

regression and classification outputs

Kiluva J, Wendy A, Daniellah M, Sururu H.

2025-06-09

```
knitr::opts_chunk$set(echo = FALSE,
                      error = FALSE,
                      warning = FALSE,
                      message = FALSE)
#=====
## REGRESSION & CLASSIFICATION #
#=====

library(here)

## here() starts at C:/Users/HP ELITEBOOK 820 G4/OneDrive/Documents/GitHub/populaarity_probability_class

setwd(here())
set.seed(42)
library(tidyverse)

## -- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
## v dplyr     1.1.4    v readr     2.1.5
## vforcats   1.0.0    v stringr   1.5.1
## v ggplot2   3.5.1    v tibble    3.2.1
## v lubridate 1.9.4    v tidyrr    1.3.1
## v purrr    1.0.2

## -- 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 errors

library(caret)

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

```
library(corrplot)

## corrplot 0.95 loaded

library(skimr)
library(DataExplorer)
library(vip)

## 
## Attaching package: 'vip'
## 
## The following object is masked from 'package:utils':
## 
##     vi

library(ranger)
library(xgboost)

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

library(e1071)
library(scales)

## 
## Attaching package: 'scales'
## 
## The following object is masked from 'package:purrr':
## 
##     discard
## 
## The following object is masked from 'package:readr':
## 
##     col_factor

library(lubridate)
library(GGally)

## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2

library(grid)
library(recipes)
```

```
##  
## Attaching package: 'recipes'  
##  
## The following object is masked from 'package:stringr':  
##  
##     fixed  
##  
## The following object is masked from 'package:stats':  
##  
##     step
```

```
library(pROC)
```

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

```
library(gridExtra)
```

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

```
library(ggpubr)  
library(knitr)  
library(MLmetrics)
```

```
##  
## Attaching package: 'MLmetrics'  
##  
## The following objects are masked from 'package:caret':  
##  
##     MAE, RMSE  
##  
## The following object is masked from 'package:base':  
##  
##     Recall
```

```
library(RColorBrewer)  
library(flextable)
```

```
##  
## Attaching package: 'flextable'  
##
```

```

## The following objects are masked from 'package:ggpubr':
##
##     border, font, rotate
##
## The following object is masked from 'package:purrr':
##
##     compose

library(officer)
library(dplyr)
library(tidyr)
library(naniar)

## 
## Attaching package: 'naniar'
##
## The following object is masked from 'package:skimr':
##
##     n_complete

library(DT)
library(htmltools)
library(tibble)

cat("\n=====\\n")

## =====

cat("Custom Dark Theme Function\\n")

## Custom Dark Theme Function

cat("=====\\n")

## =====

theme_dark_custom <- function(title_text = "") {
  theme_dark(base_size = 14) +
  theme(
    plot.background = element_rect(fill = "black"),
    panel.background = element_rect(fill = "gray20"),
    panel.grid.major = element_line(color = "gray40"),
    panel.grid.minor = element_blank(),
    plot.title = element_text(color = "white", size = 18, face = "bold", hjust = 0.5),
    plot.subtitle = element_text(color = "white", size = 14, hjust = 0.5),
    axis.text = element_text(color = "white", size = 12),
    axis.title = element_text(color = "white", size = 12),
    legend.background = element_rect(fill = "gray20"),
    legend.text = element_text(color = "white", size = 16),
    legend.title = element_text(color = "white", size = 16),

```

```

        strip.background = element_rect(fill = "gray40"),
        strip.text = element_text(color = "white", size = 14)
    )
}

cat("\n-----\n")

## -----
## =====

cat("1. DATA LOADING AND INITIAL EXPLORATION (SHARED)\n")

## 1. DATA LOADING AND INITIAL EXPLORATION (SHARED)

cat("-----\n")

## ----

spotify_charts <- read_csv("~/school docs/universal_top_spotify_songs.new.csv")

## Rows: 1750032 Columns: 25
## -- Column specification -----
## Delimiter: ","
## chr  (5): spotify_id, name, artists, country, album_name
## dbl  (17): daily_rank, daily_movement, weekly_movement, popularity, duration...
## lgl  (1): is_explicit
## date (2): snapshot_date, album_release_date
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

cat("\n-----\n")

## -----
## =====

cat("Initial data exploration\n")

## Initial data exploration

cat("-----\n")

## ----

cat("\nSummary:\n")

## 
## Summary:

```

```
print(summary(spotify_charts))
```

```
##   spotify_id          name      artists    daily_rank
##   Length:1750032    Length:1750032    Length:1750032    Min.   : 1.00
##   Class :character  Class :character  Class :character  1st Qu.:13.00
##   Mode  :character  Mode  :character  Mode  :character  Median  :25.00
##                                         Mode  :character  Mean    :25.49
##                                         Mode  :character  3rd Qu.:38.00
##                                         Mode  :character  Max.   :50.00
##
##   daily_movement    weekly_movement   country    snapshot_date
##   Min.   :-49.00000  Min.   :-49.0000  Length:1750032    Min.   :2023-10-18
##   1st Qu.:-1.00000  1st Qu.:-3.0000  Class :character  1st Qu.:2024-02-15
##   Median : 0.00000  Median : 0.0000  Mode  :character  Median :2024-06-17
##   Mean   : 0.92510  Mean   : 2.7890                Mean   :2024-06-17
##   3rd Qu.: 2.00000  3rd Qu.: 5.0000  3rd Qu.:2024-10-15
##   Max.   :49.00000  Max.   :49.0000  Max.   :2025-02-23
##
##   popularity      is_explicit     duration_ms    album_name
##   Min.   : 0.00  Mode :logical    Min.   :     0  Length:1750032
##   1st Qu.: 65.00 FALSE:1176171  1st Qu.:161655  Class :character
##   Median : 80.00 TRUE :573861   Median :185917   Mode  :character
##   Mean   : 76.06                      Mean   :193466
##   3rd Qu.: 88.00                      3rd Qu.:218423
##   Max.   :100.00                      Max.   :939666
##
##   album_release_date  danceability    energy        key
##   Min.   :1900-01-01  Min.   :0.00000  Min.   :2.01e-05  Min.   : 0.000
##   1st Qu.:2023-05-19  1st Qu.:0.5850  1st Qu.:5.51e-01  1st Qu.: 2.000
##   Median :2023-12-01  Median :0.7010  Median :6.68e-01  Median : 6.000
##   Mean   :2022-03-16  Mean   :0.6791  Mean   :6.49e-01  Mean   : 5.543
##   3rd Qu.:2024-05-23  3rd Qu.:0.7830  3rd Qu.:7.65e-01  3rd Qu.: 9.000
##   Max.   :2025-02-21  Max.   :0.9880  Max.   :9.98e-01  Max.   :11.000
##   NA's   :658
##
##   loudness          mode       speechiness    acousticness
##   Min.   :-54.341  Min.   :0.00000  Min.   :0.000000  Min.   :0.0000034
##   1st Qu.:-7.805   1st Qu.:0.00000  1st Qu.:0.03850  1st Qu.:0.0670000
##   Median :-6.025   Median :1.00000  Median :0.05780  Median :0.1890000
##   Mean   :-6.638   Mean   :0.5379   Mean   :0.09493  Mean   :0.2743197
##   3rd Qu.:-4.712   3rd Qu.:1.00000  3rd Qu.:0.11000  3rd Qu.:0.4370000
##   Max.   : 3.233   Max.   :1.00000  Max.   :0.93900  Max.   :0.9960000
##
##   instrumentalness  liveness    valence        tempo
##   Min.   :0.0000000  Min.   :0.0139  Min.   :0.0000  Min.   : 0.0
##   1st Qu.:0.0000000  1st Qu.:0.0961  1st Qu.:0.3710  1st Qu.:100.0
##   Median :0.0000013  Median :0.1210  Median :0.5520  Median :120.0
##   Mean   :0.0206821  Mean   :0.1708  Mean   :0.5493  Mean   :122.2
##   3rd Qu.:0.0000857  3rd Qu.:0.2050  3rd Qu.:0.7350  3rd Qu.:140.1
##   Max.   :0.9950000  Max.   :0.9780  Max.   :0.9920  Max.   :236.1
##
##   time_signature
##   Min.   :0.000
##   1st Qu.:4.000
```

```

## Median :4.000
## Mean   :3.901
## 3rd Qu.:4.000
## Max.   :5.000
## 

cat("\nSkim:\n")

##
## Skim:

print(skim(spotify_charts))

## -- Data Summary -----
##                               Values
## Name                      spotify_charts
## Number of rows            1750032
## Number of columns          25
##
## -----
## Column type frequency:
##   character                5
##   Date                      2
##   logical                   1
##   numeric                  17
##
## -----
## Group variables           None
##
## -- Variable type: character -----
##   skim_variable n_missing complete_rate min max empty n_unique whitespace
## 1 spotify_id             0          1     22    22      0    20796          0
## 2 name                    30         1.00   1 286      0    18316          0
## 3 artists                 29         1.00   1 284      0    11756          0
## 4 country                 23907       0.986   2    2      0      72          0
## 5 album_name              821         1.00   1 286      0    13853          0
##
## -- Variable type: Date -----
##   skim_variable      n_missing complete_rate min          max          median
## 1 snapshot_date          0          1 2023-10-18 2025-02-23 2024-06-17
## 2 album_release_date     658        1.00 1900-01-01 2025-02-21 2023-12-01
##   n_unique
## 1      483
## 2     2630
##
## -- Variable type: logical -----
##   skim_variable n_missing complete_rate mean count
## 1 is_explicit           0          1 0.328 FAL: 1176171, TRU: 573861
##
## -- Variable type: numeric -----
##   skim_variable      n_missing complete_rate      mean        sd        p0
## 1 daily_rank             0          1      25.5      14.4        1
## 2 daily_movement          0          1      0.925      7.00     -49
## 3 weekly_movement         0          1      2.79      12.1     -49
## 4 popularity              0          1      76.1      15.8        0

```

```

## 5 duration_ms          0           1 193466.    49650.      0
## 6 danceability         0           1     0.679    0.141      0
## 7 energy                0           1     0.649    0.167  0.0000201
## 8 key                   0           1      5.54     3.58      0
## 9 loudness              0           1     -6.64     3.40   -54.3
## 10 mode                  0           1     0.538    0.499      0
## 11 speechiness          0           1     0.0949   0.0909      0
## 12 acousticness         0           1     0.274    0.251  0.00000345
## 13 instrumentalness     0           1     0.0207   0.105      0
## 14 liveness              0           1     0.171    0.125  0.0139
## 15 valence               0           1     0.549    0.230      0
## 16 tempo                 0           1     122.     28.1       0
## 17 time_signature        0           1      3.90    0.405      0

##          p25      p50      p75      p100 hist
## 1       13      2.5 e+1  3.8 e+1    50
## 2       -1        0      2 e+0     49
## 3       -3        0      5 e+0     49
## 4       65      8 e+1   8.8 e+1   100
## 5  161655  1.86e+5  2.18e+5  939666
## 6       0.585  7.01e-1  7.83e-1   0.988
## 7       0.551  6.68e-1  7.65e-1   0.998
## 8       2       6 e+0   9 e+0     11
## 9      -7.80  -6.03e+0 -4.71e+0   3.23
## 10      0       1 e+0   1 e+0     1
## 11      0.0385 5.78e-2  1.1 e-1   0.939
## 12      0.067  1.89e-1  4.37e-1   0.996
## 13      0       1.28e-6 8.57e-5   0.995
## 14      0.0961 1.21e-1  2.05e-1   0.978
## 15      0.371  5.52e-1  7.35e-1   0.992
## 16     100.    1.20e+2  1.40e+2   236.
## 17       4       4 e+0   4 e+0     5

```

```

missing_values <- colSums(is.na(spotify_charts))
cat("\nMissing values per column:\n")

```

```

## 
## Missing values per column:

```

```

print(missing_values[missing_values > 0])

```

	name	artists	country	album_name
##	30	29	23907	821
## album_release_date	658			

```

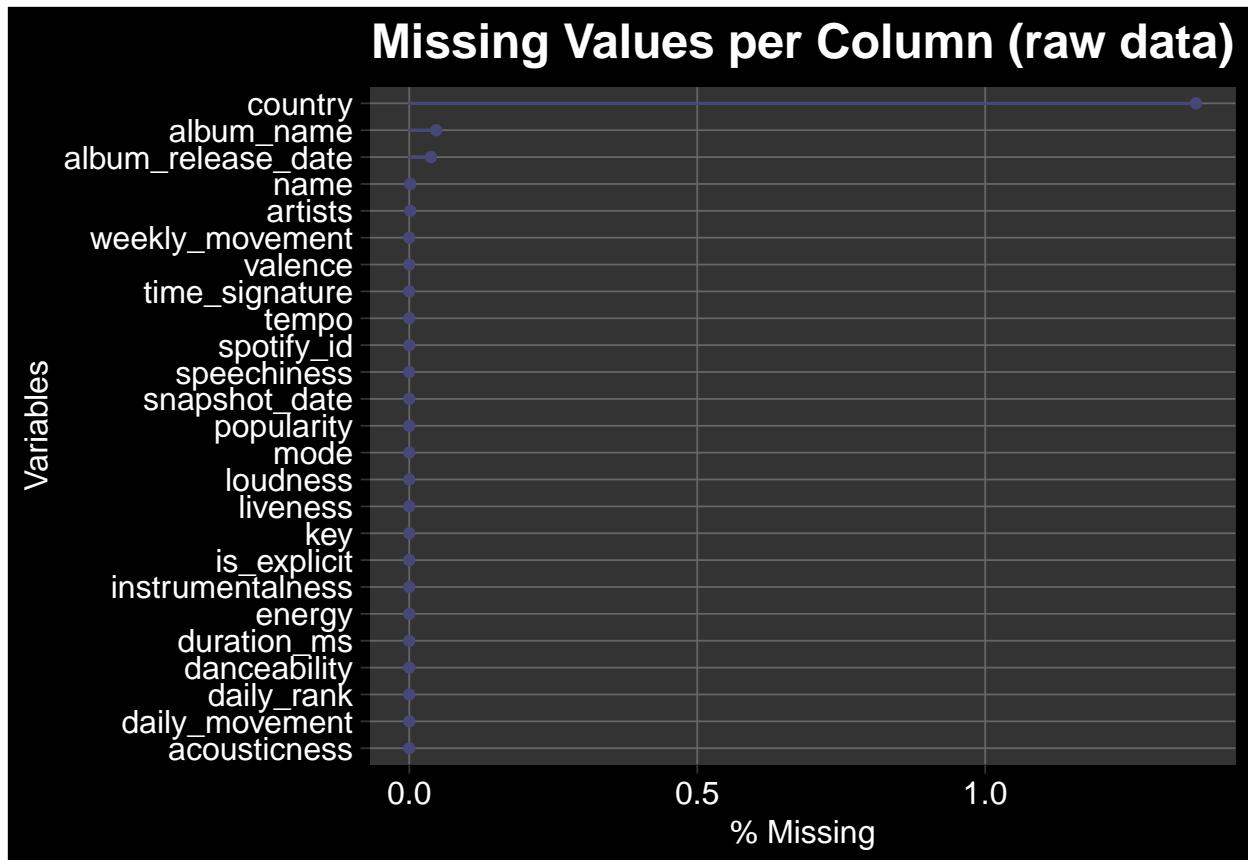
missing_heatmap_1 <- gg_miss_var(spotify_charts, show_pct = TRUE) +
  labs(title = "Missing Values per Column (raw data)") +
  theme_dark_custom()

```

```

print(missing_heatmap_1)

```



```

cat("\n=====\n")
## -----
## -----
cat("2. DATA CLEANING AND FEATURE ENGINEERING (SHARED)\n")

## 2. DATA CLEANING AND FEATURE ENGINEERING (SHARED)

cat("=====\n")
## -----
spotify_charts <- spotify_charts %>%
  group_by(spotify_id) %>%
  mutate(
    market_count = n_distinct(country, na.rm = TRUE),
    other_charted_countries = paste(country[!duplicated(country)], collapse = ", "),
  ) %>%
  slice_max(order_by = popularity, n = 1, with_ties = FALSE) %>%
  ungroup() %>%
  mutate(
    artist_count = sapply(strsplit(artists, ","), length),
    snapshot_date = ymd(snapshot_date),
  )

```

```

album_release_date = ymd(album_release_date),
days_out = as.numeric(snapshot_date - album_release_date),
is_explicit = as.integer(is_explicit),
duration_min = duration_ms / 60000
) %>%
select(-duration_ms)

colnames(spotify_charts)[colnames(spotify_charts) == "duration_min"] <- "duration_min"

view(spotify_charts)

cat("\nCleaned and Engineered Data (First few rows):\n")

## 
## Cleaned and Engineered Data (First few rows):

print(head(spotify_charts))

## # A tibble: 6 x 29
##   spotify_id      name  artists daily_rank daily_movement weekly_movement country
##   <chr>          <chr>  <chr>       <dbl>        <dbl>        <dbl> <chr>
## 1 000n6Lx4yqUAs~ béke Azahri~      20         -2        -5 HU
## 2 001TLpmtuQMWJ~ All ~ Olexes~     23         27        27 DE
## 3 003vvx7Niy0yv~ Mr. ~ The Ki~     41          5         7 IE
## 4 005L1nFVHbcd~ Obě ~ Nik Te~    43         -3         7 CZ
## 5 005coccyIL36CV~ En S~ Postgi~   50         -6        -16 NO
## 6 006oGnrSZevqZ~ Diva~ Melike~    38          2         -3 TR
## # i 22 more variables: snapshot_date <date>, popularity <dbl>,
## #   is_explicit <int>, album_name <chr>, album_release_date <date>,
## #   danceability <dbl>, energy <dbl>, key <dbl>, loudness <dbl>, mode <dbl>,
## #   speechiness <dbl>, acousticness <dbl>, instrumentalness <dbl>,
## #   liveness <dbl>, valence <dbl>, tempo <dbl>, time_signature <dbl>,
## #   market_count <int>, other_charted_countries <chr>, artist_count <int>,
## #   days_out <dbl>, duration_min <dbl>

cat("\nPreparing dataset for regression modeling\n")

## 
## Preparing dataset for regression modeling

regression_data <- spotify_charts %>%
  select(-country, -other_charted_countries, -snapshot_date, -name, -artists,
         -album_name, -album_release_date, -spotify_id) %>%
  mutate(across(where(is.character), as.factor)) %>%
  mutate(across(where(is.numeric), ~if_else(is.na(.), median(., na.rm = TRUE), .))) %>%
  filter(popularity != 0)

cat("\nArranging columns\n")

## 
## Arranging columns

```

```

regression_data <- regression_data %>%
  select(popularity, market_count, daily_rank, days_out, artist_count, daily_movement, weekly_movement,
         duration_min, is_explicit, mode, danceability, energy, loudness, speechiness,
         acousticness, instrumentalness, time_signature, liveness, valence, key, tempo)

cat("\nRegression Modeling Data (Summary):\n")

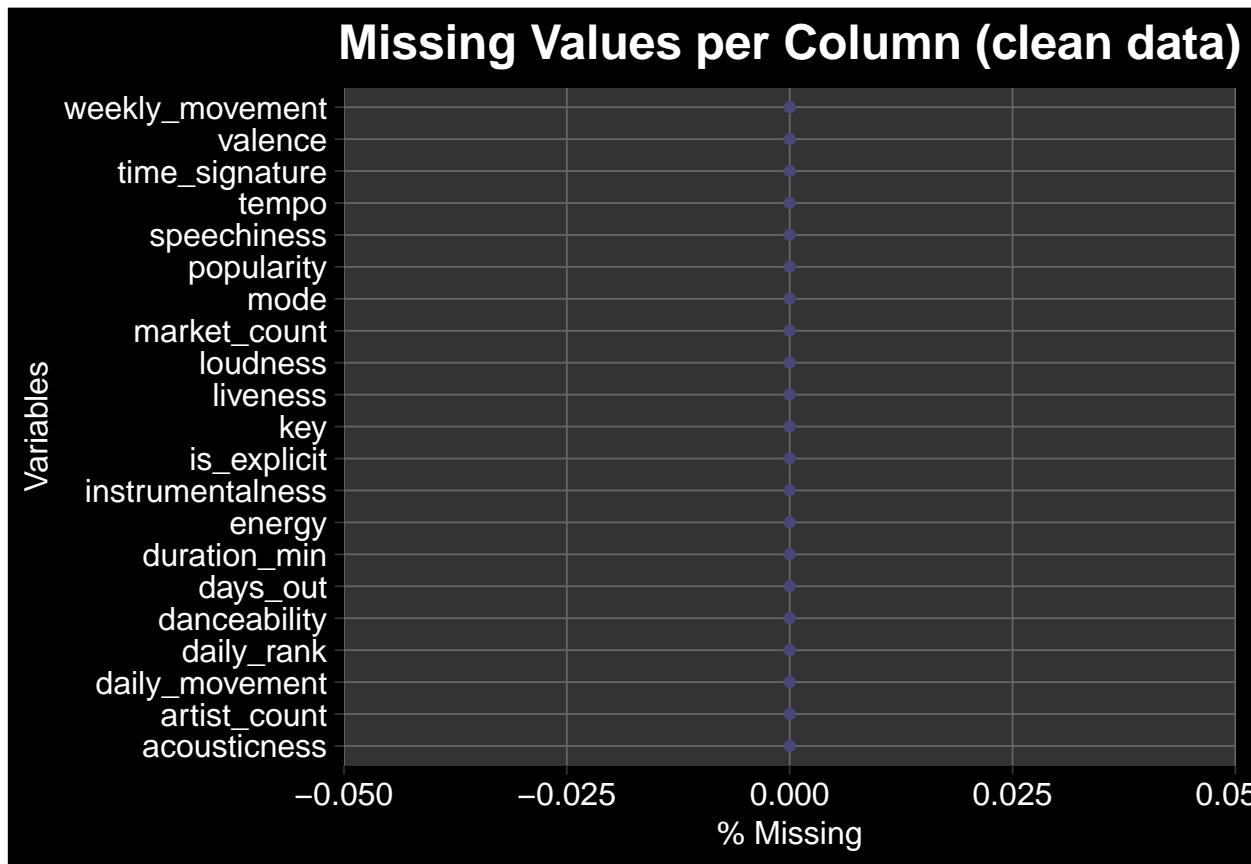
## 
## Regression Modeling Data (Summary):

view(regression_data)

missing_regression <- gg_miss_var(regression_data, show_pct = TRUE) +
  labs(title = "Missing Values per Column (clean data)") +
  theme_dark_custom()

print(missing_regression)

```



```

cat("\n=====\\n=====\\n=====\")\n\n##\n## =====\n
```

```

cat("DESCRIPTIVE & SUMMARY STATISTICS, SCATTER AND CORRELATION PLOTS\n")

## DESCRIPTIVE & SUMMARY STATISTICS, SCATTER AND CORRELATION PLOTS

cat("=====\\n")

## =====

cat("\nDescriptive Statistics for All Numeric Features (Flextable with Dark Theme)\n")

## 
## Descriptive Statistics for All Numeric Features (Flextable with Dark Theme)

numeric_cols_desc <- regression_data %>%
  select(where(is.numeric))

descriptive_stats <- numeric_cols_desc %>%
  summarise(across(everything(), list(
    Mean = ~round(mean(., na.rm = TRUE), 2),
    SE = ~round(sd(., na.rm = TRUE) / sqrt(n()), 2),
    Median = ~round(median(., na.rm = TRUE), 2),
    SD = ~round(sd(., na.rm = TRUE), 2),
    Min = ~round(min(., na.rm = TRUE), 2),
    Max = ~round(max(., na.rm = TRUE), 2),
    N = ~n()
  ))) %>%
  pivot_longer(cols = everything(),
               names_to = c("Attribute", ".value"),
               names_sep = "_(?=[^_]+$)")
)

descriptive_stats$Attribute <- as.character(descriptive_stats$Attribute)

if (nrow(descriptive_stats) > 0) {
  table_flex <- flextable(descriptive_stats) %>%
    set_caption(caption = "Descriptive Statistics for All Numeric Features") %>%
    autofit() %>%
    fontsize(size = 8, part = "all") %>%
    theme_box() %>%
    bg(bg = "grey20", part = "header") %>%
    color(color = "white", part = "header") %>%
    bg(bg = "grey15", part = "body") %>%
    color(color = "grey85", part = "body") %>%
    border_outer(part = "all", border = fp_border(color = "grey50")) %>%
    border_inner_h(part = "all", border = fp_border(color = "grey40")) %>%
    border_inner_v(part = "all", border = fp_border(color = "grey40")) %>%
    set_formatter(
      Mean = function(x) sprintf("%.2f", x),
      SE = function(x) sprintf("%.2f", x),
      Median = function(x) sprintf("%.2f", x),
      SD = function(x) sprintf("%.2f", x),

```

```

        Min = function(x) sprintf("%.2f", x),
        Max = function(x) sprintf("%.2f", x)
    )
    print(table_flex)
} else {
  cat("No numeric columns found for descriptive statistics table.\n")
}

## a flextable object.
## col_keys: `Attribute`, `Mean`, `SE`, `Median`, `SD`, `Min`, `Max`, `N`
## header has 1 row(s)
## body has 21 row(s)
## original dataset sample:
##      Attribute   Mean     SE Median      SD Min   Max     N
## 1 popularity 58.22  0.13     60 17.93   1 100 20147
## 2 market_count 2.20  0.03     1  4.73   0  69 20147
## 3 daily_rank 35.10  0.10     40 14.36   1  50 20147
## 4 days_out 947.76 19.16    46 2719.65   1 45311 20147
## 5 artist_count 1.63  0.01     1  1.00   1   26 20147

cat("\n-----\n")

## -----
## Density Distributions for Numerical Features

## Density Distributions for Numerical Features

cat("-----\n")

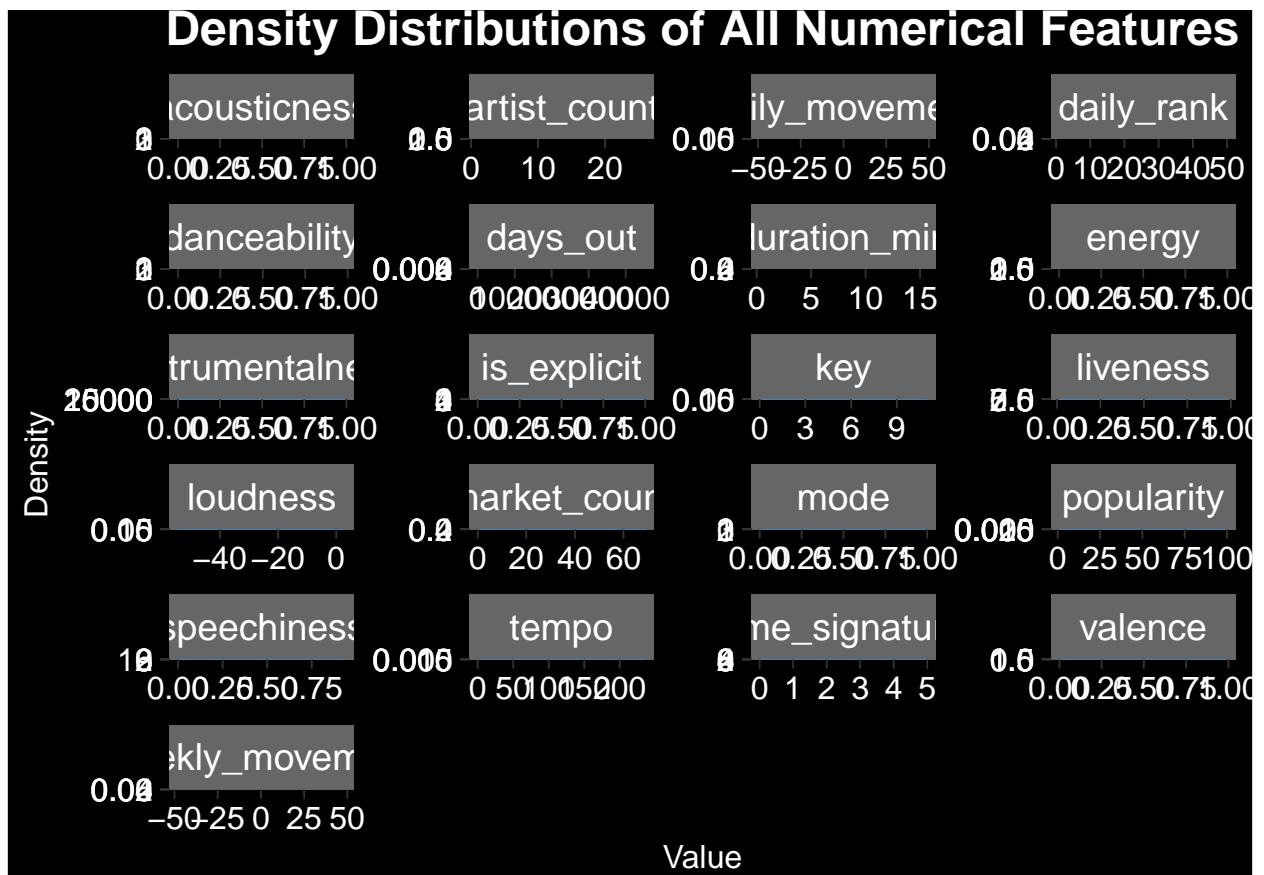
## -----
numeric_features_for_density <- regression_data %>%
  select(where(is.numeric))

if (ncol(numeric_features_for_density) > 0) {
  data_long_features <- numeric_features_for_density %>%
    pivot_longer(cols = everything(), names_to = "Feature", values_to = "Value")

  all_features_density_plot <- ggplot(data = data_long_features, aes(x = Value)) +
    geom_density(fill = "skyblue", alpha = 0.7, color = "steelblue") +
    facet_wrap(~ Feature, scales = "free", ncol = 4) +
    labs(title = "Density Distributions of All Numerical Features",
         x = "Value", y = "Density") +
    theme_dark_custom()

  print(all_features_density_plot)
} else {
  cat("No numeric features to plot densities.\n")
}

```



```

cat("\n")
## -----
## Generate the scatter plot matrices\n

## Generate the scatter plot matrices

cat("-----\n")

## -----
print(is.null(regression_data))

## [1] FALSE

plot1_dark <- ggpairs(
  regression_data,
  columns = 1:5,
  upper = list(continuous = wrap("cor", alpha = 0.7, color = "gold")),
  lower = list(continuous = wrap("points", alpha = 0.5, color = "skyblue")),
  diag = list(continuous = wrap("densityplot",
    args = list(fill = "#F0E68C", adjust = 1.5)))
)

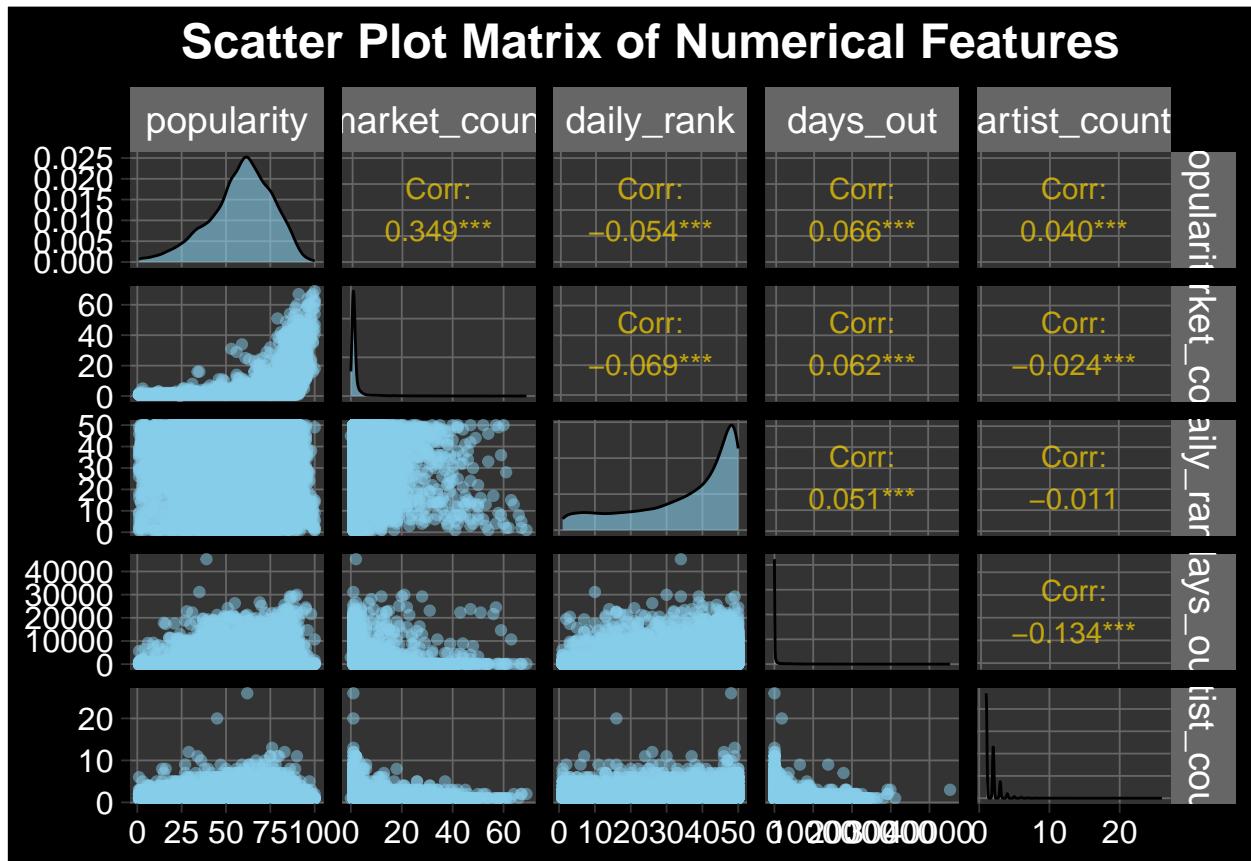
```

```

diag = list(continuous = wrap("densityDiag", fill = "skyblue", alpha = 0.6))
) +
theme_dark_custom() +
labs(title = "Scatter Plot Matrix of Numerical Features")

print(plot1_dark)

```



```
cat("\n-----\n")
```

```
##  
## -----
```

```
cat("Correlation matrix\n")
```

```
## Correlation matrix
```

```
cat("\n-----\n")
```

```
## -----
```

```
correlation_matrix <- cor(regression_data, use = "complete.obs")
```

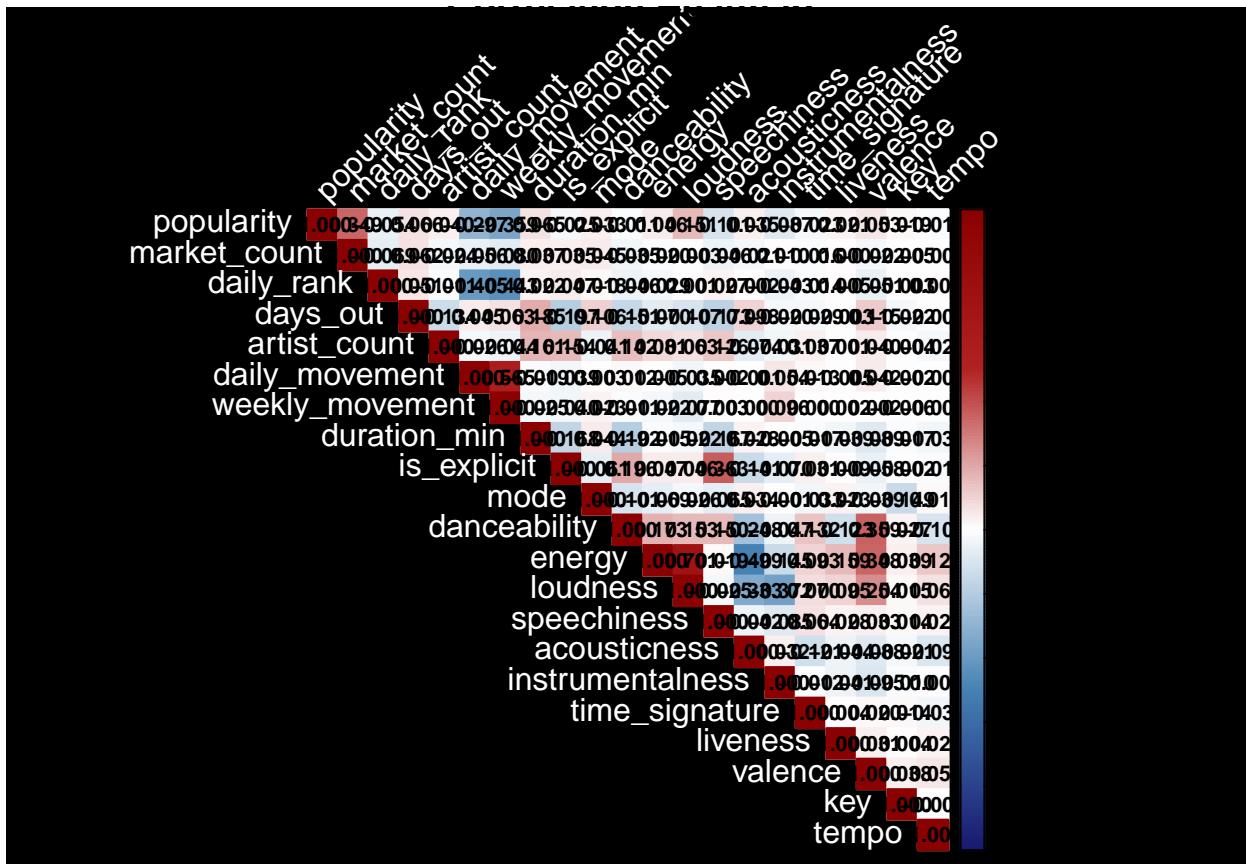
```
par(bg = "black", mar = c(0, 0, 2, 0))
```

```

corrplot(
  correlation_matrix,
  method = "color",
  col = colorRampPalette(c("midnightblue", "steelblue", "white", "firebrick", "darkred"))(200),
  tl.col = "white",
  tl.srt = 45,
  type = "upper",
  addCoef.col = "black",
  number.cex = 0.7,
  number.digits = 3,
  main = "Correlation Heatmap"
)

```

Warning in ind1:ind2: numerical expression has 2 elements: only the first used



```

par(bg = "transparent")

cat("\nExtract the correlations with 'popularity'\n")

##
## Extract the correlations with 'popularity'

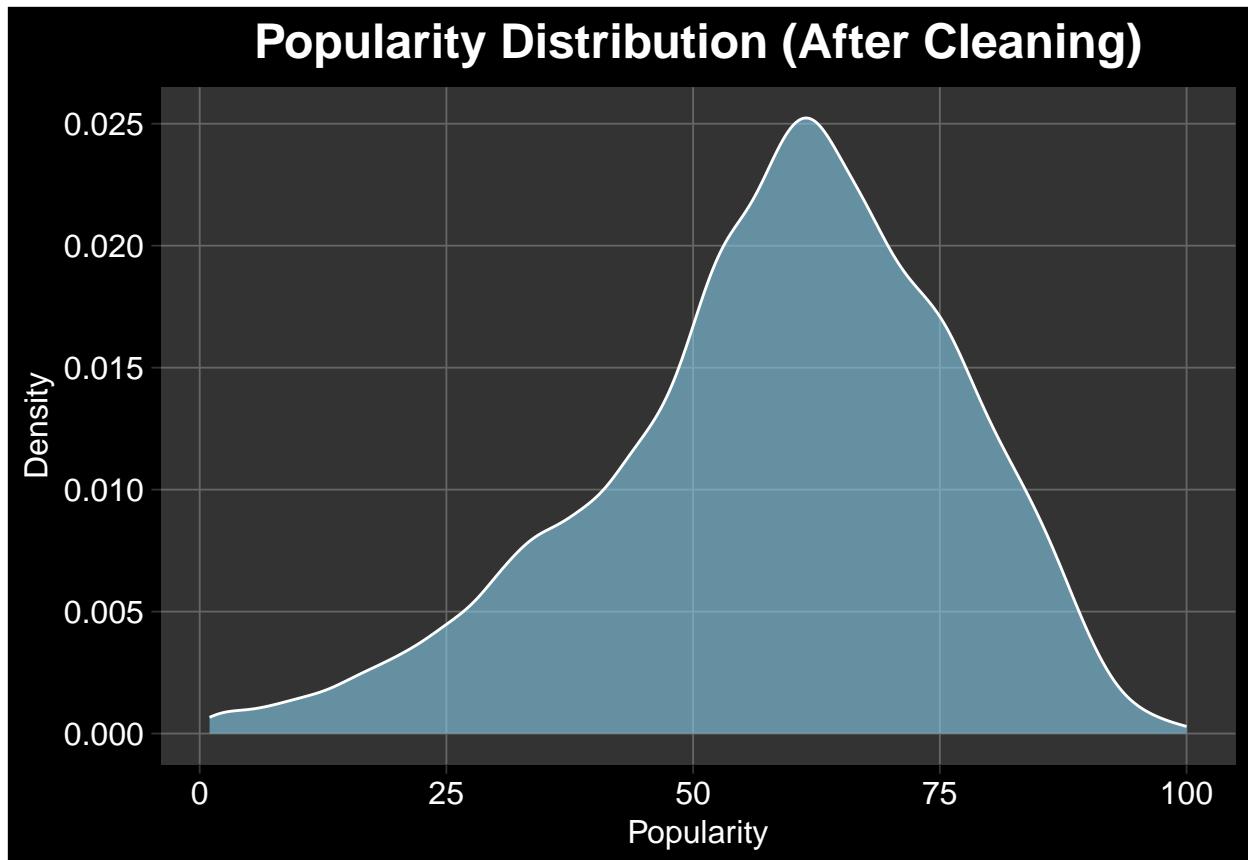
```

```
popularity_correlations <- correlation_matrix["popularity", ]  
print(popularity_correlations)
```

```
##      popularity      market_count      daily_rank      days_out  
## 1.000000000   0.34944530  -0.05355698   0.06588692  
##  artist_count  daily_movement  weekly_movement duration_min  
## 0.03992112  -0.29728253  -0.35904008  0.06474324  
##  is_explicit          mode  danceability      energy  
## -0.02513331  0.03287952  -0.01125413  0.04624822  
##  loudness  speechiness  acousticness instrumentalness  
## 0.15149381  -0.10112939  0.03516400  -0.08702787  
##  time_signature    liveness      valence          key  
## -0.02273048  0.02055590  0.05344534  0.01900805  
##          tempo  
## -0.01456841
```

```
cat("\nPopularity distribution density plot\n")
```

```
##  
## Popularity distribution density plot  
  
popularity_dist_density_plot <- ggplot(data = regression_data, aes(x = popularity)) +  
  geom_density(fill = "skyblue", alpha = 0.6, color = "white") +  
  labs(title = "Popularity Distribution (After Cleaning)",  
       x = "Popularity",  
       y = "Density") +  
  theme_dark_custom()  
print(popularity_dist_density_plot)
```



```
cat("\n-----\n")  
##  
## -----  
  
cat("3. MULTIPLE REGRESSION MODEL TRAINING AND EVALUATION\n")  
  
## 3. MULTIPLE REGRESSION MODEL TRAINING AND EVALUATION  
  
cat("-----\n")  
## -----  
  
cat("\nMultiple Regression Model Training and Evaluation\n")  
  
##  
## Multiple Regression Model Training and Evaluation  
  
cat("\nSplit data for regression\n")  
  
##  
## Split data for regression
```

```

set.seed(42)
train_index_reg <- createDataPartition(regression_data$popularity, p = 0.8, list = FALSE)
train_data_reg <- regression_data[train_index_reg, ]
test_data_reg <- regression_data[-train_index_reg, ]

cat("\nCreate preprocessing recipe for regression\n")

##
## Create preprocessing recipe for regression

preprocess_recipe_reg <- recipe(popularity ~ ., data = train_data_reg) %>%
  step_center(all_numeric(), -all_outcomes()) %>%
  step_scale(all_numeric(), -all_outcomes()) %>%
  step_dummy(all_nominal(), -all_outcomes())

prep_recipe_reg <- prep(preprocess_recipe_reg, training = train_data_reg)
train_processed_reg <- bake(prep_recipe_reg, new_data = train_data_reg)
test_processed_reg <- bake(prep_recipe_reg, new_data = test_data_reg)

cat("\nDefine resampling strategy\n")

##
## Define resampling strategy

ctrl_reg <- trainControl(method = "cv", number = 5, verboseIter = FALSE, savePredictions = "final")

regression_models <- list()

cat("\nTraining Random Forest (Regression)...\n")

##
## Training Random Forest (Regression)...

regression_models$R.F <- train(
  popularity ~.,
  data = train_processed_reg,
  method = "ranger",
  trControl = ctrl_reg,
  tuneGrid = expand.grid(mtry = c(5, 7, 9), splitrule = "variance", min.node.size = c(1, 3, 5)),
  importance = 'impurity',
  num.tree = 250
)

## Growing trees.. Progress: 89%. Estimated remaining time: 3 seconds.
## Growing trees.. Progress: 97%. Estimated remaining time: 1 seconds.
## Growing trees.. Progress: 88%. Estimated remaining time: 4 seconds.
## Growing trees.. Progress: 94%. Estimated remaining time: 1 seconds.
## Growing trees.. Progress: 86%. Estimated remaining time: 5 seconds.
## Growing trees.. Progress: 96%. Estimated remaining time: 1 seconds.
## Growing trees.. Progress: 88%. Estimated remaining time: 4 seconds.
## Growing trees.. Progress: 94%. Estimated remaining time: 1 seconds.

```

```

## Growing trees.. Progress: 86%. Estimated remaining time: 5 seconds.
## Growing trees.. Progress: 92%. Estimated remaining time: 2 seconds.
## Growing trees.. Progress: 90%. Estimated remaining time: 3 seconds.

cat("Random Forest (Regression) trained.\n")

## Random Forest (Regression) trained.

print(regression_models$R.F)

## Random Forest
##
## 16119 samples
##    20 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 12896, 12897, 12895, 12894, 12894
## Resampling results across tuning parameters:
##
##   mtry  min.node.size  RMSE      Rsquared     MAE
##   5      1             11.25225  0.6103679  8.574256
##   5      3             11.24425  0.6111896  8.574224
##   5      5             11.24677  0.6108632  8.572814
##   7      1             11.23683  0.6100312  8.528838
##   7      3             11.22223  0.6110970  8.516619
##   7      5             11.23256  0.6103468  8.524465
##   9      1             11.23706  0.6096054  8.513823
##   9      3             11.23604  0.6096087  8.516104
##   9      5             11.23816  0.6095047  8.523308
##
## Tuning parameter 'splitrule' was held constant at a value of variance
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were mtry = 7, splitrule = variance
## and min.node.size = 3.

saveRDS(regression_models$R.F, "RF_model.rds")

cat("\nTraining Gradient Boosting (Regression)... \n")

##
## Training Gradient Boosting (Regression)...

regression_models$GBM <- train(
  popularity ~ .,
  data = train_processed_reg,
  method = "gbm",
  trControl = ctrl_reg,
  tuneGrid = expand.grid(n.trees = c(150, 250), interaction.depth = c(3, 5), shrinkage = c(0.1, 0.5), n
)
cat("Gradient Boosting (Regression) trained.\n")

```

```

## Gradient Boosting (Regression) trained.

print(regression_models$GBM)

## Stochastic Gradient Boosting
##
## 16119 samples
##    20 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 12895, 12895, 12896, 12894, 12896
## Resampling results across tuning parameters:
##
##     shrinkage  interaction.depth  n.trees   RMSE    Rsquared   MAE
##     0.1          3                150      11.45237  0.5939985  8.743546
##     0.1          3                250      11.40279  0.5974534  8.694739
##     0.1          5                150      11.35192  0.6010417  8.655392
##     0.1          5                250      11.34931  0.6012436  8.634296
##     0.5          3                150      11.83981  0.5690041  9.073561
##     0.5          3                250      12.00404  0.5594809  9.157301
##     0.5          5                150      12.19315  0.5479393  9.292603
##     0.5          5                250      12.53091  0.5286793  9.524627
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were n.trees = 250, interaction.depth =
## 5, shrinkage = 0.1 and n.minobsinnode = 10.

```

```

saveRDS(regression_models$GBM, "GBM_model.rds")

cat("\nTraining XGBoost (Regression)...\n")

```

```

##
## Training XGBoost (Regression)...

```

```

regression_models$XGB <- train(
  popularity ~ .,
  data = train_processed_reg,
  method = "xgbTree",
  trControl = ctrl_reg,
  tuneGrid = expand.grid(nrounds = c(150, 250), max_depth = c(3, 6), eta = c(0.1, 0.5), gamma = 0,
                         colsample_bytree = c(0.5, 0.75), min_child_weight = c(1, 3), subsample = 0.75),
  verbose = FALSE
)

```

```

## [00:01:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:01:33] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:01:38] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:01:42] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:01:49] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:01:58] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead

```



```

## [00:07:35] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:07:44] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:07:53] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:07:59] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:08:04] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:08:07] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:08:12] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:08:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:08:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:08:34] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:08:47] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:08:53] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:08:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:09:01] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:09:06] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:09:09] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:09:16] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:09:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:09:37] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:09:43] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead

cat("XGBoost (Regression) trained.\n")

## XGBoost (Regression) trained.

print(regression_models$XGB)

## eXtreme Gradient Boosting
##
## 16119 samples
##    20 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 12896, 12893, 12895, 12897, 12895
## Resampling results across tuning parameters:
##
##     eta  max_depth  colsample_bytree  min_child_weight  nrounds  RMSE
##     0.1    3          0.50            1              150   11.31973
##     0.1    3          0.50            1              250   11.30102
##     0.1    3          0.50            3              150   11.31546
##     0.1    3          0.50            3              250   11.29587
##     0.1    3          0.75            1              150   11.31679
##     0.1    3          0.75            1              250   11.29894
##     0.1    3          0.75            3              150   11.30534
##     0.1    3          0.75            3              250   11.29151
##     0.1    6          0.50            1              150   11.29009
##     0.1    6          0.50            1              250   11.33817
##     0.1    6          0.50            3              150   11.26562
##     0.1    6          0.50            3              250   11.31624
##     0.1    6          0.75            1              150   11.24366
##     0.1    6          0.75            1              250   11.30405
##     0.1    6          0.75            3              150   11.23638

```

```

##   0.1   6      0.75      3      250    11.29054
##   0.5   3      0.50      1      150    11.72602
##   0.5   3      0.50      1      250    11.99102
##   0.5   3      0.50      3      150    11.82782
##   0.5   3      0.50      3      250    12.01543
##   0.5   3      0.75      1      150    11.82009
##   0.5   3      0.75      1      250    12.01067
##   0.5   3      0.75      3      150    11.78906
##   0.5   3      0.75      3      250    12.01758
##   0.5   6      0.50      1      150    12.99658
##   0.5   6      0.50      1      250    13.23570
##   0.5   6      0.50      3      150    12.99259
##   0.5   6      0.50      3      250    13.30242
##   0.5   6      0.75      1      150    13.10325
##   0.5   6      0.75      1      250    13.29656
##   0.5   6      0.75      3      150    13.04764
##   0.5   6      0.75      3      250    13.27448
## Rsquared    MAE
##  0.6035413  8.639645
##  0.6047253  8.609417
##  0.6038284  8.621011
##  0.6051264  8.597898
##  0.6036602  8.621617
##  0.6048905  8.601486
##  0.6045587  8.607125
##  0.6054556  8.584552
##  0.6055183  8.580545
##  0.6023070  8.611489
##  0.6072517  8.582943
##  0.6038935  8.601958
##  0.6087371  8.530931
##  0.6048505  8.576186
##  0.6092876  8.537565
##  0.6057589  8.578561
##  0.5772076  8.945510
##  0.5609970  9.159663
##  0.5702632  9.037549
##  0.5595938  9.179469
##  0.5710667  9.002956
##  0.5601629  9.161566
##  0.5732281  8.978095
##  0.5598855  9.181323
##  0.5051270  9.959299
##  0.4941354  10.167992
##  0.5069339  9.961742
##  0.4912436  10.211237
##  0.4983185  10.014026
##  0.4897411  10.181238
##  0.5023238  10.001360
##  0.4916449  10.177028
##
## Tuning parameter 'gamma' was held constant at a value of 0
## Tuning
## parameter 'subsample' was held constant at a value of 0.75

```

```

## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nrounds = 150, max_depth = 6, eta
## = 0.1, gamma = 0, colsample_bytree = 0.75, min_child_weight = 3 and
## subsample = 0.75.

saveRDS(regression_models$XGB, "XGB_model.rds")

cat("\nEvaluate all regression models on the test set\n")

##
## Evaluate all regression models on the test set

regression_evaluation_results <- lapply(names(regression_models), function(model_name) {
  predictions <- predict(regression_models[[model_name]], newdata = test_processed_reg)
  performance <- postResample(pred = predictions, obs = test_processed_reg$popularity)
  cat(paste0("\nPerformance of ", model_name, " (Regression) on Test Set:\n"))
  print(performance)
  return(list(performance = performance, predictions = predictions))
})

## Performance of R.F (Regression) on Test Set:
##      RMSE    Rsquared      MAE
## 11.1621423  0.6066587  8.4647240
##
## Performance of GBM (Regression) on Test Set:
##      RMSE    Rsquared      MAE
## 11.3140282  0.5952943  8.6396861
##
## Performance of XGB (Regression) on Test Set:
##      RMSE    Rsquared      MAE
## 11.1882146  0.6043203  8.4954310

names(regression_evaluation_results) <- names(regression_models)

cat("\nCompile performance metrics for comparison\n")

##
## Compile performance metrics for comparison

regression_results_df <- data.frame(
  Model = names(regression_models),
  RMSE = sapply(regression_evaluation_results, function(x) x$performance["RMSE"]),
  Rsquared = sapply(regression_evaluation_results, function(x) x$performance["Rsquared"]),
  MAE = sapply(regression_evaluation_results, function(x) x$performance["MAE"])
)

regression_results_df <- as.data.frame(t(regression_results_df[,-1]))
colnames(regression_results_df) <- names(regression_models)
rownames(regression_results_df) <- c("RMSE", "Rsquared", "MAE")

cat("\nRegression Model Performance Comparison:\n")

```

```

##  

## Regression Model Performance Comparison:  
  

print(regression_results_df)  
  

##  
          R.F      GBM      XGB  

## RMSE    11.1621423 11.3140282 11.1882146  

## Rsquared 0.6066587  0.5952943  0.6043203  

## MAE     8.4647240  8.6396861  8.4954310  
  

cat("\nDetermine the best performing regression model based on Rsquared\n")  
  

##  

## Determine the best performing regression model based on Rsquared  
  

best_reg_model_name <- names(which.max(regression_results_df["Rsquared",]))  

best_reg_model <- regression_models[[best_reg_model_name]]  
  

cat(paste("\nBest performing regression model (based on Rsquared):", best_reg_model_name, "\n"))  
  

##  

## Best performing regression model (based on Rsquared): R.F  
  

cat("\nGet predictions from the best regression model\n")  
  

##  

## Get predictions from the best regression model  
  

predictions_best_reg <- predict(best_reg_model, newdata = test_processed_reg)  

cat("\n-----  
  

##  

## -----  
  

cat("Visualization of Regression Model Performance\n")  
  

## Visualization of Regression Model Performance  
  

cat("-----  
  

## -----  
  

cat("\nCreate a data frame for plotting performance metrics\n")  
  

##  

## Create a data frame for plotting performance metrics

```

```

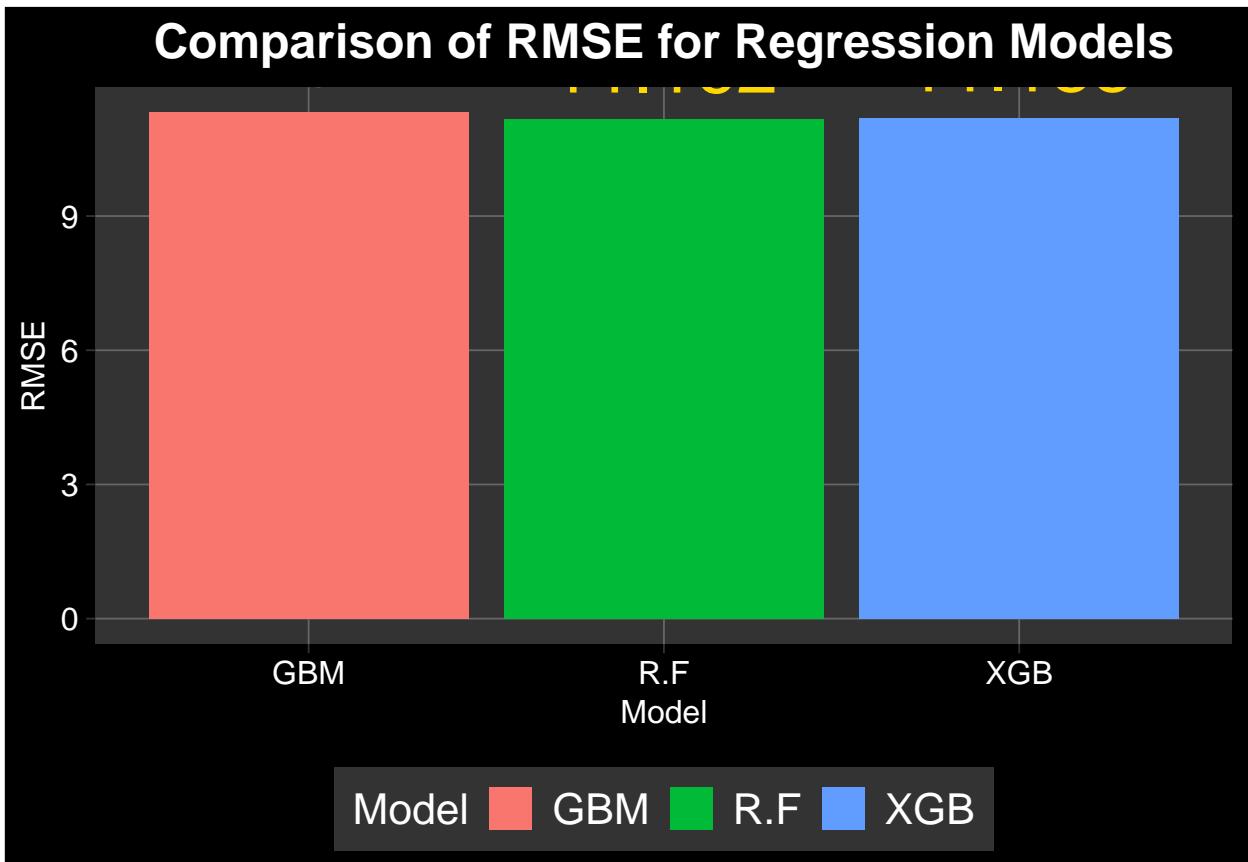
performance_plot_data <- gather(regression_results_df, key = "Model", value = "Value") %>%
  mutate(Metric = rep(rownames(regression_results_df), times = length(regression_models)))

cat("\nPlotting RMSE with numeric values\n")

## 
## Plotting RMSE with numeric values

rmse_plot <- ggplot(performance_plot_data %>% filter(Metric == "RMSE"), aes(x = Model, y = Value, fill =
  geom_bar(stat = "identity") +
  geom_text(aes(label = round(Value, 3), vjust = -0.5), size = 10, col = "gold") +
  labs(title = "Comparison of RMSE for Regression Models", y = "RMSE") +
  theme_dark_custom() +
  theme(legend.position = "bottom")
print(rmse_plot)

```



```
cat("\nPlotting Rsquared with numeric values\n")
```

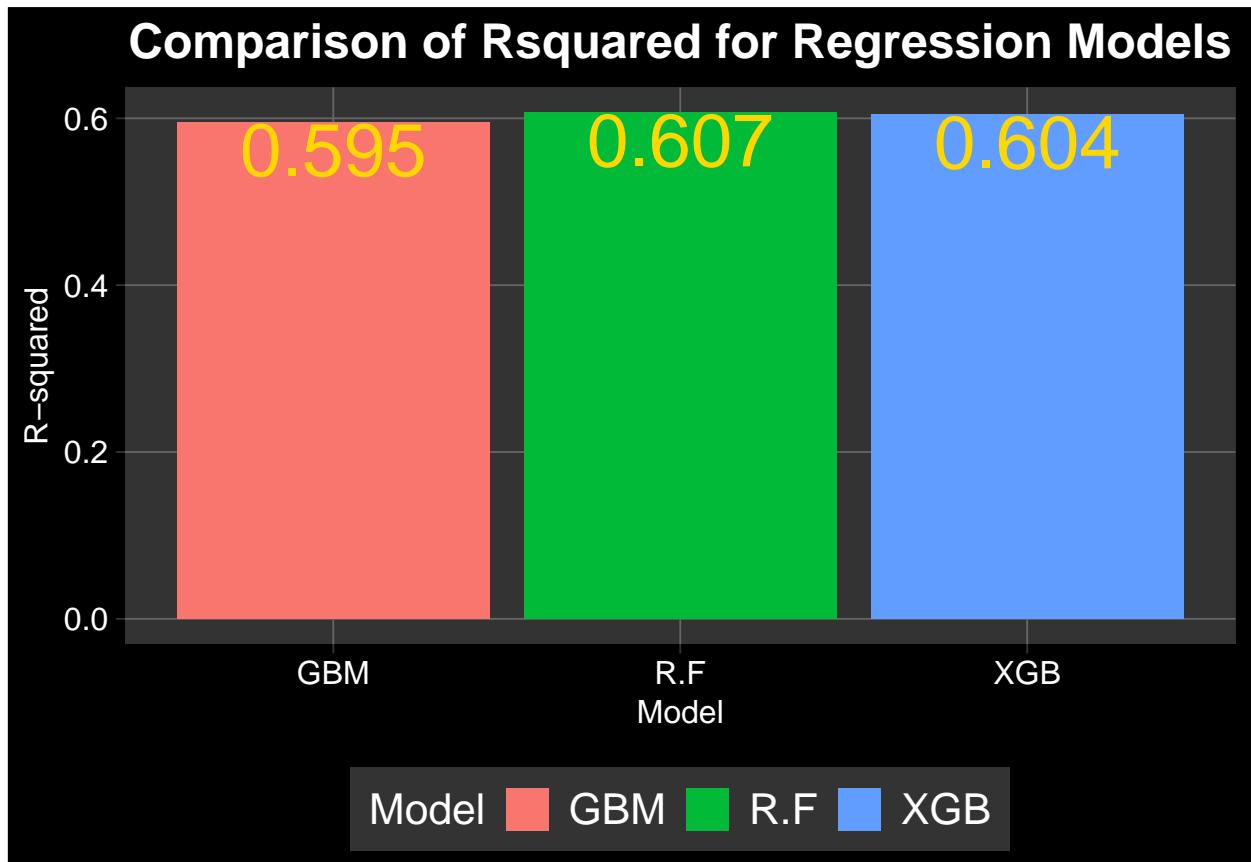
```
## 
## Plotting Rsquared with numeric values
```

```
rsquared_plot <- ggplot(performance_plot_data %>% filter(Metric == "Rsquared"), aes(x = Model, y = Value,
  geom_bar(stat = "identity") +
```

```

geom_text(aes(label = round(Value, 3), vjust = 1), size = 10, col = "gold") +
  labs(title = "Comparison of Rsquared for Regression Models", y = "R-squared") +
  theme_dark_custom() +
  theme(legend.position = "bottom")
print(rsquared_plot)

```



```

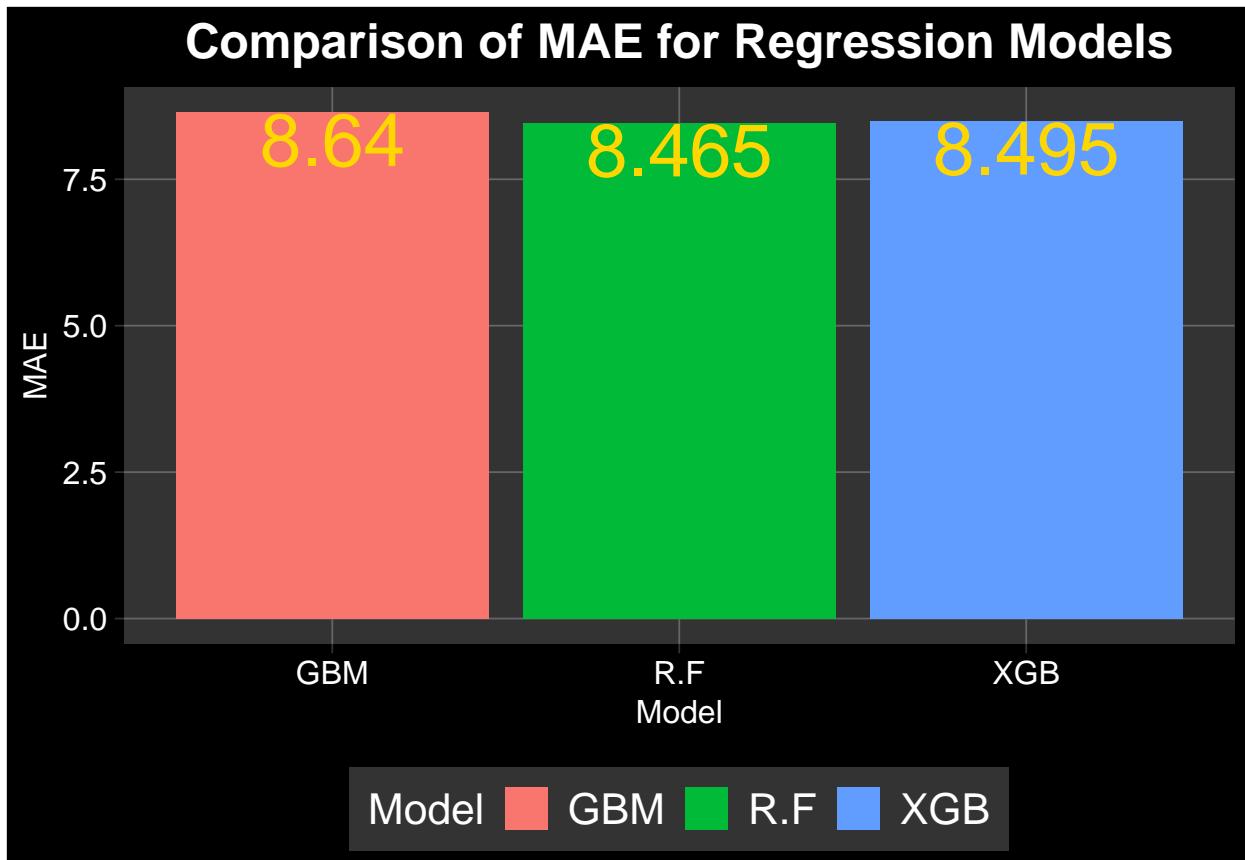
cat("\nPlotting MAE with numeric values\n")

##  

## Plotting MAE with numeric values

mae_plot <- ggplot(performance_plot_data %>% filter(Metric == "MAE"), aes(x = Model, y = Value, fill = Model)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = round(Value, 3), vjust = 1), size = 10, col = "gold") +
  labs(title = "Comparison of MAE for Regression Models", y = "MAE") +
  theme_dark_custom() +
  theme(legend.position = "bottom")
print(mae_plot)

```



```

cat("\nCreate a data frame for plotting predicted vs actual values\n")

##
## Create a data frame for plotting predicted vs actual values

plot_data_reg <- data.frame(
  Model = rep(names(regression_models), each = nrow(test_processed_reg)),
  Predicted = unlist(lapply(regression_evaluation_results, function(x) x$predictions)),
  Actual = rep(test_processed_reg$popularity, times = length(regression_models))
)

cat("\nScatter plot of predicted vs actual values for each model\n")

##
## Scatter plot of predicted vs actual values for each model

regression_scatter_plot <- ggplot(plot_data_reg, aes(x = Actual, y = Predicted, color = Model)) +
  geom_point(alpha = 0.7) +
  geom_abline(intercept = 0, slope = 1, color = "gold", linetype = "dashed") +
  geom_smooth(aes(group = Model), method = "lm", se = FALSE) +
  facet_wrap(~Model, scales = "free") +
  labs(title = "Predicted vs. Actual Popularity for Regression Models",
       x = "Actual Popularity",
       y = "Predicted Popularity") +

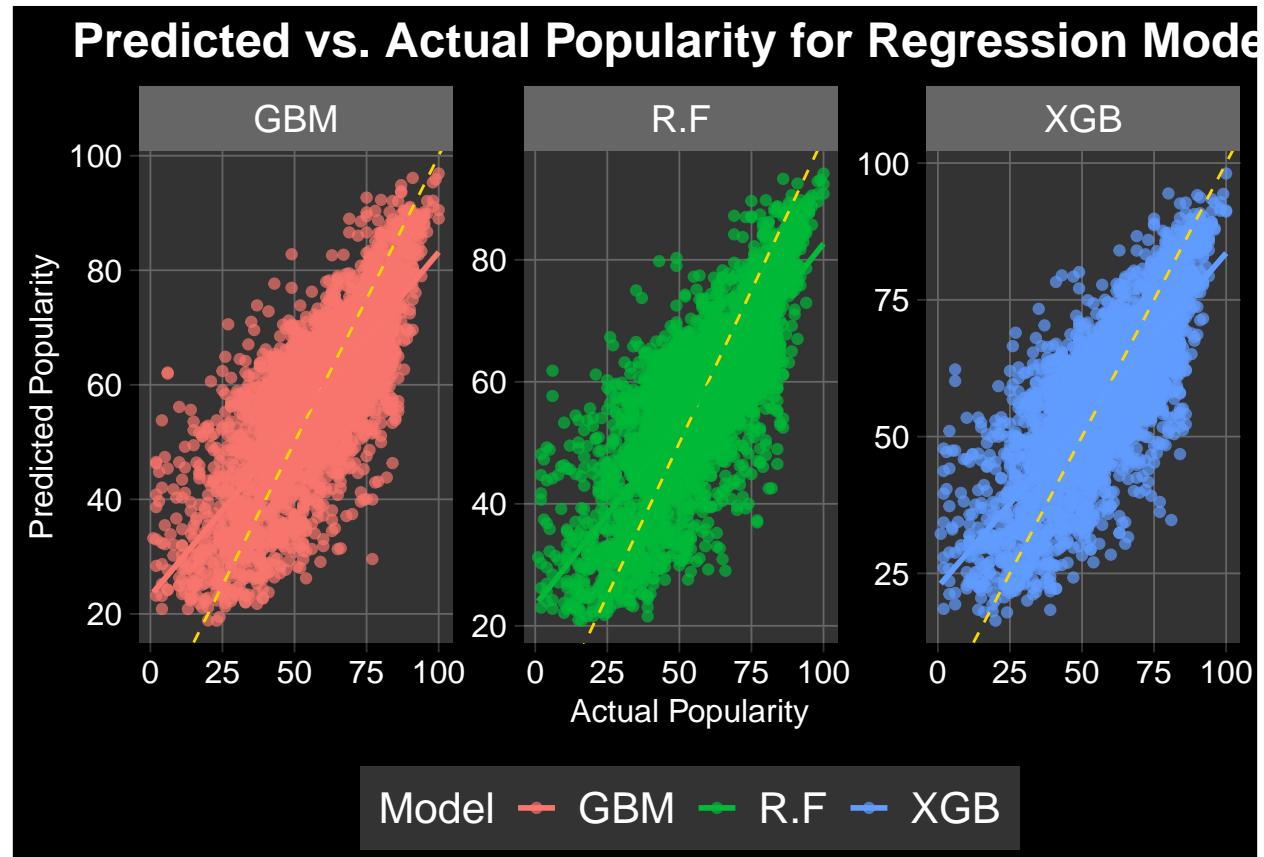
```

```

theme_dark_custom() +
  theme(legend.position = "bottom")
print(regression_scatter_plot)

```

```
## `geom_smooth()` using formula = 'y ~ x'
```



```
cat("\n-----\n")
```

```
##  
## -----
```

```
cat("Feature Importance Plot (Best Regression Model)\n")
```

```
## Feature Importance Plot (Best Regression Model)
```

```
cat("-----\n")
```

```
## -----
```

```
cat("\nPlot feature importance for the best regression model\n")
```

```
##
```

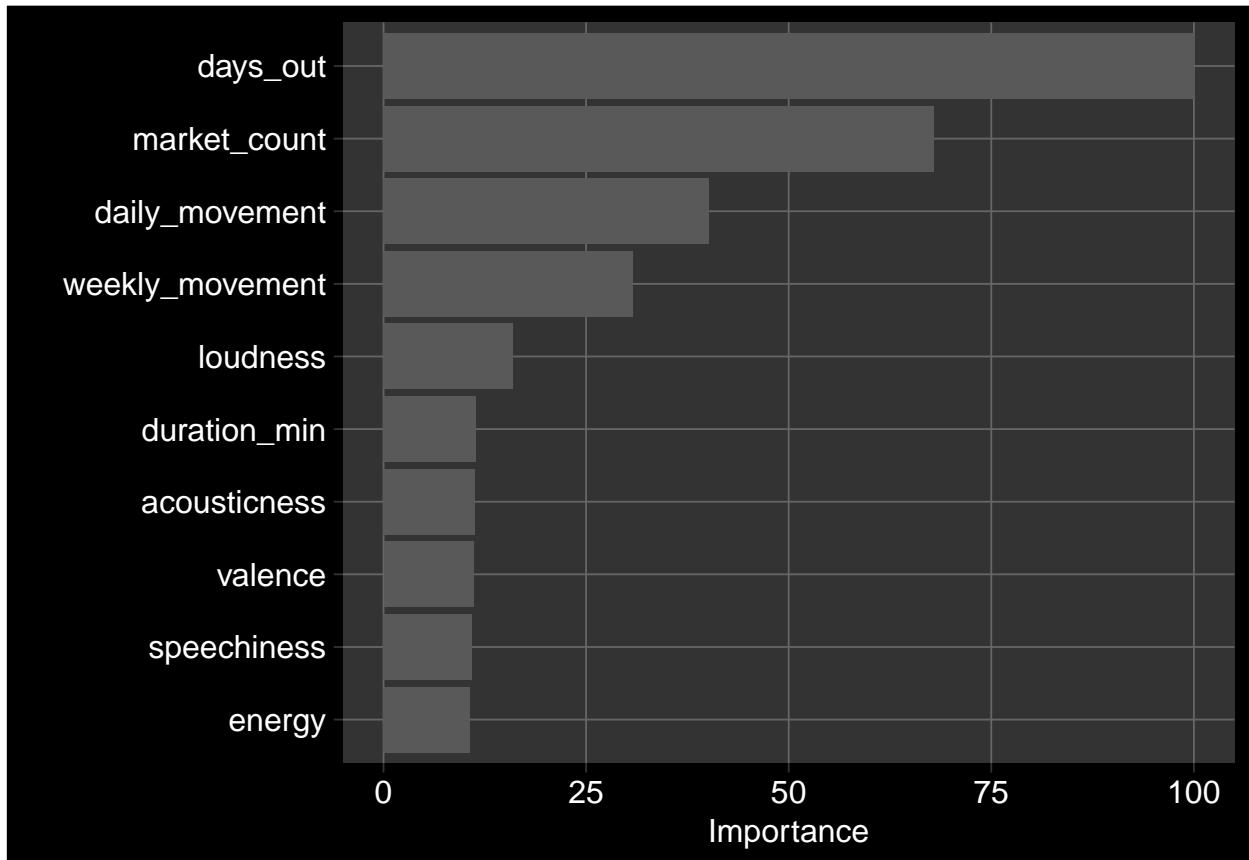
```
## Plot feature importance for the best regression model
```

```

vip_plot_reg <- vip(
  best_reg_model,
  main = paste("Feature Importance in", best_reg_model_name, "(Regression)")) +
  theme_dark_custom()

print(vip_plot_reg)

```



```

cat("\n=====\n")
## 
## =====

cat("4. PREPARE DATA FOR CLASSIFICATION (FROM BEST REGRESSION OUTPUT)\n")

## 4. PREPARE DATA FOR CLASSIFICATION (FROM BEST REGRESSION OUTPUT)

cat("=====\n")
## =====

cat("\nClassification Based on Best Regression Model Predictions\n")

##
## Classification Based on Best Regression Model Predictions

```

```

cat("\nDefine popularity thresholds\n")

##  

## Define popularity thresholds

very_high_threshold <- 75  

high_threshold <- 50  

very_low_threshold <- 25

cat("\nCategorize the predicted popularity from the best regression model\n")

##  

## Categorize the predicted popularity from the best regression model

predicted_popularity_level <- case_when(  

  predictions_best_reg >= very_high_threshold ~ "very_high",  

  predictions_best_reg < very_low_threshold ~ "very_low",  

  predictions_best_reg >= high_threshold ~ "high",  

  predictions_best_reg < high_threshold ~ "low"  

)
predicted_popularity_level <- factor(predicted_popularity_level, levels = c("very_low", "low", "high", "very_high"))

cat("\nCreate the actual popularity levels from the test set\n")

##  

## Create the actual popularity levels from the test set

actual_popularity_level <- case_when(  

  test_data_reg$popularity >= very_high_threshold ~ "very_high",  

  test_data_reg$popularity < very_low_threshold ~ "very_low",  

  test_data_reg$popularity >= high_threshold ~ "high",  

  test_data_reg$popularity < high_threshold ~ "low"  

)
actual_popularity_level <- factor(actual_popularity_level, levels = c("very_low", "low", "high", "very_high"))

cat("\nPrepare data for classification model training\n")

##  

## Prepare data for classification model training

classification_data <- data.frame(  

  popularity_level = actual_popularity_level,  

  test_processed_reg
)
cat("\nRemove the original 'popularity' column to avoid redundancy/potential issues\n")

##  

## Remove the original 'popularity' column to avoid redundancy/potential issues

```

```

classification_data <- classification_data %>% select(-popularity)

cat("\nSplit data for classification\n")

## 
## Split data for classification

set.seed(42)
train_index_class <- createDataPartition(classification_data$popularity_level, p = 0.8, list = FALSE)
train_data_class <- classification_data[train_index_class, ]
test_data_class <- classification_data[-train_index_class, ]

cat("\nDerive scores and predicted levels for the classification test set (for plotting)\n")

## 
## Derive scores and predicted levels for the classification test set (for plotting)

actual_popularity_for_classification_test_num <- test_data_reg[-train_index_class, ]$popularity
predicted_numerical_popularity_for_classification_test <- predictions_best_reg[-train_index_class]

predicted_popularity_level_for_plot <- case_when(
  predicted_numerical_popularity_for_classification_test >= very_high_threshold ~ "very_high",
  predicted_numerical_popularity_for_classification_test < very_low_threshold ~ "very_low",
  predicted_numerical_popularity_for_classification_test >= high_threshold ~ "high",
  predicted_numerical_popularity_for_classification_test < high_threshold ~ "low"
)
predicted_popularity_level_for_plot <- factor(predicted_popularity_level_for_plot,
                                               levels = c("very_low", "low", "high", "very_high"))

cat("\n=====\\n")

## 
## =====

cat("5. CLASSIFICATION MODEL TRAINING AND EVALUATION\\n")

## 5. CLASSIFICATION MODEL TRAINING AND EVALUATION

cat("=====\\n")

## 
## =====

cat("\nClassification Model Training and Evaluation\\n")

## 
## Classification Model Training and Evaluation

```

```

cat("\nCustom summary function for multi-class ROC\n")

##  

## Custom summary function for multi-class ROC

multiClassSummary <- function (data, lev = NULL, model = NULL) {  

  if (length(lev) > 2) {  

    rocs <- pROC::multiclass.roc(data$obs, data[, lev])  

    auc <- pROC::auc(rocs)  

    names(auc) <- "AUC"  

    accuracy <- mean(data$obs == data$pred)  

    names(accuracy) <- "Accuracy"  

    return(c(AUC = auc, Accuracy = accuracy))  

  } else {  

    return(defaultSummary(data, lev, model))  

  }
}

cat("\nDefine resampling strategy for classification\n")

##  

## Define resampling strategy for classification

trainControl_roc <- trainControl(method = "cv",  

                                    number = 5,  

                                    allowParallel = TRUE,  

                                    summaryFunction = multiClassSummary,  

                                    classProbs = TRUE,  

                                    savePredictions = TRUE)

classification_models <- list()

cat("\nTraining Classification Model (Random Forest)... \n")

##  

## Training Classification Model (Random Forest)...

classification_models$R.F <- train(  

  popularity_level ~ .,  

  data = train_data_class,  

  method = "ranger",  

  trControl = trainControl_roc,  

  tuneGrid = expand.grid(mtry = c(5, 7, 9),  

                        min.node.size = c(1, 3, 5),  

                        splitrule = "gini"),  

  num.trees = 250,  

  metric = "AUC"
)

## Warning in train.default(x, y, weights = w, ...): The metric "AUC" was not in  

## the result set. AUC.AUC will be used instead.

```

```

cat("Random Forest Classification trained.\n")

## Random Forest Classification trained.

print(classification_models$R.F)

## Random Forest
##
## 3223 samples
##    20 predictor
##      4 classes: 'very_low', 'low', 'high', 'very_high'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 2578, 2580, 2578, 2577, 2579
## Resampling results across tuning parameters:
##
##   mtry  min.node.size  AUC.AUC    Accuracy.Accuracy
##   5      1            0.8478300  0.6940940
##   5      3            0.8451799  0.6894385
##   5      5            0.8446820  0.6906841
##   7      1            0.8478490  0.6916085
##   7      3            0.8467087  0.6944007
##   7      5            0.8463405  0.6906865
##   9      1            0.8445634  0.6888212
##   9      3            0.8452649  0.6928537
##   9      5            0.8455274  0.6903740
##
## Tuning parameter 'splitrule' was held constant at a value of gini
## AUC.AUC was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 7, splitrule = gini
## and min.node.size = 1.

print(classification_models$R.F$results)

##   mtry min.node.size splitrule  AUC.AUC Accuracy.Accuracy  AUC.AUCSD
## 1     5             1       gini 0.8478300  0.6940940  0.01548483
## 2     5             3       gini 0.8451799  0.6894385  0.01302378
## 3     5             5       gini 0.8446820  0.6906841  0.01238808
## 4     7             1       gini 0.8478490  0.6916085  0.01319797
## 5     7             3       gini 0.8467087  0.6944007  0.01282207
## 6     7             5       gini 0.8463405  0.6906865  0.01231607
## 7     9             1       gini 0.8445634  0.6888212  0.01234270
## 8     9             3       gini 0.8452649  0.6928537  0.01317911
## 9     9             5       gini 0.8455274  0.6903740  0.01402722
##
##   Accuracy.AccuracySD
## 1           0.02222283
## 2           0.01679219
## 3           0.02120189
## 4           0.01770734
## 5           0.02038249
## 6           0.02696380

```

```

## 7      0.02366248
## 8      0.02424969
## 9      0.02346741

cat("\nTraining Classification Model (XGBoost)...\n")

##
## Training Classification Model (XGBoost)...

classification_models$XGB <- train(
  popularity_level ~ .,
  data = train_data_class,
  method = "xgbTree",
  trControl = trainControl_roc,
  tuneGrid = expand.grid(nrounds = c(150, 250),
                         max_depth = c(3, 6),
                         eta = c(0.1, 0.5),
                         gamma = 0,
                         colsample_bytree = c(0.5, 0.75),
                         min_child_weight = c(1, 3),
                         subsample = 0.75),
  metric = "AUC",
  verbose = FALSE
)

## Warning in train.default(x, y, weights = w, ...): The metric "AUC" was not in
## the result set. AUC.AUC will be used instead.

## [00:11:46] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:11:46] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:11:50] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:11:50] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:11:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:11:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:12:05] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:12:05] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:12:13] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:12:13] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:12:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:12:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:12:34] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:12:35] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:12:49] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:12:49] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:12:52] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:12:52] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:12:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:12:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:13:02] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:13:02] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:13:09] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:13:09] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead

```



```

## [00:20:41] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:20:41] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:20:48] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:20:48] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:21:03] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:21:03] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:21:17] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:21:17] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:21:34] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:21:34] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:21:50] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:21:50] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:21:56] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:21:57] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:22:02] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:22:02] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:22:10] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:22:10] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:22:16] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:22:16] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:22:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:22:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:22:32] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:22:32] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:22:41] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:22:41] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:22:50] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead
## [00:22:50] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead

cat("XGBoost Classification trained.\n")

## XGBoost Classification trained.

print(classification_models$XGB)

## eXtreme Gradient Boosting
##
## 3223 samples
##   20 predictor
##   4 classes: 'very_low', 'low', 'high', 'very_high'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 2578, 2578, 2577, 2579, 2580
## Resampling results across tuning parameters:
##
##     eta  max_depth  colsample_bytree  min_child_weight  nrounds  AUC.AUC
##     0.1    3           0.50            1                150    0.8471965
##     0.1    3           0.50            1                250    0.8441051
##     0.1    3           0.50            3                150    0.8471562
##     0.1    3           0.50            3                250    0.8448742
##     0.1    3           0.75            1                150    0.8470788
##     0.1    3           0.75            1                250    0.8439682

```

##	0.1	3	0.75	3	150	0.8468363	
##	0.1	3	0.75	3	250	0.8431086	
##	0.1	6	0.50	1	150	0.8381922	
##	0.1	6	0.50	1	250	0.8345700	
##	0.1	6	0.50	3	150	0.8408294	
##	0.1	6	0.50	3	250	0.8370225	
##	0.1	6	0.75	1	150	0.8393807	
##	0.1	6	0.75	1	250	0.8360153	
##	0.1	6	0.75	3	150	0.8417988	
##	0.1	6	0.75	3	250	0.8367599	
##	0.5	3	0.50	1	150	0.8217919	
##	0.5	3	0.50	1	250	0.8195591	
##	0.5	3	0.50	3	150	0.8250680	
##	0.5	3	0.50	3	250	0.8197260	
##	0.5	3	0.75	1	150	0.8262211	
##	0.5	3	0.75	1	250	0.8205599	
##	0.5	3	0.75	3	150	0.8252410	
##	0.5	3	0.75	3	250	0.8219665	
##	0.5	6	0.50	1	150	0.8188519	
##	0.5	6	0.50	1	250	0.8176220	
##	0.5	6	0.50	3	150	0.8186569	
##	0.5	6	0.50	3	250	0.8172406	
##	0.5	6	0.75	1	150	0.8236683	
##	0.5	6	0.75	1	250	0.8224824	
##	0.5	6	0.75	3	150	0.8222595	
##	0.5	6	0.75	3	250	0.8208097	
##	Accuracy.Accuracy						
##	0.6912985						
##	0.6847796						
##	0.6894336						
##	0.6822932						
##	0.6897336						
##	0.6838339						
##	0.6906768						
##	0.6757671						
##	0.6798121						
##	0.6748388						
##	0.6844541						
##	0.6835181						
##	0.6822850						
##	0.6813548						
##	0.6816735						
##	0.6723572						
##	0.6608824						
##	0.6512598						
##	0.6577878						
##	0.6506431						
##	0.6624390						
##	0.6565451						
##	0.6500224						
##	0.6528016						
##	0.6692530						
##	0.6658393						
##	0.6534309						

```

## 0.6534285
## 0.6676983
## 0.6661431
## 0.6608906
## 0.6574677
##
## Tuning parameter 'gamma' was held constant at a value of 0
## Tuning
## parameter 'subsample' was held constant at a value of 0.75
## AUC.AUC was used to select the optimal model using the largest value.
## The final values used for the model were nrounds = 150, max_depth = 3, eta
## = 0.1, gamma = 0, colsample_bytree = 0.5, min_child_weight = 1 and subsample
## = 0.75.

print(classification_models$XGB$results)

##     eta max_depth gamma colsample_bytree min_child_weight subsample nrounds
## 1  0.1          3     0            0.50              1       0.75    150
## 3  0.1          3     0            0.50              3       0.75    150
## 5  0.1          3     0            0.75              1       0.75    150
## 7  0.1          3     0            0.75              3       0.75    150
## 17 0.5          3     0            0.50              1       0.75    150
## 19 0.5          3     0            0.50              3       0.75    150
## 21 0.5          3     0            0.75              1       0.75    150
## 23 0.5          3     0            0.75              3       0.75    150
## 9  0.1          6     0            0.50              1       0.75    150
## 11 0.1          6     0            0.50              3       0.75    150
## 13 0.1          6     0            0.75              1       0.75    150
## 15 0.1          6     0            0.75              3       0.75    150
## 25 0.5          6     0            0.50              1       0.75    150
## 27 0.5          6     0            0.50              3       0.75    150
## 29 0.5          6     0            0.75              1       0.75    150
## 31 0.5          6     0            0.75              3       0.75    150
## 2  0.1          3     0            0.50              1       0.75    250
## 4  0.1          3     0            0.50              3       0.75    250
## 6  0.1          3     0            0.75              1       0.75    250
## 8  0.1          3     0            0.75              3       0.75    250
## 18 0.5          3     0            0.50              1       0.75    250
## 20 0.5          3     0            0.50              3       0.75    250
## 22 0.5          3     0            0.75              1       0.75    250
## 24 0.5          3     0            0.75              3       0.75    250
## 10 0.1          6     0            0.50              1       0.75    250
## 12 0.1          6     0            0.50              3       0.75    250
## 14 0.1          6     0            0.75              1       0.75    250
## 16 0.1          6     0            0.75              3       0.75    250
## 26 0.5          6     0            0.50              1       0.75    250
## 28 0.5          6     0            0.50              3       0.75    250
## 30 0.5          6     0            0.75              1       0.75    250
## 32 0.5          6     0            0.75              3       0.75    250
##          AUC.AUC Accuracy.Accuracy      AUC.AUCSD Accuracy.AccuracySD
## 1  0.8471965      0.6912985  0.004780348      0.019791458
## 3  0.8471562      0.6894336  0.005052652      0.015809634
## 5  0.8470788      0.6897336  0.005852254      0.012029837
## 7  0.8468363      0.6906768  0.006042619      0.015507421

```

```

## 17 0.8217919      0.6608824 0.014804834      0.015264801
## 19 0.8250680      0.6577878 0.008582047      0.017597665
## 21 0.8262211      0.6624390 0.014152165      0.012459426
## 23 0.8252410      0.6500224 0.012517161      0.015621568
## 9   0.8381922      0.6798121 0.006419690      0.012455717
## 11 0.8408294      0.6844541 0.006276105      0.015325563
## 13 0.8393807      0.6822850 0.005796057      0.010174891
## 15 0.8417988      0.6816735 0.008982473      0.010745674
## 25 0.8188519      0.6692530 0.015635727      0.012522169
## 27 0.8186569      0.6534309 0.010288992      0.013364202
## 29 0.8236683      0.6676983 0.011311210      0.010047824
## 31 0.8222595      0.6608906 0.011824892      0.018504329
## 2   0.8441051      0.6847796 0.005979236      0.013288417
## 4   0.8448742      0.6822932 0.005928244      0.015165941
## 6   0.8439682      0.6838339 0.007620035      0.010445429
## 8   0.8431086      0.6757671 0.006116087      0.011563283
## 18 0.8195591      0.6512598 0.014337442      0.018293686
## 20 0.8197260      0.6506431 0.010386932      0.009231630
## 22 0.8205599      0.6565451 0.013958231      0.016378643
## 24 0.8219665      0.6528016 0.012410840      0.015498555
## 10 0.8345700      0.6748388 0.008216414      0.008783153
## 12 0.8370225      0.6835181 0.008144093      0.012842109
## 14 0.8360153      0.6813548 0.007142089      0.010796819
## 16 0.8367599      0.6723572 0.010716422      0.014199800
## 26 0.8176220      0.6658393 0.015616427      0.011562755
## 28 0.8172406      0.6534285 0.011821935      0.016198839
## 30 0.8224824      0.6661431 0.012772905      0.011850953
## 32 0.8208097      0.6574677 0.012503777      0.016906064

```

```
cat("\nModified plot_roc_curves function to return a ggplot object\n")
```

```

## 
## Modified plot_roc_curves function to return a ggplot object

plot_roc_curves_gg <- function(model, test_data, model_name) {
  pred_probs_class <- predict(model, newdata = test_data %>% select(-popularity_level), type = "prob")
  class_levels <- levels(test_data$popularity_level)
  roc_objects_class <- list()

  for (i in seq_along(class_levels)) {
    current_class <- class_levels[i]
    binary_response <- factor(ifelse(test_data$popularity_level == current_class, 1, 0), levels = c(0, 1))
    predictor <- pred_probs_class[, current_class]
    roc_objects_class[[current_class]] <- roc(response = binary_response, predictor = predictor)
  }

  roc_data_list <- lapply(names(roc_objects_class), function(class_name) {
    roc_obj <- roc_objects_class[[class_name]]
    data.frame(
      FPR = 1 - roc_obj$specificities,
      TPR = roc_obj$sensitivities,
      Class = class_name
    )
  })
}
```

```

    })

roc_data_df <- bind_rows(roc_data_list)

auc_values <- sapply(roc_objects_class, auc)
auc_labels <- data.frame(
  Class = names(auc_values),
  AUC = round(auc_values, 3)
)

roc_plot <- ggplot(roc_data_df, aes(x = FPR, y = TPR, color = Class)) +
  geom_line(size = 1) +
  geom_abline(linetype = "dashed", color = "gold") +
  geom_text(data = auc_labels,
            aes(x = 0.7, y = 0.1 + (match(Class, class_levels) - 1) * 0.08,
                label = paste0("AUC (", Class, ") = ", AUC)),
            color = "white", size = 4, hjust = 0) +
  labs(title = paste("One-vs-Rest ROC Curves (", model_name, ")"),
       x = "False Positive Rate (1 - Specificity)",
       y = "True Positive Rate (Sensitivity)") +
  theme_dark_custom() +
  scale_color_manual(values = c("very_low" = "green", "low" = "blue", "high" = "orange", "very_high" =
  theme(legend.position = "bottom"))

return(roc_plot)
}

cat("\nGenerate ROC plots for both models\n")

##  

## Generate ROC plots for both models

roc_rf_plot <- plot_roc_curves_gg(classification_models$R.F, test_data_class, "Random Forest")

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

## 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.

roc_xgb_plot <- plot_roc_curves_gg(classification_models$XGB, test_data_class, "XGBoost")

```

```

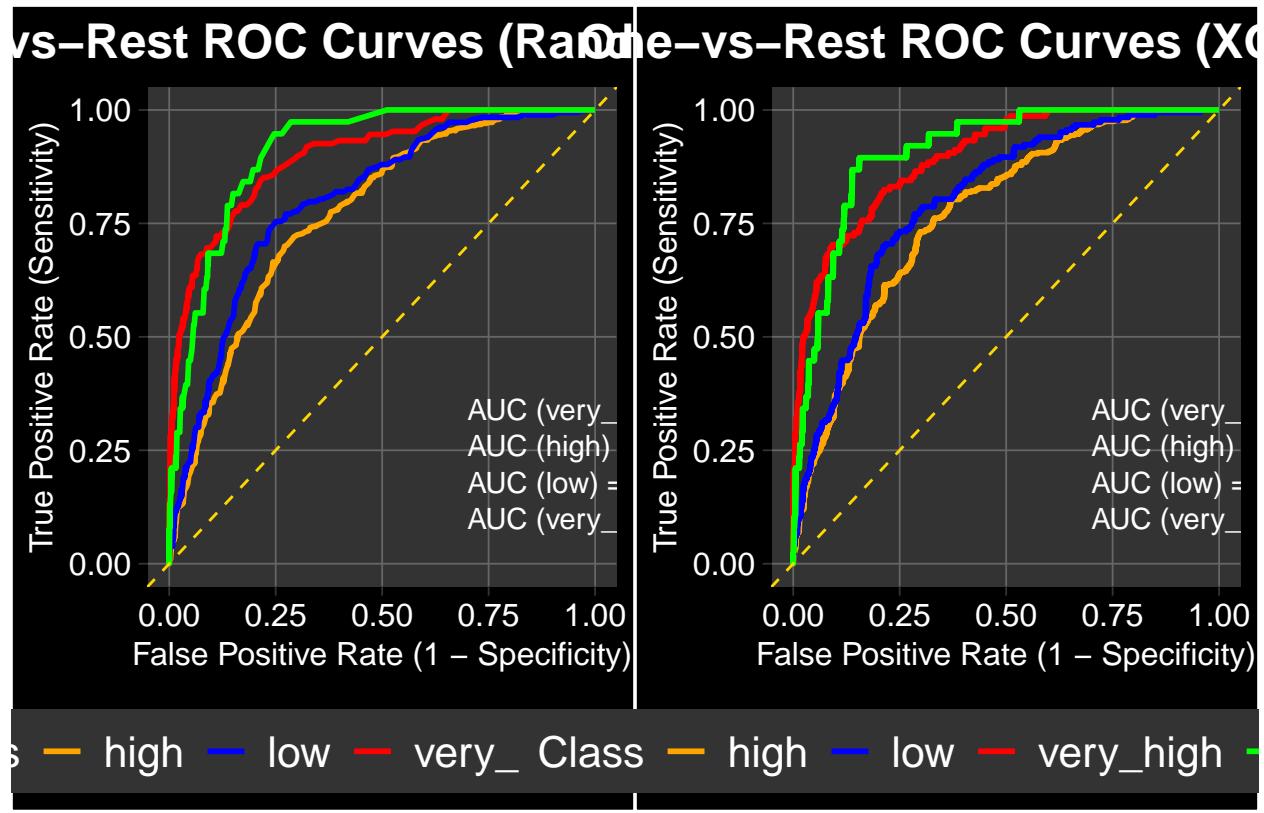
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: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

cat("\nArrange ROC curves side-by-side\n")

## 
## Arrange ROC curves side-by-side

grid.arrange(roc_rf_plot, roc_xgb_plot, ncol = 2,
             top = textGrob("One-vs-Rest ROC Curves for Classification Models",
                            gp = gpar(col = "white", fontsize = 20, fontface = "bold")))

```



```

cat("\n=====\n")
## 
## =====

```

```

cat("Frequency Plots for Popularity Class Labels\n")

## Frequency Plots for Popularity Class Labels

cat("=====\n")

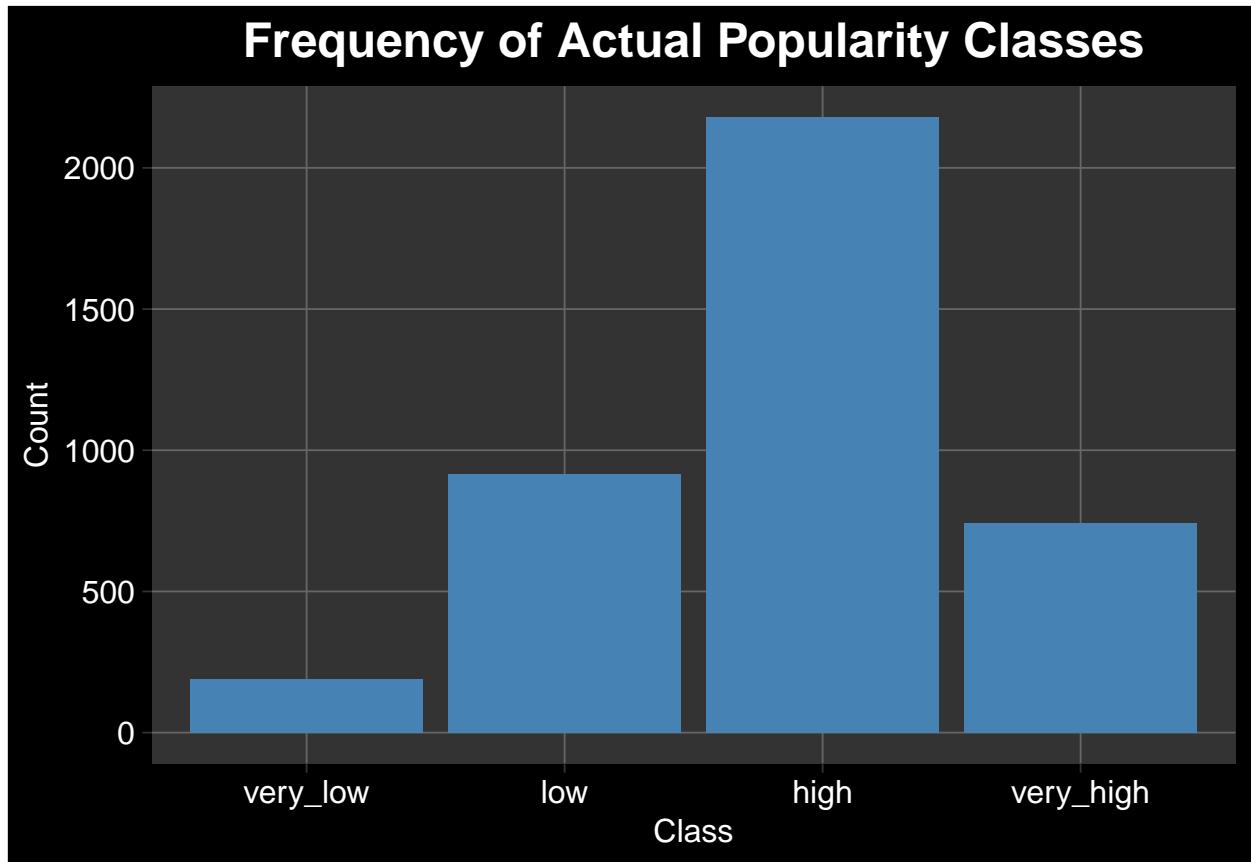
## =====

cat("\nActual popularity class distribution\n")

## 
## Actual popularity class distribution

actual_class_plot <- ggplot(data.frame(Class = actual_popularity_level), aes(x = Class)) +
  geom_bar(fill = "steelblue") +
  labs(title = "Frequency of Actual Popularity Classes", x = "Class", y = "Count") +
  theme_dark_custom()
print(actual_class_plot)

```



```

cat("\nPredicted popularity class distribution\n")

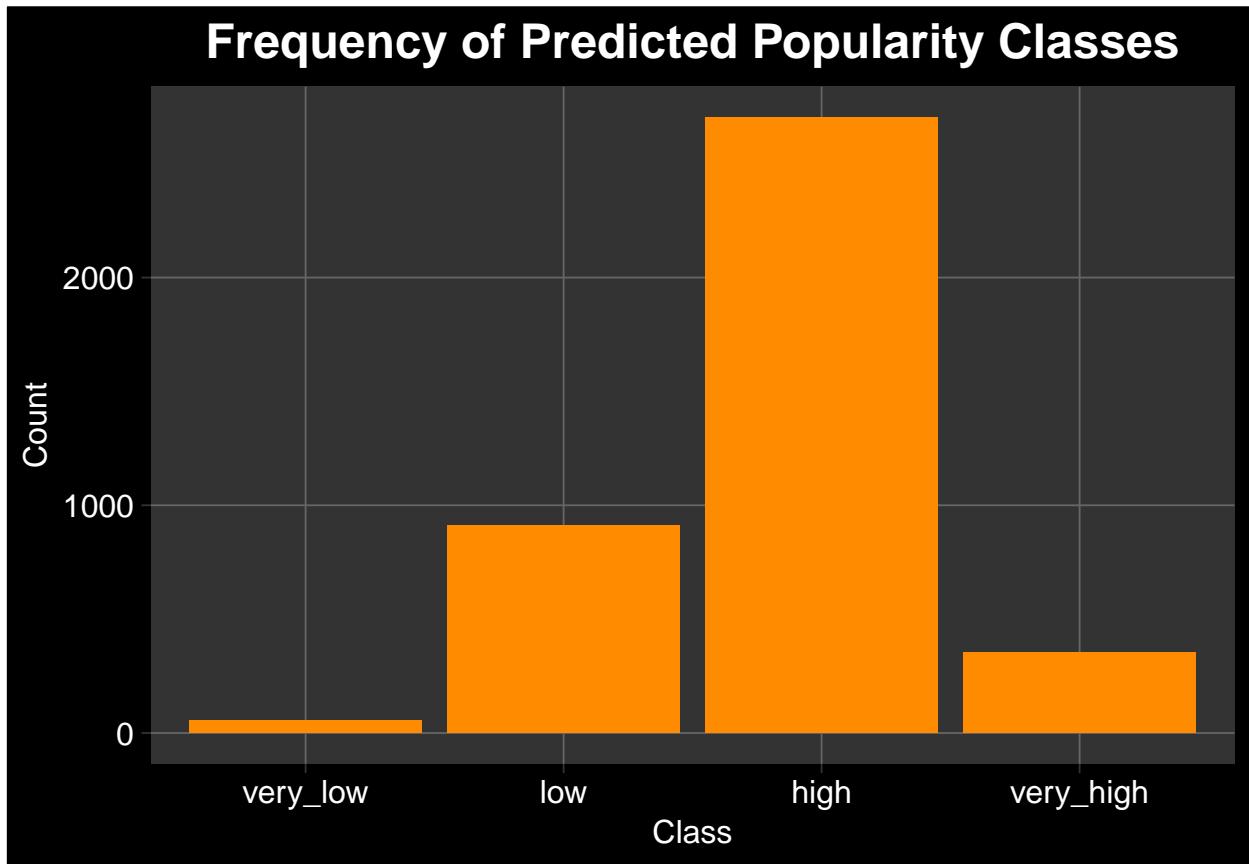
## 
## Predicted popularity class distribution

```

```

predicted_class_plot <- ggplot(data.frame(Class = predicted_popularity_level), aes(x = Class)) +
  geom_bar(fill = "darkorange") +
  labs(title = "Frequency of Predicted Popularity Classes", x = "Class", y = "Count") +
  theme_dark_custom()
print(predicted_class_plot)

```



```

cat("\n")
## -----
cat("Integrated Plot: Actual vs. Predicted Popularity Scores with Class Zones\n")
## Integrated Plot: Actual vs. Predicted Popularity Scores with Class Zones
cat("-----\n")
## -----
plot_data_actual_vs_predicted_with_predicted_classes <- data.frame(
  Actual_Popularity_Score = actual_popularity_for_classification_test_num,
  Predicted_Popularity_Score = predicted_numerical_popularity_for_classification_test,
  Predicted_Popularity_Class = predicted_popularity_level_for_plot
)

```

```

)

quadrant_data <- expand.grid(
  actual_class_label = c("very_low", "low", "high", "very_high"),
  predicted_class_label = c("very_low", "low", "high", "very_high")
) %>%
  mutate(
    xmin = case_when(
      actual_class_label == "very_low" ~ 0,
      actual_class_label == "low" ~ very_low_threshold,
      actual_class_label == "high" ~ high_threshold,
      actual_class_label == "very_high" ~ very_high_threshold
    ),
    xmax = case_when(
      actual_class_label == "very_low" ~ very_low_threshold,
      actual_class_label == "low" ~ high_threshold,
      actual_class_label == "high" ~ very_high_threshold,
      actual_class_label == "very_high" ~ 100
    ),
    ymin = case_when(
      predicted_class_label == "very_low" ~ 0,
      predicted_class_label == "low" ~ very_low_threshold,
      predicted_class_label == "high" ~ high_threshold,
      predicted_class_label == "very_high" ~ very_high_threshold
    ),
    ymax = case_when(
      predicted_class_label == "very_low" ~ very_low_threshold,
      predicted_class_label == "low" ~ high_threshold,
      predicted_class_label == "high" ~ very_high_threshold,
      predicted_class_label == "very_high" ~ 100
    ),
    fill_color = "gray30"
  )

actual_vs_predicted_with_colored_zones_plot <- ggplot(plot_data_actual_vs_predicted_with_predicted_class,
  aes(x = Actual_Popularity_Score,
      y = Predicted_Popularity_Score,
      color = Predicted_Popularity_Class)) +
  geom_rect(data = quadrant_data, aes(xmin = xmin, xmax = xmax, ymin = ymin, ymax = ymax),
            fill = "gray30", alpha = 0.5, inherit.aes = FALSE) +
  geom_point(alpha = 0.7, size = 2.5) +
  geom_abline(intercept = 0, slope = 1, color = "gold", linetype = "dashed", size = 1) +
  geom_vline(xintercept = very_low_threshold, linetype = "dotted", color = "orange", size = 0.8) +
  geom_vline(xintercept = high_threshold, linetype = "dotted", color = "blue", size = 0.8) +
  geom_vline(xintercept = very_high_threshold, linetype = "dotted", color = "green", size = 0.8) +
  geom_hline(yintercept = very_low_threshold, linetype = "dotted", color = "orange", size = 0.8) +
  geom_hline(yintercept = high_threshold, linetype = "dotted", color = "blue", size = 0.8) +
  geom_hline(yintercept = very_high_threshold, linetype = "dotted", color = "green", size = 0.8) +

```

```

annotate("text", x = 12.5, y = 98, label = "Actual:\nVery Low", color = "white", size = 3, hjust = 0.5)
annotate("text", x = 37.5, y = 98, label = "Actual:\nLow", color = "white", size = 3, hjust = 0.5, vjust = 0.5)
annotate("text", x = 62.5, y = 98, label = "Actual:\nHigh", color = "white", size = 3, hjust = 0.5, vjust = 0.5)
annotate("text", x = 87.5, y = 98, label = "Actual:\nVery High", color = "white", size = 3, hjust = 0.5, vjust = 0.5)

annotate("text", x = 2, y = 12.5, label = "Predicted:\nVery Low", color = "white", size = 3, angle = 90, hjust = 1)
annotate("text", x = 2, y = 37.5, label = "Predicted:\nLow", color = "white", size = 3, angle = 90, hjust = 1)
annotate("text", x = 2, y = 62.5, label = "Predicted:\nHigh", color = "white", size = 3, angle = 90, hjust = 1)
annotate("text", x = 2, y = 87.5, label = "Predicted:\nVery High", color = "white", size = 3, angle = 90, hjust = 1)

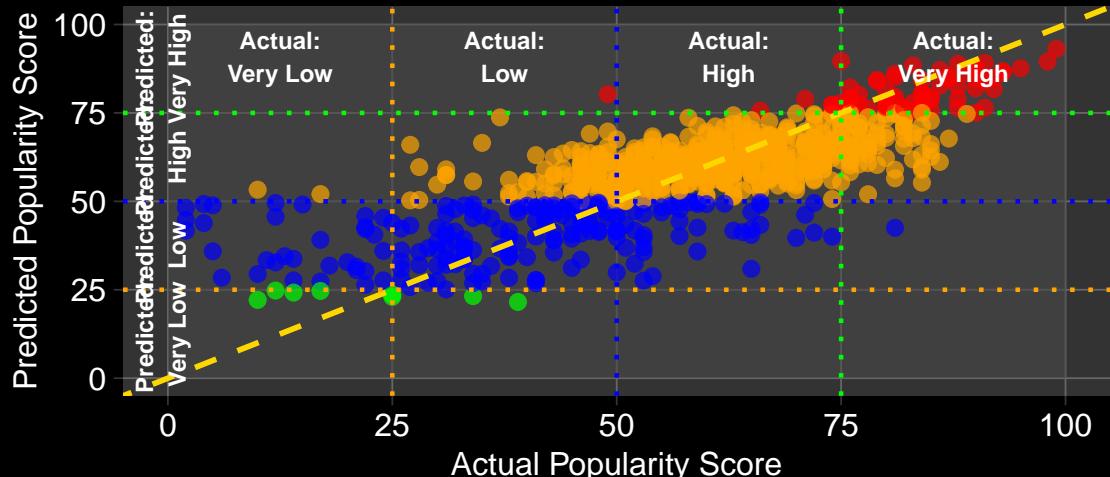
labs(
  title = "Actual vs. Predicted Popularity Scores (Classification Test Set)",
  subtitle = "Points colored by Predicted Popularity Class with Class Boundaries and Zones",
  x = "Actual Popularity Score",
  y = "Predicted Popularity Score",
  color = "Predicted Class"
) +
  xlim(0, 100) +
  ylim(0, 100) +
  scale_color_manual(values = c("very_low" = "green",
                                "low" = "blue",
                                "high" = "orange",
                                "very_high" = "red")) +
  theme_dark_custom() +
  theme(legend.position = "bottom",
        plot.margin = unit(c(1, 1, 1, 1), "cm"))

print(actual_vs_predicted_with_colored_zones_plot)

```

Actual vs. Predicted Popularity Scores (Classification Test)

Points colored by Predicted Popularity Class with Class Boundaries and Z



Predicted Class • very_low • low • high • very_high

```
cat("\n-----\n")  
##  
## -----  
  
cat("7. CONFUSION MATRIX AND CLASSIFICATION METRICS\n")  
  
## 7. CONFUSION MATRIX AND CLASSIFICATION METRICS  
  
cat("-----\n")  
## -----  
  
cat("\nConfusion Matrix and Classification Metrics\n")  
  
##  
## Confusion Matrix and Classification Metrics  
  
cat("\nFunction to evaluate and print classification metrics\n")  
  
##  
## Function to evaluate and print classification metrics
```

```

evaluate_classification_model <- function(model, test_data, model_name) {
  predictions_class <- predict(model, newdata = test_data %>% select(-popularity_level))

  conf_matrix_class <- confusionMatrix(data = predictions_class, reference = test_data$popularity_level)

  cat(paste("\nConfusion Matrix (", model_name, " on Test Set):\n", sep = ""))
  print(conf_matrix_class)

  overall_metrics_class <- data.frame(conf_matrix_class$overall)
  cat("\nOverall Classification Metrics:\n")
  print(overall_metrics_class)

  class_metrics_class <- data.frame(conf_matrix_class$byClass)
  cat("\nClass-Specific Classification Metrics:\n")
  print(class_metrics_class)

  return(list(confusion_matrix = conf_matrix_class,
              overall_metrics = overall_metrics_class,
              class_metrics = class_metrics_class))
}

cat("\nplot confusion matrices\n")

## 
## ggplot confusion matrices

plot_confusion_matrix_gg <- function(conf_matrix_obj, model_name) {
  cm_df <- as.data.frame(conf_matrix_obj$table)
  colnames(cm_df) <- c("Prediction", "Reference", "Freq")

  ggplot(cm_df, aes(x = Prediction, y = Reference, fill = Freq)) +
    geom_tile(color = "white") +
    geom_text(aes(label = Freq), color = "white", size = 5) +
    scale_fill_gradient(low = "skyblue", high = "blue") +
    labs(title = paste("Confusion Matrix - ", model_name),
         x = "Predicted Class", y = "Actual Class") +
    theme_dark_custom()
}

cat("\nEvaluate Random Forest and XGBoost\n")

## 
## Evaluate Random Forest and XGBoost

rf_evaluation_results <- evaluate_classification_model(classification_models$R.F, test_data_class, "Random Forest")

## 
## Confusion Matrix (Random Forest on Test Set):
## Confusion Matrix and Statistics
## 
##             Reference
## Prediction   very_low low high very_high

```

```

##   very_low      6   3   0      0
##   low          21  75  44      2
##   high         11 104 372     70
##   very_high     0   1   20     76
##
## Overall Statistics
##
##           Accuracy : 0.6571
##           95% CI : (0.6232, 0.6899)
##   No Information Rate : 0.5416
##   P-Value [Acc > NIR] : 1.859e-11
##
##           Kappa : 0.3904
##
## McNemar's Test P-Value : NA
##
## Statistics by Class:
##
##           Class: very_low Class: low Class: high Class: very_high
## Sensitivity          0.157895  0.40984   0.8532    0.51351
## Specificity          0.996089  0.89228   0.4986    0.96804
## Pos Pred Value       0.666667  0.52817   0.6679    0.78351
## Neg Pred Value       0.959799  0.83710   0.7419    0.89831
## Prevalence           0.047205  0.22733   0.5416    0.18385
## Detection Rate       0.007453  0.09317   0.4621    0.09441
## Detection Prevalence 0.011180  0.17640   0.6919    0.12050
## Balanced Accuracy    0.576992  0.65106   0.6759    0.74078
##
## Overall Classification Metrics:
##           conf_matrix_class.overall
## Accuracy           6.571429e-01
## Kappa              3.904343e-01
## AccuracyLower      6.232027e-01
## AccuracyUpper      6.899270e-01
## AccuracyNull       5.416149e-01
## AccuracyPValue     1.858858e-11
## McNemarPValue      NaN
##
## Class-Specific Classification Metrics:
##           Sensitivity Specificity Pos.Pred.Value Neg.Pred.Value
## Class: very_low    0.1578947  0.9960887   0.6666667  0.9597990
## Class: low          0.4098361  0.8922830   0.5281690  0.8371041
## Class: high         0.8532110  0.4986450   0.6678636  0.7419355
## Class: very_high    0.5135135  0.9680365   0.7835052  0.8983051
##           Precision   Recall      F1 Prevalence Detection.Rate
## Class: very_low    0.6666667  0.1578947  0.2553191  0.04720497  0.007453416
## Class: low          0.5281690  0.4098361  0.4615385  0.22732919  0.093167702
## Class: high         0.6678636  0.8532110  0.7492447  0.54161491  0.462111801
## Class: very_high    0.7835052  0.5135135  0.6204082  0.18385093  0.094409938
##           Detection.Prevalence Balanced.Accuracy
## Class: very_low     0.01118012   0.5769917
## Class: low           0.17639752   0.6510595
## Class: high          0.69192547   0.6759280
## Class: very_high     0.12049689   0.7407750

```

```

xgb_evaluation_results <- evaluate_classification_model(classification_models$XGB, test_data_class, "XGBoost")

## 
## Confusion Matrix (XGBoost on Test Set):
## Confusion Matrix and Statistics
## 
##             Reference
## Prediction   very_low low high very_high
##   very_low       5    6    0        0
##   low           26   71   44        0
##   high          7  105  372       71
##   very_high      0    1   20       77
## 
## Overall Statistics
## 
##           Accuracy : 0.6522
##           95% CI : (0.6181, 0.6851)
##   No Information Rate : 0.5416
##   P-Value [Acc > NIR] : 1.271e-10
## 
##           Kappa : 0.383
## 
## Mcnemar's Test P-Value : NA
## 
## Statistics by Class:
## 
##           Class: very_low Class: low Class: high Class: very_high
## Sensitivity            0.131579    0.3880    0.8532    0.52027
## Specificity             0.992177    0.8875    0.5041    0.96804
## Pos Pred Value          0.454545    0.5035    0.6703    0.78571
## Neg Pred Value          0.958438    0.8313    0.7440    0.89958
## Prevalence              0.047205    0.2273    0.5416    0.18385
## Detection Rate          0.006211    0.0882    0.4621    0.09565
## Detection Prevalence    0.013665    0.1752    0.6894    0.12174
## Balanced Accuracy       0.561878    0.6377    0.6786    0.74415
## 
## Overall Classification Metrics:
##           conf_matrix_class.overall
## Accuracy           6.521739e-01
## Kappa              3.830067e-01
## AccuracyLower      6.181408e-01
## AccuracyUpper      6.850877e-01
## AccuracyNull       5.416149e-01
## AccuracyPValue     1.270787e-10
## McnemarPValue      NaN
## 
## Class-Specific Classification Metrics:
##           Sensitivity Specificity Pos.Pred.Value Neg.Pred.Value
## Class: very_low    0.1315789   0.9921773     0.4545455    0.9584383
## Class: low         0.3879781   0.8874598     0.5035461    0.8313253
## Class: high        0.8532110   0.5040650     0.6702703    0.7440000
## Class: very_high   0.5202703   0.9680365     0.7857143    0.8995757
##           Precision    Recall      F1 Prevalence Detection.Rate

```

```

## Class: very_low 0.4545455 0.1315789 0.2040816 0.04720497 0.00621118
## Class: low 0.5035461 0.3879781 0.4382716 0.22732919 0.08819876
## Class: high 0.6702703 0.8532110 0.7507568 0.54161491 0.46211180
## Class: very_high 0.7857143 0.5202703 0.6260163 0.18385093 0.09565217
## Detection.Prevalence Balanced.Accuracy
## Class: very_low 0.0136646 0.5618781
## Class: low 0.1751553 0.6377190
## Class: high 0.6894410 0.6786380
## Class: very_high 0.1217391 0.7441534

cat("\nGet confusion matrix plots\n")

##
## Get confusion matrix plots

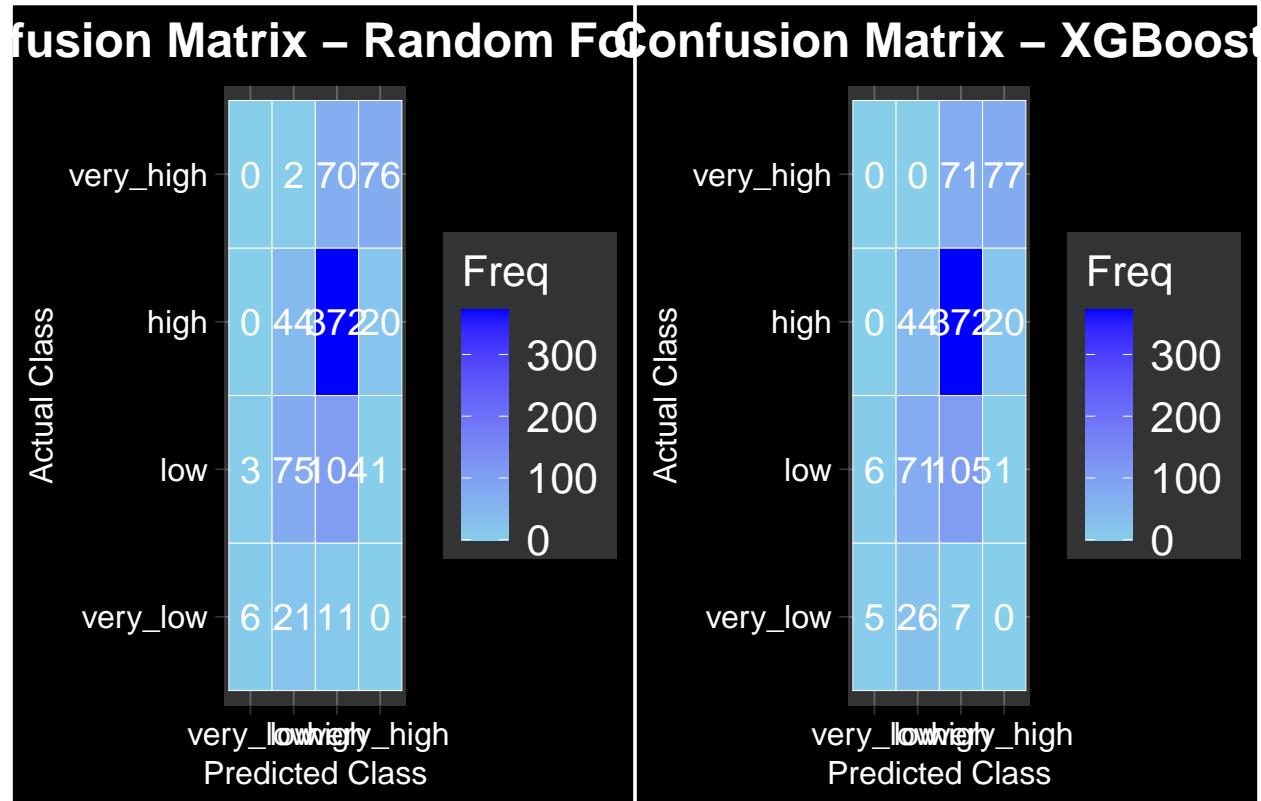
rf_cm_plot <- plot_confusion_matrix_gg(rf_evaluation_results$confusion_matrix, "Random Forest")
xgb_cm_plot <- plot_confusion_matrix_gg(xgb_evaluation_results$confusion_matrix, "XGBoost")

cat("\nArrange confusion matrices side-by-side\n")

##
## Arrange confusion matrices side-by-side

grid.arrange(rf_cm_plot, xgb_cm_plot, ncol = 2,
             top = textGrob("Confusion Matrices for Classification Models",
                            gp = gpar(col = "white", fontsize = 20, fontface = "bold")))

```



```

cat("\n=====\n")

## =====

cat("Average Model Accuracy Comparison\n")

## Average Model Accuracy Comparison

cat("=====\n")

## =====

cat("\nExtract overall accuracy for Random Forest\n")

## 
## Extract overall accuracy for Random Forest

rf_accuracy <- rf_evaluation_results$overall_metrics["Accuracy", ]
cat(paste0("Random Forest Model Accuracy: ", round(rf_accuracy, 4), "\n"))

## Random Forest Model Accuracy: 0.6571

cat("\nExtract overall accuracy for XGBoost\n")

## 
## Extract overall accuracy for XGBoost

xgb_accuracy <- xgb_evaluation_results$overall_metrics["Accuracy", ]
cat(paste0("XGBoost Model Accuracy: ", round(xgb_accuracy, 4), "\n"))

## XGBoost Model Accuracy: 0.6522

cat("\nCompare and highlight the best model\n")

## 
## Compare and highlight the best model

if (rf_accuracy > xgb_accuracy) {
  cat("\nRandom Forest shows higher overall accuracy.\n")
} else if (xgb_accuracy > rf_accuracy) {
  cat("\nXGBoost shows higher overall accuracy.\n")
} else {
  cat("\nBoth models have similar overall accuracy.\n")
}

## 
## Random Forest shows higher overall accuracy.

```

```

cat("\nVisualize accuracy comparison with a bar plot\n")

##  

## Visualize accuracy comparison with a bar plot

accuracy_comparison_df <- data.frame(  

  Model = c("Random Forest", "XGBoost"),  

  Accuracy = c(rf_accuracy, xgb_accuracy)  

)

accuracy_plot <- ggplot(accuracy_comparison_df, aes(x = Model, y = Accuracy, fill = Model)) +  

  geom_bar(stat = "identity", position = "dodge") +  

  geom_text(aes(label = round(Accuracy, 3)), vjust = -0.5, color = "white") +  

  labs(title = "Comparison of Average Model Accuracy",  

       x = "Classification Model",  

       y = "Accuracy") +  

  theme_dark_custom() +  

  scale_fill_manual(values = c("Random Forest" = "green", "XGBoost" = "orange"))

print(accuracy_plot)

```

