Appendix C: Code

INTRODUCTION

```r
# SETTING EVERYTHING UP
library(imputeTS)
library(ggplot2)
library(scales)
library(gridExtra)
library(knitr)
library(xtable)
library(kableExtra)
library(dplyr)
library(ggpubr)
library(GGally)
library(caret)
library(mlbench)
library(tidyverse)
library(broom)
library(grid)
library(naivebayes)
library(klaR)
library(xgboost)
library(heplots)
library(class)
library(dbscan)
library(factoextra)

# Load in the models
setwd("C:/Users/asada/OneDrive/Desktop/Term 3B/STAT 441/Project/STAT 441 - Angela")
load("ldaFit.rda")
load("ldaFitS.rda")
load("qdaFit.rda")
load("rdaFit.rda")
load("lda.final.rda")
load("lda.test.pred.rda")
load("logistic.pca.rda")
load("logistic.interaction.varselect.rda")
load("logistic.interaction.rda")
load("logistic.vanilla.rda")
load("logistic.smooth.rda")
load("final.logistic.rda")
load("final.nb.rda")
load("nb.var.select.rda")
load("nb.pca.rda")
load("nb.rda")
load("rf_selected.rda")
load("rf_selected_test.rda")
load("rf_importance_selected.rda")
load("rf_check.rda")
load("rf_fit.rda")
load("knn_best_sub.rda")
load("knn_selected.rda")
load("knn_fit_vs.rda")
load("knn_fit_pc.rda")
load("knn_fit_all.rda")
load("xgbTrain_final.rda")
load("xgb.rfe.rda")
load("xgbRed.rda")
load("xgbPred.rda")

# Read in the data
dat <- read.csv("Indian Liver Patient Dataset (ILPD).csv", header=F)

# Define function
g_legend<-function(a.gplot){
  tmp <- ggplot_gtable(ggplot_build(a.gplot))
  leg <- which(sapply(tmp$grobs, function(x) x$name) == "guide-box")
  legend <- tmp$grobs[[leg]]
  return(legend)}

### DATA READING

# Note that 2 = Liver Disease and 1 = No Liver Disease for the diagnosis variable; rescale them
colnames(dat) <- c("Age", "Gender", "TB", "DB", "ALP", "ALT", "AST", "TP", "ALB", "RAG", "Diagnosis")
dat$Diagnosis[dat$Diagnosis==1] <- "Liver Disease"
dat$Diagnosis[dat$Diagnosis==2] <- "No Disease"

# Add the log transformed versions of variables
dat$logDB <- log(dat$DB)
dat$logTB <- log(dat$TB)
dat$logALP <- log(dat$ALP)
dat$logALT <- log(dat$ALT)
dat$logAST <- log(dat$AST)
dat$logTP <- log(dat$TP)

# Split the dataset into training and test (80/20 split)
set.seed(999)
perm<-sample(x=nrow(dat))
train <- dat[which(perm<=465),] # Training set
test <- dat[which(perm>465),] # Test set

# Impute missing values in RAG with sample mean
train <- na.mean(train)
test <- na.mean(test)

# Create the predictor matrix and response
predictors <- train[,c("Age", "Gender", "logTB", "logDB", "logALT", "logALP",
                             "logAST", "logTP", "ALB", "RAG")]
binary.response <- ifelse(train$Diagnosis=="Liver Disease", y=1, n=0)
test$binary.response <- ifelse(test$Diagnosis=="Liver Disease", y=1, n=0)

# OUTLIER DETECTION
# Load libraries required

# Set seed for reproducibility
set.seed(999)

# Add gender indicator and prepare matrix for dbscan function
gender.ind.train <- ifelse(train$Gender == "Male",1,0)
train.outlier.detect <- data.frame(train[,names(train)%in%c(
  "Age","TB","DB","ALP","ALT","AST","TP","ALB","RAG")], gender.ind.train)

# Check optimal values for epsilon for a given minimum number of points in cluster k
# k=10
dbscan::kNNdistplot(train.outlier.detect, k=(2*10))
abline(h=220)

# Run dbscan
db <- dbscan(train.outlier.detect,eps=220,minPts=10)
fviz_cluster(db, train.outlier.detect, geom = "point") +
  theme(plot.title = element_text(hjust = 0.5))

### DATA VIZ

# Create a data visualization function

# Creates a histogram for a single variable over the entire dataset
dataviz1 <- function(data, variable, colour, nbin)
{
  ggplot(data, aes_string(x=variable)) +
    geom_histogram(fill=colour, color='black', bins=nbin, alpha=0.7) +
    ylab("Frequency") + xlab(variable) +
    ggtitle(paste("Histogram of", variable, "(All Individuals)")) +
    theme(plot.title = element_text(size = 10))
}

# Creates separate density plots by diagnosis status
dataviz2 <- function(data, variable, colour2){
  ggplot(data, aes_string(x=variable)) + geom_density(fill=colour2, alpha=0.8) +
    ylab("Density") + xlab(variable) +
    ggtitle(paste("Density of", variable, "by Diagnosis")) + facet_grid(rows=vars(Diagnosis)) +
    theme(plot.title = element_text(size = 10))
}

# Create a table function which produces summaries for a variable for each group
tablefn <- function(v1, v2){
  t1 <- as.array(summary(v1))
  t2 <- as.array(summary(v2))
  names(t1) <- c("Minimum", "1st Quartile", "Median", "Mean", "3rd Quartile", "Maximum")
  names(t2) <- c("Minimum", "1st Quartile", "Median", "Mean", "3rd Quartile", "Maximum")
  knitr::kable(list(t1,t2),
               caption = "Summary for Non-Diseased Patients (left) VS Diseased (right)",
               col.names = c("Summary Statistic", "Value"), digits=2)
  %>% kable_styling(latex_options="hold_position")
}
```

```r
# Create subsets of training set for diseased and non-diseased (to be used in tablefn later)
train_disease <- subset(train, Diagnosis=="Liver Disease")
train_control <- subset(train, Diagnosis=="No Disease")

# Combine results; this produces a histogram (all data), two densities (one per diagnosis),
# and a table (one per diagnosis)
dataviz <- function(data, variable, colour, colour2, nbin, v1, v2){
 p1 <- dataviz1(data, variable, colour, nbin=nbin)
 p2 <- dataviz2(data, variable, colour2)
 grid.arrange(p1, p2, nrow=1)
 tablefn(v1, v2)
}

# Make all the plots
dataviz(data=train, variable="Age", colour="coral4",
        colour2="coral3", nbin=10, train_control$Age, train_disease$Age)

ggplot(train, aes(x=train$Gender, fill=Diagnosis)) +
  geom_bar(color='black', alpha=0.6) +
  ylab("Frequency") + xlab("Gender") + ggtitle("Summary of Gender by Diagnosis") +
  scale_fill_manual(values=c("darkorchid4", "blue"))

kable(table(train$Gender, train$Diagnosis),
      caption="Contingency Table of Gender and Diagnosis")

dataviz(train, "logTB", "aquamarine4", "aquamarine3", nbin=6,
        train_control$logTB, train_disease$logTB)

dataviz(train, "logDB", "cadetblue4", "cadetblue3", nbin=5,
        train_control$logDB, train_disease$logDB)

dataviz(train, "logALP", "darkolivegreen", "darkolivegreen4",
        nbin=10, train_control$logALP, train_disease$logALP)

dataviz(train, "logALT", "firebrick", "firebrick3", nbin=8,
        train_control$logALT, train_disease$logALT)

dataviz(train, "logAST", "deeppink4", "deeppink3", nbin=8,
        train_control$logAST, train_disease$logAST)

dataviz(train, "logTP", "goldenrod4", "goldenrod3", nbin=8,
        train_control$logTP, train_disease$logTP)

dataviz(train, "ALB", "lightpink4", "lightpink3", nbin=8,
        train_control$ALB, train_disease$ALB)

dataviz(train, "RAG", "sienna4", "sienna3", nbin=8,
        train_control$RAG, train_disease$RAG)
```

ANALYSIS

KNN

```r
# KNN

##check overall variable importance

##Preprocessing
train2 <- train[,c("Age", "Gender", "TB", "DB", "ALT", "ALP",
                             "AST", "TP", "ALB", "RAG", "Diagnosis")]
train2$Gender <- as.numeric(train2$Gender)
train2$Diagnosis <- as.factor(train2$Diagnosis)

##Fitting model
trctrl <- trainControl(method= "repeatedcv", repeats= 5, number = 5, savePredictions = T)
knn_all_grid <- expand.grid(k=c(5*c(1:15)))
knn_fit_all <- train(Diagnosis~. , data = train2, method="knn",
                     preProcess= c("center","scale"),
                     trControl=trctrl, tuneGrid = knn_all_grid)

##Find the best subset
##Preprocessing
train2x <- train2[,names(train2) != "Diagnosis"]
preproc_xvals <- preProcess(train2x)
preproc_train2x <- predict(preproc_xvals, train2x)
preproc_train2x <- as.data.frame(preproc_train2x)
custom_knn_Diagnosis <- ifelse(train2$Diagnosis=="Liver Disease",y="Liver.Disease",n="No.Disease")
custom_knn_Diagnosis <- as.factor(custom_knn_Diagnosis)

##Fitting model
rfectrl <- rfeControl(functions = caretFuncs,
                      method = "repeatedcv", repeats = 1, number =  5, verbose = FALSE)
subsets <- c(1:10)
knn_best_sub <- rfe(preproc_train2x, custom_knn_Diagnosis,
                    sizes = subsets, rfeControl = rfectrl, method="knn")

##Importance Results
knn_importance_all <- varImp(knn_fit_all, scale=FALSE)
p1<- ggplot(knn_importance_all)+ggtitle("Variable Importance")+
  theme(plot.title = element_text(hjust = 0.5))

##Results
var_names <- knn_best_sub$control$functions$selectVar(knn_best_sub$variables,10)
p2<- ggplot(knn_best_sub)+ggtitle("Recursive Feature Elimination")+
  theme(plot.title = element_text(hjust = 0.5))+
  scale_x_continuous(name="Cumulatively Added Variables",
                     breaks=c(1:10),labels=var_names)

grid.arrange(p1, p2, ncol=2, nrow=1)


##Preprocessing
```

```r
train2 <- train[,c("Age", "Gender", "TB", "DB", "ALT", "ALP",
                             "AST", "TP", "ALB", "RAG", "Diagnosis")]
train2$Gender <- as.numeric(train2$Gender)
##train2$Diagnosis <- ifelse(train2$Diagnosis=="Liver Disease", y=1, n=0)
train2$Diagnosis <- as.factor(train2$Diagnosis)

##Fitting model
trctrl <- trainControl(method= "repeatedcv", repeats= 5, number = 5, savePredictions = T)
knn_grid <- expand.grid(k=c(5*c(1:15)))
knn_fit_pc <- train(Diagnosis~. , data = train2, method="knn",
                    preProcess= c("center","scale","pca"),
                    trControl=trctrl, tuneGrid = knn_grid)

p1<- ggplot(knn_fit_all)+ggtitle("Parameter Selection with All Variables")+
  theme(plot.title = element_text(hjust = 0.5,size=8))+
  ylab("Accuracy")+xlab("Neighbours")
p2<- ggplot(knn_fit_vs)+ggtitle("Parameter Selection with Selected Variables")+
  theme(plot.title = element_text(hjust = 0.5,size=8))+
  ylab("Accuracy")+xlab("Neighbours")
p3<- ggplot(knn_fit_pc)+ggtitle("Parameter Selection with PCA")+
  theme(plot.title = element_text(hjust = 0.5,size=8))+
  ylab("Accuracy")+xlab("Neighbours")

grid.arrange(p1, p2, p3, ncol=2, nrow=2,
             top = textGrob("Parameter Selection via Out-of-Sample Accuracy",
                            gp=gpar(fontsize=12)))

# Now, create summary tables
oos.errors <- c(round(getTrainPerf(knn_fit_all)[1,1],3),
                round(getTrainPerf(knn_fit_vs)[1,1],3),
                round(getTrainPerf(knn_fit_pc)[1,1],3))
kappa.vals <- c(round(getTrainPerf(knn_fit_all)[1,2],3),
                round(getTrainPerf(knn_fit_vs)[1,2],3),
                round(getTrainPerf(knn_fit_pc)[1,2],3))

labels <- c("KNN with All Predictors", "KNN with Variable Selection","KNN with PCA")
summary.table <- data.frame(labels, oos.errors, kappa.vals)
colnames(summary.table) <- c("Model", "Mean Out of Sample Accuracy", "Mean Kappa Value")
t1 <- tableGrob(summary.table, rows=NULL, theme=ttheme_default(base_size = 10))

# Create error plot; get Accuracy and Kappa per model
df <- data.frame(
  Indices=rep(1:25, 3),
  Accuracy=c(knn_fit_all$resample$Accuracy, knn_fit_vs$resample$Accuracy,
             knn_fit_pc$resample$Accuracy),
  Kappa = c(knn_fit_all$resample$Kappa, knn_fit_vs$resample$Kappa,
            knn_fit_pc$resample$Kappa),
  Model=c(rep("KNN with All Predictors", 25),
          rep("KNN with Variable Selection", 25),
          rep("KNN with PCA",25)))

p1 <- ggplot(data = df, aes(x=Indices, y=Accuracy)) + geom_line(aes(colour=Model)) +
  ggtitle("Out-of-Sample Prediction Accuracy by Model") + theme(legend.position="top") +
  theme(plot.title = element_text(size=10), legend.text=element_text(size=8)) +
  guides(colour=guide_legend(nrow=2))

p2 <- ggplot(data = df, aes(x=Indices, y=Kappa)) + geom_line(aes(colour=Model)) +
  ggtitle("Out-of-Sample Kappa Values by Model") +
  theme(legend.position = "none", plot.title = element_text(size=10))

g_legend<-function(a.gplot){
  tmp <- ggplot_gtable(ggplot_build(a.gplot))
  leg <- which(sapply(tmp$grobs, function(x) x$name) == "guide-box")
  legend <- tmp$grobs[[leg]]
  return(legend)}

mylegend<-g_legend(p1)

grid.arrange(arrangeGrob(p1 + theme(legend.position="none"),
                         p2 + theme(legend.position="none"),
                         nrow=1), mylegend, t1,  nrow=3, heights=c(5, 2, 4))

knn_importance_vs <- varImp(knn_fit_vs, scale=FALSE)

ggplot(knn_importance_vs)+ggtitle("Variable Importance for Selected Variables")+
  theme(plot.title = element_text(hjust = 0.5))
```

DISCRIMINANT ANALYSIS

```
train$gender.ind <- ifelse(train$Gender=="Male",1,0)
test$gender.ind <- ifelse(test$Gender=="Male",1,0)

boxm <- heplots::boxM(train[,names(train)%in%c(
  "Age","TB","DB","ALP","ALT","AST","TP","ALB","RAG", "gender.ind")],train$Diagnosis)

# Training controls for discriminant analyses
da.control <- trainControl(method = "repeatedcv", number = 5, repeats=5,
                           savePredictions=TRUE, verbose=FALSE)

# Fit LDA classifier on original training data
ldaFit <- train(x=train[,names(train)%in%c("Age","TB","DB","ALP","ALT","AST",
                                          "TP","ALB","RAG","gender.ind")],
  y=train$Diagnosis,method="lda",
  trControl=da.control)
ldaFit

# Scale the predictors
trainScaled <- apply(train[,names(train)%in%c("Age","TB","DB","ALP","ALT","AST",
                                             "TP","ALB","RAG","gender.ind")],2,scale)

# Fit LDA classifier on scaled training data for easier interpretability of
# the discriminant functions
ldaFitS <- train(x=trainScaled,y=train$Diagnosis, method="lda",
                 trControl=da.control)
ldaFitS
```

```r
# Fit QDA classifier on original training data
qdaFit <- train(x=train[,names(train)%in%c("Age","TB","DB","ALP","ALT","AST",
                                           "TP","ALB","RAG","gender.ind")],
  y=train$Diagnosis,method="qda",
  trControl=da.control)
qdaFit

# Fit regularized da classifier on original training data

# Need klaR package
library(klaR)

# Note that the rda function can find the optimized parameter
# values for the regularization terms using a
# Nelder-Mead-Simplex algorithm

# Set up train control for rda
rda.control <- trainControl(method = "repeatedcv", number = 5, repeats=5,
                            savePredictions=TRUE, verbose=FALSE)

rdaFit <- train(x=train[,names(train)%in%c("Age","TB","DB","ALP","ALT","AST",
                                           "TP","ALB","RAG","gender.ind")],
  y=train$Diagnosis,method="rda",
  trControl=rda.control)
rdaFit

# Now, create summary tables; we want out-of-sample accuracy and Kappa values
oos.errors <- c(round(getTrainPerf(ldaFit)[1,1],3),
                round(getTrainPerf(qdaFit)[1,1],3),
                round(getTrainPerf(rdaFit)[1,1],3))
kappa.vals <- c(round(getTrainPerf(ldaFit)[1,2],3),
                round(getTrainPerf(qdaFit)[1,2],3),
                round(getTrainPerf(rdaFit)[1,2],3))

labels <- c("LDA", "QDA", "RDA")
summary.table <- data.frame(labels, oos.errors, kappa.vals)
colnames(summary.table) <- c("Model", "Mean Out of Sample Accuracy", "Mean Kappa Value")
t1 <- tableGrob(summary.table, rows=NULL, theme=ttheme_default(base_size = 10))

# Create accuracy and Kappa plot
df <- data.frame(
  Indices=rep(1:25, 3),
  Accuracy=c(ldaFit$resample$Accuracy, qdaFit$resample$Accuracy,
             rdaFit$resample$Accuracy),
  Kappa = c(ldaFit$resample$Kappa, qdaFit$resample$Kappa,
            rdaFit$resample$Kappa),
  Model=c(rep("LDA", 25),
          rep("QDA", 25),
          rep("RDA", 25)))

# Plot accuracy and Kappa values over the 25 folds, and display them with the summary table
p1 <- ggplot(data = df, aes(x=Indices, y=Accuracy)) + geom_line(aes(colour=Model)) +
  ggtitle("Out-of-Sample Prediction Accuracy by Model") + theme(legend.position="top") +
  theme(plot.title = element_text(size=10), legend.text=element_text(size=8)) +
  guides(colour=guide_legend(nrow=1))

p2 <- ggplot(data = df, aes(x=Indices, y=Kappa)) + geom_line(aes(colour=Model)) +
  ggtitle("Out-of-Sample Kappa Values by Model") +
  theme(legend.position = "none", plot.title = element_text(size=10))

mylegend <- g_legend(p1)

grid.arrange(arrangeGrob(p1 + theme(legend.position="none"),
                         p2 + theme(legend.position="none"),
                         nrow=1), mylegend, t1,  nrow=3, heights=c(5, 2, 4))

lda.test.pred <- predict(ldaFit, newdata=test[,-c(2,18)])

lda.final <- lda(Diagnosis ~ Age+Gender+TB+DB+ALP+ALT+AST+TP+ALB+RAG, data=train)

plot(lda.final, dimen=1, type="histogram",
     main="Values Along First Canonical Variate")
```

RANDOM FOREST

```r
##Preprocessing
train2 <- train[,c("Age", "Gender", "TB", "DB", "ALT", "ALP",
                              "AST", "TP", "ALB", "RAG", "Diagnosis")]
train2$Gender <- as.numeric(train2$Gender)
train2$Diagnosis <- as.factor(train2$Diagnosis)

##Find the best subset
rfectrl <- rfeControl(functions = rfFuncs, method = "repeatedcv",
                      repeats = 1, number =  5, verbose = FALSE)
subsets <- c(1:10)
rf_check <- rfe(train2[,names(train2) != "Diagnosis"],
                train2$Diagnosis, sizes = subsets, rfeControl = rfectrl)

##Results
var_names <- rf_check$control$functions$selectVar(rf_check$variables,10)
ggplot(rf_check)+ggtitle("Recursive Feature Elimination")+
  theme(plot.title = element_text(hjust = 0.5))+
  scale_x_continuous(name="Cumulatively Added Variables",
                     breaks=c(1:10),labels=var_names)

##Create a function to tune ntree and mtry simultaneously using train from Caret
RF_new <- list(type = "Classification", library = "randomForest", loop = NULL)
RF_new$parameters <- data.frame(parameter = c("mtry", "ntree"),
                                class = rep("numeric", 2), label = c("mtry", "ntree"))
RF_new$grid <- function(x, y, len = NULL, search = "grid") {}
RF_new$fit <- function(x, y, wts, param, lev, last, weights, classProbs, ...) {
  randomForest(x, y, mtry = param$mtry, ntree=param$ntree, ...)
}
RF_new$predict <- function(modelFit, newdata, preProc = NULL, submodels = NULL)
   predict(modelFit, newdata)
```

```r
RF_new$prob <- function(modelFit, newdata, preProc = NULL, submodels = NULL)
   predict(modelFit, newdata, type = "prob")
RF_new$sort <- function(x) x[order(x[,1]),]
RF_new$levels <- function(x) x$classes


##Fit model with selected variables
##Preprocessing
train4 <- train2[,names(train2) != "TP"]

##Find optimum values for tuning parameters ntree and mtry
rf_control <- trainControl(method="repeatedcv", number=5, repeats=5)
rf_grid <- expand.grid(.mtry=c(1,2,3,5,7,9), .ntree=c(50,100,250,500,1000,1500))
rf_fit <- train(Diagnosis~., data=train4, method=RF_new,
                tuneGrid=rf_grid, trControl=rf_control)

##plot the results
p1<- plot(rf_fit)
p2<- plot(rf_fit,
     plotType = "line",
     metric = rf_fit$perfNames[2])

grid.arrange(p1, p2, ncol=2, nrow=1)


# Now, create summary tables
oos.errors <- c(round(rf_fit$results[rf_fit$results$mtry==3&rf_fit$results$ntree==1000,][3],3))
kappa.vals <- c(round(rf_fit$results[rf_fit$results$mtry==3&rf_fit$results$ntree==1000,][4],3))

labels <- c("Random Forest")
summary.table <- data.frame(labels, oos.errors, kappa.vals)
colnames(summary.table) <- c("Model", "Mean Out of Sample Accuracy", "Mean Kappa Value")
t1 <- tableGrob(summary.table, rows=NULL, theme=ttheme_default(base_size = 10))
grid.arrange(t1,nrow=1)

rf_grid_selected <- expand.grid(.mtry=3, .ntree=1000)
rf_fit_selected <- train(Diagnosis~., data=train4,
                         method=RF_new, tuneGrid=rf_grid_selected, trControl=rf_control)
rf_importance_selected <- varImp(rf_fit_selected, scale=FALSE)
ggplot(rf_importance_selected)+ggtitle("Variable Importance")+
  theme(plot.title = element_text(hjust = 0.5))
```

NAIVE BAYES

```r
set.seed(999)
# Initialize the 5-fold CV
NB.control <- trainControl(method = "repeatedcv", number = 5, repeats=5, savePredictions = T)
search_grid <- expand.grid(usekernel = c(TRUE, FALSE), laplace = c(0),
                           adjust = seq(0.1, 8, by = 0.2))

# Try with variable selection
control <- rfeControl(functions=rfFuncs, method="cv", number=5)
```

```r
results <- rfe(x=predictors, y=factor(binary.response), sizes=c(1:8), rfeControl=control)
opt.vars <- results$optVariables

# Consider the following
# Using kernel=FALSE means we assume Gaussian distribution for each continuous predictor
# If Kernel=TRUE, we use kernel density estimation
# Adjust is our bandwidth parameter (note if kernel=FALSE and we change this, no effect)

# Naive Bayes with PCA
nb.pca <- train(
  x = predictors,
  y = factor(binary.response),
  method = "naive_bayes",
  trControl = NB.control,
  tuneGrid = search_grid,
  preProc = c("pca")
  )

# Naive Bayes (default setting, no variable selection or PCA)
nb <- train(
  x = predictors,
  y = factor(binary.response),
  method = "naive_bayes",
  trControl = NB.control,
  tuneGrid = search_grid)

# Naive Bayes with recursive feature elimination
nb.var.select <- train(
  x = predictors[results$optVariables],
  y = factor(binary.response),
  method = "naive_bayes",
  trControl = NB.control,
  tuneGrid = search_grid)

# Results, plot the optimal tuning parameter values
p1 <- plot(nb.pca, main = list("Naive Bayes with PCA", cex=0.75),
           ylim = c(0.4, 0.8), ylab = "", xlab=list("Bandwidth", cex=0.75),
           par.settings = list(fontsize = list(text = 10, points = 10)))
p2 <- plot(nb, main = list("Naive Bayes without PCA", cex=0.75),
           ylim = c(0.4, 0.8), ylab = "", xlab=list("Bandwidth", cex=0.75),
           par.settings = list(fontsize = list(text = 10, points = 10)))
p3 <- plot(nb.var.select, main = list("Naive Bayes with Variable Selection", cex=0.75),
           ylim = c(0.4, 0.8), ylab="", xlab=list("Bandwidth", cex=0.75),
           par.settings = list(fontsize = list(text = 10, points = 10)))
grid.arrange(p1, p2, p3, ncol=2, nrow=2,
             top = textGrob("Hyperparameter Tuning via Out-of-Sample Accuracy",
                            gp=gpar(fontsize=14, col = "brown4")))

# Now, create summary tables
oos.errors <- c(round(getTrainPerf(nb)[1,1],3),
                round(getTrainPerf(nb.pca)[1,1],3),
                round(getTrainPerf(nb.var.select)[1,1],3))
kappa.vals <- c(round(getTrainPerf(nb)[1,2],3),
                round(getTrainPerf(nb.pca)[1,2],3),
                round(getTrainPerf(nb.var.select)[1,2],3))

labels <- c("Naive Bayes with All Predictors", "Naive Bayes with PCA",
            "Naive Bayes with Variable Selection")
summary.table <- data.frame(labels, oos.errors, kappa.vals)
colnames(summary.table) <- c("Model", "Mean Out of Sample Accuracy", "Mean Kappa Value")
t1 <- tableGrob(summary.table, rows=NULL, theme=ttheme_default(base_size = 10))

# Create error plot; get Accuracy and Kappa per model
df <- data.frame(
  Indices=rep(1:25, 3),
  Accuracy=c(nb$resample$Accuracy, nb.pca$resample$Accuracy,
             nb.var.select$resample$Accuracy),
  Kappa = c(nb$resample$Kappa, nb.pca$resample$Kappa, nb.var.select$resample$Kappa),
  Model=c(rep("Naive Bayes with All Predictors", 25),
                rep("Naive Bayes with PCA", 25),
                rep("Naive Bayes with Variable Selection", 25)))

p1 <- ggplot(data = df, aes(x=Indices, y=Accuracy)) + geom_line(aes(colour=Model)) +
  ggtitle("Out-of-Sample Prediction Accuracy by Model") + theme(legend.position="top") +
  theme(plot.title = element_text(size=10), legend.text=element_text(size=8)) +
  guides(colour=guide_legend(nrow=2))

p2 <- ggplot(data = df, aes(x=Indices, y=Kappa)) + geom_line(aes(colour=Model)) +
  ggtitle("Out-of-Sample Kappa Values by Model") +
  theme(legend.position = "none", plot.title = element_text(size=10))

mylegend<-g_legend(p1)

grid.arrange(arrangeGrob(p1 + theme(legend.position="none"),
                         p2 + theme(legend.position="none"),
                         nrow=1), mylegend, t1,  nrow=3, heights=c(5, 2, 4))

ggplot(varImp(nb.var.select))
```

XGBOOST

```r
# Set seed for reproducibility
set.seed(999)

# Use caret package to train the xgboost classifier!
xgb.control <- trainControl(method="cv", number=5, savePredictions=TRUE,
                            verboseIter=FALSE, returnData=FALSE)

# Add gender indicator
train$gender.ind <- ifelse(train$Gender=="Male",1,0)
test$gender.ind <- ifelse(test$Gender=="Male",1,0)

# Transform training and test sets into xgb.DMatrix objects
xgbTrainX <- xgb.DMatrix(data=as.matrix(train[,names(train)%in%c(
  "Age","TB","DB","ALP","ALT","AST","TP","ALB","RAG","gender.ind")]))
```

```r
xgbTrainY <- train$Diagnosis
xgbTestX <- xgb.DMatrix(data=as.matrix(test[,names(test)%in%c(
  "Age","TB","DB","ALP","ALT","AST","TP","ALB","RAG","gender.ind")]))
xgbTestY <- test$Diagnosis

paramGrid <- expand.grid(eta=c(0.01,0.1,0.3,0.5),
                         gamma=c(0,0.1,0.2,0.3,0.4),
                         max_depth=2*(1:3)-1,
                         min_child_weight=c(5,15),
                         subsample=c(0.5,0.75,1),
                         colsample_bytree=c(0.5,0.75,1),
                         nrounds=c(100,300,500))

xgbTrain <- caret::train(x=xgbTrainX,y=xgbTrainY,method="xgbTree",
                         trControl=xgb.control, tuneGrid=paramGrid, nthread=1)

set.seed(999)

# Add gender indicator
train$gender.ind <- ifelse(train$Gender=="Male",1,0)
test$gender.ind <- ifelse(test$Gender=="Male",1,0)

# Try with variable selection
xgb.rfe.control <- rfeControl(functions=caretFuncs, method="cv", number=5,
                              verbose=FALSE)
xgb.rfe <- rfe(x=train[,names(train)%in%c(
  "Age","TB","DB","ALP","ALT","AST","TP","ALB","RAG","gender.ind")],
  y=train$Diagnosis, size=c(1:10),rfeControl=xgb.rfe.control)
opt.vars <- xgb.rfe$optVariables

set.seed(999)

xgb.control <- trainControl(method="cv", number=5, savePredictions=TRUE,
                            verboseIter=FALSE, returnData=FALSE)

# Add gender indicator
train$gender.ind <- ifelse(train$Gender=="Male",1,0)
test$gender.ind <- ifelse(test$Gender=="Male",1,0)

# Transform training and test sets into xgb.DMatrix objects
xgbTrainXRed <- xgb.DMatrix(data=as.matrix(train[,opt.vars]))
xgbTrainY <- train$Diagnosis

xgbParamGrid <- expand.grid(eta=c(0.01,0.1,0.3,0.5),
                         gamma=c(0,0.1,0.2,0.3,0.4),
                         max_depth=2*(1:3)-1,
                         min_child_weight=c(5,15),
                         subsample=c(0.5,0.75,1),
                         colsample_bytree=c(0.5,0.75,1),
                         nrounds=c(100,300,500))

xgbRed <- caret::train(x=xgbTrainXRed,y=xgbTrainY,method="xgbTree",
                       trControl=xgb.control,tuneGrid=xgbParamGrid, nthread=1)
```

```r
# Now, create summary tables; we want out-of-sample accuracy and Kappa values
oos.errors <- c(round(getTrainPerf(xgbTrain)[1,1],3),
                round(getTrainPerf(xgbRed)[1,1],3))
kappa.vals <- c(round(getTrainPerf(xgbTrain)[1,2],3),
                round(getTrainPerf(xgbRed)[1,2],3))

labels <- c("XGBoost with all Predictors", "XGBoost with Variable Selection")
summary.table <- data.frame(labels, oos.errors, kappa.vals)
colnames(summary.table) <- c("Model", "Mean Out of Sample Accuracy", "Mean Kappa Value")
t1 <- tableGrob(summary.table, rows=NULL, theme=ttheme_default(base_size = 10))

# Create accuracy and Kappa plot
df <- data.frame(
  Indices=rep(1:5, 2),
  Accuracy=c(xgbTrain$resample$Accuracy, xgbRed$resample$Accuracy),
  Kappa = c(xgbTrain$resample$Kappa, xgbRed$resample$Kappa),
  Model=c(rep("XGBoost with all Predictors", 5),
          rep("XGBoost with Variable Selection", 5)))

# Plot accuracy and Kappa values over the 25 folds, and display them with the summary table
p1 <- ggplot(data = df, aes(x=Indices, y=Accuracy)) + geom_line(aes(colour=Model)) +
  ggtitle("Out-of-Sample Prediction Accuracy by Model") + theme(legend.position="top") +
  theme(plot.title = element_text(size=10), legend.text=element_text(size=8)) +
  guides(colour=guide_legend(nrow=1))

p2 <- ggplot(data = df, aes(x=Indices, y=Kappa)) + geom_line(aes(colour=Model)) +
  ggtitle("Out-of-Sample Kappa Values by Model") +
  theme(legend.position = "none", plot.title = element_text(size=10))

mylegend <- g_legend(p1)

grid.arrange(arrangeGrob(p1 + theme(legend.position="none"),
                         p2 + theme(legend.position="none"),
                         nrow=1), mylegend, t1,  nrow=3, heights=c(5, 2, 4))

xgb.control <- trainControl(method="none", savePredictions=TRUE,
                            verboseIter=FALSE, returnData=FALSE)

# Add gender indicator
train$gender.ind <- ifelse(train$Gender=="Male",1,0)
test$gender.ind <- ifelse(test$Gender=="Male",1,0)

# Transform training and test sets into xgb.DMatrix objects
xgbTrainX <- xgb.DMatrix(data=as.matrix(train[,names(train)%in%c(
  "Age","TB","DB","ALP","ALT","AST","TP","ALB","RAG","gender.ind")]))
xgbTrainY <- train$Diagnosis

xgbFinalGrid <- expand.grid(eta=0.5,gamma=0.1,max_depth=5,nrounds=100,
                            min_child_weight=15,
                            subsample=0.75,colsample_bytree=0.5)

xgbFinal <- caret::train(x=xgbTrainX,y=xgbTrainY,method="xgbTree",
                         trControl=xgb.control,tuneGrid=xgbFinalGrid, nthread=1)
```

```r
xgbImp <- varImp(xgbFinal, scale=FALSE)

colnames <- c("Age","TB","DB","ALP","ALT","AST","TP","ALB","RAG","gender.ind")
order_colnames <- colnames[c(5,3,0,1,7,4,2,6,8)+1]
plot(xgbImp,ylim=rev(order_colnames),main="Variable Importance of XGBoost Classifier")
```

CONCLUSION

```r
# Re-fit each model on the entire training set
final.logistic <- glm(
  factor(Diagnosis)~Age+factor(Gender)+logTB+logDB+logALP+logALT+logAST+logTP+ALB+RAG+
                   factor(Gender)*logTB + factor(Gender)*logDB +
                   factor(Gender)*logAST + factor(Gender)*ALB +
                   Age*logTB + Age*logDB + Age*logTP + Age*ALB + Age*RAG,
                   data=train, family=binomial())

final.nb <- naive_bayes(
  x = predictors[results$optVariables],
  y = factor(binary.response),
  usekernel=TRUE, adjust=0.9)

# KNN
train3x <- train2[,c("AST", "TB", "DB", "ALP")]
preproc_xvals <- preProcess(train3x)
preproc_train3x <- predict(preproc_xvals, train3x)
preproc_train3x <- as.data.frame(preproc_train3x)

test2x <- test[,c("AST", "TB", "DB", "ALP")]
preproc_test_xvals <- preProcess(test2x)
preproc_test2x <- predict(preproc_test_xvals, test2x)
preproc_test2x <- as.data.frame(preproc_test2x)

knn_selected <- knn(preproc_train3x,preproc_test2x,train2$Diagnosis,k=25)
cm_table_knn <- table(knn_selected,test$Diagnosis)
accuracy_func <- function(x){sum(diag(x)/(sum(rowSums(x))))}
knn_selected_accuracy <- accuracy_func(cm_table_knn)

# Discriminant Analysis
lda.final <- lda(Diagnosis ~ Age+Gender+TB+DB+ALP+ALT+AST+TP+ALB+RAG, data=train)
predict(ldaFit, newdata=test)

# Random Forest
test2 <- test[,c("Age", "Gender", "TB", "DB", "ALT", "ALP",
                          "AST", "ALB", "RAG", "Diagnosis")]
test2$Gender <- as.numeric(test2$Gender)
test2$Diagnosis <- as.factor(test2$Diagnosis)

rf_selected <- randomForest(data=train4, Diagnosis~.,
                       importance=TRUE, ntree=1000, mtry=3,keep.forest=TRUE)

rf_selected_test <- predict(rf_selected, newdata=test2, type="response")
```

```r
rf_selected_test_misclass <- mean(ifelse(rf_selected_test == test2$Diagnosis, yes=0, no=1))
rf_selected_accuracy <- 1-rf_selected_test_misclass

# XGBoost
xgb.control <- trainControl(method="none", savePredictions=TRUE,
                            verboseIter=FALSE, returnData=FALSE)

xgbFinalGrid <- expand.grid(eta=0.5,gamma=0.1,max_depth=5,nrounds=100,
                            min_child_weight=15,
                            subsample=0.75,colsample_bytree=0.5)

xgbFinal <- caret::train(x=xgbTrainX,y=xgbTrainY,method="xgbTree",
                         trControl=xgb.control,
                         tuneGrid=xgbFinalGrid, nthread=1)

xgbPred <- predict(xgbFinal, newdata=xgbTestX)
xgbTest <- mean(ifelse(xgbPred == test$Diagnosis, yes=1, no=0))

# Get the final test error estimates; NOTE these are labelled 'error' but refer to accuracy

# Logistic
logistic.predictions <- ifelse(predict(final.logistic, newdata=test,
                                       type="response") > 0.5, y=1, n=0)
logistic.test.error <- sum(logistic.predictions == test$binary.response)/nrow(test)

# Naive Bayes
nb.test.error <- sum(predict(final.nb, newdata=test) == test$binary.response)/nrow(test)

# DA
#load("lda.test.pred.rda")
lda.temp <- ifelse(test$Diagnosis == "Liver Disease", y="Liver.Disease", n="No.Disease")
lda.test.error <- sum(lda.test.pred == lda.temp)/nrow(test)

# XGBoost
xgb.test.error <- mean(ifelse(xgbPred == test$Diagnosis, yes=1, no=0))

# Random Forest
rf.test.error <- mean(ifelse(rf_selected_test==test$Diagnosis,1,0))

# KNN
knn.test.error <- mean(ifelse(knn_selected==test$Diagnosis,1,0))

# Have test accuracies in one vector
testErrors <- c(nb.test.error,knn.test.error,lda.test.error,
                rf.test.error,xgb.test.error)

labels <- c("Naive Bayes", "KNN", "LDA",
            "Random Forest", "XGBoost")

# Get the validation accuracy and Kappas from before
valid.acc <- c(0.664, 0.713, 0.683, 0.712, 0.727)
kappas <- c(0.326, 0.297, 0.084, 0.269, 0.307)
```

```
test.acc.table <- data.frame(labels, valid.acc, kappas, testErrors)
colnames(test.acc.table) <- c("Model", "Out-of-Sample Accuracy",
                              "Mean Kappa Value", "Test Accuracy")
kable(test.acc.table, digits=3, caption = "Summary of final results")
%>% kable_styling(latex_options="hold_position")

##Confusion Matrices

##KNN
knn.confusion <- confusionMatrix(as.factor(knn_selected),as.factor(test$Diagnosis))

##Random Forest
rf.confusion <- confusionMatrix(as.factor(rf_selected_test),as.factor(test$Diagnosis))

##Gradient Boosting
xgb.confusion <- confusionMatrix(as.factor(xgbPred),as.factor(test$Diagnosis))

p1<-knn.confusion$table
p2<-rf.confusion$table
p3<-xgb.confusion$table

par(mfrow=c(2,2))
fourfoldplot(p1,main="KNN")
fourfoldplot(p2,main="Random Forest")
fourfoldplot(p3, main="XGBoost")
```

APPENDIX A: VARIABLE INTERACTION

```
p1 <- ggplot(train,aes(x=logTB, fill=Gender)) + geom_density(alpha=0.25) + ggtitle("logTB") +
  theme(legend.position="none") + facet_grid(. ~ Diagnosis)
p2 <- ggplot(train,aes(x=logDB, fill=Gender)) + geom_density(alpha=0.25) + ggtitle("logDB") +
  theme(legend.position="none") + facet_grid(. ~ Diagnosis)
p3 <- ggplot(train,aes(x=logALP, fill=Gender)) + geom_density(alpha=0.25) + ggtitle("logALP") +
  theme(legend.position="none") + facet_grid(. ~ Diagnosis)
p4 <- ggplot(train,aes(x=logALT, fill=Gender)) + geom_density(alpha=0.25) + ggtitle("logALT") +
  theme(legend.position="none") + facet_grid(. ~ Diagnosis)
p5 <- ggplot(train,aes(x=logAST, fill=Gender)) + geom_density(alpha=0.25) + ggtitle("logAST") +
  theme(legend.position="none") + facet_grid(. ~ Diagnosis)
p6 <- ggplot(train,aes(x=logTP, fill=Gender)) + geom_density(alpha=0.25) + ggtitle("logTP") +
  theme(legend.position="none") + facet_grid(. ~ Diagnosis)
p7 <- ggplot(train,aes(x=ALB, fill=Gender)) + geom_density(alpha=0.25) + ggtitle("ALB") +
  theme(legend.position="none") + facet_grid(. ~ Diagnosis)
p8 <- ggplot(train,aes(x=RAG, fill=Gender)) + geom_density(alpha=0.25) + ggtitle("RAG") +
  facet_grid(. ~ Diagnosis)
p9 <- ggplot(train,aes(x=Age, fill=Gender)) +
  geom_histogram(alpha=0.65, color="black") + ggtitle("Age") +
  facet_grid(. ~ Diagnosis)

ggarrange(p1, p2, p3, p4, p5, p6, p7, p8, p9, ncol=3, nrow=3,
          common.legend = TRUE, legend="bottom")
```

```r
# Partition age
AgeGrps <- cut_interval(train$Age, n = 3)

# Look at groups
p1 <- ggplot(train,aes(x=logTB, fill=AgeGrps)) + geom_density(alpha=0.25) + ggtitle("logTB") +
  theme(legend.position="none") + facet_grid(. ~ Diagnosis)
p2 <- ggplot(train,aes(x=logDB, fill=AgeGrps)) + geom_density(alpha=0.25) + ggtitle("logDB") +
  theme(legend.position="none") + facet_grid(. ~ Diagnosis)
p3 <- ggplot(train,aes(x=logALP, fill=AgeGrps)) + geom_density(alpha=0.25) + ggtitle("logALP") +
  theme(legend.position="none") + facet_grid(. ~ Diagnosis)
p4 <- ggplot(train,aes(x=logALT, fill=AgeGrps)) + geom_density(alpha=0.25) + ggtitle("logALT") +
  theme(legend.position="none") + facet_grid(. ~ Diagnosis)
p5 <- ggplot(train,aes(x=logAST, fill=AgeGrps)) + geom_density(alpha=0.25) + ggtitle("logAST") +
  theme(legend.position="none") + facet_grid(. ~ Diagnosis)
p6 <- ggplot(train,aes(x=logTP, fill=AgeGrps)) + geom_density(alpha=0.25) + ggtitle("logTP") +
  theme(legend.position="none") + facet_grid(. ~ Diagnosis)
p7 <- ggplot(train,aes(x=ALB, fill=AgeGrps)) + geom_density(alpha=0.25) + ggtitle("ALB") +
  theme(legend.position="none") + facet_grid(. ~ Diagnosis)
p8 <- ggplot(train,aes(x=RAG, fill=AgeGrps)) +
  geom_density(alpha=0.25, color="black") + ggtitle("RAG") +
  facet_grid(. ~ Diagnosis)

ggarrange(p1, p2, p3, p4, p5, p6, p7, p8, ncol=3, nrow=3, common.legend = TRUE, legend="bottom")

# Looking at each pair
gpairs_lower <- function(g){
  g$plots <- g$plots[-(1:g$nrow)]
  g$yAxisLabels <- g$yAxisLabels[-1]
  g$nrow <- g$nrow -1

  g$plots <- g$plots[-(seq(g$ncol, length(g$plots), by = g$ncol))]
  g$xAxisLabels <- g$xAxisLabels[-g$ncol]
  g$ncol <- g$ncol - 1

  g
}

g <- ggpairs(train[,c("logTB", "logDB", "logALP",
                 "logALT", "logAST", "logTP",
                 "ALB", "RAG")], aes(colour = train$Diagnosis, alpha = 0.3),
             lower  = list(continuous = "points"),
             upper  = list(continuous = "blank"),
             diag  = list(continuous = "blankDiag"))
gpairs_lower(g)

cor.mat <- cor(train[,c("logTB", "logDB", "logALP",
                        "logALT", "logAST", "logTP", "ALB", "RAG")],
               use="complete.obs")
cor.mat[lower.tri(cor.mat,diag=TRUE)] <- 0

kable(cor(train[,c("logTB", "logDB", "logALP", "logALT",
                   "logAST", "logTP", "ALB", "RAG")],
          use="complete.obs"), caption="Correlation Matrix", digits=2) %>%
        kable_styling(latex_options="hold_position")
```

APPENDIX B: LOGISTIC REGRESSION

```
# Test the model assumptions by plotting linearity within logit
temp_data <- train[, !colnames(train) %in% c("DB", "TB", "ALP", "ALT", "AST", "TP")]

model <- glm(factor(Diagnosis)~., data=temp_data, family="binomial")
probabilities <- predict(model, type = "response")
probabilities <- predict(model, type = "response")
predicted.classes <- ifelse(probabilities > 0.5, "pos", "neg")


mydata <- temp_data %>%
  dplyr::select_if(is.numeric)
vars <- colnames(mydata)

mydata <- mydata %>%
  mutate(logit = log(probabilities/(1-probabilities))) %>%
  gather(key = "vars", value = "predictor.value", -logit)

ggplot(mydata, aes(logit, predictor.value))+
  geom_point(size = 0.8, alpha = 0.5) +
  geom_smooth(method = "lm", color = adjustcolor("red", 0.5), se=F) +
  theme_bw() +
  facet_wrap(~vars, scales = "free_y") +
  xlab("Log Odds of Probability Estimates") +
  ylab("Covariate")

set.seed(20552745)
logistic.control <- trainControl(method = "repeatedcv", number = 5, repeats=5, savePredictions = T)

# First, fit with vanilla GLM (main effect only)
logistic.vanilla <- train(
  x = predictors,
  y = factor(binary.response),
  method = "glm",
  trControl = logistic.control,
  family=binomial())

# Now consider interaction
logistic.interaction <- train(
  factor(Diagnosis)~Age+factor(Gender)+logTB+logDB+logALP+logALT+logAST+logTP+ALB+RAG+
                   factor(Gender)*logTB + factor(Gender)*logDB + factor(Gender)*logAST +
                   factor(Gender)*ALB +
                  Age*logTB + Age*logDB + Age*logTP + Age*ALB + Age*RAG, data=train,
  method = "glm", trControl = logistic.control, family=binomial())

# Now consider interaction and variable selection
logistic.interaction.varselect <- train(
  factor(Diagnosis)~Age+factor(Gender)+logTB+logDB+logALP+logALT+logAST+logTP+ALB+RAG+
```

```r
                   factor(Gender)*logTB + factor(Gender)*logDB +
                  factor(Gender)*logAST + factor(Gender)*ALB +
                   Age*logTB + Age*logDB + Age*logTP + Age*ALB + Age*RAG, data=train,
  method = "glmStepAIC", trControl = logistic.control, family=binomial(), importance=T)

# Do PCA and interaction
logistic.pca <- train(
  factor(Diagnosis)~Age+factor(Gender)+logTB+logDB+logALP+logALT+logAST+logTP+ALB+RAG+
                   factor(Gender)*logTB + factor(Gender)*logDB + factor(Gender)*logAST +
                   factor(Gender)*ALB +
                   Age*logTB + Age*logDB + Age*logTP + Age*ALB + Age*RAG, data=train,
  method = "glm", trControl = logistic.control, family=binomial(), preProc = c("pca"))

# Now try with non-parametric logistic
logistic.smooth <- train(x = predictors, y = factor(binary.response),
  method = "gamSpline",trControl = logistic.control,
  family=binomial())

# Now, create summary tables; we want out-of-sample accuracy and Kappa values
oos.errors <- c(round(getTrainPerf(logistic.vanilla)[1,1],3),
                round(getTrainPerf(logistic.interaction)[1,1],3),
                round(getTrainPerf(logistic.interaction.varselect)[1,1],3),
                round(getTrainPerf(logistic.pca)[1,1],3),
                round(getTrainPerf(logistic.smooth)[1,1],3))

kappa.vals <- c(round(getTrainPerf(logistic.vanilla)[1,2],3),
                round(getTrainPerf(logistic.interaction)[1,2],3),
                round(getTrainPerf(logistic.interaction.varselect)[1,2],3),
                round(getTrainPerf(logistic.pca)[1,2],3),
                round(getTrainPerf(logistic.smooth)[1,2],3))

labels <- c("Logistic Regression with Main Effects only", "Logistic Regression with Interactions",
            "Logistic Regression with Variable Selection", "Logistic Regression with PCA",
            "Non-Parametric Logistic Regression")
summary.table <- data.frame(labels, oos.errors, kappa.vals)
colnames(summary.table) <- c("Model", "Mean Out of Sample Accuracy", "Mean Kappa Value")
t1 <- tableGrob(summary.table, rows=NULL, theme=ttheme_default(base_size = 10))

# Create accuracy and Kappa plot
df <- data.frame(
  Indices=rep(1:25, 5),
  Accuracy=c(logistic.vanilla$resample$Accuracy, logistic.interaction$resample$Accuracy,
             logistic.interaction.varselect$resample$Accuracy, logistic.pca$resample$Accuracy,
             logistic.smooth$resample$Accuracy),
  Kappa = c(logistic.vanilla$resample$Kappa, logistic.interaction$resample$Kappa,
             logistic.interaction.varselect$resample$Kappa, logistic.pca$resample$Kappa,
             logistic.smooth$resample$Kappa),
  Model=c(rep("Logistic Regression with Main Effects only", 25),
                rep("Logistic Regression with Interactions", 25),
                rep("Logistic Regression with Variable Selection", 25),
                rep("Logistic Regression with PCA", 25),
                rep("Non-Parametric Logistic Regression", 25)))
```

```r
# Plot accuracy and Kappa values over the 25 folds, and display them with the summary table
p1 <- ggplot(data = df, aes(x=Indices, y=Accuracy)) + geom_line(aes(colour=Model)) +
  ggtitle("Out-of-Sample Prediction Accuracy by Model") + theme(legend.position="top") +
  theme(plot.title = element_text(size=10),
        legend.text=element_text(size=8)) + guides(colour=guide_legend(nrow=2))

p2 <- ggplot(data = df, aes(x=Indices, y=Kappa)) + geom_line(aes(colour=Model)) +
  ggtitle("Out-of-Sample Kappa Values by Model") +
  theme(legend.position = "none", plot.title = element_text(size=10))

mylegend<-g_legend(p1)

grid.arrange(arrangeGrob(p1 + theme(legend.position="none"),
                         p2 + theme(legend.position="none"),
                         nrow=1), mylegend, t1,  nrow=3, heights=c(5, 2, 4))

# Extract variable importance, re-scale then sort
logistic.importance <- varImp(logistic.interaction.varselect$finalModel)
logistic.importance <- logistic.importance/max(logistic.importance)*100
logistic.importance <- data.frame(rownames(logistic.importance), logistic.importance$Overall)
colnames(logistic.importance) <- c("Variable", "Importance")

ggplot(data=logistic.importance, aes(x=reorder(Variable, Importance), y=Importance)) +
  geom_bar(stat="identity", fill = adjustcolor("deepskyblue4", 0.8)) + coord_flip() +
  ylab("Relative Variable Importance") + xlab("Covariate")
```