Data 621 - Homework 2

Group 4 Layla Quinones, Ian Costello, Dmitriy Burtsev & Esteban Aramayo

October 10, 2021

```r
library(kableExtra)
library(tidyverse)
library(tidymodels)
library(caret)
library(pROC)
```

1. Download the classification output data set (attached in Blackboard to the assignment).

```r
HW2_url = "https://raw.githubusercontent.com/MsQCompSci/Data621Group4/main/HW2/classification-output-da
df <- read.csv(HW2_url)
kable(head(df)) %>%
  kable_styling()
```

2. The data set has three key columns we will use:

- class: the actual class for the observation
- scored.class: the predicted class for the observation (based on a threshold of 0.5)
- scored.probability: the predicted probability of success for the observation

Use the table() function to get the raw confusion matrix for this scored dataset. Make sure you understand the output. In particular, do the rows represent the actual or predicted class? The columns?

The rows represents the *actual* class of for the observation. Given the medical statistics in the data, let's say this relates to the presence (or absence) of a medical condition. In this example, zero represents the absence

pregnant	glucose	diastolic	skinfold	insulin	bmi	pedigree	age	class	scored.class	scored.probability
7	124	70	33	215	25.5	0.161	37	0	0	0.3284523
2	122	76	27	200	35.9	0.483	26	0	0	0.2731904
3	107	62	13	48	22.9	0.678	23	1	0	0.1096604
1	91	64	24	0	29.2	0.192	21	0	0	0.0559984
4	83	86	19	0	29.3	0.317	34	0	0	0.1004907
1	100	74	12	46	19.5	0.149	28	0	0	0.0551546

	0	1
0	119	5
1	30	27

of the condition, one represents the presence. The columns are the scored class, meaning based on the data provided, the prediction of whether (to continue with our example) a condition *may be* present. Zero again representing absence, and one representing presence.

Now, where the actual and scored classes differ are the errors. Errors can be false positives (type I) or false negatives (type II). In table 1 below, the 30 value are the false negatives, where the predictor scored zero, but the actual class is one. And the 5 value is the false positive, where the predictor scored one, but the actual class is zero.

```
kable(table(df$class, df$scored.class)) %>%
  kable_styling()
```

3. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the accuracy of the predictions.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

```
confmat <- function(df){
  data.frame(TN=nrow(df[df$class==0 & df$scored.class==0,]),
             FN=nrow(df[df$class==1 & df$scored.class==0,]),
             FP=nrow(df[df$class==0 & df$scored.class==1,]),
             TP=nrow(df[df$class==1 & df$scored.class==1,])

  )
}

confmat(df)
```

```
##    TN FN FP TP
## 1 119 30  5 27
```

```
accuracy_score <- function(df){

  # Get confusion matrix from df data
  cm <- confmat(df)

  # Calculate accuracy from confusion matrix
  round(( cm$TP + cm$TN ) / ( cm$TP + cm$FP + cm$TN + cm$FN ),4)

}

# test accuracy_score
print(paste("Accuracy Score: ",accuracy_score(df)))
```

```
## [1] "Accuracy Score:  0.8066"
```

4. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the classification error rate of the predictions.

$$Classification\ Error\ Rate = \frac{FP + FN}{TP + FP + TN + FN}$$

```r
class_err_rate <- function(df){

  # Get confusion matrix from df data
  cm <- confmat(df)

  # Calculate Classification Error Rate from confusion matrix
  round(( cm$FP + cm$FN ) / ( cm$TP + cm$FP + cm$TN + cm$FN ),4)

}

# test class_err_rate
print(paste("Classification Error Rate: ", class_err_rate(df)))
```

```
## [1] "Classification Error Rate:  0.1934"
```

Verify that you get an accuracy and an error rate that sums to one.

```r
as <- accuracy_score(df)
cer <- class_err_rate(df)
print(paste("Accuracy Score: ", as))
```

```
## [1] "Accuracy Score:  0.8066"
```

```r
print(paste("Classification Error: ", cer))
```

```
## [1] "Classification Error:  0.1934"
```

```r
print(paste("Accuracy Score + Classification Error: ", as + cer))
```

```
## [1] "Accuracy Score + Classification Error:  1"
```

5. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the precision of the predictions.

$$Precision = \frac{TP}{TP + FP}$$

```r
precision_score <- function(df){

  # Get confusion matrix from df data
  cm <- confmat(df)

  # Calculate Precision from confusion matrix
  round(cm$TP / ( cm$TP + cm$FP ), 4)

}

# test precision_score
print(paste("Precision Score: ", precision_score(df)))
```

```
## [1] "Precision Score:  0.8438"
```

6. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the sensitivity of the predictions. Sensitivity is also known as recall.

$$Sensitivity = \frac{TP}{TP + FN}$$

```r
sensitivity_score <- function(df){

  # Get confusion matrix from df data
  cm <- confmat(df)

  # Calculate Sensitivity from confusion matrix
  round(cm$TP / ( cm$TP + cm$FN ), 4)

}

# test sensitivity_score
print(paste("Sensitivity Score: ", sensitivity_score(df)))
```

```
## [1] "Sensitivity Score:  0.4737"
```

7. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the specificity of the predictions.

$$Specificity = \frac{TN}{TN + FP}$$

```r
specificity_score <- function(df){

  # Get confusion matrix from df data
  cm <- confmat(df)
  # Calculate Specificity from confusion matrix
  round(cm$TN / ( cm$TN + cm$FP ), 4)

}

# test specificity_score
print(paste("Specificity Score: ", specificity_score(df)))
```

```
## [1] "Specificity Score:  0.9597"
```

8. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the F1 score of the predictions.

$$F1\ Score = \frac{2 * Precision * Sensitivity}{Precision + Sensitivity}$$

```
F1_score <- function(df){

  precis <- precision_score(df)
  sensit <- sensitivity_score(df)

  # Calculate F1 Score
  round(( 2 * precis * sensit ) / ( precis + sensit ), 4)

}

# test F1_score
print(paste("F1 Score: ", F1_score(df)))
```

```
## [1] "F1 Score:  0.6068"
```

9. Before we move on, let's consider a question that was asked: What are the bounds on the F1 score? Show that the F1 score will always be between 0 and 1.

(Hint: If $0 < a < 1$ and $0 < b < 1$ then $ab < a$.)

The maximum theoretical score for both precision and sensitivity is 1. The theoretical minimum for both is 0. By plugging in these values to the F1 score formula, we demonstrate that any score for precision and sensitivity could fall between 0, the theoretical minimum and 1, the theoretical maximum. If both sensitivty and precision are zero, the result is undefined. Even so, if either is zero and the other is greater than zero, the overall F1 score must also be zero.

###By definition, Precision and Sensitivity can only have values between 0 and 1.

Let's consider that Maximum case

If $Precision = 1$ and $Sensitivity = 1, then:$

$F1\ Score = \frac{2*Precision*Sensitivity}{Precision+Sensitivity}$

$F1\ Score = \frac{2*1*1}{1+1} = \frac{2}{2} = 1$ **(a)**

Let's consider Mininum cases

If $Precision = 0$ and $0 < Sensitivity < 1$, then:

$F1\ Score = \frac{2*Precision*Sensitivity}{Precision+Sensitivity}$

$F1\ Score = \frac{2*0*Sensitivity}{0+Sensitivity} = \frac{0}{Sensitivity} = 0$ **(b)**

–

If $0 < Precision < 1$ and $Sensitivity = 0$, then:

$F1\ Score = \frac{2*Precision*Sensitivity}{Precision+Sensitivity}$

$F1\ Score = \frac{2*Precision*0}{Precision+0} = \frac{0}{Precision} = 0$ **(c)**

Let's consider the undefined case

If $Precision = 0$ and $Sensitivity = 0$, then:

$F1\ Score = \frac{2*Precision*Sensitivity}{Precision+Sensitivity}$

$F1\ Score = \frac{2*0*0}{0+0} = \frac{0}{0} = Undefined$

Let's consider other cases

If $0 < Precision < 1$ and \$ 0 < Sensitivity 1, then:\$

By definition $F1\ Score = \frac{2*Precision*Sensitivity}{Precision+Sensitivity}$ **(d)**

We also know that, for any two real numbers a,b: if $0 < a < 1$ and $0 < b < 1$ then $ab < a$

Then, $Precision * Sensitivity < Precision$

So, the numerator "$2 * Precision * Sensitivity$" of the right side of equation **(d)**, will be less than its denominator "$Precision + Sensitivity$"

Consequently, the right side of equation **(d)** will take values in the interval $< 0, 1 >$ **(f)**

Finally, from results **(a)**, **(b)**, and **(f)**, we conclude that F1-Score can only take values between 0 and 1.

10. Write a function that generates an ROC curve from a data set with a true classification column (class in our example) and a probability column (scored.probability in our example).

Your function should return a list that includes the plot of the ROC curve and a vector that contains the calculated area under the curve (AUC). Note that I recommend using a sequence of thresholds ranging from 0 to 1 at 0.01 intervals.

```
AUC_ROC <- function(df){

  # Purpose: It returns a list that includes the plot of the ROC curve and a
  # vector that contains the calculated area under the curve (AUC)
  #
  # Note: AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve

  library(ggplot2)

  #seq_int <- seq(0,1,by=0.01)
  thresholds <- seq(0, 1, by = 0.01)


  TPR_vector <- c()
  FPR_vector <- c()


  # Build the TPR and FPR vectors using the thresholds vector
  for (i in 1:length(thresholds)){

    scored_class <- ifelse(df$scored.probability >= thresholds[i], 1, 0)

    rev_df <- data.frame(scored.class = scored_class, class = df$class)

    df_table <- with(rev_df, table(scored.class, class))

    TPR <- (df_table[4]) / (df_table[4] + df_table[3])

    FPR <- (df_table[2] / (df_table[2] + df_table[1]))

    TPR_vector[i] <- TPR
    FPR_vector[i] <- FPR

  }

  # define dataframe to be used for the ROC curve
  plot_df <- data.frame(TRUE_POSITIVE = TPR_vector, FALSE_POSITIVE = FPR_vector)




  # In order to roughly calculate the area under the curve, we must remove the NA values
  AUC_df <- plot_df[complete.cases(plot_df),]

  # Now to calculate the AUC
  x <- abs(diff(AUC_df$FALSE_POSITIVE))
  y <- AUC_df$TRUE_POSITIVE

  area_under_curve <- sum(x*y)

  plot_title <- paste0("Receiver Operating Characteristics Curve - (AUC = ", round(area_under_curve, 4)

  # store ROC plot in a variable
```

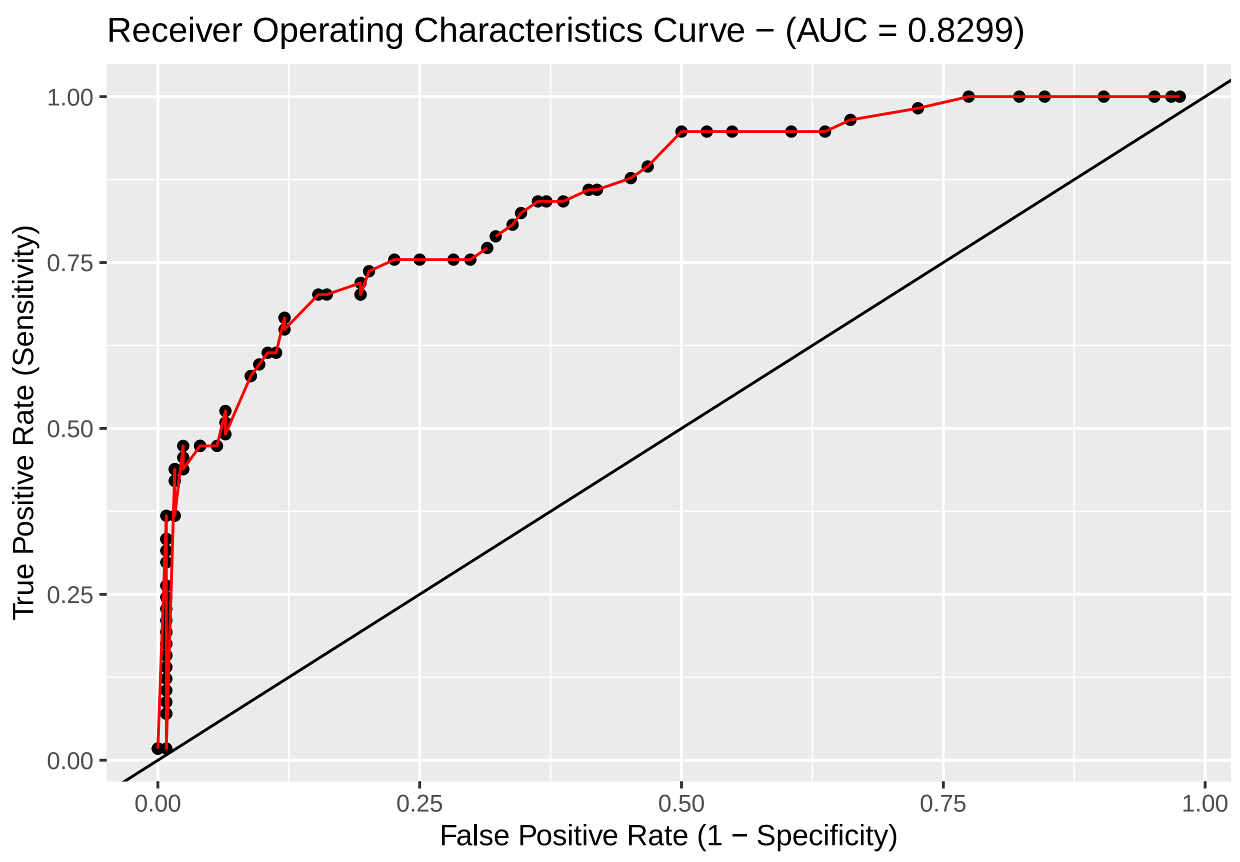
```r
  ROC_plot <- ggplot(plot_df, aes( x = FALSE_POSITIVE, y = TRUE_POSITIVE)) +
    geom_point() +
    geom_line(col="red") +
    geom_abline(intercept = 0, slope = 1) +
    labs(title = plot_title,
         x = "False Positive Rate (1 - Specificity)",
         y = "True Positive Rate (Sensitivity)")

  return(list(ROC_plot, round(area_under_curve, 4)))
}

# ROC_list <- AUC_ROC(df)
# ROC_plot <- ROC_list[[1]]
# AUC <- ROC_list[[2]]
# ROC_plot
# print(AUC)
```

11. Use your created R functions and the provided classification output data set to produce all of the classification metrics discussed above.

```r
metrics_names <- c("Accuracy",
                    "Classification Error Rate",
                    "Precision",
                    "Sensitivity",
                    "Specificity",
                    "F1 Score")

metrics_values <- c(accuracy_score(df),
             class_err_rate(df),
             precision_score(df),
             sensitivity_score(df),
             specificity_score(df),
             F1_score(df))



metrics_df <- data.frame(metrics_names, metrics_values)
colnames(metrics_df) <- c("Metric","Value")

kable(metrics_df) %>%
  kable_styling()
```

```r
ROC_list <- AUC_ROC(df)
ROC_plot <- ROC_list[[1]]
AUC <- ROC_list[[2]]

 ROC_plot
```

Metric	Value
Accuracy	0.8066
Classification Error Rate	0.1934
Precision	0.8438
Sensitivity	0.4737
Specificity	0.9597
F1 Score	0.6068



```
print(AUC)
```

```
## [1] 0.8299
```

12. Investigate the caret package. In particular, consider the functions confusionMatrix, sensitivity, and specificity. Apply the functions to the data set. How do the results compare with your own functions?

[IC Added: I really liked this overview of the `caret` package and its capabilities. https://www.machinelearningplus.com/machine-learning/caret-package/]

```
# Calculate the confusion matrix using Caret function
caret_cm <- caret::confusionMatrix(as.factor(df$scored.class),
                                   as.factor(df$class),mode='everything', positive = "1")


measure_names <- c("Accuracy",
                   "Sensitivity",
                   "Specificity",
                   "Classification Error Rate",
                   "Precision",
                   "F1 Score")

# Calculate performance measures using
caret_funcs_values <- c(round(caret_cm$overall["Accuracy"],4),
                    round(caret_cm$byClass["Sensitivity"], 4),
                    round(caret_cm$byClass["Specificity"],4),
                    NA,
                    round(caret_cm$byClass["Pos Pred Value"],4),
                    round(caret_cm$byClass["F1"],4))

custom_funcs_values <- c(accuracy_score(df),
                     sensitivity_score(df),
                     specificity_score(df),
                     class_err_rate(df),
                     precision_score(df),
                     F1_score(df)
                     )

comparison_df <- tibble(measure_names, caret_funcs_values, custom_funcs_values)


comparison_df <- comparison_df %>%
  mutate(Difference = caret_funcs_values - custom_funcs_values)

colnames(comparison_df) <- c("Performance Measure",
                             "Caret function result",
                             "Custom function result",
                             "Difference")
```

The table below shows the comparisons of the performance metrics between the Caret library's functions and our custom functions. In most cases we got the same results after rounding to 4 decimals. There is a minor difference for the F1 Score.

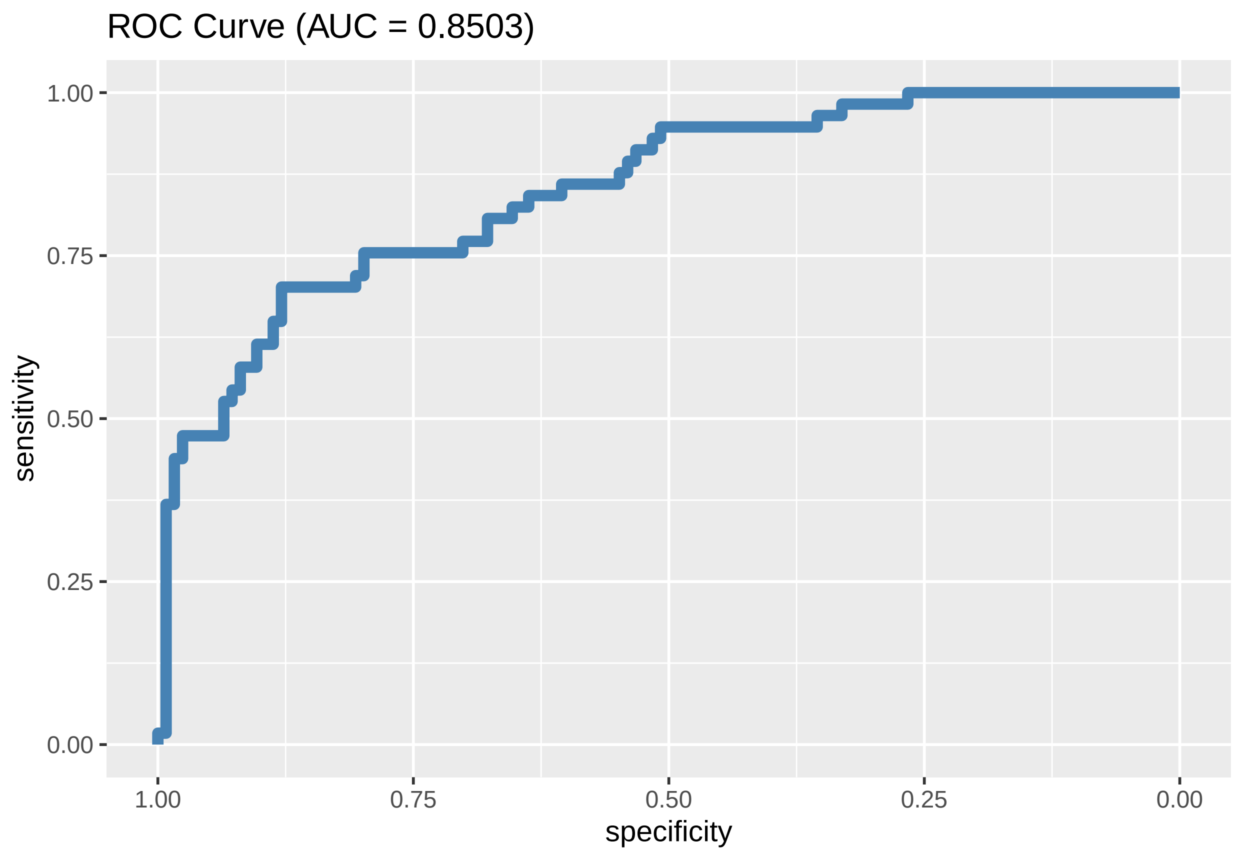
We did not find an equivalent function in the Caret library for the Classification Error Rate.

```
# Show results comparison
kable(comparison_df) %>%
  kable_styling()
```

Performance Measure	Caret function result	Custom function result	Difference
Accuracy	0.8066	0.8066	0e+00
Sensitivity	0.4737	0.4737	0e+00
Specificity	0.9597	0.9597	0e+00
Classification Error Rate	NA	0.1934	NA
Precision	0.8438	0.8438	0e+00
F1 Score	0.6067	0.6068	-1e-04

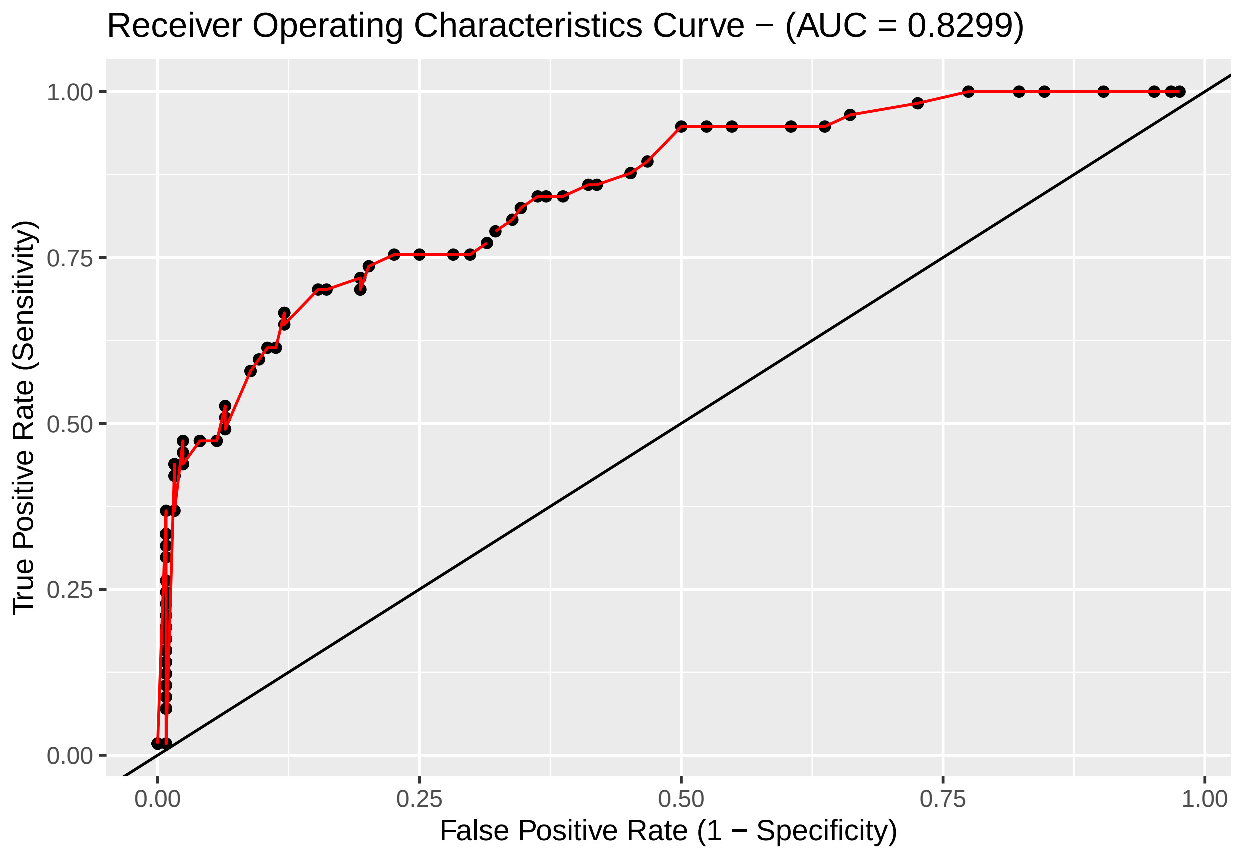
13. Investigate the pROC package. Use it to generate an ROC curve for the data set. How do the results compare with your own functions?

```
#define object to plot and calculate AUC
rocobj <- pROC::roc(df$class, df$scored.probability)
auc <- round(pROC::auc(df$class, df$scored.probability),4)

#create ROC plot
ggroc(rocobj, colour = 'steelblue', size = 2) +
  ggtitle(paste0('ROC Curve ', '(AUC = ', auc, ')'))
```



```
ROC_list <- AUC_ROC(df)
ROC_plot <- ROC_list[[1]]
```

```r
AUC <- ROC_list[[2]]

ROC_plot
```



Comparing the results of our custom functions vs the pROC's, we can see that the plots look very similar.

We can also see that our function returned AUC = 0.8299, while the pROC's version returned 0.8503. Our custom function result is very close to the library's.