project2

Weidai He

2025-08-04

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.2     v tibble    3.3.0
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.1.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```r
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
library(ranger)
library(xgboost)
```

```
##
## Attaching package: 'xgboost'
##
```

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

```r
library(MASS)
```

```
##
## Attaching package: 'MASS'
##
## The following object is masked from 'package:dplyr':
##
##     select
```

```r
library(readr)
mergedData <- read_csv("C:/Users/David/Desktop/Traffic Crashes Project/Traffic_Crashes_-_Crashes_202507
    col_types = cols(CRASH_DATE = col_datetime(format = "%m/%d/%Y %I:%M:%S %p")))
```

```r
#mergedData <- data
```

```r
# geo types
northDistricts = c(16, 17, 31, 24, 20, 19)
westDistricts = c(25, 14, 15, 11, 12, 10)
centralDistricts = c(1,18)
southDistricts = c(9, 8, 7, 2, 3, 6, 22, 5, 4)

# traffic way type
roadIntersection = c("T-INTERSECTION", "UNKNOWN INTERSECTION TYPE",
                     "Y-INTERSECTION", "ROUNDABOUT", "L-INTERSECTION")
roadNotRoad = c("PARKING LOT", "ALLEY", "DRIVEWAY")
roadDivided = c("DIVIDED - W/MEDIAN (NOT RAISED)", "DIVIDED - W/MEDIAN BARRIER")

# first contact in crash types
crashRearCar = c("REAR END", "REAR TO FRONT", "REAR TO REAR", "REAR TO SIDE")
crashMiscCar = c("ANGLE", "HEAD ON", "OVERTURNED", "SIDESWIPE OPPOSITE DIRECTION",
                 "TURNING", "SIDESWIPE SAME DIRECTION")
crashNonCar = c("ANIMAL", "FIXED OBJECT", "OTHER OBJECT", "OTHER NONCOLLISION",
                "PARKED MOTOR VEHICLE", "PEDALCYCLIST", "PEDESTRIAN", "TRAIN")

# crash fault types
faultDriverBehavior = c("FOLLOWING TOO CLOSELY", "FAILING TO YIELD RIGHT-OF-WAY",
                        "IMPROPER TURNING/NO SIGNAL",
                        "IMPROPER BACKING", "IMPROPER LANE USAGE",
                        "DRIVING SKILLS/KNOWLEDGE/EXPERIENCE",
                        "OPERATING VEHICLE IN ERRATIC, RECKLESS,
                         CARELESS, NEGLIGENT OR AGGRESSIVE MANNER",
                        "EXCEEDING AUTHORIZED SPEED LIMIT",
                        "EXCEEDING SAFE SPEED FOR CONDITIONS",
                        "DRIVING ON WRONG SIDE/WRONG WAY")

faultDistraction = c("DISTRACTION - FROM INSIDE VEHICLE", "DISTRACTION - FROM OUTSIDE VEHICLE",
                     "TEXTING", "CELL PHONE USE OTHER THAN TEXTING",
                     "DISTRACTION - OTHER ELECTRONIC DEVICE (NAVIGATION DEVICE,
```

```
                      DVD PLAYER, ETC.)")

faultImpairment = c("UNDER THE INFLUENCE OF ALCOHOL/DRUGS (USE WHEN ARREST IS EFFECTED)",
                    "HAD BEEN DRINKING (USE WHEN ARREST IS NOT MADE)",
                    "PHYSICAL CONDITION OF DRIVER")

faultEnviroment = c("WEATHER", "ROAD ENGINEERING/SURFACE/MARKING DEFECTS",
                    "ROAD CONSTRUCTION/MAINTENANCE", "VISION OBSCURED (SIGNS,
                     TREE LIMBS, BUILDINGS, ETC.)",
                    "OBSTRUCTED CROSSWALKS")

faultTrafficViolation = c("DISREGARDING TRAFFIC SIGNALS", "DISREGARDING STOP SIGN",
                          "DISREGARDING YIELD SIGN", "DISREGARDING OTHER TRAFFIC SIGNS",
                          "DISREGARDING ROAD MARKINGS", "TURNING RIGHT ON RED",
                          "PASSING STOPPED SCHOOL BUS")

faultOther = c("EQUIPMENT - VEHICLE CONDITION", "EVASIVE ACTION DUE TO ANIMAL,
                OBJECT, NONMOTORIST",
               "RELATED TO BUS STOP", "ANIMAL",
               "MOTORCYCLE ADVANCING LEGALLY ON RED LIGHT",
               "BICYCLE ADVANCING LEGALLY ON RED LIGHT")
```

Combine

```
# Check dplyr version
if (packageVersion("dplyr") < "1.0.0") {
  stop("Please update dplyr to version 1.0.0 or later: install.packages('dplyr')")
}

# Ensure dplyr::select is used to avoid conflicts
select <- dplyr::select

# Check for required columns
required_cols <- c("INJURIES_TOTAL", "POSTED_SPEED_LIMIT", "TRAFFIC_CONTROL_DEVICE",
                   "WEATHER_CONDITION", "LIGHTING_CONDITION", "ROADWAY_SURFACE_COND", "DAMAGE")
missing_cols <- setdiff(required_cols, names(mergedData))
if (length(missing_cols) > 0) {
  stop(paste("Missing required columns:", paste(missing_cols, collapse = ", ")))
}

# Create binary outcome variable and predictors as described in the report
dataBinary <- mergedData %>%
  mutate(injuryReported = case_when(
    INJURIES_TOTAL == 0 ~ 0,
    is.na(INJURIES_TOTAL) ~ NA_real_,
    TRUE ~ 1
  )) %>%
  mutate(
    postedSpeedLimit = POSTED_SPEED_LIMIT,
    trafficControlPresent = case_when(
      TRAFFIC_CONTROL_DEVICE == "NO CONTROLS" ~ 0,
      TRAFFIC_CONTROL_DEVICE == "UNKNOWN" ~ NA_real_,
```

```r
      TRUE ~ 1
    ),
    weatherClear = case_when(
      WEATHER_CONDITION == "CLEAR" ~ 1,
      WEATHER_CONDITION == "UNKNOWN" ~ NA_real_,
      TRUE ~ 0
    ),
    isDaylight = case_when(
      LIGHTING_CONDITION == "DAYLIGHT" ~ 1,
      LIGHTING_CONDITION == "UNKNOWN" ~ NA_real_,
      TRUE ~ 0
    ),
    roadSurface = case_when(
      ROADWAY_SURFACE_COND == "NO DEFECTS" ~ 0,
      ROADWAY_SURFACE_COND == "UNKNOWN" ~ NA_real_,
      TRUE ~ 1
    ),
    damageOver1500 = case_when(
      DAMAGE == "OVER $1,500" ~ 1,
      is.na(DAMAGE) ~ NA_real_,
      TRUE ~ 0
    )
  )

# Remove rows with NA in injuryReported
dataBinary <- dataBinary %>% filter(!is.na(injuryReported))

# Split data into training and testing sets (80-20 split)
set.seed(432)
trainIndex <- createDataPartition(dataBinary$injuryReported, p = 0.8, list = FALSE)
training_factor <- dataBinary[trainIndex, ]
testing_factor <- dataBinary[-trainIndex, ]

# Define predictors
predictors <- c("postedSpeedLimit", "trafficControlPresent", "weatherClear",
                "isDaylight", "roadSurface", "damageOver1500")

# Subset data to include only selected predictors and outcome using base R
train_subset <- training_factor[, c("injuryReported", predictors)]
test_subset <- testing_factor[, c("injuryReported", predictors)]

# Remove near-zero variance predictors using caret
nzv <- nearZeroVar(train_subset, saveMetrics = TRUE)
train_clean <- train_subset[, !nzv$nzv]
test_clean <- test_subset[, names(train_clean)]

# Remove rows with NA in predictors
train_clean <- train_clean %>% na.omit()
test_clean <- test_clean %>% na.omit()

# Ensure injuryReported is a factor
train_clean$injuryReported <- as.factor(train_clean$injuryReported)
test_clean$injuryReported <- as.factor(test_clean$injuryReported)
```

```r
# Assign class weights (1.5 for positive class)
weights <- ifelse(train_clean$injuryReported == 1, 1.5, 1)

# Fit logistic regression model
logit_model <- glm(injuryReported ~ ., data = train_clean,
                   family = "binomial", weights = weights)
```

```
## Warning in eval(family$initialize): non-integer #successes in a binomial glm!
```

```r
# Predict probabilities on test set
logit_probs <- predict(logit_model, newdata = test_clean, type = "response")

# Optimize threshold using ROC curve
roc_obj <- roc(as.numeric(as.character(test_clean$injuryReported)), logit_probs)
```

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

```
## Setting direction: controls < cases
```

```r
best_coords <- coords(roc_obj, "best", ret = "threshold", transpose = FALSE)
opt_thresh <- best_coords$threshold

# Make predictions using optimal threshold
logit_preds_opt <- ifelse(logit_probs > opt_thresh, 1, 0)

# Evaluate model performance
conf_logit_opt <- confusionMatrix(as.factor(logit_preds_opt),
                                  test_clean$injuryReported, positive = "1")
print(conf_logit_opt)
```

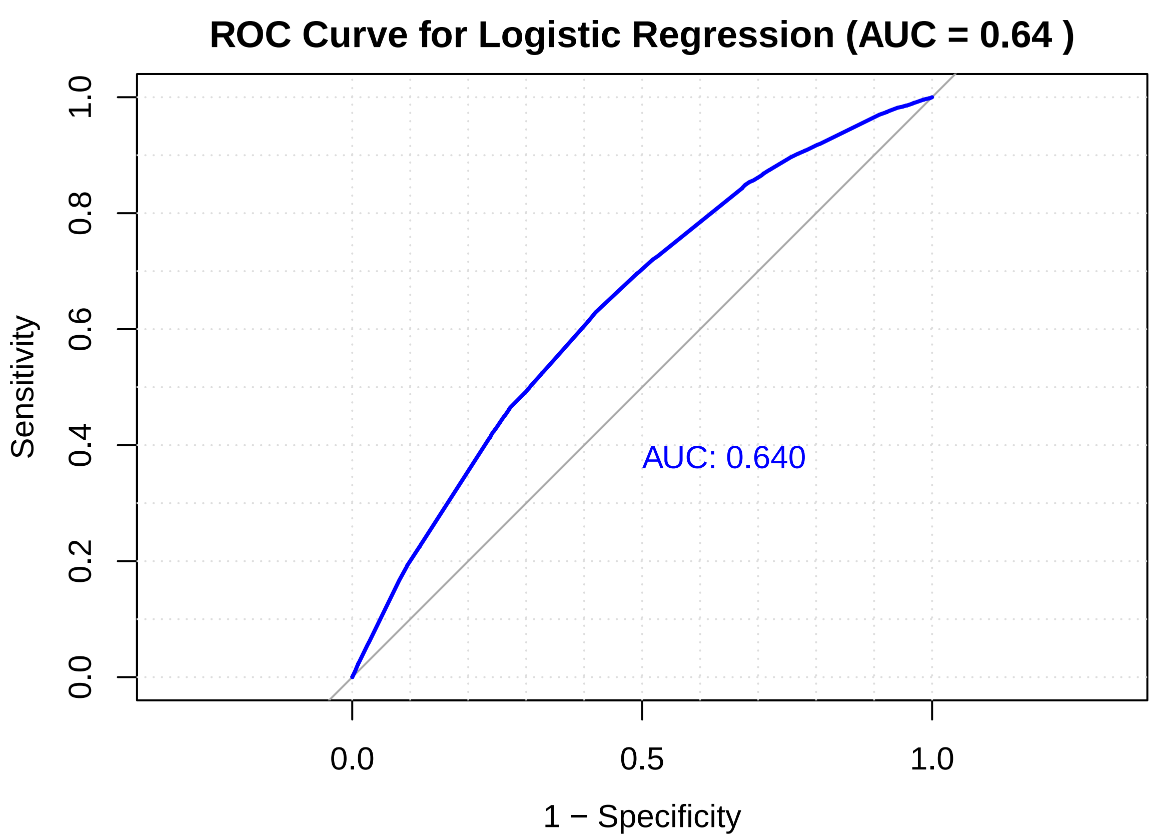
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 85847  9660
##          1 62202 16413
##
##                Accuracy : 0.5873
##                  95% CI : (0.585, 0.5896)
##     No Information Rate : 0.8503
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1144
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.62950
##             Specificity : 0.57986
##          Pos Pred Value : 0.20878
##          Neg Pred Value : 0.89886
##              Prevalence : 0.14974
```

```
##          Detection Rate : 0.09426
##    Detection Prevalence : 0.45149
##       Balanced Accuracy : 0.60468
##
##        'Positive' Class : 1
##
```

```r
# Calculate AUC
auc_val <- auc(roc_obj)
cat("AUC:", auc_val, "\n")
```

```
## AUC: 0.6402528
```

```r
plot.roc(roc_obj,
         main = paste("ROC Curve for Logistic Regression (AUC =", round(auc_val, 3), ")"),
         col = "blue",
         print.auc = TRUE,
         print.auc.y = 0.4,
         legacy.axes = TRUE, # 1-specificity on x-axis
         grid = TRUE)
```



KNN

```r
library(tidyverse)
library(caret)
library(FNN)
library(pROC)
```

```r
dataBinary <- mergedData %>%
  mutate(injuryReported = case_when(
    INJURIES_TOTAL == 0 ~ 0,
    is.na(INJURIES_TOTAL) ~ NA_real_,
    TRUE ~ 1
  )) %>%
  mutate(
    postedSpeedLimit = POSTED_SPEED_LIMIT,
    trafficControlPresent = case_when(
      TRAFFIC_CONTROL_DEVICE == "NO CONTROLS" ~ 0,
      TRAFFIC_CONTROL_DEVICE == "UNKNOWN" ~ NA_real_,
      TRUE ~ 1
    ),
    weatherClear = case_when(
      WEATHER_CONDITION == "CLEAR" ~ 1,
      WEATHER_CONDITION == "UNKNOWN" ~ NA_real_,
      TRUE ~ 0
    ),
    isDaylight = case_when(
      LIGHTING_CONDITION == "DAYLIGHT" ~ 1,
      LIGHTING_CONDITION == "UNKNOWN" ~ NA_real_,
      TRUE ~ 0
    ),
    roadSurface = case_when(
      ROADWAY_SURFACE_COND == "NO DEFECTS" ~ 0,
      ROADWAY_SURFACE_COND == "UNKNOWN" ~ NA_real_,
      TRUE ~ 1
    ),
    damageOver1500 = case_when(
      DAMAGE == "OVER $1,500" ~ 1,
      is.na(DAMAGE) ~ NA_real_,
      TRUE ~ 0
    )
  )

# Remove rows with NA in injuryReported
dataBinary <- dataBinary %>% filter(!is.na(injuryReported))

# Split data into training and testing sets (80-20 split)
set.seed(432)
trainIndex <- createDataPartition(dataBinary$injuryReported, p = 0.8, list = FALSE)
training_factor <- dataBinary[trainIndex, ]
testing_factor <- dataBinary[-trainIndex, ]

# Define predictors
predictors <- c("postedSpeedLimit", "trafficControlPresent", "weatherClear",
                "isDaylight", "roadSurface", "damageOver1500")
```

```r
# Subset data to include only selected predictors and outcome using base R
train_subset <- training_factor[, c("injuryReported", predictors)]
test_subset <- testing_factor[, c("injuryReported", predictors)]

# Remove near-zero variance predictors using caret
nzv <- nearZeroVar(train_subset, saveMetrics = TRUE)
train_clean <- train_subset[, !nzv$nzv]
test_clean <- test_subset[, names(train_clean)]

# Remove rows with NA in predictors
train_clean <- train_clean %>% na.omit()
test_clean <- test_clean %>% na.omit()

# Check if test set is empty
if (nrow(test_clean) == 0) {
  stop("Test set is empty after preprocessing. Ensure the dataset has enough valid rows.")
}

# Update predictors list to include only columns in train_clean (excluding injuryReported)
predictors <- names(train_clean)[names(train_clean) != "injuryReported"]

# Check if any predictors remain
if (length(predictors) == 0) {
  stop("No predictors remain after preprocessing. Check variance and NA values in predictors.")
}

# Prepare data for KNN: convert to matrices and scale predictors
train_x <- as.matrix(train_clean[, predictors])
test_x <- as.matrix(test_clean[, predictors])
y_train <- train_clean$injuryReported
y_test <- test_clean$injuryReported

# Scale predictors
scale_params <- preProcess(train_x, method = c("center", "scale"))
train_x_scaled <- predict(scale_params, train_x)
test_x_scaled <- predict(scale_params, test_x)

# Test different k values (3, 5, 7, 9, 11)
k_values <- c(3, 5, 7, 9, 11)
results <- data.frame(k = k_values, AUC = NA, BalancedAccuracy = NA)

for (i in seq_along(k_values)) {
  k <- k_values[i]
  knn_result <- knn.reg(train = train_x_scaled, test = test_x_scaled, y = as.numeric(y_train), k = k)
  probs <- knn_result$pred
  preds <- ifelse(probs > 0.5, 1, 0)
  conf <- confusionMatrix(as.factor(preds), as.factor(y_test), positive = "1")
  roc_val <- roc(as.numeric(as.character(y_test)), probs)
  results$AUC[i] <- auc(roc_val)
  results$BalancedAccuracy[i] <- conf$byClass["Balanced Accuracy"]
}
```

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

```
## Setting direction: controls < cases

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

## Setting direction: controls < cases

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

## Setting direction: controls < cases

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

## Setting direction: controls < cases

## Warning in confusionMatrix.default(as.factor(preds), as.factor(y_test), :
## Levels are not in the same order for reference and data. Refactoring data to
## match.

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

```r
# Select best k based on AUC
best_k <- results$k[which.max(results$AUC)]
cat("Best k:", best_k, "\n")
```

```
## Best k: 11
```

```r
# Fit KNN with best k
knn_best <- knn.reg(train = train_x_scaled, test = test_x_scaled, y = as.numeric(y_train), k = best_k)
knn_probs <- knn_best$pred
knn_preds <- ifelse(knn_probs > 0.5, 1, 0)

# Evaluate best KNN model
conf_knn_opt <- confusionMatrix(as.factor(knn_preds), as.factor(y_test), positive = "1")
```

```
## Warning in confusionMatrix.default(as.factor(knn_preds), as.factor(y_test), :
## Levels are not in the same order for reference and data. Refactoring data to
## match.
```

```r
print(conf_knn_opt)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      0      1
##          0 148049  26073
##          1      0      0
##
##                Accuracy : 0.8503
##                  95% CI : (0.8486, 0.8519)
```

```
##     No Information Rate : 0.8503
##     P-Value [Acc > NIR] : 0.5017
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.0000
##             Specificity : 1.0000
##          Pos Pred Value :    NaN
##          Neg Pred Value : 0.8503
##              Prevalence : 0.1497
##          Detection Rate : 0.0000
##    Detection Prevalence : 0.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 1
##
```

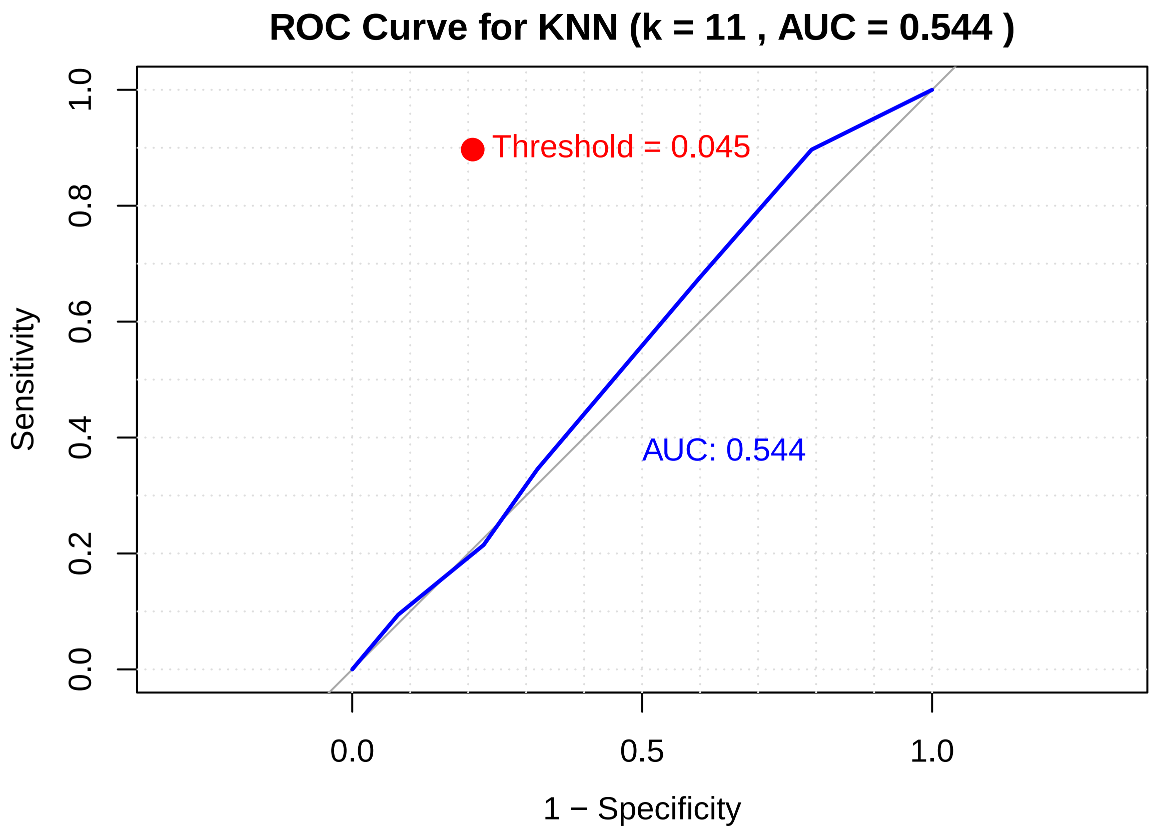
```
# Calculate AUC for best model
roc_knn_opt <- roc(as.numeric(as.character(y_test)), knn_probs)
```

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

```
auc_knn_opt <- auc(roc_knn_opt)
cat("AUC for best k:", auc_knn_opt, "\n")
```

```
## AUC for best k: 0.5442413
```

```
plot.roc(roc_knn_opt,
         main = paste("ROC Curve for KNN (k =", best_k, ", AUC =", round(auc_knn_opt, 3), ")"),
         col = "blue",
         print.auc = TRUE,
         print.auc.y = 0.4,
         legacy.axes = TRUE, # 1-specificity on x-axis
         grid = TRUE)

# Add optimal threshold point
best_coords <- coords(roc_knn_opt, "best", ret = c("threshold", "specificity", "sensitivity"), transpose
opt_thresh <- best_coords$threshold
points(1 - best_coords$specificity, best_coords$sensitivity,
       col = "red", pch = 19, cex = 1.5)
text(1 - best_coords$specificity, best_coords$sensitivity,
     labels = paste("Threshold =", round(opt_thresh, 3)),
     pos = 4, col = "red")
```



Ramdoom Forest

```
library(tidyverse)
library(caret)
library(randomForest)
```

```
## randomForest 4.7-1.2

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ranger':
##
##     importance

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

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(smotefamily)
library(pROC)
```

```r
dataBinary <- mergedData %>%
  mutate(injuryReported = case_when(
    INJURIES_TOTAL == 0 ~ 0,
    is.na(INJURIES_TOTAL) ~ NA_real_,
    TRUE ~ 1
  )) %>%
  mutate(
    postedSpeedLimit = POSTED_SPEED_LIMIT,
    trafficControlPresent = case_when(
      TRAFFIC_CONTROL_DEVICE == "NO CONTROLS" ~ 0,
      TRAFFIC_CONTROL_DEVICE == "UNKNOWN" ~ NA_real_,
      TRUE ~ 1
    ),
    weatherClear = case_when(
      WEATHER_CONDITION == "CLEAR" ~ 1,
      WEATHER_CONDITION == "UNKNOWN" ~ NA_real_,
      TRUE ~ 0
    ),
    isDaylight = case_when(
      LIGHTING_CONDITION == "DAYLIGHT" ~ 1,
      LIGHTING_CONDITION == "UNKNOWN" ~ NA_real_,
      TRUE ~ 0
    ),
    roadSurface = case_when(
      ROADWAY_SURFACE_COND == "NO DEFECTS" ~ 0,
      ROADWAY_SURFACE_COND == "UNKNOWN" ~ NA_real_,
      TRUE ~ 1
    ),
    damageOver1500 = case_when(
      DAMAGE == "OVER $1,500" ~ 1,
      is.na(DAMAGE) ~ NA_real_,
      TRUE ~ 0
    ),
    crashHour = CRASH_HOUR,
    crashDayOfWeek = CRASH_DAY_OF_WEEK,
    crashMonth = CRASH_MONTH
  )

# Remove rows with NA in injuryReported
dataBinary <- dataBinary %>% filter(!is.na(injuryReported))

# Split data into training and testing sets (80-20 split)
set.seed(432)
trainIndex <- createDataPartition(dataBinary$injuryReported, p = 0.8, list = FALSE)
training_factor <- dataBinary[trainIndex, ]
testing_factor <- dataBinary[-trainIndex, ]

# Define predictors
predictors <- c("postedSpeedLimit", "trafficControlPresent", "weatherClear",
                "isDaylight", "roadSurface", "damageOver1500",
```

```r
                "crashHour", "crashDayOfWeek", "crashMonth")

# Subset data to include only selected predictors and outcome using base R
train_subset <- training_factor[, c("injuryReported", predictors)]
test_subset <- testing_factor[, c("injuryReported", predictors)]

# Remove near-zero variance predictors using caret
nzv <- nearZeroVar(train_subset, saveMetrics = TRUE)
train_clean <- train_subset[, !nzv$nzv]
test_clean <- test_subset[, names(train_clean)]

# Remove rows with NA in predictors
train_clean <- train_clean %>% na.omit()
test_clean <- test_clean %>% na.omit()

# Check if test set is empty
if (nrow(test_clean) == 0) {
  stop("Test set is empty after preprocessing. Ensure the dataset has enough valid rows.")
}

# Update predictors list to include only columns in train_clean (excluding injuryReported)
predictors <- names(train_clean)[names(train_clean) != "injuryReported"]

# Check if any predictors remain
if (length(predictors) == 0) {
  stop("No predictors remain after preprocessing. Check variance and NA values in predictors.")
}

# Free memory before SMOTE
gc()
```

```
##              used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells    5233095  279.5    7898787  421.9   6031981  322.2
## Vcells  227369440 1734.7  401208860 3061.0 400487066 3055.5
```

```r
# Apply SMOTE to balance training data
set.seed(432)
train_clean$injuryReported <- factor(train_clean$injuryReported, levels = c("0", "1"))
train_balanced <- smotefamily::SMOTE(X = train_clean[, predictors],
                                     target = train_clean$injuryReported,
                                     K = 5, dup_size = 1)$data
names(train_balanced)[names(train_balanced) == "class"] <- "injuryReported"
train_balanced$injuryReported <- factor(train_balanced$injuryReported, levels = c("0", "1"))

# Free memory after SMOTE
gc()
```

```
##              used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells    5235068  279.6    7898787  421.9   7898787  421.9
## Vcells  234156490 1786.5  401208860 3061.0 401206612 3061.0
```

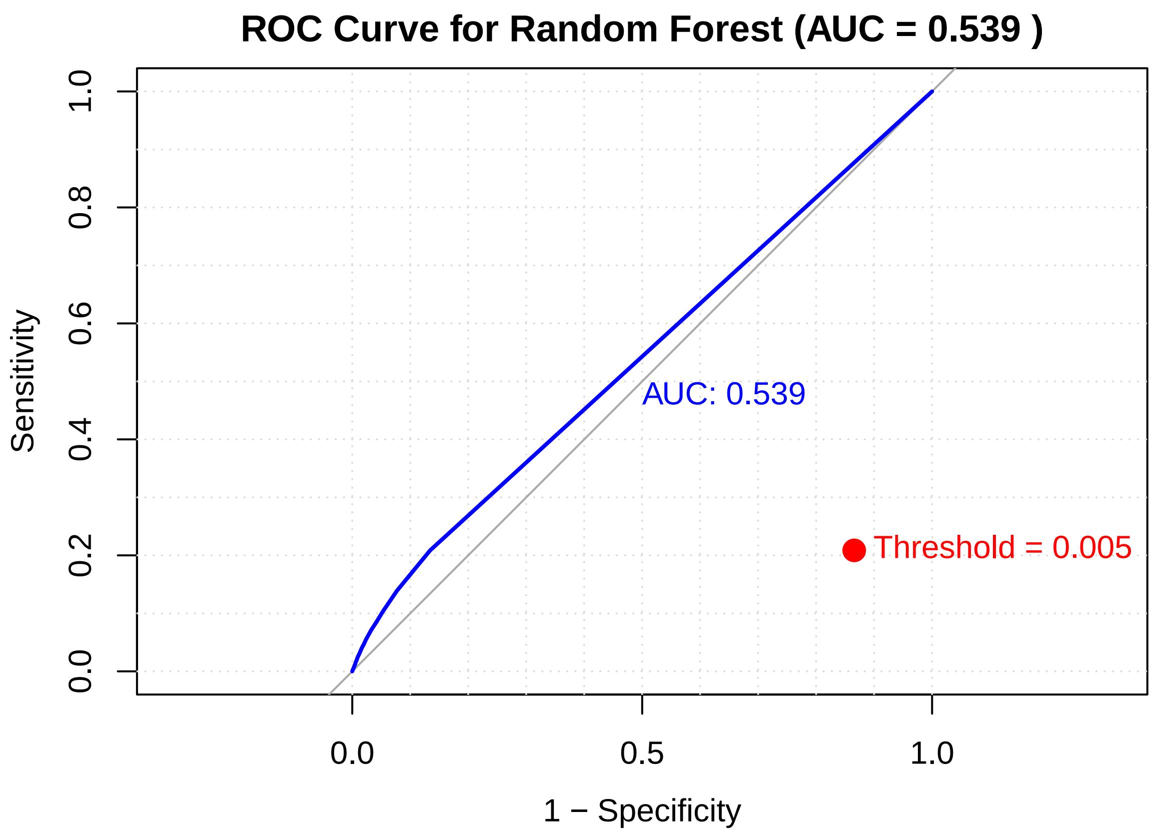
```r
# Prepare data for Random Forest
train_x <- train_balanced[, predictors]
test_x <- test_clean[, predictors]
y_train <- train_balanced$injuryReported
y_test <- factor(test_clean$injuryReported, levels = c("0", "1"))

# Fit Random Forest model with reduced parameters
set.seed(432)
rf_model <- randomForest(x = train_x, y = y_train,
                         ntree = 100, mtry = floor(sqrt(length(predictors))),
                         importance = TRUE, proximity = FALSE, keep.forest = TRUE)

# Predict probabilities on test set
rf_probs <- predict(rf_model, newdata = test_x, type = "prob")[, "1"]

# Optimize threshold using ROC to maximize sensitivity
roc_rf_opt <- roc(as.numeric(as.character(y_test)), rf_probs, levels = c("0", "1"), direction = "<")
best_coords <- coords(roc_rf_opt, "best", ret = c("threshold", "sensitivity", "specificity"),
                      best.method = "youden")
opt_thresh <- best_coords$threshold
rf_preds <- factor(ifelse(rf_probs > opt_thresh, "1", "0"), levels = c("0", "1"))

# Evaluate Random Forest model
conf_rf_opt <- confusionMatrix(rf_preds, y_test, positive = "1")
print(conf_rf_opt)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      0      1
##          0 128167  20634
##          1  19882   5439
##
##                Accuracy : 0.7673
##                  95% CI : (0.7653, 0.7693)
##     No Information Rate : 0.8503
##     P-Value [Acc > NIR] : 1.0000000
##
##                   Kappa : 0.0752
##
##  Mcnemar's Test P-Value : 0.0001907
##
##             Sensitivity : 0.20861
##             Specificity : 0.86571
##          Pos Pred Value : 0.21480
##          Neg Pred Value : 0.86133
##              Prevalence : 0.14974
##          Detection Rate : 0.03124
##    Detection Prevalence : 0.14542
##       Balanced Accuracy : 0.53716
##
##        'Positive' Class : 1
##
```

```r
# Calculate AUC
auc_rf_opt <- auc(roc_rf_opt)
cat("AUC:", auc_rf_opt, "\n")
```

```
## AUC: 0.539042
```

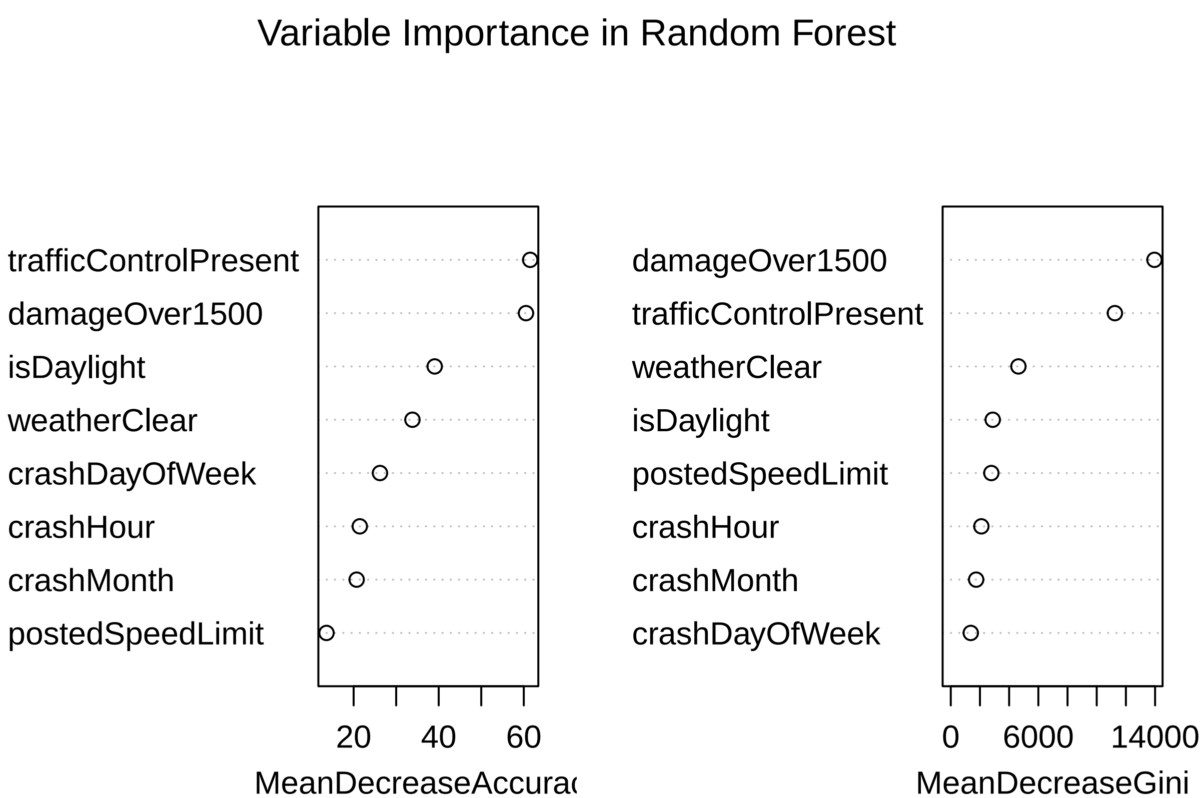
```r
# Plot ROC curve
plot.roc(roc_rf_opt, main = paste("ROC Curve for Random Forest (AUC =", round(auc_rf_opt, 3), ")"),
         col = "blue", print.auc = TRUE, grid = TRUE, legacy.axes = TRUE)
points(1 - best_coords$specificity, best_coords$sensitivity, col = "red", pch = 19, cex = 1.5)
text(1 - best_coords$specificity, best_coords$sensitivity,
     labels = paste("Threshold =", round(opt_thresh, 3)), pos = 4, col = "red")
```




```r
# Print variable importance
importance(rf_model)
```

```
##                                0        1 MeanDecreaseAccuracy MeanDecreaseGini
## postedSpeedLimit        8.926372 10.17315             13.62132         2779.976
## trafficControlPresent 54.242690 51.02357             61.50078        11245.133
## weatherClear          32.326570 33.45530             33.82366         4636.259
## isDaylight            18.872238 20.76324             39.06017         2875.402
## damageOver1500        51.386577 62.07875             60.51647        13955.370
## crashHour              5.398860 12.61025             21.44778         2095.960
## crashDayOfWeek         7.546829 13.67105             26.19305         1370.185
## crashMonth            14.460918 20.62816             20.70934         1740.526
```

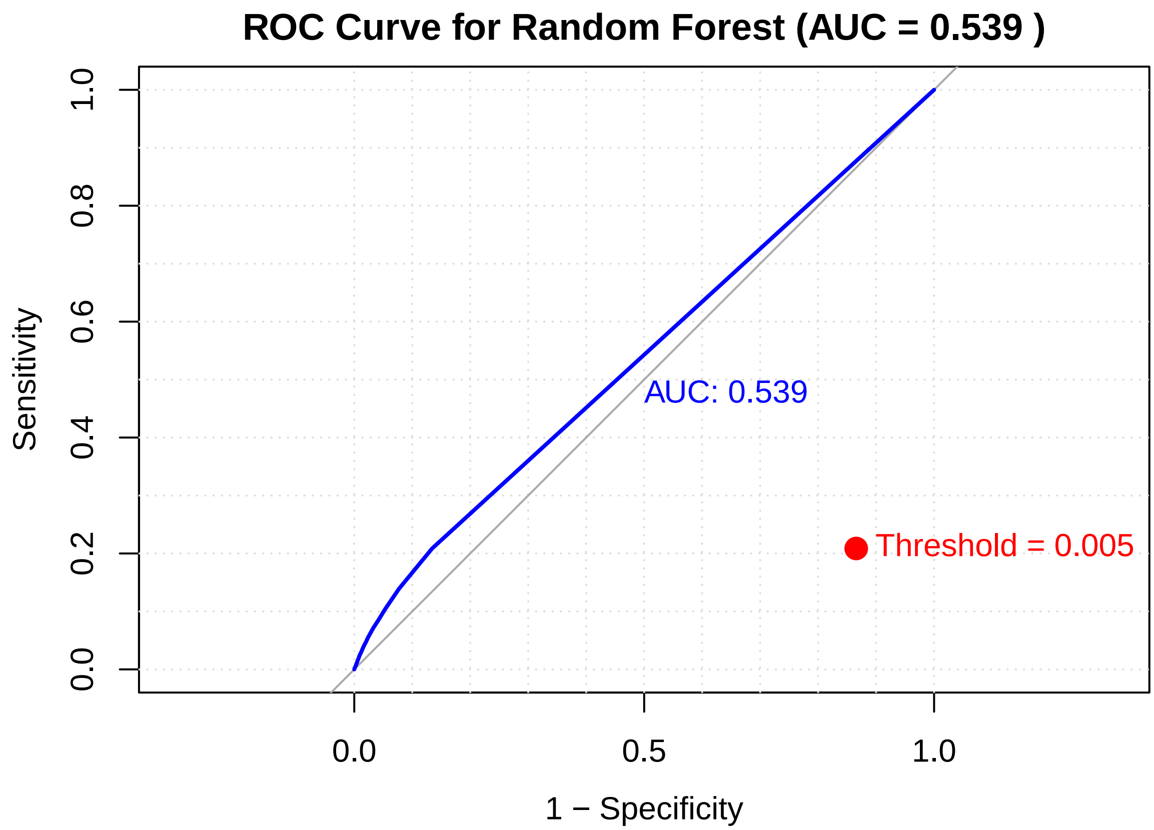
```r
varImpPlot(rf_model, main = "Variable Importance in Random Forest")
```



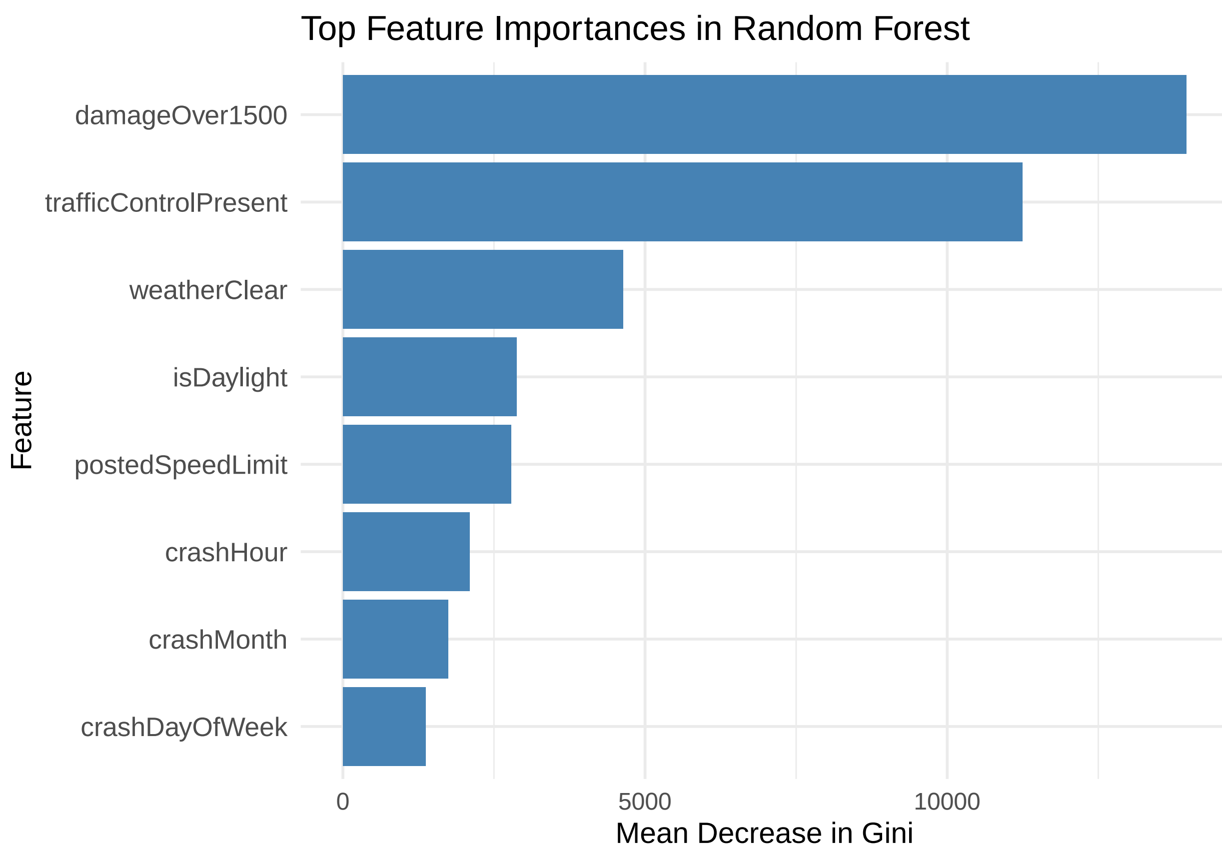

```r
# Free memory after execution
gc()
```

```
##              used   (Mb) gc trigger   (Mb)   max used    (Mb)
## Ncells    5353593  286.0    7898787  421.9    7898787   421.9
## Vcells 237079443 1808.8 1161912864 8864.7 1393274733 10629.9
```

```r
plot.roc(roc_rf_opt, main = paste("ROC Curve for Random Forest (AUC =", round(auc_rf_opt, 3), ")"),
         col = "blue", print.auc = TRUE, grid = TRUE, legacy.axes = TRUE)
points(1 - best_coords$specificity, best_coords$sensitivity, col = "red", pch = 19, cex = 1.5)
text(1 - best_coords$specificity, best_coords$sensitivity,
     labels = paste("Threshold =", round(opt_thresh, 3)), pos = 4, col = "red")
```




```r
# Extract and plot top 15 feature importances
imp <- importance(rf_model, type = 2)  # Mean Decrease in Gini
imp_df <- data.frame(Feature = rownames(imp), Importance = imp[, "MeanDecreaseGini"])
imp_df <- imp_df[order(imp_df$Importance, decreasing = TRUE), ]
imp_df_top <- head(imp_df, min(15, nrow(imp_df)))  # Top 15 or all if <15
imp_df_top$Feature <- factor(imp_df_top$Feature, levels = imp_df_top$Feature)

# Create bar plot with ggplot2
ggplot(imp_df_top, aes(x = reorder(Feature, Importance), y = Importance)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  labs(title = "Top Feature Importances in Random Forest",
       x = "Feature", y = "Mean Decrease in Gini") +
  theme_minimal() +
  theme(axis.text.y = element_text(size = 10))
```



Boosting

```
library(tidyverse)
library(caret)
library(xgboost)
library(smotefamily)
library(pROC)
```

```
dataBinary <- mergedData %>%
  mutate(injuryReported = case_when(
    INJURIES_TOTAL == 0 ~ 0,
    is.na(INJURIES_TOTAL) ~ NA_real_,
    TRUE ~ 1
  )) %>%
  mutate(
    postedSpeedLimit = POSTED_SPEED_LIMIT,
    trafficControlPresent = case_when(
      TRAFFIC_CONTROL_DEVICE == "NO CONTROLS" ~ 0,
      TRAFFIC_CONTROL_DEVICE == "UNKNOWN" ~ NA_real_,
      TRUE ~ 1
    ),
    weatherClear = case_when(
      WEATHER_CONDITION == "CLEAR" ~ 1,
      WEATHER_CONDITION == "UNKNOWN" ~ NA_real_,
```

```r
      TRUE ~ 0
    ),
    isDaylight = case_when(
      LIGHTING_CONDITION == "DAYLIGHT" ~ 1,
      LIGHTING_CONDITION == "UNKNOWN" ~ NA_real_,
      TRUE ~ 0
    ),
    roadSurface = case_when(
      ROADWAY_SURFACE_COND == "NO DEFECTS" ~ 0,
      ROADWAY_SURFACE_COND == "UNKNOWN" ~ NA_real_,
      TRUE ~ 1
    ),
    damageOver1500 = case_when(
      DAMAGE == "OVER $1,500" ~ 1,
      is.na(DAMAGE) ~ NA_real_,
      TRUE ~ 0
    ),
    crashHour = CRASH_HOUR,
    crashDayOfWeek = CRASH_DAY_OF_WEEK,
    crashMonth = CRASH_MONTH
  )

# Remove rows with NA in injuryReported
dataBinary <- dataBinary %>% filter(!is.na(injuryReported))

# Split data into training and testing sets (80-20 split)
set.seed(432)
trainIndex <- createDataPartition(dataBinary$injuryReported, p = 0.8, list = FALSE)
training_factor <- dataBinary[trainIndex, ]
testing_factor <- dataBinary[-trainIndex, ]

# Define predictors
predictors <- c("postedSpeedLimit", "trafficControlPresent", "weatherClear",
                "isDaylight", "roadSurface", "damageOver1500",
                "crashHour", "crashDayOfWeek", "crashMonth")

# Subset data to include only selected predictors and outcome using base R
train_subset <- training_factor[, c("injuryReported", predictors)]
test_subset <- testing_factor[, c("injuryReported", predictors)]

# Remove near-zero variance predictors using caret
nzv <- nearZeroVar(train_subset, saveMetrics = TRUE)
train_clean <- train_subset[, !nzv$nzv]
test_clean <- test_subset[, names(train_clean)]

# Remove rows with NA in predictors
train_clean <- train_clean %>% na.omit()
test_clean <- test_clean %>% na.omit()

# Check if test set is empty
if (nrow(test_clean) == 0) {
  stop("Test set is empty after preprocessing. Ensure the dataset has enough valid rows.")
}
```

```r
# Update predictors list to include only columns in train_clean (excluding injuryReported)
predictors <- names(train_clean)[names(train_clean) != "injuryReported"]

# Check if any predictors remain
if (length(predictors) == 0) {
  stop("No predictors remain after preprocessing. Check variance and NA values in predictors.")
}

# Free memory before SMOTE
gc()
```

```
##              used   (Mb) gc trigger   (Mb)   max used    (Mb)
## Ncells    5487007  293.1    9518544  508.4    7898787   421.9
## Vcells  239144989 1824.6  929530292 7091.8 1393274733 10629.9
```

```r
# Apply SMOTE to balance training data
set.seed(432)
train_clean$injuryReported <- factor(train_clean$injuryReported, levels = c("0", "1"))
train_balanced <- smotefamily::SMOTE(X = train_clean[, predictors],
                                     target = train_clean$injuryReported,
                                     K = 5, dup_size = 1)$data
names(train_balanced)[names(train_balanced) == "class"] <- "injuryReported"
train_balanced$injuryReported <- factor(train_balanced$injuryReported, levels = c("0", "1"))

# Free memory after SMOTE
gc()
```

```
##              used   (Mb) gc trigger   (Mb)   max used    (Mb)
## Ncells    5487039  293.1    9518544  508.4    9518544   508.4
## Vcells  245927840 1876.3  743624234 5673.5 1393274733 10629.9
```

```r
# Prepare data for XGBoost
train_x <- as.matrix(train_balanced[, predictors])
test_x <- as.matrix(test_clean[, predictors])
y_train <- as.numeric(as.character(train_balanced$injuryReported))
y_test <- factor(test_clean$injuryReported, levels = c("0", "1"))

# Create DMatrix for XGBoost
dtrain <- xgb.DMatrix(data = train_x, label = y_train)
dtest <- xgb.DMatrix(data = test_x, label = as.numeric(as.character(y_test)))

# Set XGBoost parameters
params <- list(
  objective = "binary:logistic",
  eta = 0.1,
  max_depth = 6,
  eval_metric = "auc"
)

# Fit XGBoost model
set.seed(432)
xgb_model <- xgb.train(params = params, data = dtrain, nrounds = 100)
```

```r
# Predict probabilities on test set
xgb_probs <- predict(xgb_model, dtest)

# Optimize threshold using ROC to maximize sensitivity
roc_xgb_opt <- roc(as.numeric(as.character(y_test)), xgb_probs, levels = c("0", "1"), direction = "<")
best_coords <- coords(roc_xgb_opt, "best", ret = c("threshold", "sensitivity", "specificity"),
                      best.method = "youden")
opt_thresh <- best_coords$threshold
xgb_preds <- factor(ifelse(xgb_probs > opt_thresh, "1", "0"), levels = c("0", "1"))

# Evaluate XGBoost model
conf_xgb_opt <- confusionMatrix(xgb_preds, y_test, positive = "1")
print(conf_xgb_opt)
```

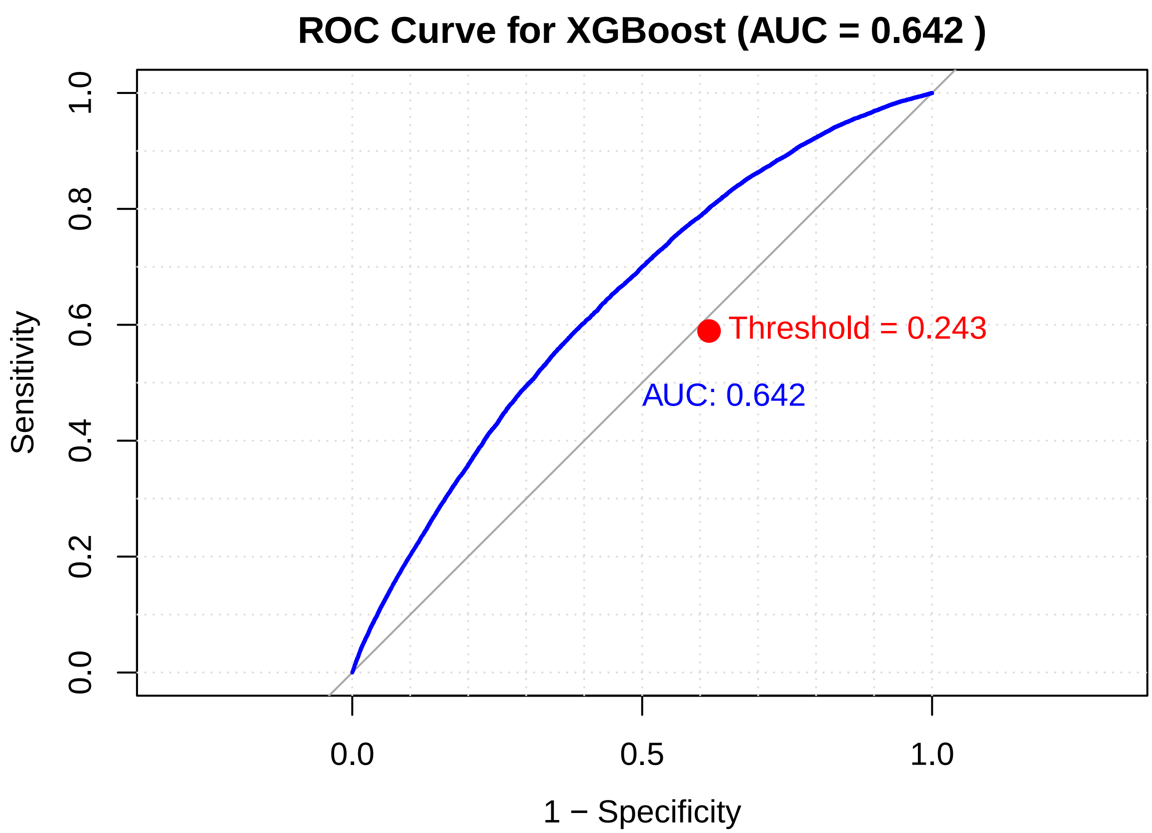
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 91099 10709
##          1 56950 15364
##
##                Accuracy : 0.6114
##                  95% CI : (0.6091, 0.6137)
##     No Information Rate : 0.8503
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1182
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.58927
##             Specificity : 0.61533
##          Pos Pred Value : 0.21246
##          Neg Pred Value : 0.89481
##              Prevalence : 0.14974
##          Detection Rate : 0.08824
##    Detection Prevalence : 0.41531
##       Balanced Accuracy : 0.60230
##
##        'Positive' Class : 1
##
```

```r
# Calculate AUC
auc_xgb_opt <- auc(roc_xgb_opt)
cat("AUC:", auc_xgb_opt, "\n")
```

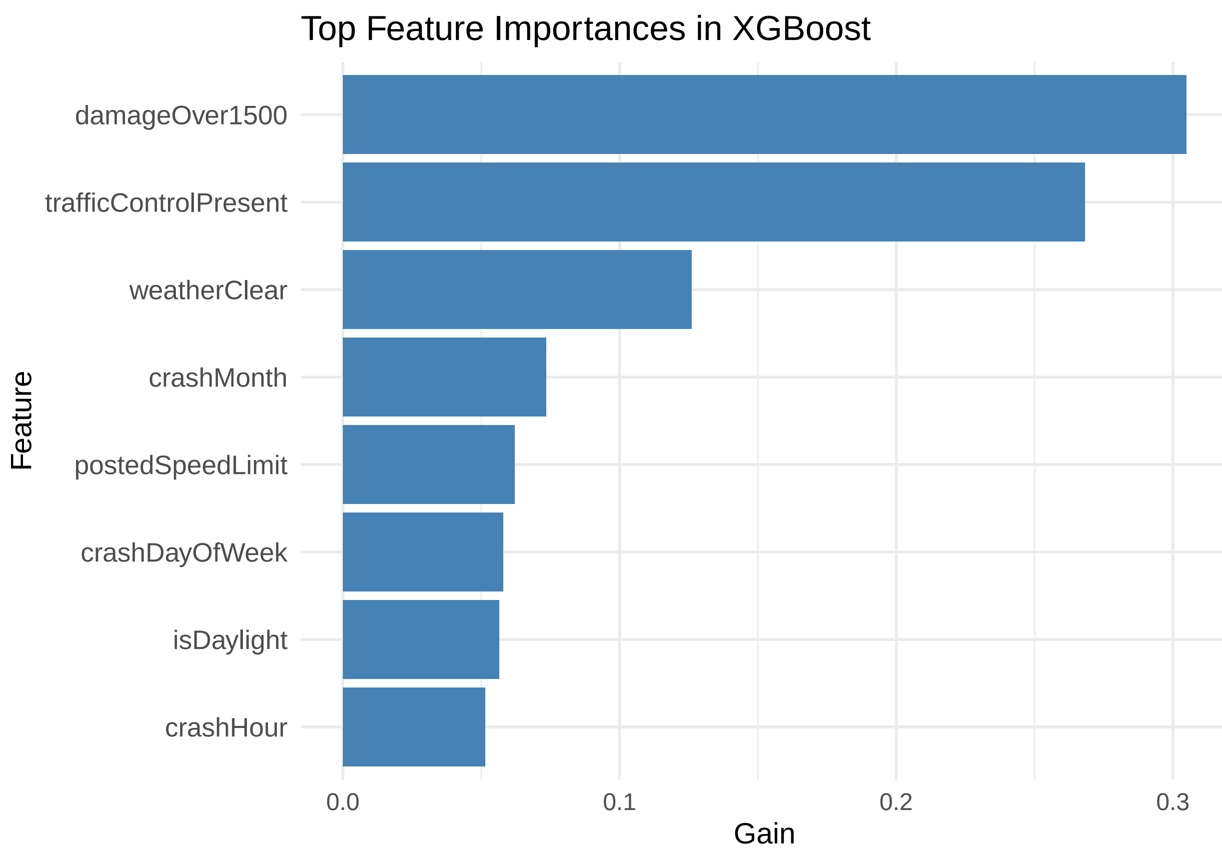
```
## AUC: 0.6424975
```

```r
# Plot ROC curve
plot.roc(roc_xgb_opt, main = paste("ROC Curve for XGBoost (AUC =", round(auc_xgb_opt, 3), ")"),
         col = "blue", print.auc = TRUE, grid = TRUE, legacy.axes = TRUE)
points(1 - best_coords$specificity, best_coords$sensitivity, col = "red", pch = 19, cex = 1.5)
```

```r
text(1 - best_coords$specificity, best_coords$sensitivity,
     labels = paste("Threshold =", round(opt_thresh, 3)), pos = 4, col = "red")
```




```r
# Extract and plot top 15 feature importances
imp <- xgb.importance(feature_names = predictors, model = xgb_model)
imp_df <- data.frame(Feature = imp$Feature, Importance = imp$Gain)
imp_df <- imp_df[order(imp_df$Importance, decreasing = TRUE), ]
imp_df_top <- head(imp_df, min(15, nrow(imp_df)))  # Top 15 or all if <15
imp_df_top$Feature <- factor(imp_df_top$Feature, levels = imp_df_top$Feature)

# Create bar plot with ggplot2
ggplot(imp_df_top, aes(x = reorder(Feature, Importance), y = Importance)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  labs(title = "Top Feature Importances in XGBoost",
       x = "Feature", y = "Gain") +
  theme_minimal() +
  theme(axis.text.y = element_text(size = 10))
```




```r
# Print feature importance table
print(imp_df)
```

```
##                  Feature Importance
## 1         damageOver1500 0.30478240
## 2 trafficControlPresent 0.26814895
## 3           weatherClear 0.12604865
## 4             crashMonth 0.07343816
## 5       postedSpeedLimit 0.06197150
## 6         crashDayOfWeek 0.05782426
## 7             isDaylight 0.05638361
## 8              crashHour 0.05140247
```

```r
# Free memory after execution
gc()
```

```
##              used   (Mb) gc trigger   (Mb)   max used    (Mb)
## Ncells    5537656  295.8    9518544  508.4    9518544   508.4
## Vcells 247904250 1891.4  743624234 5673.5 1393274733 10629.9
```

Discriminant Analysis and PCA

```r
library(tidyverse)
library(caret)
library(MASS)
library(smotefamily)
library(pROC)
```

```r
dataBinary <- mergedData %>%
  mutate(injuryReported = case_when(
    INJURIES_TOTAL == 0 ~ 0,
    is.na(INJURIES_TOTAL) ~ NA_real_,
    TRUE ~ 1
  )) %>%
  mutate(
    postedSpeedLimit = POSTED_SPEED_LIMIT,
    trafficControlPresent = case_when(
      TRAFFIC_CONTROL_DEVICE == "NO CONTROLS" ~ 0,
      TRAFFIC_CONTROL_DEVICE == "UNKNOWN" ~ NA_real_,
      TRUE ~ 1
    ),
    weatherClear = case_when(
      WEATHER_CONDITION == "CLEAR" ~ 1,
      WEATHER_CONDITION == "UNKNOWN" ~ NA_real_,
      TRUE ~ 0
    ),
    isDaylight = case_when(
      LIGHTING_CONDITION == "DAYLIGHT" ~ 1,
      LIGHTING_CONDITION == "UNKNOWN" ~ NA_real_,
      TRUE ~ 0
    ),
    roadSurface = case_when(
      ROADWAY_SURFACE_COND == "NO DEFECTS" ~ 0,
      ROADWAY_SURFACE_COND == "UNKNOWN" ~ NA_real_,
      TRUE ~ 1
    ),
    damageOver1500 = case_when(
      DAMAGE == "OVER $1,500" ~ 1,
      is.na(DAMAGE) ~ NA_real_,
      TRUE ~ 0
    ),
    crashHour = CRASH_HOUR,
    crashDayOfWeek = CRASH_DAY_OF_WEEK,
    crashMonth = CRASH_MONTH
  )

# Remove rows with NA in injuryReported
dataBinary <- dataBinary %>% filter(!is.na(injuryReported))

# Split data into training and testing sets (80-20 split)
set.seed(432)
trainIndex <- createDataPartition(dataBinary$injuryReported, p = 0.8, list = FALSE)
training_factor <- dataBinary[trainIndex, ]
testing_factor <- dataBinary[-trainIndex, ]
```

```r
# Define predictors
predictors <- c("postedSpeedLimit", "trafficControlPresent", "weatherClear",
                "isDaylight", "roadSurface", "damageOver1500",
                "crashHour", "crashDayOfWeek", "crashMonth")

# Subset data to include only selected predictors and outcome using base R
train_subset <- training_factor[, c("injuryReported", predictors)]
test_subset <- testing_factor[, c("injuryReported", predictors)]

# Remove near-zero variance predictors using caret
nzv <- nearZeroVar(train_subset, saveMetrics = TRUE)
if (sum(nzv$nzv) == length(predictors)) {
  stop("All predictors have near-zero variance. Check predictor variability or increase dataset size.")
}
train_clean <- train_subset[, !nzv$nzv]
test_clean <- test_subset[, names(train_clean)]

# Remove rows with NA in predictors
train_clean <- train_clean %>% na.omit()
test_clean <- test_clean %>% na.omit()

# Check if test set is empty
if (nrow(test_clean) == 0) {
  stop("Test set is empty after preprocessing. Ensure the dataset has enough valid rows.")
}

# Update predictors list to include only columns in train_clean (excluding injuryReported)
predictors <- names(train_clean)[names(train_clean) != "injuryReported"]

# Check if any predictors remain
if (length(predictors) == 0) {
  stop("No predictors remain after preprocessing. Check variance and NA values in predictors.")
}

# Check if enough observations for LDA
if (nrow(train_clean) <= length(predictors)) {
  stop("Number of observations must exceed number of predictors for LDA. Use a larger dataset.")
}

# Free memory before SMOTE
gc()
```

```
##              used   (Mb) gc trigger   (Mb)   max used    (Mb)
## Ncells    5529499  295.4    9518544  508.4    9518544   508.4
## Vcells 247619784 1889.2  743624234 5673.5 1393274733 10629.9
```

```r
# Apply SMOTE to balance training data
set.seed(432)
train_clean$injuryReported <- factor(train_clean$injuryReported, levels = c("0", "1"))
train_balanced <- smotefamily::SMOTE(X = train_clean[, predictors],
                                     target = train_clean$injuryReported,
                                     K = 5, dup_size = 1)$data
names(train_balanced)[names(train_balanced) == "class"] <- "injuryReported"
```

```r
train_balanced$injuryReported <- factor(train_balanced$injuryReported, levels = c("0", "1"))

# Check if SMOTE produced valid data
if (nrow(train_balanced) < length(predictors)) {
  stop("SMOTE produced insufficient data for LDA. Increase dup_size or dataset size.")
}

# Free memory after SMOTE
gc()
```

```
##              used   (Mb) gc trigger   (Mb)   max used    (Mb)
## Ncells   5529543  295.4    9518544  508.4    9518544   508.4
## Vcells 247197790 1886.0  743624234 5673.5 1393274733 10629.9
```

```r
# Prepare data for LDA and PCA
train_x <- as.matrix(train_balanced[, predictors])
test_x <- as.matrix(test_clean[, predictors])
y_train <- train_balanced$injuryReported
y_test <- factor(test_clean$injuryReported, levels = c("0", "1"))

# Scale predictors for PCA and LDA
scale_params <- preProcess(train_x, method = c("center", "scale"))
train_x_scaled <- predict(scale_params, train_x)
test_x_scaled <- predict(scale_params, test_x)

# --- Linear Discriminant Analysis (LDA) ---
# Fit LDA model
set.seed(432)
lda_model <- tryCatch(
  {
    lda(injuryReported ~ ., data = data.frame(train_x_scaled, injuryReported = y_train))
  },
  error = function(e) {
    stop("LDA model fitting failed: ", e$message, "\nCheck for sufficient observations or multicollinea
  }
)

# Check if lda_model$scaling is valid
if (is.null(lda_model$scaling) || length(lda_model$scaling) == 0) {
  stop("LDA model produced no scaling coefficients. Check predictor variability or dataset size.")
}

# Predict probabilities on test set
lda_probs <- predict(lda_model, newdata = data.frame(test_x_scaled))$posterior[, "1"]

# Optimize threshold using ROC
roc_lda_opt <- roc(as.numeric(as.character(y_test)), lda_probs, levels = c("0", "1"), direction = "<")
best_coords_lda <- coords(roc_lda_opt, "best", ret = c("threshold", "sensitivity", "specificity"),
                          best.method = "youden")
opt_thresh_lda <- best_coords_lda$threshold
lda_preds <- factor(ifelse(lda_probs > opt_thresh_lda, "1", "0"), levels = c("0", "1"))

# Evaluate LDA model
```

```r
conf_lda_opt <- confusionMatrix(lda_preds, y_test, positive = "1")
print("LDA Confusion Matrix and Statistics:")
```

```
## [1] "LDA Confusion Matrix and Statistics:"
```

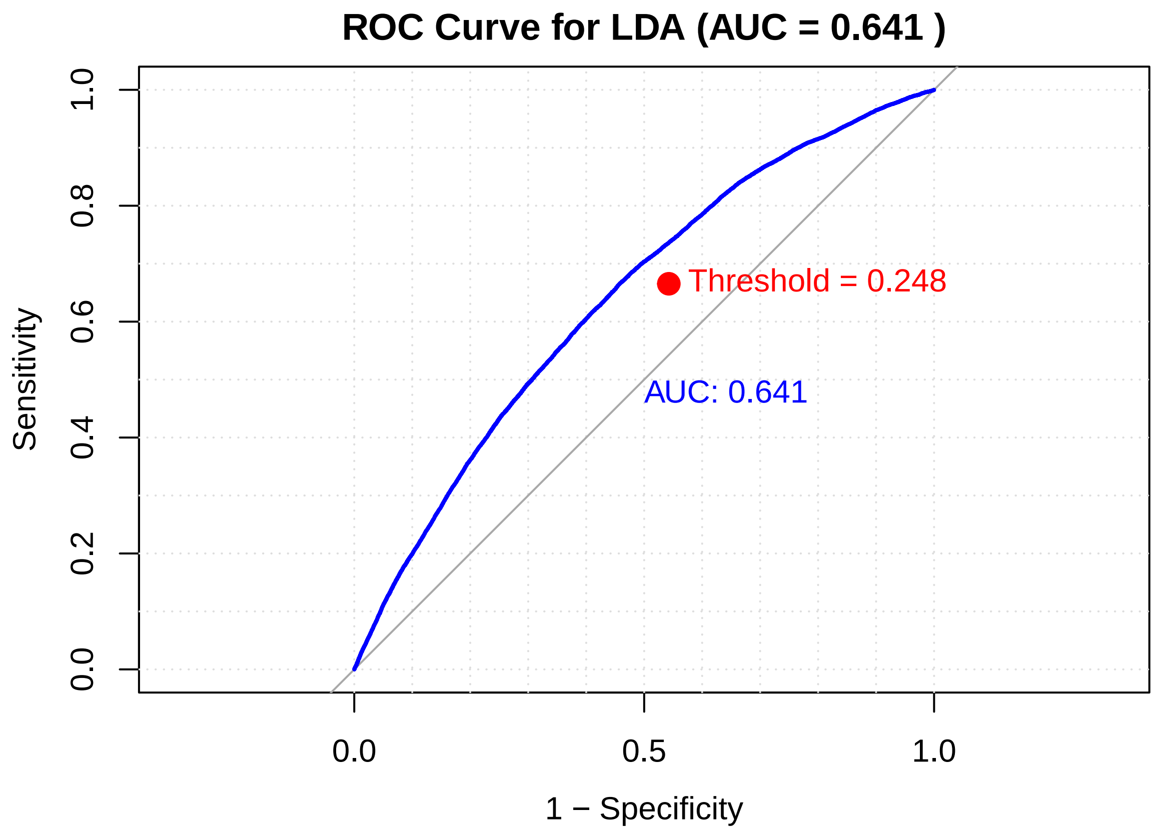
```r
print(conf_lda_opt)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 80320  8726
##          1 67729 17347
##
##                Accuracy : 0.5609
##                  95% CI : (0.5586, 0.5632)
##     No Information Rate : 0.8503
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1076
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.66532
##             Specificity : 0.54252
##          Pos Pred Value : 0.20390
##          Neg Pred Value : 0.90201
##              Prevalence : 0.14974
##          Detection Rate : 0.09963
##    Detection Prevalence : 0.48860
##       Balanced Accuracy : 0.60392
##
##        'Positive' Class : 1
##
```

```r
# Calculate AUC
auc_lda_opt <- auc(roc_lda_opt)
cat("LDA AUC:", auc_lda_opt, "\n")
```

```
## LDA AUC: 0.6405252
```

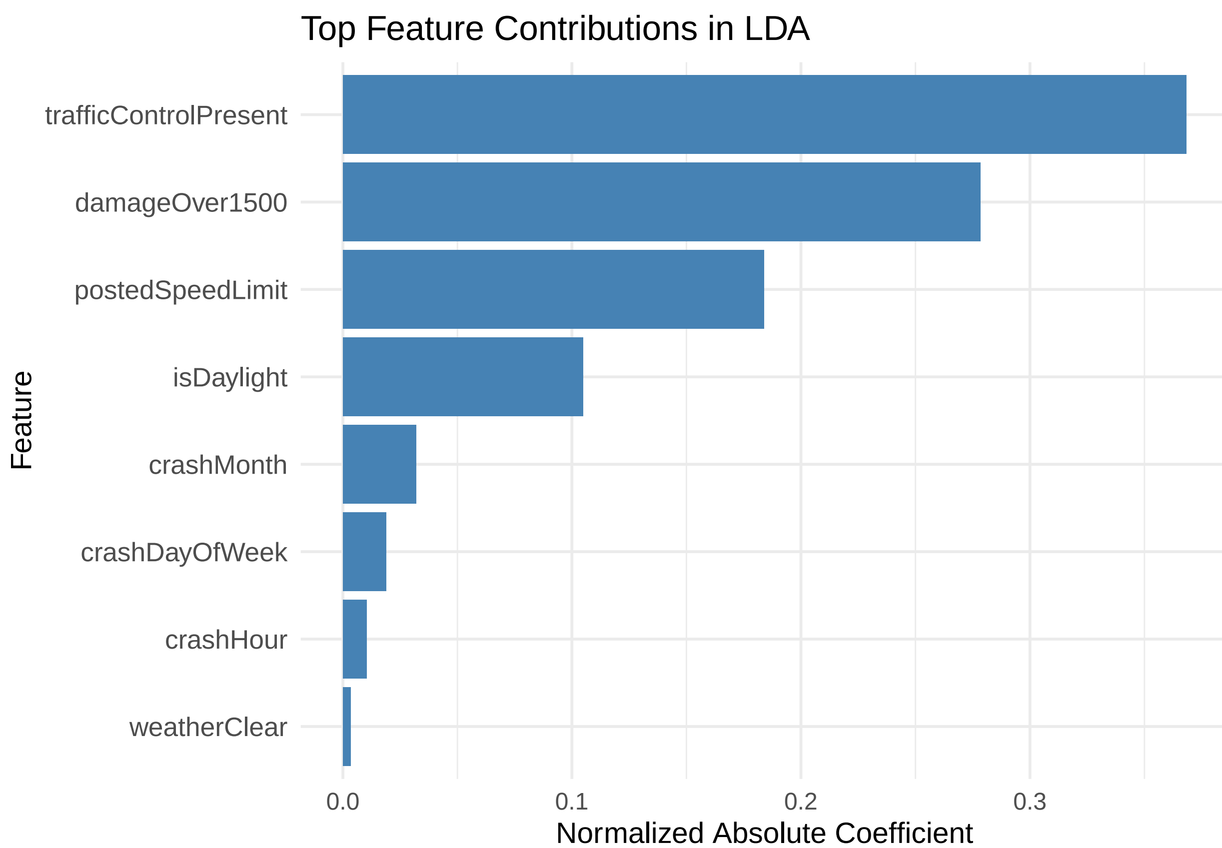
```r
# Plot ROC curve for LDA
plot.roc(roc_lda_opt, main = paste("ROC Curve for LDA (AUC =", round(auc_lda_opt, 3), ")"),
         col = "blue", print.auc = TRUE, grid = TRUE, legacy.axes = TRUE)
points(1 - best_coords_lda$specificity, best_coords_lda$sensitivity, col = "red", pch = 19, cex = 1.5)
text(1 - best_coords_lda$specificity, best_coords_lda$sensitivity,
     labels = paste("Threshold =", round(opt_thresh_lda, 3)), pos = 4, col = "red")
```




```
# Extract LDA feature contributions (correlations with discriminant function)
lda_scaling <- lda_model$scaling
lda_contrib <- abs(lda_scaling) / sum(abs(lda_scaling))  # Normalized absolute coefficients
if (length(names(lda_contrib)) == 0 || length(lda_contrib) != length(predictors)) {
  warning("LDA scaling coefficients do not match predictors. Using predictor names.")
  names(lda_contrib) <- predictors
}
```

```
## Warning: LDA scaling coefficients do not match predictors. Using predictor
## names.
```

```
imp_lda_df <- data.frame(Feature = names(lda_contrib), Importance = as.vector(lda_contrib))
imp_lda_df <- imp_lda_df[order(imp_lda_df$Importance, decreasing = TRUE), ]
imp_lda_df_top <- head(imp_lda_df, min(15, nrow(imp_lda_df)))
imp_lda_df_top$Feature <- factor(imp_lda_df_top$Feature, levels = imp_lda_df_top$Feature)

# Plot LDA feature contributions
ggplot(imp_lda_df_top, aes(x = reorder(Feature, Importance), y = Importance)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  labs(title = "Top Feature Contributions in LDA", x = "Feature", y = "Normalized Absolute Coefficient"
  theme_minimal() +
  theme(axis.text.y = element_text(size = 10))
```




```r
# Print LDA feature contributions
print("LDA Feature Contributions:")
```

```
## [1] "LDA Feature Contributions:"
```

```r
print(imp_lda_df)
```

```
##                 Feature  Importance
## 2 trafficControlPresent 0.368235191
## 5        damageOver1500 0.278276286
## 1       postedSpeedLimit 0.183928155
## 4            isDaylight 0.104828323
## 8            crashMonth 0.031949330
## 7        crashDayOfWeek 0.018850508
## 6             crashHour 0.010437181
## 3          weatherClear 0.003495026
```

```r
# --- PCA + LDA ---
# Perform PCA on training data
set.seed(432)
pca_model <- tryCatch(
  {
    prcomp(train_x_scaled, center = FALSE, scale. = FALSE)  # Already scaled
  },
```

```r
  error = function(e) {
    stop("PCA failed: ", e$message, "\nCheck for sufficient observations or predictor variability.")
  }
)

# Check if PCA produced valid components
explained_variance <- summary(pca_model)$importance[2, ]
if (length(explained_variance) == 0) {
  stop("PCA produced no components. Check predictor variability or dataset size.")
}
n_components <- min(sum(cumsum(explained_variance) < 0.95) + 1, length(predictors))  # Retain ~95% vari
if (n_components == 0) {
  n_components <- 1  # Use at least one component
}

# Transform data to PCA space
train_pca <- predict(pca_model, train_x_scaled)[, 1:n_components, drop = FALSE]
test_pca <- predict(pca_model, test_x_scaled)[, 1:n_components, drop = FALSE]

# Fit LDA on PCA components
lda_pca_model <- tryCatch(
  {
    lda(injuryReported ~ ., data = data.frame(train_pca, injuryReported = y_train))
  },
  error = function(e) {
    stop("LDA on PCA components failed: ", e$message, "\nCheck for sufficient observations or multicolli
  }
)

# Predict probabilities on test set
lda_pca_probs <- predict(lda_pca_model, newdata = data.frame(test_pca))$posterior[, "1"]

# Optimize threshold using ROC
roc_lda_pca_opt <- roc(as.numeric(as.character(y_test)), lda_pca_probs, levels = c("0", "1"), direction
best_coords_lda_pca <- coords(roc_lda_pca_opt, "best", ret = c("threshold", "sensitivity", "specificity
                              best.method = "youden")
opt_thresh_lda_pca <- best_coords_lda_pca$threshold
lda_pca_preds <- factor(ifelse(lda_pca_probs > opt_thresh_lda_pca, "1", "0"), levels = c("0", "1"))

# Evaluate LDA + PCA model
conf_lda_pca_opt <- confusionMatrix(lda_pca_preds, y_test, positive = "1")
print("LDA + PCA Confusion Matrix and Statistics:")
```

```
## [1] "LDA + PCA Confusion Matrix and Statistics:"
```

```r
print(conf_lda_pca_opt)
```

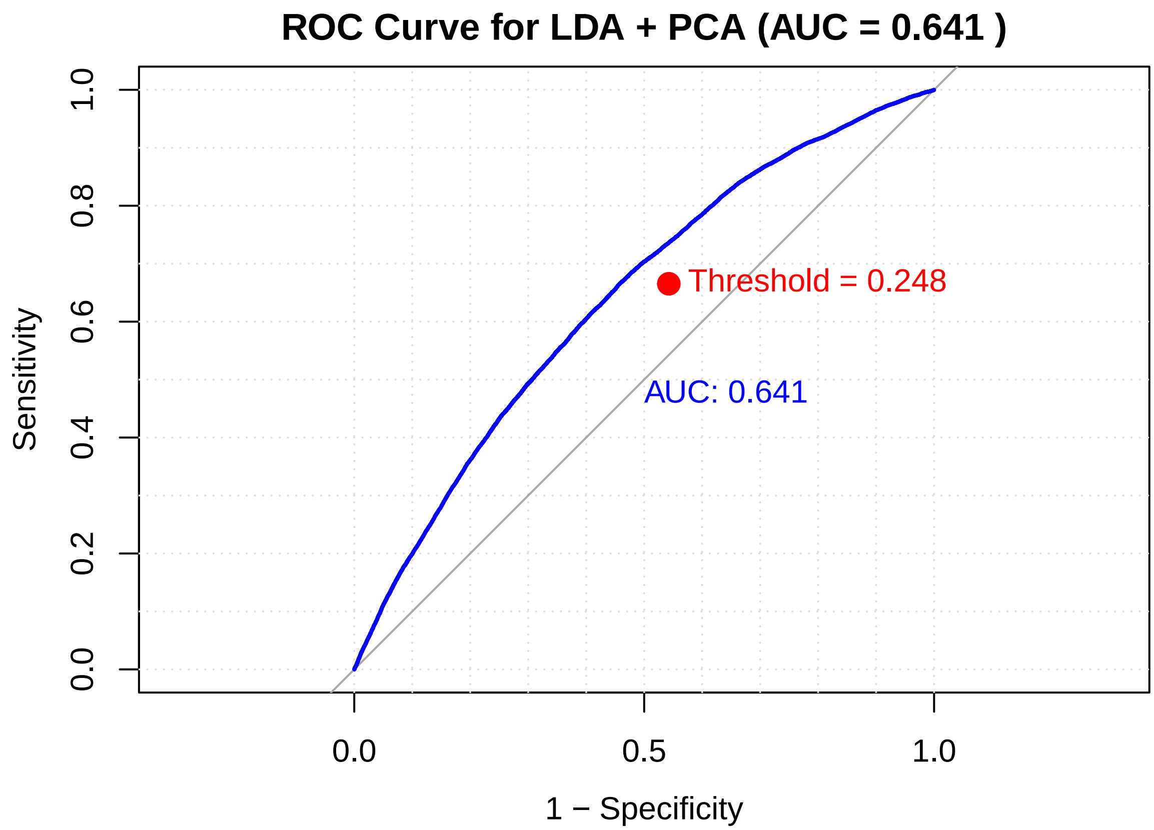
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 80320  8726
```

```
##          1 67729 17347
##
##                Accuracy : 0.5609
##                  95% CI : (0.5586, 0.5632)
##     No Information Rate : 0.8503
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1076
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.66532
##             Specificity : 0.54252
##          Pos Pred Value : 0.20390
##          Neg Pred Value : 0.90201
##              Prevalence : 0.14974
##          Detection Rate : 0.09963
##    Detection Prevalence : 0.48860
##       Balanced Accuracy : 0.60392
##
##        'Positive' Class : 1
##
```

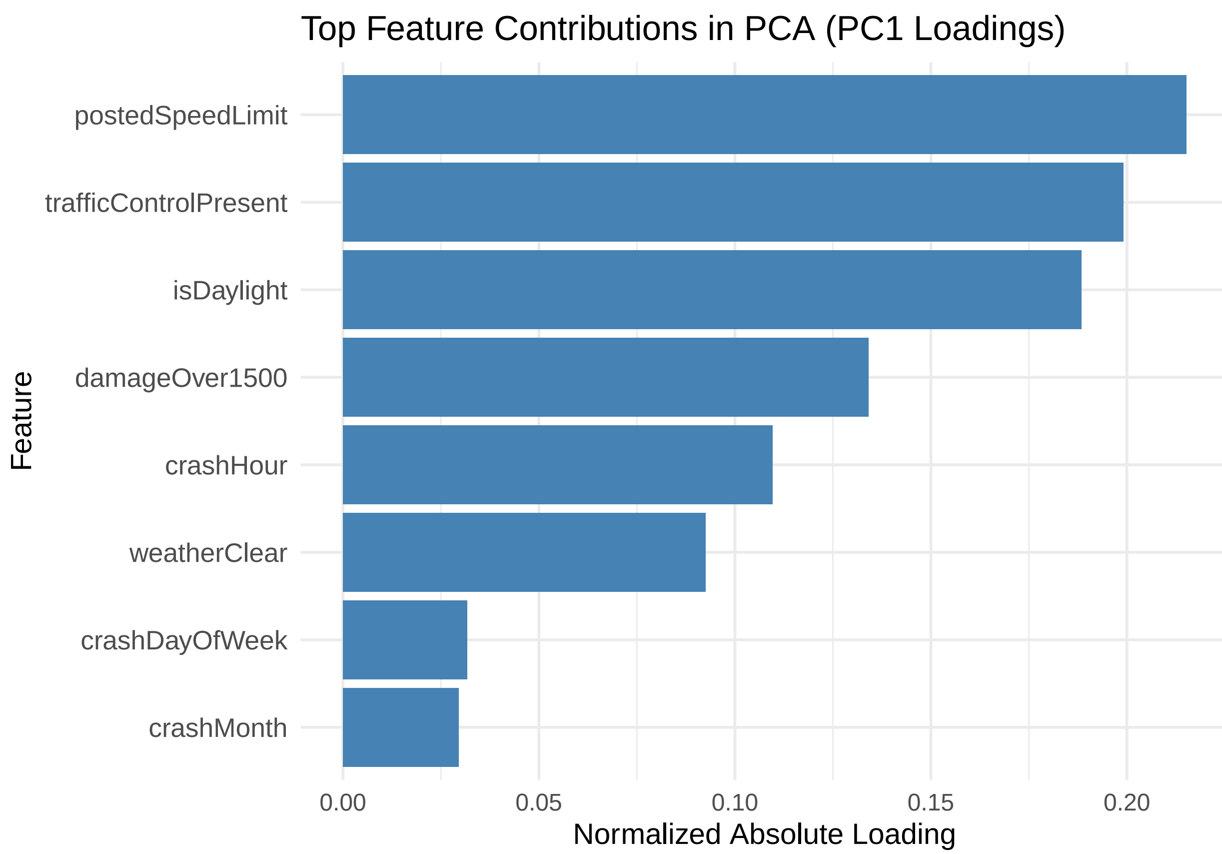
```r
# Calculate AUC
auc_lda_pca_opt <- auc(roc_lda_pca_opt)
cat("LDA + PCA AUC:", auc_lda_pca_opt, "\n")
```

```
## LDA + PCA AUC: 0.6405252
```

```r
# Plot ROC curve for LDA + PCA
plot.roc(roc_lda_pca_opt, main = paste("ROC Curve for LDA + PCA (AUC =", round(auc_lda_pca_opt, 3), ")")
         col = "blue", print.auc = TRUE, grid = TRUE, legacy.axes = TRUE)
points(1 - best_coords_lda_pca$specificity, best_coords_lda_pca$sensitivity, col = "red", pch = 19, cex
text(1 - best_coords_lda_pca$specificity, best_coords_lda_pca$sensitivity,
     labels = paste("Threshold =", round(opt_thresh_lda_pca, 3)), pos = 4, col = "red")
```




```r
# Extract PCA feature contributions (loadings from PC1)
pca_loadings <- abs(pca_model$rotation[, 1])  # Absolute loadings for PC1
pca_loadings <- pca_loadings / sum(pca_loadings)  # Normalize
imp_pca_df <- data.frame(Feature = names(pca_loadings), Importance = as.vector(pca_loadings))
imp_pca_df <- imp_pca_df[order(imp_pca_df$Importance, decreasing = TRUE), ]
imp_pca_df_top <- head(imp_pca_df, min(15, nrow(imp_pca_df)))
imp_pca_df_top$Feature <- factor(imp_pca_df_top$Feature, levels = imp_pca_df_top$Feature)

# Plot PCA feature contributions
ggplot(imp_pca_df_top, aes(x = reorder(Feature, Importance), y = Importance)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  labs(title = "Top Feature Contributions in PCA (PC1 Loadings)", x = "Feature", y = "Normalized Absolu
  theme_minimal() +
  theme(axis.text.y = element_text(size = 10))
```




```r
# Print PCA feature contributions
print("PCA Feature Contributions (PC1 Loadings):")
```

```
## [1] "PCA Feature Contributions (PC1 Loadings):"
```

```r
print(imp_pca_df)
```

```
##                 Feature Importance
## 1      postedSpeedLimit 0.21513116
## 2 trafficControlPresent 0.19908727
## 4            isDaylight 0.18838172
## 5        damageOver1500 0.13407458
## 6             crashHour 0.10960051
## 3          weatherClear 0.09254261
## 7        crashDayOfWeek 0.03168491
## 8            crashMonth 0.02949725
```

```r
# Free memory after execution
gc()
```

```
##              used   (Mb) gc trigger   (Mb)   max used    (Mb)
## Ncells    5553530  296.6    9518544  508.4    9518544   508.4
## Vcells 267677126 2042.3  743624234 5673.5 1393274733 10629.9
```