15.072 Project Code

2023-11-29

Preprocess Data

```r
library(readr)
library(dplyr)
library(caret)
library(pROC)
library(rpart)
library(randomForest)
library(xgboost)
library(glmnet)
```

```r
# Import the dataset
data <- read_csv("final.table.csv")

data$year <- as.factor(data$year)
data$month <- as.factor(data$month)
data$dominant_topic <- as.factor(data$dominant_topic)
data$OneHundred.Million <- data$`100.Million`
data$`100.Million` <- NULL

# Remove rows with na
data <- na.omit(data)

# Create subsets for EDM and R&B music
edm <- data %>%
  filter(playlist_genre == "edm")
rb <- data %>%
  filter(playlist_genre == "r&b")

# Remove columns
cols_remove <- c("track_name", "track_artist", "lyrics", "track_album_release_date",
                 "playlist_genre", "language", "year")
edm <- edm[, !(colnames(edm) %in% cols_remove)]
rb <- rb[, !(colnames(rb) %in% cols_remove)]
```

```r
# Define track popularity as above third quartile
third_quartile_edm <- quantile(edm$track_popularity, 0.75)
edm <- edm %>%
  mutate(Popular = ifelse(track_popularity > third_quartile_edm, 1, 0)) %>%
  select(-track_popularity)
edm$Popular <- as.factor(edm$Popular)

third_quartile_rb <- quantile(rb$track_popularity, 0.75)
rb <- rb %>%
  mutate(Popular = ifelse(track_popularity > third_quartile_rb, 1, 0)) %>%
```

```
  select(-track_popularity)
rb$Popular <- as.factor(rb$Popular)
```

```
# One-hot encode the data
cols_categorical <- c("month", "day_of_week",
                      "dominant_topic", "SentimentClass_Bing", "SentimentClass_nrc")

encoded_edm <- model.matrix(~ . + 0, data = data.frame(edm[, cols_categorical]))
encoded_rb <- model.matrix(~ . + 0, data = data.frame(rb[, cols_categorical]))

# Combine with the original data frame
edm_onehot <- cbind(edm, encoded_edm)
rb_onehot <- cbind(rb, encoded_rb)

# Remove the original categorical columns
edm_onehot <- edm_onehot[, !names(edm_onehot) %in% cols_categorical]
rb_onehot <- rb_onehot[, !names(rb_onehot) %in% cols_categorical]
```

```
# Create dataframe for results
results_edm <- data.frame(
  Model = character(),
  Baseline_Accuracy = numeric(),
  Model_Accuracy = numeric(),
  AUC_Value = numeric()
)

results_rb <- data.frame(
  Model = character(),
  Baseline_Accuracy = numeric(),
  Model_Accuracy = numeric(),
  AUC_Value = numeric()
)
```

Logistic Regression

EDM

Identify significant columns

```
# Fit logistic regression model
logistic_model <- glm(Popular ~ . - speechiness - One.Billion - key - instrumentalness
                      - mode - SentimentClass_nrc, data = edm, family = "binomial")

# Display summary to see p-values
summary(logistic_model)
```

```
##
## Call:
## glm(formula = Popular ~ . - speechiness - One.Billion - key -
##     instrumentalness - mode - SentimentClass_nrc, family = "binomial",
##     data = edm)
##
## Coefficients:
##                                Estimate Std. Error z value Pr(>|z|)
## (Intercept)                  -3.521e+00  2.121e+00  -1.660 0.096905 .
```

```
## danceability                -2.477e-01  1.238e+00  -0.200 0.841433
## energy                       1.876e+00  1.614e+00   1.163 0.245025
## loudness                    -1.543e-01  9.925e-02  -1.554 0.120082
## acousticness                 1.806e+00  1.107e+00   1.632 0.102676
## liveness                     7.524e-01  7.288e-01   1.032 0.301874
## valence                      4.535e-01  6.951e-01   0.652 0.514123
## tempo                       -4.857e-03  8.280e-03  -0.587 0.557458
## Lead.Streams                -1.129e-10  5.183e-11  -2.179 0.029354 *
## Feats                        1.032e-10  9.497e-11   1.086 0.277436
## Tracks                      -1.213e-03  6.025e-04  -2.014 0.044043 *
## month2                      -8.606e-01  8.065e-01  -1.067 0.285937
## month3                      -2.440e-01  6.276e-01  -0.389 0.697440
## month4                       5.090e-01  6.296e-01   0.808 0.418837
## month5                       4.679e-01  5.875e-01   0.796 0.425799
## month6                      -6.550e-01  6.464e-01  -1.013 0.310876
## month7                      -7.002e-02  6.964e-01  -0.101 0.919907
## month8                       1.039e-01  6.116e-01   0.170 0.865061
## month9                       4.563e-01  6.230e-01   0.732 0.463880
## month10                      4.493e-01  5.899e-01   0.762 0.446201
## month11                      1.542e-01  6.137e-01   0.251 0.801592
## month12                      9.358e-02  6.370e-01   0.147 0.883216
## day_of_weekMonday           -1.188e-01  5.183e-01  -0.229 0.818703
## day_of_weekSaturday         -1.170e-01  9.440e-01  -0.124 0.901374
## day_of_weekSunday            4.522e-02  9.100e-01   0.050 0.960367
## day_of_weekThursday          1.665e+00  4.587e-01   3.629 0.000284 ***
## day_of_weekTuesday           5.531e-01  4.262e-01   1.298 0.194322
## day_of_weekWednesday        -6.813e-01  8.783e-01  -0.776 0.437932
## dominant_topic2              2.648e-01  3.596e-01   0.736 0.461579
## dominant_topic3              7.056e-01  4.768e-01   1.480 0.138944
## dominant_topic4              9.921e-02  4.362e-01   0.227 0.820078
## dominant_topic5             -1.666e+01  8.525e+02  -0.020 0.984412
## SentimentClass_BingNeutral   5.912e-02  3.855e-01   0.153 0.878105
## SentimentClass_BingPositive  5.168e-01  4.090e-01   1.264 0.206346
## OneHundred.Million           5.806e-02  2.227e-02   2.608 0.009117 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 481.41  on 431  degrees of freedom
## Residual deviance: 415.29  on 397  degrees of freedom
## AIC: 485.29
##
## Number of Fisher Scoring iterations: 16
```

```r
# Split the data into training and testing sets
set.seed(123)  # for reproducibility
train_indices <- createDataPartition(edm$Popular, p = 0.7, list = FALSE)
train_data <- edm[train_indices, ]
test_data <- edm[-train_indices, ]

# Fit logistic regression model
logistic_model <- glm(Popular ~ . - speechiness - One.Billion - key - instrumentalness
                      - mode, data = train_data, family = "binomial")
```

```r
predictions <- predict(logistic_model, newdata = test_data, type = "response")

# Make predictions on the test set
predictions <- predict(logistic_model, newdata = test_data, type = "response")

# Convert 'popular' to a binary factor for model evaluation
test_data$Popular <- as.factor(test_data$Popular)

# Calculate baseline
baseline <- (table(train_data$Popular)[1]) / (table(train_data$Popular)[1] + table(train_data$Popular)[

# Calculate accuracy
predicted_class <- factor(ifelse(predictions > 0.5, "1", "0"), levels = levels(test_data$Popular))
confusion_matrix <- confusionMatrix(predicted_class, test_data$Popular)
accuracy <- confusion_matrix$overall["Accuracy"]

# Calculate AUC
roc_curve <- roc(test_data$Popular, predictions)
```

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

## Setting direction: controls < cases
```

```r
auc <- as.numeric(auc(roc_curve))

results_edm <<- rbind(results_edm, data.frame(
    Model = "Logistic Regression",
    AUC_Value = round(auc, 4),
    Baseline_Accuracy = round(baseline, 4),
    Model_Accuracy = round(accuracy, 4)
  ))

# Print results
cat("Baseline:", round(baseline, 4), "\n")
```

```
## Baseline: 0.7533
```

```r
cat("Accuracy:", round(accuracy, 4), "\n")
```

```
## Accuracy: 0.7031
```

```r
cat("AUC:", round(auc, 4), "\n")
```

```
## AUC: 0.6485
```

R&B

Identify significant columns

```r
# Fit logistic regression model
logistic_model <- glm(Popular ~ . - speechiness - energy - SentimentClass_nrc
                      - valence - dominant_topic - key - instrumentalness - tempo, data = rb, family =

# Display summary to see p-values
summary(logistic_model)
```

```
##
```

```
## Call:
## glm(formula = Popular ~ . - speechiness - energy - SentimentClass_nrc -
##     valence - dominant_topic - key - instrumentalness - tempo,
##     family = "binomial", data = rb)
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                 -1.872e+00  7.942e-01  -2.357  0.01844 *
## danceability                 1.969e+00  7.764e-01   2.536  0.01122 *
## loudness                     9.278e-02  4.402e-02   2.108  0.03506 *
## mode                         4.643e-01  2.064e-01   2.250  0.02444 *
## acousticness                 8.867e-01  4.373e-01   2.028  0.04259 *
## liveness                    -1.842e+00  1.012e+00  -1.820  0.06873 .
## Lead.Streams                 1.655e-10  9.998e-11   1.655  0.09788 .
## Feats                       -1.840e-10  5.287e-11  -3.481  0.00050 ***
## Tracks                      -2.679e-03  1.000e-03  -2.678  0.00742 **
## One.Billion                 -1.964e-01  1.492e-01  -1.316  0.18824
## month2                       1.028e-02  5.800e-01   0.018  0.98586
## month3                       4.518e-01  4.893e-01   0.923  0.35576
## month4                       2.084e-01  4.950e-01   0.421  0.67375
## month5                       1.981e-01  5.209e-01   0.380  0.70377
## month6                       6.136e-02  4.604e-01   0.133  0.89398
## month7                      -9.196e-04  5.924e-01  -0.002  0.99876
## month8                       2.982e-01  4.526e-01   0.659  0.50994
## month9                       9.138e-01  4.788e-01   1.909  0.05629 .
## month10                      2.561e-01  4.954e-01   0.517  0.60518
## month11                      2.810e-01  4.052e-01   0.693  0.48805
## month12                      1.409e+00  4.528e-01   3.111  0.00186 **
## day_of_weekMonday           -1.097e+00  4.806e-01  -2.282  0.02249 *
## day_of_weekSaturday          6.798e-01  5.165e-01   1.316  0.18812
## day_of_weekSunday           -6.771e-01  5.294e-01  -1.279  0.20089
## day_of_weekThursday         -1.213e-01  3.638e-01  -0.334  0.73875
## day_of_weekTuesday          -5.238e-01  3.592e-01  -1.458  0.14485
## day_of_weekWednesday        -1.377e+00  5.210e-01  -2.642  0.00824 **
## SentimentClass_BingNeutral  -2.855e-01  2.654e-01  -1.076  0.28205
## SentimentClass_BingPositive -6.127e-01  3.306e-01  -1.853  0.06382 .
## OneHundred.Million          -2.449e-02  3.475e-02  -0.705  0.48105
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 749.61  on 698  degrees of freedom
## Residual deviance: 642.94  on 669  degrees of freedom
## AIC: 702.94
##
## Number of Fisher Scoring iterations: 5
```

Run Logistic Regression

```r
# Split the data into training and testing sets
set.seed(123)  # for reproducibility
train_indices <- createDataPartition(rb$Popular, p = 0.7, list = FALSE)
train_data <- rb[train_indices, ]
test_data <- rb[-train_indices, ]
```

```r
# Fit logistic regression model
logistic_model <- glm(Popular ~ . - speechiness - energy - SentimentClass_nrc
                      - valence - dominant_topic - key - instrumentalness - tempo
                      - OneHundred.Million , data = train_data, family = "binomial")
# NOTE: Accuracy is best (0.8134) when no - tempo and - 100.Million

predictions <- predict(logistic_model, newdata = test_data, type = "response")

# Make predictions on the test set
predictions <- predict(logistic_model, newdata = test_data, type = "response")

# Convert 'popular' to a binary factor for model evaluation
test_data$Popular <- as.factor(test_data$Popular)

# Calculate baseline
baseline <- (table(train_data$Popular)[1]) / (table(train_data$Popular)[1] + table(train_data$Popular)[

# Calculate accuracy
predicted_class <- factor(ifelse(predictions > 0.5, "1", "0"), levels = levels(test_data$Popular))
confusion_matrix <- confusionMatrix(predicted_class, test_data$Popular)
accuracy <- confusion_matrix$overall["Accuracy"]

# Calculate AUC
roc_curve <- roc(test_data$Popular, predictions)
```

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

```r
auc <- as.numeric(auc(roc_curve))

results_rb <<- rbind(results_rb, data.frame(
    Model = "Logistic Regression",
    AUC_Value = round(auc, 4),
    Baseline_Accuracy = round(baseline, 4),
    Model_Accuracy = round(accuracy, 4)
  ))

# Print results
cat("Baseline:", round(baseline, 4), "\n")
```

```
## Baseline: 0.7714
```

```r
cat("Accuracy:", round(accuracy, 4), "\n")
```

```
## Accuracy: 0.799
```

```r
cat("AUC:", round(auc, 4), "\n")
```

```
## AUC: 0.7199
```

CART

EDM

Find significant columns

```r
# Split the data into training and testing sets
set.seed(123)  # for reproducibility
train_indices <- createDataPartition(edm$Popular, p = 0.7, list = FALSE)
train_data <- edm[train_indices, ]
test_data <- edm[-train_indices, ]

# Fit CART model
cart_model <- rpart(
  Popular ~ . - speechiness,
  data = train_data,
  method = "class",
  control = rpart.control(
    cp = 0.01,
    minsplit = 10,
    minbucket = 5,
    maxdepth = 5
  )
)

# Make predictions on the test set
predictions <- predict(cart_model, newdata = test_data, type = "class")

# Convert 'Popular' to a binary factor for model evaluation
test_data$Popular <- as.factor(test_data$Popular)

# Calculate baseline
table <- table(train_data$Popular)[1] + table(train_data$Popular)[2]
baseline <- (table(train_data$Popular)[1]) / (table(train_data$Popular)[1] + table(train_data$Popular)[

# Calculate accuracy
confusion_matrix <- confusionMatrix(predictions, test_data$Popular)
accuracy <- confusion_matrix$overall["Accuracy"]

# Calculate AUC
roc_curve <- roc(test_data$Popular, as.numeric(predictions))
```

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

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

```r
auc <- as.numeric(auc(roc_curve))

# Return a data frame with the results
results_edm <<- rbind(results_edm, data.frame(
    Model = "CART",
    AUC_Value = round(auc, 4),
    Baseline_Accuracy = round(baseline, 4),
    Model_Accuracy = round(accuracy, 4)
  ))


# Print results
cat("Baseline:", round(baseline, 4), "\n")
```

```
## Baseline: 0.7533
```

```r
cat("Accuracy:", round(accuracy, 4), "\n")
```

```
## Accuracy: 0.7109
```

```r
cat("AUC:", round(auc, 4), "\n")
```

```
## AUC: 0.5788
```

Build model

```r
# Fit a CART model
cart_model <- rpart(
  Popular ~ . - speechiness,
  data = train_data,
  method = "class",
  control = rpart.control(
    cp = 0.01,
    minsplit = 10,
    minbucket = 5,
    maxdepth = 5
  )
)

# Display the tree
print(cart_model)
```

```
## n= 304
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 304 75 0 (0.7532895 0.2467105)
##    2) day_of_week=Friday,Monday,Saturday,Sunday,Tuesday,Wednesday 281 60 0 (0.7864769 0.2135231)
##      4) acousticness< 0.02335 142 20 0 (0.8591549 0.1408451)
##        8) Lead.Streams< 2.537415e+10 137 17 0 (0.8759124 0.1240876) *
##        9) Lead.Streams>=2.537415e+10 5  2 1 (0.4000000 0.6000000) *
##      5) acousticness>=0.02335 139 40 0 (0.7122302 0.2877698)
##       10) loudness>=-4.0505 53  8 0 (0.8490566 0.1509434) *
##       11) loudness< -4.0505 86 32 0 (0.6279070 0.3720930)
##         22) liveness< 0.1435 46 11 0 (0.7608696 0.2391304)
##           44) acousticness>=0.04135 37  5 0 (0.8648649 0.1351351) *
##           45) acousticness< 0.04135 9  3 1 (0.3333333 0.6666667) *
##         23) liveness>=0.1435 40 19 1 (0.4750000 0.5250000)
##           46) dominant_topic=4,5 7  0 0 (1.0000000 0.0000000) *
##           47) dominant_topic=1,2,3 33 12 1 (0.3636364 0.6363636) *
##    3) day_of_week=Thursday 23  8 1 (0.3478261 0.6521739)
##      6) month=1,3,4 6  1 0 (0.8333333 0.1666667) *
##      7) month=2,5,6,7,9,10,11,12 17  3 1 (0.1764706 0.8235294) *
```

```r
var_importance <- varImp(cart_model)

print(var_importance)
```

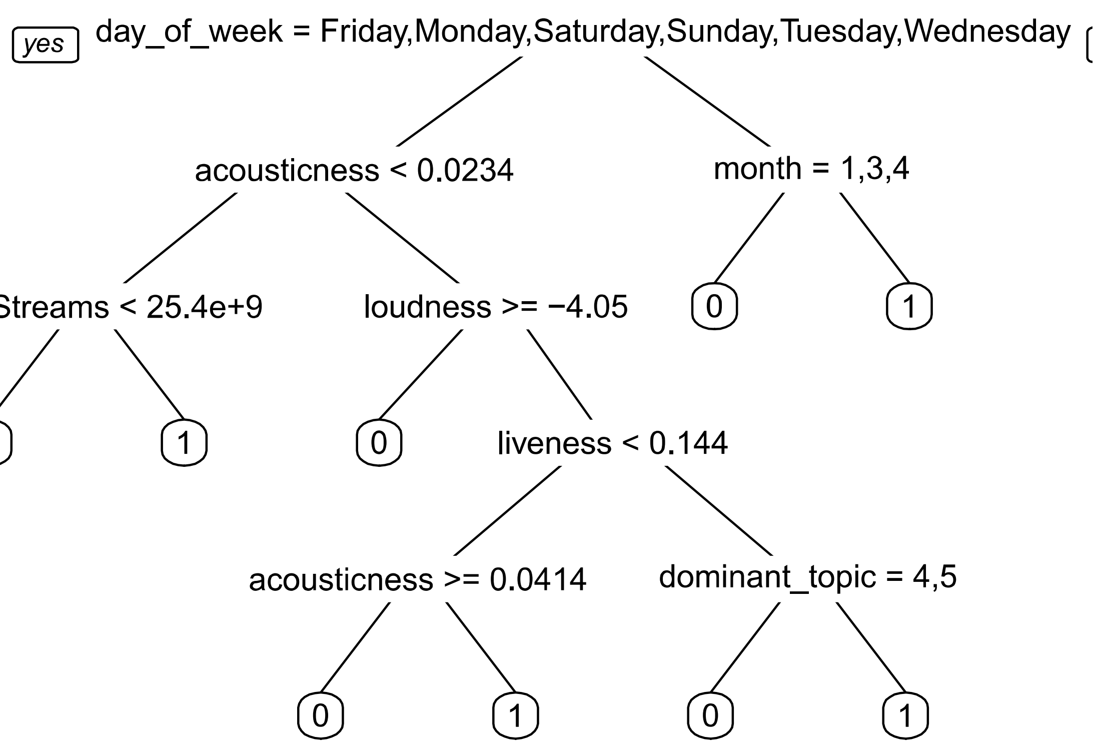
```
##                      Overall
## acousticness       12.741628
## danceability        4.093406
```

```
## day_of_week          10.759986
## dominant_topic        8.879846
## Feats                 5.404928
## instrumentalness      2.809247
## Lead.Streams          4.149370
## liveness             10.172795
## loudness              5.366435
## month                12.580465
## OneHundred.Million    4.703006
## SentimentClass_Bing   1.256451
## tempo                11.212689
## Tracks                6.556324
## valence               4.846366
## energy                0.000000
## key                   0.000000
## mode                  0.000000
## One.Billion           0.000000
## SentimentClass_nrc    0.000000
```

```r
# Plot the tree
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 4.3.1
```

```r
prp(cart_model, digits=3, split.font=1, varlen = 0, faclen = 0)
```



R&B

Build model

```
# Split the data into training and testing sets
set.seed(123)  # for reproducibility
train_indices <- createDataPartition(rb$Popular, p = 0.7, list = FALSE)
train_data <- rb[train_indices, ]
test_data <- rb[-train_indices, ]

# Fit CART model
cart_model <- rpart(Popular ~ . - speechiness - energy - danceability
                    -loudness, data = train_data, method = "class")

# Make predictions on the test set
predictions <- predict(cart_model, newdata = test_data, type = "class")

# Convert 'Popular' to a binary factor for model evaluation
test_data$Popular <- as.factor(test_data$Popular)

# Calculate baseline
table <- table(train_data$Popular)[1] + table(train_data$Popular)[2]
baseline <- (table(train_data$Popular)[1]) / (table(train_data$Popular)[1] + table(train_data$Popular)[

# Calculate accuracy
confusion_matrix <- confusionMatrix(predictions, test_data$Popular)
accuracy <- confusion_matrix$overall["Accuracy"]

# Calculate AUC
roc_curve <- roc(test_data$Popular, as.numeric(predictions))
```

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

```
auc <- as.numeric(auc(roc_curve))

# Return a data frame with the results
results_rb <<- rbind(results_rb, data.frame(
    Model = "CART",
    AUC_Value = round(auc, 4),
    Baseline_Accuracy = round(baseline, 4),
    Model_Accuracy = round(accuracy, 4)
  ))

# Print results
cat("Baseline:", round(baseline, 4), "\n")
```

```
## Baseline: 0.7714
```

```
cat("Accuracy:", round(accuracy, 4), "\n")
```

```
## Accuracy: 0.7703
```

```
cat("AUC:", round(auc, 4), "\n")
```

```
## AUC: 0.6631
```

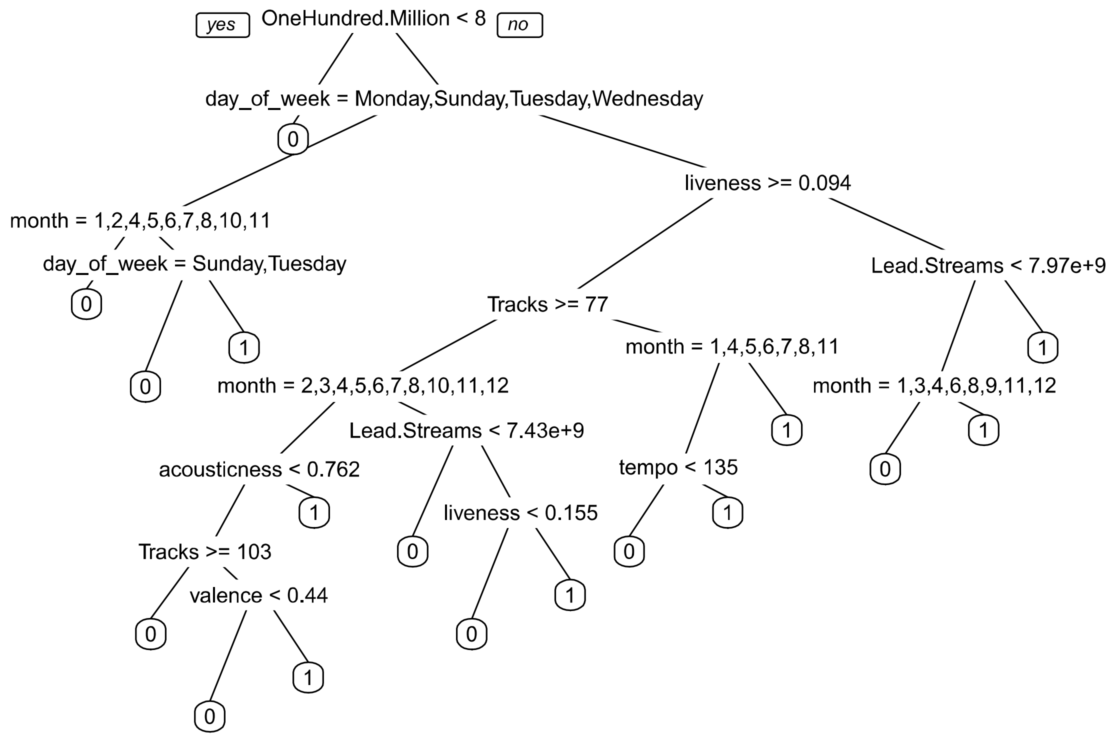
Find significant columns

```r
# Fit a CART model
cart_model <- rpart(Popular ~ . - speechiness - energy - danceability -loudness, data = rb, method = "c

# Display the tree
print(cart_model)
```

```
## n= 699
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 699 159 0 (0.77253219 0.22746781)
##     2) OneHundred.Million< 7.5 262  26 0 (0.90076336 0.09923664) *
##     3) OneHundred.Million>=7.5 437 133 0 (0.69565217 0.30434783)
##       6) day_of_week=Monday,Sunday,Tuesday,Wednesday 149  23 0 (0.84563758 0.15436242)
##        12) month=1,2,4,5,6,7,8,10,11 112  10 0 (0.91071429 0.08928571) *
##        13) month=3,9,12 37  13 0 (0.64864865 0.35135135)
##          26) day_of_week=Sunday,Tuesday 23   4 0 (0.82608696 0.17391304) *
##          27) day_of_week=Monday,Wednesday 14   5 1 (0.35714286 0.64285714) *
##       7) day_of_week=Friday,Saturday,Thursday 288 110 0 (0.61805556 0.38194444)
##        14) liveness>=0.09395 241  81 0 (0.66390041 0.33609959)
##          28) Tracks>=77 187  52 0 (0.72192513 0.27807487)
##            56) month=2,3,4,5,6,7,8,10,11,12 158  38 0 (0.75949367 0.24050633)
##             112) acousticness< 0.7615 143  30 0 (0.79020979 0.20979021)
##               224) Tracks>=102.5 116  20 0 (0.82758621 0.17241379) *
##               225) Tracks< 102.5 27  10 0 (0.62962963 0.37037037)
##                 450) valence< 0.4395 19   4 0 (0.78947368 0.21052632) *
##                 451) valence>=0.4395 8   2 1 (0.25000000 0.75000000) *
##             113) acousticness>=0.7615 15   7 1 (0.46666667 0.53333333) *
##            57) month=1,9 29  14 0 (0.51724138 0.48275862)
##             114) Lead.Streams< 7.429889e+09 8   1 0 (0.87500000 0.12500000) *
##             115) Lead.Streams>=7.429889e+09 21   8 1 (0.38095238 0.61904762)
##               230) liveness< 0.1545 14   6 0 (0.57142857 0.42857143) *
##               231) liveness>=0.1545 7   0 1 (0.00000000 1.00000000) *
##          29) Tracks< 77 54  25 1 (0.46296296 0.53703704)
##            58) month=1,4,5,6,7,8,11 40  15 0 (0.62500000 0.37500000)
##             116) tempo< 134.9665 31   8 0 (0.74193548 0.25806452) *
##             117) tempo>=134.9665 9   2 1 (0.22222222 0.77777778) *
##            59) month=3,9,10,12 14   0 1 (0.00000000 1.00000000) *
##        15) liveness< 0.09395 47  18 1 (0.38297872 0.61702128)
##          30) Lead.Streams< 7.968422e+09 29  12 0 (0.58620690 0.41379310)
##            60) month=1,3,4,6,8,9,11,12 22   6 0 (0.72727273 0.27272727) *
##            61) month=2,5,10 7   1 1 (0.14285714 0.85714286) *
##          31) Lead.Streams>=7.968422e+09 18   1 1 (0.05555556 0.94444444) *
```

```r
# Plot the tree
library(rpart.plot)
prp(cart_model, digits=3, split.font=1, varlen = 0, faclen = 0)
```



Random Forest

EDM

```
# Set seed for reproducibility
set.seed(123)

# Split the data into training and testing sets
train_indices <- createDataPartition(edm$Popular, p = 0.7, list = FALSE)
train_data <- edm[train_indices, ]
test_data <- edm[-train_indices, ]

# Fit Random Forest model
rf_model <- randomForest(Popular ~ ., data = train_data)

# Make predictions on the test set
predictions <- predict(rf_model, newdata = test_data)

# Convert 'Popular' to a binary factor for model evaluation
test_data$Popular <- as.factor(test_data$Popular)

# Calculate baseline
table <- table(train_data$Popular)[1] + table(train_data$Popular)[2]
baseline <- (table(train_data$Popular)[1]) / (table(train_data$Popular)[1] + table(train_data$Popular)[
```

```r
# Calculate accuracy
confusion_matrix <- confusionMatrix(predictions, test_data$Popular)
accuracy <- confusion_matrix$overall["Accuracy"]

# Calculate AUC
roc_curve <- roc(test_data$Popular, as.numeric(predictions))
```

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

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

```r
auc <- as.numeric(auc(roc_curve))

# Return a data frame with the results
results_edm <- rbind(results_edm, data.frame(
    Model = "Random Forest",
    AUC_Value = round(auc, 4),
    Baseline_Accuracy = round(baseline, 4),
    Model_Accuracy = round(accuracy, 4)
  ))

# Print results
cat("Baseline:", round(baseline, 4), "\n")
```

```
## Baseline: 0.7533
```

```r
cat("Accuracy:", round(accuracy, 4), "\n")
```

```
## Accuracy: 0.7812
```

```r
cat("AUC:", round(auc, 4), "\n")
```

```
## AUC: 0.5703
```

R&B

```r
# Set seed for reproducibility
set.seed(123)

# Split the data into training and testing sets
train_indices <- createDataPartition(rb$Popular, p = 0.7, list = FALSE)
train_data <- rb[train_indices, ]
test_data <- rb[-train_indices, ]

# Fit Random Forest model
rf_model <- randomForest(Popular ~ . - energy - Feats - Tracks, data = train_data)

# Make predictions on the test set
predictions <- predict(rf_model, newdata = test_data)

# Convert 'Popular' to a binary factor for model evaluation
test_data$Popular <- as.factor(test_data$Popular)

# Calculate baseline
table <- table(train_data$Popular)[1] + table(train_data$Popular)[2]
baseline <- (table(train_data$Popular)[1]) / (table(train_data$Popular)[1] + table(train_data$Popular)[
```

```r
# Calculate accuracy
confusion_matrix <- confusionMatrix(predictions, test_data$Popular)
accuracy <- confusion_matrix$overall["Accuracy"]

# Calculate AUC
roc_curve <- roc(test_data$Popular, as.numeric(predictions))
```

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

## Setting direction: controls < cases
```

```r
auc <- as.numeric(auc(roc_curve))

# Return a data frame with the results
results_rb <- rbind(results_rb, data.frame(
    Model = "Random Forest",
    AUC_Value = round(auc, 4),
    Baseline_Accuracy = round(baseline, 4),
    Model_Accuracy = round(accuracy, 4)
  ))

# Print results
cat("Baseline:", round(baseline, 4), "\n")
```

```
## Baseline: 0.7714
```

```r
cat("Accuracy:", round(accuracy, 4), "\n")
```

```
## Accuracy: 0.8134
```

```r
cat("AUC:", round(auc, 4), "\n")
```

```
## AUC: 0.6078
```

```r
# Extract feature importance from the trained Random Forest model
importance <- importance(rf_model)

importance
```

```
##                       MeanDecreaseGini
## danceability                 13.455013
## key                           6.792569
## loudness                     14.910450
## mode                          2.011203
## speechiness                  12.100603
## acousticness                 13.576957
## instrumentalness              6.846799
## liveness                     12.278837
## valence                      11.525530
## tempo                        11.648526
## Lead.Streams                 12.148155
## One.Billion                   5.166577
## month                        17.994713
## day_of_week                   5.524048
## dominant_topic                6.751823
## SentimentClass_Bing           2.940734
## SentimentClass_nrc            4.068813
```

```
## OneHundred.Million         12.023621
```

```r
colnames(train_data)
```

```
##  [1] "danceability"       "energy"             "key"
##  [4] "loudness"           "mode"               "speechiness"
##  [7] "acousticness"       "instrumentalness"   "liveness"
## [10] "valence"            "tempo"              "Lead.Streams"
## [13] "Feats"              "Tracks"             "One.Billion"
## [16] "month"              "day_of_week"        "dominant_topic"
## [19] "SentimentClass_Bing" "SentimentClass_nrc" "OneHundred.Million"
## [22] "Popular"
```

XGBoost

EDM

```r
set.seed(123)

# Split the data into training and testing sets
train_indices <- createDataPartition(edm$Popular, p = 0.7, list = FALSE)
train_data <- edm[train_indices, ]
test_data <- edm[-train_indices, ]

X.train = train_data%>%select(-Popular) #fix
X.test = test_data%>%select(-Popular)
y.train = train_data$Popular
y.test = test_data%>%select(Popular)
X.train <- model.matrix(~.-1,data = X.train)
X.test <- model.matrix(~.-1,data = X.test)

hyper_grid <- expand.grid(
  nrounds = c(50,  150),
  eta = c(0.01, 0.1),
  max_depth = c(3, 9),
  subsample = c(0.5,  1),
  colsample_bytree = c(0.5,  1),
  gamma = c(0, 0.1, 1, 5),
  min_child_weight = c(1, 2, 10)
)

train_control <- trainControl(method = "cv", number = 10, summaryFunction = twoClassSummary, classProbs

suppressWarnings({
xgb_mod <- train(
  x = X.train, y = factor(y.train, levels = c("0", "1"), labels = c("Class0", "Class1")),
  method = "xgbTree",
  trControl = train_control,
  tuneGrid = hyper_grid,
  metric = "LogLoss", verbose = FALSE
)})
```

```
## [13:03:33] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:03:33] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
```

```
## [13:31:30] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:30] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:31] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:31] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
```

```
test_predict_xgb = predict(xgb_mod, X.test)
prediction <- ifelse(test_predict_xgb == "Class0", 0, 1)
levels(prediction) <- levels(test_data$Popular)
prediction <- as.factor(prediction)

# Calculate baseline
table <- table(train_data$Popular)[1] + table(train_data$Popular)[2]
baseline <- (table(train_data$Popular)[1]) / (table(train_data$Popular)[1] + table(train_data$Popular)[

# Calculate accuracy
confusion_matrix <- confusionMatrix(prediction, test_data$Popular)
accuracy <- confusion_matrix$overall["Accuracy"]

# Calculate AUC
roc_curve <- roc(test_data$Popular, as.numeric(test_predict_xgb))
```

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

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

```
auc <- as.numeric(auc(roc_curve))

results_edm <- rbind(results_edm, data.frame(
  Model = "XGBoost",
  AUC_Value = round(auc, 4),
  Baseline_Accuracy = round(baseline, 4),
  Model_Accuracy = round(accuracy, 4)
))

# Print results
cat("Baseline:", round(baseline, 4), "\n")
```

```
## Baseline: 0.7533
```

```
cat("Accuracy:", round(accuracy, 4), "\n")
```

```
## Accuracy: 0.7266
```

```
cat("AUC:", round(auc, 4), "\n")
```

```
## AUC: 0.4794
```

R&B

```
set.seed(123)

# Split the data into training and testing sets
train_indices <- createDataPartition(rb$Popular, p = 0.7, list = FALSE)
train_data <- rb[train_indices, ]
test_data <- rb[-train_indices, ]

X.train = train_data%>%select(-Popular) #fix
```

```
X.test = test_data%>%select(-Popular)
y.train = train_data$Popular
y.test = test_data%>%select(Popular)
X.train <- model.matrix(~.-1,data = X.train)
X.test <- model.matrix(~.-1,data = X.test)

hyper_grid <- expand.grid(
  nrounds = c(50,  150),
  eta = c(0.01, 0.1),
  max_depth = c(3, 9),
  subsample = c(0.5,  1),
  colsample_bytree = c(0.5,  1),
  gamma = c(0, 0.1, 1, 5),
  min_child_weight = c(1, 2, 10)
)

train_control <- trainControl(method = "cv", number = 10, summaryFunction = twoClassSummary, classProbs

suppressWarnings({
xgb_mod <- train(
  x = X.train, y = factor(y.train, levels = c("0", "1"), labels = c("Class0", "Class1")),
  method = "xgbTree",
  trControl = train_control,
  tuneGrid = hyper_grid,
  metric = "LogLoss", verbose = FALSE
)})
```

```
## [13:31:33] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:33] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:34] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:34] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:35] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:35] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:36] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:36] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:36] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:36] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:37] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:37] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:38] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:38] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:38] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:38] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:39] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:39] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:39] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:39] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:40] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:40] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:41] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:41] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:42] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:42] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [13:31:42] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
```

```
## [14:07:21] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:30] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:30] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:31] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:31] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:33] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:33] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:34] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:34] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:35] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:35] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:36] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:36] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:39] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:39] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:41] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:41] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:43] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:43] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:44] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:44] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:44] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
## [14:07:44] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` inste
```

```r
test_predict_xgb = predict(xgb_mod, X.test)
prediction <- ifelse(test_predict_xgb == "Class0", 0, 1)
levels(prediction) <- levels(test_data$Popular)
prediction <- as.factor(prediction)

# Calculate baseline
table <- table(train_data$Popular)[1] + table(train_data$Popular)[2]
baseline <- (table(train_data$Popular)[1]) / (table(train_data$Popular)[1] + table(train_data$Popular)[

# Calculate accuracy
confusion_matrix <- confusionMatrix(prediction, test_data$Popular)
accuracy <- confusion_matrix$overall["Accuracy"]

# Calculate AUC
roc_curve <- roc(test_data$Popular, as.numeric(test_predict_xgb))
```

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

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

```r
auc <- as.numeric(auc(roc_curve))

results_rb <- rbind(results_rb, data.frame(
  Model = "XGBoost",
  AUC_Value = round(auc, 4),
  Baseline_Accuracy = round(baseline, 4),
  Model_Accuracy = round(accuracy, 4)
))

# Print results
cat("Baseline:", round(baseline, 4), "\n")
```

```
## Baseline: 0.7714
```

```r
cat("Accuracy:", round(accuracy, 4), "\n")
```

```
## Accuracy: 0.7799
```

```r
cat("AUC:", round(auc, 4), "\n")
```

```
## AUC: 0.5559
```

Lasso and Ridge

```r
# Split the data into training and testing sets
set.seed(123)  # for reproducibility
train_indices <- createDataPartition(edm$Popular, p = 0.7, list = FALSE)
train_data <- edm[train_indices, ]
test_data <- edm[-train_indices, ]

# Fit logistic regression model with Lasso regularization
lasso_model <- cv.glmnet(
  x = model.matrix(Popular ~ . - speechiness - One.Billion - key - instrumentalness - mode, data = trai
  y = as.factor(train_data$Popular),
  family = "binomial",
  alpha = 0  # Set alpha to 1 for Lasso regularization
)

# Make predictions on the test set
predictions <- predict(lasso_model, newx = model.matrix(Popular ~ . - speechiness - One.Billion - key -

# Convert 'popular' to a binary factor for model evaluation
test_data$Popular <- as.factor(test_data$Popular)

# Calculate baseline
baseline <- (table(train_data$Popular)[1]) / sum(table(train_data$Popular))

# Calculate accuracy
predicted_class <- factor(ifelse(predictions > 0.5, "1", "0"), levels = levels(test_data$Popular))
confusion_matrix <- confusionMatrix(predicted_class, test_data$Popular)
accuracy <- confusion_matrix$overall["Accuracy"]

# Calculate AUC
roc_curve <- roc(test_data$Popular, predictions)
```

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

```
## Warning in roc.default(test_data$Popular, predictions): Deprecated use a matrix
## as predictor. Unexpected results may be produced, please pass a numeric vector.
```

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

```r
auc <- as.numeric(auc(roc_curve))

# Print results
cat("Baseline:", round(baseline, 4), "\n")
```

```
## Baseline: 0.7533
```

```r
cat("Accuracy:", round(accuracy, 4), "\n")
```

```
## Accuracy: 0.7578
```

```r
cat("AUC:", round(auc, 4), "\n")
```

```
## AUC: 0.6162
```