hw2_group_final

2024-10-28

1. Overview data

```r
getwd()
```

```
## [1] "/Users/lilykuo/Desktop/MSBA/Business Analytics in R/hw2"
```

```r
setwd("/Users/lilykuo/Desktop/MSBA/Business Analytics in R/hw2")
library(dplyr)
```

```
##
## 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
```

```r
xyzdata <- read.csv("XYZData.csv")
summary(xyzdata)
```

```
##     user_id             age             male          friend_cnt
##  Min.   :     10   Min.   : 8.00   Min.   :0.0000   Min.   :   1.00
##  1st Qu.: 424372   1st Qu.:20.00   1st Qu.:0.0000   1st Qu.:   3.00
##  Median : 850230   Median :23.00   Median :1.0000   Median :   7.00
##  Mean   : 853236   Mean   :24.01   Mean   :0.6245   Mean   :  19.44
##  3rd Qu.:1281096   3rd Qu.:26.00   3rd Qu.:1.0000   3rd Qu.:  19.00
##  Max.   :1708935   Max.   :78.00   Max.   :1.0000   Max.   :6437.00
##  avg_friend_age avg_friend_male  friend_country_cnt subscriber_friend_cnt
##  Min.   : 9.00   Min.   :0.0000   Min.   :  0.000    Min.   :  0.0000
##  1st Qu.:20.75   1st Qu.:0.4286   1st Qu.:  1.000    1st Qu.:  0.0000
##  Median :23.00   Median :0.6667   Median :  2.000    Median :  0.0000
##  Mean   :24.07   Mean   :0.6185   Mean   :  4.088    Mean   :  0.4596
##  3rd Qu.:26.20   3rd Qu.:0.9000   3rd Qu.:  4.000    3rd Qu.:  0.0000
##  Max.   :77.00   Max.   :1.0000   Max.   :119.000    Max.   :225.0000
##  songsListened     lovedTracks           posts              playlists
##  Min.   :     0   Min.   :    0.00   Min.   :    0.000   Min.   : 0.0000
##  1st Qu.:  1444   1st Qu.:    1.00   1st Qu.:    0.000   1st Qu.: 0.0000
##  Median :  8102   Median :   16.00   Median :    0.000   Median : 0.0000
##  Mean   : 18598   Mean   :   92.52   Mean   :    5.465   Mean   : 0.5657
```

```
##  3rd Qu.: 24164   3rd Qu.:   78.00   3rd Qu.:    0.000   3rd Qu.: 1.0000
##  Max.   :922370   Max.   :10252.00   Max.   :10602.000   Max.   :98.0000
##      shouts          delta_friend_cnt    delta_avg_friend_age
##  Min.   :    0.00   Min.   :-212.0000   Min.   :-24.0000
##  1st Qu.:    1.00   1st Qu.:   0.0000   1st Qu.:  0.0000
##  Median :    4.00   Median :   0.0000   Median :  0.2500
##  Mean   :   30.64   Mean   :   0.8867   Mean   :  0.2771
##  3rd Qu.:   15.00   3rd Qu.:   0.0000   3rd Qu.:  0.4545
##  Max.   :15004.00   Max.   : 396.0000   Max.   : 27.5000
##  delta_avg_friend_male delta_friend_country_cnt delta_subscriber_friend_cnt
##  Min.   :-0.8000000    Min.   :-25.00           Min.   :-18.00000
##  1st Qu.: 0.0000000    1st Qu.:  0.00           1st Qu.:  0.00000
##  Median : 0.0000000    Median :  0.00           Median :  0.00000
##  Mean   :-0.0001958    Mean   :  0.11           Mean   : -0.02265
##  3rd Qu.: 0.0000000    3rd Qu.:  0.00           3rd Qu.:  0.00000
##  Max.   : 1.0000000    Max.   : 41.00           Max.   : 19.00000
##  delta_songsListened delta_lovedTracks  delta_posts       delta_playlists
##  Min.   :-135022.0   Min.   :-951.00   Min.   : -1.0000   Min.   :-3.000000
##  1st Qu.:       0.0   1st Qu.:   0.00   1st Qu.:  0.0000   1st Qu.: 0.000000
##  Median :       0.0   Median :   0.00   Median :  0.0000   Median : 0.000000
##  Mean   :     970.9   Mean   :   4.57   Mean   :  0.1077   Mean   : 0.003009
##  3rd Qu.:     950.5   3rd Qu.:   0.00   3rd Qu.:  0.0000   3rd Qu.: 0.000000
##  Max.   : 217876.0   Max.   :2319.00   Max.   :557.0000   Max.   : 9.000000
##   delta_shouts           tenure        good_country    delta_good_country
##  Min.   :-451.0000   Min.   :  1.00   Min.   :0.0000   Min.   :-1.0000000
##  1st Qu.:   0.0000   1st Qu.: 29.00   1st Qu.:0.0000   1st Qu.: 0.0000000
##  Median :   0.0000   Median : 45.00   Median :0.0000   Median : 0.0000000
##  Mean   :   0.9955   Mean   : 44.38   Mean   :0.3527   Mean   : 0.0003611
##  3rd Qu.:   0.0000   3rd Qu.: 59.00   3rd Qu.:1.0000   3rd Qu.: 0.0000000
##  Max.   :2036.0000   Max.   :108.00   Max.   :1.0000   Max.   : 1.0000000
##     adopter
##  Min.   :0.00000
##  1st Qu.:0.00000
##  Median :0.00000
##  Mean   :0.03707
##  3rd Qu.:0.00000
##  Max.   :1.00000
```

2. remove unnecessary value to predict (unique value):

```
xyzdata <- xyzdata %>% select(-user_id)
```

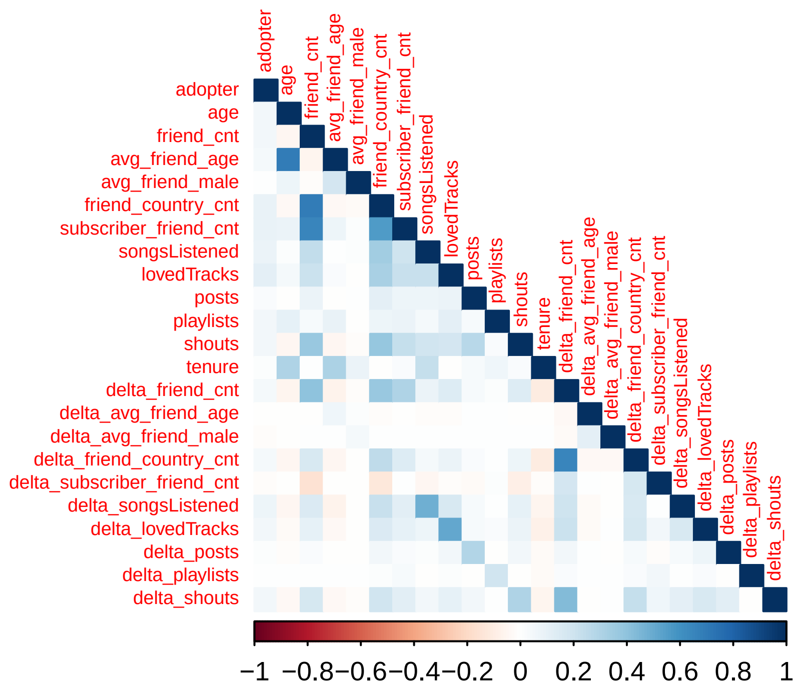
3. Overview overall correlation by visualization

```
library(ggplot2)
library(corrplot)
```

```
## corrplot 0.95 loaded
```

```
# choose only numeric variables
numericdata <- xyzdata[, c("adopter", "age", "friend_cnt", "avg_friend_age", "avg_friend_male",
                          "friend_country_cnt", "subscriber_friend_cnt", "songsListened",
```

```r
                              "lovedTracks", "posts", "playlists", "shouts", "tenure",
                              "delta_friend_cnt", "delta_avg_friend_age", "delta_avg_friend_male",
                              "delta_friend_country_cnt", "delta_subscriber_friend_cnt",
                              "delta_songsListened", "delta_lovedTracks", "delta_posts",
                              "delta_playlists", "delta_shouts")]
corr_matrix <- cor(numericdata)
corrplot(corr_matrix, method = "color", type = "lower", tl.cex = 0.6)
```



We can not see the distinct correlation between adopter and other variables with this map. However, we can check highly correalted variables and we can use this in the step of filter selection or additional analysis. Highly correlated variables : (avg_friend_age - age), (friend_country_cnt - friend_cnt), (subscriber_friend_cnt - friend_cnt), (delta_song_listened - song_listened), (delta_lovedTracks - lovedTracks), (delta_friend_country_cnt - delta_friend_cnt), …

4. Data Splitting We split the data in three-way to protect overfit.

```r
library(rpart)
library(caret)
```

```
## Loading required package: lattice
```

```r
set.seed(123)  # For reproducibility
# Create an initial split to separate training and the rest
train_rows <- createDataPartition(y = xyzdata$adopter, p = 0.70, list = FALSE)
```

```r
xyzdata_train <- xyzdata[train_rows,]
xyzdata_temp <- xyzdata[-train_rows,]  # Remaining data (30%)

# Split the remaining data into validation and test sets
val_rows <- createDataPartition(y = xyzdata_temp$adopter, p = 0.5, list = FALSE)  # 50% of the remainin
xyzdata_val <- xyzdata_temp[val_rows,]
xyzdata_test <- xyzdata_temp[-val_rows,]

table(xyzdata$adopter)
```

```
##
##     0     1
## 40000  1540
```

5. Handling Class Imbalance Smote and Rose do not work so we applied random over-sampling

```r
# Random over-sampling function
over_sample <- function(data, target, target_class) {
  # Get majority and minority class
  majority <- data[data[[target]] == target_class, ]
  minority <- data[data[[target]] != target_class, ]

  # Randomly sample with replacement from the minority class
  minority_sample <- minority[sample(nrow(minority), size = nrow(majority), replace = TRUE), ]

  # Combine the majority class with the sampled minority class
  balanced_data <- rbind(majority, minority_sample)

  return(balanced_data)
}

# Apply over-sampling
balanced_data <- over_sample(xyzdata_train, "adopter", target_class = 1)

# Check the class distribution
table(balanced_data$adopter)
```

```
##
##    0    1
## 1113 1113
```

```r
prop.table(table(balanced_data$adopter))
```

```
##
##   0   1
## 0.5 0.5
```

Train Random Forest Model

```r
# Install and load required libraries

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:ggplot2':
##
##     margin

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

```r
library(caret)
library(doParallel)  # Parallel processing library
```

```
## Loading required package: foreach

## Loading required package: iterators

## Loading required package: parallel
```

```r
balanced_data$adopter <- as.factor(balanced_data$adopter)

# Step 4: Set up parallel processing
cl <- makeCluster(detectCores() - 1)
registerDoParallel(cl)

# Step 5: Define cross-validation method
control <- trainControl(method = "cv", number = 5, verboseIter = TRUE)

# Step 6: Train the Random Forest model using balanced data
rf_cv_model <- train(adopter ~ .,
                     data = balanced_data,
                     method = "rf",
                     trControl = control,
                     tuneLength = 3,
                     ntree = 100,
                     importance = TRUE)
```

```
## Aggregating results
## Selecting tuning parameters
## Fitting mtry = 2 on full training set
```

```r
# Stop parallel processing
stopCluster(cl)

# Check the results
print(rf_cv_model)
```

```
## Random Forest
##
## 2226 samples
##   25 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1781, 1780, 1781, 1781, 1781
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.7156306  0.4312774
##   13    0.7097879  0.4196007
##   25    0.7061934  0.4124045
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

Evaluate the Model on the Validation Data - Confusion Matrix

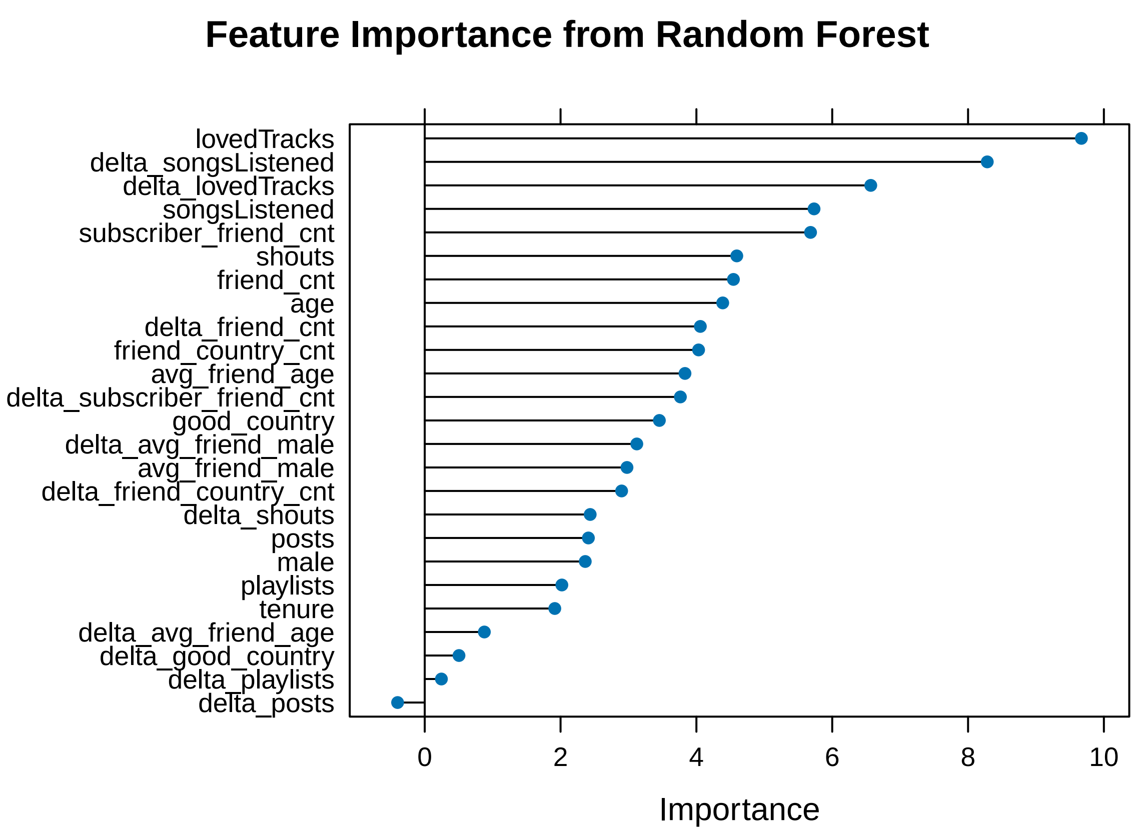
```r
# Ensure the target variable in the validation set is a factor
xyzdata_val$adopter <- as.factor(xyzdata_val$adopter)

# Make predictions using the trained Random Forest model
rf_predictions <- predict(rf_cv_model, xyzdata_val)

# Ensure predictions are factors with the same levels as the actual target variable
rf_predictions <- factor(rf_predictions, levels = levels(xyzdata_val$adopter))

# Confusion matrix to evaluate model performance
conf_matrix <- confusionMatrix(rf_predictions, xyzdata_val$adopter, positive = '1')
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4235   54
##          1 1776  166
##
##                Accuracy : 0.7063
##                  95% CI : (0.6948, 0.7176)
##     No Information Rate : 0.9647
##     P-Value [Acc > NIR] : 1
##
```

```
##                    Kappa : 0.0962
##
##   Mcnemar's Test P-Value : <2e-16
##
##              Sensitivity : 0.75455
##              Specificity : 0.70454
##           Pos Pred Value : 0.08548
##           Neg Pred Value : 0.98741
##               Prevalence : 0.03531
##           Detection Rate : 0.02664
##     Detection Prevalence : 0.31167
##        Balanced Accuracy : 0.72954
##
##         'Positive' Class : 1
##
```

Feature Importance

```r
# Get feature importance
importance_values <- varImp(rf_cv_model, scale = FALSE)

# Plot the feature importance
plot(importance_values, main = "Feature Importance from Random Forest")
```



Evaluate the Model on the Validation Data - ROC and AUC

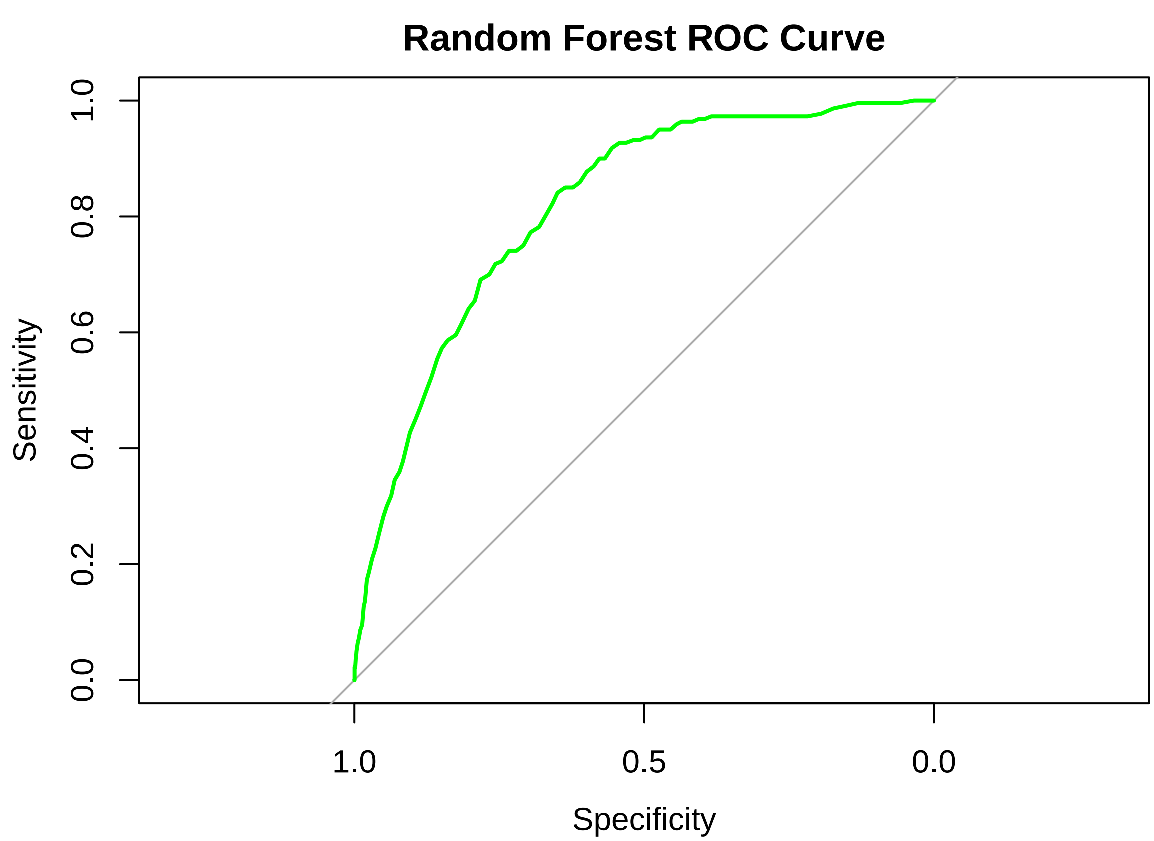
```r
# Predict probabilities (for ROC curve)
rf_probabilities <- predict(rf_cv_model, xyzdata_val, type = "prob")[,2]

# Load library for ROC and AUC
library(pROC)
```

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

##
## Attaching package: 'pROC'

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

```r
# Generate ROC curve
roc_curve_rf <- roc(xyzdata_val$adopter, rf_probabilities)
```

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

## Setting direction: controls < cases
```

```r
# Plot ROC curve
plot(roc_curve_rf, col = "green", main = "Random Forest ROC Curve")
```



```r
# AUC value
auc(roc_curve_rf)
```

```
## Area under the curve: 0.8156
```

Parameter tuning, using parallel processing to optimize speed

```r
str(xyzdata_train)
```

```
## 'data.frame':    29078 obs. of  26 variables:
##  $ age                      : int  24 22 18 24 23 22 41 21 33 26 ...
##  $ male                     : int  0 1 0 1 1 1 1 1 0 1 ...
##  $ friend_cnt               : int  20 4 3 131 15 13 17 71 13 1 ...
##  $ avg_friend_age           : num  26.3 21.2 18.5 23.9 21.6 ...
##  $ avg_friend_male          : num  0.778 0.75 0.667 0.431 0.462 ...
##  $ friend_country_cnt       : int  6 1 1 22 2 3 3 21 1 1 ...
##  $ subscriber_friend_cnt    : int  0 0 0 4 1 1 0 2 0 0 ...
##  $ songsListened            : int  37804 774 14036 3457 7506 2409 49303 68901 44156 4797 ...
##  $ lovedTracks              : int  4 0 1 227 0 12 178 37 9 4 ...
##  $ posts                    : int  20 0 0 1 18 0 2 41 0 0 ...
##  $ playlists                : int  1 0 1 2 1 0 2 1 0 0 ...
##  $ shouts                   : int  47 3 10 247 28 3 26 292 3 1 ...
##  $ delta_friend_cnt         : int  0 0 0 6 3 1 0 5 0 0 ...
##  $ delta_avg_friend_age     : num  0.222 1 0 0.315 0.841 ...
##  $ delta_avg_friend_male    : num  0 0 0 0.0129 -0.0385 ...
##  $ delta_friend_country_cnt : int  0 0 0 0 0 1 0 5 0 0 ...
##  $ delta_subscriber_friend_cnt: int  0 0 0 2 0 0 0 1 0 0 ...
##  $ delta_songsListened      : int  54 0 0 865 -4 630 717 2693 2460 0 ...
##  $ delta_lovedTracks        : int  0 0 0 7 0 3 1 0 0 0 ...
##  $ delta_posts              : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ delta_playlists          : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ delta_shouts             : int  0 0 0 4 6 1 0 0 0 0 ...
##  $ tenure                   : int  79 60 41 79 70 10 86 70 82 65 ...
##  $ good_country             : int  0 0 1 0 0 0 1 0 0 1 ...
##  $ delta_good_country       : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ adopter                  : int  0 0 0 0 0 0 0 0 0 0 ...
```

```r
xyzdata_train$adopter <- as.factor(xyzdata_train$adopter)

library(caret)
library(randomForest)
library(doParallel)
# Example using a small subset
set.seed(123)
# Ensure 'adopter' is a factor
clean_data <- na.omit(xyzdata_train)  # Remove any rows with NA values
clean_data$adopter <- as.factor(clean_data$adopter)

# Define control
control <- trainControl(method = "cv", number = 5, verboseIter = TRUE)

# Define tuning grid
```

```r
tune_grid <- expand.grid(mtry = c(2, 13, 25))

library(doParallel)

# Set up parallel processing
cl <- makeCluster(detectCores() - 1)  # Leave one core free
registerDoParallel(cl)

# Train your model as before
rf_tuned <- train(adopter ~ .,
                  data = xyzdata_train,
                  method = "rf",
                  trControl = control,
                  tuneGrid = tune_grid,
                  ntree = 100)  # Set number of trees
```

```
## Aggregating results
## Selecting tuning parameters
## Fitting mtry = 2 on full training set
```

```r
# Stop parallel processing
stopCluster(cl)

# Check best model
print(rf_tuned$bestTune)
```

```
##   mtry
## 1    2
```

```r
# Ensure the target variable in the validation set is a factor
xyzdata_val$adopter <- as.factor(xyzdata_val$adopter)

# Make predictions using the trained Random Forest model
rf_predictions <- predict(rf_cv_model, xyzdata_val)

# Ensure predictions are factors with the same levels as the actual target variable
rf_predictions <- factor(rf_predictions, levels = levels(xyzdata_val$adopter))

# Check if levels are consistent
print(levels(rf_predictions))
```

```
## [1] "0" "1"
```

```r
print(levels(xyzdata_val$adopter))
```

```
## [1] "0" "1"
```

```r
conf_matrix <- confusionMatrix(rf_predictions, xyzdata_val$adopter, positive = '1')
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4228   52
##          1 1783  168
##
##                Accuracy : 0.7055
##                  95% CI : (0.694, 0.7168)
##     No Information Rate : 0.9647
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0975
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.76364
##             Specificity : 0.70338
##          Pos Pred Value : 0.08611
##          Neg Pred Value : 0.98785
##              Prevalence : 0.03531
##          Detection Rate : 0.02696
##    Detection Prevalence : 0.31311
##       Balanced Accuracy : 0.73351
##
##        'Positive' Class : 1
##
```

Calculate Metrics

```r
# Extract values from confusion matrix
TP <- 168
FP <- 1783
FN <- 52

# Calculate precision and recall
precision <- TP / (TP + FP)
recall <- TP / (TP + FN)

# Calculate F1 Score
f1_score <- 2 * (precision * recall) / (precision + recall)

# Print results
cat("Precision:", precision, "\n")
```

```
## Precision: 0.08610969
```

```r
cat("Recall:", recall, "\n")
```

```
## Recall: 0.7636364
```

```r
cat("F1 Score:", f1_score, "\n")
```

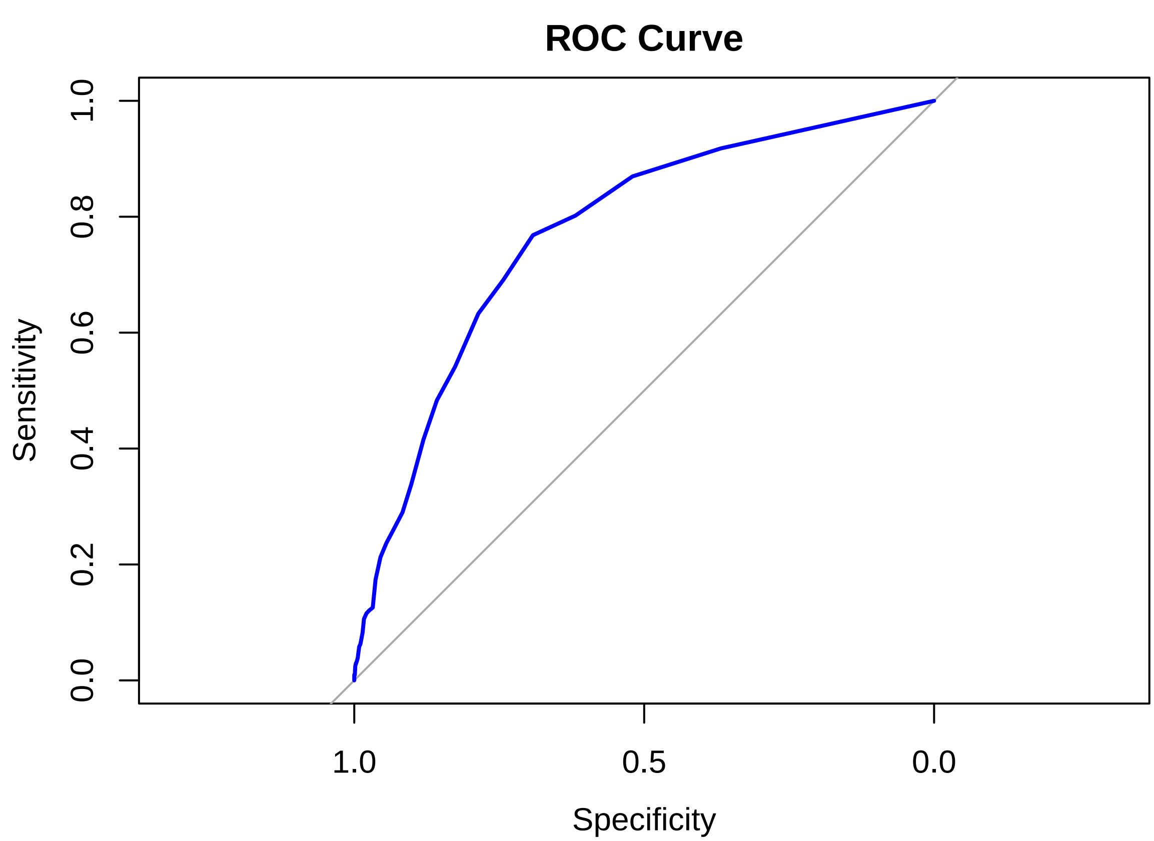
```
## F1 Score: 0.1547674
```

Evaluate the Model on the Testing Data - ROC and AUC

```r
# Load necessary libraries
library(pROC)

# Ensure the target variable in the test set is a factor
xyzdata_test$adopter <- as.factor(xyzdata_test$adopter)

# Get predicted probabilities for the positive class
rf_probabilities <- predict(rf_tuned, xyzdata_test, type = "prob")[, 2]  # Probability for class '1'

# Calculate the ROC curve
roc_curve <- roc(xyzdata_test$adopter, rf_probabilities)
```

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

## Setting direction: controls < cases
```

```r
# Plot the ROC curve
plot(roc_curve, col = "blue", main = "ROC Curve", lwd = 2)
```

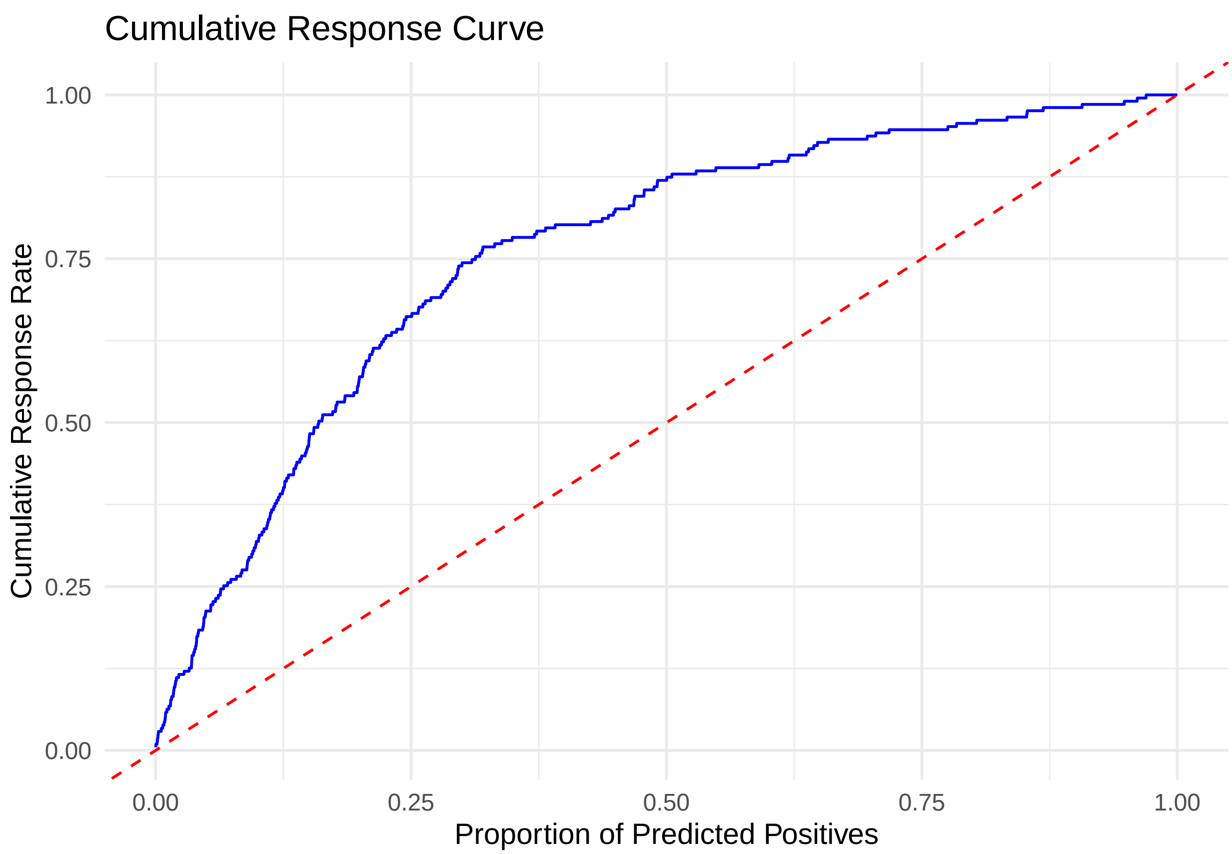


```r
# Calculate and print AUC
auc_value <- auc(roc_curve)
cat("AUC:", auc_value, "\n")
```

```
## AUC: 0.7717772
```

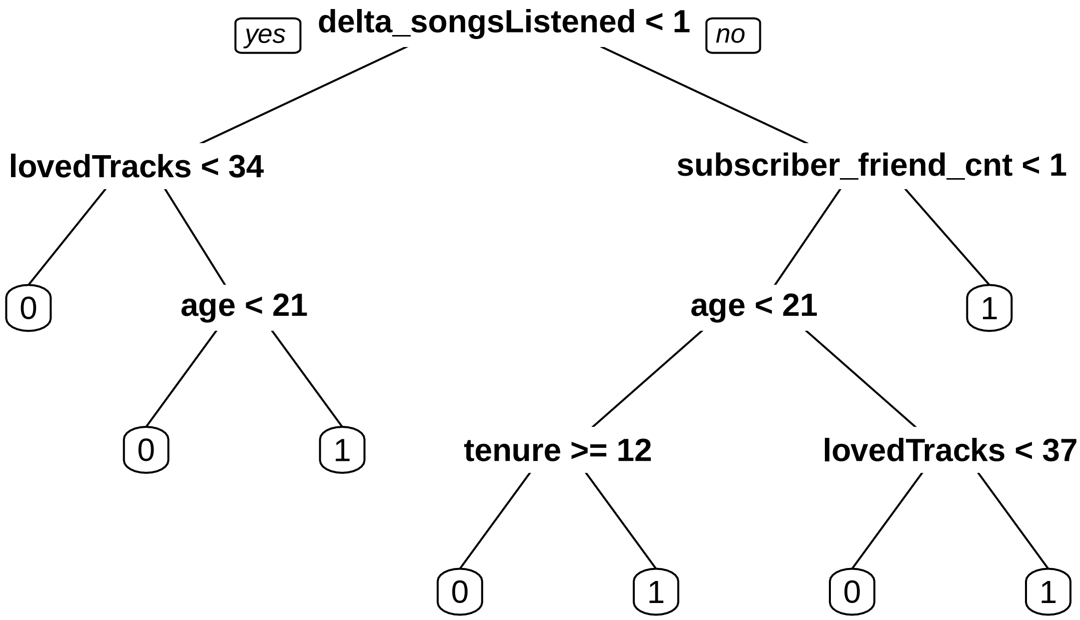
Evaluate the Model on the Testing Data - Cumulative Response Curve

```r
# Load necessary libraries
library(dplyr)
library(ggplot2)

# Assuming you already have predicted probabilities from your model
rf_probabilities <- predict(rf_tuned, xyzdata_test, type = "prob")[, 2]  # Probability for class '1'

# Create a data frame with actual values and predicted probabilities
results <- data.frame(actual = xyzdata_test$adopter, predicted_prob = rf_probabilities)

# Sort by predicted probabilities
results <- results %>%
  arrange(desc(predicted_prob))

# Calculate cumulative true positives and total positives
results$cumulative_true_positives <- cumsum(results$actual == '1')
total_positives <- sum(results$actual == '1')

# Calculate the proportion of predicted positives and cumulative response rate
results <- results %>%
  mutate(cumulative_response_rate = cumulative_true_positives / total_positives,
         proportion_predicted_positives = seq(1, nrow(results)) / nrow(results))

# Plot the Cumulative Response Curve
ggplot(results, aes(x = proportion_predicted_positives, y = cumulative_response_rate)) +
  geom_line(color = "blue") +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed", color = "red") +
  labs(title = "Cumulative Response Curve",
       x = "Proportion of Predicted Positives",
       y = "Cumulative Response Rate") +
  theme_minimal()
```



Besides random forest model, we also trained a decision tree model to compare model performance

```
# Modeling - Decision Tree on the balanced training dataset
# training Decision Tree
tree = rpart(adopter ~ ., data = balanced_data,
             method = "class",
             parms = list(split = "information"))

# print out the tree
library(rpart.plot)
prp(tree, varlen = 0)
```



Evaluate the Model on the Validation Data - Confusion Matrix

```
pred_tree = predict(tree, xyzdata_val, type="class")

confusionMatrix(data = pred_tree,
                reference = xyzdata_val$adopter,
                mode = "prec_recall",
                positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3948   52
##          1 2063  168
##
##                Accuracy : 0.6606
##                  95% CI : (0.6487, 0.6723)
##     No Information Rate : 0.9647
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0778
##
##  Mcnemar's Test P-Value : <2e-16
##
##               Precision : 0.07530
```

```
##                  Recall : 0.76364
##                      F1 : 0.13709
##              Prevalence : 0.03531
##          Detection Rate : 0.02696
##    Detection Prevalence : 0.35805
##       Balanced Accuracy : 0.71022
##
##        'Positive' Class : 1
##
```

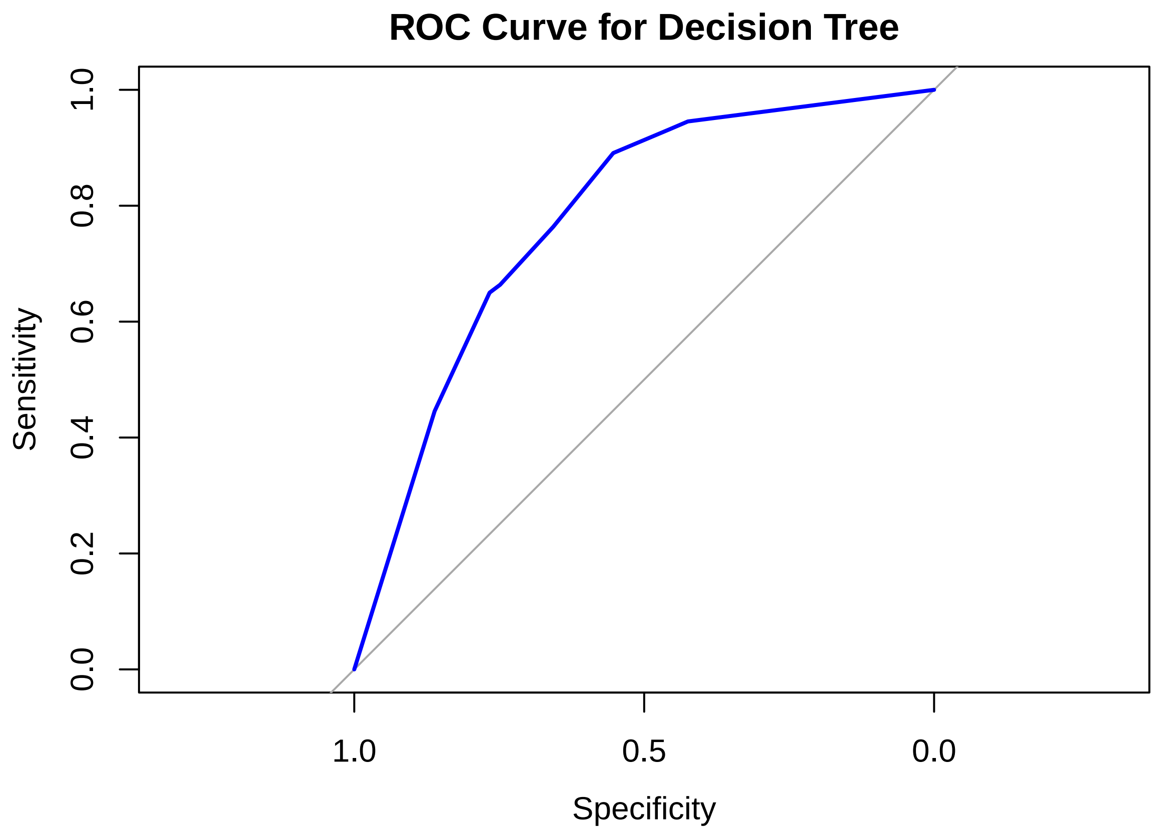
Evaluate the Model on the Validation Data - ROC and AUC

```r
# Predict probabilities for the validation set (needed for ROC/AUC)
pred_prob_tree <- predict(tree, xyzdata_val, type = "prob")[, 2]

# Check that the adopter column in the test set is a factor or numeric
xyzdata_val$adopter <- as.factor(xyzdata_val$adopter)

# Calculate ROC and AUC
roc_curve <- roc(xyzdata_val$adopter, pred_prob_tree)
```

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

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

```r
# Plot ROC curve
plot(roc_curve, col = "blue", main = "ROC Curve for Decision Tree")
```




```r
# Calculate AUC value
auc_value <- auc(roc_curve)
print(paste("AUC Value: ", auc_value))
```

```
## [1] "AUC Value:  0.776927526806914"
```

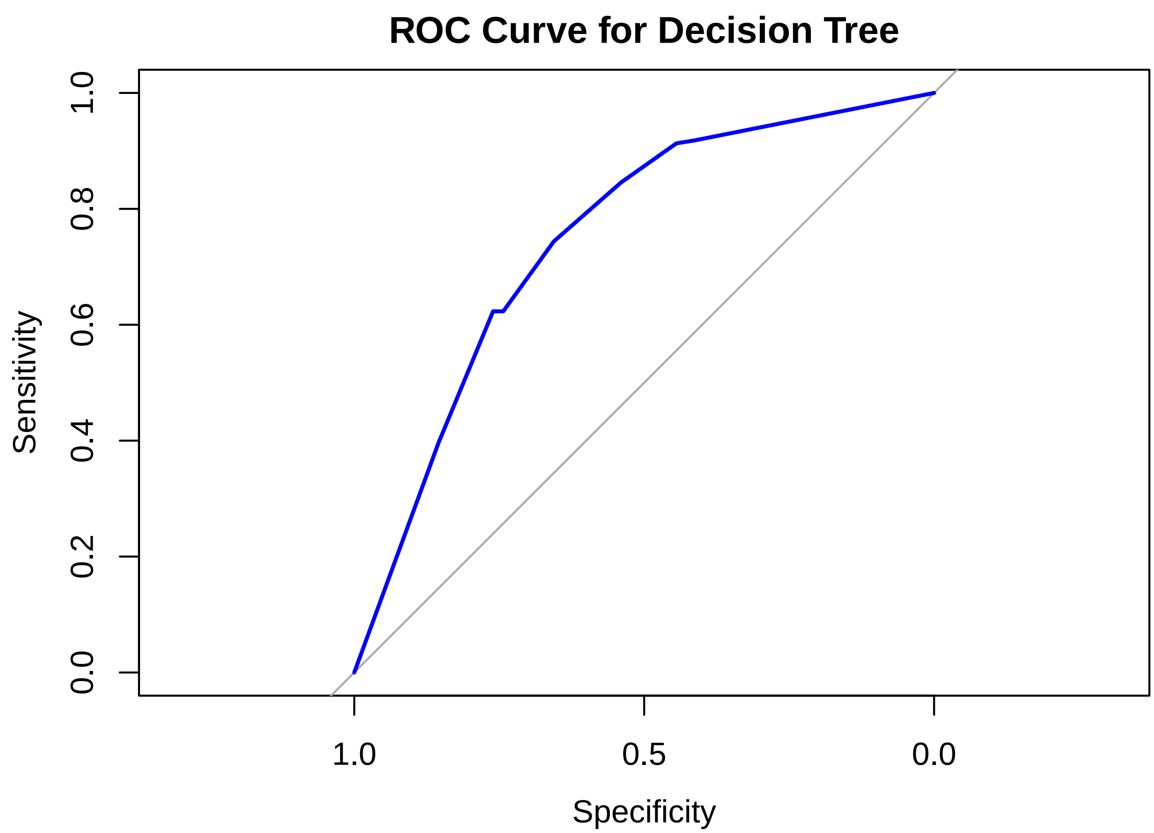
Evaluate the Model on the Testing Data - ROC and AUC.
The AUC of decision tree model(0.748), is lower than the AUC of random forest model(0.77)

```r
# Predict probabilities for the test set (needed for ROC/AUC)
pred_prob_tree <- predict(tree, xyzdata_test, type = "prob")[, 2]

# Check that the adopter column in the test set is a factor or numeric
xyzdata_test$adopter <- as.factor(xyzdata_test$adopter)

# Calculate ROC and AUC
roc_curve <- roc(xyzdata_test$adopter, pred_prob_tree)
```

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

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

```r
# Plot ROC curve
plot(roc_curve, col = "blue", main = "ROC Curve for Decision Tree")
```




```r
# Calculate AUC value
auc_value <- auc(roc_curve)
print(paste("AUC Value: ", auc_value))
```

```
## [1] "AUC Value:  0.74824013126239"
```

Evaluate the Model on the Testing Data - Confusion Matrix.

```r
# Convert predicted probabilities to class predictions using a threshold of 0.3
pred_tree_class <- ifelse(pred_prob_tree > 0.6, "1", "0")

# Convert to factors to match the test set 'adopter' column
pred_tree_class <- as.factor(pred_tree_class)

# Print confusion matrix
confusionMatrix(data = pred_tree_class,
                reference = xyzdata_test$adopter,
                mode = "prec_recall",
                positive = '1')
```

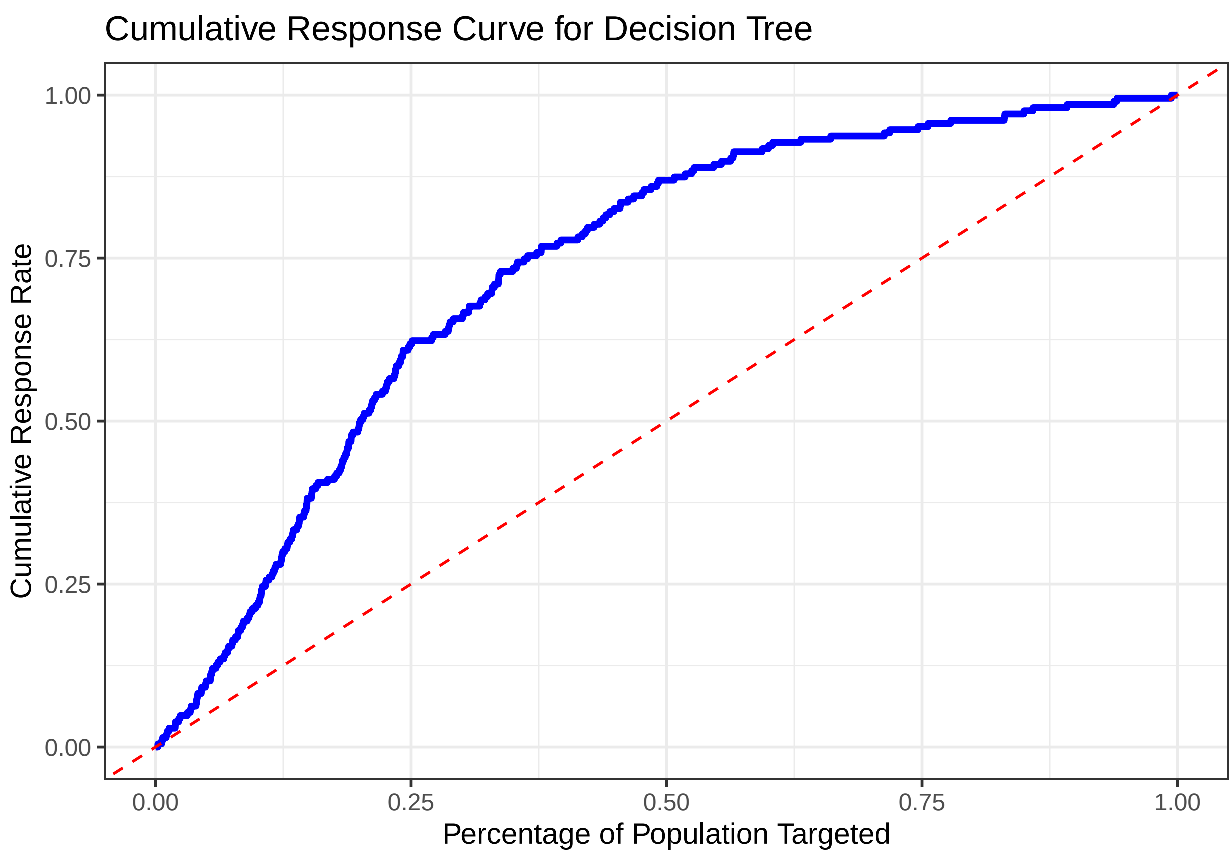
```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction    0    1
##          0 4476   78
##          1 1548  129
##
##                Accuracy : 0.739
##                  95% CI : (0.728, 0.7499)
##     No Information Rate : 0.9668
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0827
##
##  Mcnemar's Test P-Value : <2e-16
##
##               Precision : 0.07692
##                  Recall : 0.62319
##                      F1 : 0.13694
##              Prevalence : 0.03322
##          Detection Rate : 0.02070
##    Detection Prevalence : 0.26914
##       Balanced Accuracy : 0.68311
##
##        'Positive' Class : 1
##
```

Evaluate the Model on the Testing Data - Cumulative Response Curve

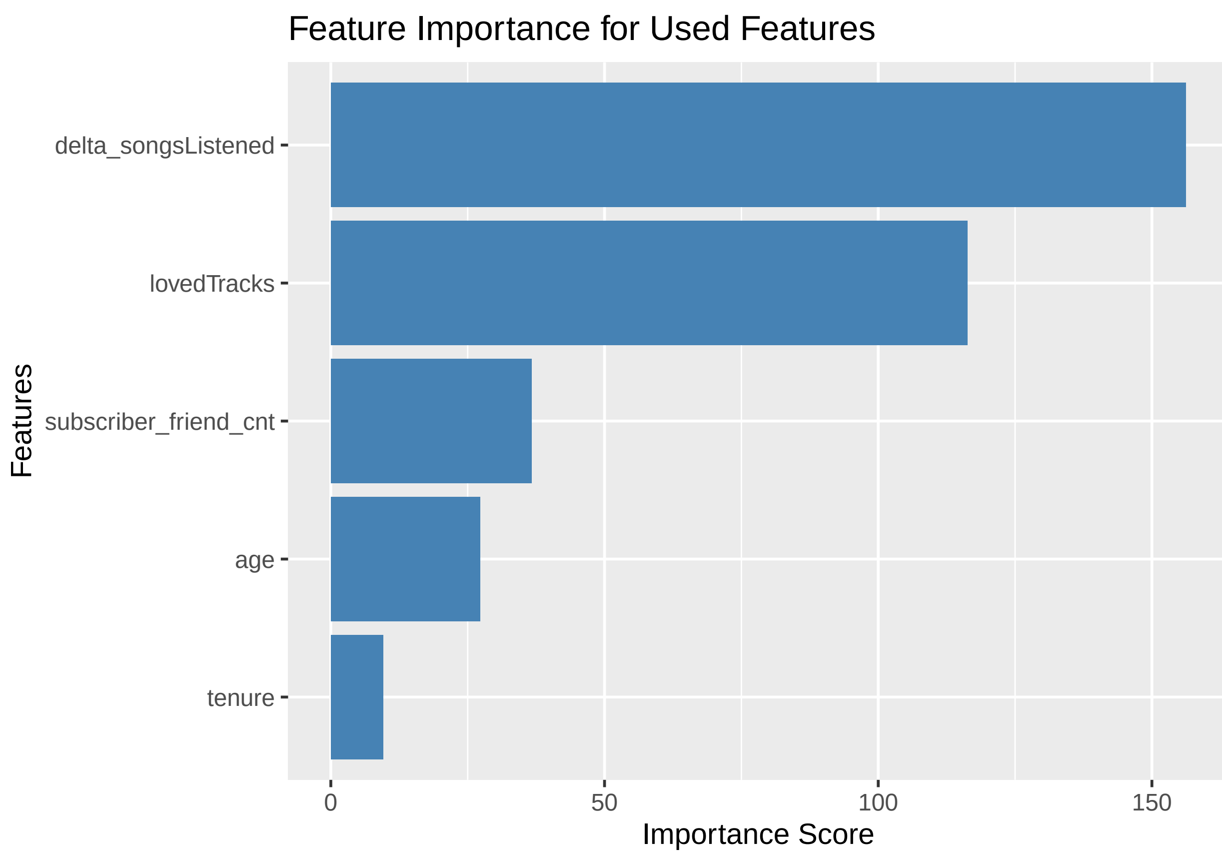
```
# Create a new dataframe for cumulative response calculations
xyz_test_cr <- xyzdata_test %>%
  mutate(prob = pred_prob_tree) %>%  # Use the predicted probabilities from your decision tree
  arrange(desc(prob)) %>%  # Arrange by predicted probabilities in descending order
  mutate(adopter_1= ifelse(adopter == "1", 1, 0)) %>%  # Create a column for actual positive responses
  # Calculate cumulative response curve values
  mutate(y = cumsum(adopter_1) / sum(adopter_1),  # Cumulative response rate
         x = row_number() / n())  # Percentage of population targeted

# Plot the cumulative response curve
ggplot(data = xyz_test_cr, aes(x = x, y = y)) +
  geom_line(color = "blue", size = 1.2) +  # Cumulative response curve
  labs(title = "Cumulative Response Curve for Decision Tree",
       x = "Percentage of Population Targeted",
       y = "Cumulative Response Rate") +
  theme_bw() +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed", color = "red")  # Add baseline for random
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



Feature Importance

```
# We want to check what features is the model using to make predictions

# Extract the names of the features used in the splits (non-leaf nodes)
used_features <- unique(tree$frame$var[tree$frame$var != "<leaf>"])

# Subset the variable importance to only include used features
importance_used <- tree$variable.importance[used_features]

# Convert to a data frame for easier plotting
importance_df_used <- as.data.frame(importance_used)
colnames(importance_df_used) <- c("Importance")

# Add feature names as a separate column
importance_df_used$features <- rownames(importance_df_used)

ggplot(importance_df_used, aes(x = reorder(features, Importance), y = Importance)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  labs(title = "Feature Importance for Used Features", x = "Features", y = "Importance Score")
```


Feature Importance for Used Features
Features
delta_songsListened
lovedTracks
subscriber_friend_cnt
age
tenure
0
50
100
150
Importance Score