipeline to add feature selection).

```
d <- read_csv("Ass3_empiricalData1.csv")
```

```
## Rows: 1889 Columns: 398
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr   (5): NewID, Diagnosis, Language, Gender, Trial
## dbl (393): PatID, Corpus, Duration_Praat, F0_Mean_Praat, F0_SD_Praat, Intens...
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
d$PatID <- as.factor(d$PatID) #Has 122 levels (?)
d$NewID <- as.factor(d$NewID) #Has 221 levels
```

Cleaning the data set, using it for global feature importance later:

```r
edata <- d %>%
  select(-ends_with("Median"),-"Trial", -"Corpus", -"PatID", -"Language", -"Gender", -"NewID") #removing unnecess
ary columns just for now, so I could do a feature importance analysis.


#Identifying and removing highly correlated variables/columns:
no_dig_edata <- edata %>% select( -"Diagnosis")
cor_matrix <- cor(no_dig_edata)


#Modify correlation matrix:
cor_matrix_rm <- cor_matrix
cor_matrix_rm[upper.tri(cor_matrix_rm)] <- 0
diag(cor_matrix_rm) <- 0


#Removing highly correlated variables:
new_emp <- no_dig_edata[ , !apply(cor_matrix_rm,
                           2,
                           function(x) any(x > 0.7))] #Chose correlation 0.8 at first, but R could not load it as
well, therefore cut it down to 0.7.


#Adding diagnosis back to the data set
emp_data <- cbind(new_emp, Diagnosis = edata$Diagnosis)
emp_data$Diagnosis <- as.factor(emp_data$Diagnosis)
```

PCA Haven't used it, keeping as a note.

```r
pca_rec <- recipe(~., data = edata) %>%
  update_role(NewID, Diagnosis, new_role = "id") %>%
  step_normalize(all_predictors()) %>%
  step_pca(all_predictors())

pca_prep <- prep(pca_rec)
tidied_pca <- tidy(pca_prep, 2)

tidied_pca %>%
  filter(component %in% paste0("PC", 1:5)) %>%
  mutate(component = fct_inorder(component)) %>%
  ggplot(aes(value, terms, fill = terms)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~component, nrow = 1) +
  labs(y = NULL)

library("tidytext")

tidied_pca %>%
  filter(component %in% paste0("PC", 1:5)) %>%
  group_by(component) %>%
  top_n(10, abs(value)) %>%
  ungroup() %>%
  mutate(terms = reorder_within(terms, abs(value), component)) %>%
  ggplot(aes(abs(value), terms, fill = value > 0)) +
  geom_col() +
  facet_wrap(~component, scales = "free_y") +
  scale_y_reordered() +
  labs(
    x = "Absolute value of contribution",
    y = NULL, fill = "Positive?"
  )

juice(pca_prep) %>%
  ggplot(aes(terms, value, label = NewID)) +
  geom_point(aes(color = Diagnosis), alpha = 0.7, size = 2) +
  geom_text(check_overlap = TRUE, hjust = "inward") +
  labs(color = NULL)
```

Running machine learning pipeline on the empirical data

1. Data budgeting:

```r
library("groupdata2", "rsample")
set.seed(100)

#Adding the ID and gender to the data set:
emp_df <- cbind(emp_data, NewID = d$NewID, Gender = d$Gender) #1889 obs.

#Re-coding levels for simplicity:
levels(emp_df$NewID) <- seq(1:221)

#Splitting the data into 20/80 partition:
emp_df_split <-  partition(emp_df, p = 0.2, id_col = "NewID", cat_col = c("Diagnosis", "Gender"))


#Creating testing and training sets:
e_test <- emp_df_split[[1]]
e_train <- emp_df_split[[2]]


#Describing the population in training and test data sets:
e_test$Gender <- as.factor(e_test$Gender)
e_train$Gender <- as.factor(e_train$Gender)

e_test$Diagnosis <- as.factor(e_test$Diagnosis)
e_train$Diagnosis <- as.factor(e_train$Diagnosis)

summary(e_test$Gender) #154 F and 213 M
```

```
##   F   M
## 139 240
```

```r
summary(e_train$Gender) #654 F and 868 M
```

```
##   F   M
## 669 841
```

```r
summary(e_test$Diagnosis) #188 CT and 179 SCZ
```

```
##  CT SCZ
## 195 184
```

```r
summary(e_train$Diagnosis) #801 CT and 721 SCZ
```

```
##  CT SCZ
## 794 716
```

2. Data pre-processing:

```r
#Scaling training data set:

rec_e_train <- e_train %>%
  recipe(Diagnosis ~ . ) %>%
  step_scale(all_numeric() ) %>%
  step_center(all_numeric() ) %>%
  prep(training = e_train, retain = TRUE)

e_train_s <- juice(rec_e_train)

e_test_s <- bake(rec_e_train, new_data = e_test, all_predictors()) %>%
  mutate(Diagnosis = e_test$Diagnosis)
.libPaths()
```

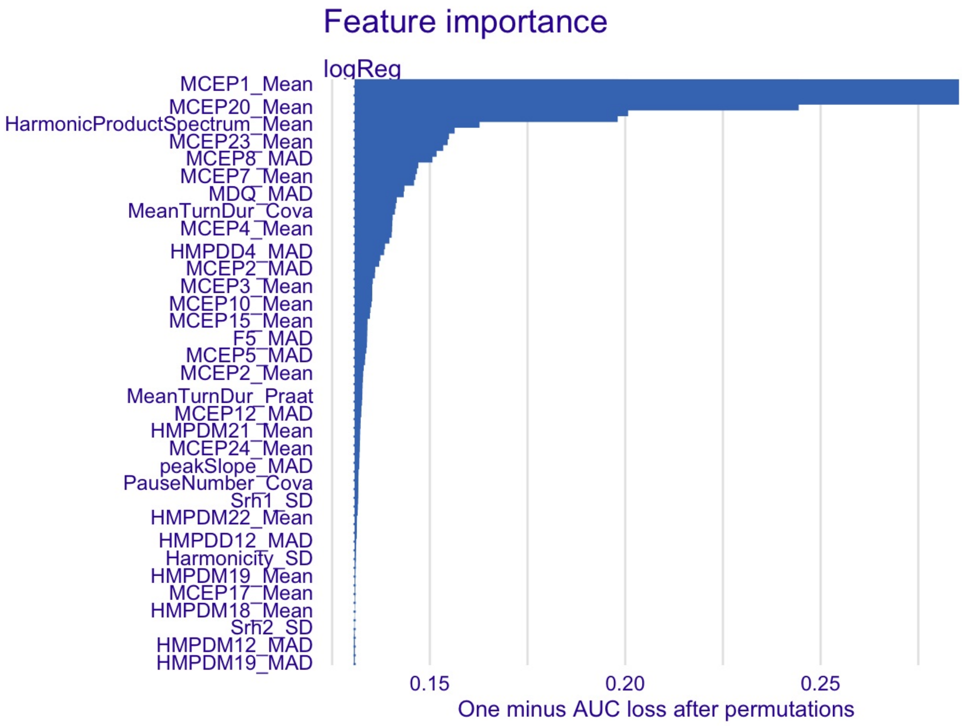
```
## [1] "/Library/Frameworks/R.framework/Versions/4.2/Resources/library"
```

Global feature importance

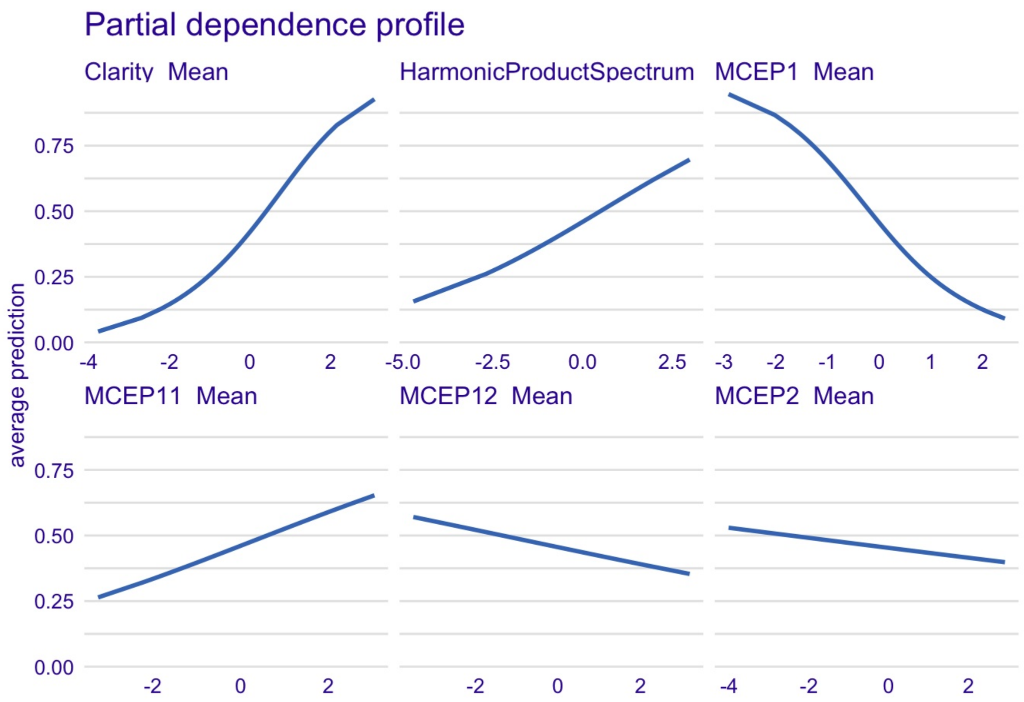
```
#Starting with analysis on feature importance:
d_e_train <- e_train_s[2:103] %>% select(-"Gender")#Leaving only necessary columns

LogisticRegression_inf_emp <- logistic_reg() %>%
  set_mode("classification") %>%
  set_engine("glm") %>%
  fit(Diagnosis ~ . , data = d_e_train)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

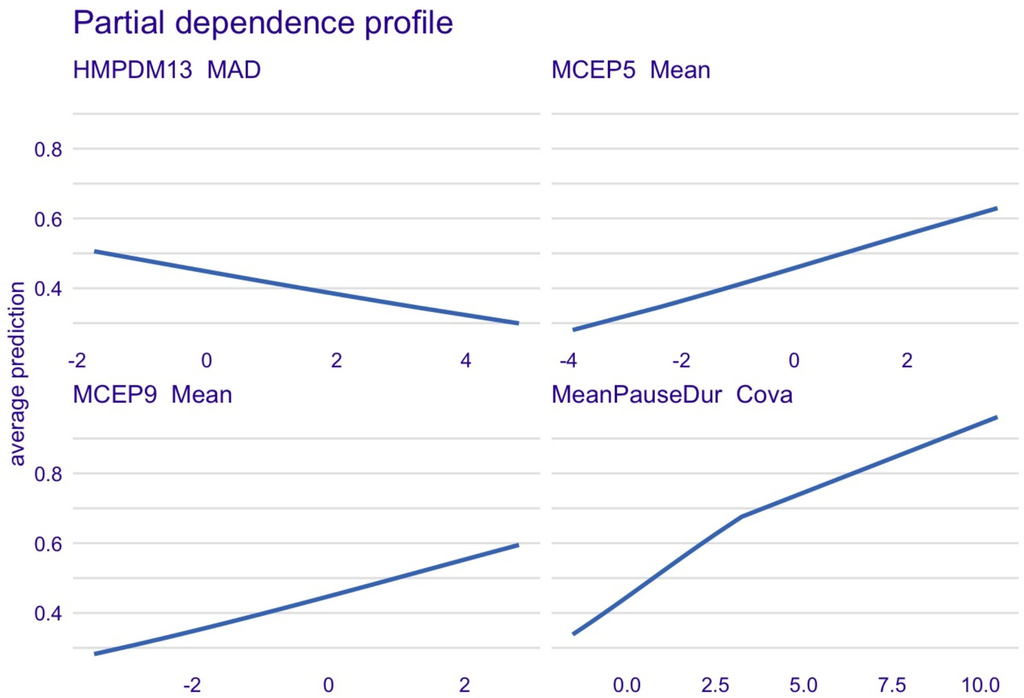
```
explainer_lm_emp <- explain_tidymodels( #for the logistic regression
  LogisticRegression_inf_emp,
  data = d_e_train,
  y = as.numeric(d_e_train$Diagnosis) - 1,
  label = "logReg",
  verbose = FALSE
)

ed_f_i_1 <- explainer_lm_emp %>% model_parts() %>% plot(show_boxplots = FALSE) + ggtitle("Feature importance ", "
") + scale_x_discrete(guide = guide_axis(check.overlap = TRUE)) #Avoids the overlapping of the text.

explainer_lm_emp %>% model_parts() %>% plot(show_boxplots = FALSE) + ggtitle("Feature importance ", "") + scale_x
_discrete(guide = guide_axis(check.overlap = TRUE))
```




```
model_profile_lm_emp <- model_profile(explainer_lm_emp, type = "partial",
                                   variables = c("MCEP1_Mean", "Clarity_Mean", "HarmonicProductSpectrum_Mean", "M
CEP11_Mean","MCEP12_Mean", "MCEP2_Mean","MCEP9_Mean", "MCEP5_Mean","MeanPauseDur_Cova", "HMPDM13_MAD"))
plot(model_profile_lm_emp, variables = c("MCEP1_Mean", "Clarity_Mean", "HarmonicProductSpectrum_Mean", "MCEP11_Me
an","MCEP12_Mean", "MCEP2_Mean")) + ggtitle("Partial dependence profile ", "")
```




```
plot(model_profile_lm_emp, variables = c("MCEP9_Mean", "MCEP5_Mean", "MeanPauseDur_Cova", "HMPDM13_MAD")) + ggtit
le("Partial dependence profile ", "")
```



```
#ggsave("ef_1_analysis.jpeg", plot = ed_f_i_1, path = "/Users/justina/Desktop/Desktop - Justina's MacBook Pro/Aar
hus_Uni/Semester_3/Methods 3/Assignment-3")

#Important variables: MCEP1_Mean, Clarity_Mean, HarmonicProcuctSpectrum_Mean, MCEP11_Mean, MCEP12_Mean, MCEP2_Mea
n, MCEP9_Mean, MCEP5_Mean, MeanPauseDur_Cova, HMPDM13_MAD, F5_MAD, TurnNumMin_Cova, F0_SD_Praat.

randomforest_inf_emp <- rand_forest() %>%
  set_mode("classification") %>%
  set_engine("randomForest") %>%
  fit(Diagnosis ~ . , data = d_e_train)

explainer_rf_emp <- explain_tidymodels( #for the logistic regression
  randomforest_inf_emp,
  data = d_e_train,
  y = as.numeric(d_e_train$Diagnosis) - 1,
  label = "random forest",
  verbose = FALSE
)

ed_f_i_2 <- explainer_rf_emp %>% model_parts() %>% plot(show_boxplots = FALSE) + ggtitle("Feature importance ", "
") + scale_x_discrete(guide = guide_axis(check.overlap = TRUE)) #Avoids the overlapping of the text.
#ggsave("ef_2_analysis.jpeg", plot = ed_f_i_2, path = "/Users/justina/Desktop/Desktop - Justina's MacBook Pro/Aar
hus_Uni/Semester_3/Methods 3/Assignment-3")
```

3. Model choice and training - Model fitting in TIDYMODELS:

```
#Building model specification
LogisticRegression_inf <- logistic_reg() %>%
  set_mode("classification") %>%
  set_engine("glm") %>%
  fit(Diagnosis ~ . , data = d_e_train) #Estimation based on the training set.
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
randomforest_inf <- rand_forest() %>%
  set_mode("classification") %>%
  set_engine("randomForest") %>%
  fit(Diagnosis ~ . , data = d_e_train)

#Train data
res_train <- e_train_s %>%
  as_tibble() %>%
  mutate(
    log_class_train = predict(LogisticRegression_inf, new_data = e_train_s) %>%
      pull(.pred_class),
    log_prob_train = predict(LogisticRegression_inf, new_data = e_train_s, type = "prob") %>%
      pull(.pred_SCZ),
    rf_class_train = predict(randomforest_inf, new_data = e_train_s) %>%
      pull(.pred_class),
    rf_prob_train = predict(randomforest_inf, new_data = e_train_s, type = "prob") %>%
      pull(.pred_SCZ)
  )

metrics(res_train, truth = Diagnosis, estimate = log_class_train) %>%
  knitr::kable()
```

.metric	.estimator	.estimate
accuracy	binary	0.7821192
kap	binary	0.5620210

```
metrics(res_train, truth = Diagnosis, estimate = rf_class_train) %>%
  knitr::kable()
```

.metric	.estimator	.estimate
accuracy	binary	1
kap	binary	1

```
#Test data
res_test <- e_test_s %>%
  as_tibble() %>%
  mutate(
    log_class = predict(LogisticRegression_inf, new_data = e_test_s) %>%
      pull(.pred_class),
    log_prob = predict(LogisticRegression_inf, new_data = e_test_s, type = "prob") %>%
      pull(.pred_SCZ),
    rf_class = predict(randomforest_inf, new_data = e_test_s) %>%
      pull(.pred_class),
    rf_prob = predict(randomforest_inf, new_data = e_test_s, type = "prob") %>%
      pull(.pred_SCZ)
  )

metrics(res_test, truth = Diagnosis, estimate = log_class) %>%
  knitr::kable()
```

.metric	.estimator	.estimate
accuracy	binary	0.7044855
kap	binary	0.4079281

```
metrics(res_test, truth = Diagnosis, estimate = rf_class) %>%
  knitr::kable()
```

.metric	.estimator	.estimate
accuracy	binary	0.6569921
kap	binary	0.3115061

At first, created model with empirical data in brms. Ended up not using it, keeping as a note. Here, used the variables that showed the greatest iimportance in the model, therefore the applying in to training data set resulted in overfitting.

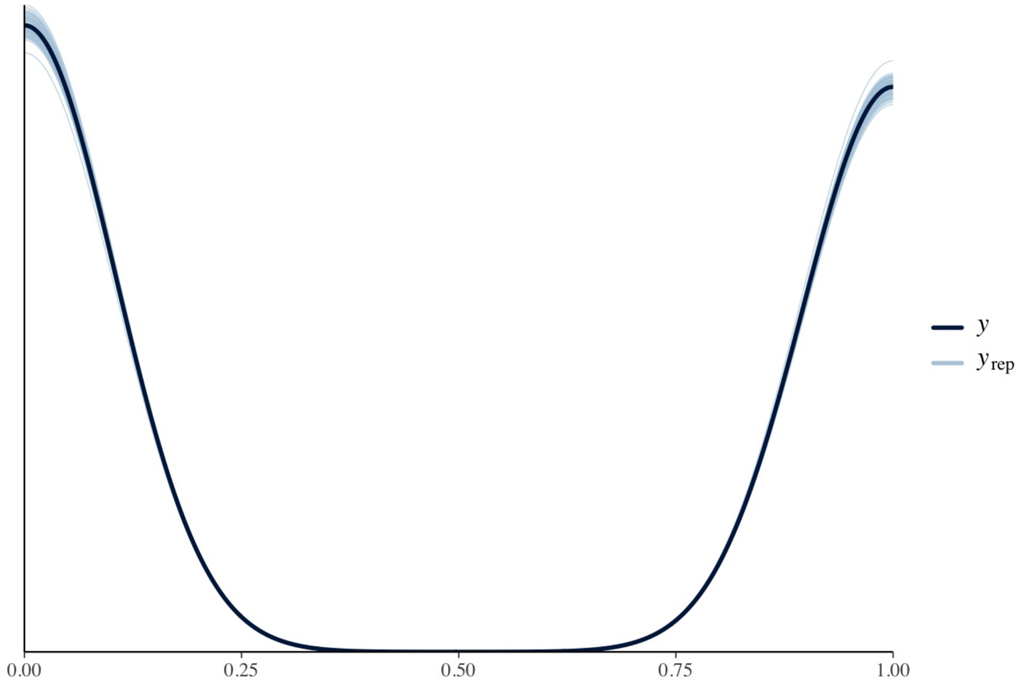
```
#Function
e_df_f <- bf(Diagnosis ~ 1 + MCEP1_Mean + Clarity_Mean + HarmonicProductSpectrum_Mean + MCEP11_Mean + MCEP12_Mean
+ MCEP2_Mean + MCEP9_Mean + MCEP5_Mean + MeanPauseDur_Cova + HMPDM13_MAD + F5_MAD + TurnNumMin_Cova + F0_SD_Praat
+ (1 + MCEP1_Mean + Clarity_Mean + HarmonicProductSpectrum_Mean + MCEP11_Mean + MCEP12_Mean + MCEP2_Mean + MCEP9_
Mean + MCEP5_Mean + MeanPauseDur_Cova + HMPDM13_MAD + F5_MAD + TurnNumMin_Cova+ F0_SD_Praat|NewID))


#get_prior(e_df_f, e_train_s, family = bernoulli)

#Priors to start with:
e_df_p0 <- c(
  prior(normal(0, 1), class = Intercept),
  prior(normal(0, 0.4), class = b),
  prior(normal(0, 0.4), class = sd)
)

#Fitting the model:
e_df_m1 <- brm(
  e_df_f,
  data = e_train_s,
  family = bernoulli,
  prior = e_df_p0,
  sample_prior = T,
  backend = "cmdstanr",
  threads = threading(2),
  chains = 2,
  core = 2,
  control = list(adapt_delt = 0.99, max_treedepth = 20))
```

```
## Start sampling
```

```
## Running MCMC with 2 parallel chains, with 2 thread(s) per chain...
##
## Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 1 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 2 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 1 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 2 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 1 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 2 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 2 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 1 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 2 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 1 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 2 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 1 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 2 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 1 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 2 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 1 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 2 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 1 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 2 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1 finished in 189.4 seconds.
## Chain 2 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2 finished in 190.7 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 190.1 seconds.
## Total execution time: 190.9 seconds.
```

```
pp_check(e_df_m1, ndraws=100) + labs(title = "Posterior-predictive check - empirical data - varying slopes")
```

Posterior-predictive check - empirical data - varying slopes



Calculating the performance of the model: (brms, not included)

```
#Training
e_train_s$PredictionsPerc0 <- predict(e_df_m1)[, 1]
e_train_s$Predictions0[e_train_s$PredictionsPerc0 > 0.5] <- "SCZ"
```

```
## Warning: Unknown or uninitialised column: `Predictions0`.
```

```
e_train_s$Predictions0[e_train_s$PredictionsPerc0 <= 0.5] <- "CT"

e_train_s <- e_train_s %>%
  mutate(
    Diagnosis  = as.factor(Diagnosis),
    Predictions0 = as.factor(Predictions0)
  )


#Testing
e_test_s$PredictionsPerc0 <- predict(e_df_m1, newdata = e_test_s, allow_new_levels = T)[, 1]
e_test_s$Predictions0[e_test_s$PredictionsPerc0 > 0.5] <- "SCZ"
```

```
## Warning: Unknown or uninitialised column: `Predictions0`.
```

```
e_test_s$Predictions0[e_test_s$PredictionsPerc0 <= 0.5] <- "CT"

e_test_s <- e_test_s %>%
  mutate(
    Diagnosis  = as.factor(Diagnosis),
    Predictions0 = as.factor(Predictions0)
  )


#TRAINING DATA
conf_mat(e_train_s, truth = Diagnosis, estimate = Predictions0, dnn = c("Prediction", "Truth"))
```

```
##           Truth
## Prediction  CT SCZ
##        CT  794   0
##        SCZ   0 716
```

```
metrics(e_train_s, truth = Diagnosis, estimate = Predictions0) %>%
  knitr::kable()
```

.metric	.estimator	.estimate
accuracy	binary	1

kap	binary	1

```
#TEST DATA (not scaled)
conf_mat(e_test_s, truth = Diagnosis, estimate = Predictions0, dnn = c("Prediction", "Truth"))
```

```
##           Truth
## Prediction  CT SCZ
##        CT  149 103
##        SCZ  46  81
```

```
metrics(e_test_s, truth = Diagnosis, estimate = Predictions0) %>%
  knitr::kable()
```

.metric	.estimator	.estimate
accuracy	binary	0.6068602
kap	binary	0.2061209

```
#The model performs poorly, therefore I will run the sensitivity analysis on priors, to see whether the prior has
to be adjusted.
```

Plotting the accuracy: (brms, not included)

```
e_df_PerformanceProb <- tibble(expand_grid(
  Sample = seq(1889), #The number of samples I have (?)
  Model = c("VaryingSlope"),
  Setup = c("Empirical_data"),
  Type = c("Training", "Test")
))

#Informed model with all predictors
train_emp <- inv_logit_scaled(posterior_linpred(e_df_m1, summary = F))
test_emp <- inv_logit_scaled(posterior_linpred(e_df_m1, summary = F,
                                            newdata = e_test_s, allow_new_levels = T))

for (i in seq(200)) {
  e_train_s$Predictions0 <- as.factor(ifelse(train_emp[i,]> 0.5, "SCZ", "CT"))
  e_test_s$Predictions0 <- as.factor(ifelse(test_emp[i,]> 0.5, "SCZ", "CT"))

  e_df_PerformanceProb$Accuracy[e_df_PerformanceProb$Sample == i & e_df_PerformanceProb$Model == "VaryingSlope" &
                             e_df_PerformanceProb$Setup == "Empirical_data" & e_df_PerformanceProb$Type == "Train
ing"] <- accuracy(e_train_s, truth = Diagnosis, estimate = Predictions0)[, ".estimate"]

  e_df_PerformanceProb$Accuracy[e_df_PerformanceProb$Sample == i & e_df_PerformanceProb$Model == "VaryingSlope" &
                             e_df_PerformanceProb$Setup == "Empirical_data" & e_df_PerformanceProb$Type == "Test"
] <- accuracy(e_test_s, truth = Diagnosis, estimate = Predictions0)[, ".estimate"]

}

e_df_PerformanceProb <- e_df_PerformanceProb %>%
  mutate(
    Model = as.factor(Model),
    Type = as.factor(Type),
    Setup = as.factor(Setup),
    Accuracy= as.numeric(Accuracy)
  )
```

```
e_df_performance <- ggplot(data=e_df_PerformanceProb, aes(x=Model, y=Accuracy, color = Type)) +
  geom_point(alpha = 0.05) +
  facet_wrap(~Setup) +
  stat_summary(geom = "line", fun = mean)

#A lot of uncertainty in the model. It over fits, classifies test data just at the chance level.
e_df_performance
```

Sensitivity analysis on priors: (brms - not included)

```
#What is the impact of the priors?
#Construct the sequence of sds to loop through for the slope
e_priSD <- seq(0.1, 1.5, length.out = 15) #defining prior confidence
e_priorsN <- e_df_p0 #using mine

#Create empty variables to store output of the loop:
e_PerformanceProb_p <- tibble(expand_grid(
  Sample = seq(4000),
  Prior = e_priSD,
  Setup = c("Empirical_data"),
  Type = c("Training", "Test")
))


for (i in 1:length(e_priSD)) {
  print(i)
  e_priorsN[2,] <- set_prior(paste0("normal(0, ", e_priSD[i], ")"), class = "b")
#model Fixed effects
  model_for_loop <- update(
    e_df_m1,
    prior = e_priorsN,
    sample_prior = T,
    backend = "cmdstanr",
    chains = 2,
    cores = 2,
    iter = 4e3,
    threads = threading(2),
    control = list(adapt_delta = 0.9, max_treedepth = 20)
  )



  train_emp_1 <- inv_logit_scaled(posterior_linpred(model_for_loop, summary = F))
  test_emp_1 <- inv_logit_scaled(posterior_linpred(model_for_loop, summary = F,
                                            newdata = e_test_s, allow_new_levels = T))
  for (j in seq(4000)) {
    e_train_s$Predictions <- as.factor(ifelse(train_emp_1[j,] > 0.5, "SCZ", "CT"))
    e_test_s$Predictions <- as.factor(ifelse(test_emp_1[j,] > 0.5, "SCZ", "CT"))

    e_PerformanceProb_p$Accuracy[e_PerformanceProb_p$Sample == j & e_PerformanceProb_p$Prior == e_priSD[i] &
                              e_PerformanceProb_p$Setup == "Empirical_data" & e_PerformanceProb_p$Type == "Traini
ng"] <- accuracy(e_train_s,  truth = Diagnosis, estimate = Predictions)[, ".estimate"]

    e_PerformanceProb_p$Accuracy[e_PerformanceProb_p$Sample == j & e_PerformanceProb_p$Prior == e_priSD[i] &
                              e_PerformanceProb_p$Setup == "Empirical_data" & e_PerformanceProb_p$Type == "Test"]
<- accuracy(e_test_s, truth = Diagnosis, estimate = Predictions)[, ".estimate"]
  }
}
```

```
e_PerformanceProb_p <- e_PerformanceProb_p %>%
  mutate(
    Type = as.factor(Type),
    Setup = as.factor(Setup),
    Prior = as.numeric(Prior),
    Accuracy= as.numeric(Accuracy)
  )

ggplot(data=e_PerformanceProb_p, aes(x=Prior, y=Accuracy, color = Type)) +
  geom_point(alpha = 0.05) +
  geom_hline(yintercept = 0.5) +
  ggtitle("Sensitivity analysis  (model with varying slopes)")
```