Techniques for Variable Selection for Psychologists - Random Forest and LASSO

Francisco N. Ramos

Random Forest for Regression for Psychologists in R - An Example

In this document, I will walk through a random forest variable selection procedure for a psychological dataset. The dataset is extremely characteristic of many psychology datasets - it has an okay but not at all large sample size (735 N before being split into test/training sets), a relatively high number of potential predictors (20+ collected as part of the survey, many of which are likely completely unhelpful in prediction), and what is likely middling to low effect sizes for predictors. The focus of the model will be regression and we will be primarily discussing model error and predictive power, and alternating focuses between the two. We'll review a few different ways of creating models, selecting features, and evaluating models.

However, be aware that this is not a tutorial on identifying the absolute best predictive model, which is described ad infinitum elsewhere. I will touch on using algorithmic methods to identify the "optimal" subset of predictors in a full model, but if you are merely looking for "the best model to fit my data" or "the best way to predict outcome from a set of predictors", this document will likely not be the most helpful.

ANother note: we will be partially demonstrating what many in the machine learning field would call "recursive feature selection", and what many statisticians would call "stepwise selection", "backwards selection" to be specific. Let me just be clear - backwards selection is NEVER the best way to do variable selection. The best feature selection method will always be expert knowledge on which variables are likely to explain the majority of the variance/signal as well as which less-important variables are likely to provide valuable interactions.If possible, you should either a priori identify the subset of variables to evaluate OR try to use automated methods with clear "reward" and "loss" functions so you better understand why it has penalized what it penalized and why it has selected what it selected. Backwards selection has the nasty little habit of occasionally overemphasizing variables that actually have little relationship with the DV because they happen to "fit in" nicely with the other variables, making the variable appear important in the specific models backwards selection evaluates, but the variable may be unimportant in the "true" best model for prediction (or the model containing the "true" subset of actually important variables, if you believe in that). We are going through

RFE because it is common but more importantly because it does help you better understand the relationships between variables in the specific model, and it is easy to run an automated method without really thinking about why those variables have been selected and what it is showing you. An example of one potential automated method is provided before the end of this document, however.

I have not seen the data ahead of time. The analyses described hereafter are entirely novel.

The Data

Importantly, we are conducting a regression task, not a classification task, as our dependent variable is continuous. The main task we'll be attempting to address is the following - we would like to predict arithmetic performance score in German students from a set of predictor variables including age, sex, self-reported math anxiety, self-reported math self-concept, and others. Our second main task is that we would like to identify which of these variables are actually useful in predicting arithmetic performance and which ones are not, so future researchers do not have to waste resources collecting unimportant variables for minimal improvement in prediction. In other words, we would like a concise model. This data was originally gathered and shared by Cipora et al. (2024). References are at the end of the document.

Since the dataset contains over 40 potential predictor variables and only around 735 N pre-training/test split, I have selected ten variables from the dataset at pseudo-random for our analysis. For the purposes of this document, only these ten variables "exist". Below, I will list the ten variables we are interested in studying as predictors of arithmetic performance score followed by the syntax for their object in R. All assessments were in German.

Dependent Variable:
Arithmetic performance/sum_arith_perf., as measured by "the number of correctly solved problems in order as instructed" on a simple arithmetic speed test.

Predictors:
Age/age, as measured in years

Sex/sex, where 1 = male, 2 = female, and 3=other. Participants who ignored this question were removed.

Neuroticism/score_BFI_N, as measured by the sum score of the 8 items of the Big Five Inventory (short version) pertaining to neuroticism.

Math anxiety/score_AMAS_total, as measured by the sum score on the Abbreviated Math Anxiety Scale.

General trait anxiety/score_GAD, as measured by sum score on the Generalized Anxiety Disorder Screener (GAD-7).

Math self-efficacy/score_PISA_ME, as measured in the PISA 2012 study using the sum score of six items.

General state anxiety/score_STAI_state_short, as assessed by the sum of the five-item scale STAI-SKD.

Test anxiety/score_TAI_short, as measured by the sum score of the 5 items on the short version of the Test Anxiety Inventory.

Math self-concept/score_SDQ_M, as measured by the sum score of the four math-related statements on the Self-Description Questionnaire III. Evaluates variables such as one's comfort/enjoyment/pride with math, whereas self-efficacy evaluates one's self-confidence in math abilities.

Language self-concept/score_SDQ_L, as measured by the sum score of the four language-related statements on the Self-Description Questionnaire III.

Cleaning the Data

We'll start with importing and cleaning the data to make sure it fits our task.

```
library(here)

library(tidyverse)
library(caret)
library(readxl)
library(readr)
library(ggplot2)
library(randomForest)
library(stats)
```

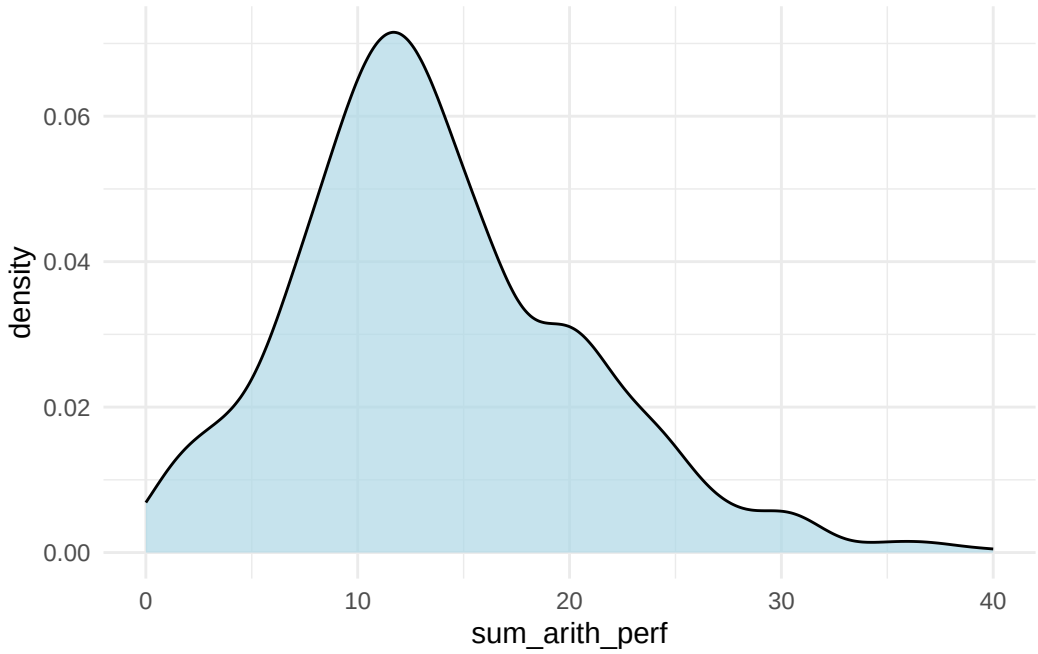
The Here function is extremely convenient for creating paths and directories that won't break when files are moved or when you reproduce the work on another computer. "Here" on your computer will be whatever the top level folder of the current project folder is (which you set up when initializing a project). Looks like "here" for me is /Users/frankie/Desktop/Dr. Lai Feature Selection Project 9.13.24/Main Script You can also set it using the set_here function. You can specify subfolders using a comma, then quotation marks to define the name of each folder or file.You don't even need to worry about working directories if you use the here function. Yay!

Below, we'll again use the here function to tell the readr function where to go to get the dataset. Our data was stored in the Main Script folder, in a subfolder called OSF archive, and named AMATUS_dataset.csv. Use that info to read the code below. Notice how we use a normalizePath function to convert relative or incomplete path names to absolute path names. That way, we don't have to worry too much about Mac vs. Windows or cloud vs. local.

```r
file_path <- normalizePath(here::here("OSF archive/AMATUS_dataset.csv"))
# adjust as needed.

amatus <- read_csv2(file_path, c("", "NA"), col_names = TRUE)
## i Using "','" as decimal and "'.'" as grouping mark. Use `read_delim()` for more control.
# it's already in the "working directory" of the main script folder,
# so we only need the subfolder OSF archive and the actual name of the dataset.

View(amatus)
# We can use this command to view our data.
table(amatus$sum_arith_perf)
##
##  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
##  4 15 12 18 15 19 24 40 40 53 62 68 71 64 57 45 42 37 18 29 36 25 23 16 18 16
## 26 27 28 29 30 31 32 34 35 36 37 38 40
##  6  8  4  3  8  6  2  1  2  1  2  1  1

#### Just some regular old data cleaning to start with.
amatus$sex <- as.factor(amatus$sex)
amatus$age_range <- as.factor(amatus$age_range)
amatus$breaks <- as.factor(amatus$breaks)
amatus$honesty <- as.factor(amatus$honesty)
amatus$native_speaker <- as.factor(amatus$native_speaker)
amatus$noise <- as.factor(amatus$noise)

amatusclean <- amatus[!is.na(amatus$sum_arith_perf), ] # removing the
# individuals who did not complete the performance test in order as instructed
amatusclean <- amatusclean[!(amatusclean$sample %in%
c("german_teachers", "belgian_teachers")), ]

# removing the other two samples from the dataset
View(amatusclean)
```

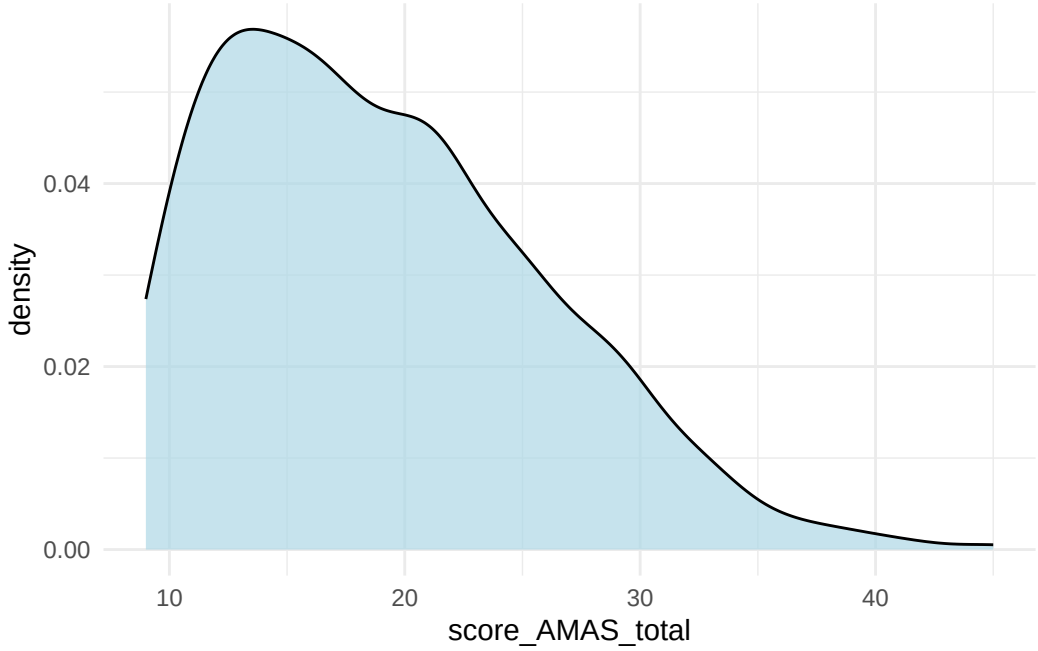
Now, we'll just check out a quick histogram of the dependent variable, arithmetic math performance.

```r
ggplot(
  amatus,
  aes(x = sum_arith_perf)
) +
  geom_density(fill = "lightblue", alpha = 0.7) +
  theme_minimal()
```

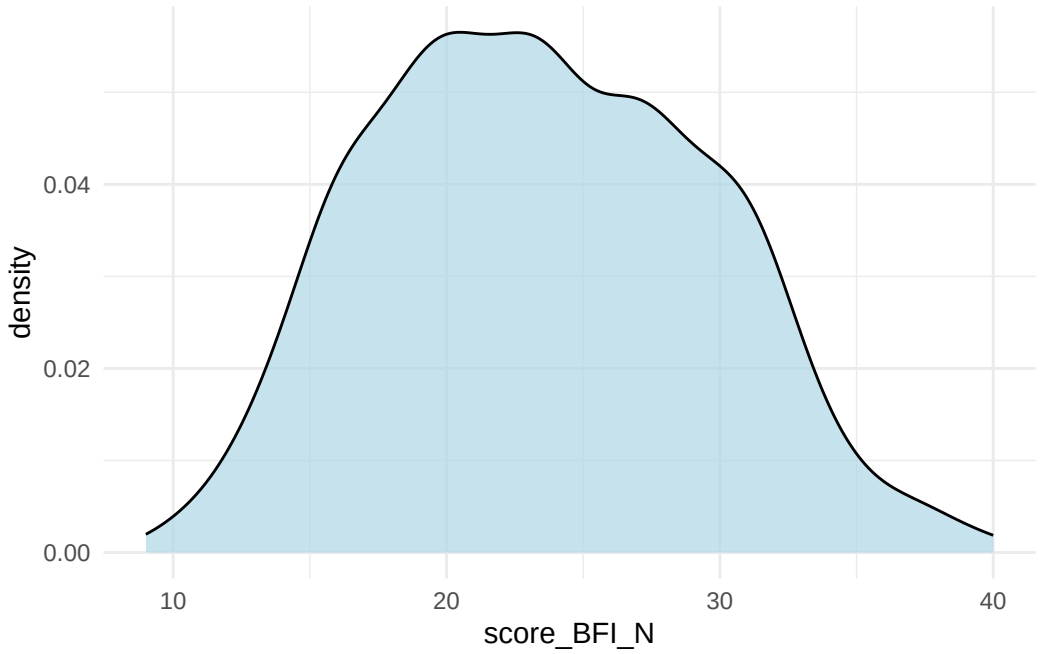


As you might expect from test scores, our data is clearly skewed right. There's no missing data for our Y variable of performance since we are not interested in participants who answered out of order against instructions. Let's also look at some predictors.

```
ggplot(
  amatus,
  aes(x = score_AMAS_total)
) +
  geom_density(fill = "lightblue", alpha = 0.7) +
  theme_minimal()
```




```
ggplot(
  amatus,
  aes(x = score_BFI_N)
) +
  geom_density(fill = "lightblue", alpha = 0.7) +
  theme_minimal()
```



One predictor's distribution is very normal, the other isn't. I'll save you the trouble and tell you that the rest of the features are a grab bag too. Normally we wouldn't care about standardization given that random forest is a tree-based model. However, because we are interested in variable importance metrics (since we are trying to focus on feature selection), we actually do need to scale our features. This is because variable importance metrics are skewed towards predictors that have a wider range of values. So a continuous variable may return a more biased estimate in the variable importance metric than a variable that can take 3 or 4 values. So we do want to scale our variables this time around.

We'll show code for an alternative way of scaling predictors, but the way we will choose to do it is use the scale() function to create a scaled training set. This will center (subtract mean from the value for a mean of 0) and scale (divide value by its standard deviation for an SD of 1) our continuous predictors. The training set will then only include the scaled variables. For our categorical variables, just remember not to worry about the scale function. We'll go ahead and get started by creating our first random forest model. Let's try using the caret package, a relatively general machine learning package.

A side note: There are several different validation methods you can use in caret, including bootstrapping, cross-validation, and repeated cross-validation. Repeated cross-validation should always be used when computationally appropriate. Bootstrapping may be compelling here due to the small dataset, but repeated cross-validation should work just as well as bootstrap in smaller samples and should do the job for us just fine. You can increase the "number" argument to increase the number of folds in the dataset and you can increase the "repeat" argument to increase the number of times the entire cross validation gets repeated, both at the expense of computational power.

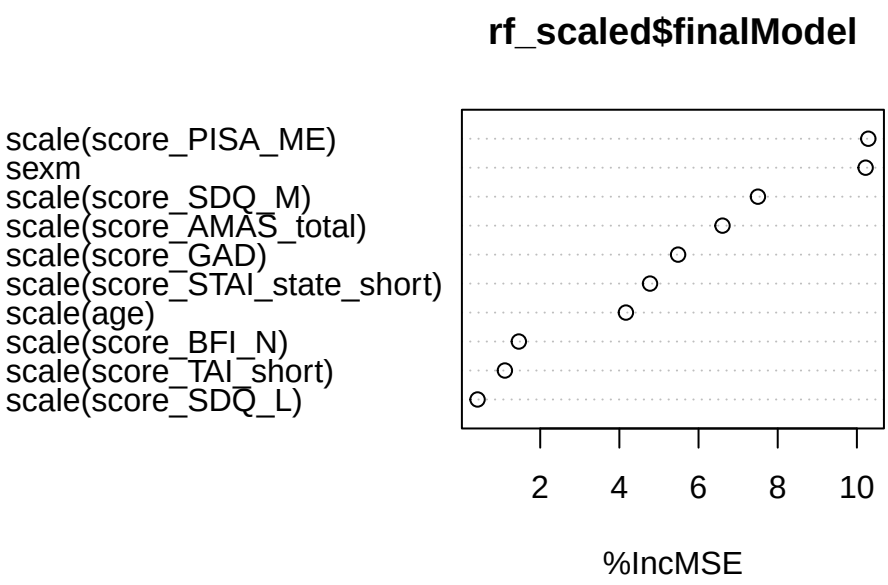
First Model Setup

```
library(caret)
set.seed(39) # I'm going to put this before every random generation just for
# clarity. Probably should've put it before the data partition.
inTrain <- createDataPartition(amatusclean$sum_arith_perf,
  p = 0.7, list = FALSE
)
# 0.7 selected to have a decent number of N in the test set

# create training vs test data.
training <- amatusclean[inTrain, ]
test <- amatusclean[-inTrain, ]
View(training)
View(test)


##### There are two methods to include scaling of predictor variables: manual
# scaling of predictors (and predictions later on), and preprocessing, which
# means using an R function to "process" your data to put it in the model.
# I will show an example with preProcess from the caret package, however
# the "recipes" package may be even more powerful for preprocessing and less
# inconvenient, since preProcess is usually used for scaling all variables
# in a dataset at once (and therefore is harder to use with subsequent models,
# since you need to tune the application of your preprocessing.)
#####
# I'm going to pull out the dependent variable.
trainnodv <- training
preProc <- preProcess(training, method = c("center", "scale"))
## Warning in preProcess.default(training, method = c("center", "scale")): Std.
## deviations could not be computed for: number_main_foci,
## preference_teaching_language, preference_teaching_mathematics,
## preference_teaching_science, ease_teaching_language, ease_teaching_mathematics,
## ease_teaching_science, score_FSMAS_SE, FSMAS_SE1, FSMAS_SE2, FSMAS_SE3,
## FSMAS_SE4, FSMAS_SE5, FSMAS_SE6, FSMAS_SE7, FSMAS_SE8, FSMAS_SE9
## Warning in preProcess.default(training, method = c("center", "scale")): These
## variables have zero variances: arith_perf30_resp, arith_perf32_resp,
## arith_perf34_resp, arith_perf35_resp, arith_perf36_resp, arith_perf37_resp,
## arith_perf38_resp, arith_perf39_resp, arith_perf40_resp
trainscaled <- predict(preProc, training)
# Again, we will not be using this, but training data would be standardized in
# trainscaled.
```

```r
# Let's try doing it in a more general way.
numeric_predictors <- c(
  "score_BFI_N", "score_AMAS_total", "age", "score_GAD",
  "score_PISA_ME", "score_STAI_state_short",
  "score_TAI_short", "score_SDQ_L", "score_SDQ_M"
)
categorical_variable <- "sex"
response_variable <- "sum_arith_perf"

# Create a new dataframe by scaling and centering numeric predictors.
library(dplyr)
scaled_training <- training %>%
  mutate(across(all_of(numeric_predictors), ~ scale(.) %>% as.vector())) %>%
  # Scale and convert back to vector just in case
  select(all_of(numeric_predictors), all_of(categorical_variable), all_of(response_variable))

# View the scaled data
View(scaled_training)

# mtry is a parameter of the number of randomly selected predictors to consider
# splitting at each node of the decision tree. More on this after the code.
control <- trainControl(
  method = "repeatedcv", number = 10, repeats = 3,
  search = "random"
) # picks random mtries in the range when tunegrid is in constant.
control <- trainControl(
  method = "repeatedcv", number = 10, repeats = 3,
  search = "grid"
)
# this one picks preselected values for mtry, which you would define
# on your own.
tunegrid <- expand.grid(.mtry = c(2:8))
# This creates a data frame from all
# combinations of the supplied vectors/factors. In other words, this creates
# the grid of values for the hyperparameter mtry. This  sets up the grid of
# different values for mtry.This one says try every value including
# and between 2 and 8. You can't try more #mtries than there are predictors.
# We're going to stick with this for the most part throughout.

# We're going to go with a grid search so our models can try and choose the
# "best" mtry between 2-8, since when we don't run it, our optimal mtry
# seems to be 1, which is incredibly low and random anyways, so we're setting 2
```

```r
# as the "lower bound".  Use the tunegrid argument and set mtry equal to your
# range of values if you want to search a grid of mtries. Example continues
# in this  chunk below.


# let's start with the default value for mtry and a random search in tunegrid.
x <- amatusclean[, 1:10]
mtry <- ncol(x) / 3
############
mtry <- ncol(x) / 3 # in random forest for regression, predictors/3 is often the
# "default" value of this parameter. In classification, it is the square root of
# the number of predictors.

control <- trainControl(
  method = "repeatedcv", number = 10, repeats = 4,
  search = "random"
)
set.seed(40)
# if you want to set seed for replicability, put this before every random thing
rf_scaled <- train(
  sum_arith_perf ~ scale(score_AMAS_total) + sex + scale(age) +
    scale(score_BFI_N) + scale(score_GAD) + scale(score_PISA_ME) +
    scale(score_STAI_state_short) + scale(score_TAI_short) +
    scale(score_SDQ_M) + scale(score_SDQ_L),
  data = training,
  importance = TRUE, method = "rf", tuneLength = 10, trControl = control
)
# Notice that we are not using the tuneGrid argument, and are instead using a
# tuneLength argument (it should search up to that many values of mtry.)
print(rf_scaled)
## Random Forest
##
## 516 samples
##  10 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 4 times)
## Summary of sample sizes: 465, 466, 462, 465, 463, 464, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##    2    6.144938  0.1431009  4.771292
```

```
##    3    6.191238  0.1353325  4.816626
##    5    6.232578  0.1277206  4.848793
##    7    6.261249  0.1259422  4.870313
##    8    6.278381  0.1224550  4.881691
##   10    6.286653  0.1225720  4.891823
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.
varImpPlot(rf_scaled$finalModel, type = 1)
```



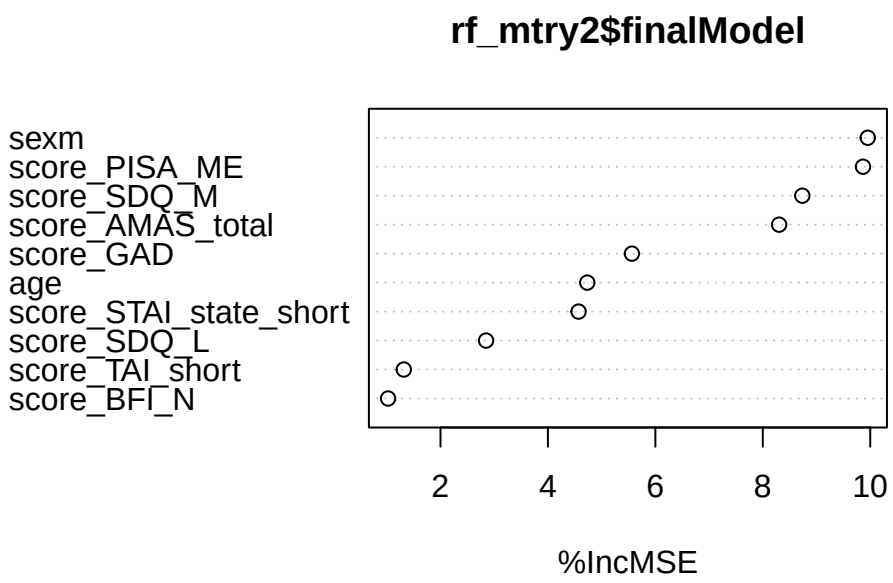
As you can see in the first model (rf_scaled), If it was up to caret, the best predictive method would have us randomly sample 1 predictor to consider splitting at each node of the decision tree. That means we probably have very few predictors that actually provide value in predicting math performance. This is expected in this dataset and in much psychological research, as we stated early on. Although an mtry=1 is too random, we clearly need to use a low mtry. The difference between the default (# of predictors/3) number for mtry and mtry=2 isn't too big, so we can stick with either for now. We'll try to set 2 as the minimum and make sure our code test a range of mtries, just in case another value ends up being better than 2 (which I don't expect to be the case).

let's exclude 1 and go again and try more options.

```
control <- trainControl(
  method = "repeatedcv", number = 10, repeats = 3,
  search = "grid"
)
```

```r
# the term "grid" means it will picks preselected values for mtry, which you
# would define on your own.
# remember to change the control variable back to search = "random" if you
# aren't using a grid search.

tunegrid <- expand.grid(.mtry = c(2:8)) # This one says try between 2-10.
# You can't try more mtries than there are predictors. We're going to stick
# with this throughout. Since our optimal mtry at first was 1, the optimal
# mtry is probably going to be low (and will likely be 2).

set.seed(40)
# I'm going to put this before every random generation just for clarity.
rf_mtry2 <- train(
  sum_arith_perf ~ score_AMAS_total + sex + age + score_BFI_N +
    score_GAD + score_PISA_ME + score_STAI_state_short + score_TAI_short +
    score_SDQ_M + score_SDQ_L,
  data = scaled_training, importance = TRUE,
  method = "rf", tuneGrid = tunegrid, trControl = control
)
# Notice we keep using the importance=TRUE parameter, which calculates variable
# importance metrics. We need this to be set to TRUE to be able to see
# varimp plots later.

print(rf_mtry2)
## Random Forest
##
## 516 samples
##  10 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 465, 466, 462, 465, 463, 464, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##   2     6.151306  0.1401146  4.785036
##   3     6.186846  0.1350186  4.817951
##   4     6.224445  0.1294847  4.846445
##   5     6.249729  0.1254135  4.868341
##   6     6.249204  0.1265625  4.858538
##   7     6.265862  0.1241394  4.878695
##   8     6.277114  0.1234517  4.893493
```

```
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.
varImpPlot(rf_mtry2$finalModel, type = 1)
```




```
# R2: 0.1466
# MSE: 6.331
```

R squared of around .14ish is obviously pretty low, but it's actually decent for psychological research. In published papers, you'll often happen to see r-squareds around .15-.25 or so, if that. It's very hard to build an extremely accurate model of human behavior namely because an incalculable number of factors influence our behavior in seen and unseen ways. Not to mention we care more about feature selection for than perfect prediction, which is why we are scaling variables in the first place. If we just wanted the best predictive model,

The VarImpplot function assesses the variable importance for each of the variables in our model, measured by the increase in Mean square error should the variable be removed from the model. As you can see, the lowest variable is extremely irrelevant to the model, that variable being test anxiety. Test anxiety may be a bad predictor of arithmetic performance, or more likely, it is not doing anything the state, math or general anxiety variables aren't doing already. Our models also seem to agree that language self-concept is also pretty useless, which makes sense conceptually.

It also looks like the models we run think that math self-concept, sex and math self-efficacy are the most important variables. Just looking at the graphs, there does indeed seem to be

around 2-5ish variables here that actually predict a significant amount of variance in our DV, so that's probably a great number for a parsimonious model.

Now, to fit our goal of parsimony, we want to cut out the variables that don't help us very much because we run the risk of creating a very nonparsimonious model, and of overfitting of course. For example, age seems to be mildly important, but is it important enough that we need it in our model to have accurate estimates? Well, from the model above, it kind of seems that there's only one, maybe two predictors that are extremely irrelevant to the model. This is probably in part because the effect sizes of most variables are so weak that removing any one variable can significantly reduce the predictive power of the model. In other words, the best predictive model might be a 9 or 10 variable model, but the best parsimonious model (which contains only the most relevant predictors at the cost of retaining all variables with even minor predictive power) may be somewhere in the 2-5 variable range.

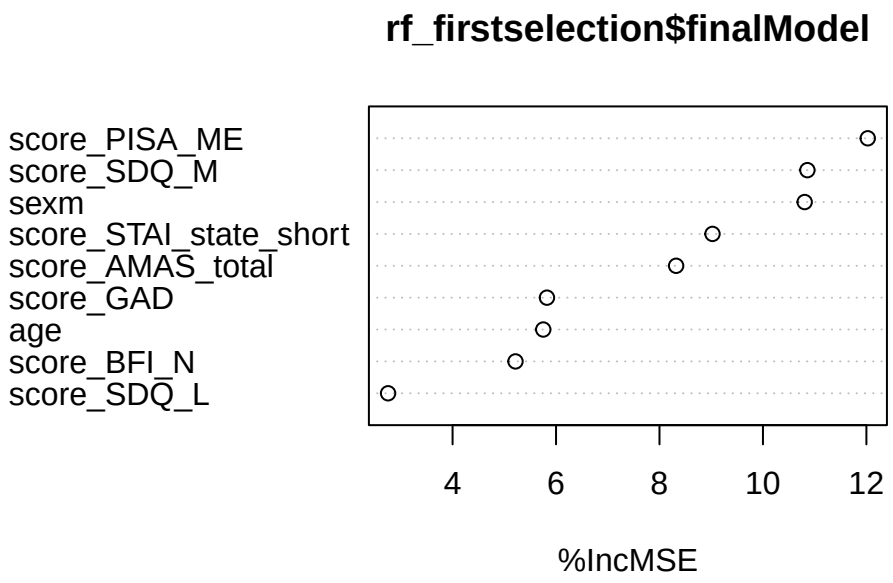
So anyways, we'll continue now. We'll start with a 9 variable model, which we'll create FIRST by cutting test anxiety.

Continuing RFE

```
set.seed(41)
rf_firstselection <- train(
  sum_arith_perf ~ score_AMAS_total + sex + age +
    score_BFI_N + score_GAD + score_PISA_ME + score_STAI_state_short +
    score_SDQ_M + score_SDQ_L,
  data = scaled_training, method = "rf",
  importance = TRUE, tuneGrid = tunegrid, trControl = control
)
print(rf_firstselection)
## Random Forest
##
## 516 samples
##   9 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 465, 466, 464, 464, 464, 463, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##   2     6.139361  0.1447147  4.751632
##   3     6.174787  0.1393946  4.777988
##   4     6.200939  0.1346102  4.793524
```

```
##   5     6.233474  0.1297989  4.825894
##   6     6.243198  0.1293169  4.831074
##   7     6.257551  0.1277391  4.845276
##   8     6.248580  0.1296425  4.828937
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.
# we'll look at variable importance again in a minute.
# R2: 0.14
# MSE: 6.355
```

```
varImpPlot(rf_firstselection$finalModel, type = 1)
```




```
# type 1 refers to estimates following perturbation.
```

We're seeing a similar pattern. Math self-confidence and sex seem to be two of the most powerful variables in every model, while math self-efficacy and math anxiety trail not too far behind (the rest seems to hang in the balance). It seems like of the ten we chose, those are some of the most important variables needed to predict math performance in German students. RMSE and R squared dropped, but these numbers fluctuate given the nature of random forest models. We'll take a look at how these models predict the test set later.

Notice that the %incMSE of the lowest variable is not close to 1 or 2, as test anxiety was. It's higher. Our model may be slightly more parsimonious. However, it may not be as good as predicting as the complete model.

Now, we're going to try to run more random forest models, this time with a reduced set of predictors since we are at least starting to identify the most important predictors. Let's see if we can find a more parsimonious model with still good predictive power.

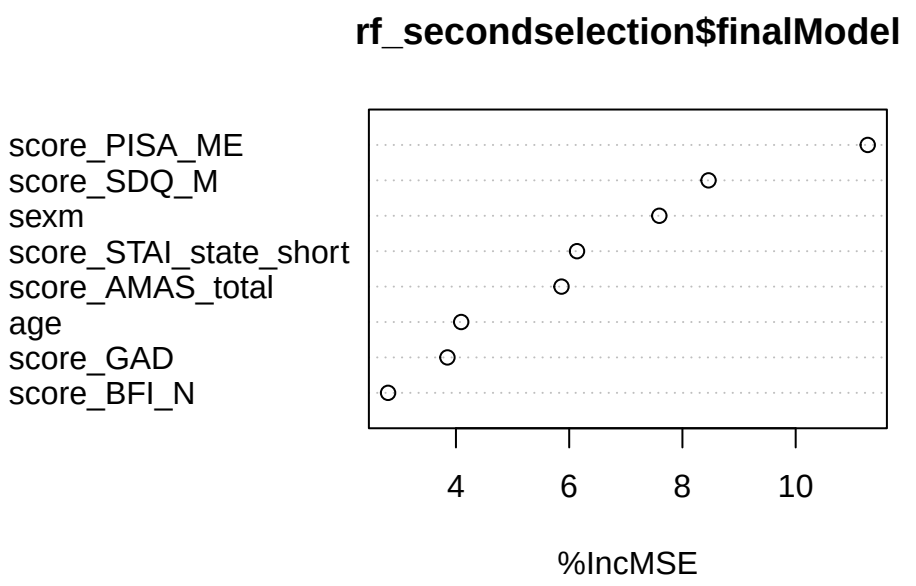
Since AMAS measures math anxiety, which conceptually seems to map on to math performance better than general anxiety, it makes sense that state anxiety and general anxiety are also relatively unimportant predictors. General anxiety doesn't seem to add much value to predicting math performance when math anxiety is already accounted for, and state anxiety isn't much better. Test anxiety (from earlier) being a weak predictor is not as intuitive, but it's possible that the most salient aspect here is that the test is on math, not that there's a test itself, so anxiety of tests in general doesn't really have predictive power of performance the same way math anxiety does.

Either way, we should start by cutting language self-efficacy. It's clearly just not a very good predictor.

```
set.seed(42)
rf_secondselection <- train(
  sum_arith_perf ~ score_AMAS_total + sex + age +
    score_BFI_N + score_STAI_state_short + score_PISA_ME + score_SDQ_M +
    score_GAD,
  data = scaled_training, method = "rf", importance = TRUE,
  tuneGrid = tunegrid, trControl = control
)
print(rf_secondselection)
## Random Forest
##
## 516 samples
##   8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 465, 464, 465, 465, 464, 465, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##   2     6.169999  0.1416024  4.775093
##   3     6.204004  0.1373119  4.811977
##   4     6.244685  0.1302838  4.845142
##   5     6.263324  0.1279773  4.864085
##   6     6.284937  0.1254186  4.876870
##   7     6.295599  0.1238701  4.888224
##   8     6.291819  0.1255025  4.892192
```

```
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.

# R2: 0.1416
# MSE: 6.325
```

```
varImpPlot(rf_secondselection$finalModel, type = 1)
```




```
# type 1 refers to estimates following perturbation.
```

Slight reduction in RMSE and increase in (training data) R-squared! Importantly, the variable importance metrics look pretty decent - the lowest incMSE% is around 5%. great news for our goal of parsimonious modeling. However, I'm not sure if this will continue. It's okay if while going through variable selection, your model doesn't necessarily have a massive reduction in RMSE or massive increase in r squared. Remember that we're looking to explain the most variance with as few variables (parsimony), so sometimes parsimony is worth slight decreases in predictive accuracy (though you should make sure you're not cutting a seemingly low-predictive power variable that nonetheless has explanatory importance).

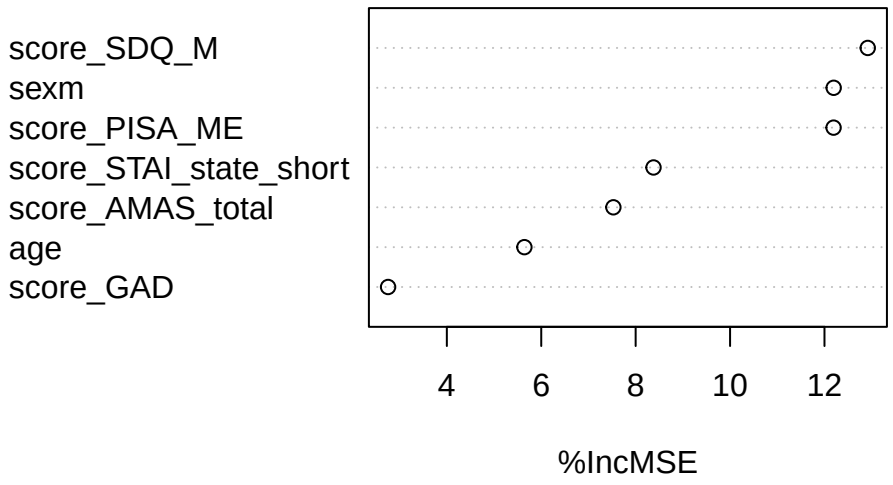
This may be a pretty good model. The fact that the $incMSE is approaching reasonable numbers like 6 instead of 1 or 2 means we're starting to run out of questionable predictors. This model is likely better than the full model, but possibly not the best model of sheer prediction. However, we may not need all those measures of anxiety, and like we said earlier, we likely are looking for a 2-5 variable model (on the lower end, I'd expect) to cover almost all of the variance. So we should keep going before we start prediction.

Let's cut the next least important variable - BFI_N.

```
set.seed(43) #
rf_thirdselection <- train(
  sum_arith_perf ~ score_AMAS_total + sex + age +
    score_STAI_state_short + score_PISA_ME + score_GAD + score_SDQ_M,
  data = scaled_training, method = "rf", importance = TRUE,
  tuneGrid = tunegrid, trControl = control
)
print(rf_thirdselection)
## Random Forest
##
## 516 samples
##   7 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 466, 462, 466, 465, 464, 465, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##   2     6.169651  0.1372980  4.792023
##   3     6.209956  0.1322796  4.813346
##   4     6.260444  0.1242485  4.848839
##   5     6.264937  0.1239306  4.855466
##   6     6.279955  0.1219697  4.874503
##   7     6.286944  0.1212538  4.876339
##   8     6.295579  0.1201983  4.883164
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.

# we'll look at variable importance again later.
# R2: 0.1224
# MSE: 6.415
```

No improvements in R squared seems odd...until we take a look at the VarImp output.

```
varImpPlot(rf_thirdselection$finalModel, type = 1)
```



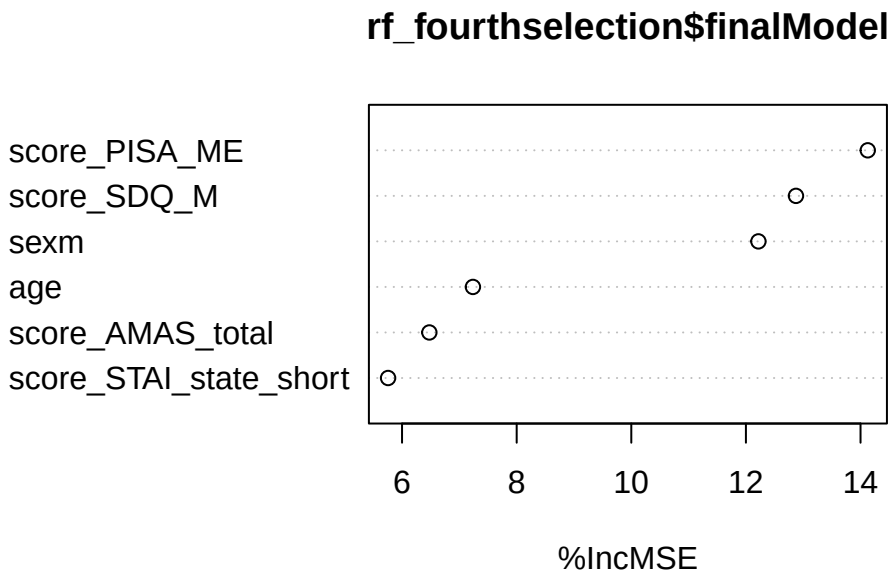

```
# type 1 refers to estimates following perturbation.
```

Jeez, the RMSE went up pretty badly. Clearly this isn't better than the previous model. Let's keep going. By the way, at this point the highest mtry being tried (7) is greater than number of predictors (6). It still works without it but it produces a lot of annoying warnings, so we'll start to include a cap line at the beginning of the chunk of code for your convenience (before the set seed, importantly).

```
tunegrid <- expand.grid(.mtry = c(2:6))
set.seed(44)
rf_fourthselection <- train(
  sum_arith_perf ~ score_AMAS_total + sex + age +
    score_STAI_state_short + score_PISA_ME + score_SDQ_M,
  data = scaled_training,
  method = "rf", importance = TRUE, tuneGrid = tunegrid, trControl = control
)
print(rf_fourthselection)
## Random Forest
##
## 516 samples
##   6 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 465, 463, 465, 464, 465, 463, ...
```

```
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##   2     6.166181  0.1423487  4.760528
##   3     6.206901  0.1386314  4.793719
##   4     6.229310  0.1359417  4.814428
##   5     6.233865  0.1371535  4.807952
##   6     6.260060  0.1333343  4.828133
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.
# we'll look at variable importance again later.
# R2: 0.1301
# MSE: 6.38
```

```
varImpPlot(rf_fourthselection$finalModel, type = 1)
```

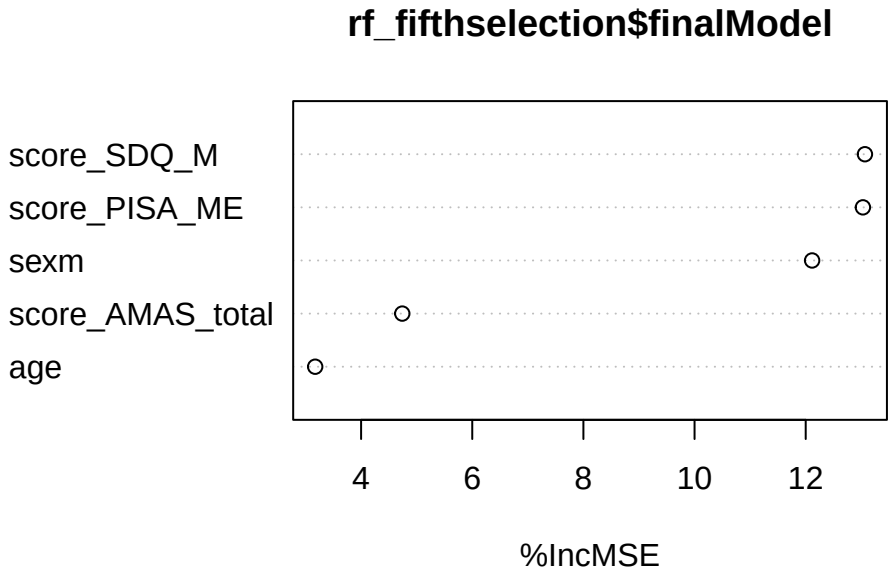



```
# type 1 refers to estimates following perturbation.
```

See how the RMSE went a little back down again? To be expected. Our hunch earlier was correct - looks like the anxiety variables aren't really helping us all that much. I suspect the MOST parsimonious model won't have these either. Let's cut the last one then.

```r
tunegrid <- expand.grid(.mtry = c(2:5))
set.seed(45)
rf_fifthselection <- train(
  sum_arith_perf ~ sex + age + score_AMAS_total +
    score_PISA_ME + score_SDQ_M,
  data = scaled_training, method = "rf",
  importance = TRUE, tuneGrid = tunegrid, trControl = control
)
print(rf_fifthselection)
## Random Forest
##
## 516 samples
##   5 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 465, 464, 465, 465, 466, 465, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##   2     6.264063  0.1253722  4.886846
##   3     6.336902  0.1191958  4.953938
##   4     6.385352  0.1147949  4.995215
##   5     6.407679  0.1127542  5.016567
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.
# R2: 0.1576
# MSE: 6.27
```

```r
varImpPlot(rf_fifthselection$finalModel, type = 1)
```



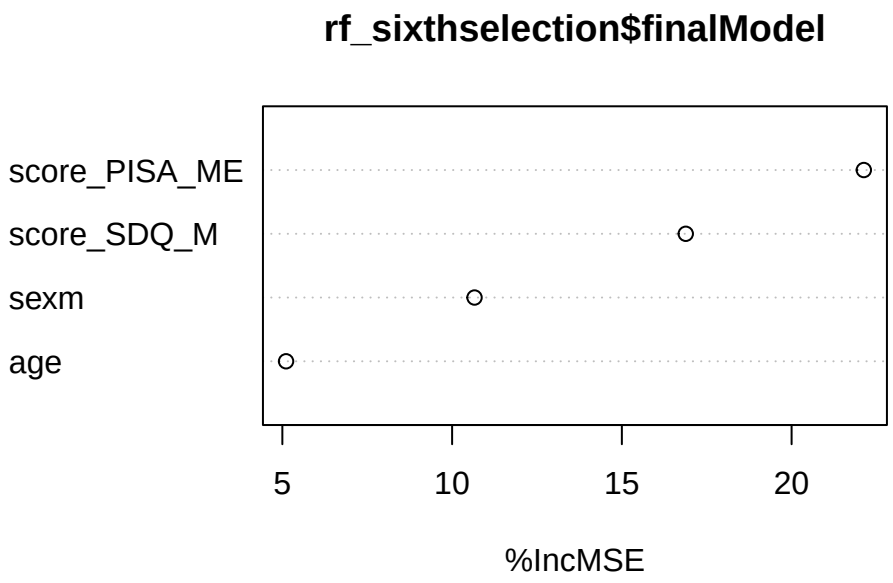

```
# type 1 refers to estimates following perturbation.
```

Woah, this model looks way better. RMSE is way down and R squared is up. In fact, this seems like it may even be better than our 8-variable model. We'll keep this in mind as we cut neuroticism, which again seems to be an extremely irrelevant predictor. However, keeping neuroticism may be useful for predictive power of our parsimonious model. Also, if you haven't noticed, age used to be one of the predictors with the lowest incMSE% in the full model, but has stuck in the models we are testing. Age is probably a good parsimonious predictors (i.e. it can explain a lot of variance in relatively sparse models), but it may be suppressed or obscured by highly collinear predictors in the full model. If this were an actual research project, I would have conducted exploratory data analysis to check age's correlation with other variables as well as evaluated the part correlation in the full model to see if age was being suppressed.

```
# just for fun, we'll use the in-formula scale() method to scale data.
tunegrid <- expand.grid(.mtry = c(2:4))
set.seed(46)
rf_sixthselection <- train(
  sum_arith_perf ~ sex + score_PISA_ME + score_SDQ_M +
    age,
  data = scaled_training, method = "rf", importance = TRUE,
  tuneGrid = tunegrid, trControl = control
)
print(rf_sixthselection)
## Random Forest
##
```

```
## 516 samples
##   4 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 465, 465, 464, 465, 465, 463, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared    MAE
##   2     6.403497  0.11263027  4.992278
##   3     6.547069  0.10108205  5.095937
##   4     6.605459  0.09783928  5.138611
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.

# R2: 0.1422
# MSE: 6.39
```

```
varImpPlot(rf_sixthselection$finalModel, type = 1)
```




```
# type 1 refers to estimates following perturbation.
```

The MSE went up again, and by a decent amount. At this point, it seems safe to say that the four variables above are the most "important" variables in terms of building a parsimonious

model, and math self-efficacy and sex are likely the top two. Given the better metrics of the fifth selection model, neuroticism also seems to be a pretty good predictor in a parsimonious model, and we likely don't want to cut it. However, you may only want to identify the "most important variables", and though neuroticism may be a good predictor in a 5 variable model, it may not be the "most important variable". The 4 or 5 variable models seem to be pretty good parsimonious models if we want to collect more than only 2 variables. In other words, we've identified 5 variables that can produce a good model that captures a good amount of the variability in scores. This model may be the best combination of parsimony and predictive power (in fact, it seems like it may be flat out the best predictive model), but this model is likely not the most parsimonious (that seems to be a model with only a couple predictors, as we've indicated before). There is also a chance it is not the most predictive (which we'll test soon).

Automated Feature Selection Methods: VSURF

Instead of just continuing the same process and cutting the last few variables, I will provide an example of an explicit method for variable selection in random forest models that will return the same output. VSURF, the method in question, essentially uses backward stepwise selection followed by forward selection based on variable importance (using a basic definition as we used above being, depending on the task, the difference between MSE/misclassification rate after permuting each predictor). There have been several methods for variable selection described in the literature, however, a recent simulation study demonstrated that VSURF often demonstrates some of the best performance for classification tasks (Speiser et al.), which may extend to regression. However, it is worth noting that VSURF is prediction oriented and as such, accepts the risk of producing false negatives.

```
if (!require(VSURF)) {
  install.packages("VSURF",
    repos = "http://cran.us.r-project.org"
  )
}
## Loading required package: VSURF

library(VSURF)
rm(x)
#we don't really care about only testing a few mtry values at this point
#since the best has been mtry=2 every time, so.
set.seed(46)
vamatus <- VSURF(sum_arith_perf ~ scale(training$score_AMAS_total) + sex +
  scale(training$age) + scale(training$score_BFI_N) +
  scale(training$score_GAD) + scale(training$score_PISA_ME) +
  scale(training$score_STAI_state_short) + scale(training$score_TAI_short) +
```

```
  scale(training$score_SDQ_M) +
  scale(training$score_SDQ_L), data = training, na.action = na.omit)
## Thresholding step
## Estimated computational time (on one core): 11.1 sec.
##
  |
  |                                                                      |   0%
  |
  |====                                                                  |   5%
  |
  |=======                                                               |  10%
  |
  |==========                                                            |  15%
  |
  |==============                                                        |  20%
  |
  |==================                                                    |  25%
  |
  |======================                                                |  30%
  |
  |=========================                                             |  35%
  |
  |============================                                          |  40%
  |
  |================================                                      |  45%
  |
  |===================================                                   |  50%
  |
  |=======================================                               |  55%
  |
  |==========================================                            |  60%
  |
  |==============================================                        |  65%
  |
  |=================================================                     |  70%
  |
  |====================================================                  |  75%
  |
  |========================================================              |  80%
  |
  |============================================================          |  85%
  |
```

```
  |=============================================================       |  90%
  |
  |=================================================================    |  95%
  |
  |=====================================================================| 100%
## Interpretation step (on 10 variables)
## Estimated computational time (on one core): between 1.7 sec. and  7.6 sec.
##
  |
  |                                                                     |   0%
  |
  |=======                                                              |  10%
  |
  |==============                                                       |  20%
  |
  |======================                                               |  30%
  |
  |=============================                                        |  40%
  |
  |====================================                                 |  50%
  |
  |============================================                         |  60%
  |
  |===================================================                  |  70%
  |
  |==========================================================           |  80%
  |
  |================================================================     |  90%
  |
  |=====================================================================| 100%
## Prediction step (on 4 variables)
## Maximum estimated computational time (on one core): 0.8 sec.
##
  |
  |                                                                     |   0%
  |
  |==================                                                   |  25%
  |
  |==================================                                   |  50%
  |
  |===================================================                  |  75%
  |
```

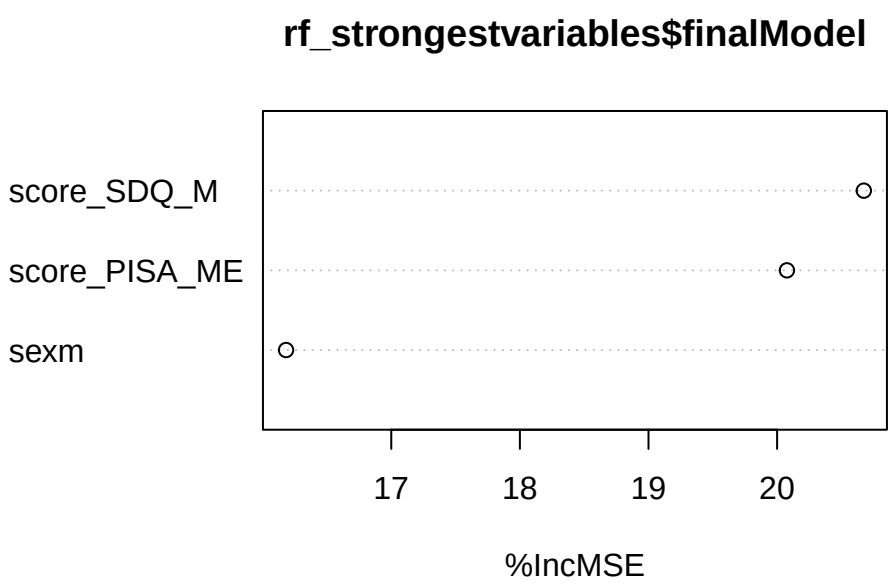
```
  |======================================================================| 100%
## Warning in VSURF.formula(sum_arith_perf ~ scale(training$score_AMAS_total) + : VSURF with
## which are indices of the input matrix based on the formula:
## you may reorder these to get indices of the original data

View(training)

print(vamatus$varselect.pred) # predictors vsurf selected at final stage.
## [1] 6 2
```

VSURF suggests that the best model would contain two variables, PISA_ME and sex. Keep in mind this is the most powerful predictive model with the fewest variables. Thus, sex may explains a lot more of the variance than a variable like self-concept when self-efficacy is included in the model (and in fact, including all three may again lead to a less parsimonious model, or worse predictions).

Let's work it out in a random forest paradigm for fun. This should hopefully show us the predictive benefit of using such an automated system to create a parsimonious model.

```
tunegrid <- expand.grid(.mtry = c(2:3))
set.seed(47)
rf_strongestvariables <- train(
  sum_arith_perf ~ sex + score_PISA_ME +
    score_SDQ_M,
  data = scaled_training, method = "rf", importance = TRUE,
  tuneGrid = tunegrid, trControl = control
)
print(rf_strongestvariables)
## Random Forest
##
## 516 samples
##   3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 464, 463, 464, 463, 465, 466, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared    MAE
##   2     6.396362  0.10793945  5.039697
##   3     6.591063  0.08948443  5.185178
##
```

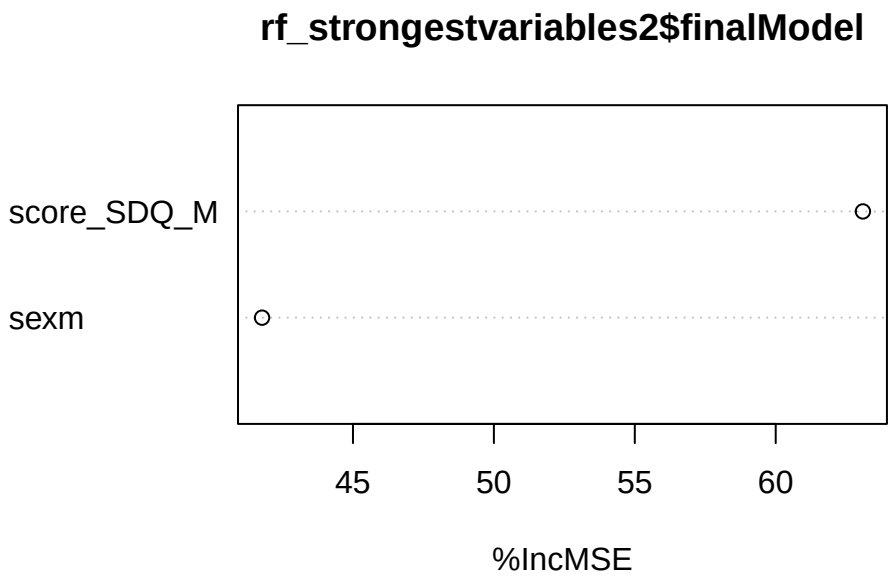
```
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.
varImpPlot(rf_strongestvariables$finalModel, type = 1)
```




```
# type 1 refers to estimates following perturbation.
# R2: 0.1022
# MSE: 6.532


tunegrid <- expand.grid(.mtry = c(2:2))
set.seed(48)
rf_strongestvariables2 <- train(sum_arith_perf ~ sex + score_SDQ_M,
  data = scaled_training, method = "rf", importance = TRUE,
  tuneGrid = tunegrid, trControl = control
)
print(rf_strongestvariables2)
## Random Forest
##
## 516 samples
##   2 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 464, 463, 465, 466, 464, 464, ...
## Resampling results:
```

```
##
##   RMSE      Rsquared   MAE
##   6.20309   0.1361983  4.800913
##
## Tuning parameter 'mtry' was held constant at a value of 2
```

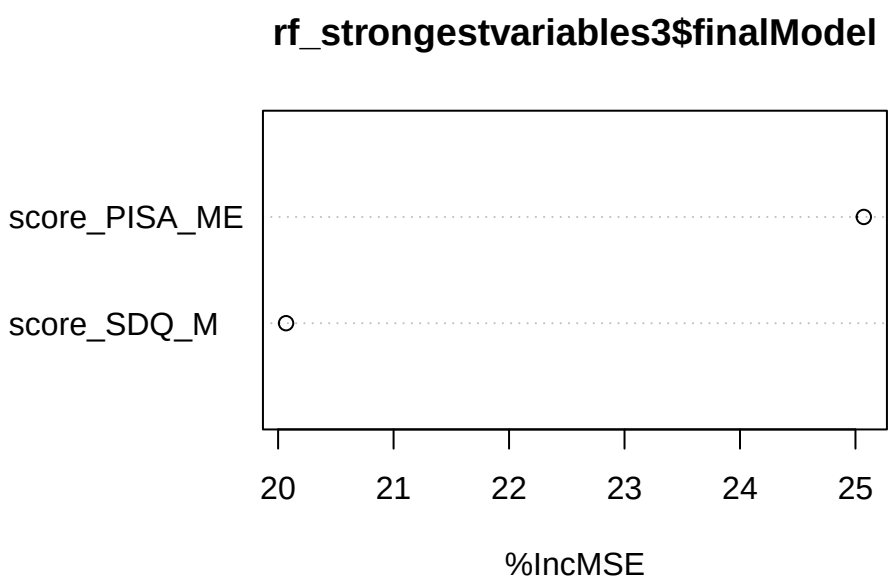
```
varImpPlot(rf_strongestvariables2$finalModel, type = 1)
```




```
# type 1 refers to estimates following perturbation.
# R2: 0.117
# MSE: 6.37
```

```
set.seed(49)
rf_strongestvariables3 <- train(sum_arith_perf ~ score_PISA_ME + score_SDQ_M,
  data = scaled_training, method = "rf", importance = TRUE,
  tuneGrid = tunegrid, trControl = control
)
print(rf_strongestvariables3)
## Random Forest
##
## 516 samples
##   2 predictor
##
```

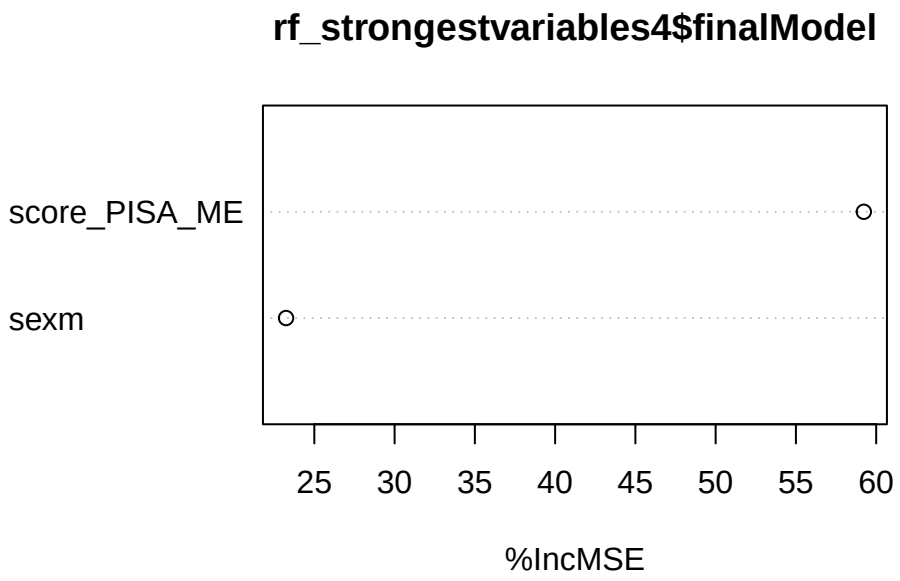
```
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 465, 466, 465, 463, 463, 466, ...
## Resampling results:
##
##   RMSE     Rsquared    MAE
##   6.58166  0.08467893  5.209105
##
## Tuning parameter 'mtry' was held constant at a value of 2

varImpPlot(rf_strongestvariables3$finalModel, type = 1)
```




```
# type 1 refers to estimates following perturbation.
# R2: 0.062
# MSE: 6.81


# this will be the VSURF selected model.
set.seed(50)
rf_strongestvariables4 <- train(sum_arith_perf ~ score_PISA_ME + sex,
  data =
    scaled_training, method = "rf", importance = TRUE, tuneGrid = tunegrid,
  trControl = control
)
print(rf_strongestvariables4)
```

```
## Random Forest
##
## 516 samples
##   2 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 464, 465, 464, 463, 466, 466, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   6.154505  0.1399721  4.86602
##
## Tuning parameter 'mtry' was held constant at a value of 2
```

```
varImpPlot(rf_strongestvariables4$finalModel, type = 1)
```




```
# type 1 refers to estimates following perturbation.
# R2: 0.127
# MSE: 6.32
```

As you can see, the model with sex and self-efficacy has by far the lowest MSE and the best R-squared. Its r-squared is comparable to models with many more variables, and its MSE isn't too bad either. It's safe to say that this is definitely the most parsimonious model in the sense that it explains a ton of the variance with only two variables. Adding a couple more variables

```
    new_sd <- scaling_params[[predictor]]$sd

    # C
    # Overwrite the original predictor with the scaled version
    test[[predictor]] <- (test[[predictor]] - new_mean) / new_sd
  } else {
    warning(paste("Variable", predictor, "not found in test dataset."))
  }
}
```

should likely increase variable power, but if you're looking to identify the model with the absolute best predictive power in the fewest variables, look no further.

Predictions on Test Set

Prediction time. Let's see what we get.

```
# View the dataset with the overwritten variables
View(test)
```

```
# Then, find predictions.
# 10 variable model
predictionsmtry2 <- predict(rf_mtry2, newdata = test)
testmtry2 <- data.frame(
  R2 = R2(predictionsmtry2, test$ sum_arith_perf),
  ## this calculates some kind of pseudo R squared. The manual formula-derived
  # R squared is just below under "standard R2 formula".
  RMSE = RMSE(predictionsmtry2, test$ sum_arith_perf),
  MAE = MAE(predictionsmtry2, test$ sum_arith_perf)
)
print(testmtry2)
##          R2     RMSE      MAE
## 1 0.1300653 6.407367 5.034175
# Standard R2 formula
actuals <- test$sum_arith_perf
SSE <- sum((predictionsmtry2 - actuals)^2) # Sum of Squared Errors for mtry2
SST <- sum((actuals - mean(actuals))^2) # Total Sum of Squares for mtry2
R_squared <- 1 - (SSE / SST)
R_squared # You can try this if you'll like
## [1] 0.1299872


###
# first selection
predictionsfirstselection <- predict(rf_firstselection, newdata = test)
# you're predicting test data from the model built on training data.
validationfirstselection <- data.frame(
  R2 = R2(predictionsfirstselection, test$ sum_arith_perf),
  RMSE = RMSE(predictionsfirstselection, test$ sum_arith_perf),
  MAE = MAE(predictionsfirstselection, test$ sum_arith_perf)
)
print(validationfirstselection)
##          R2    RMSE      MAE
```

```
## 1 0.1332341 6.39637 5.039182

actuals <- test$sum_arith_perf
SSE <- sum((predictionsfirstselection - actuals)^2) # Sum of Squared Errors
SST <- sum((actuals - mean(actuals))^2) # Total Sum of Squares
R_squared <- 1 - (SSE / SST)
R_squared
## [1] 0.1329712



###
### Second selection
predictionssecondselection <- predict(rf_secondselection, test)
validationsecondselection <- data.frame(
  R2 = R2(predictionssecondselection, test$ sum_arith_perf),
  RMSE = RMSE(predictionssecondselection, test$ sum_arith_perf),
  MAE = MAE(predictionssecondselection, test$ sum_arith_perf)
)
# these statistical functions are in caret technically under trainControl(),
# which passes this argument through defaultSummary() I believe.
print(validationsecondselection)
##          R2     RMSE      MAE
## 1 0.1115082 6.485156 5.094273

actuals <- test$sum_arith_perf
SSE <- sum((predictionssecondselection - actuals)^2) # Sum of Squared Errors
SST <- sum((actuals - mean(actuals))^2) # Total Sum of Squares
R_squared <- 1 - (SSE / SST)
R_squared
## [1] 0.108734


#### Third selection
predictionsthirdselection <- predict(rf_thirdselection, test)
validationthirdselection <- data.frame(
  R2 = R2(predictionsthirdselection, test$ sum_arith_perf),
  RMSE = RMSE(predictionsthirdselection, test$ sum_arith_perf),
  MAE = MAE(predictionsthirdselection, test$ sum_arith_perf)
)
# these statistical functions are in caret technically under trainControl(),
# which passes this argument through defaultSummary() I believe.
```

```r
print(validationthirdselection)
##          R2     RMSE      MAE
## 1 0.1018598 6.540332 5.140025

actuals <- test$sum_arith_perf
SSE <- sum((predictionsthirdselection - actuals)^2) # Sum of Squared Errors
SST <- sum((actuals - mean(actuals))^2) # Total Sum of Squares
R_squared <- 1 - (SSE / SST)
R_squared
## [1] 0.0935036


###### fourth selection
predictionsfourthselection <- predict(rf_fourthselection, test)
validationfourthselection <- data.frame(
  R2 = R2(predictionsfourthselection, test$ sum_arith_perf),
  RMSE = RMSE(predictionsfourthselection, test$ sum_arith_perf),
  MAE = MAE(predictionsfourthselection, test$ sum_arith_perf)
)
print(validationfourthselection)
##          R2     RMSE      MAE
## 1 0.1006701 6.551254 5.168939

actuals <- test$sum_arith_perf
SSE <- sum((predictionsfourthselection - actuals)^2) # Sum of Squared Errors
SST <- sum((actuals - mean(actuals))^2) # Total Sum of Squares
R_squared <- 1 - (SSE / SST)
R_squared
## [1] 0.09047348
## BEST PREDICTIVE MODEL!

##### fifth selection
predictionsfifthselection <- predict(rf_fifthselection, test)
validationfifthselection <- data.frame(
  R2 = R2(predictionsfifthselection, test$ sum_arith_perf),
  RMSE = RMSE(predictionsfifthselection, test$ sum_arith_perf),
  MAE = MAE(predictionsfifthselection, test$ sum_arith_perf)
)
print(validationfifthselection)
##         R2     RMSE      MAE
## 1 0.078556 6.691532 5.274087
```

```r
actuals <- test$sum_arith_perf
SSE <- sum((predictionsfifthselection - actuals)^2) # Sum of Squared Errors
SST <- sum((actuals - mean(actuals))^2) # Total Sum of Squares
R_squared <- 1 - (SSE / SST)
R_squared
## [1] 0.05110631


#### sixth selection
predictionssixthselection <- predict(rf_sixthselection, test)
validationsixthselection <- data.frame(
  R2 = R2(predictionssixthselection, test$ sum_arith_perf),
  RMSE = RMSE(predictionssixthselection, test$ sum_arith_perf),
  MAE = MAE(predictionssixthselection, test$ sum_arith_perf)
)
print(validationsixthselection)
##           R2     RMSE      MAE
## 1 0.09670236 6.641542 5.195073

actuals <- test$sum_arith_perf
SSE <- sum((predictionssixthselection - actuals)^2) # Sum of Squared Errors
SST <- sum((actuals - mean(actuals))^2) # Total Sum of Squares
R_squared <- 1 - (SSE / SST)
R_squared
## [1] 0.06523103


### two variablemodel
predictionstwovariablemodel <- predict(rf_strongestvariables4, test)
validationtwovariablemodel <- data.frame(
  R2 = R2(predictionstwovariablemodel, test$ sum_arith_perf),
  RMSE = RMSE(predictionstwovariablemodel, test$ sum_arith_perf),
  MAE = MAE(predictionstwovariablemodel, test$ sum_arith_perf)
)
print(validationtwovariablemodel)
##          R2    RMSE      MAE
## 1 0.1056989 6.52584 5.081187

actuals <- test$sum_arith_perf
SSE <- sum((predictionstwovariablemodel - actuals)^2) # Sum of Squared Errors
SST <- sum((actuals - mean(actuals))^2) # Total Sum of Squares
R_squared <- 1 - (SSE / SST)
```

```
R_squared
## [1] 0.09751648


#### Just to double check our code above.
```

Like we posited earlier, the full model did indeed turn out to be the best model for prediction. Our parsimonious model turned out to be one of the best models for prediction (though seemingly worse than the 10, 9 and 8 variable models), and our in-between models with the 4/5 most key variables, despite predicting the train data the best, predicted the test set the worst. Thus, feature selection in this case turned out to be wholly unnecessary for prediction, and was only useful for identifying a parsimonious model and the "strongest" (but not best at predicting) predictors from the list of ten. This may happen in psychology data because of the relatively small effect sizes of even some of the best predictors - this is why train/test splitting (or train, test, validation splitting) is important. Therefore, if we were only looking to predict arithmetic performance in German students, it would be best to use the full model or use another algorithmic variable selection method, such as LASSO (which we will use in an addendum shortly). If we were more interested in describing the most important predictors in this dataset than prediction, the two-variable model would be a good choice. The 4-5 variable models might also be a good choice to explain a few additional key predictors, but given the poor fit of the model on test data, I would not be so convinced. I would again try some additional algorithmic variable selection methods to see if there was a better fitting model in that range.

I want to finish by reminding you that random forests are random, so you will get a new model every time you run the train command. With good predictors and excellent repeated cross validation, it won't change much, but it will change. The "stepwise" procedure I followed here was primarily for narrative purposes and may not be the best choice to build the single best model at prediction or another task, but it will allow you to evaluate the variable importance at each step and see how the different models predict the data (and which variables they emphasize). This will help you to select the model that best fits your goal of parsimony, variable selection, prediction, all three, or another goal. Of course, if you have a lot of predictors, it may be best for you to use an automatic method such as VSURF, which is very similar to the process we utilized but tuned with an algorithm it attempts to optimize rather than with our statistical knowledge.

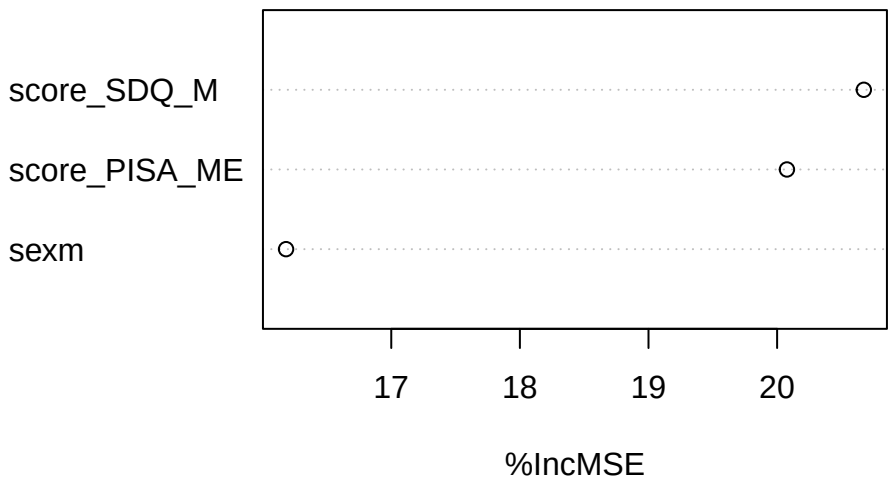
TRYING OUT MODELS SIMILAR TO THOSE LASSO SUGGESTED.

```
set.seed(59)
rf_lassopretest <- train(
  sum_arith_perf ~ sex + score_TAI_short +
    score_STAI_state_short + score_PISA_ME + score_SDQ_M + score_SDQ_L + age,
  data = scaled_training, method = "rf", importance = TRUE,
```

```
  tuneGrid = tunegrid, trControl = control
)
print(rf_lassopretest)
## Random Forest
##
## 516 samples
##   7 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 463, 464, 464, 464, 463, 465, ...
## Resampling results:
##
##   RMSE     Rsquared   MAE
##   6.22724  0.1243477  4.853981
##
## Tuning parameter 'mtry' was held constant at a value of 2

rf_lassopretest$dev.ratio
## NULL

set.seed(58)
rf_lassopretest2 <- train(
  sum_arith_perf ~ sex + score_TAI_short +
    score_STAI_state_short + score_PISA_ME + score_SDQ_M + age,
  data = scaled_training, method = "rf", importance = TRUE,
  tuneGrid = tunegrid, trControl = control
)
print(rf_lassopretest2)
## Random Forest
##
## 516 samples
##   6 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 464, 465, 465, 464, 464, 465, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   6.227636  0.1307277  4.804795
##
```

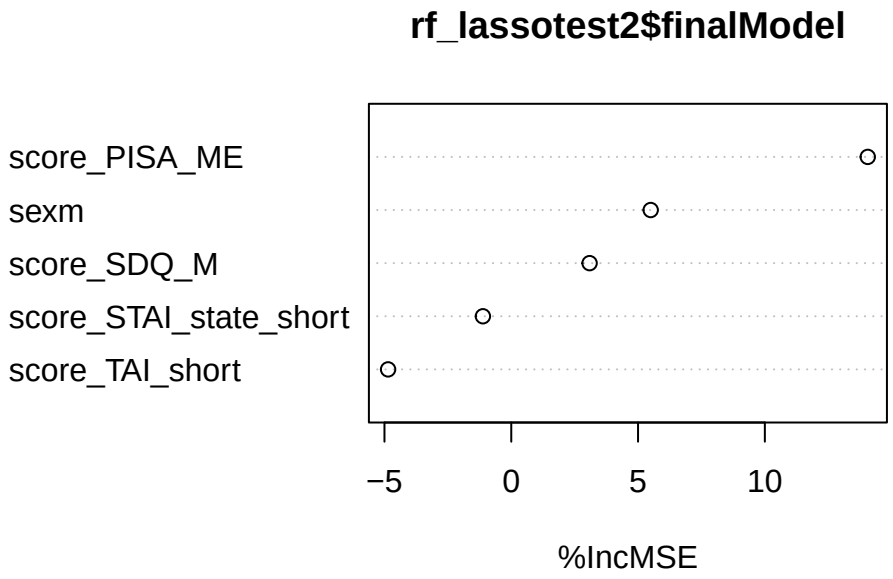
```
## Tuning parameter 'mtry' was held constant at a value of 2
```

```
set.seed(60)
rf_lassotest <- train(
  sum_arith_perf ~ sex + score_TAI_short +
    score_STAI_state_short + score_PISA_ME + score_SDQ_M + score_SDQ_L,
  data =
    scaled_training, method = "rf", importance = TRUE, tuneGrid = tunegrid,
  trControl = control
)
print(rf_lassotest)
## Random Forest
##
## 516 samples
##   6 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 465, 464, 465, 463, 466, 466, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   6.375484  0.1030721  4.957119
##
## Tuning parameter 'mtry' was held constant at a value of 2
varImpPlot(rf_strongestvariables$finalModel, type = 1)
```



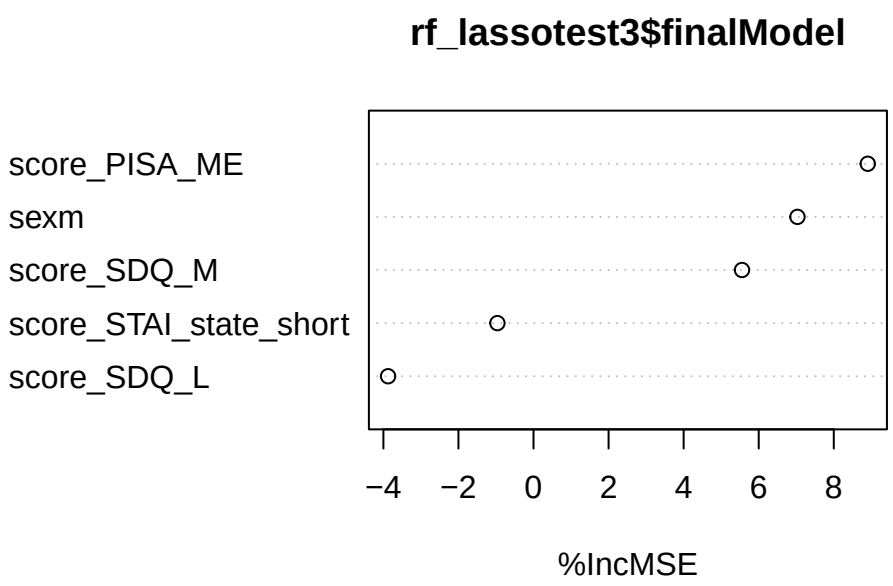

```
# R2: 0.1059
# MSE: 6.596


set.seed(61)
rf_lassotest2 <- train(
  sum_arith_perf ~ sex + score_TAI_short +
    score_STAI_state_short + score_PISA_ME + score_SDQ_M,
  data = scaled_training,
  method = "rf", importance = TRUE, tuneGrid = tunegrid, trControl = control
)
print(rf_lassotest2)
## Random Forest
##
## 516 samples
##   5 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 466, 465, 464, 464, 465, 464, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   6.383921  0.1075884  4.922998
##
## Tuning parameter 'mtry' was held constant at a value of 2
```

```r
varImpPlot(rf_lassotest2$finalModel, type = 1)
```



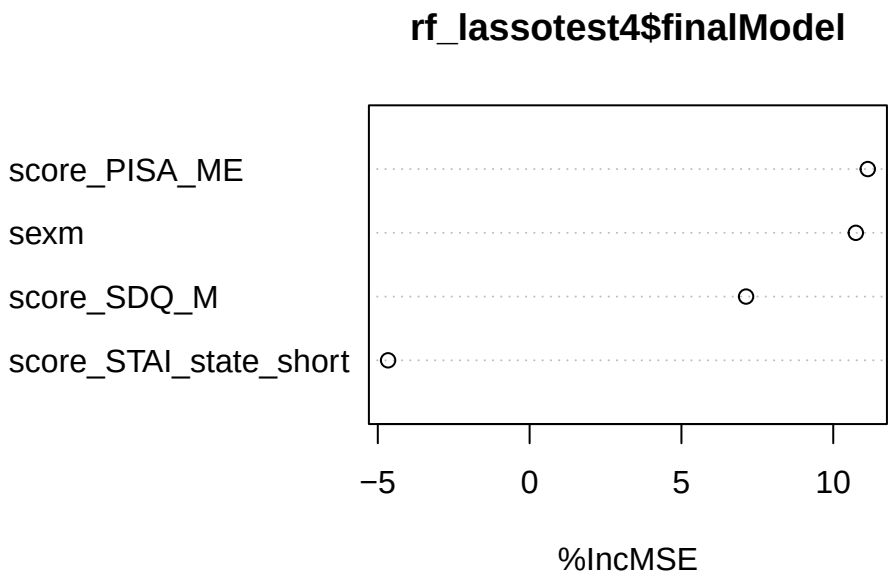

```r
# R2: 0.1059
# MSE: 6.596

set.seed(62)
rf_lassotest3 <- train(
  sum_arith_perf ~ sex + score_STAI_state_short +
    score_PISA_ME + score_SDQ_M + score_SDQ_L,
  data = scaled_training,
  method = "rf", importance = TRUE, tuneGrid = tunegrid, trControl = control
)
print(rf_lassotest3)
## Random Forest
##
## 516 samples
##   5 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 464, 464, 464, 463, 464, 464, ...
## Resampling results:
##
##   RMSE      Rsquared    MAE
##   6.446462  0.09181291  5.040711
```

```
##
## Tuning parameter 'mtry' was held constant at a value of 2
varImpPlot(rf_lassotest3$finalModel, type = 1)
```




```
# R2: 0.1059
# MSE: 6.596



set.seed(63)
rf_lassotest4 <- train(
  sum_arith_perf ~ sex + score_STAI_state_short +
    score_PISA_ME + score_SDQ_M,
  data = scaled_training, method = "rf",
  importance = TRUE, tuneGrid = tunegrid, trControl = control
)
print(rf_lassotest4)
## Random Forest
##
## 516 samples
##   4 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 464, 466, 464, 465, 464, 464, ...
```

```
## Resampling results:
##
##   RMSE     Rsquared    MAE
##   6.51849  0.08897489  4.994942
##
## Tuning parameter 'mtry' was held constant at a value of 2
varImpPlot(rf_lassotest4$finalModel, type = 1)
```




```
# R2: 0.1059
# MSE: 6.596

######
#### predictions
predictionslassotest <- predict(rf_lassotest, test)
validationlassotest <- data.frame(
  R2 = R2(predictionslassotest, test$ sum_arith_perf),
  RMSE = RMSE(predictionslassotest, test$ sum_arith_perf),
  MAE = MAE(predictionslassotest, test$ sum_arith_perf)
)
print(validationlassotest)
##          R2     RMSE      MAE
## 1 0.1239627 6.452196 5.110822


predictionslassotest2 <- predict(rf_lassotest2, test)
```

```r
validationlassotest2 <- data.frame(
  R2 = R2(predictionslassotest2, test$ sum_arith_perf),
  RMSE = RMSE(predictionslassotest2, test$ sum_arith_perf),
  MAE = MAE(predictionslassotest2, test$ sum_arith_perf)
)
print(validationlassotest2)
##           R2     RMSE      MAE
## 1 0.08583951 6.648574 5.247412

predictionslassotest3 <- predict(rf_lassotest3, test)
validationlassotest3 <- data.frame(
  R2 = R2(predictionslassotest3, test$ sum_arith_perf),
  RMSE = RMSE(predictionslassotest3, test$ sum_arith_perf),
  MAE = MAE(predictionslassotest3, test$ sum_arith_perf)
)
print(validationlassotest3)
##          R2     RMSE      MAE
## 1 0.1171929 6.498683 5.129196

predictionslassotest4 <- predict(rf_lassotest4, test)
validationlassotest4 <- data.frame(
  R2 = R2(predictionslassotest4, test$ sum_arith_perf),
  RMSE = RMSE(predictionslassotest4, test$ sum_arith_perf),
  MAE = MAE(predictionslassotest4, test$ sum_arith_perf)
)
print(validationlassotest4)
##           R2     RMSE      MAE
## 1 0.08287062 6.693499 5.285294

predictionslassopretest <- predict(rf_lassopretest, test)
validationlassopretest <- data.frame(
  R2 = R2(predictionslassopretest, test$ sum_arith_perf),
  RMSE = RMSE(predictionslassopretest, test$ sum_arith_perf),
  MAE = MAE(predictionslassopretest, test$ sum_arith_perf)
)
print(validationlassopretest)
##          R2     RMSE      MAE
## 1 0.1198723 6.461348 5.109173


predictionslassopretest2 <- predict(rf_lassopretest2, test)
validationlassopretest2 <- data.frame(
```

```
  R2 = R2(predictionslassopretest2, test$ sum_arith_perf),
  RMSE = RMSE(predictionslassopretest2, test$ sum_arith_perf),
  MAE = MAE(predictionslassopretest2, test$ sum_arith_perf)
)
print(validationlassopretest2)
##          R2     RMSE      MAE
## 1 0.1047693 6.535067 5.122902
```

Overall, the best predictive model of our test set was our fourth selection, which included some anxiety variables. These anxiety variables also showed up when using different values of lambda in the LASSO model, which we will see later on. It does indeed seem as though the inclusions of state anxiety and possibly math anxiety may be able to create a better predictive model than when they are excluded.

Add more here! And review!

I think I should add more interpretation, at least of like. Maybe variable relationships and strength of those relationships IF possible.

Check out lintr! it automatically checks your code to stick to a certain style! You should clean up this code cause its ugly as fuck! or maybe like tidyr worse comes to worst. OR can have your own style, but that sounds hard! ALso one called STYLER! USED IT!!! WHAT DID IT LEAVE LEFT OVER??????

Long lines. Gotta make long lines shorter (maybe can automatically wrap it, but u probably can do it yourself)

```
library(styler)

# style_file("/Users/frankie/Desktop/Dr. Lai Feature Selection Project 9.13.24/Main Script/he
##################
# Load required packages
# library(future)
# library(rmarkdown)

# Set the plan to use all available cores
# plan(multisession)

# Define the knit function
# knit_document <- function(file) {
# rmarkdown::render(file)
# }

# Specify the Quarto file you want to knit
```

```
# quarto_file <- "hello.qmd" # Change this to your actual Quarto file

# Run the knitting in parallel using future
# result <- future({
# knit_document(quarto_file)
# })

# Collect the result (optional, depending on what you want to do with it)
# result <- value(result) # This will retrieve the result, if needed
```