DS3 Hackathon Heart Disease Prediction

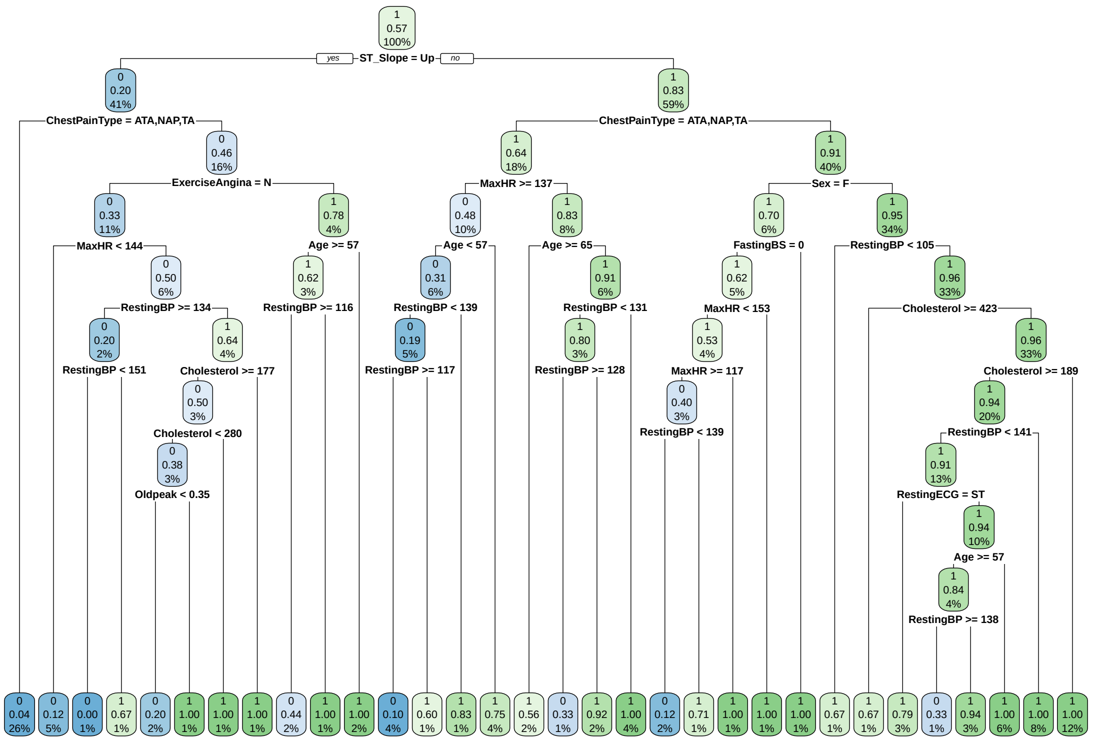
Jeffrey Zhou

Setup packages, load data, and further split the training (labeled) data into a subset for training and a subset for testing

```r
library(tidyverse)
library(rpart)
library(rpart.plot)
library(randomForest)
library(gbm)
library(xgboost)
library(kableExtra)
library(ggplot2)
library(dplyr)
library(caret)
library(RCurl)

tr_url <- "https://raw.githubusercontent.com/jeffreyz374/DS3-Hackathon-2023/main/heart/heart_train.csv"
heart_train <- getURL(tr_url)
heart_train <- read.csv(text = heart_train)
heart_train <- subset(heart_train, select = -c(X, id))

te_url <- "https://raw.githubusercontent.com/jeffreyz374/DS3-Hackathon-2023/main/heart/heart_test.csv"
heart_test <- getURL(te_url)
heart_test <- read.csv(text = heart_test)
heart_test <- subset(heart_test, select = -c(X))

heart_test_no_ids <- subset(heart_test, select = -c(id))

set.seed(1234)
train_idx <- sample(1:nrow(heart_train), round(0.8 * nrow(heart_train)))
train <- heart_train[train_idx,]
test <- heart_train[-train_idx,]
```

The first classifier we will use is the ordinary decision tree

```
heart_tree <- rpart(HeartDisease~., data = train, method = "class",
                    control = list(minsplit = 10), minbucket = 3, cp = 0, xval = 10)

rpart.plot(heart_tree)
```



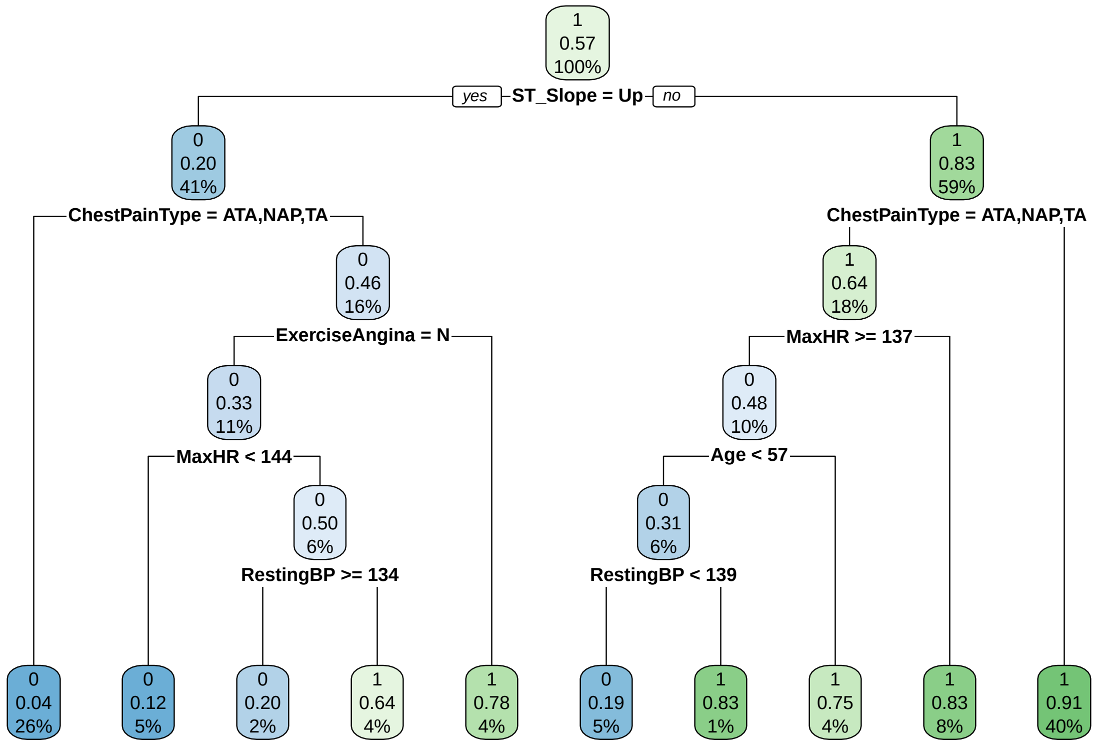
- Since this tree is probably overfitting, we can prune the tree by looking at the complexity parameter:

```
optimalcp <- heart_tree$cptable[which.min(heart_tree$cptable[,"xerror"]), "CP"]
optimalcp
```

```
## [1] 0.00896861
```

- Prune the tree using the best value for the complexity parameter and draw the resulting tree:

```
heart_tree_prune <- prune(heart_tree, cp = optimalcp)

rpart.plot(heart_tree_prune)
```



- Compute the test misclassification error

```
heart_pred <- predict(heart_tree_prune, test)
heart_pred <- as.data.frame(heart_pred)
heart_pred$HeartDisease <- ifelse(heart_pred$"0" > 0.5, 0, 1)

confmatrix_table <- table(true = test$HeartDisease, predicted = heart_pred$HeartDisease)
confmatrix_table
```

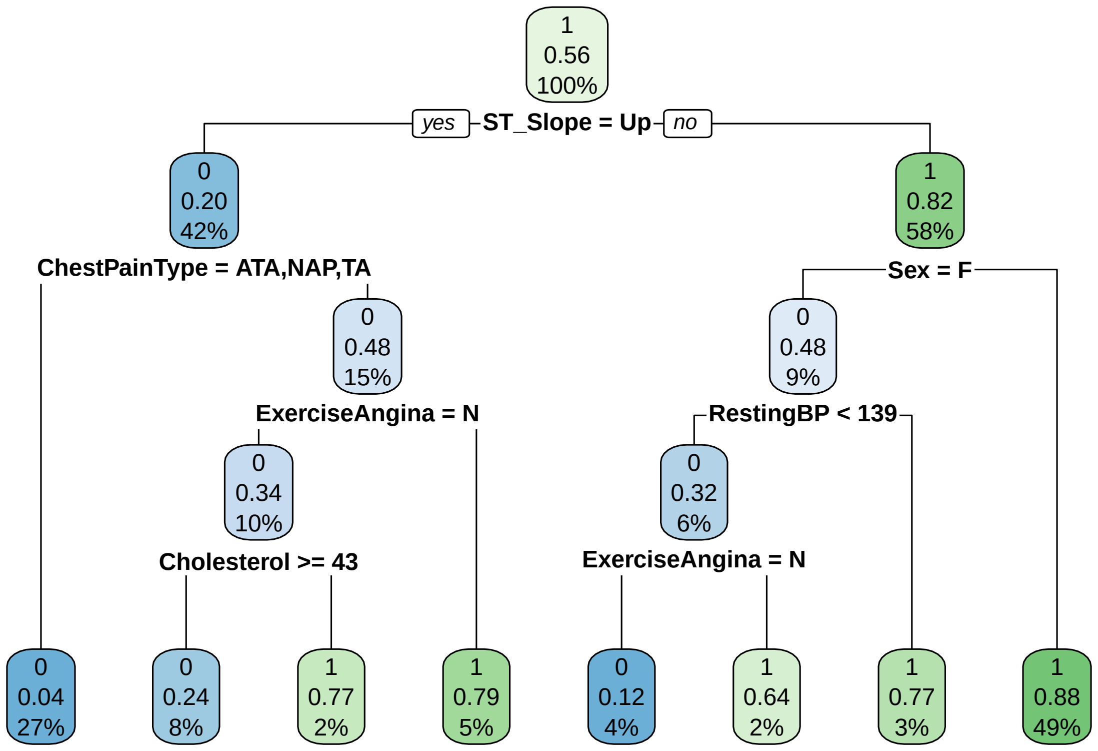
```
##     predicted
## true  0  1
##    0 49 12
##    1  9 59
```

```
misclass_err <- (confmatrix_table[1, 2] + confmatrix_table[2, 1]) / nrow(test)
misclass_err
```

```
## [1] 0.1627907
```

- Fit the tree with the optimal complexity parameter to the full data:

```
heart_tree <- rpart(HeartDisease~., data = heart_train, method = "class",
                    control = list(cp = optimalcp))
rpart.plot(heart_tree)
```



- Predict on testing data:

```
heart_tree_full <- rpart(HeartDisease~., data = heart_train, method = "class",
                         control = list(cp = optimalcp))

heart_pred_full <- predict(heart_tree_full, heart_test_no_ids)
heart_pred_full <- as.data.frame(heart_pred_full)
heart_pred_full$HeartDisease <- ifelse(heart_pred_full$"0" > 0.5, 0, 1)
```

- Join predictions with ids and export:

```
pruned_tree_predictions <- heart_pred_full$HeartDisease
x_pruned <- as.data.frame(list(id = heart_test$id, output = pruned_tree_predictions))

# Uncomment and change the destination directory to export
# write.csv(x_pruned, "~/Downloads/celestialSubmissionPruned.csv", row.names = FALSE)
```

- This classifier ultimately produced an accuracy of 0.8203108

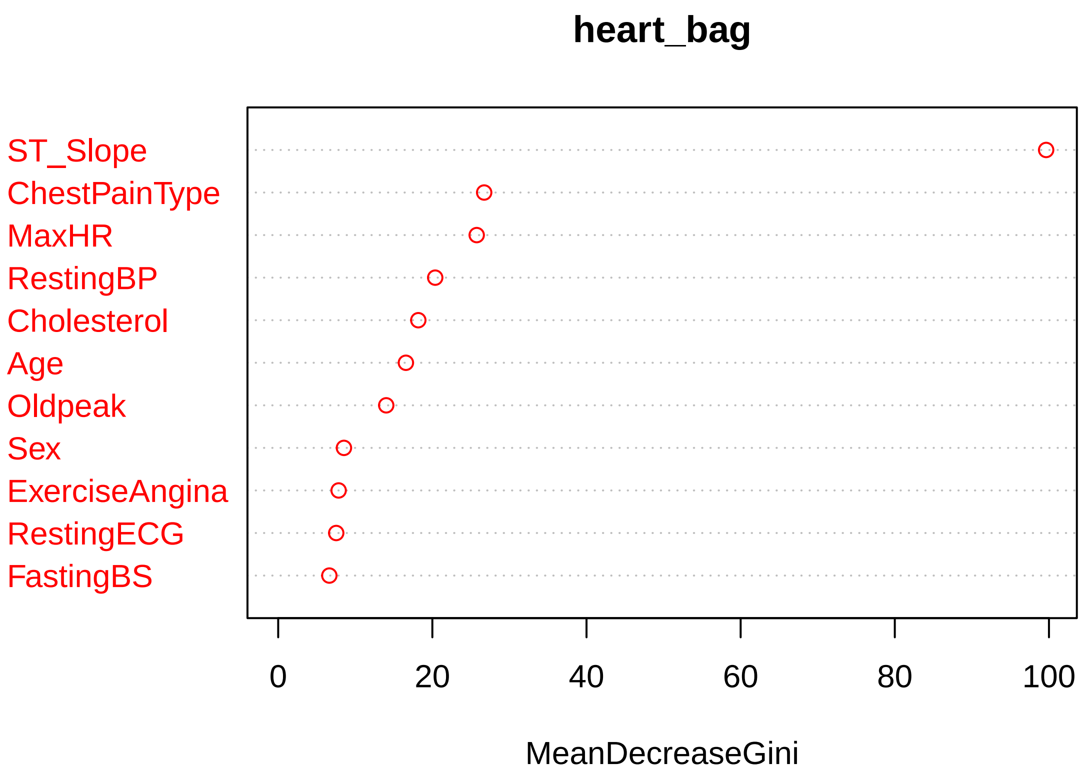
The next classifier we will use is bagging

- We will set `mtry` equal to the number of features (all other parameters at their default values). We will also generate the variable importance plot using `varImpPlot` and extract variable importance from the `randomForest` fitted object using the `importance` function:

```
heart_bag <- randomForest(as.factor(HeartDisease)~., data = train,
                          mtry = 11, na.action = na.omit)

# Error rate
sum(heart_bag$err.rate[,1])
```

```
## [1] 71.03285
```

```
# Variable importance plot
varImpPlot(heart_bag, n.var = 11, col = "red")
```



- Compute the test misclassification error:

```
heart_pred_bag <- predict(heart_bag, test)
heart_pred_bag <- as.data.frame(heart_pred_bag)
heart_pred_bag$HeartDisease <-
  ifelse(heart_pred_bag$heart_pred_bag == test$HeartDisease, 1, 0)
```

```
confmatrix_table <- table(true = test$HeartDisease,
                          predicted = heart_pred_bag$heart_pred_bag)
confmatrix_table
```

```
##      predicted
## true  0  1
##    0 49 12
##    1  7 61
```

```
misclass_err <- (confmatrix_table[1, 2] + confmatrix_table[2, 1]) / nrow(test)
misclass_err
```

```
## [1] 0.1472868
```

- This is a slight improvement from a single decision tree. Now, train it on the full data and predict on the testing data:

```
heart_bag_full <- randomForest(as.factor(HeartDisease)~., data = heart_train,
                               mtry = 11, na.action = na.omit)

heart_bag_pred_full <- predict(heart_bag_full, heart_test_no_ids)
heart_bag_pred_full <- as.data.frame(heart_bag_pred_full)
```

- Join predictions with ids and export:

```
x_bag <- as.data.frame(list(id = heart_test$id, output = heart_bag_pred_full$heart_bag_pred_full))

# Uncomment and change the destination directory to export
# write.csv(x_bag, "~/Downloads/heartSubmissionBag.csv", row.names = FALSE)
```

- This classifier ultimately produced an accuracy of 0.835001

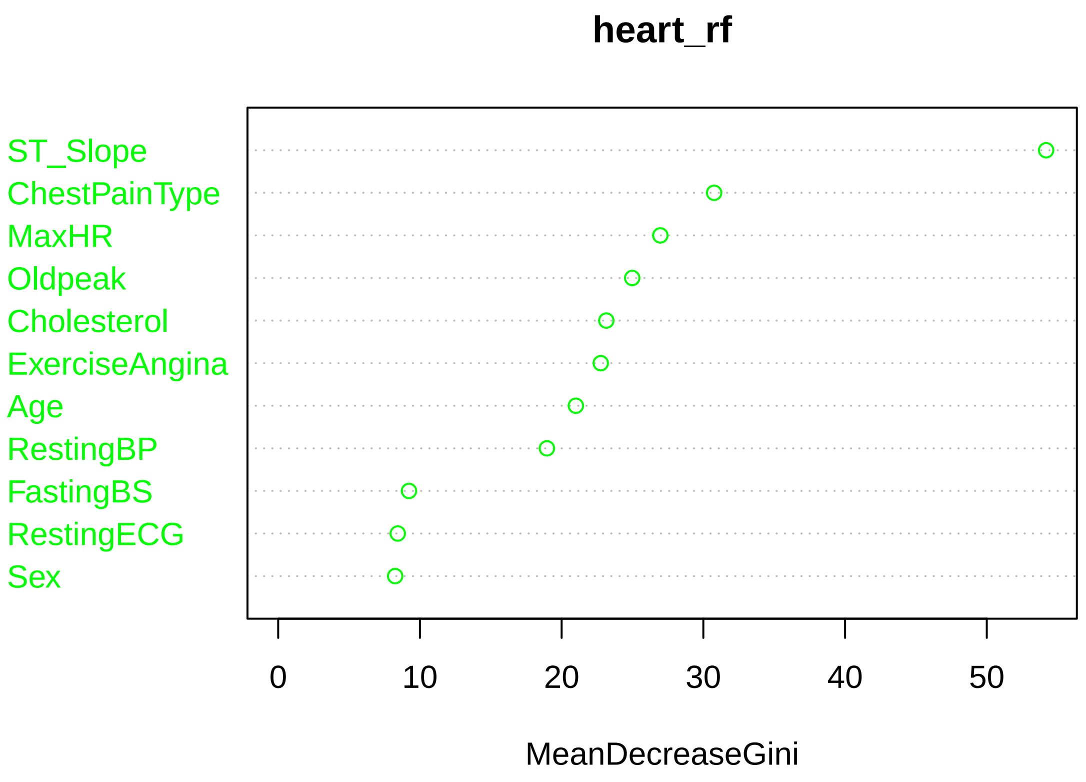
The next classifier we will use is random forest

- This time, we will use `randomForest` with the default parameters. We will again generate the variable importance plot using `varImpPlot` and extract variable importance from the `randomForest` fitted object using the `importance` function

```
heart_rf <- randomForest(as.factor(HeartDisease)~., data = train, na.action = na.omit)

# Error rate
sum(heart_rf$err.rate[,1])
```

```
## [1] 66.15847
```

```
# Variable importance plot
varImpPlot(heart_rf, n.var = 11, col = "green")
```



- Compute the test misclassification error:

```
heart_pred_rf <- predict(heart_rf, test)
heart_pred_rf <- as.data.frame(heart_pred_rf)
heart_pred_rf$HeartDisease <-
  ifelse(heart_pred_rf$heart_pred_rf == test$HeartDisease, 1, 0)

confmatrix_table <- table(true = test$HeartDisease,
                                predicted = heart_pred_rf$heart_pred_rf)
confmatrix_table
```

```
##     predicted
## true  0  1
##    0 51 10
##    1  3 65
```

```
misclass_err <- (confmatrix_table[1, 2] + confmatrix_table[2, 1]) / nrow(test)
misclass_err
```

```
## [1] 0.1007752
```

- This is an improvement from bagging. Now, train it on the full data and predict on the test data:

```
heart_rf_full <- randomForest(as.factor(HeartDisease)~., data = heart_train,
                              na.action = na.omit)

heart_rf_pred_full <- predict(heart_rf_full, heart_test_no_ids)
heart_rf_pred_full <- as.data.frame(heart_rf_pred_full)
```

- Join predictions with ids and export:

```
x_rf <- as.data.frame(list(id = heart_test$id,
                           output = heart_rf_pred_full$heart_rf_pred_full))
# Uncomment and change the destination directory to export
# write.csv(x_rf, "~/Downloads/heartSubmissionRF.csv", row.names = FALSE)
```

- This classifier ultimately produced an accuracy of 0.8530977

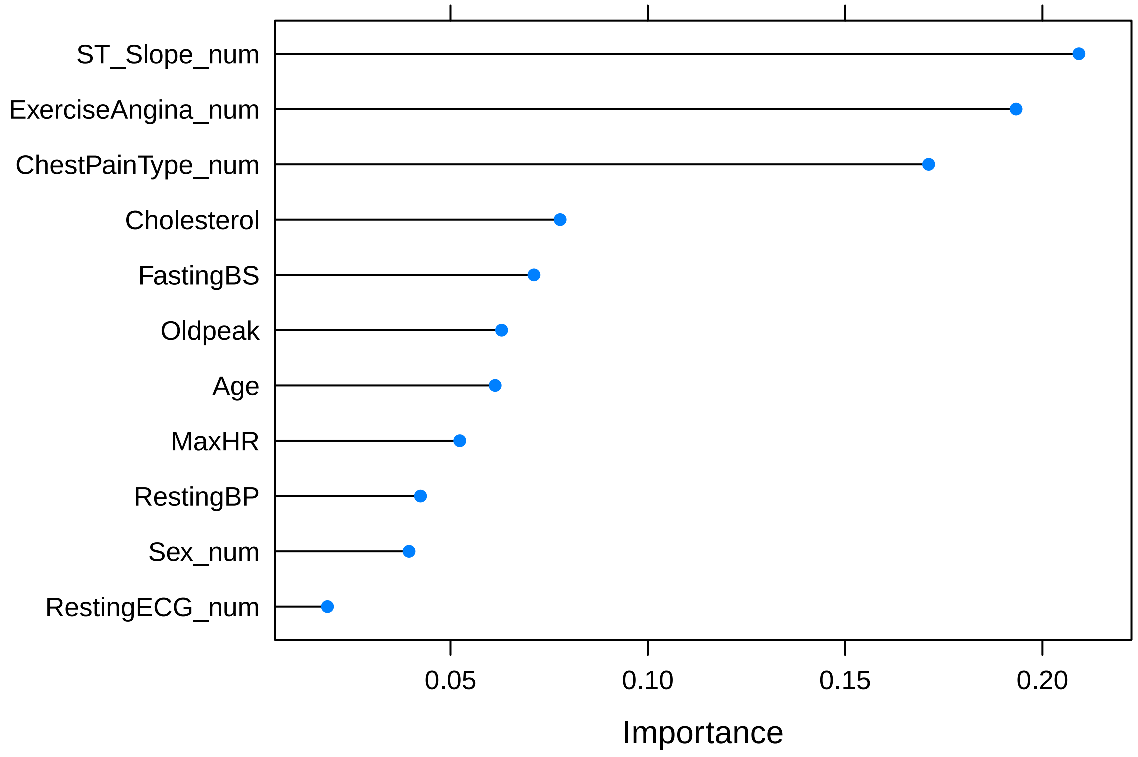
The final classifier we will use is extreme gradient boosting

- Clean the dataset to fit requirements necessary for extreme gradient boosting:

```
set.seed(1234)

# Replace categorical variables with numerical placeholders
heart_train$Sex_num <- ifelse(heart_train$Sex=="M",1,0)

heart_train$ChestPainType_num <-
  ifelse(heart_train$ChestPainType=="ATA", 1,
         ifelse(heart_train$ChestPainType=="NAP", 2,
                ifelse(heart_train$ChestPainType=="ASY",3,0)))

heart_train$RestingECG_num <-
  ifelse(heart_train$RestingECG=="Normal", 1,
         ifelse(heart_train$RestingECG=="ST",2,0))

heart_train$ExerciseAngina_num <- ifelse(heart_train$ExerciseAngina=="Y", 1, 0)

heart_train$ST_Slope_num <-
  ifelse(heart_train$ST_Slope=="Up", 1, ifelse(heart_train$ST_Slope=="Flat", 2, 0))

# Reassign labeled data, now with newly created numerical variables
heart_train <- heart_train %>%
  select(-c(Sex, ChestPainType, RestingECG, ExerciseAngina, ST_Slope))

# Re-split the labeled data into training and testing datasets
train <- sample(1:nrow(heart_train), floor(nrow(heart_train) * 0.8))
test <- setdiff(1:nrow(heart_train), train)
```

- Train extreme gradient boosting model with `xgboost` and perform a grid search for tuning the number of trees and the maximum depth of the tree. Then, we perform 10-fold cross-validation and determine the variable importance:

```
train_control <- trainControl(method = "cv", number = 10, search = "grid")

tune_grid <- expand.grid(max_depth = c(1, 3, 5, 7), nrounds = (1:10) * 50,
                        eta = c(0.01, 0.1, 0.3), gamma = 0, subsample = 1,
                        min_child_weight = 1, colsample_bytree = 0.6)

heart_xgb <- caret::train(HeartDisease~., data = heart_train[train,],
                          method = "xgbTree", trControl = train_control,
                          tuneGrid = tune_grid, verbose = FALSE, verbosity = 0)

# Variable importance plot
plot(varImp(heart_xgb, scale = FALSE))
```



- Compute the test MSE:

```
yhat_xgb <- predict(heart_xgb, newdata = heart_train[test,])
mean((yhat_xgb - heart_train[test, "HeartDisease"]) ** 2)
```

```
## [1] 0.09101545
```

- This is an improvement from random forest. Now, train it on the full data, and predict on the test data:

```
train_control <- trainControl(method = "cv", number = 10, search = "grid")

tune_grid <- expand.grid(max_depth = c(1, 3, 5, 7), nrounds = (1:10) * 50,
                        eta = c(0.01, 0.1, 0.3), gamma = 0, subsample = 1,
                        min_child_weight = 1, colsample_bytree = 0.6)

heart_xgb_full <- caret::train(HeartDisease~., data = heart_train,
                               method = "xgbTree", trControl = train_control,
                               tuneGrid = tune_grid, verbose = FALSE, verbosity = 0)

heart_test$Sex_num <- ifelse(heart_test$Sex=="M",1,0)

heart_test$ChestPainType_num <-
  ifelse(heart_test$ChestPainType=="ATA", 1,
         ifelse(heart_test$ChestPainType=="NAP", 2,
                ifelse(heart_test$ChestPainType=="ASY", 3, 0)))

heart_test$RestingECG_num <-
  ifelse(heart_test$RestingECG=="Normal", 1,
         ifelse(heart_test$RestingECG=="ST", 2, 0))

heart_test$ExerciseAngina_num <- ifelse(heart_test$ExerciseAngina=="Y", 1, 0)

heart_test$ST_Slope_num <-
  ifelse(heart_test$ST_Slope=="Up", 1, ifelse(heart_test$ST_Slope=="Flat", 2, 0))

heart_test_no_ids <- heart_test %>%
  select(-c(id, Sex, ChestPainType, RestingECG, ExerciseAngina, ST_Slope))

heart_xgb_pred_full <- predict(heart_xgb_full, heart_test_no_ids)
heart_xgb_pred_full <- as.data.frame(heart_xgb_pred_full)
```

- Join predictions with ids and export:

```
x_xgb <- as.data.frame(list(id = heart_test$id,
                            output = ifelse(heart_xgb_pred_full$heart_xgb_pred_full >= 0.5, 1, 0)))

# Uncomment and change the destination directory to export
# write.csv(x_xgb, "~/Downloads/celestialSubmissionXGB.csv", row.names = FALSE)
```

- This classifier ultimately produced an accuracy of 0.8595912

Conclusion

All in all, the accuracies we received can be summarized in the below table:

```
accs <- c("0.8203108", "0.835001", "0.8530977", "0.8595912")
rownames <- c("Pruned Decision Tree", "Bagging", "Random Forest", "Extreme Gradient Boosting")
colname <- c("Accuracy")

acc_table <- cbind(accs)
rownames(acc_table) <- rownames
colnames(acc_table) <- colname

acc_table %>%
  kable(align = c("r"))
```

	Accuracy
Pruned Decision Tree	0.8203108
Bagging	0.835001
Random Forest	0.8530977
Extreme Gradient Boosting	0.8595912