Code

The function below was used to generate artificial sample points using Synthetic Mapping oversampling Procedure(SMOTE) in R. Before using SMOTE, we arranged the dataframe in R by making the first column the response variable and the other columns to be features. The function takes a numerical dataframe, and an integer k denoting the number of nearest neighbors to be used to impute data.

```
syn_data_borderline <- function(dataframe,k){
    library(smotefamily)
    set.seed(1005)
    data <- ADAS(dataframe[,2:ncol(dataframe)],dataframe[,1],K=k)
    newdf <- data.frame("y"=data$data$class)
    newdf <- cbind(newdf,data$data[,1:length(data$data)-1])
    return (newdf)
}
```

The function below was used to do 5-fold cross validation for random forests. The cross validation procedure was used to tune the 'mtry' argument and arrive at the optimal number of predictors to be used in each tree. The function takes a numerical dataframe; and tree size ,which is the number of predictors to be used in each tree for the random forest. The model generates synthetic datapoints using the function above for every sub-training set

```
cross_validation_trees <- function(trainingdata,treesize){
  library(caret)
  library(randomForest)
  library(pROC)
  #Divide the data into 5 folds
  data_folds <- createFolds(trainingdata$y,k=5,list=FALSE)

  #Create different training sets
  data_train_fold1<- trainingdata[data_folds!=5,]
  data_train_fold2<- trainingdata[data_folds!=1,]
  data_train_fold3<- trainingdata[data_folds!=2,]
  data_train_fold4<- trainingdata[data_folds!=3,]
  data_train_fold5<- trainingdata[data_folds!=4,]

  #Create different test sets
  test_data_fold1 <- trainingdata[data_folds==5,]
  test_data_fold2 <- trainingdata[data_folds==1,]
  test_data_fold3 <- trainingdata[data_folds==2,]
  test_data_fold4 <- trainingdata[data_folds==3,]
  test_data_fold5 <- trainingdata[data_folds==4,]

  #Create a vector to store number of correctly classified points at every step.
  result_list <- c(rep(0,5))
  area_under_curve <- c(rep(0,5))
  trainlist <- c(rep(0,5))
  testlist <- c(rep(0,5))
  #Create list to store each training and test fold. The lists will be used inside a for loop to build
  trainlist <- list(data_train_fold1,data_train_fold2,data_train_fold3,data_train_fold4,data_train_fold5
  testlist <- list(test_data_fold1,test_data_fold2,test_data_fold3,test_data_fold4,test_data_fold5)
  for (i in 1:5){
    trainlist[[i]] <- syn_data_borderline(trainlist[[i]],5)
```

```
    randomforest_model <- randomForest(factor(y)~.,data=trainlist[[i]],mtry=treesize,importance=TRUE)
    prediction <- predict(randomforest_model,newdata=testlist[[i]],type='response')
    table <- table(prediction,testlist[[i]]$y)
    result_list[i] <- (table[1]+table[4])
    rf.roc <- roc(testlist[[i]]$y,ifelse(prediction==1,1,0))
    area_under_curve[i] <- rf.roc$auc[1]

  }
  return (list(result_list,randomforest_model,area_under_curve))
}
```

The function below was used to generate principal compnents for the economic features in our dataset. The economic features were correlated to the response, but also highly correlated to one another. We used PCA to generate two derived economic features which are representative of all the economic features in our dataset.

```
Principal_Component <- function(dataframe){
  "Subset of raw data with only the economic features"
  dataframe_features_subset <- data.frame("emp.var.rate"=dataframe$emp.var.rate,"cons.price.idx"=datafra

  "Create Principal Components of the economic features"
  dataframe_subset_princomp <- prcomp(dataframe_features_subset,center=TRUE,scale.=TRUE)

  "Create a new dataframe by removing the original economic features and replacing them with the princi
  dataframe_new <- dataframe[, !(colnames(dataframe) %in% c("emp.var.rate","cons.price.idx","cons.conf.
  dataframe_new$PC1 <- dataframe_subset_princomp$x[,1]
  dataframe_new$PC2 <- dataframe_subset_princomp$x[,2]
  dataframe_new$PC3 <- dataframe_subset_princomp$x[,3]
  dataframe_new$PC4 <- dataframe_subset_princomp$x[,4]
  dataframe_new$PC5 <- dataframe_subset_princomp$x[,5]

  "Return the new dataframe"
  return (dataframe_new)
}
```

The code below is used to run the random forest model. A function called augmented_dataframe is used for this. The function was used to clean the data and generate a numerical dataframe. The code for the function is long anf redundant. The professor asked us to exclude code used for cleaning data. Hence we chose to not include any code used for data cleaning. The code chunk below begins with importing and cleaning the data. Then, we use the cross validation function described above to find the optimal tuning parameter for random forests by doing a grid search. With the optimal tuning parameter, we fit the model on the entire training set and predict on the test set.

```
train <- read.csv("~/Documents/565Project/Data/basic_train.csv")
test <- read.csv("~/Documents/565Project/Data/basic_test.csv")
train <- augmented_dataframe(train)
#Grid search and cv on basic train and test set
#Cross Validation TO IDENTIFY OPTIMAL TREESIZE
treesize = c(4,5,6,7)
validation <- c(rep(0,length(treesize)))
auc_tree <- list(rep(0,5))

for (i in 1:4){
  results <- cross_validation_trees(train,treesize[i])
  validation[i] <- sum(results[[1]])/nrow(train)
  auc_tree[[i]] <- results[[3]]
```

```
}
print (validation)
print (auc_tree)
write.csv(auc_tree,file="basicmodelauc.csv")
write.csv(validation, file = "basicmodelvalidation.csv")

#USING CV RESULTS TO TRAIN MODEL ON ENTIRE TRAINING SET AND PREDICT ON TEST SET

train <- syn_data_borderline(train,5)

test <- augmented_dataframe(test)

modeldf <- rbind(train,test)
#######Fit the Model
model <- randomForest(factor(y)~.,data=modeldf[1:nrow(train),],mtry=5,importance=TRUE)
library(pROC)
prediction_basic <- predict(model,test,type='response')
pred_table_basic <- table(prediction_basic,test$y)
rf.roc_bestmodel_basic <- roc(test$y,ifelse(prediction_basic==1,1,0))
area_under_curve_bestmodel_basic <- rf.roc_bestmodel_basic
save(rf.roc_bestmodel_basic,file="~/Documents/565Project/evaluation/comparison.R")
```

The code below is used to run a random forest model after feature engineering on the training set. We changed the 'p-days' variable by modifying it to be ordinal with 5 different levels. We also replaced the economic features with 2 derived economic variables using PCA. The augmented_dataframe_combodata is a function used to data cleaning. The rest of the process is the same as above.

```
modified_train <- read.csv("~/Documents/565Project/Data/modified_train.csv")
modified_test <- read.csv("~/Documents/565Project/Data/modified_test.csv")
modified_train <- augmented_dataframe_combodata(modified_train)
names(modified_train)[25] <- "default_no"
names(modified_train)[26] <- "default_unknown"
names(modified_train)[27] <- "housing_no"
names(modified_train)[28] <- "housing_unknown"

#USING MODIFIED TRAINING DATA TO PICK OPTIMAL TREE SIZE WITH CROSS VALIDATION
treesize = c(4,5,6,7)
validation_complexmodel <- c(rep(0,length(treesize)))
auc_tree_complexmodel <- list(rep(0,5))

z=0
for (i in 1:4){
  results <- cross_validation_trees(modified_train,treesize[i])
  validation_complexmodel[i] <- sum(results[[1]])/nrow(modified_train)
  auc_tree_complexmodel[[i]] <- results[[3]]

}
print (validation_complexmodel)
print (auc_tree_complexmodel)

write.csv(validation_complexmodel, file = "complexmodelvalidation.csv")

#USING CV TO BUILD A RF MODEL ON ENTIRE TRAINING SET AND PREDICT ON TEST SET
modified_train <- syn_data_borderline(modified_train,5)
```

```
modified_test <- augmented_dataframe_combodata(modified_test)

names(modified_test)[25] <- "default_no"
names(modified_test)[26] <- "default_unknown"
names(modified_test)[27] <- "housing_no"
names(modified_test)[28] <- "housing_unknown"

dataframe <-rbind(modified_train,modified_test)
modified_model <- randomForest(factor(y)~.,data=dataframe[1:55093,],mtry=5,importance=TRUE)
prediction_modified <- predict(modified_model,modified_test,type='response')
pred_table_modified <- table(prediction_modified,modified_test$y)
rf.roc_bestmodel_modified <- roc(modified_test$y,ifelse(prediction_modified==1,1,0))
area_under_curve_bestmodel_modified <- rf.roc_bestmodel_modified
save(rf.roc_bestmodel_modified,file="~/Documents/565Project/evaluation/comparison.R")
```

Logistic Regression with LASSO:

```
library(glmnet)
library(pROC)
bank.train.mod <- read.csv("C:\\Users\\tbian\\Documents\\GitHub\\565project\\data\\modified_train.csv")
bank.test.mod <- read.csv("C:\\Users\\tbian\\Documents\\GitHub\\565project\\data\\modified_test.csv")
bank.train <- read.csv("C:\\Users\\tbian\\Documents\\GitHub\\565project\\data\\basic_train.csv")
bank.test <- read.csv("C:\\Users\\tbian\\Documents\\GitHub\\565project\\data\\basic_test.csv")
```

Step2. do the lASSO logistic model based on the basic training dataset:

```
set.seed(1005)
x.matrix.b<-model.matrix(~.,bank.train[,-20])[,-1]
x.test.b<-model.matrix(~.,bank.test[,-20])[,-1]
y.test.b=bank.test$y
foldid=sample(1:4,size=length(bank.train$y),replace=TRUE)
bank.lasso.b<-cv.glmnet(x.matrix.b,bank.train$y, family="binomial", type.measure="class", alpha=1)
plot(bank.lasso.b)
min(bank.lasso.b$cvm)
```

Step 3. Fit the model with tuning lamda and generate test error and ROC Curve.

```
set.seed(1005)
fit.b<-glmnet(x.matrix.b,bank.train$y, family="binomial", alpha=1,lambda = bank.lasso.b$lambda.1se)
logistic.predict.b<-predict (fit.b, newx = x.test.b , type="response")
log.pre.b<-ifelse(logistic.predict.b<0.5,0,1)
y.test.b<-ifelse(y.test.b=='no',0,1)
table(y.test.b, log.pre.b)
1-mean(y.test.b==log.pre.b)  #####[1] 0.1012044
log.roc.b <- roc(y.test.b, as.numeric(logistic.predict.b))
plot(log.roc.b)
log.roc.b$auc
```

Now, we will do the lasso logistic regression on the modified train dataset and test dataset:

```
set.seed(1005)
x.matrix.m<-model.matrix(~.,bank.train.mod[,-14])[,-1]
x.test.m<-model.matrix(~.,bank.test.mod[,-14])[,-1]
y.test.m=bank.test.mod$y
foldid.m=sample(1:4,size=length(bank.train.mod$y),replace=TRUE)
bank.lasso.m<-cv.glmnet(x.matrix.m,bank.train.mod$y, family="binomial", type.measure="class", alpha=1)
```

```
plot(bank.lasso.m)
min(bank.lasso.m$cvm)
```

Check the test error and generate ROC curve:

```
set.seed(1005)
fit.m<-glmnet(x.matrix.m,bank.train.mod$y, family="binomial", alpha=1,lambda = bank.lasso.m$lambda.1se)
logistic.predict.m<-predict (fit.m, newx = x.test.m , type="response")
log.pre.m<-ifelse(logistic.predict.m<0.5,0,1)
y.test.m<-ifelse(y.test.m=="no",0,1)
table(y.test.m, log.pre.m)
1-mean(y.test.m==log.pre.m)   #####[1] 0.1007187
log.roc.m <- roc(y.test.m, as.numeric(logistic.predict.m))
plot(log.roc.m)
log.roc.m$auc
```