Assignment 2 - Language Development in ASD - Making predictions

Søren Orm Hansen

September, 2020

Welcome to the second exciting part of the Language Development in ASD exercise

In this exercise we will delve more in depth with different practices of model comparison and model selection, by first evaluating your models from last time against some new data. Does the model generalize well? Then we will learn to do better by cross-validating models and systematically compare them.

The questions to be answered (in a separate document) are: 1- Discuss the differences in performance of your model in training and testing data 2- Which individual differences should be included in a model that maximizes your ability to explain/predict new data? 3- Predict a new kid's performance (let's call him Bernie) and discuss it against expected performance of the two groups

Learning objectives

- Critically appraise the predictive framework (contrasted to the explanatory framework)
- Learn the basics of machine learning workflows: training/testing, cross-validation, feature selections

Let's go

N.B. There are several datasets for this exercise, so pay attention to which one you are using!

1. The (training) dataset from last time (the awesome one you produced :-)).
2. The (test) datasets on which you can test the models from last time:

- Demographic and clinical data: https://www.dropbox.com/s/5pc05mh5jwvdfjk/demo_test.csv?dl=0
- Utterance Length data: https://www.dropbox.com/s/eegu8fea2entdqv/LU_test.csv?dl=0
- Word data: https://www.dropbox.com/s/cf4p84mzn2p1bev/token_test.csv?dl=0

Exercise 1) Testing model performance

How did your models from last time perform? In this exercise you have to compare the results on the training data and on the test data. Report both of them. Compare them. Discuss why they are different.

- recreate the models you chose last time (just write the model code again and apply it to your training data (from the first assignment))
- calculate performance of the model on the training data: root mean square error is a good measure. (Tip: google the function rmse())
- create the test dataset (apply the code from assignment 1 to clean up the 3 test datasets)
- test the performance of the models on the test data (Tips: google the functions "predict()")
- optional: predictions are never certain, can you identify the uncertainty of the predictions? (e.g. google predictinterval())

```r
# Loading packages
library(pacman)
pacman::p_load(readr, lmerTest, Metrics, caret, merTools, tidyverse, ggpubr)
```

```r
# Load test data
LU_test <- read_csv('LU_test.csv')
```

```
## Parsed with column specification:
## cols(
##   SUBJ = col_character(),
##   VISIT = col_character(),
##   MOT_MLU = col_double(),
##   MOT_LUstd = col_double(),
##   MOT_LU_q1 = col_double(),
##   MOT_LU_q2 = col_double(),
##   MOT_LU_q3 = col_double(),
##   CHI_MLU = col_double(),
##   CHI_LUstd = col_double(),
##   CHI_LU_q1 = col_double(),
##   CHI_LU_q2 = col_double(),
##   CHI_LU_q3 = col_double()
## )
```

```r
token_test <- read_csv('token_test.csv')
```

```
## Parsed with column specification:
## cols(
##   SUBJ = col_character(),
##   VISIT = col_character(),
##   types_MOT = col_double(),
##   types_CHI = col_double(),
##   types_shared = col_double(),
##   tokens_MOT = col_double(),
##   tokens_CHI = col_double(),
##   X = col_logical()
## )
```

```r
demo_test <- read_csv('demo_test.csv')
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Child.ID = col_character(),
##   Ethnicity = col_character(),
##   Diagnosis = col_character(),
##   Birthdate = col_character()
## )

## See spec(...) for full column specifications.
```

```r
## Clean up function, included to inspire you
CleanUpData <- function(Demo, LU, Word) {
  Speech <- merge(LU, Word) %>%
    rename(ID = SUBJ) %>%
    mutate(VISIT = as.numeric(str_extract(VISIT, "\\d")),
           ID = gsub("\\.", "", ID)) %>%
    dplyr::select(ID,
                  VISIT,
                  MOT_MLU,
                  CHI_MLU,
                  types_MOT,
                  types_CHI,
                  tokens_MOT,
                  tokens_CHI)

  Demo <- Demo %>%
    dplyr::select(
      Child.ID,
      Visit,
      Ethnicity,
      Diagnosis,
      Gender,
      Age,
      ADOS,
      MullenRaw,
      ExpressiveLangRaw,
      Socialization
    ) %>% rename(ID = Child.ID, VISIT = Visit) %>%
    mutate(ID = gsub("\\.", "", ID))

  Data = merge(Demo, Speech, all = T)

  Data1 = Data %>%
    subset(VISIT == "1") %>%
    dplyr::select(ID, ADOS, ExpressiveLangRaw, MullenRaw, Socialization) %>%
    rename(
      ADOS1 = ADOS,
      vIQ1 = ExpressiveLangRaw,
      nvIQ1 = MullenRaw,
      Socialization1 = Socialization
    )

  Data = merge(Data, Data1, all = T) %>%
    mutate(
      ID = as.numeric(as.factor(as.character(ID))),
      VISIT = as.numeric(as.character(VISIT)),
      Gender = recode(Gender,
                      "1" = "M",
                      "2" = "F"),
      Diagnosis = recode(Diagnosis,
                         "A"  = "ASD",
                         "B"  = "TD")
    )
```

```
  return(Data)
}
```

```
# Load training Data
df_train <- read_csv('cleanedData.csv')
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Diagnosis = col_character(),
##   Ethnicity = col_character(),
##   Gender = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
# recreate the models you chose last time (just write the code again and apply it to Train Data)
m <- lmer(CHI_MLU ~
            Diagnosis * VISIT + vIQ1 +
            (1 + VISIT | ID),
          df_train, REML = F)

gcm <- lmer(CHI_MLU ~
              I(VISIT)^2 * Diagnosis + vIQ1 +
              (1 | ID),
             df_train, REML = F)

#- calculate performance of the model on the training data: root mean square error is a good measure. (

# Subsetting interesting variables and omiting na's + predicting the values using the predict function
sub_df_train <- df_train %>%
  dplyr::select(c(ID, CHI_MLU, VISIT, vIQ1, Diagnosis)) %>%
  na.omit() %>%
  mutate(pred_CHI_MLU_m = fitted(m),
         pred_CHI_MLU_gcm = fitted(gcm))

rmse_m <- rmse(sub_df_train$CHI_MLU, sub_df_train$pred_CHI_MLU_m) # rmse = 0.355
rmse_gcm <- rmse(sub_df_train$CHI_MLU, sub_df_train$pred_CHI_MLU_gcm) # rmse = 0.416
```

```
#- create the test dataset (apply the code from assignment 1 or my function to clean up the 3 test data
# Test data

df_test <- CleanUpData(demo_test, LU_test, token_test)

#- test the performance of the models on the test data (Tips: google the functions "predict()")

# Subsetting interesting variables and omitting NAs and predicting the values using the predict functio
sub_df_test <- df_test  %>%
  dplyr::select(c(ID, CHI_MLU, VISIT, vIQ1, Diagnosis)) %>%
  na.omit()

sub_df_test <- sub_df_test %>%
```

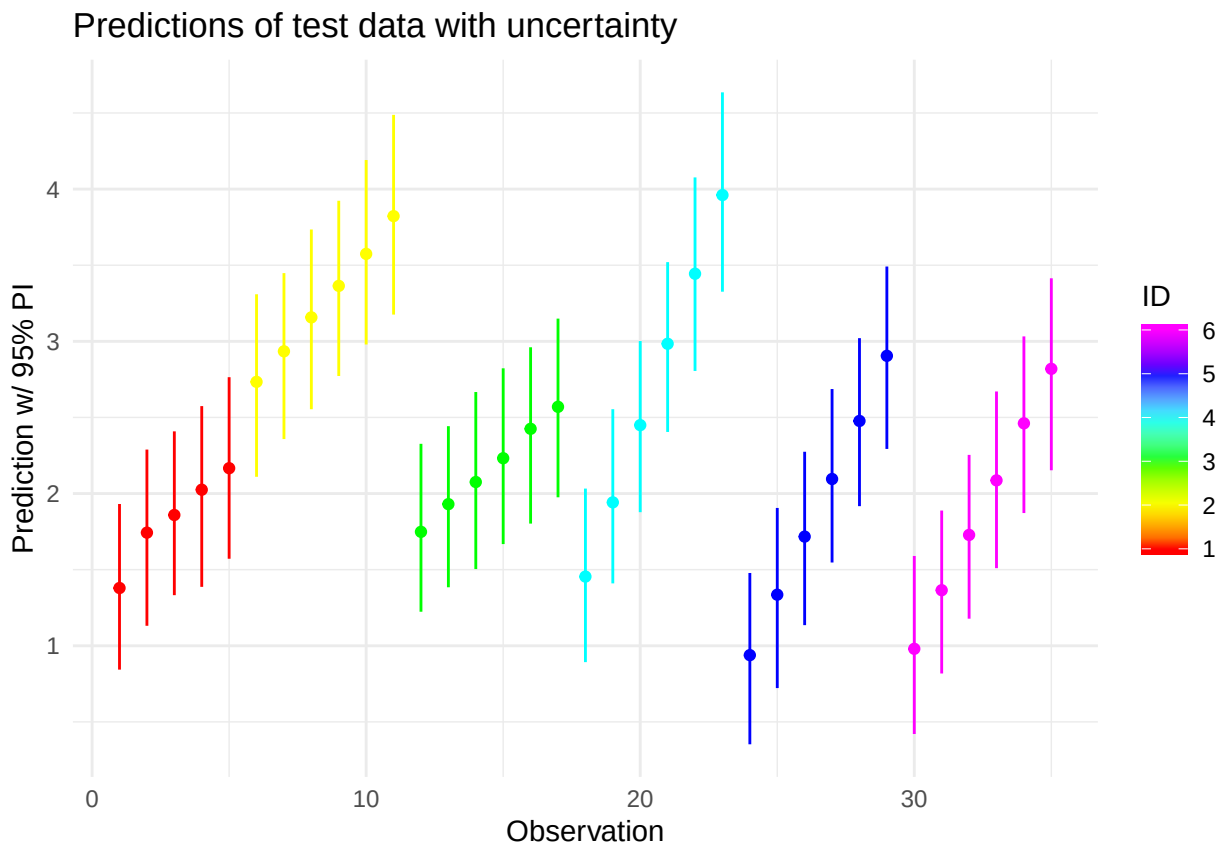
```
  mutate(pred_CHI_MLU_m = predict(m, newdata = sub_df_test),
         pred_CHI_MLU_gcm = predict(gcm, newdata = sub_df_test))



rmse(sub_df_test$CHI_MLU, sub_df_test$pred_CHI_MLU_m) # rmse = 0.726
```

```
## [1] 0.7256848
```

```
rmse(sub_df_test$CHI_MLU, sub_df_test$pred_CHI_MLU_gcm) # rmse = 0.703
```

```
## [1] 0.7027163
```

```
#- optional: predictions are never certain, can you identify the uncertainty of the predictions? (e.g. 

intFit <- merTools::predictInterval(m, newdata = sub_df_test) %>%
  mutate(ID = sub_df_test$ID, obs = 1:35)

intFit %>%
  ggplot(aes( x = obs, y = fit, ymin = lwr, ymax = upr, color = ID)) +
  geom_point() +
  geom_linerange() +
  scale_color_gradientn(colours = rainbow(6)) +
  labs(x = "Observation", y = "Prediction w/ 95% PI", title = 'Predictions of test data with uncertainty
  theme_minimal()
```



[HERE GOES YOUR ANSWER]

Exercise 2) Model Selection via Cross-validation (N.B: ChildMLU!)

One way to reduce bad surprises when testing a model on new data is to train the model via cross-validation.

In this exercise you have to use cross-validation to calculate the predictive error of your models and use this predictive error to select the best possible model.

- Use cross-validation to compare your model from last week with the basic model (Child MLU as a function of Time and Diagnosis, and don't forget the random effects!)
- (Tips): google the function "createFolds"; loop through each fold, train both models on the other folds and test them on the fold)
- Now try to find the best possible predictive model of ChildMLU, that is, the one that produces the best cross-validated results.
- Bonus Question 1: What is the effect of changing the number of folds? Can you plot RMSE as a function of number of folds?
- Bonus Question 2: compare the cross-validated predictive error against the actual predictive error on the test data

```
#- Create the basic model of ChildMLU as a function of Time and Diagnosis (don't forget the random effe

# Train ID ends at 66 - test ID should start at 67
df_test$ID <- df_test$ID + 66

# Making a df with both the train and test data
df_tot <- merge(df_test, df_train, all = T)

# Make a cross-validated version of the model. (Tips: google the function "createFolds";  loop through

# Subsetting the total df with only the values we use in the models
sub_df1 <- df_tot %>%
  dplyr::select(c(ID, CHI_MLU, VISIT, Diagnosis, vIQ1)) %>%
  na.omit()

# We are going to be folding a lot - let's make a function with model syntax as the input, so we can lo
folding <- function(syntax){
  k = 10 # number of folds
  folds <- createFolds(unique(sub_df1$ID), k = k, list = TRUE, returnTrain = FALSE) # creating the fold
  trainRMSE <- rep(NA, k) # preparing slot for saving RMSE's for the training folds
  testRMSE <- rep(NA, k) # preparing slot for saving RMSE's for the testing folds
  AIC = rep(NA, k) # preparing slot for saving AIC's for the models build on the training folds

i = 1 # creating value for indexing the different loops
for (fold in folds) {
  train = subset(sub_df1,!(ID %in% fold)) # subsetting the training data for the folds
  test = subset(sub_df1, ID %in% fold) # subsetting the test data for the folds
  model = lmer(syntax, train, REML = F) # defining the model
  test$prediction = predict(model, test, allow.new.levels = TRUE) # predicting new values for test data
  train$prediction = fitted(model) # predicting new values for training data
  trainRMSE[i] = rmse(train$CHI_MLU, fitted(model)) # saving RMSE value for the training data at index
```

```
  testRMSE[i] = rmse(test$CHI_MLU, test$prediction) # saving RMSE value for the test data at index i
  AIC[i] = summary(model)[["AICtab"]][["AIC"]] # saving AIC value for the model at index i
  i = i + 1
}
  AIC_mean = mean(AIC) # finding the AIC mean
  m_train = mean(trainRMSE) # finding the RMSE mean for the training data
  m_test = mean(testRMSE) # finding the RMSE mean for the test data
  DiffTrainTest = m_train - m_test  # finding the difference in the RMSE mean for the training and the
return(c(AIC_mean, m_train, m_test, DiffTrainTest)) # specifying what we want the function to return
}
```

```
# saving the syntax of our models
m1 <-  CHI_MLU ~ Diagnosis + VISIT + vIQ1 + (1 + VISIT | ID)
m2 <-  CHI_MLU ~ Diagnosis * VISIT + vIQ1 + (1 + VISIT | ID)

gcm1 <- CHI_MLU ~ I(VISIT^2) + Diagnosis + vIQ1 + (1 | ID)
gcm2 <- CHI_MLU ~ I(VISIT^2) * Diagnosis + vIQ1 + (1 | ID)
#gcm3 <- CHI_MLU ~ I(VISIT^2) * Diagnosis + vIQ1 + (1 + I(VISIT^2) | ID) # doesn't converge

# storing the syntax and names of our models so we can loop through them
models <- c(m1, m2, gcm1, gcm2)
names <- c('m1', 'm2', 'gcm1', 'gcm2')

# creating place holder df for the loop
obj <- matrix(0, ncol = 5, nrow = 4) %>% data.frame()

# chaning the col-names in our place holder df
colnames(obj)[1] <- 'ModelNames'
colnames(obj)[2] <- 'AICMean'
colnames(obj)[3] <- 'TrainMean'
colnames(obj)[4] <- 'TestMean'
colnames(obj)[5] <- 'DiffTrainTest'

# creating a place holder for the mean scores for looping through the folding function 20 times
m_scores <- matrix(0, ncol = 20, nrow = 1) %>% data.frame()
```

```
# Now try to find the best possible predictive model of ChildMLU, that is, the one that produces the be

# looping through the folding function 20 times
for(j in 1:20) {

i = 1 # indexing at i

for (model in models) {
  obj[i, 1] <- names[i] # saving the name of the models
  obj[i, 2:5] <- folding(model) # using the folding function to find the AIC, RMSE means for training a
  i = i + 1 # increasing index by 1
}

# arranging absolute values in ascending order and scoring from 1 to 4 points for AIC, RMSE means for t
obj <- obj %>% arrange(abs(TestMean))
obj$ScoreTestMean <- 1:4
```

```
obj <- obj %>% arrange(abs(TrainMean))
obj$ScoreTrainMean <- 1:4

obj <- obj %>% arrange(abs(DiffTrainTest))
obj$ScoreDiffTrainTest <- 1:4

obj <- obj %>% arrange(abs(AICMean))
obj$ScoreAIC <- 1:4

# finding the sum of the points and arranging by that sum
obj <-
  obj %>% group_by(ModelNames) %>% mutate(Score = sum(ScoreTestMean, ScoreTrainMean, ScoreDiffTrainTest
obj <- obj %>% arrange(abs(Score))

# saving only the lowest scoring model (the best model) from each loop
m_scores[1, j] <- obj[1, 1]
}
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00337227
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00405901
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.0247463
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00297522
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.0027934
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00923066
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00414301
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.003565
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00234479
## (tol = 0.002, component 1)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00209179
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00224801
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00235325
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00385805
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00490528
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00446895
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00258281
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00561516
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00481772
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00206017
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00380302
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00202256
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.002577
## (tol = 0.002, component 1)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.010735
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.0024337
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00617312
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.0573107
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.002442
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.0151659
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.0111689
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00204151
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00285043
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00598182
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00228651
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00923879
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00290326
## (tol = 0.002, component 1)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00263272
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00630981
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00381879
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.0052406
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00502216
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.0101767
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00278353
## (tol = 0.002, component 1)

## boundary (singular) fit: see ?isSingular

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00574558
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00288027
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00282176
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00329408
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00392078
## (tol = 0.002, component 1)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00560114
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00211069
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.0023203
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.0025501
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00246258
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00531509
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00336589
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00392719
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00240073
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00544847
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00245823
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00352073
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00698483
## (tol = 0.002, component 1)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00317992
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.0101683
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00617716
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00290471
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00231642
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00523138
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00289653
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00375008
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00234866
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00522099
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00418683
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.0393542
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00242289
## (tol = 0.002, component 1)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.0205732
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00896169
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00210413
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00368448
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00218866
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00464824
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00253141
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00947936
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00256551
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00251833
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00821074
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00385192
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00212619
## (tol = 0.002, component 1)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00203886
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00244989
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00428752
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00216856
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00497111
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00713923
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00275921
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00598757
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00276987
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00469564
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00204331
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00254156
## (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00603546
## (tol = 0.002, component 1)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| = 0.00251377
## (tol = 0.002, component 1)
```

```
# finding the counts of different 'winning' models
table(unlist(m_scores))
```

```
##
## m2
## 20
```

```
# Bonus Question 1: What is the effect of changing the number of folds? Can you plot RMSE as a function

# Bonus Question 2: compare the cross-validated predictive error against the actual predictive error on
```

```
# building the best model
m2 <-  lmer(CHI_MLU ~ Diagnosis * VISIT + vIQ1 +
  (1 + VISIT | ID), df_tot, REML = F)

# summary of the best model
summary(m2)
```

```
## Linear mixed model fit by maximum likelihood . t-tests use
##   Satterthwaite's method [lmerModLmerTest]
## Formula: CHI_MLU ~ Diagnosis * VISIT + vIQ1 + (1 + VISIT | ID)
##    Data: df_tot
##
##      AIC      BIC   logLik deviance df.resid
##    555.7    591.3   -268.8    537.7      378
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -2.64140 -0.57530 -0.03791  0.46086  3.01485
##
## Random effects:
##  Groups   Name        Variance Std.Dev. Corr
##  ID       (Intercept) 0.09581  0.3095
##           VISIT       0.01111  0.1054   -0.44
##  Residual             0.15698  0.3962
## Number of obs: 387, groups:  ID, 67
##
## Fixed effects:
##                     Estimate Std. Error        df t value Pr(>|t|)
## (Intercept)       -0.008686   0.151819 82.324412  -0.057 0.954516
## DiagnosisTD       -0.402631   0.119972 65.883472  -3.356 0.001316 **
## VISIT              0.100949   0.025307 67.557921   3.989 0.000166 ***
## vIQ1               0.075627   0.007154 67.258195  10.571 6.15e-16 ***
## DiagnosisTD:VISIT  0.251747   0.035107 67.999711   7.171 6.98e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
```

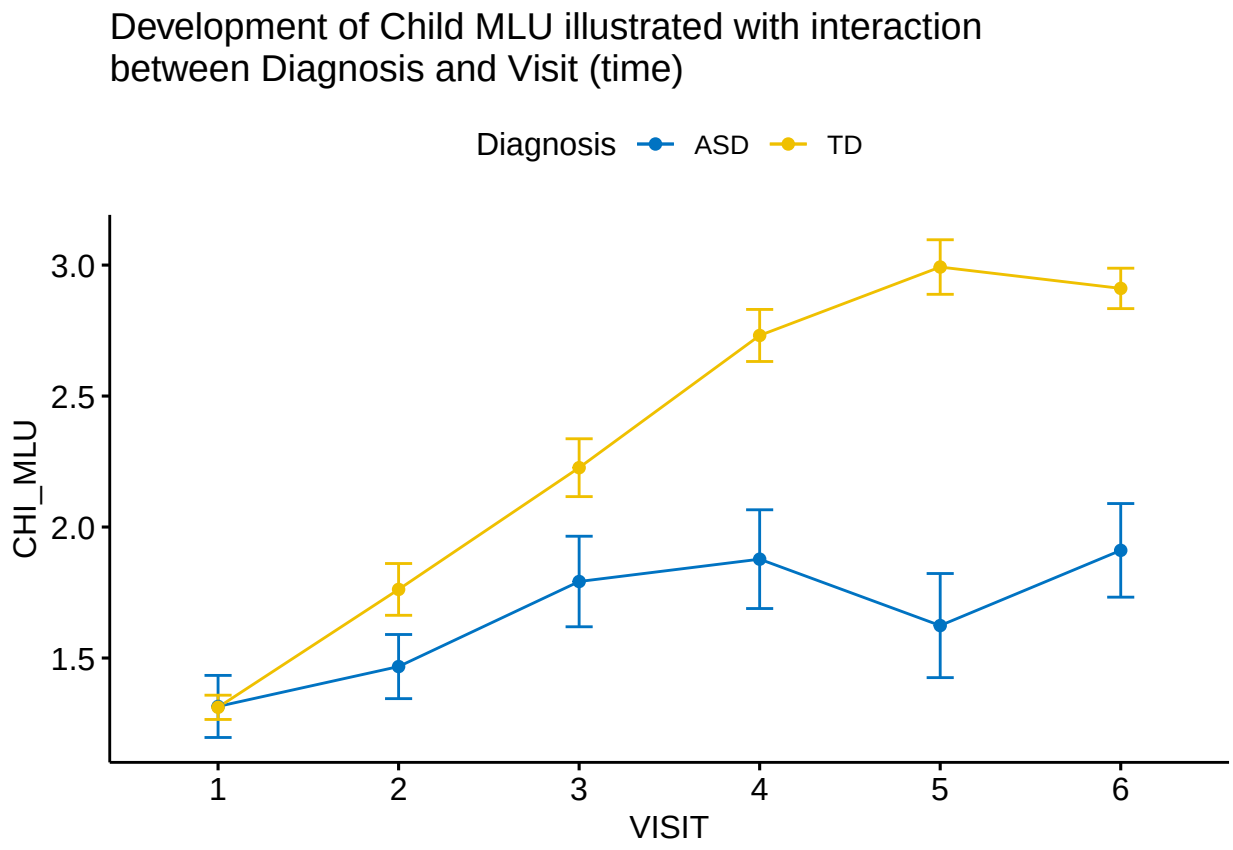
```
##             (Intr) DgnsTD VISIT  vIQ1
## DiagnosisTD -0.289
## VISIT       -0.376  0.482
## vIQ1        -0.825 -0.141 -0.005
## DgnTD:VISIT  0.272 -0.668 -0.721  0.003
```

```
# plotting the interaction
l <- ggline(df_tot,
       x = "VISIT",
       y = "CHI_MLU",
       col = "Diagnosis",
       add = c("mean_se", "dodge"),
       palette = "jco") + ggtitle("Development of Child MLU illustrated with interaction
between Diagnosis and Visit (time)")
```

```
## Warning: `group_by_()` is deprecated as of dplyr 0.7.0.
## Please use `group_by()` instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
l
```

```
## Warning: Removed 21 rows containing non-finite values (stat_summary).
```



[HERE GOES YOUR ANSWER]

Exercise 3) Assessing the single child

Let's get to business. This new kiddo - Bernie - has entered your clinic. This child has to be assessed according to his group's average and his expected development.

Bernie is one of the six kids in the test dataset, so make sure to extract that child alone for the following analysis.

You want to evaluate:

- how does the child fare in ChildMLU compared to the average TD child at each visit? Define the distance in terms of absolute difference between this Child and the average TD.
- how does the child fare compared to the model predictions at Visit 6? Is the child below or above expectations? (tip: use the predict() function on Bernie's data only and compare the prediction with the actual performance of the child)

```
#cleaning up data without removing names
CleanUpData <- function(Demo, LU, Word) {
  Speech <- merge(LU, Word) %>%
    rename(ID = SUBJ) %>%
    mutate(VISIT = as.numeric(str_extract(VISIT, "\\d")), ID = gsub("\\.", "", ID)) %>%
    dplyr::select(ID,
                  VISIT,
                  MOT_MLU,
                  CHI_MLU,
                  types_MOT,
                  types_CHI,
                  tokens_MOT,
                  tokens_CHI)

  Demo <- Demo %>%
    dplyr::select(
      Child.ID,
      Visit,
      Ethnicity,
      Diagnosis,
      Gender,
      Age,
      ADOS,
      MullenRaw,
      ExpressiveLangRaw,
      Socialization
    ) %>% rename(ID = Child.ID, VISIT = Visit) %>%
    mutate(ID = gsub("\\.", "", ID))

  Data = merge(Demo, Speech, all = T)

  Data1 = Data %>%
    subset(VISIT == "1") %>%
    dplyr::select(ID, ADOS, ExpressiveLangRaw, MullenRaw, Socialization) %>%
    rename(
      ADOS1 = ADOS,
      vIQ1 = ExpressiveLangRaw,
      nvIQ1 = MullenRaw,
```

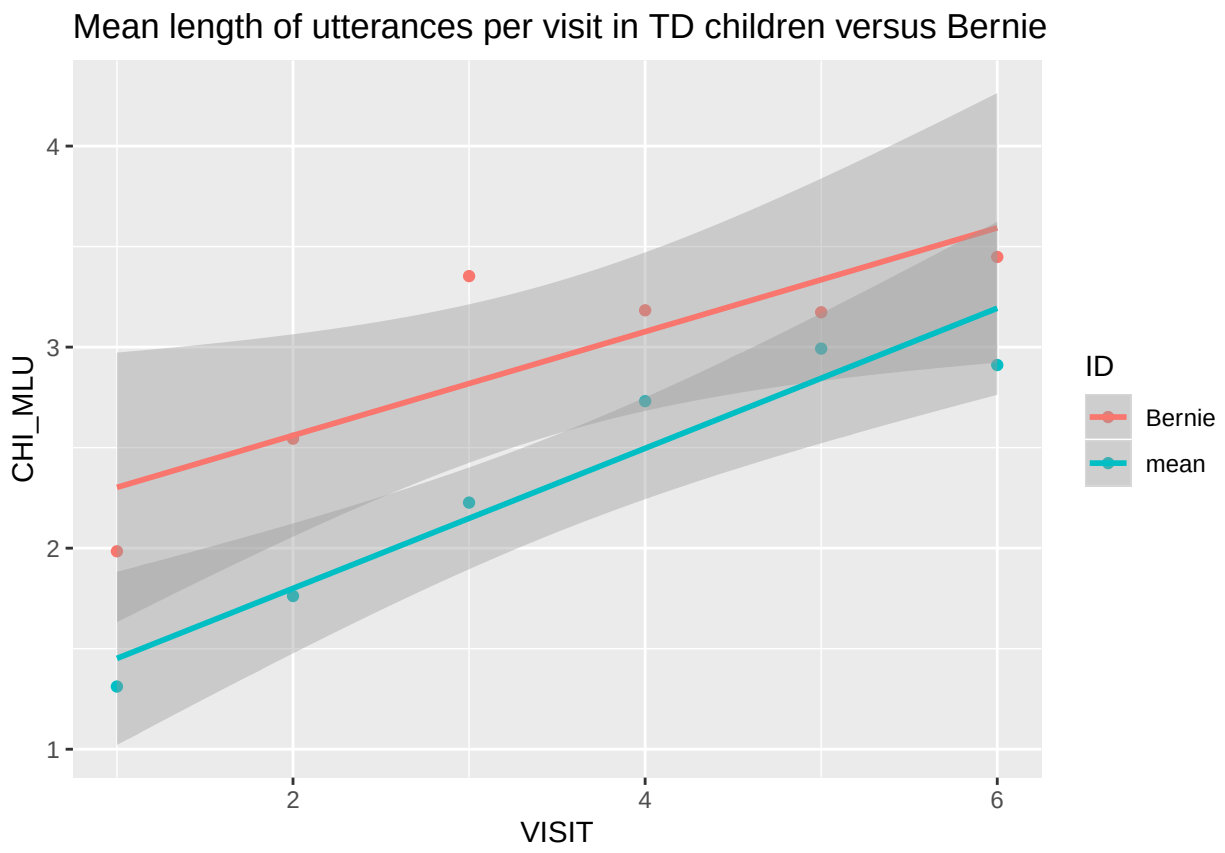
```
      Socialization1 = Socialization
    )

  Data = merge(Data, Data1, all = T) %>%
    mutate(
      VISIT = as.numeric(as.character(VISIT)),
      Gender = recode(Gender,
                      "1" = "M",
                      "2" = "F"),
      Diagnosis = recode(Diagnosis,
                         "A"  = "ASD",
                         "B"  = "TD")
    )

  return(Data)
}
```

```
df_test_bernie <- CleanUpData(demo_test, LU_test, token_test)

#extracting Bernie
bernie <- df_test_bernie %>% filter(ID == "Bernie")
bernie1 <- bernie

#df without Bernie
notbernie <- df_test_bernie %>% filter(ID != "Bernie")

#merging one big dataset without Bernie
bernie_df <- merge(notbernie, df_train, all = T)

#extracting only TD children
bernie_df <- bernie_df %>% filter(Diagnosis == "TD")

#how does the child fare in ChildMLU compared to the average TD child at each visit? Define the distanc

#dataframe with means per visit of MLU
MLUpervisit <-
  bernie_df %>%
  group_by(VISIT) %>%
  summarise(mean(CHI_MLU, na.rm = T))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
MLUpervisit$ID <- ("mean")

colnames(MLUpervisit)[2] <- "CHI_MLU"

#extracting relevant columns Bernie
bernie1 <- bernie %>% dplyr::select(ID, CHI_MLU, VISIT, Diagnosis, vIQ1)
bernie <- bernie %>% dplyr::select(ID, CHI_MLU, VISIT)

comparativeMLU <- rbind(MLUpervisit, bernie)

#PLOTTING!
```

```r
ggplot(comparativeMLU, aes(x = VISIT, y = CHI_MLU, color = ID)) +
  geom_point() +
  geom_smooth(method = lm) +
  ggtitle("Mean length of utterances per visit in TD children versus Bernie")
```




```r
#making a dataframe to subtract and compare mean values per visit
comparativeMLU <- cbind(MLUpervisit, bernie)
colnames(comparativeMLU) <-
  c("VISIT_mean",
    "MEAN_MLU",
    "mean_ID",
    "Bernie_ID",
    "Bernie_MLU",
    "VISIT_bernie")

comparativeMLU$MEAN_MLU <- as.numeric(comparativeMLU$MEAN_MLU)
comparativeMLU$Bernie_MLU <- as.numeric(comparativeMLU$Bernie_MLU)

#subtracting the MLUs to compare
comparativeMLU <-
  comparativeMLU %>% mutate(difference = Bernie_MLU - MEAN_MLU)

#2

#Filtering only visit 6 for Bernie
```

```
bernievisit6 <- bernie1 %>% filter(VISIT == 6)

#predicting MLU value for Bernie at visit 6
predict(m, bernievisit6, allow.new.levels = T)
```

```
##       1
## 2.71095
```

```
#general rmse predicition for all visits, just for funs
berniepredictions <- predict(m, bernie1, allow.new.levels = T)
rmse(bernie1$CHI_MLU, berniepredictions)
```

```
## [1] 0.6198079
```

[HERE GOES YOUR ANSWER]

OPTIONAL: Exercise 4) Model Selection via Information Criteria

Another way to reduce the bad surprises when testing a model on new data is to pay close attention to the relative information criteria between the models you are comparing. Let's learn how to do that!

Re-create a selection of possible models explaining ChildMLU (the ones you tested for exercise 2, but now trained on the full dataset and not cross-validated).

Then try to find the best possible predictive model of ChildMLU, that is, the one that produces the lowest information criterion.

- Bonus question for the optional exercise: are information criteria correlated with cross-validated RMSE? That is, if you take AIC for Model 1, Model 2 and Model 3, do they co-vary with their cross-validated RMSE?

OPTIONAL: Exercise 5): Using Lasso for model selection

Welcome to the last secret exercise. If you have already solved the previous exercises, and still there's not enough for you, you can expand your expertise by learning about penalizations. Check out this tutorial: http://machinelearningmastery.com/penalized-regression-in-r/ and make sure to google what penalization is, with a focus on L1 and L2-norms. Then try them on your data!