Untitled

2024-09-05

##library loading

library(caret)

## Warning: package 'caret' was built under R version 4.3.3

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 4.3.3

## Loading required package: lattice

library(nnet)

## Warning: package 'nnet' was built under R version 4.3.3

library(pROC)

## Warning: package 'pROC' was built under R version 4.3.3

## Type 'citation("pROC")' for a citation.

##   
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':  
##   
## cov, smooth, var

library(gmodels)

## Warning: package 'gmodels' was built under R version 4.3.3

##   
## Attaching package: 'gmodels'

## The following object is masked from 'package:pROC':  
##   
## ci

library(dplyr)

## Warning: package 'dplyr' was built under R version 4.3.3

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(e1071)

## Warning: package 'e1071' was built under R version 4.3.3

library(xgboost)

## Warning: package 'xgboost' was built under R version 4.3.3

##   
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':  
##   
## slice

library(corrplot)

## Warning: package 'corrplot' was built under R version 4.3.3

## corrplot 0.94 loaded

library(imbalance)

## Warning: package 'imbalance' was built under R version 4.3.3

##importing data into

StrokePred <- read.csv("C:/Users/katta/Downloads/healthcare-dataset-stroke-data cleaned.csv")

##droping ununcessary columns

StrokePred[,c('work\_type','Residence\_type','smoking\_status','id')] <- list(NULL)

##spliting data into train-validation and test

set.seed(1234)  
#spliting 17% into test dataset  
sample <- sample.int(n = nrow(StrokePred), size = nrow(StrokePred)\*0.17, replace = F)  
strokepred\_test <- StrokePred[sample, ] ##Yields held-out test dataset  
strokepred\_trainvalidation <- StrokePred[-sample, ]

imbalanceRatio(strokepred\_trainvalidation, classAttr = 'stroke')

## [1] 0.04301075

positives <- mwmote(strokepred\_trainvalidation, classAttr = 'stroke', numInstances = 1500)  
strokepred\_balanced <- rbind(strokepred\_trainvalidation, positives)  
table(strokepred\_balanced$stroke)

##   
## 0 1   
## 3906 1668

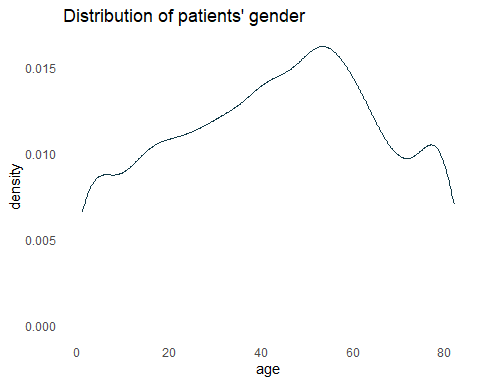
imbalanceRatio(strokepred\_balanced, classAttr = 'stroke')

## [1] 0.4270353

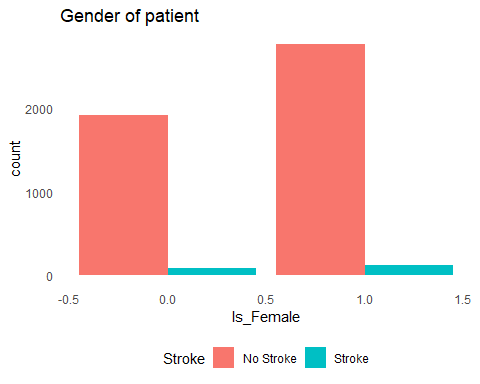
# Distribution of age by gender

ggplot(StrokePred, aes(x = age, fill = Is\_Female)) +  
 geom\_density(alpha = 0.5, color = "#103846") +  
 labs(title = "Distribution of patients' gender",  
 color = "#1D4B5B") +  
 theme\_minimal() +  
 theme(panel.grid = element\_blank()) # Remove gridlines

## Warning: The following aesthetics were dropped during statistical transformation: fill.  
## ℹ This can happen when ggplot fails to infer the correct grouping structure in  
## the data.  
## ℹ Did you forget to specify a `group` aesthetic or to convert a numerical  
## variable into a factor?



ggplot(data = StrokePred, aes(x = Is\_Female, fill = factor(stroke))) +  
 geom\_bar(position = "dodge") + # Stacked bar based on Stroke variable  
 labs(title = "Gender of patient", x = "Is\_Female", color = "#1D4B5B") +  
 scale\_fill\_discrete(name = "Stroke", labels = c("No Stroke", "Stroke")) + # Legend labels  
 theme\_minimal() +  
 theme(panel.grid = element\_blank(), # Remove gridlines  
 legend.position = "bottom")

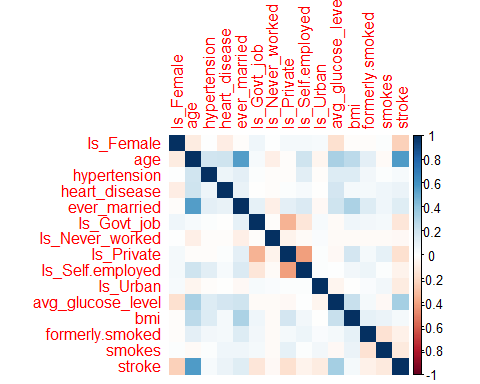


##correlation chart

install.packages("corrplot")

## Warning: package 'corrplot' is in use and will not be installed

corr\_data<-cor(strokepred\_balanced)  
corrplot(corr\_data,method = 'color')



##spliting data into train and validation

set.seed(1234)  
#spliting 17% into validation dataset  
sample <- sample.int(n = nrow(strokepred\_balanced), size = nrow(strokepred\_balanced)\*0.17, replace = F)  
strokepred\_validation <- strokepred\_balanced[sample, ] ##Yields validation dataset  
strokepred\_train <- strokepred\_balanced[-sample, ]

#checking number of row for each dataset

nrow(strokepred\_train)

## [1] 4627

nrow(strokepred\_validation)

## [1] 947

nrow(strokepred\_test)

## [1] 834

strokepred\_train\_scaled <- scale(strokepred\_train[,-ncol(strokepred\_train)])  
strokepred\_validation\_scaled <- scale(strokepred\_validation[,-ncol(strokepred\_validation)])  
strokepred\_test\_scaled <- scale(strokepred\_test[,-ncol(strokepred\_test)])

stroke\_column <- strokepred\_train$stroke  
strokepred\_train\_scaled\_xg <- cbind(strokepred\_train\_scaled, stroke\_column)  
stroke\_column <- strokepred\_validation$stroke  
strokepred\_validation\_scaled\_xg <- cbind(strokepred\_validation\_scaled, stroke\_column)  
stroke\_column <- strokepred\_test$stroke  
strokepred\_test\_scaled\_xg <- cbind(strokepred\_test\_scaled, stroke\_column)

##feature selection

control <- rfeControl(functions = lmFuncs, # linear regression  
 method = "repeatedcv", # repeated cv  
 repeats = 5, # number of repeats  
 number = 20) # number of folds

## Run recursive feature elimination (RFE)

result\_rfe1 <- rfe(x = strokepred\_train\_scaled,   
 y = strokepred\_train$stroke,   
 sizes = c(1:14),  
 rfeControl = control)  
  
  
result\_rfe1

##   
## Recursive feature selection  
##   
## Outer resampling method: Cross-Validated (20 fold, repeated 5 times)   
##   
## Resampling performance over subset size:  
##   
## Variables RMSE Rsquared MAE RMSESD RsquaredSD MAESD Selected  
## 1 0.3810 0.3077 0.3321 0.009503 0.04056 0.01066   
## 2 0.3742 0.3323 0.3163 0.010025 0.04435 0.01117   
## 3 0.3519 0.4086 0.2861 0.012328 0.04619 0.01174   
## 4 0.3183 0.5155 0.2466 0.013411 0.04020 0.01182   
## 5 0.3140 0.5284 0.2437 0.014050 0.04060 0.01198   
## 6 0.3111 0.5370 0.2399 0.014773 0.04178 0.01206   
## 7 0.3085 0.5448 0.2390 0.014570 0.04122 0.01186   
## 8 0.3068 0.5498 0.2374 0.014419 0.04090 0.01180   
## 9 0.3059 0.5524 0.2368 0.014891 0.04167 0.01201   
## 10 0.3059 0.5524 0.2364 0.014862 0.04195 0.01198   
## 11 0.3048 0.5556 0.2356 0.014971 0.04215 0.01208   
## 12 0.3045 0.5564 0.2353 0.014958 0.04224 0.01211   
## 13 0.3041 0.5577 0.2347 0.015046 0.04246 0.01219 \*  
## 14 0.3041 0.5577 0.2347 0.015040 0.04247 0.01218   
##   
## The top 5 variables (out of 13):  
## age, Is\_Private, Is\_Self.employed, Is\_Govt\_job, avg\_glucose\_level

# Print the selected features  
predictors(result\_rfe1)

## [1] "age" "Is\_Private" "Is\_Self.employed"   
## [4] "Is\_Govt\_job" "avg\_glucose\_level" "ever\_married"   
## [7] "Is\_Female" "formerly.smoked" "heart\_disease"   
## [10] "hypertension" "smokes" "Is\_Urban"   
## [13] "Is\_Never\_worked"

##Running Logestic Regression Model

logistic\_regression\_model <- glm(stroke ~., data=strokepred\_train, family="binomial") ##Or, can use all predictors except one using the ~ . -EXCLUDEDVARIABLE notation  
summary(logistic\_regression\_model) ##Outputs summary of model & coefficients

##   
## Call:  
## glm(formula = stroke ~ ., family = "binomial", data = strokepred\_train)  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -4.666e+00 3.159e-01 -14.770 < 2e-16 \*\*\*  
## Is\_Female -1.017e+00 1.091e-01 -9.321 < 2e-16 \*\*\*  
## age 1.256e-01 4.431e-03 28.336 < 2e-16 \*\*\*  
## hypertension -4.042e-01 1.564e-01 -2.585 0.00975 \*\*   
## heart\_disease -8.483e-01 1.822e-01 -4.656 3.22e-06 \*\*\*  
## ever\_married -7.544e-01 1.602e-01 -4.709 2.49e-06 \*\*\*  
## Is\_Govt\_job -4.612e+00 2.860e-01 -16.128 < 2e-16 \*\*\*  
## Is\_Never\_worked -1.343e+01 3.412e+02 -0.039 0.96859   
## Is\_Private -3.879e+00 2.368e-01 -16.383 < 2e-16 \*\*\*  
## Is\_Self.employed -4.909e+00 2.702e-01 -18.170 < 2e-16 \*\*\*  
## Is\_Urban -3.005e-01 1.063e-01 -2.826 0.00472 \*\*   
## avg\_glucose\_level 8.500e-03 9.776e-04 8.694 < 2e-16 \*\*\*  
## bmi 9.072e-03 8.939e-03 1.015 0.31013   
## formerly.smoked -8.600e-01 1.453e-01 -5.920 3.23e-09 \*\*\*  
## smokes -5.483e-01 1.788e-01 -3.066 0.00217 \*\*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 5634.0 on 4626 degrees of freedom  
## Residual deviance: 2372.1 on 4612 degrees of freedom  
## AIC: 2402.1  
##   
## Number of Fisher Scoring iterations: 14

#predicting for training data set  
TRAINING\_PREDICTIONS <- predict(logistic\_regression\_model, newdata=strokepred\_train,type="response")  
strokepred\_train$LOGIT\_PRED = TRAINING\_PREDICTIONS  
  
#predicting for validation data set  
VALIDATION\_PREDICTIONS <- predict(logistic\_regression\_model, newdata=strokepred\_validation,type="response")  
strokepred\_validation$LOGIT\_PRED = VALIDATION\_PREDICTIONS  
  
#predicting for testing data set  
TEST\_PREDICTIONS <- predict(logistic\_regression\_model, newdata=strokepred\_test,type="response")  
strokepred\_test$LOGIT\_PRED = TEST\_PREDICTIONS

##8. Evaluate validation & test predictions  
postResample(pred = VALIDATION\_PREDICTIONS, obs =  
strokepred\_validation$stroke)

## RMSE Rsquared MAE   
## 0.2578320 0.6904355 0.1386886

postResample(pred = TEST\_PREDICTIONS, obs = strokepred\_test$stroke)

## RMSE Rsquared MAE   
## 0.27168688 0.05980218 0.14542971

#training  
myroc <- roc(strokepred\_train$stroke, strokepred\_train$LOGIT\_PRED)

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

auc(myroc) ##Print out AUC of training

## Area under the curve: 0.947

#validation  
myroc <- roc(strokepred\_validation$stroke, strokepred\_validation$LOGIT\_PRED)

## Setting levels: control = 0, case = 1  
## Setting direction: controls < cases

auc(myroc) ##Print out AUC of validation

## Area under the curve: 0.9633

#test  
myroc <- roc(strokepred\_test$stroke, strokepred\_test$LOGIT\_PRED)

## Setting levels: control = 0, case = 1  
## Setting direction: controls < cases

auc(myroc) ##Print out AUC of test

## Area under the curve: 0.7816

strokepred\_validation <- strokepred\_validation %>% mutate(LOGIT\_CLASSIFICATION = 1\*(LOGIT\_PRED >= 0.5))  
  
strokepred\_test <- strokepred\_test %>% mutate(LOGIT\_CLASSIFICATION = 1\*(LOGIT\_PRED >= 0.5))

validation\_performance <- confusionMatrix(data=as.factor(strokepred\_validation$LOGIT\_CLASSIFICATION), reference = as.factor(strokepred\_validation$stroke),positive="1") ##Generate confusion matrix (based on probability cutoff)  
validation\_performance

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 630 58  
## 1 26 233  
##   
## Accuracy : 0.9113   
## 95% CI : (0.8914, 0.9286)  
## No Information Rate : 0.6927   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.7851   
##   
## Mcnemar's Test P-Value : 0.0007186   
##   
## Sensitivity : 0.8007   
## Specificity : 0.9604   
## Pos Pred Value : 0.8996   
## Neg Pred Value : 0.9157   
## Prevalence : 0.3073   
## Detection Rate : 0.2460   
## Detection Prevalence : 0.2735   
## Balanced Accuracy : 0.8805   
##   
## 'Positive' Class : 1   
##

test\_performance <- confusionMatrix(data=as.factor(strokepred\_test$LOGIT\_CLASSIFICATION), reference = as.factor(strokepred\_test$stroke),positive="1") ##Generate confusion matrix (based on probability cutoff)  
test\_performance

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 737 25  
## 1 56 16  
##   
## Accuracy : 0.9029   
## 95% CI : (0.8807, 0.9221)  
## No Information Rate : 0.9508   
## P-Value [Acc > NIR] : 1.0000000   
##   
## Kappa : 0.2353   
##   
## Mcnemar's Test P-Value : 0.0008581   
##   
## Sensitivity : 0.39024   
## Specificity : 0.92938   
## Pos Pred Value : 0.22222   
## Neg Pred Value : 0.96719   
## Prevalence : 0.04916   
## Detection Rate : 0.01918   
## Detection Prevalence : 0.08633   
## Balanced Accuracy : 0.65981   
##   
## 'Positive' Class : 1   
##

##Running a SVM model

model <- svm(strokepred\_train\_scaled\_xg[, -ncol(strokepred\_train\_scaled\_xg)],   
 kernel = "radial", cost = 1000, gamma = 0.1)

# Predict on validation data  
validation\_pred <- predict(model, strokepred\_validation\_scaled\_xg[, -ncol(strokepred\_validation\_scaled\_xg)])  
  
  
  
# Predict on test data  
test\_pred <- predict(model, strokepred\_test\_scaled\_xg[, -ncol(strokepred\_test\_scaled\_xg)])

validation\_pred\_numeric <- ifelse(validation\_pred == TRUE, 1, 0)  
test\_pred\_numeric <- ifelse(test\_pred == TRUE, 1, 0)

# Validation Confusion Matrix  
  
  
strokepred\_validation\_scaled$preds = validation\_pred

## Warning in strokepred\_validation\_scaled$preds = validation\_pred: Coercing LHS  
## to a list

strokepred\_validation\_scaled$preds <- as.integer(as.logical(strokepred\_validation\_scaled$preds))  
# Generate confusion matrix  
  
validation\_pred\_factor <- factor(strokepred\_validation\_scaled$preds, levels = c(0, 1))  
  
# Assuming true labels are numeric (0/1)  
strokepred\_validation$stroke\_factor <- factor(strokepred\_validation$stroke, levels = c(0, 1))  
  
# Generate confusion matrix  
validation\_performance <- confusionMatrix(data = validation\_pred\_factor, reference = strokepred\_validation$stroke\_factor, positive = "1")  
print(validation\_performance)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 331 138  
## 1 325 153  
##   
## Accuracy : 0.5111   
## 95% CI : (0.4787, 0.5434)  
## No Information Rate : 0.6927   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.0257   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.5258   
## Specificity : 0.5046   
## Pos Pred Value : 0.3201   
## Neg Pred Value : 0.7058   
## Prevalence : 0.3073   
## Detection Rate : 0.1616   
## Detection Prevalence : 0.5048   
## Balanced Accuracy : 0.5152   
##   
## 'Positive' Class : 1   
##

# Test Confusion Matrix  
  
  
strokepred\_test\_scaled$preds = test\_pred

## Warning in strokepred\_test\_scaled$preds = test\_pred: Coercing LHS to a list

strokepred\_test\_scaled$preds <- as.integer(as.logical(strokepred\_test\_scaled$preds))  
# Generate confusion matrix  
  
test\_pred\_factor <- factor(strokepred\_test\_scaled$preds, levels = c(0, 1))  
  
# Assuming true labels are numeric (0/1)  
strokepred\_test$stroke\_factor <- factor(strokepred\_test$stroke, levels = c(0, 1))  
  
# Generate confusion matrix  
test\_performance <- confusionMatrix(data = test\_pred\_factor, reference = strokepred\_test$stroke\_factor, positive = "1")  
print(test\_performance)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 346 27  
## 1 447 14  
##   
## Accuracy : 0.4317   
## 95% CI : (0.3977, 0.4661)  
## No Information Rate : 0.9508   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : -0.0379   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.34146   
## Specificity : 0.43632   
## Pos Pred Value : 0.03037   
## Neg Pred Value : 0.92761   
## Prevalence : 0.04916   
## Detection Rate : 0.01679   
## Detection Prevalence : 0.55276   
## Balanced Accuracy : 0.38889   
##   
## 'Positive' Class : 1   
##

##Running XGBoost Model

# Create DMatrix objects for XGBoost  
  
dtrain <- xgb.DMatrix(data = as.matrix(strokepred\_train\_scaled\_xg[, -ncol(strokepred\_train\_scaled\_xg)]), label = strokepred\_train\_scaled\_xg[, ncol(strokepred\_train\_scaled\_xg)])  
dvalidation <- xgb.DMatrix(data = as.matrix(strokepred\_validation\_scaled\_xg[, -ncol(strokepred\_validation\_scaled\_xg)]), label = strokepred\_validation\_scaled\_xg[, ncol(strokepred\_validation\_scaled\_xg)])  
dtest <- xgb.DMatrix(data = as.matrix(strokepred\_test\_scaled\_xg[, -ncol(strokepred\_test\_scaled\_xg)]), label = strokepred\_test\_scaled\_xg[, ncol(strokepred\_test\_scaled\_xg)])

params <- list(  
 booster = "gbtree",  
 objective = "binary:logistic", # For classification  
 eval\_metric = "error", # Evaluation metric  
 max\_depth = 3,  
 eta = 0.1,  
 nrounds = 100  
)  
  
model <- xgboost(params = params, data = dtrain, nrounds = params$nrounds)

## [00:31:59] WARNING: src/learner.cc:767:   
## Parameters: { "nrounds" } are not used.  
##   
## [1] train-error:0.164469   
## [2] train-error:0.165550   
## [3] train-error:0.152799   
## [4] train-error:0.146099   
## [5] train-error:0.148260   
## [6] train-error:0.143289   
## [7] train-error:0.143506   
## [8] train-error:0.135725   
## [9] train-error:0.135293   
## [10] train-error:0.129025   
## [11] train-error:0.131619   
## [12] train-error:0.137238   
## [13] train-error:0.130106   
## [14] train-error:0.129890   
## [15] train-error:0.126864   
## [16] train-error:0.124487   
## [17] train-error:0.121893   
## [18] train-error:0.121245   
## [19] train-error:0.120164   
## [20] train-error:0.121029   
## [21] train-error:0.119516   
## [22] train-error:0.116706   
## [23] train-error:0.119732   
## [24] train-error:0.115626   
## [25] train-error:0.117355   
## [26] train-error:0.115626   
## [27] train-error:0.114113   
## [28] train-error:0.114113   
## [29] train-error:0.113897   
## [30] train-error:0.110871   
## [31] train-error:0.112168   
## [32] train-error:0.110439   
## [33] train-error:0.105900   
## [34] train-error:0.105900   
## [35] train-error:0.102658   
## [36] train-error:0.103523   
## [37] train-error:0.102658   
## [38] train-error:0.102226   
## [39] train-error:0.101578   
## [40] train-error:0.101145   
## [41] train-error:0.100281   
## [42] train-error:0.100281   
## [43] train-error:0.100497   
## [44] train-error:0.100065   
## [45] train-error:0.097687   
## [46] train-error:0.098336   
## [47] train-error:0.097471   
## [48] train-error:0.097904   
## [49] train-error:0.098552   
## [50] train-error:0.098984   
## [51] train-error:0.098768   
## [52] train-error:0.097471   
## [53] train-error:0.097471   
## [54] train-error:0.095742   
## [55] train-error:0.095742   
## [56] train-error:0.097039   
## [57] train-error:0.096607   
## [58] train-error:0.095959   
## [59] train-error:0.096175   
## [60] train-error:0.095094   
## [61] train-error:0.095310   
## [62] train-error:0.094878   
## [63] train-error:0.094662   
## [64] train-error:0.093797   
## [65] train-error:0.093797   
## [66] train-error:0.092284   
## [67] train-error:0.092284   
## [68] train-error:0.093149   
## [69] train-error:0.092284   
## [70] train-error:0.092284   
## [71] train-error:0.092068   
## [72] train-error:0.092068   
## [73] train-error:0.092068   
## [74] train-error:0.090123   
## [75] train-error:0.090772   
## [76] train-error:0.090555   
## [77] train-error:0.090123   
## [78] train-error:0.090339   
## [79] train-error:0.090555   
## [80] train-error:0.090339   
## [81] train-error:0.089907   
## [82] train-error:0.089259   
## [83] train-error:0.089691   
## [84] train-error:0.089259   
## [85] train-error:0.088178   
## [86] train-error:0.087746   
## [87] train-error:0.087746   
## [88] train-error:0.086017   
## [89] train-error:0.087962   
## [90] train-error:0.084936   
## [91] train-error:0.084288   
## [92] train-error:0.082991   
## [93] train-error:0.082991   
## [94] train-error:0.082559   
## [95] train-error:0.082559   
## [96] train-error:0.083207   
## [97] train-error:0.082775   
## [98] train-error:0.081046   
## [99] train-error:0.080614   
## [100] train-error:0.080398

# Make predictions on validation and test data  
validation\_pred <- predict(model, dvalidation)  
test\_pred <- predict(model, dtest)

thresholds <- seq(0.1, 0.9, by = 0.1)  
best\_threshold <- NULL  
best\_f1\_score <- 0  
  
for (threshold in thresholds) {  
 validation\_pred\_numeric <- ifelse(validation\_pred > threshold, 1, 0)  
 validation\_cm <- table(actual = strokepred\_validation\_scaled\_xg[, ncol(strokepred\_validation\_scaled\_xg)], predicted = validation\_pred\_numeric)  
 validation\_precision <- sum(validation\_cm[1, 1]) / sum(validation\_cm[, 1])  
 validation\_recall <- sum(validation\_cm[2, 2]) / sum(validation\_cm[2, ])  
 validation\_f1\_score <- 2 \* validation\_precision \* validation\_recall / (validation\_precision + validation\_recall)  
  
 if (validation\_f1\_score > best\_f1\_score) {  
 best\_threshold <- threshold  
 best\_f1\_score <- validation\_f1\_score  
 }  
}  
  
cat("Best threshold:", best\_threshold, "\n")

## Best threshold: 0.1

# ... (rest of your code)  
  
# Apply the optimal threshold to the test predictions  
test\_pred\_numeric <- ifelse(test\_pred > 0.1, 1, 0)  
  
# Calculate the confusion matrix and other metrics  
test\_cm <- table(actual = strokepred\_test\_scaled\_xg[, ncol(strokepred\_test\_scaled\_xg)], predicted = test\_pred\_numeric)  
test\_accuracy <- sum(diag(test\_cm)) / sum(test\_cm)  
test\_recall <- sum(test\_cm[2, 2]) / sum(test\_cm[2, ])  
test\_precision <- sum(test\_cm[1, 1]) / sum(test\_cm[, 1])  
test\_f1\_score <- 2 \* test\_precision \* test\_recall / (test\_precision + test\_recall)  
  
# Print results  
cat("Test accuracy:", test\_accuracy, "\n")

## Test accuracy: 0.5539568

cat("Test recall:", test\_recall, "\n")

## Test recall: 0.8536585

cat("Test precision:", test\_precision, "\n")

## Test precision: 0.9861432

cat("Test F1-score:", test\_f1\_score, "\n")

## Test F1-score: 0.9151307