SLHW2

Ava Exelbirt, Sam Reade

2024-12-01

```r
# Install necessary libraries if not installed
# install.packages(c("ggplot2", "dplyr", "scales", "lubridate"))
# install.packages("caret")
# install.packages("GGally")
# install.packages("tidyverse")
#install.packages("randomForest")
```

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

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(scales)
library(lubridate)
```

```
## Warning: package 'lubridate' was built under R version 4.3.3

##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v forcats 1.0.0     v stringr 1.5.0
## v purrr   1.0.2     v tibble  3.2.1
## v readr   2.1.4     v tidyr   1.3.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x readr::col_factor() masks scales::col_factor()
## x purrr::discard()    masks scales::discard()
## 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
```

```r
library(GGally)
```

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

```r
library(caret)
```

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

```r
library(rpart)
library(rpart.plot)
library(nnet)
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 4.3.3

## Loaded gbm 2.2.2
## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.c
```

```r
library(MASS)
```

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

```r
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.3.3

## randomForest 4.7-1.2
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(pdp)
```

```
## Warning: package 'pdp' was built under R version 4.3.3

##
## Attaching package: 'pdp'
##
```

```
## The following object is masked from 'package:purrr':
##
##     partial
```

```r
library(fastshap)
```

```
##
## Attaching package: 'fastshap'
##
## The following object is masked from 'package:dplyr':
##
##     explain
```

About the Data

Import Data

```r
#tuesdata <- tidytuesdayR::tt_load('2022-11-01')
tuesdata <- tidytuesdayR::tt_load(2022, week = 44)
```

```
## ---- Compiling #TidyTuesday Information for 2022-11-01 ----
## --- There is 1 file available ---
##
##
## -- Downloading files ----------------------------------------------------------
##
##   1 of 1: "horror_movies.csv"
```

```r
horror <- tuesdata$horror_movies
```

```r
glimpse(horror)
```

```
## Rows: 32,540
## Columns: 20
## $ id                <dbl> 760161, 760741, 882598, 756999, 772450, 1014226, 717~
## $ original_title    <chr> "Orphan: First Kill", "Beast", "Smile", "The Black P~
## $ title             <chr> "Orphan: First Kill", "Beast", "Smile", "The Black P~
## $ original_language <chr> "en", "en", "en", "en", "es", "es", "en", "en", "en"~
## $ overview          <chr> "After escaping from an Estonian psychiatric facilit~
## $ tagline           <chr> "There's always been something wrong with Esther.", ~
## $ release_date      <date> 2022-07-27, 2022-08-11, 2022-09-23, 2022-06-22, 202~
## $ poster_path       <chr> "/pHkKbIRoCe7zIFvqan9LFSaQAde.jpg", "/xIGr7UHsKf0URW~
## $ popularity        <dbl> 5088.584, 2172.338, 1863.628, 1071.398, 1020.995, 93~
## $ vote_count        <dbl> 902, 584, 114, 2736, 83, 1, 125, 1684, 73, 1035, 637~
## $ vote_average      <dbl> 6.9, 7.1, 6.8, 7.9, 7.0, 1.0, 5.8, 7.0, 6.5, 6.8, 7.~
## $ budget            <dbl> 0, 0, 17000000, 18800000, 0, 0, 20000000, 68000000, ~
## $ revenue           <dbl> 9572765, 56000000, 45000000, 161000000, 0, 0, 289259~
## $ runtime           <dbl> 99, 93, 115, 103, 0, 0, 88, 130, 90, 106, 98, 89, 97~
## $ status            <chr> "Released", "Released", "Released", "Released", "Rel~
## $ adult             <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FAL~
## $ backdrop_path     <chr> "/5GA3vV1aWWHTSDO5eno8V5zDo8r.jpg", "/2k9tBql5GYH328~
## $ genre_names       <chr> "Horror, Thriller", "Adventure, Drama, Horror", "Hor~
## $ collection        <dbl> 760193, NA, NA, NA, NA, NA, 94899, NA, NA, 950289, N~
## $ collection_name   <chr> "Orphan Collection", NA, NA, NA, NA, NA, "Jeepers Cr~
```

Data Dictionary

1. The `id` variable is an integer that serves as a unique identifier for each movie.
2. The `original_title` variable is a character string representing the movie's original title.
3. The `title` variable is a character string containing the localized or alternative movie title.
4. The `original_language` variable is a character field indicating the language in which the movie was originally made.
5. The `overview` variable is a character field providing a brief description or synopsis of the movie.
6. The `tagline` variable is a character field capturing the movie's catchphrase or slogan.
7. The `release_date` variable is a date field that records the date when the movie was first released.
8. The `poster_path` variable is a character field containing the URL to the movie's poster image.
9. The `popularity` variable is a numerical value representing the movie's popularity score based on audience interactions.
10. The `vote_count` variable is an integer field that records the total number of audience votes received.
11. The `vote_average` variable is a numerical field that represents the average audience rating on a scale from 0 to 10.
12. The `budget` variable is an integer field capturing the movie's production budget in USD.
13. The `revenue` variable is an integer field indicating the total revenue earned by the movie in USD.
14. The `runtime` variable is an integer field that specifies the duration of the movie in minutes.
15. The `status` variable is a character field that indicates the current status of the movie, such as "Released."
16. The `adult` variable is a boolean that indicates whether the movie is intended for adult audiences.
17. The `backdrop_path` variable is a character field that provides the URL to the backdrop image for the movie.
18. The `genre_names` variable is a character field listing the genres associated with the movie, separated by commas.
19. The `collection` variable is a numerical field containing the unique ID of the collection the movie belongs to, which may be null for movies not part of a collection.
20. The `collection_name` variable is a character field representing the name of the collection, which may also be null if the movie does not belong to one.

Available Data

The dataset contains detailed information on a wide range of horror movies, about ~35,000 pieces of entertainment, including various features such as title, genre, release date, runtime, popularity, budget, and revenue. Additional details include the movie's runtime, vote count, average vote, genre names, and collection association. Notably, the dataset also contains the poster and backdrop image URLs for each movie, as well as whether the movie is intended for adults. These data points provide a comprehensive view of each movie's performance, reception, and thematic elements, enabling further analysis on trends, movie popularity, and financial success within the horror genre. These features will be used to train a classification model to predict whether each entry is a successful movie or not. The objective is to leverage these data points to build an accurate classification model, focusing on identifying the key predictors that contribute most to the classification process.

Motivation

As the entertainment industry expands, identifying the success of a movie is critical to content platforms and production companies. Success in the entertainment industry is typically measured by revenue, budget, and audience reception. Predicting whether a movie is likely to be successful or not can help guide investment decisions, optimize content strategies, and improve user recommendations. However, accurately predicting success is a challenge due to the multifaceted nature of what contributes to a movie's success, including budget, genre, release time, and audience engagement factors.

In this context, predicting a movie's success involves analyzing historical data and identifying patterns that correlate with positive outcomes. By doing so, production teams and platforms can better allocate resources, strategize marketing efforts, and predict the potential success of future movies. The motivation behind this project is to build a classification model that can predict whether a movie will be successful based on various features, thus improving decision-making processes in the entertainment industry.

Goal

The primary goal of this project is to develop a classification model that predicts whether a given movie is successful or not. The project will focus on feature selection, model interpretation, and the comparison of predictor sets to determine the most significant factors contributing to a movie's success. By analyzing a variety of features such as budget, revenue, genre, and popularity, the goal is to build a model that classifies movies as "successful" or "unsuccessful" with high accuracy. This will allow content platforms and production companies to make data-driven decisions and better understand the elements that contribute to the success of a movie.

Data Preprocessing and Visualization Tools

```r
summary(horror)
```

```
##        id          original_title        title           original_language
##  Min.   :     17   Length:32540       Length:32540       Length:32540
##  1st Qu.: 146495   Class :character   Class :character   Class :character
##  Median : 426521   Mode  :character   Mode  :character   Mode  :character
##  Mean   : 445911
##  3rd Qu.: 707534
##  Max.   :1033095
##
##    overview           tagline           release_date        poster_path
##  Length:32540       Length:32540       Min.   :1950-01-01   Length:32540
##  Class :character   Class :character   1st Qu.:2000-10-20   Class :character
##  Mode  :character   Mode  :character   Median :2012-12-09   Mode  :character
##                                        Mean   :2007-02-18
##                                        3rd Qu.:2018-10-03
##                                        Max.   :2022-12-31
##
##    popularity         vote_count         vote_average        budget
##  Min.   :   0.000   Min.   :    0.00   Min.   : 0.000   Min.   :        0
##  1st Qu.:   0.600   1st Qu.:    0.00   1st Qu.: 0.000   1st Qu.:        0
##  Median :   0.840   Median :    2.00   Median : 4.000   Median :        0
##  Mean   :   4.013   Mean   :   62.69   Mean   : 3.336   Mean   :   543127
##  3rd Qu.:   2.243   3rd Qu.:   11.00   3rd Qu.: 5.700   3rd Qu.:        0
##  Max.   :5088.584   Max.   :16900.00   Max.   :10.000   Max.   :200000000
##
##     revenue             runtime          status             adult
##  Min.   :        0   Min.   :  0.00   Length:32540       Mode :logical
##  1st Qu.:        0   1st Qu.: 14.00   Class :character   FALSE:32540
##  Median :        0   Median : 80.00   Mode  :character
##  Mean   :  1349747   Mean   : 62.14
##  3rd Qu.:        0   3rd Qu.: 91.00
##  Max.   :701842551   Max.   :683.00
##
##  backdrop_path       genre_names          collection       collection_name
##  Length:32540       Length:32540       Min.   :    656   Length:32540
##  Class :character   Class :character   1st Qu.: 155421   Class :character
##  Mode  :character   Mode  :character   Median : 471259   Mode  :character
```

```
##                                         Mean   : 481535
##                                         3rd Qu.: 759067
##                                         Max.   :1033032
##                                         NA's   :30234
```

Data Cleanup

Handling NA Values

We will look at how many NA values are in each column to better understand our data set.

```r
na_counts <- colSums(is.na(horror))

print(na_counts)
```

```
##                id    original_title             title original_language
##                 0                 0                 0                 0
##          overview           tagline      release_date       poster_path
##              1286             19833                 0              4474
##        popularity        vote_count      vote_average            budget
##                 0                 0                 0                 0
##           revenue           runtime            status             adult
##                 0                 0                 0                 0
##     backdrop_path       genre_names        collection   collection_name
##             18995                 0             30234             30234
```

```r
na_counts_df <- data.frame(Column = names(na_counts), NA_Count = na_counts)
print(na_counts_df)
```

```
##                              Column NA_Count
## id                               id        0
## original_title       original_title        0
## title                         title        0
## original_language original_language        0
## overview                   overview     1286
## tagline                     tagline    19833
## release_date           release_date        0
## poster_path             poster_path     4474
## popularity               popularity        0
## vote_count               vote_count        0
## vote_average           vote_average        0
## budget                       budget        0
## revenue                     revenue        0
## runtime                     runtime        0
## status                       status        0
## adult                         adult        0
## backdrop_path         backdrop_path    18995
## genre_names             genre_names        0
## collection               collection    30234
## collection_name     collection_name    30234
```

```r
sum(horror$revenue == 0)
```

```
## [1] 30964
```

```r
sum(horror$budget == 0)
```

```
## [1] 27339
```

```r
sum(horror$budget != 0 & horror$revenue != 0)
```

```
## [1] 1098
```

```r
sum(horror$budget == 0 & horror$revenue == 0)
```

```
## [1] 26861
```

For numeric columns, we will fill missing values with the median values of that column. These include id, release_date, popularity, vote_count, vote_average, revenue, and runtime. We will then fill missing character columns with "Unknown." These include original_title, title, original_language, tagline, overview, poster_path, status, adult, and backdrop_path.

```r
numeric_cols <- sapply(horror, is.numeric)
horror[numeric_cols] <- lapply(horror[numeric_cols], function(x) {
  ifelse(is.na(x), median(x, na.rm = TRUE), x)
})

## Fill missing character columns with "Unknown"
char_cols <- sapply(horror, is.character)
horror[char_cols] <- lapply(horror[char_cols], function(x) {
  ifelse(is.na(x), "Unknown", x)
})
```

Drop Columns

We will remove the columns ids and paths as these are not needed for our overall analysis.

```r
library(dplyr)

# Drop the specified columns
#horror <- horror |> select(-id, -poster_path, -backdrop_path, -collection, -collection_name)

horror <- dplyr::select(horror, -id, -poster_path, -backdrop_path, -collection, -collection_name)
```

Feature Engineering

As part of feature engineering we need to create our boolean-like columns to logical data types. We will do so for the adult column. If the observation is FALSE, then it will convert to a logical operator of 0. If the observation is TRUE for this column, then it will be converted to 1. We must also convert categorical columns to factors. This includes original_language, status, and genre_names. Finally, we will extract year from release_date because this will help in further analysis.

```r
horror$adult <- as.logical(horror$adult)

categorical_cols <- c("original_language", "status", "genre_names")
horror[categorical_cols] <- lapply(horror[categorical_cols], as.factor)

horror$release_year <- as.numeric(substr(horror$release_date, 1, 4))
```

Handling Outliers

We will replace some outliers. Specifically, for runtime we will replace runtime with the 0 if there is a runtime that is defined as an outlier, we will replace it with 0. We will also remove rows with outliers regarding popularity that is defined as popularity above 10000. We will also categorize budget levels. We categorize movies into "Low", "Medium", or "High" budget based on the budget column:

```r
runQ1 <- quantile(horror$runtime, 0.25, na.rm = TRUE)
runQ3 <- quantile(horror$runtime, 0.75, na.rm = TRUE)
IQR <- runQ3 - runQ1
lower_bound <- runQ1 - 1.5 * IQR
upper_bound <- runQ3 + 1.5 * IQR
horror$runtime[horror$runtime < lower_bound | horror$runtime > upper_bound] <- 0

#horror$runtime[horror$runtime <= 0 | horror$runtime > 300] <- NA

horror <- horror[!(horror$popularity > 10000), ]

horror$budget_category <- ifelse(horror$budget == 0, "No Budget",
                           ifelse(horror$budget < 1e7, "Low",
                           ifelse(horror$budget < 5e7, "Medium", "High")))
```

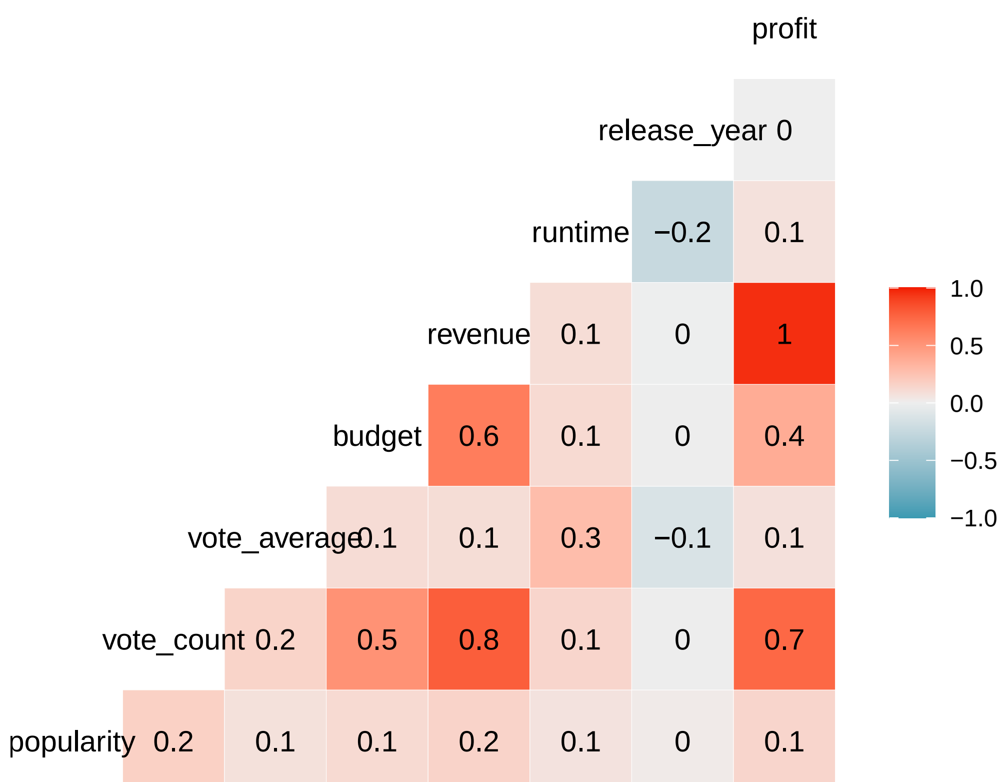
Create Target Variables

We first create a profit variable that is the revenue minus budget of a movie. We then create a success variable: if profit > 0, movie is considered successful

```r
horror$profit <- horror$revenue - horror$budget
horror$success <- ifelse(horror$profit > 0, "Success", "No Success")
```

Correlation Analysis

```r
ggcorr(horror[ , sapply(horror, is.numeric)], label = TRUE)
```

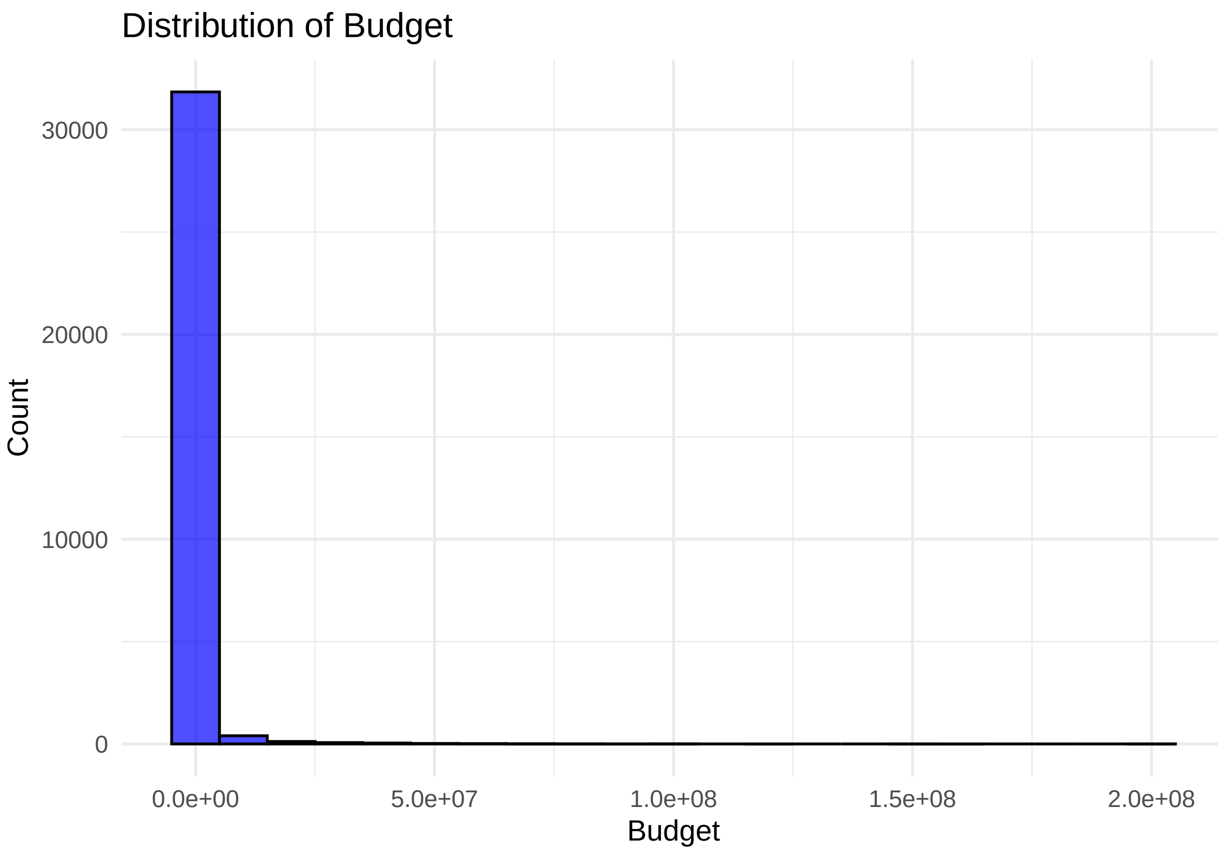


Visualization Tools

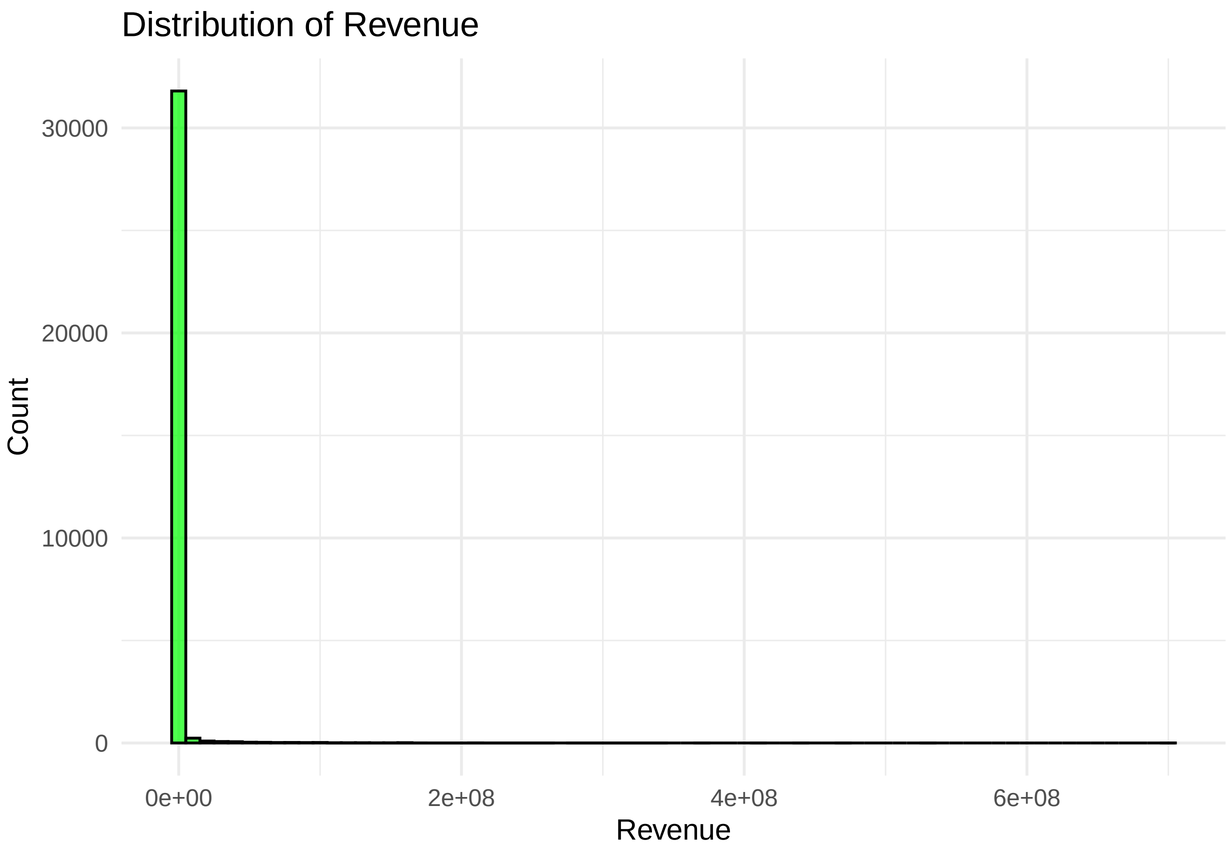
Distribution of Numeric Features

We will plot the distributions of numeric features, specifically budget, revenue, and runtime.

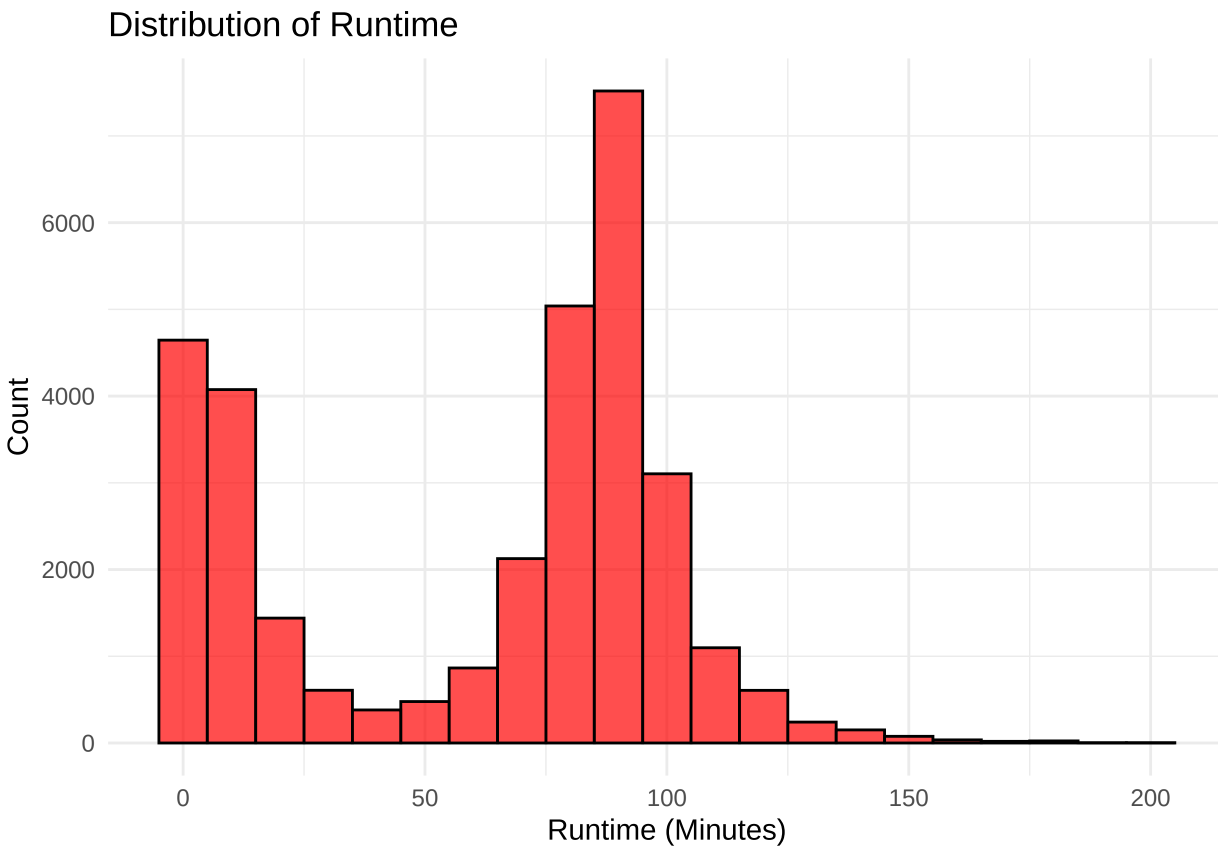
```r
ggplot(horror, aes(x = budget)) +
  geom_histogram(binwidth = 1e7, fill = "blue", color = "black", alpha = 0.7) +
  labs(title = "Distribution of Budget", x = "Budget", y = "Count") +
  theme_minimal()
```



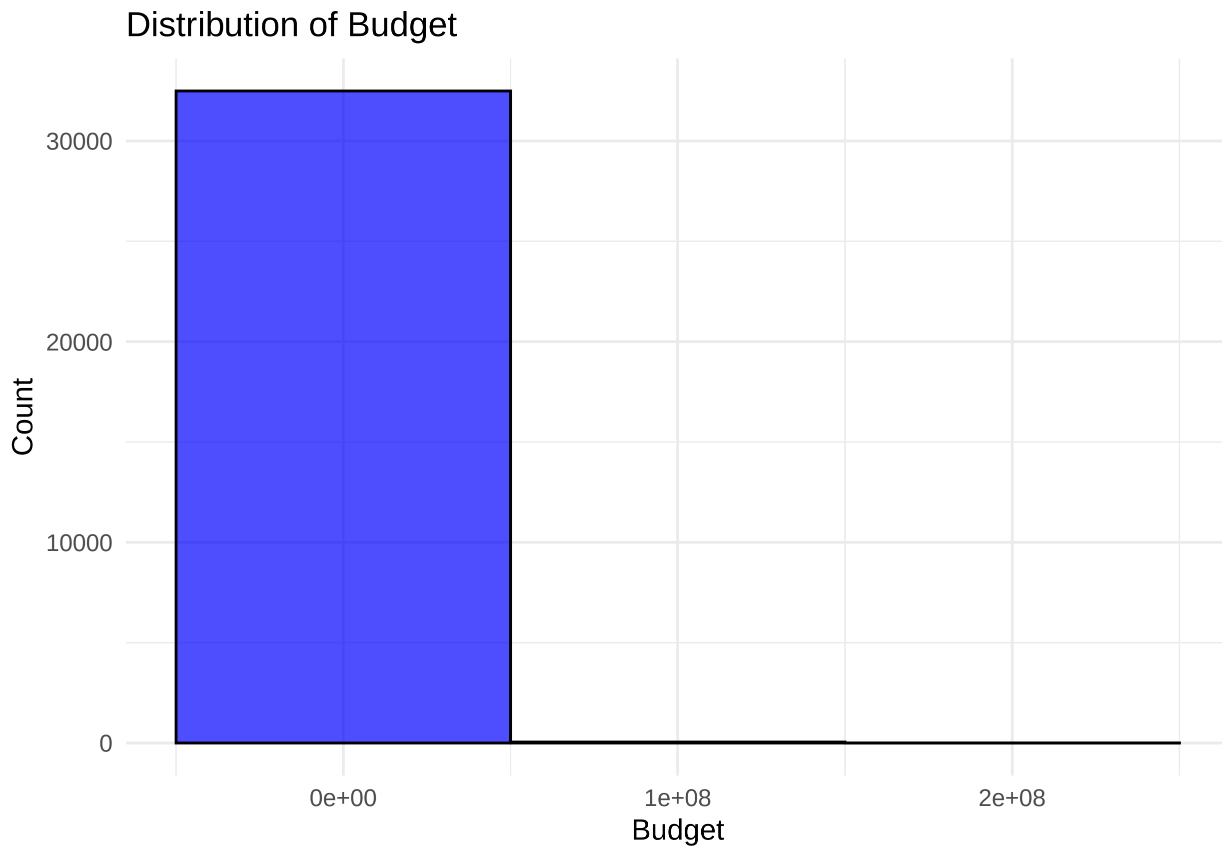

```r
ggplot(horror, aes(x = revenue)) +
  geom_histogram(binwidth = 1e7, fill = "green", color = "black", alpha = 0.7) +
  labs(title = "Distribution of Revenue", x = "Revenue", y = "Count") +
  theme_minimal()
```



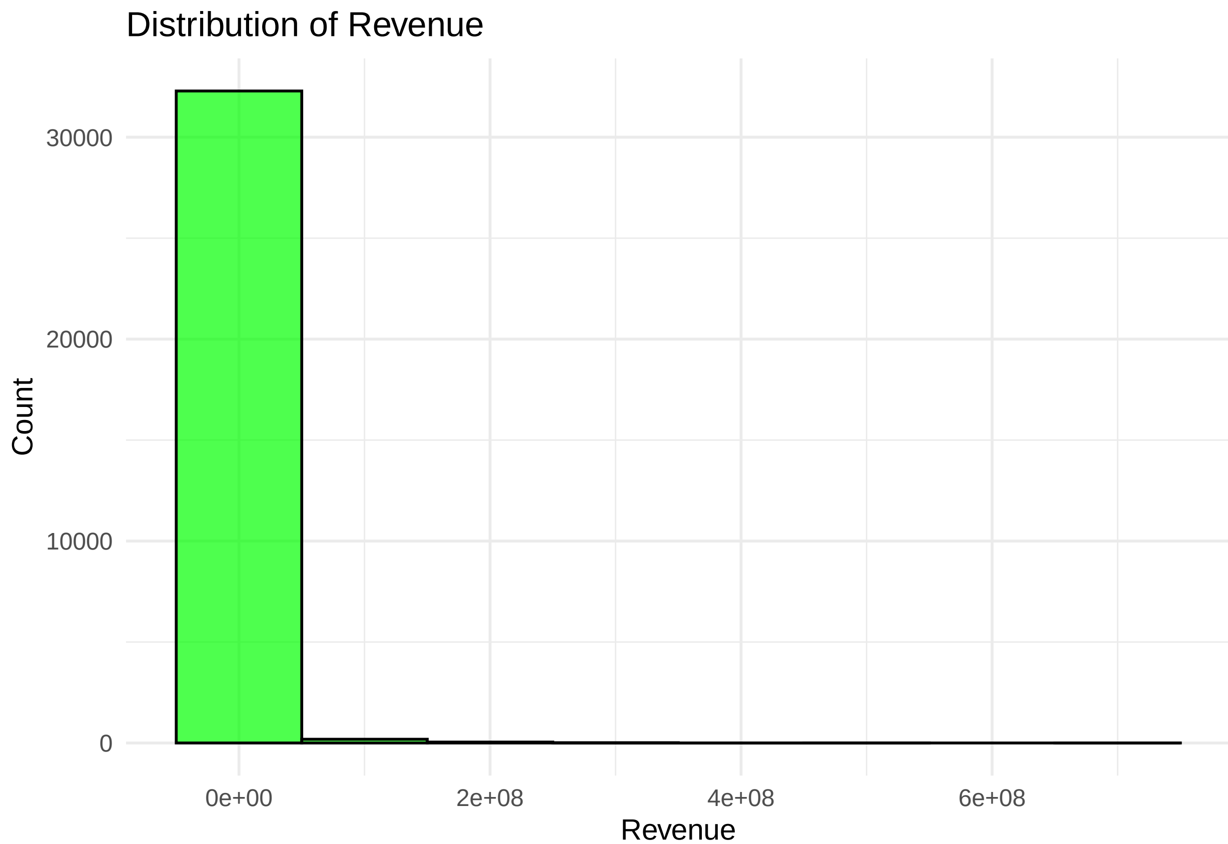

```r
ggplot(horror, aes(x = runtime)) +
  geom_histogram(binwidth = 10, fill = "red", color = "black", alpha = 0.7) +
  labs(title = "Distribution of Runtime", x = "Runtime (Minutes)", y = "Count") +
  theme_minimal()
```



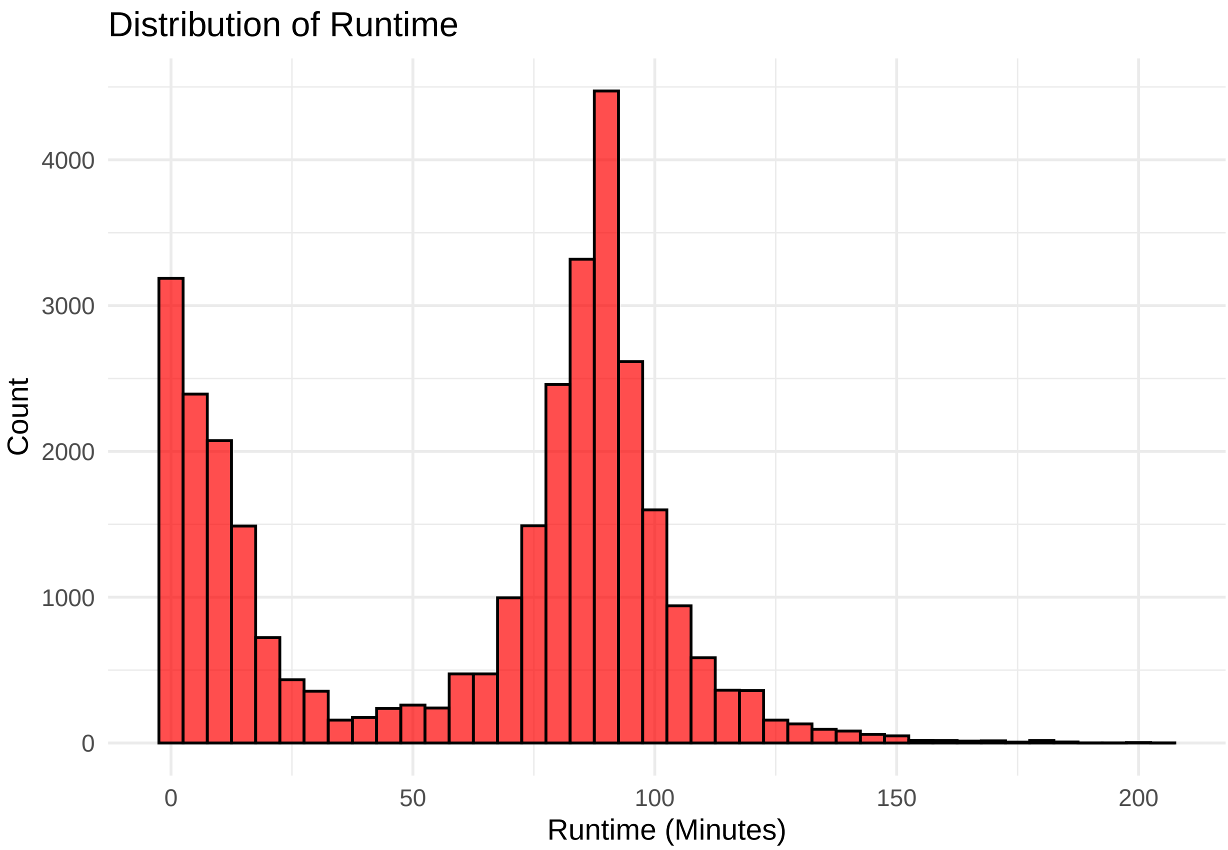

```r
# For Budget (which has large values)
ggplot(horror, aes(x = budget)) +
  geom_histogram(binwidth = 1e8, fill = "blue", color = "black", alpha = 0.7) +  # Increase binwidth
  labs(title = "Distribution of Budget", x = "Budget", y = "Count") +
  theme_minimal()
```




```r
# For Revenue (which also has large values)
ggplot(horror, aes(x = revenue)) +
  geom_histogram(binwidth = 1e8, fill = "green", color = "black", alpha = 0.7) +  # Increase binwidth
  labs(title = "Distribution of Revenue", x = "Revenue", y = "Count") +
  theme_minimal()
```




```r
# For Runtime (which typically has smaller values)
ggplot(horror, aes(x = runtime)) +
  geom_histogram(binwidth = 5, fill = "red", color = "black", alpha = 0.7) +  # Adjust binwidth for run
  labs(title = "Distribution of Runtime", x = "Runtime (Minutes)", y = "Count") +
  theme_minimal()
```

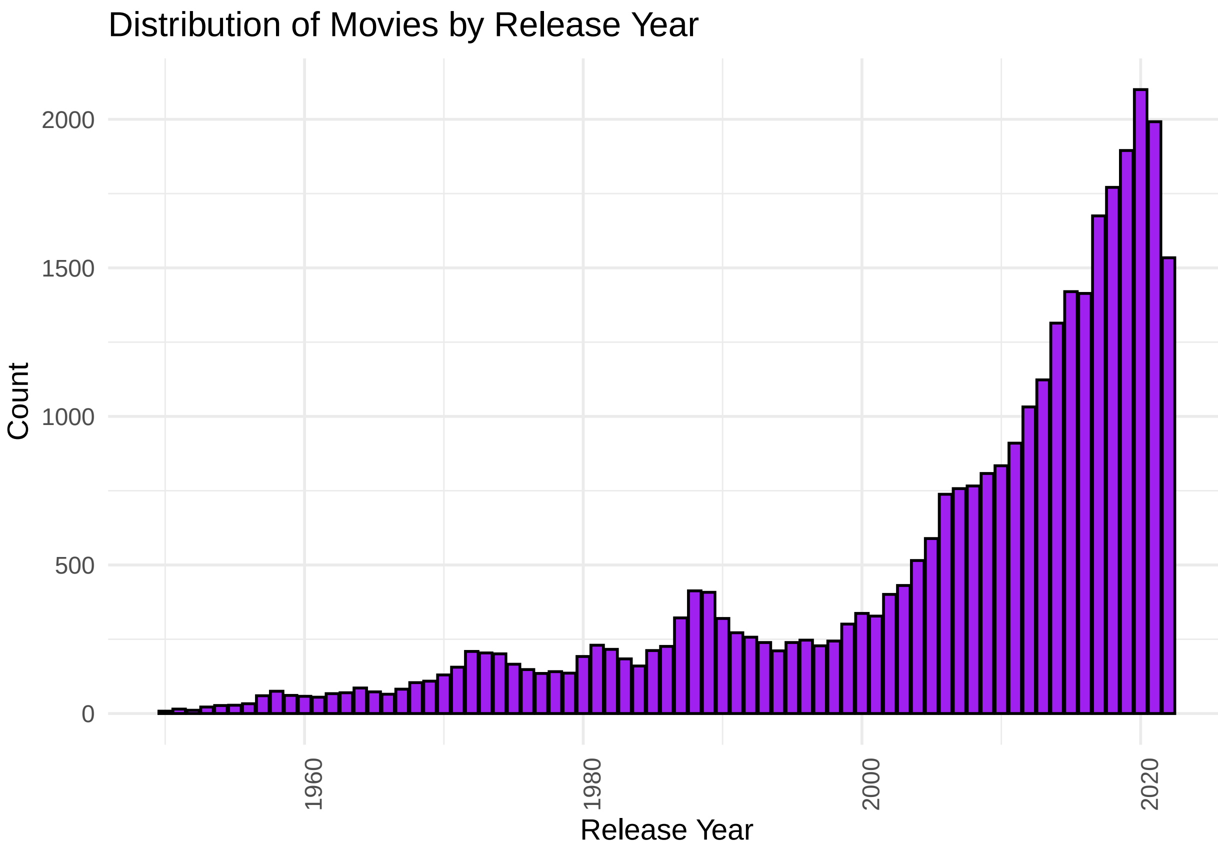


Budget and revenue are both right skewed and unimodal. Runtime is also right skewed, but bimodal.

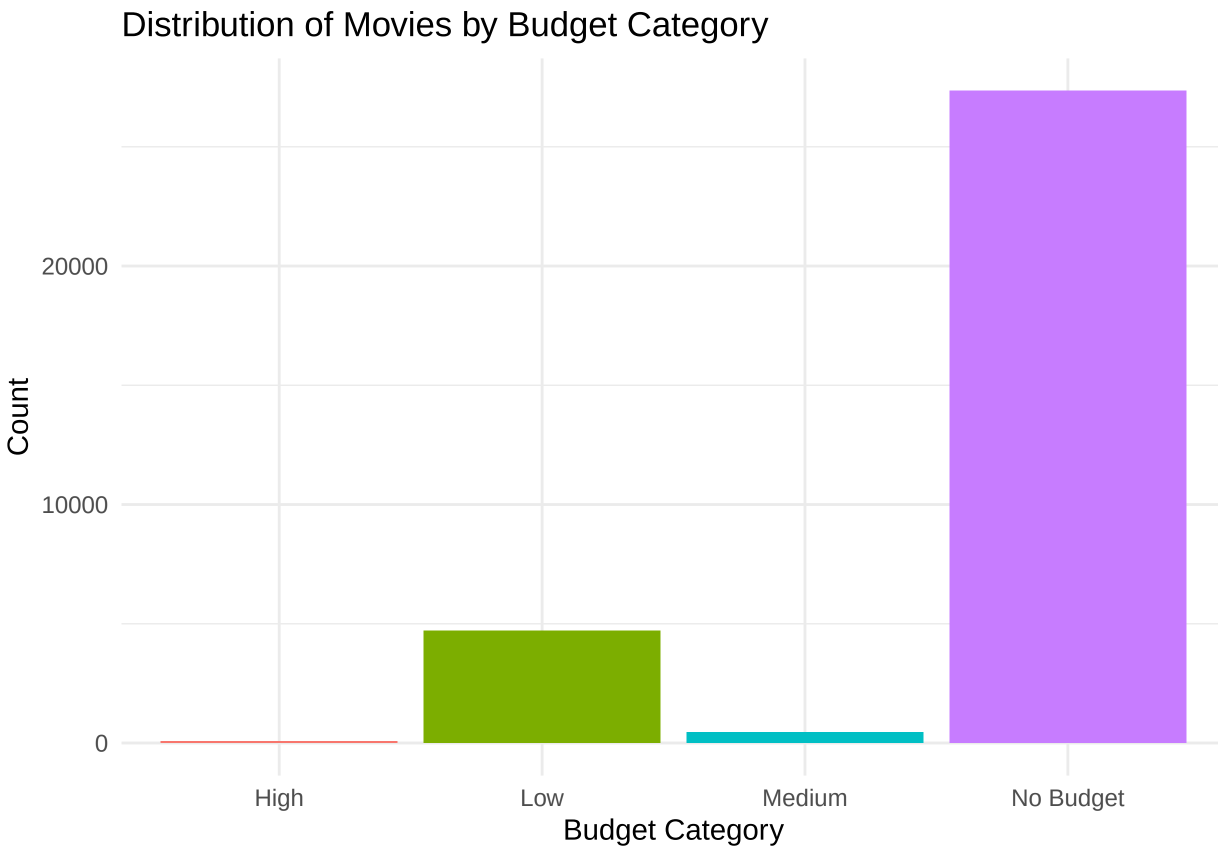
Distribution of Some Categorical Features

We will plot the distributions of some categorical features, specifically release_year and budget_category.

```
ggplot(horror, aes(x = release_year)) +
  geom_bar(fill = "purple", color = "black") +
  labs(title = "Distribution of Movies by Release Year", x = "Release Year", y = "Count") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```




```r
ggplot(horror, aes(x = budget_category, fill = budget_category)) +
  geom_bar() +
  labs(title = "Distribution of Movies by Budget Category", x = "Budget Category", y = "Count") +
  theme_minimal() +
  theme(legend.position = "none")
```



Release year is left skewed and unimodal.

Distribution of Target Variable

We will now look at the distribution of our target variable, show_or_movie.

###CHANGE WITH NEW PREDICTOR

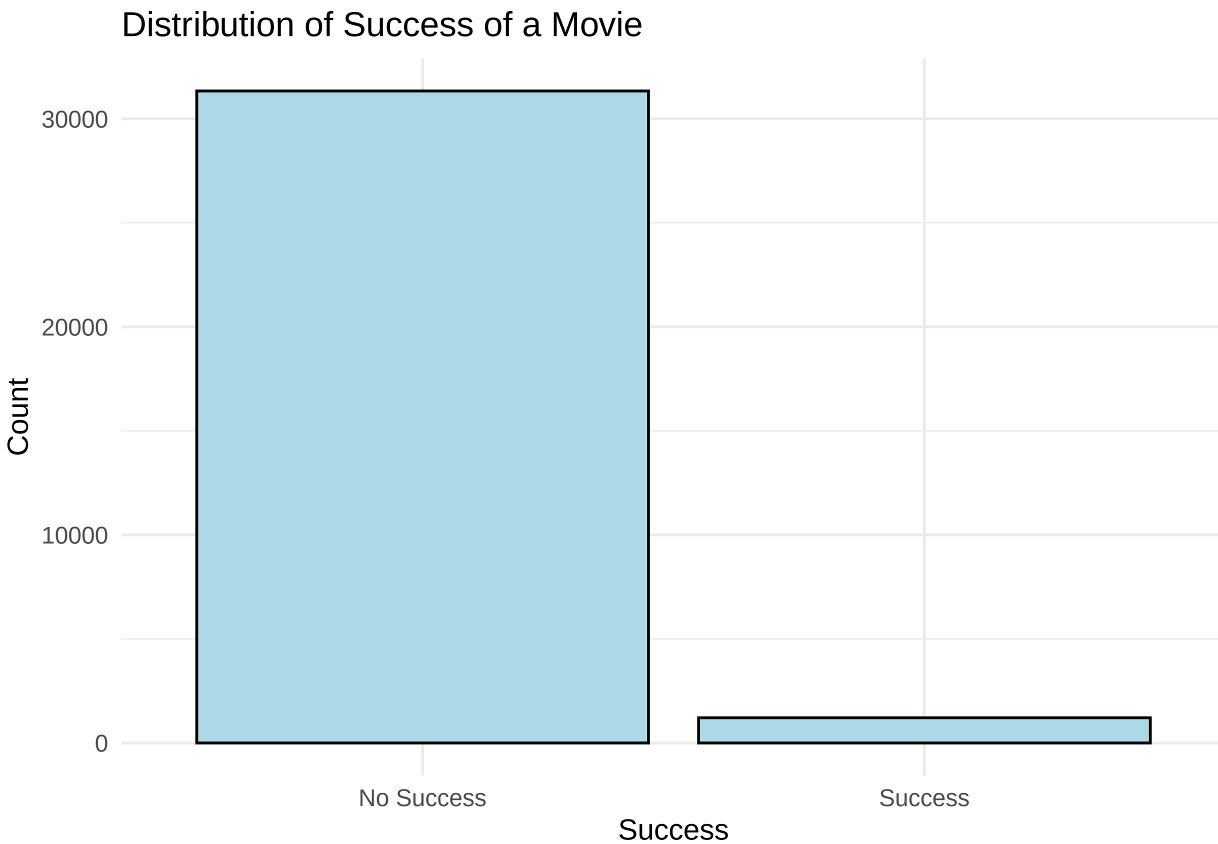
```r
sum(horror$success == "Success")
```

```
## [1] 1211
```

```r
sum(horror$success == "No Success")
```

```
## [1] 31329
```

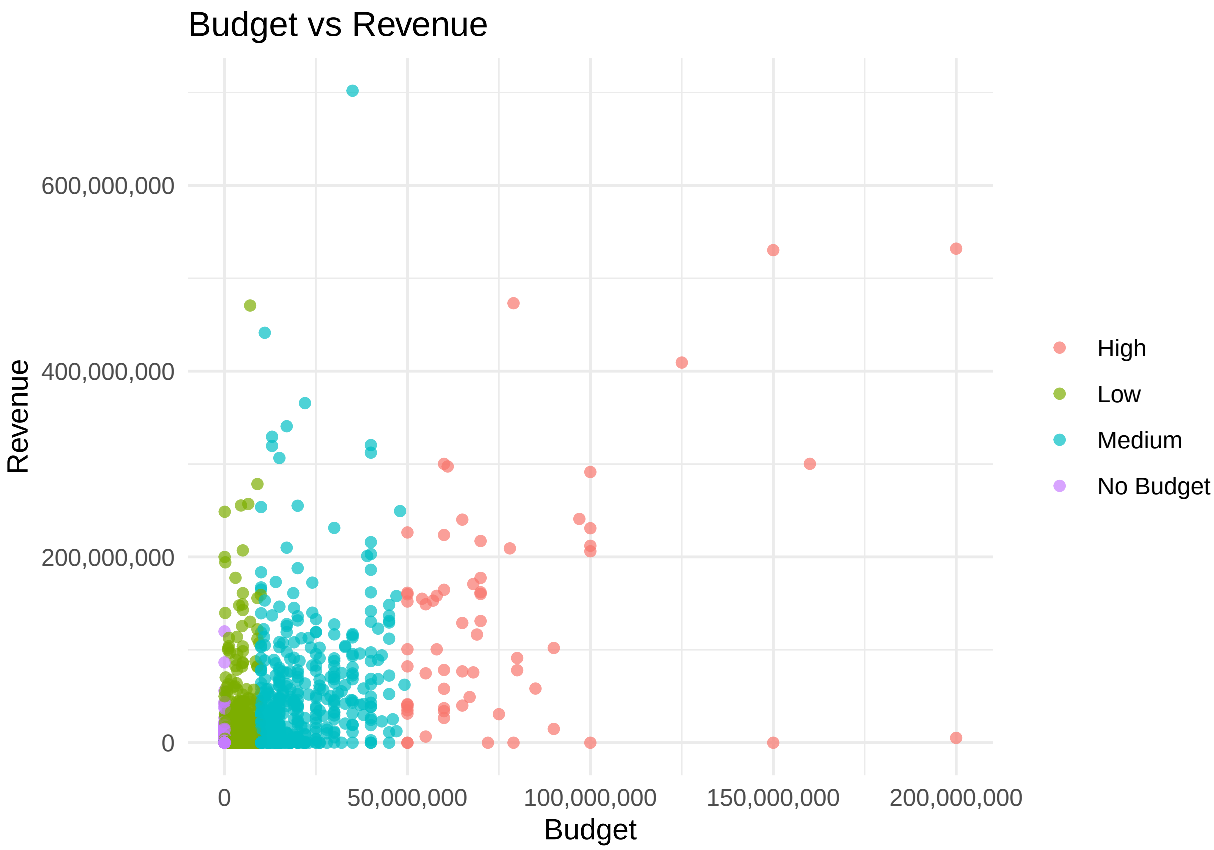
```r
ggplot(horror, aes(x = success)) +
  geom_bar(fill = "lightblue", color = "black") +
  labs(title = "Distribution of Success of a Movie", x = "Success", y = "Count") +
  theme_minimal()
```



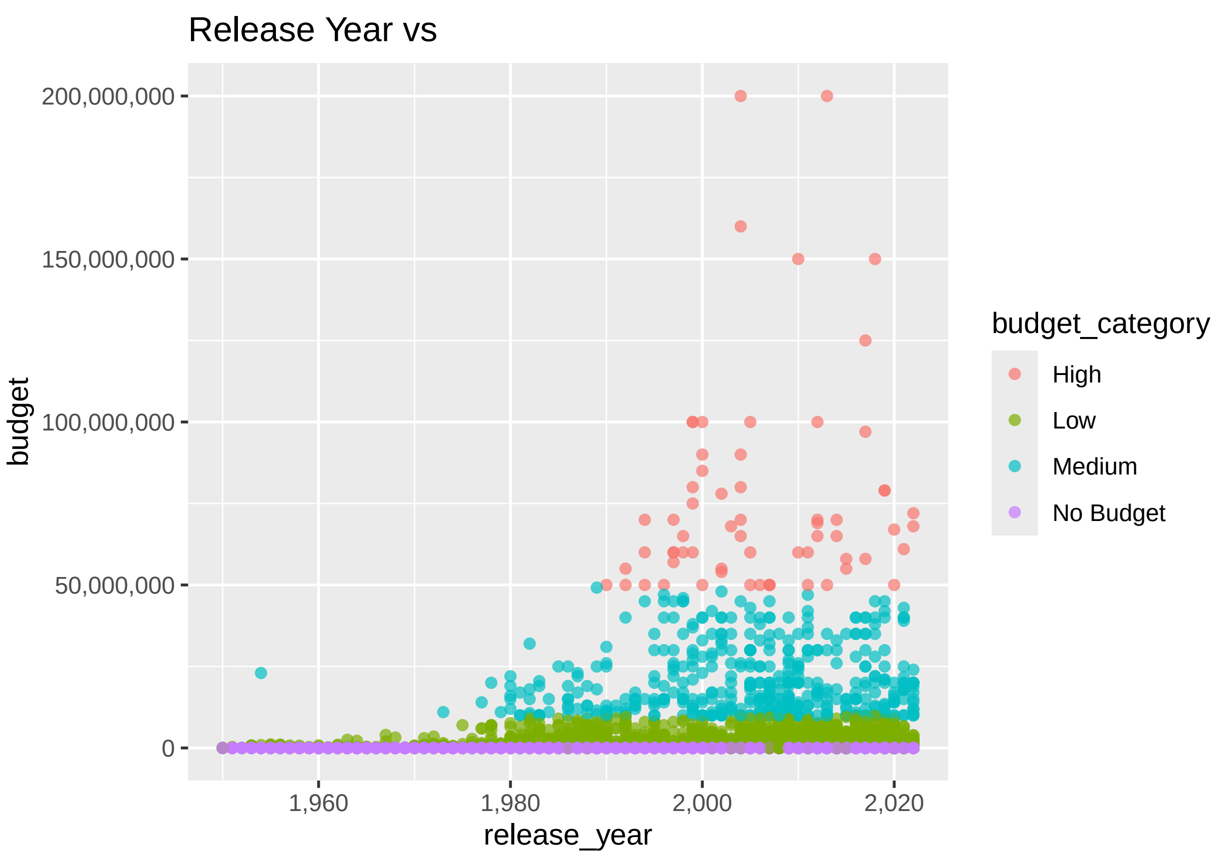
Variable Relationships

We first examine the relationship between budget and revenue for each horror movie, with points colored by their budget category, helping to identify patterns and outliers in how budget impacts revenue. We can also visualize how the budget has evolved over the years by plotting release_year versus budget. We then examine the relationship between `popularity` and `vote_average` to see if there's a trend in how movies' popularity correlates with their ratings. We also examine the relationship between `budget` and `profit`. Each point represents a movie, and the color differentiates between successful and non-successful movies. The idea is to see if movies with higher budgets tend to generate more profit. Finally we show how the `profit` distribution differs between successful and non-successful movies. This boxplot shows the distribution of `profit` for movies categorized as "Success" or "No Success". The plot helps visualize how profits are distributed across these categories, revealing if successful movies tend to have higher profits.

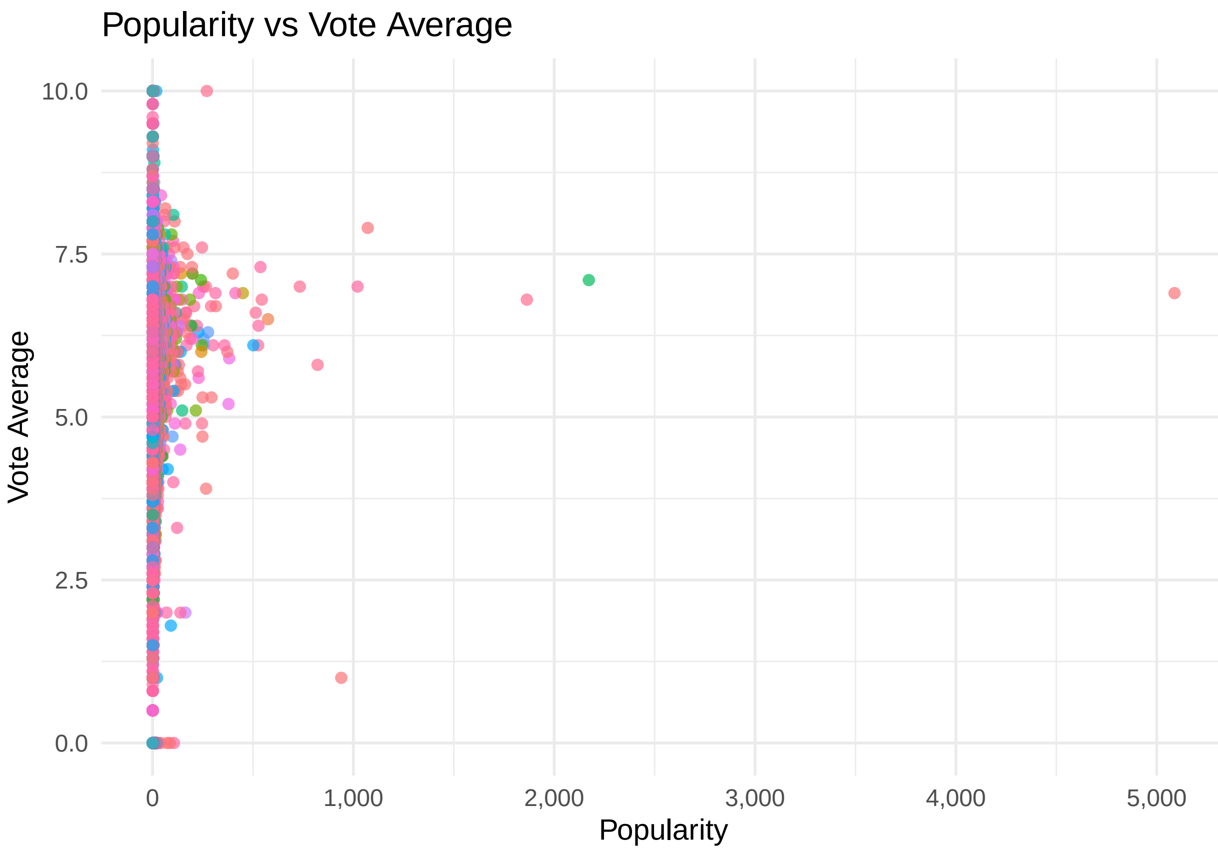
```
ggplot(horror, aes(x = budget, y = revenue)) +
  geom_point(aes(color = budget_category), alpha = 0.7) +
  scale_x_continuous(labels = scales::comma) +
  scale_y_continuous(labels = scales::comma) +
  labs(title = "Budget vs Revenue", x = "Budget", y = "Revenue") +
  theme_minimal() +
  theme(legend.title = element_blank())
```



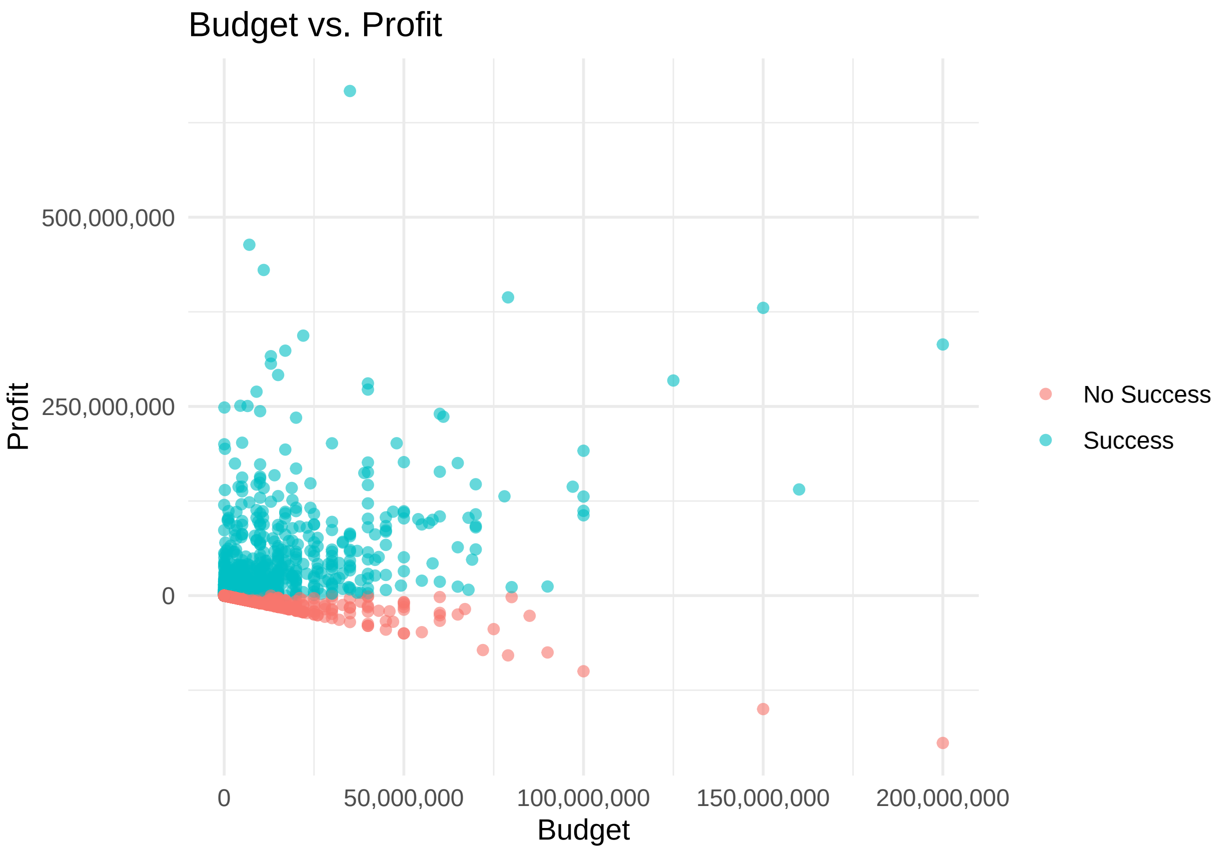

```r
ggplot(horror, aes(x = release_year, y = budget)) +
  geom_point(aes(color = budget_category), alpha = 0.7) +
  scale_x_continuous(labels = scales::comma) +
  scale_y_continuous(labels = scales::comma) +
  labs(title = "Release Year vs")
```



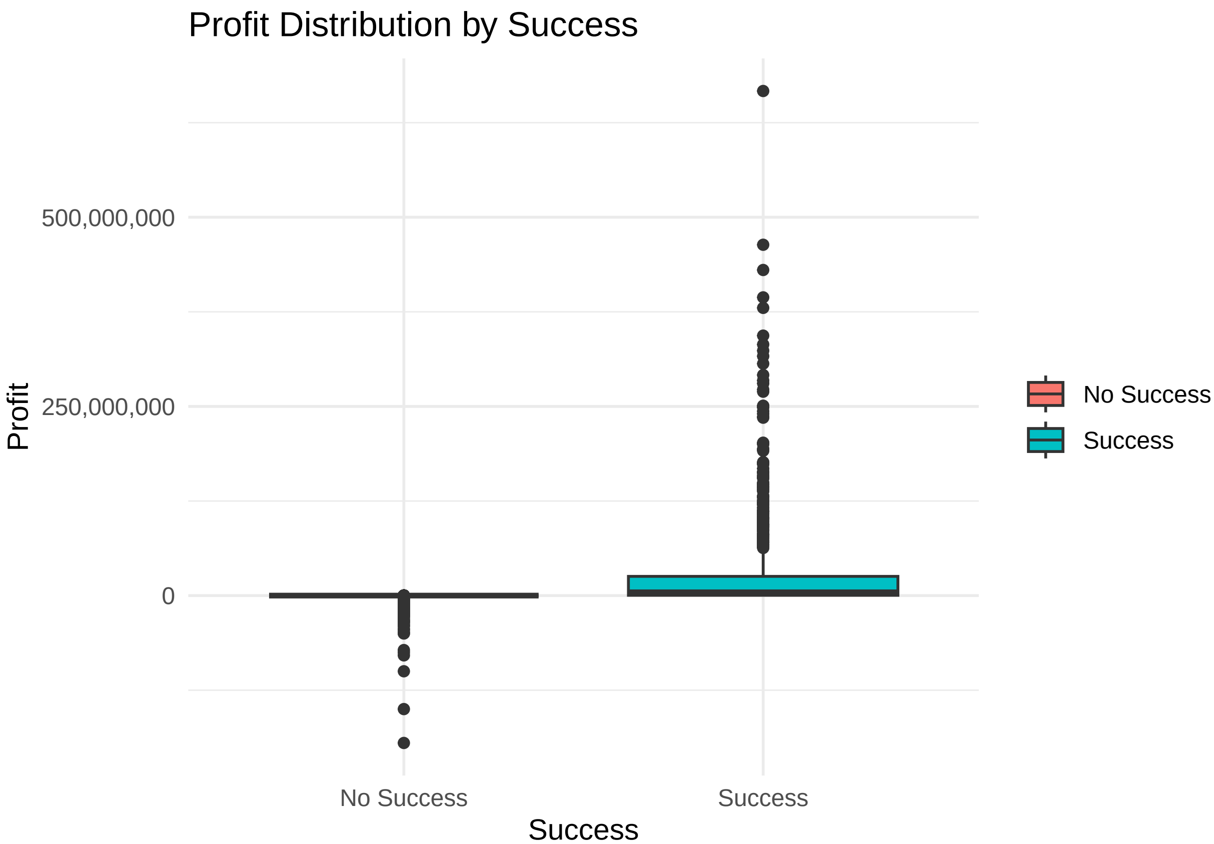

```
ggplot(horror, aes(x = popularity, y = vote_average)) +
  geom_point(aes(color = genre_names), alpha = 0.7) +
  scale_x_continuous(labels = scales::comma) +
  labs(title = "Popularity vs Vote Average", x = "Popularity", y = "Vote Average") +
  theme_minimal() +
  theme(legend.position = "none")
```




```r
ggplot(horror, aes(x = budget, y = profit)) +
  geom_point(aes(color = success), alpha = 0.6) +
  scale_x_continuous(labels = scales::comma) +
  scale_y_continuous(labels = scales::comma) +
  labs(title = "Budget vs. Profit", x = "Budget", y = "Profit") +
  theme_minimal() +
  theme(legend.title = element_blank())
```




```
ggplot(horror, aes(x = success, y = profit, fill = success)) +
  geom_boxplot() +
  scale_y_continuous(labels = scales::comma) +
  labs(title = "Profit Distribution by Success", x = "Success", y = "Profit") +
  theme_minimal() +
  theme(legend.title = element_blank())
```




```r
sum(horror$budget == 0)
```

```
## [1] 27339
```

```r
sum(horror$budget != 0)
```

```
## [1] 5201
```

Split the Data

Train and Test Data

We will split the data into training (60%) and testing (40%) sets. We will then look at the new data by checking the number of rows in training and testing sets and looking at the summary of the training set.

```r
set.seed(123)

in_train <- createDataPartition(horror$budget_category, p = 0.6, list = FALSE)
training <- horror[in_train, ]
testing <- horror[-in_train, ]

nrow(training)
```

```
## [1] 19526
```

```r
nrow(testing)
```

```
## [1] 13014
```

```r
summary(training)
```

```
##  original_title        title            original_language   overview
##  Length:19526       Length:19526       en     :13112     Length:19526
##  Class :character   Class :character   es     : 1007     Class :character
##  Mode  :character   Mode  :character   ja     :  968     Mode  :character
##                                        pt     :  404
##                                        de     :  393
##                                        fr     :  368
##                                        (Other): 3274
##    tagline           release_date          popularity          vote_count
##  Length:19526       Min.   :1950-01-01   Min.   :   0.000   Min.   :    0.00
##  Class :character   1st Qu.:2001-01-01   1st Qu.:   0.600   1st Qu.:    0.00
##  Mode  :character   Median :2013-01-01   Median :   0.815   Median :    2.00
##                     Mean   :2007-03-15   Mean   :   4.025   Mean   :   61.54
##                     3rd Qu.:2018-10-05   3rd Qu.:   2.217   3rd Qu.:   10.00
##                     Max.   :2022-12-31   Max.   :2172.338   Max.   :16900.00
##
##   vote_average        budget               revenue              runtime
##  Min.   : 0.000   Min.   :        0   Min.   :        0   Min.   :  0.00
##  1st Qu.: 0.000   1st Qu.:        0   1st Qu.:        0   1st Qu.: 14.00
##  Median : 4.000   Median :        0   Median :        0   Median : 80.00
##  Mean   : 3.299   Mean   :   541349   Mean   :  1342095   Mean   : 61.65
##  3rd Qu.: 5.600   3rd Qu.:        0   3rd Qu.:        0   3rd Qu.: 91.00
##  Max.   :10.000   Max.   :200000000   Max.   :701842551   Max.   :205.00
##
##             status        adult                                genre_names
##  In Production  :   42   Mode :logical   Horror                      :7569
##  Planned        :    1   FALSE:19526     Horror, Thriller            :1957
##  Post Production:   39                   Comedy, Horror              :1743
##  Released       :19444                   Drama, Horror               : 800
##                                          Horror, Science Fiction     : 519
##                                          Horror, Mystery, Thriller: 507
##                                          (Other)                     :6431
##   release_year  budget_category        profit               success
##  Min.   :1950   Length:19526       Min.   :-194775779   Length:19526
##  1st Qu.:2001   Class :character   1st Qu.:         0   Class :character
##  Median :2013   Mode  :character   Median :         0   Mode  :character
##  Mean   :2007                      Mean   :    800746
##  3rd Qu.:2018                      3rd Qu.:         0
##  Max.   :2022                      Max.   : 666842551
##
```

Distribution of Target Variable

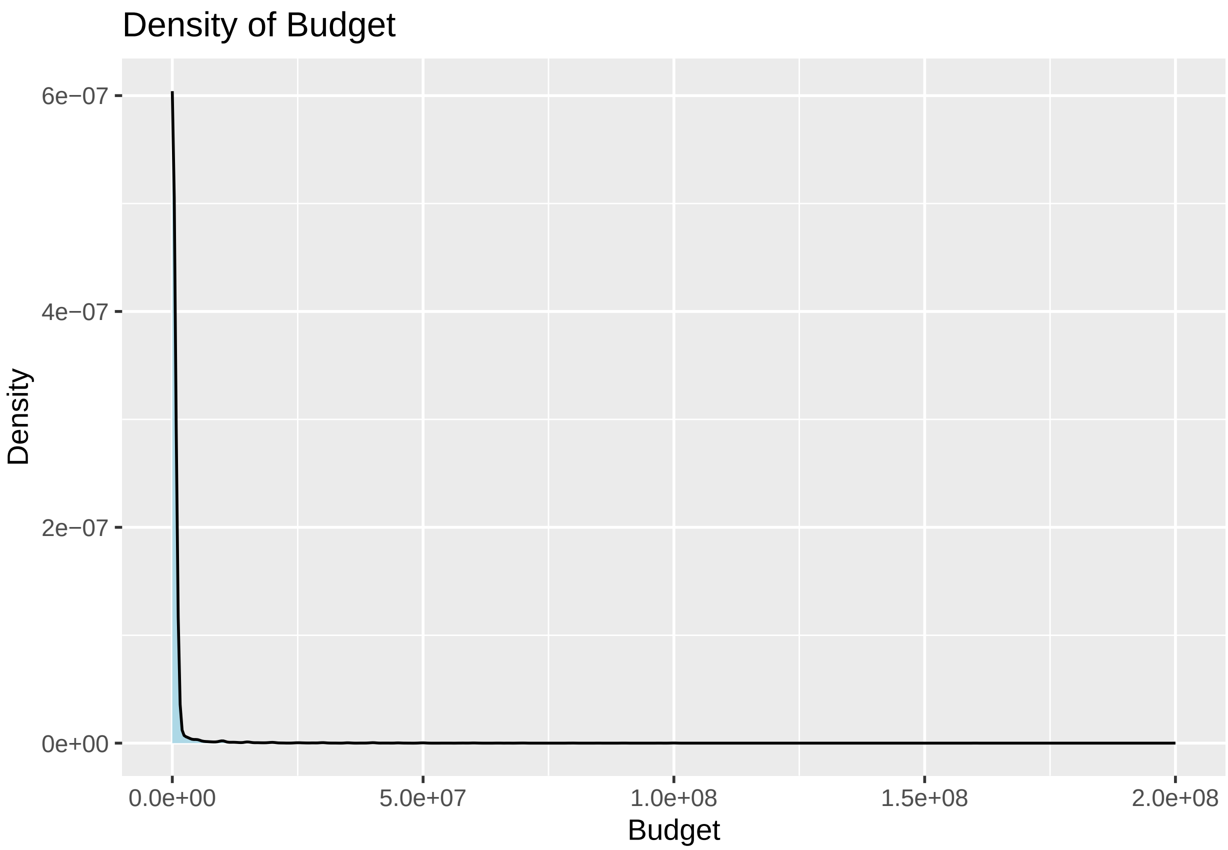
```r
table(training$success) / length(training$success)
```

```
##
## No Success    Success
## 0.96220424 0.03779576
```

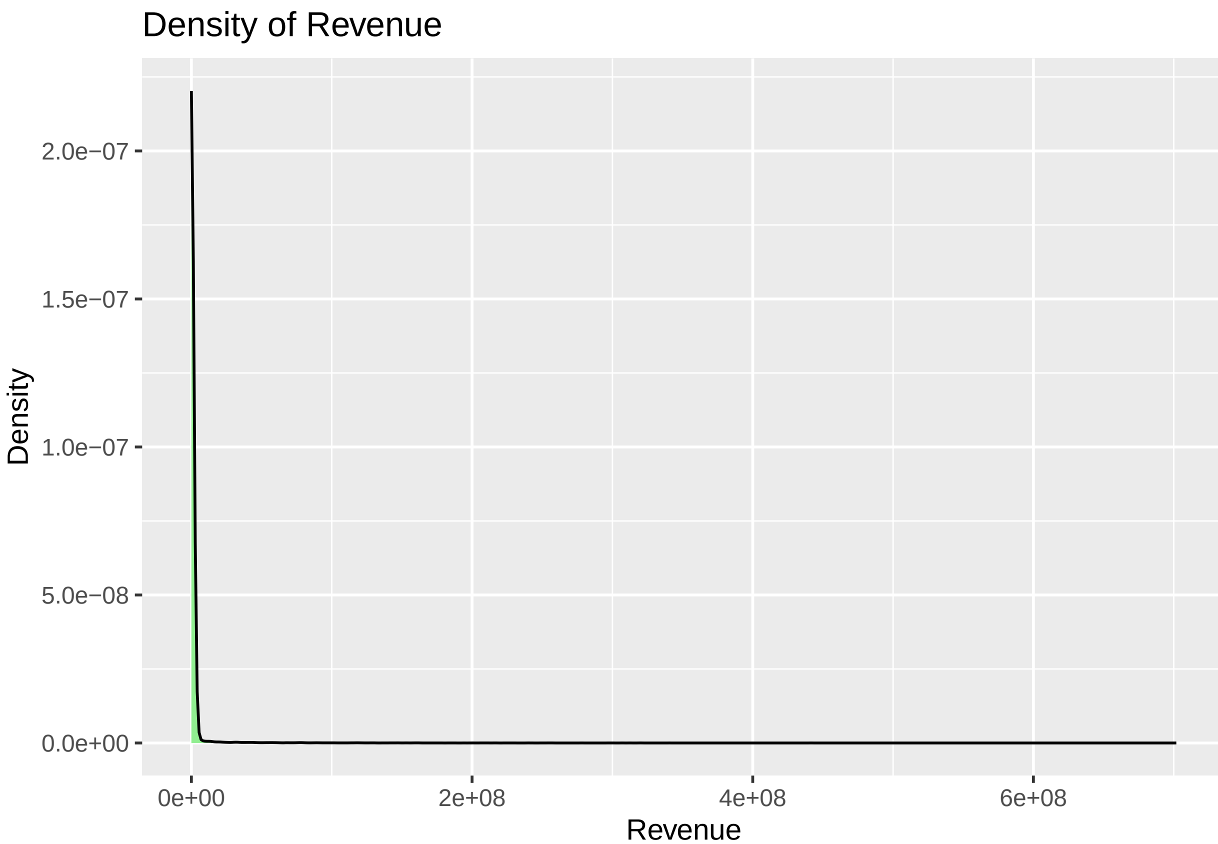
4% successful movies, 96% unsuccessful movies: unbalanced dataset

Training Visuals

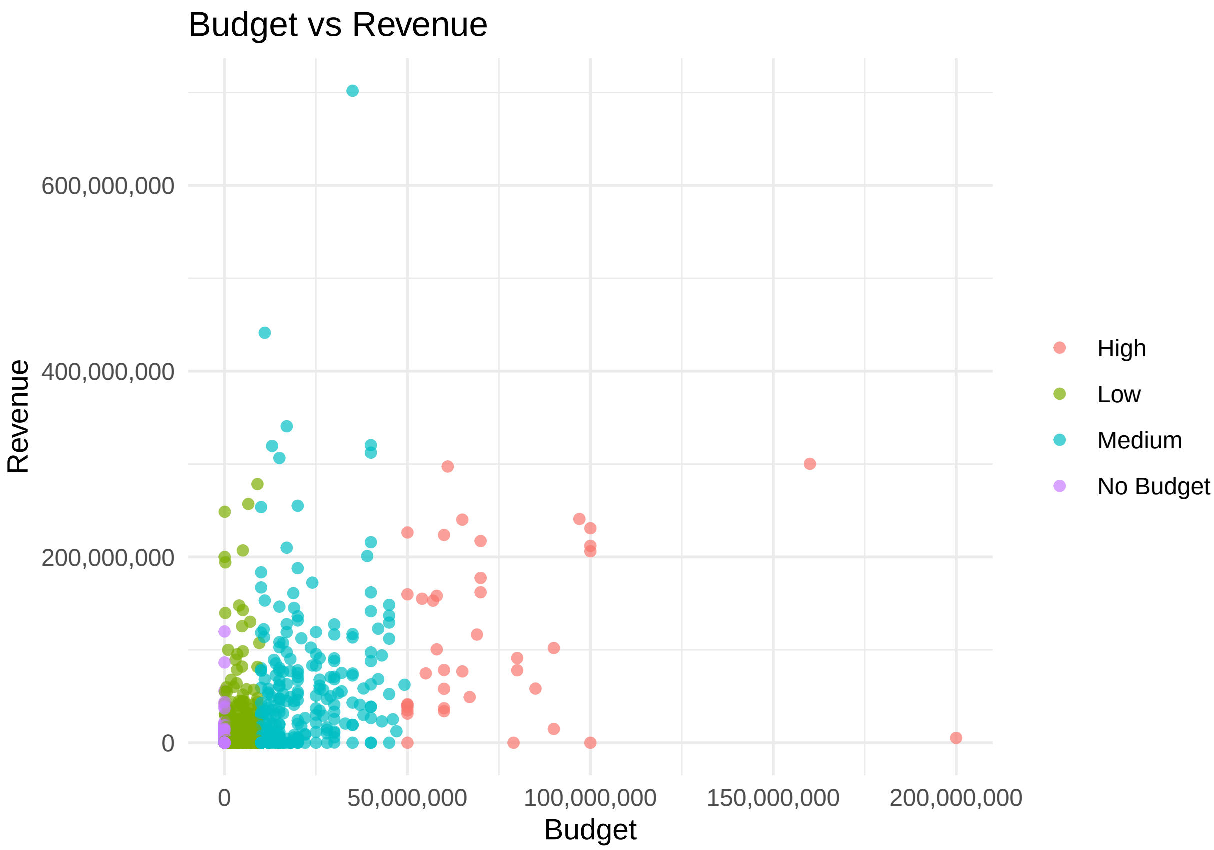
```r
# Visualizing the distribution of 'budget' in training data
ggplot(training, aes(x = budget)) +
  geom_density(fill = "lightblue") +
  labs(title = "Density of Budget", x = "Budget", y = "Density")
```



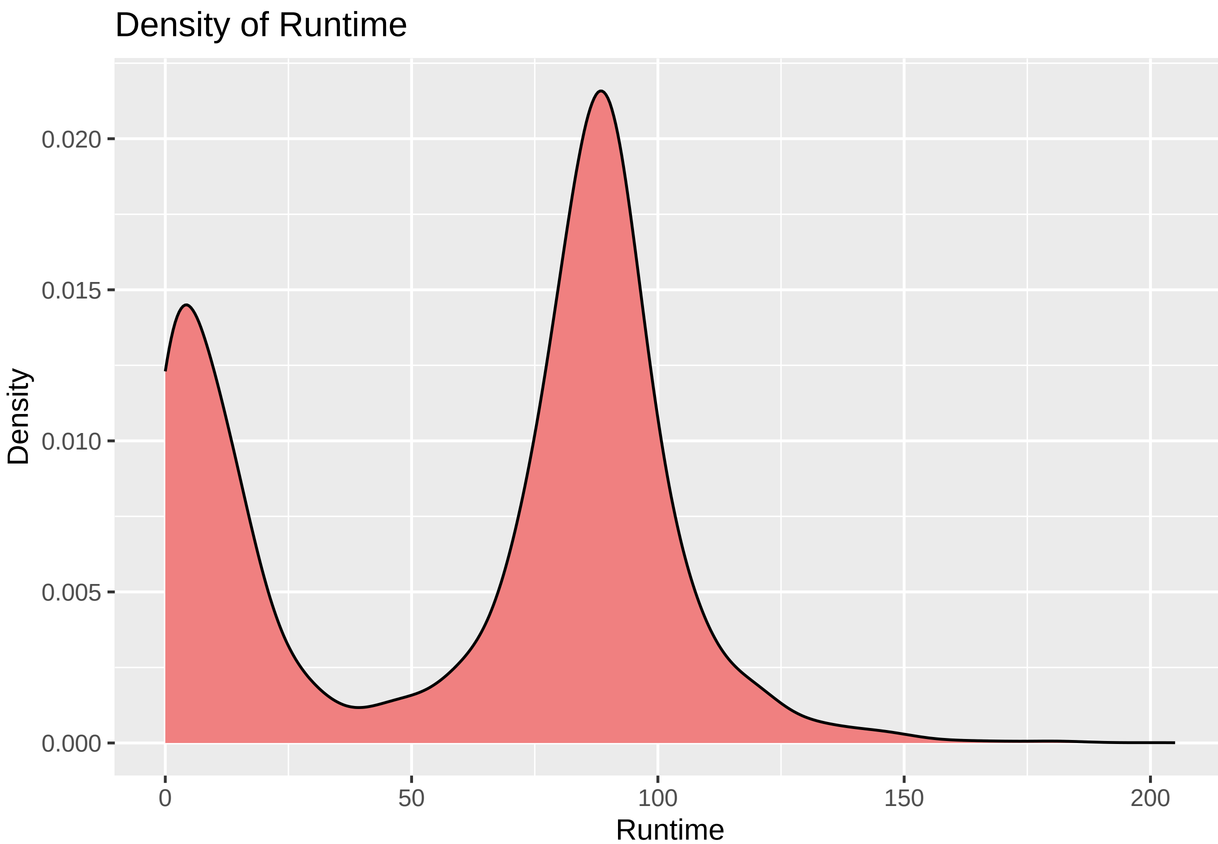

```r
# Visualizing the distribution of 'revenue' in training data
ggplot(training, aes(x = revenue)) +
  geom_density(fill = "lightgreen") +
  labs(title = "Density of Revenue", x = "Revenue", y = "Density")
```



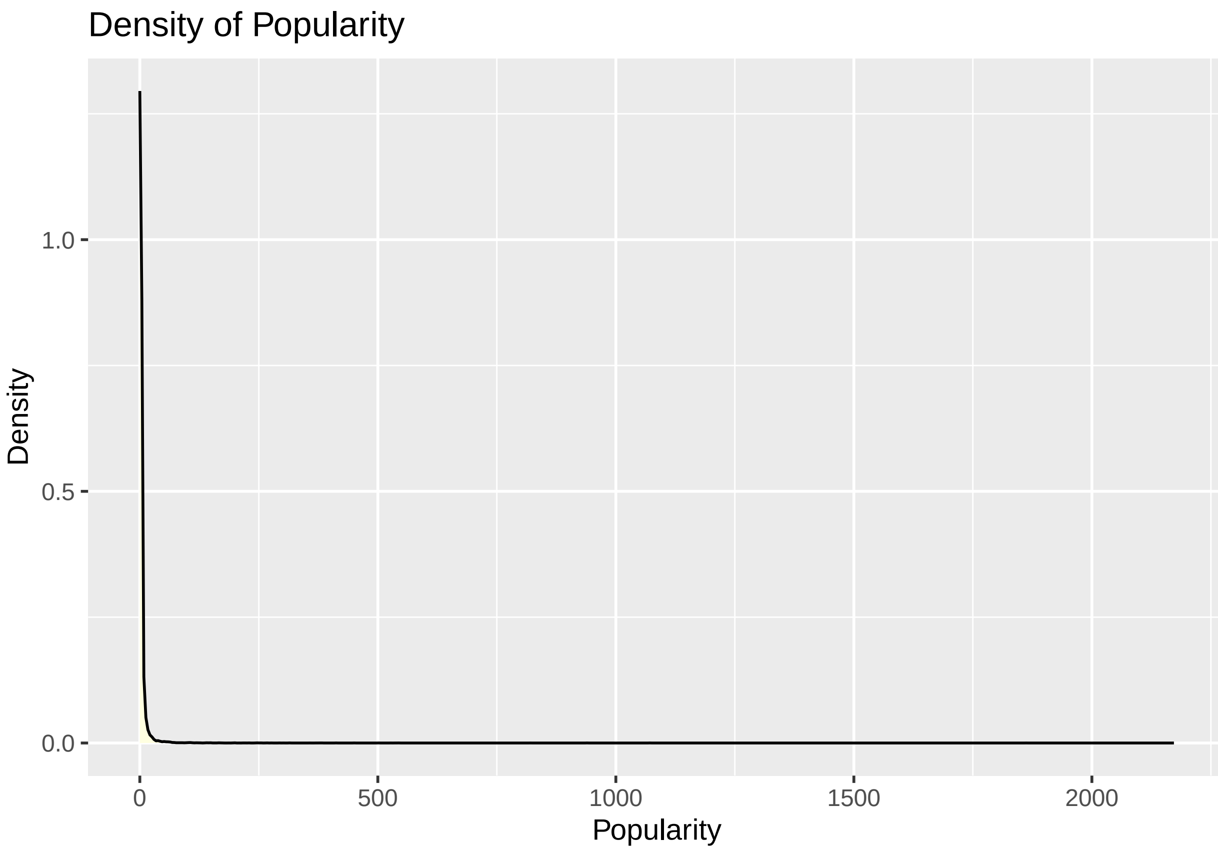

```r
# Visualizing 'budget' against 'revenue'
ggplot(training, aes(x = budget, y = revenue)) +
  geom_point(aes(color = budget_category), alpha = 0.7) +
  scale_x_continuous(labels = scales::comma) +
  scale_y_continuous(labels = scales::comma) +
  labs(title = "Budget vs Revenue", x = "Budget", y = "Revenue") +
  theme_minimal() +
  theme(legend.title = element_blank())
```



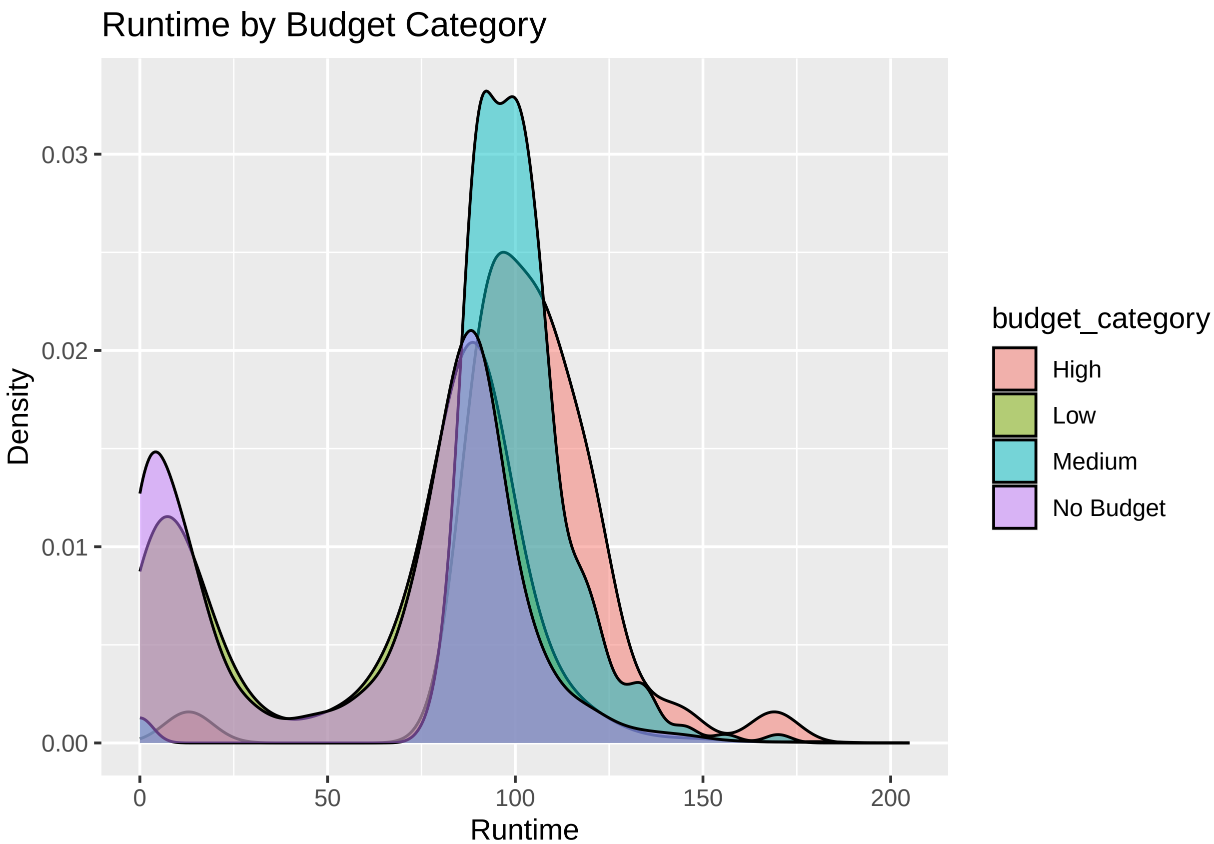

```
# Exploring 'runtime' distribution
ggplot(training, aes(x = runtime)) +
  geom_density(fill = "lightcoral") +
  labs(title = "Density of Runtime", x = "Runtime", y = "Density")
```



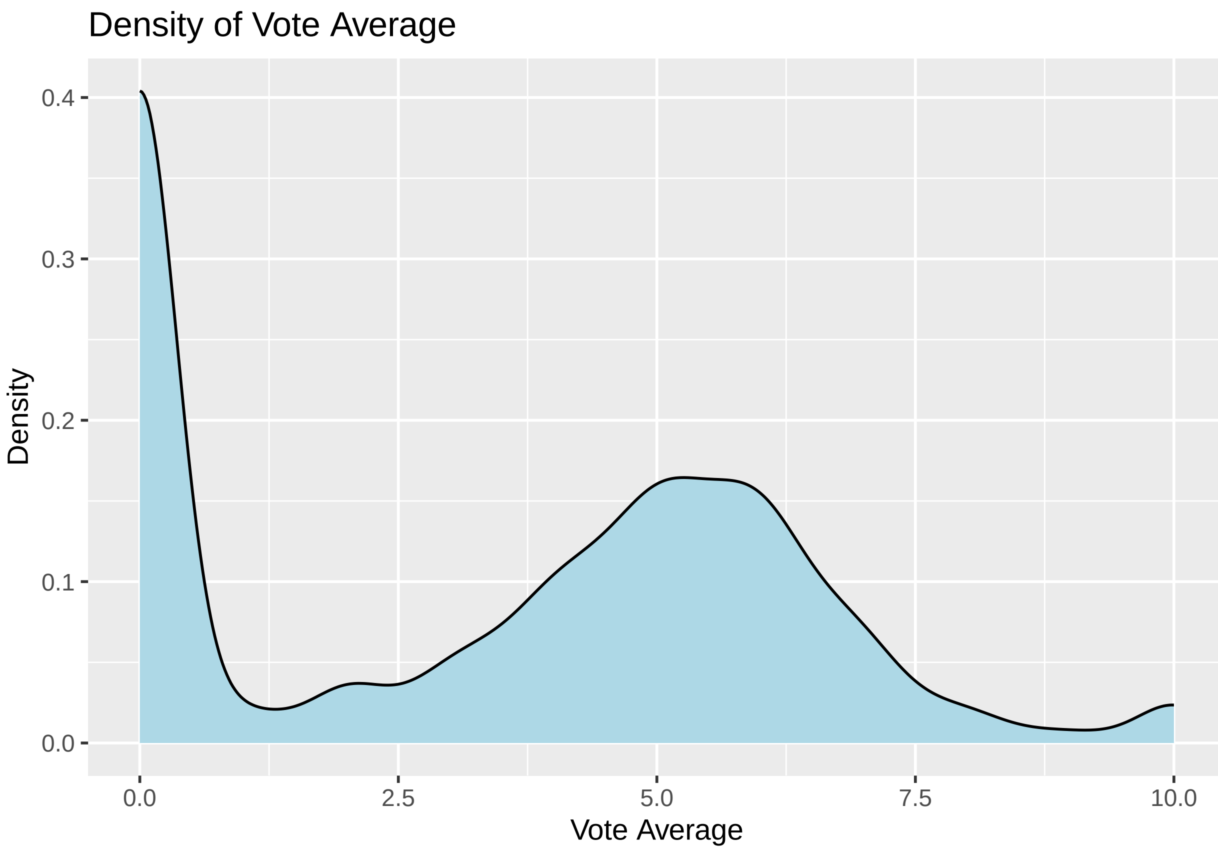

```r
# Exploring 'popularity' distribution
ggplot(training, aes(x = popularity)) +
  geom_density(fill = "lightyellow") +
  labs(title = "Density of Popularity", x = "Popularity", y = "Density")
```



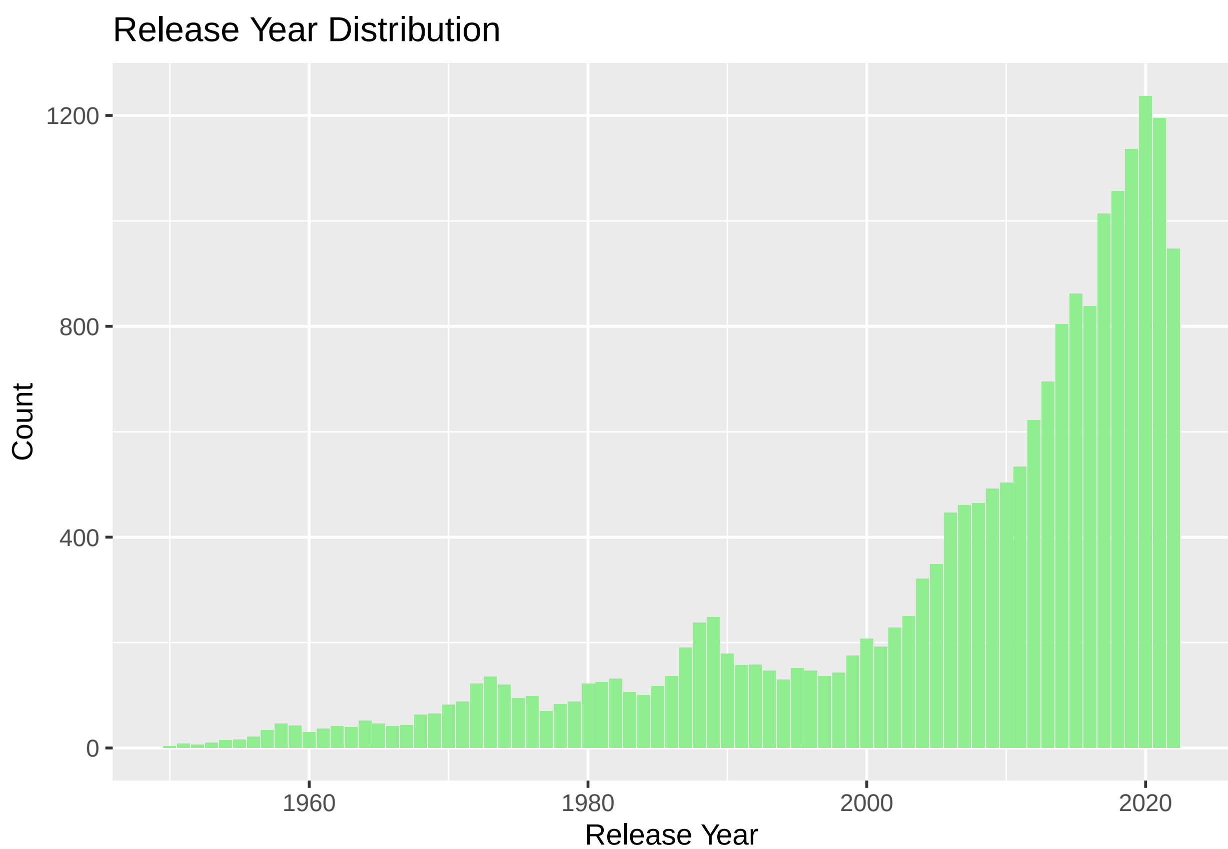

```r
# Decomposing runtime by 'budget_category'
ggplot(training, aes(x = runtime, fill = budget_category)) +
  geom_density(alpha = 0.5) +
  labs(title = "Runtime by Budget Category", x = "Runtime", y = "Density")
```



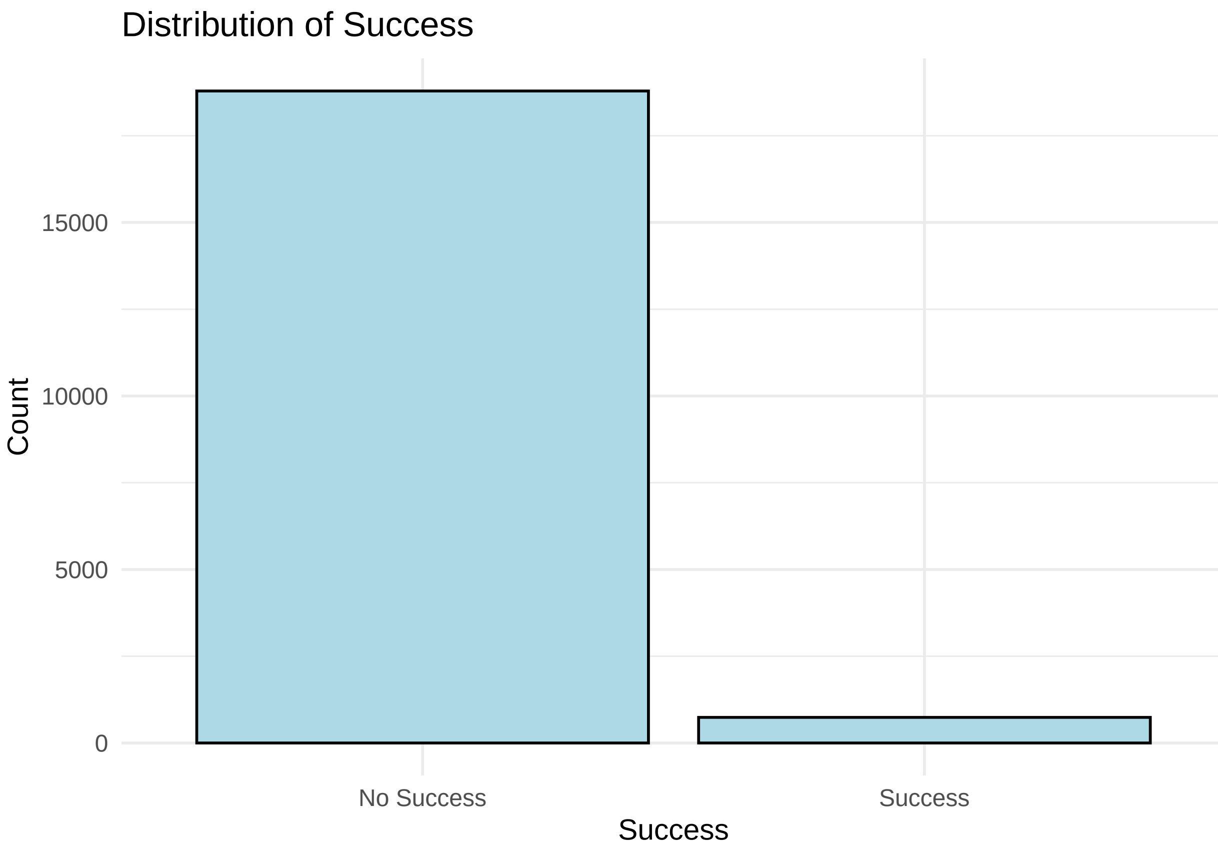

```
# Visualizing 'vote_average' distribution
ggplot(training, aes(x = vote_average)) +
  geom_density(fill = "lightblue") +
  labs(title = "Density of Vote Average", x = "Vote Average", y = "Density")
```




```
# Visualizing 'release_year' distribution
ggplot(training, aes(x = release_year)) +
  geom_bar(fill = "lightgreen") +
  labs(title = "Release Year Distribution", x = "Release Year", y = "Count")
```




```r
# Visualizing the distribution of 'success' in training data
ggplot(training, aes(x = success)) +
  geom_bar(fill = "lightblue", color = "black") +
  labs(title = "Distribution of Success", x = "Success", y = "Count") +
  theme_minimal()
```



Classification with Emphasis on Prediction

Questions We Aim to Answer:

1. "How well can we predict a movie's revenue based on its budget, popularity, and release year?" (Regression)
2. Predicting popularity (Regression)
3. "Can we predict whether a movie will be profitable?" (QDA and LDA)
4. "predicting success categories (hit, average, flop) (QDA and LDA)
5. Predicting Budget using logistic regression

```
# Create a new data frame without rows where revenue is 0
df_for_class = subset(horror, revenue != 0 & budget != 0)
df_for_class$success = as.factor(df_for_class$success)

head(df_for_class)
```

```
## # A tibble: 6 x 19
##   original_title           title original_language overview tagline release_date
##   <chr>                    <chr> <fct>             <chr>    <chr>   <date>
## 1 Smile                    Smile en                After w~ Once y~ 2022-09-23
## 2 The Black Phone          The ~ en                Finney ~ Never ~ 2022-06-22
## 3 Jeepers Creepers: Reborn Jeep~ en                Forced ~ Evil R~ 2022-09-15
## 4 Nope                     Nope  en                Residen~ What's~ 2022-07-20
## 5 X                        X     en                In 1979~ Dying ~ 2022-03-17
## 6 Dahmer                   Dahm~ en                On Febr~ The mi~ 2002-06-21
## # i 13 more variables: popularity <dbl>, vote_count <dbl>, vote_average <dbl>,
```

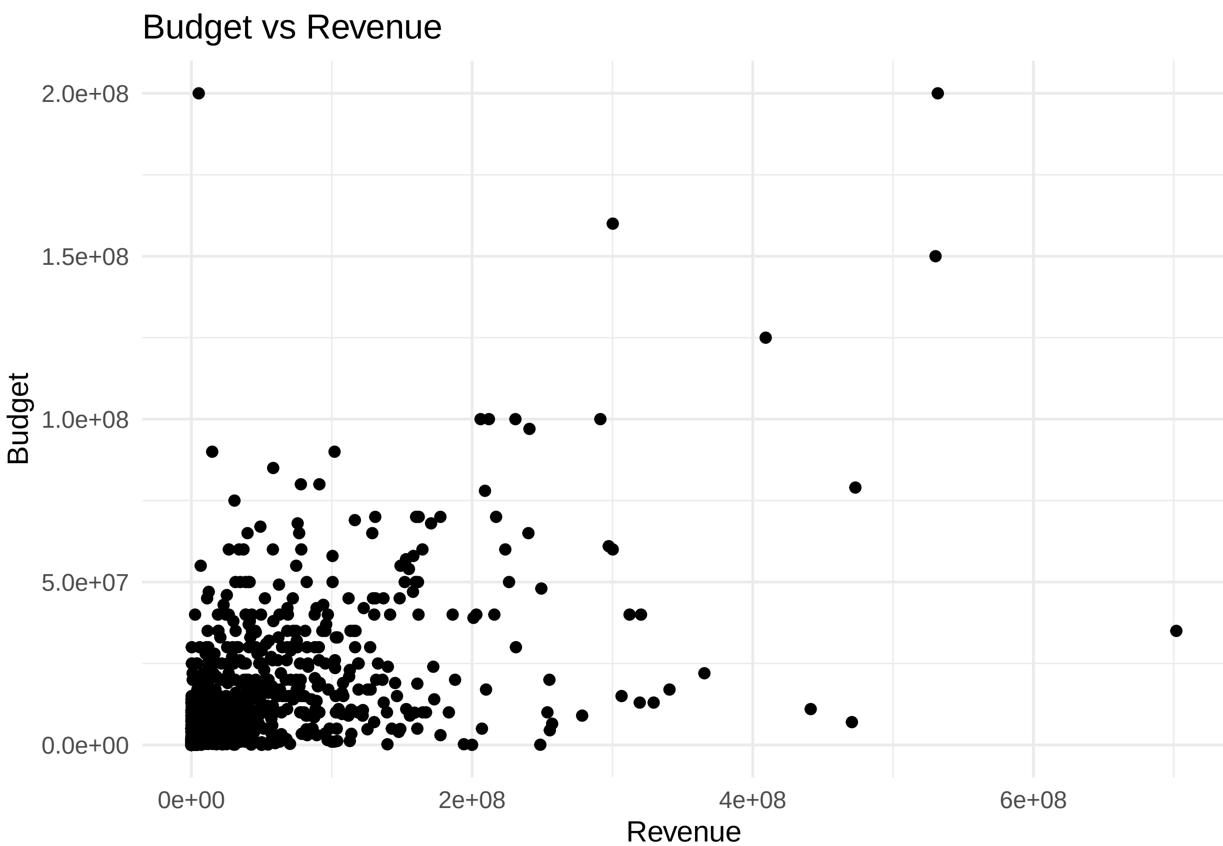
```
## #   budget <dbl>, revenue <dbl>, runtime <dbl>, status <fct>, adult <lgl>,
## #   genre_names <fct>, release_year <dbl>, budget_category <chr>, profit <dbl>,
## #   success <fct>
```

```r
dim(df_for_class)
```

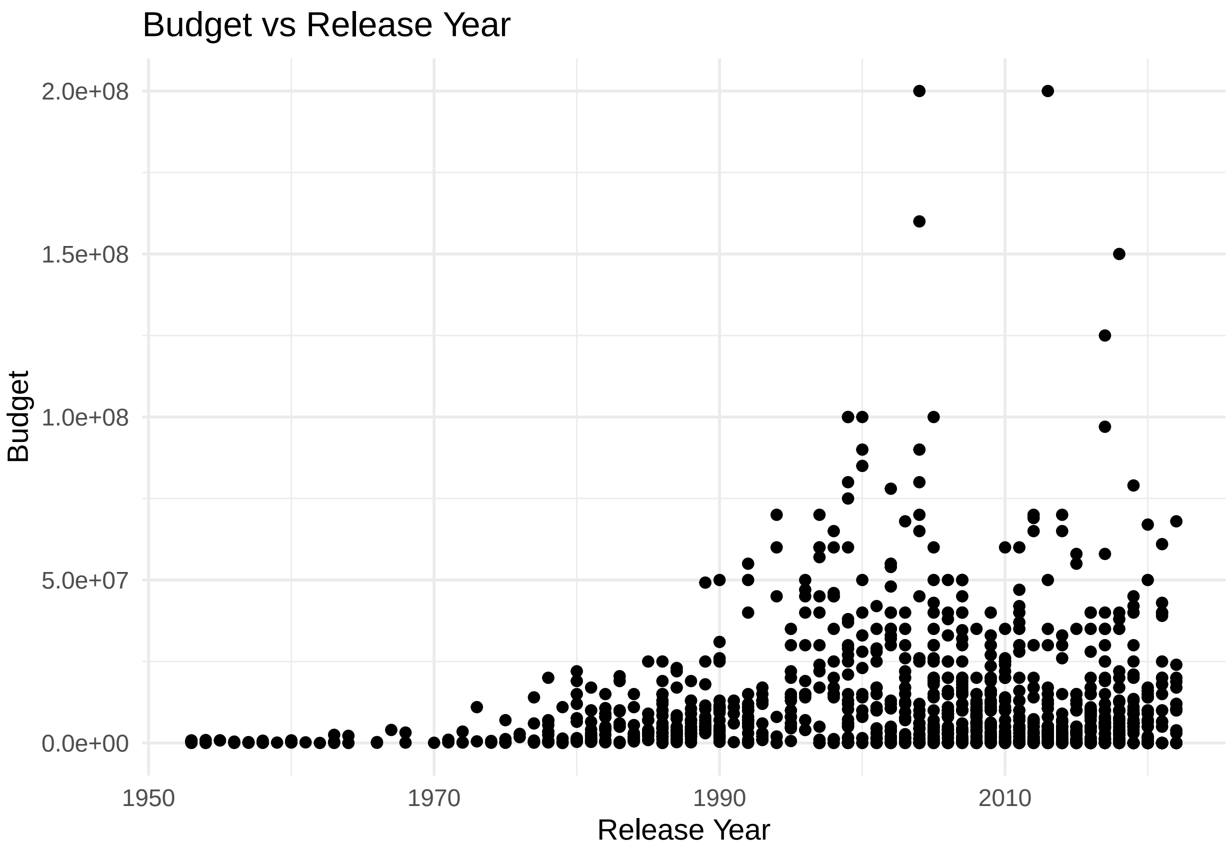
```
## [1] 1098   19
```

Visualizing the Relationships

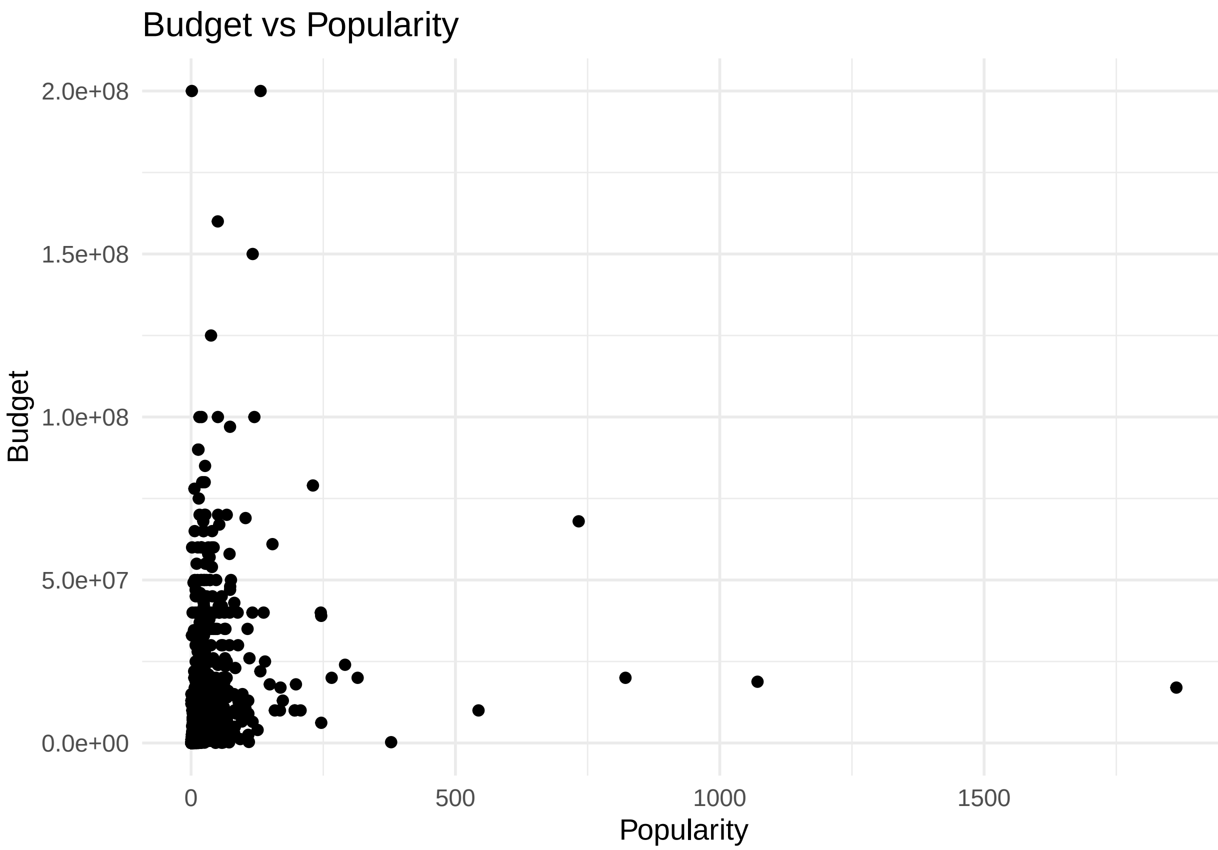
```r
library(ggplot2)

# Scatter plots to see the relationships
ggplot(df_for_class, aes(x = revenue, y = budget)) +
  geom_point() +
  labs(title = "Budget vs Revenue", x = "Revenue", y = "Budget") +
  theme_minimal()
```



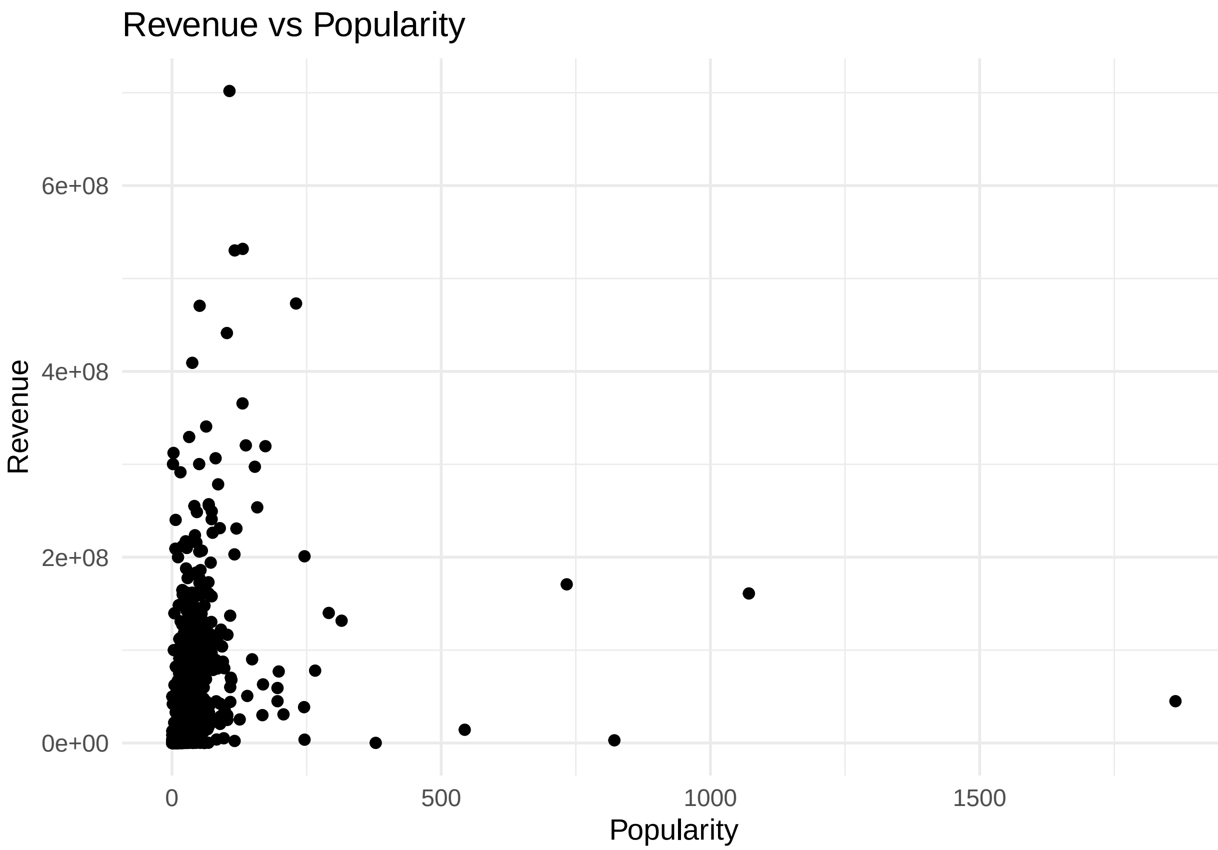

```r
ggplot(df_for_class, aes(x = release_year, y = budget)) +
  geom_point() +
  labs(title = "Budget vs Release Year", x = "Release Year", y = "Budget") +
  theme_minimal()
```



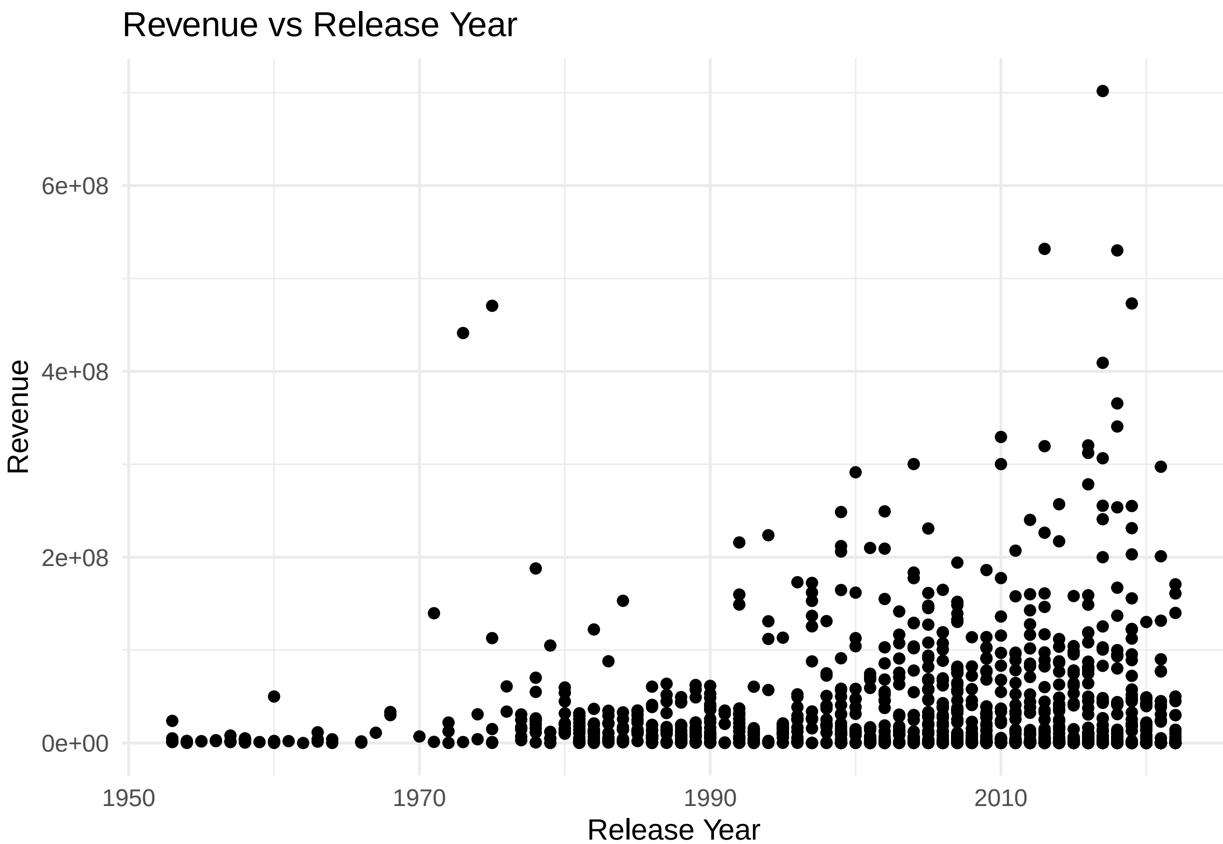

```r
ggplot(df_for_class, aes(x = popularity, y = budget)) +
  geom_point() +
  labs(title = "Budget vs Popularity", x = "Popularity", y = "Budget") +
  theme_minimal()
```



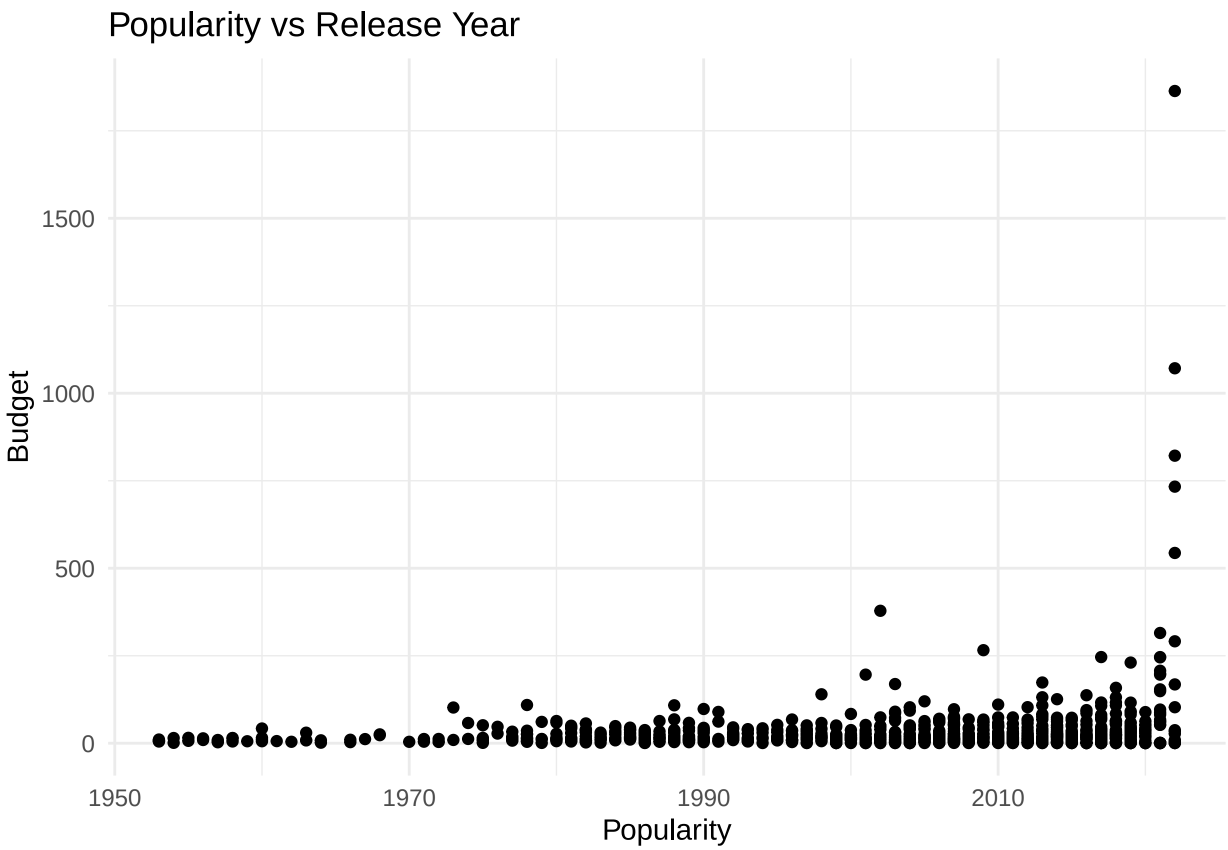

```
ggplot(df_for_class, aes(x = popularity, y = revenue)) +
  geom_point() +
  labs(title = "Revenue vs Popularity", x = "Popularity", y = "Revenue") +
  theme_minimal()
```




```r
ggplot(df_for_class, aes(x = release_year, y = revenue)) +
  geom_point() +
  labs(title = "Revenue vs Release Year", x = "Release Year", y = "Revenue") +
  theme_minimal()
```




```r
ggplot(df_for_class, aes(x = release_year, y = popularity)) +
  geom_point() +
  labs(title = "Popularity vs Release Year", x = "Popularity", y = "Budget") +
  theme_minimal()
```



Classifying Success with LDA:

Training the model

```
lda_model = lda(success ~ budget + release_year + runtime + popularity, data = df_for_class)

lda_model
```

```
## Call:
## lda(success ~ budget + release_year + runtime + popularity, data = df_for_class)
##
## Prior probabilities of groups:
## No Success    Success
##  0.3324226  0.6675774
##
## Group means:
##              budget release_year  runtime popularity
## No Success 10822277     2006.455 85.61096   17.15807
## Success    13549131     2001.524 94.57162   36.44876
##
## Coefficients of linear discriminants:
##                        LD1
## budget        6.931965e-09
## release_year -4.680875e-02
## runtime       1.679305e-02
## popularity    6.118678e-03
```

Predicting

We will first predict success categories and then add the predictions to the LDA dataset for visualizing the predictions.

```
predictions_LDA = predict(lda_model, newdata = df_for_class)

predicted_classes_LDA = predictions_LDA$class

predicted_probs_LDA = predictions_LDA$posterior

predictions_LDA_counts = table(predicted_classes_LDA)
print(predictions_LDA_counts)
```

```
## predicted_classes_LDA
## No Success    Success
##         86       1012
```

```
predictions <- predict(lda_model, newdata = df_for_class)

df_for_class$lda_predicted_success <- predictions$class
```

Evaluating the Model

We will use the confusion matrix and accuracy calculations to show the accuracy of the model.

```
confusion_matrix_LDA = table(Predicted_LDA = predicted_classes_LDA, Actual = df_for_class$success)
print(confusion_matrix_LDA)
```

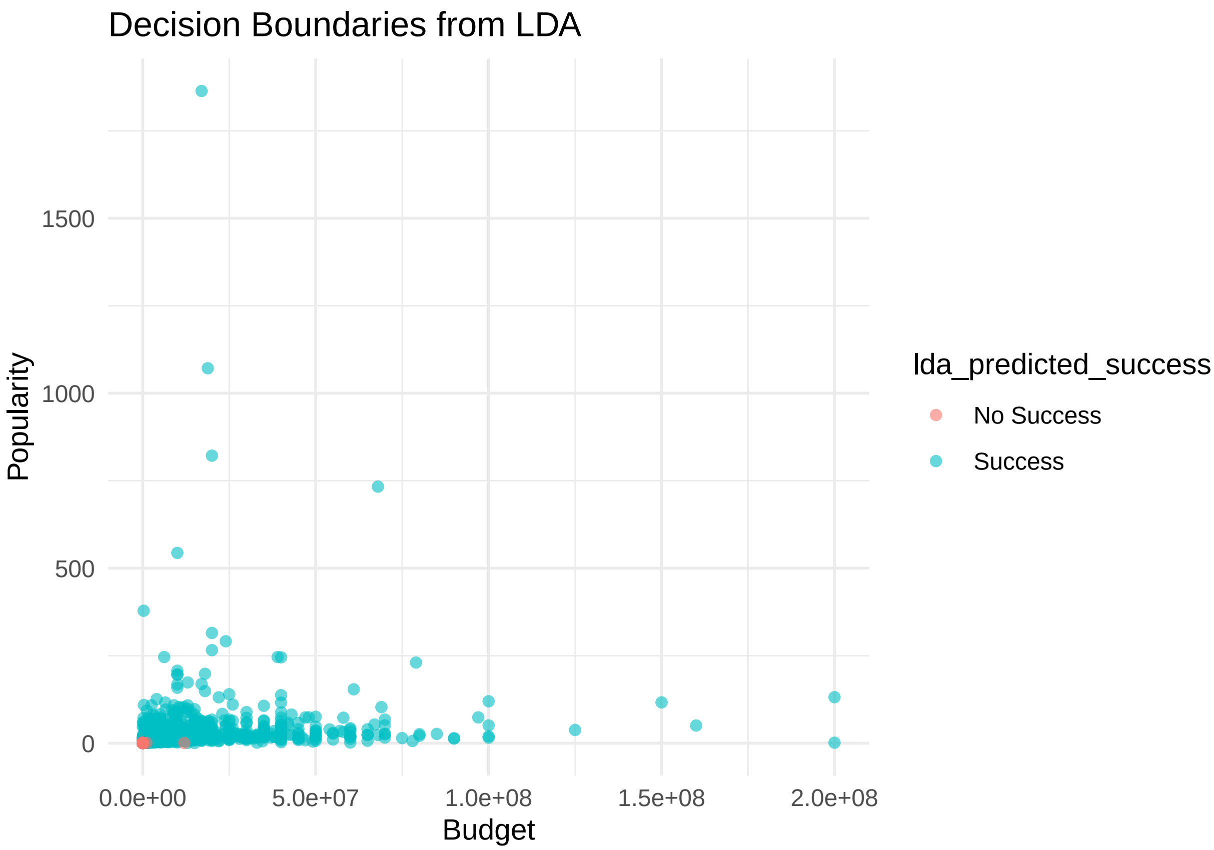
```
##              Actual
## Predicted_LDA No Success Success
##    No Success         49      37
##    Success           316     696
```

```
accuracy_LDA <- sum(predicted_classes_LDA == df_for_class$success) / nrow(df_for_class)
print(paste("Accuracy:", round(accuracy_LDA * 100, 4), "%"))
```

```
## [1] "Accuracy: 67.8506 %"
```

Visualizing Decision Boundaries

```
ggplot(df_for_class, aes(x = budget, y = popularity, color = lda_predicted_success)) +
  geom_point(alpha = 0.6) +
  labs(title = "Decision Boundaries from LDA", x = "Budget", y = "Popularity") +
  theme_minimal()
```



Classifying Success with QDA

Training the model

```
qda_model = qda(success ~ budget + release_year + runtime + popularity, data = df_for_class)

qda_model
```

```
## Call:
## qda(success ~ budget + release_year + runtime + popularity, data = df_for_class)
##
## Prior probabilities of groups:
## No Success    Success
##  0.3324226  0.6675774
##
## Group means:
##              budget release_year  runtime popularity
## No Success 10822277     2006.455 85.61096   17.15807
## Success    13549131     2001.524 94.57162   36.44876
```

Predicting

```
predictions_QDA = predict(qda_model, newdata = df_for_class)

predicted_classes_QDA = predictions_QDA$class

predicted_probs_QDA = predictions_QDA$posterior
```

```r
predictions_QDA_counts = table(predicted_classes_QDA)
print(predictions_QDA_counts)
```

```
## predicted_classes_QDA
## No Success    Success
##        442        656
```

Evaluating the Model

We will use the confusion matrix and accuracy calculations to show the accuracy of the model.

```r
confusion_matrix_QDA = table(Predicted_QDA = predicted_classes_QDA, Actual = df_for_class$success)
print(confusion_matrix_QDA)
```

```
##              Actual
## Predicted_QDA No Success Success
##    No Success        214     228
##    Success           151     505
```

```r
accuracy_QDA = sum(predicted_classes_QDA == df_for_class$success) / nrow(df_for_class)
print(paste("Accuracy:", round(accuracy_QDA * 100, 4), "%"))
```

```
## [1] "Accuracy: 65.4827 %"
```

Classifying Success with Naive Bayes:

Naive Bayes is a probabilistic model based on Bayes' Theorem. The model assumes that the features are conditionally independent given the target variable, success.

Training the Model

```r
library(e1071)

nb_model <- naiveBayes(success ~ budget + release_year + runtime + popularity, data = df_for_class)

print(nb_model)
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
## No Success    Success
##  0.3324226  0.6675774
##
## Conditional probabilities:
##             budget
## Y                 [,1]     [,2]
##   No Success 10822277 18135468
##   Success    13549131 20392214
##
##             release_year
```

```
## Y                [,1]     [,2]
##   No Success 2006.455 11.45590
##   Success    2001.524 15.98372
##
##             runtime
## Y                [,1]     [,2]
##   No Success 85.61096 32.37515
##   Success    94.57162 23.95629
##
##             popularity
## Y                [,1]     [,2]
##   No Success 17.15807 51.58351
##   Success    36.44876 91.63935
```

Predicting

```r
predictions_nb <- predict(nb_model, df_for_class)

predictions_nb_counts <- table(predictions_nb)
print(predictions_nb_counts)
```

```
## predictions_nb
## No Success    Success
##        431        667
```

Evaluating the Model

We will use a confusion matrix to show model accuracy.

```r
confusion_matrix_nb <- table(Predicted_NB = predictions_nb, Actual = df_for_class$success)
print(confusion_matrix_nb)
```

```
##             Actual
## Predicted_NB No Success Success
##   No Success        211     220
##   Success           154     513
```

```r
accuracy_nb <- sum(predictions_nb == df_for_class$success) / nrow(df_for_class)
print(paste("Accuracy: ", round(accuracy_nb *100, 4), "%"))
```

```
## [1] "Accuracy:  65.9381 %"
```

Classifying Success with Shrinkage:

Shrinkage methods we will use are Lasso and Ridge Regression. Lasso helps with feature selection and Ridge Regression helps with handling multicollinearity.

Ridge Regression

Ridge regression reduces variance in the presence of highly correlated predictors like budget and popularity, ensuring effective predictions. Although all predictors are kept, their coefficients are shrunk, reflecting their relative importance. For instance, popularity might have a higher coefficient than runtime, indicating its stronger influence on predicting success. This is useful because some filmmakers may want a holistic view of all contributing factors, even those with smaller effects.

Training the Model We will fit the Ridge model with cross-validation to find the optimal lambda. Then, we will use that best lambda to train the model.

```
library(glmnet)
```

```
## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 4.1-8
```

```
x <- model.matrix(success ~ budget + release_year + runtime + popularity, data = df_for_class)[, -1]
y <- ifelse(df_for_class$success == "Success", 1, 0)

set.seed(123)
ridge_cv <- cv.glmnet(x, y, alpha = 0, family = "binomial")

best_lambda_ridge <- ridge_cv$lambda.min
print(paste("Optimal Lambda for Ridge Regression: ", best_lambda_ridge))
```

```
## [1] "Optimal Lambda for Ridge Regression:  0.00739057100470154"
```

```
ridge_model <- glmnet(x, y, alpha = 0, family = "binomial", lambda = best_lambda_ridge)
```

```
ridge_probabilities <- predict(ridge_model, newx = x, type = "response")

ridge_predictions <- ifelse(ridge_probabilities > 0.5, 1, 0)


ridge_prediction_counts <- table(ridge_predictions)
print(ridge_prediction_counts)
```

Predicting

```
## ridge_predictions
##    0    1
##   86 1012
```

Evaluating the Model We will use a confusion matrix to evaluate the model accuracy.

```
ridge_confusion_matrix <- table(Predicted = ridge_predictions, Actual = y)
print(ridge_confusion_matrix)
```

```
##          Actual
## Predicted   0   1
##         0  48  38
##         1 317 695
```

```
ridge_accuracy <- sum(ridge_predictions == y) / length(y)
print(paste("Accuracy of Ridge Regression: ", round(ridge_accuracy * 100, 4), "%"))
```

```
## [1] "Accuracy of Ridge Regression:  67.6685 %"
```

```r
ridge_coefficients <- as.matrix(coef(ridge_model))
print(ridge_coefficients)
```

```
##                         s0
## (Intercept)   5.189632e+01
## budget        1.236525e-09
## release_year -2.594853e-02
## runtime       5.599066e-03
## popularity    1.164649e-02
```

Lasso Regression

Training the Model We will fit the Lasso model with cross-validation to find the optimal lambda. Then, using this best lambda we will train the Lasso model with it.

```r
set.seed(123)
lasso_cv <- cv.glmnet(x, y, alpha = 1, family = "binomial")

best_lambda_lasso <- lasso_cv$lambda.min
print(paste("Optimal Lambda for Lasso Regression: ", best_lambda_lasso))
```

```
## [1] "Optimal Lambda for Lasso Regression:  0.000849736071263926"
```

```r
lasso_model <- glmnet(x, y, alpha = 1, family = "binomial", lambda = best_lambda_lasso)
```

Predicting We will use a threshold of 0.5 to classify a success or not.

```r
lasso_probabilities <- predict(lasso_model, newx = x, type = "response")


lasso_predictions <- ifelse(lasso_probabilities > 0.5, 1, 0)

lasso_prediction_counts <- table(lasso_predictions)
print(lasso_prediction_counts)
```

```
## lasso_predictions
##   0   1
## 100 998
```

Evaluating the Model We will use a confusion matrix to evaluate the model accuracy.

```r
lasso_confusion_matrix <- table(Predicted = lasso_predictions, Actual = y)
print(lasso_confusion_matrix)
```

```
##          Actual
## Predicted   0   1
##         0  55  45
##         1 310 688
```

```r
lasso_accuracy <- sum(lasso_predictions == y) / length(y)
print(paste("Accuracy of Lasso Regression: ", round(lasso_accuracy * 100, 4), "%"))
```

```
## [1] "Accuracy of Lasso Regression:  67.6685 %"
```

```r
lasso_coefficients <- as.matrix(coef(lasso_model))
print(lasso_coefficients)
```

```
##                         s0
## (Intercept)  56.108532296
## budget        0.000000000
## release_year -0.028048515
## runtime       0.004534481
## popularity    0.016929854
```

Lasso regression automatically sets some coefficients to zero, removing less important predictors like budget if they do not significantly contribute to the model. By focusing on a smaller set of predictors, Lasso provides a more interpretable model. For example, it reveals that release_year, runtime, and popularity alone are sufficient to predict success. This helps filmmakers focus on the most influential factors, reducing unnecessary expenditures on less critical aspects to make a movie successful.

Classifying Success with Logistic Regression:

Objective is to predict whether a movie is successful using logistic regression based on budget, release year, runtime, and popularity.

Training the model

```r
lg_model <- glm(success ~ budget + release_year + runtime + popularity,
                     data = df_for_class,
                     family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
summary(lg_model)
```

```
##
## Call:
## glm(formula = success ~ budget + release_year + runtime + popularity,
##     family = binomial, data = df_for_class)
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)   5.706e+01  1.028e+01   5.550 2.86e-08 ***
## budget       -2.637e-10  3.837e-09  -0.069   0.9452
## release_year -2.853e-02  5.108e-03  -5.585 2.34e-08 ***
## runtime       4.507e-03  2.564e-03   1.758   0.0787 .
## popularity    1.776e-02  3.385e-03   5.249 1.53e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1396.4  on 1097  degrees of freedom
## Residual deviance: 1303.8  on 1093  degrees of freedom
## AIC: 1313.8
##
## Number of Fisher Scoring iterations: 6
```

Predicting:

```r
probabilities_lg = predict(lg_model, newdata = df_for_class, type = "response")
```

```r
predictions_lg = ifelse(probabilities_lg > 0.5, 1, 0)

predictions_lg_counts = table(predictions_lg)
print(predictions_lg_counts)
```

```
## predictions_lg
##   0   1
## 102 996
```

Evaluating the mode

```r
predictions_lg <- factor(predictions_lg, levels = c(0, 1), labels = c("No Success", "Success"))

confusion_matrix_lg <- table(Predicted_lg = predictions_lg, Actual = df_for_class$success)
print(confusion_matrix_lg)
```

```
##             Actual
## Predicted_lg No Success Success
##   No Success         55      47
##   Success           310     686
```

```r
accuracy_lg <- sum(predictions_lg == df_for_class$success) / length(predictions_lg)
print(paste("Accuracy: ", round(accuracy_lg * 100, 4), "%"))
```

```
## [1] "Accuracy:  67.4863 %"
```

The confusion matrix shows the distribution of correct and incorrect predictions. The accuracy percentage provides a measure of how well the model predicts movie success.

Classifying Budget into four predicted groups with multinomial logistic regression:

```r
if (!require("nnet")) install.packages("nnet")
library(nnet)
```

Preparing target

```r
budget_quartiles <- quantile(df_for_class$budget, probs = c(0, 0.25, 0.5, 0.75, 1), na.rm = TRUE)
budget_quartiles <- unique(budget_quartiles)  # Remove duplicate values

if (length(budget_quartiles) - 1 != 4) {
  stop("Unable to create exactly 4 quartile groups due to duplicate breaks. Please inspect the data.")
}

# Assign budget categories
df_for_class$budget_category <- cut(
  df_for_class$budget,
  breaks = budget_quartiles,
  labels = c("Low", "Medium", "High", "Very High"),
  include.lowest = TRUE
)

table(df_for_class$budget_category)
```

```
##
```

```
##       Low    Medium      High Very High
##       293       256       288       261
```

Fitting the model

```r
df_for_class$budget_category <- as.factor(df_for_class$budget_category)

multinom_model <- multinom(budget_category ~ revenue + release_year + popularity, data = df_for_class)
```

```
## # weights:  20 (12 variable)
## initial  value 1522.151209
## iter  10 value 1383.473742
## iter  20 value 1306.572797
## iter  30 value 1305.826683
## iter  40 value 1305.723097
## iter  50 value 1305.667329
## iter  60 value 1305.656597
## iter  60 value 1305.656593
## iter  70 value 1305.653550
## final  value 1305.653469
## converged
```

Check the model summary

```r
summary(multinom_model)
```

```
## Call:
## multinom(formula = budget_category ~ revenue + release_year +
##     popularity, data = df_for_class)
##
## Coefficients:
##           (Intercept)      revenue release_year popularity
## Medium       2.276862 1.432285e-08 -0.001637166 0.05107461
## High        -6.214695 2.652341e-08  0.002459970 0.05361715
## Very High  -11.708963 3.579893e-08  0.004854702 0.05479629
##
## Std. Errors:
##            (Intercept)      revenue release_year   popularity
## Medium    1.223018e-16 4.280678e-09 2.445149e-13 2.536082e-15
## High      1.151017e-16 4.093087e-09 2.302143e-13 2.035855e-15
## Very High 8.763339e-17 4.091328e-09 1.753910e-13 1.715601e-15
##
## Residual Deviance: 2611.307
## AIC: 2635.307
```

Predict classifications

```r
predicted_categories = predict(multinom_model, newdata = df_for_class)

category_counts = table(predicted_categories)
print(category_counts)
```

```
## predicted_categories
##       Low    Medium      High Very High
```

```
##       536       121       240       201
```

Evaluating model with confusion matrix and calculating accuracy

```r
table(Predicted = predicted_categories, Actual = df_for_class$budget_category)
```

```
##            Actual
## Predicted   Low Medium High Very High
##   Low       255    142  108        31
##   Medium     16     53   39        13
##   High       15     39   96        90
##   Very High   7     22   45       127
```

```r
accuracy <- mean(predicted_categories == df_for_class$budget_category)
cat("Accuracy: ", round(accuracy * 100, 4), "%")
```

```
## Accuracy:  48.3607 %
```

Classification with Emphasis on Interpretation

Classifying Success with Decision Trees

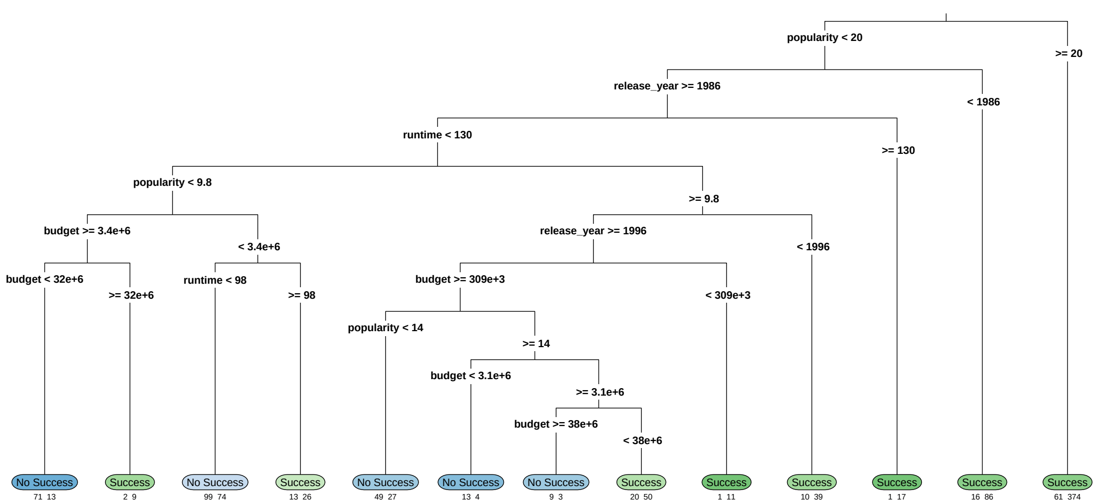
The decision tree structure shows how features like budget and popularity split the data to classify movies. The path from root to leaf highlights the decision rules. This easily identifies the most important features based on the splits.

Fit the Model

```r
tree_model <- rpart(success ~ budget + release_year + runtime + popularity, data = df_for_class, method
```

Visualize the Trees

```r
if (!require("rpart.plot")) install.packages("rpart.plot")
library(rpart.plot)
rpart.plot(tree_model, type = 3, extra = 101, under = TRUE, fallen.leaves = TRUE)
```



Predicting

```
tree_predictions <- predict(tree_model, newdata = df_for_class, type = "class")
```

Evaluating the Model

Use the confusion matrix to evaluate the model.

```
tree_confusion <- table(Predicted = tree_predictions, Actual = df_for_class$success)
accuracy_tree <- sum(tree_predictions == df_for_class$success) / nrow(df_for_class)

cat("Decision Tree Confusion Matrix:\n")
```

```
## Decision Tree Confusion Matrix:
```

```
print(tree_confusion)
```

```
##              Actual
## Predicted     No Success Success
##   No Success         241     121
##   Success            124     612
```

```
cat("Decision Tree Accuracy: ", round(accuracy_tree * 100, 4), "%\n")
```

```
## Decision Tree Accuracy:  77.6867 %
```

Classifying Success with Random Forest (with Feature Importance)

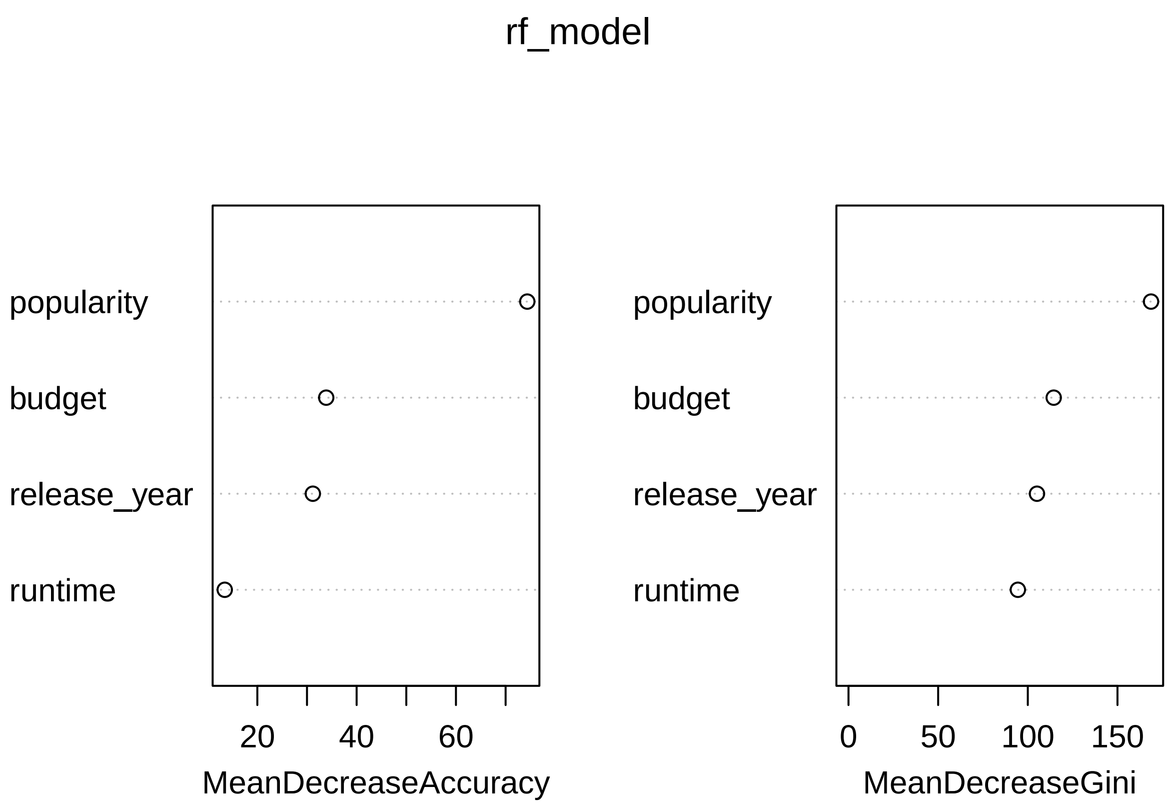
Fit the Model

```
library(randomForest)

rf_model <- randomForest(success ~ budget + release_year + runtime + popularity, data = df_for_class, in
```

View Variable Importance

This shows which features contribute most to the model.

```
importance <- importance(rf_model)
varImpPlot(rf_model)
```



Prediction

```
rf_predictions <- predict(rf_model, newdata = df_for_class)
```

Evaluating the Model

Use a confusion matrix to evaluate the model.

```
rf_confusion <- table(Predicted = rf_predictions, Actual = df_for_class$success)
accuracy_rf <- sum(rf_predictions == df_for_class$success) / nrow(df_for_class)

cat("Random Forest Confusion Matrix:\n")
```

```
## Random Forest Confusion Matrix:
```

```
print(rf_confusion)
```

```
##              Actual
## Predicted     No Success Success
##   No Success         365       2
##   Success              0     731
```

```
cat("Random Forest Accuracy: ", round(accuracy_rf * 100, 4), "%\n")
```

```
## Random Forest Accuracy:  99.8179 %
```

Classifying Success with Neural Networks

In the context of movie success classification, the neural network captures nonlinear relationships between features like budget and popularity interacting in unexpected ways.

Fit the Model

```r
set.seed(123)
df_for_class$success <- as.factor(df_for_class$success)
trainIndex <- createDataPartition(df_for_class$success, p = 0.8, list = FALSE)
trainData <- df_for_class[trainIndex, ]
testData <- df_for_class[-trainIndex, ]

nn_model <- nnet(success ~ budget + release_year + runtime + popularity,
                 data = trainData,
                 size = 5,
                 decay = 0.01,
                 maxit = 500)
```

```
## # weights:  31
## initial  value 776.946860
## iter  10 value 552.486494
## iter  20 value 551.074786
## iter  30 value 550.495807
## iter  40 value 549.056962
## iter  50 value 548.452636
## iter  60 value 548.303236
## iter  70 value 547.186584
## iter  80 value 546.182894
## iter  90 value 544.703541
## iter 100 value 544.516248
## iter 110 value 544.126308
## iter 120 value 543.488494
## iter 130 value 543.010122
## iter 140 value 542.829330
## iter 150 value 542.819757
## final  value 542.819722
## converged
```

```r
print(summary(nn_model))
```

```
## a 4-5-1 network with 31 weights
## options were - entropy fitting  decay=0.01
##  b->h1 i1->h1 i2->h1 i3->h1 i4->h1
##  -0.02   0.36  -0.05   0.92   2.27
##  b->h2 i1->h2 i2->h2 i3->h2 i4->h2
##   0.00   0.07  -0.18  -0.34   0.00
##  b->h3 i1->h3 i2->h3 i3->h3 i4->h3
##   0.03  -1.35   0.01  -3.30  -2.84
##  b->h4 i1->h4 i2->h4 i3->h4 i4->h4
##   0.00   0.02  -0.13   0.03   0.00
##  b->h5 i1->h5 i2->h5 i3->h5 i4->h5
##   0.00  -0.02   0.08   0.01   0.00
##  b->o h1->o h2->o h3->o h4->o h5->o
##  2.13  1.87 -4.04  1.55  0.84 -3.20
```

Prediction

```r
nn_predictions <- predict(nn_model, newdata = testData, type = "class")
```

Evaluating the Model

Use the confusion matrix to evaluate model accuracy.

```
confusion_matrix_nn <- table(Predicted = nn_predictions, Actual = testData$success)
print(confusion_matrix_nn)
```

```
##             Actual
## Predicted    No Success Success
##   No Success          6       4
##   Success            67     142
```

```
accuracy_nn <- sum(diag(confusion_matrix_nn)) / sum(confusion_matrix_nn)
print(paste("Accuracy: ", round(accuracy_nn * 100, 4), "%"))
```

```
## [1] "Accuracy:  67.5799 %"
```

Classifying with Gradient Boosting

Gradient boosting combines multiple weak learners (decision trees) to create a strong predictive model.

Fit the Model

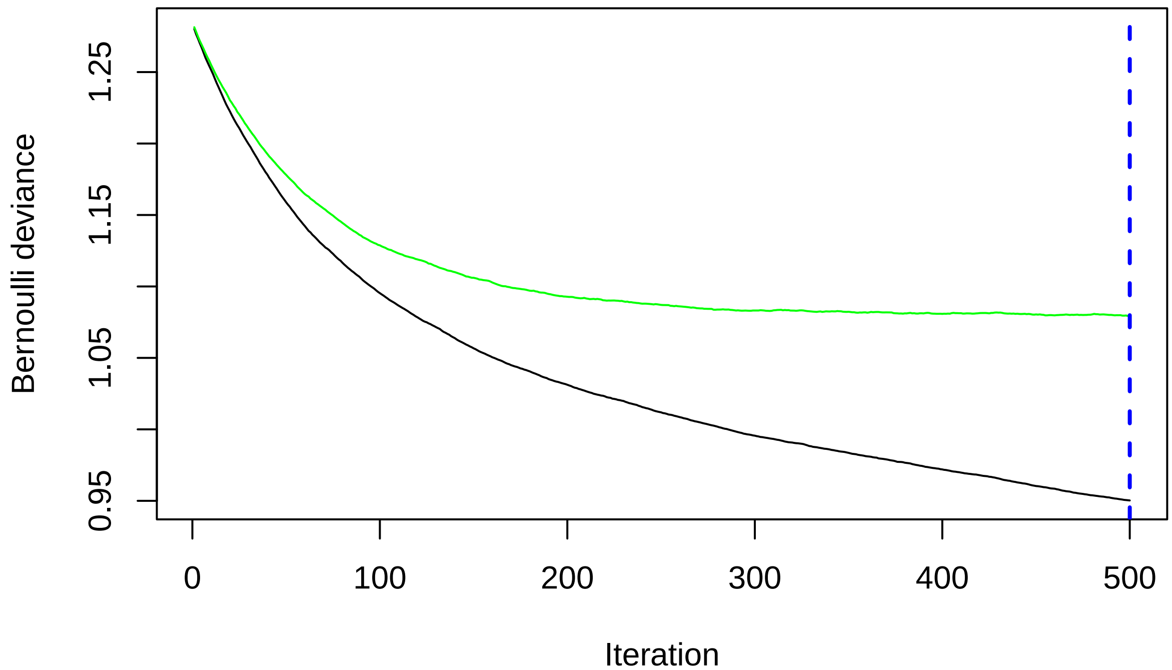
We will use a bernoulli distribution since we are working with binary classification.

```
df_for_class$success <- ifelse(df_for_class$success == "Success", 1, 0)

set.seed(123)
trainIndex <- createDataPartition(df_for_class$success, p = 0.8, list = FALSE)
trainData <- df_for_class[trainIndex, ]
testData <- df_for_class[-trainIndex, ]

set.seed(123)
gbm_model <- gbm(success ~ budget + release_year + runtime + popularity,
                 data = trainData,
                 distribution = "bernoulli",
                 n.trees = 500,
                 interaction.depth = 3,
                 shrinkage = 0.01,
                 cv.folds = 5)
```

Prediction

We will find the best number of decision trees to use for the probabilities using cross validation.

```
best_iter <- gbm.perf(gbm_model, method = "cv")
```




```
gbm_probabilities <- predict(gbm_model, newdata = testData, n.trees = best_iter, type = "response")
gbm_predictions <- ifelse(gbm_probabilities > 0.5, "Success", "No Success")
```

Evaluating the Model

Use the confusion matrix to evaluate the model accuracy.

```
confusion_matrix_gbm <- table(Predicted = gbm_predictions, Actual = testData$success)
print(confusion_matrix_gbm)
```

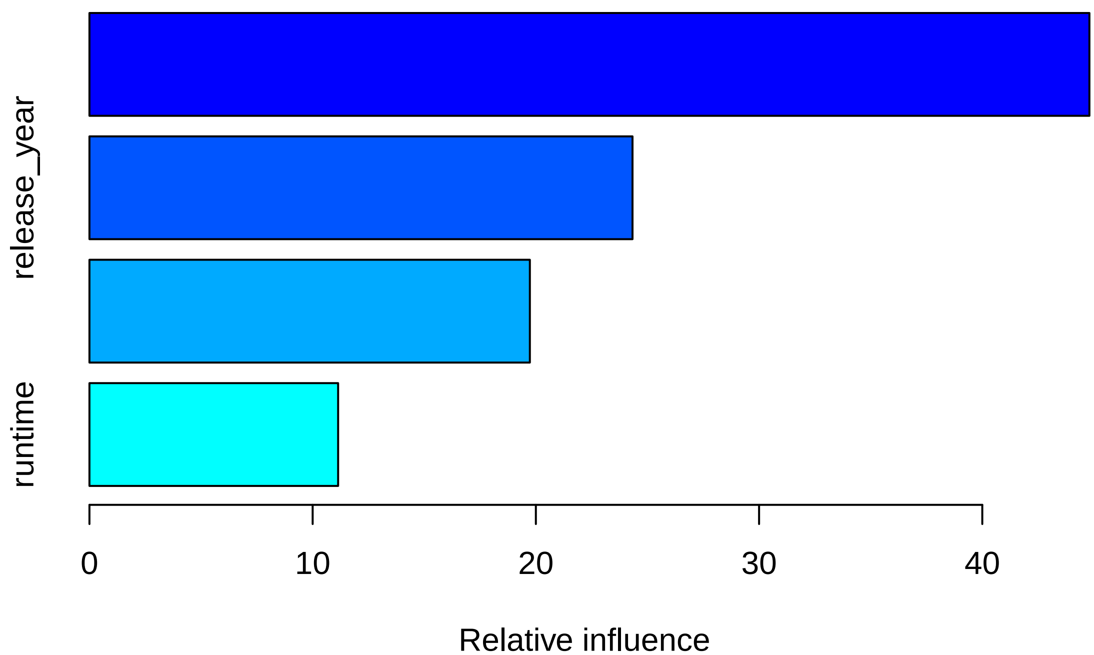
```
##             Actual
## Predicted      0   1
##   No Success  29  24
##   Success     36 130
```

```
accuracy_gbm <- sum(diag(confusion_matrix_gbm)) / sum(confusion_matrix_gbm)
print(paste("Accuracy: ", round(accuracy_gbm * 100, 4), "%"))
```

```
## [1] "Accuracy:  72.6027 %"
```

Feature Importance

```
importance <- summary(gbm_model)
```




```
print(importance)
```

```
##                       var  rel.inf
## popularity     popularity 44.79918
## release_year release_year 24.32952
## budget             budget 19.73093
## runtime           runtime 11.14037
```

Prediction using Logistic regression answers the question, "Can we predict success?"

Interpretation using LDA highlights "Why are some movies predicted as successful or unsuccessful?" by examining variable relationships and decision boCundaries.

Feature Selection and Comparison of Predictor Sets

Feature Importance Analysis

Correlation Analysis for Continuous Variables

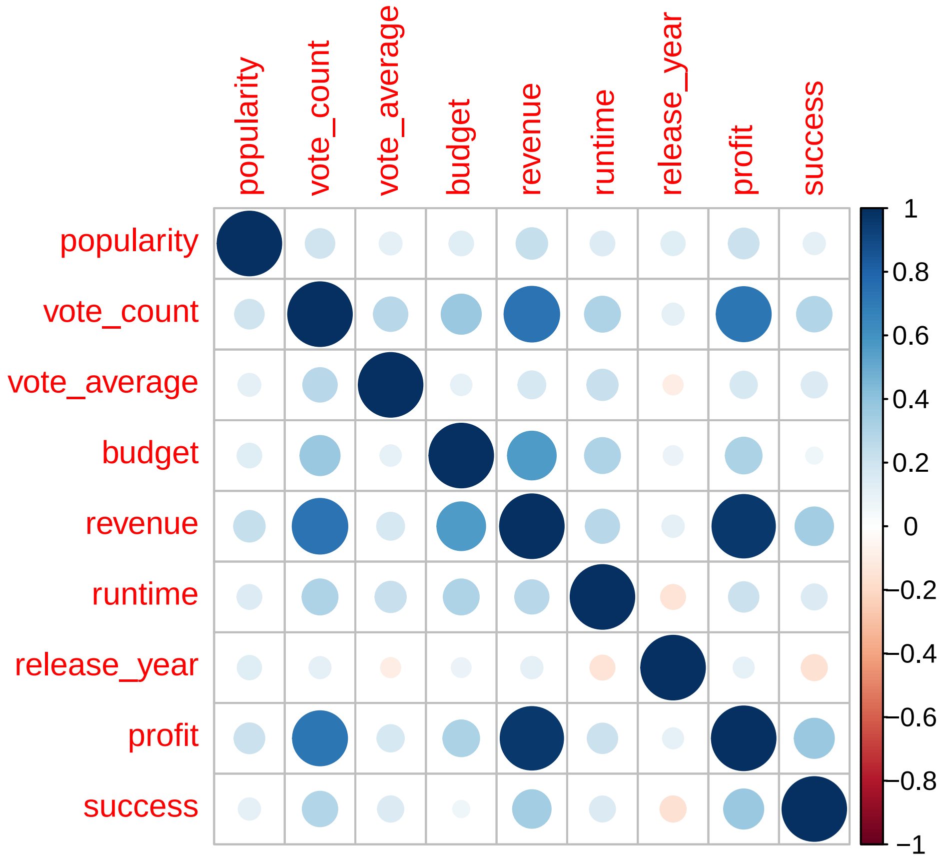
We will analyze the correlation between predictors and the target variable, success. We will identify multicollinearity among predictors to avoid redudancy. We will do so by looking at a correlation matrix for numeric predictors.

```
numeric_vars <- df_for_class[, sapply(df_for_class, is.numeric)]
correlation_matrix <- cor(numeric_vars)
library(corrplot)
```

```
## Warning: package 'corrplot' was built under R version 4.3.3
```

```
## corrplot 0.94 loaded
```

```
corrplot(correlation_matrix, method = "circle")
```



Interpretation: (ADD with context)

- Strong correlations (close to ±1) between a predictor and `success` suggest high relevance.
- Avoid using highly correlated predictors simultaneously to prevent redundancy.

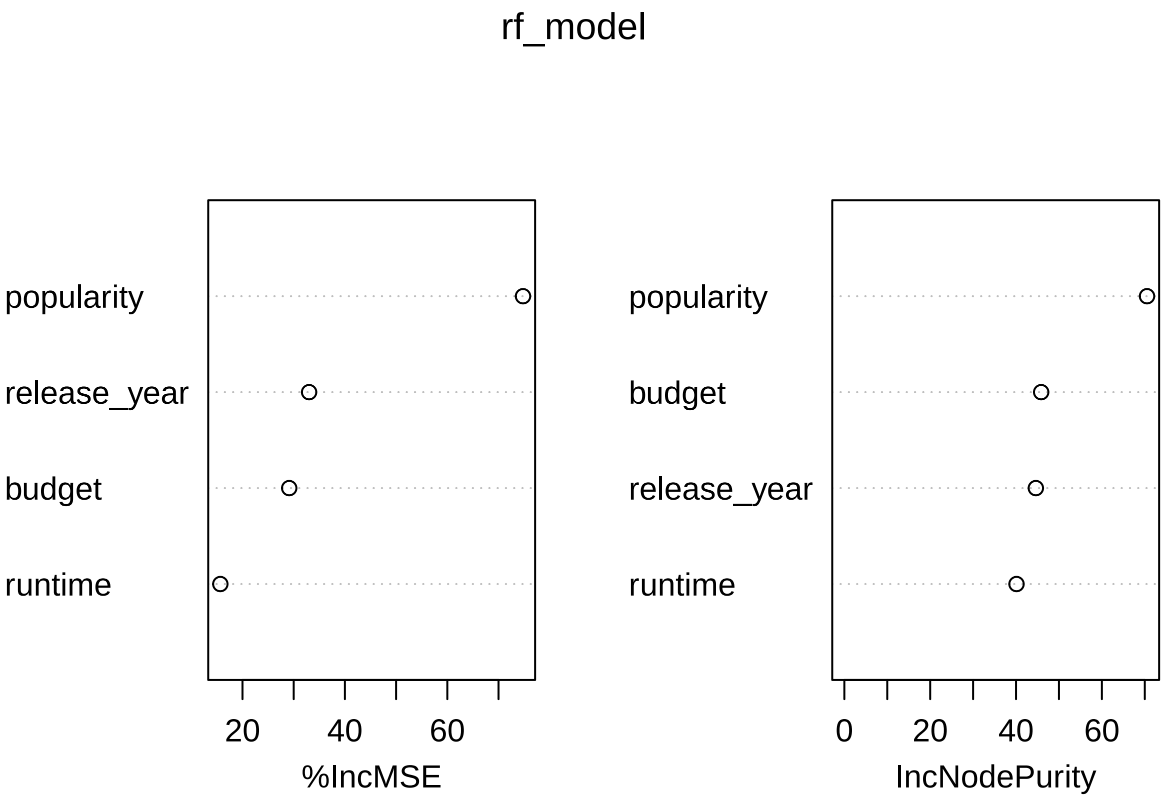
Variable Importance from Random Forest

Random Forest provides a direct measure of feature importance.

```
rf_model <- randomForest(success ~ budget + release_year + runtime + popularity, data = df_for_class, i
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values.  Are you sure you want to do regression?
```

```
varImpPlot(rf_model)
```

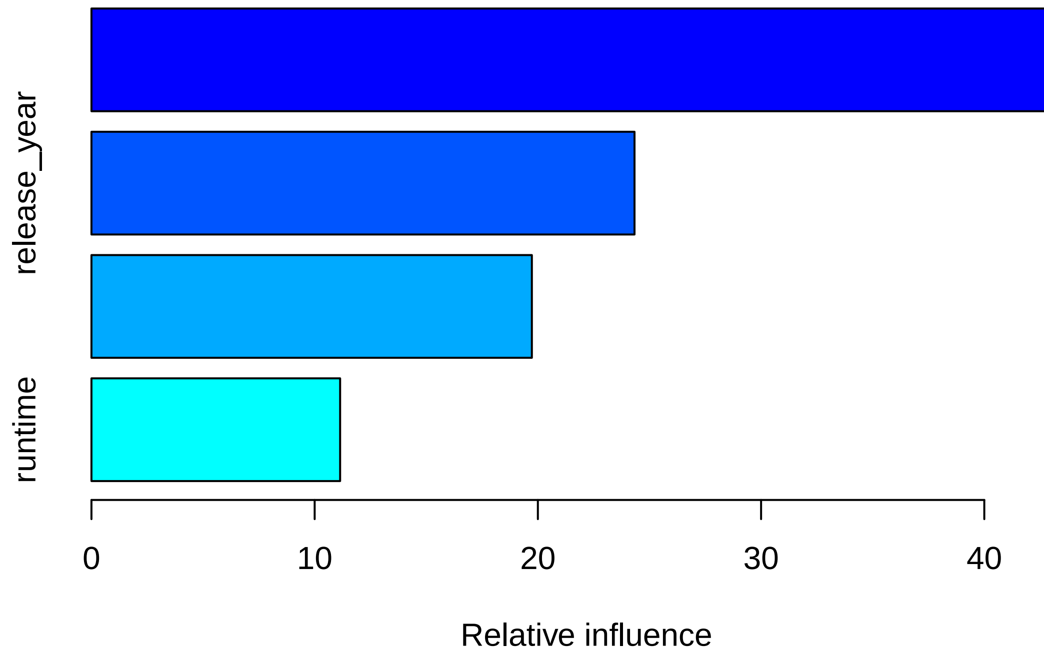


Interpretation: (ADD with context)

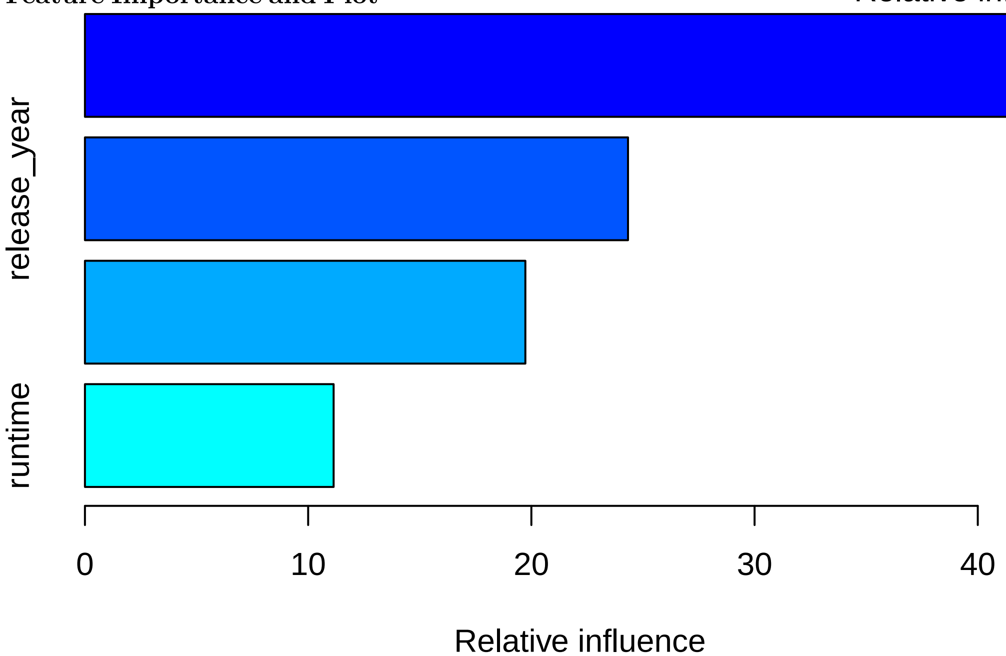
- The variable importance plot ranks features based on their contribution to classification accuracy.
- Features with higher Mean Decrease Accuracy or Mean Decrease Gini should be prioritized.

Gradient Boosting Feature Importance and Partial Dependence

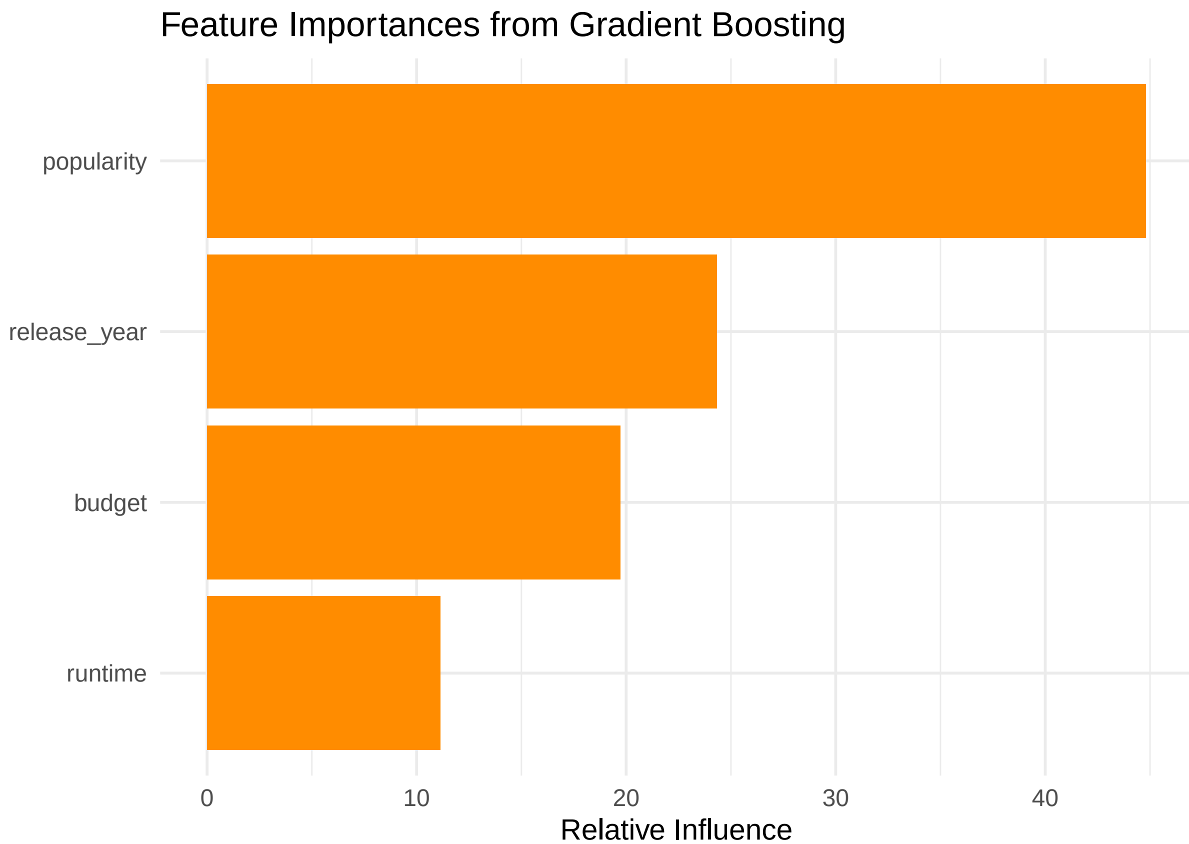
```
importance_df_gbm <- data.frame(
  Feature = summary(gbm_model)$var,
  Importance = summary(gbm_model)$rel.inf
)
```



Feature Importance and Plot




```r
# Plot Feature Importances
ggplot(importance_df_gbm, aes(x = reorder(Feature, Importance), y = Importance)) +
  geom_bar(stat = "identity", fill = "darkorange") +
  coord_flip() +
  ggtitle("Feature Importances from Gradient Boosting") +
  xlab("") +
  ylab("Relative Influence") +
  theme_minimal()
```



ADD Interpretations

```r
# Assuming 'trainData' is your dataset used to train the 'gbm_model'

# Generate the partial dependence plot for 'runtime'
pdp_runtime <- partial(gbm_model,
                       pred.var = "runtime",
                       n.trees = gbm.perf(gbm_model, method = "cv"),
                       train = trainData)  # Ensure the training data is passed

# Plot Partial Dependence Plot with rug
autoplot(pdp_runtime, rug = TRUE) +
  ggtitle("Partial Dependence Plot for Runtime") +
  xlab("Runtime") +
  ylab("Predicted Probability of Success") +
  theme_minimal()
```

Partial Dependence Plot

Stepwise Selection

We can use stepwise regression to identify the most relevant features for logistic regression.

```r
full_model <- glm(success ~ budget + release_year + runtime + popularity, data = df_for_class, family =
step_model <- stepAIC(full_model, direction = "both")
summary(step_model)
```

Interpretation: (ADD with context)

- Features retained in the final model are likely to be the most predictive.

SHAP Values for Model Interpretability

```r
predict_function <- function(object, newdata) {
  predict(object, newdata = newdata, type = "prob")[, "Success"]
}

# Explanatory dataset
X <- horror[, c("budget", "release_year", "runtime", "popularity")]
y <- horror$success
```

SHAP Analysis with Random Forest

```r
# Convert 'success' to a factor if it's not already
df_for_class$success <- factor(df_for_class$success, levels = c(0, 1), labels = c("No Success", "Succes
# Define the prediction function for SHAP
predict_function <- function(object, newdata) {
  predict(object, newdata = newdata, type = "prob")[, "Success"]  # Get probabilities for 'Success' cla
}

# Compute SHAP values
set.seed(123)
shap_values <- fastshap::explain(
  object = rf_model,
  X = df_for_class[, c("budget", "release_year", "runtime", "popularity")],  # Use the feature columns
  pred_wrapper = predict_function,
  nsim = 50,
  adjust = TRUE
)

# Mean Absolute SHAP values
mean_abs_shap <- colMeans(abs(shap_values))
shap_importance <- data.frame(
  Feature = names(mean_abs_shap),
  MeanAbsShap = mean_abs_shap
)
```

Compute SHAP Values + Mean Absolute SHAP Values

```r
ggplot(shap_importance, aes(x = reorder(Feature, MeanAbsShap), y = MeanAbsShap)) +
  geom_bar(stat = "identity", fill = "purple") +
  coord_flip() +
  ggtitle("SHAP Feature Importance for Random Forest") +
  xlab("") +
  ylab("Mean |SHAP Value|") +
  theme_minimal()
```

SHAP Feature Importance Plot

```r
ggplot(data = data.frame(
  SHAP_value = shap_values$budget,
  Feature_value = X$budget
), aes(x = Feature_value, y = SHAP_value)) +
  geom_point(alpha = 0.6) +
  geom_smooth(method = "loess", se = FALSE, color = "blue") +
  ggtitle("SHAP Dependence Plot for Budget") +
  xlab("Budget") +
  ylab("SHAP Value") +
  theme_minimal()
```

SHAP Dependence Plot

Multiple Regression with t-values for Variable Importance

```r
# View the summary of the model to check t-values and p-values
summary(lm_model)

# The t-values are listed in the "t value" column of the summary.
# High t-values indicate important features, low t-values indicate less important features.
```

- A **higher absolute t-value** (greater than 2 or less than -2) indicates that the predictor is more significant.
- A **low t-value** (near 0) suggests that the predictor doesn't contribute much to the model and may be removed.

Recursive Feature Elimination (RFE)

```r
# Define the control parameters for RFE
ctrl <- rfeControl(functions = rfFuncs, method = "cv", number = 10)

# Perform Recursive Feature Elimination using Random Forest as the model
rfe_result <- rfe(df_for_class[, c("budget", "release_year", "runtime", "popularity")],
                  df_for_class$success,
                  sizes = c(1:4),  # Try selecting from 1 to 4 features
                  rfeControl = ctrl)

# View the results
print(rfe_result)

# Plot the results
plot(rfe_result, type = c("g", "o"))
```

Comparing Predictor Sets

Base Set of Predictors

We will use all available predictors: budget, release_year, runtime, and popularity.

```r
base_model <- glm(success ~ budget + release_year + runtime + popularity, data = df_for_class, family =
base_probs <- predict(base_model, type = "response")
base_preds <- ifelse(base_probs > 0.5, 1, 0)
```

```
base_accuracy <- mean(base_preds == df_for_class$success)
cat("Base Model Accuracy: ", base_accuracy, "\n")
```

Reduced Set of Predictors

We will now use only the most important predictors identifies through feature selection **(ADD WHEN FOUND ABOVE).**

```
#EXAMPLE BUT CHANGE WHEN FIND PREDICTORS TO USE
reduced_model <- glm(success ~ budget + popularity, data = df_for_class, family = binomial)
reduced_probs <- predict(reduced_model, type = "response")
reduced_preds <- ifelse(reduced_probs > 0.5, 1, 0)
reduced_accuracy <- mean(reduced_preds == df_for_class$success)
cat("Reduced Model Accuracy: ", reduced_accuracy, "\n")
```

Comparing Models with Different Predictor Sets

```
# Load necessary libraries
library(randomForest)
library(caret)
library(ROCR)

# Define the different predictor sets
predictor_set_1 <- c("budget", "release_year", "runtime")
predictor_set_2 <- c("budget", "release_year", "popularity")
predictor_set_3 <- c("budget", "runtime", "popularity")
predictor_set_4 <- c("budget", "release_year", "runtime", "popularity")

# Define a function to train Random Forest and return performance metrics
train_rf_model <- function(predictors, data) {
  # Train Random Forest model
  model <- randomForest(success ~ ., data = data[, c(predictors, "success")], ntree = 500)

  # Get predicted probabilities for the test data
  prob <- predict(model, type = "prob")[,2]

  # Calculate AUC
  pred <- prediction(prob, data$success)
  perf <- performance(pred, measure = "auc")
  auc <- perf@y.values[[1]]

  # Return AUC
  return(auc)
}

# Define a function to train Multiple Linear Regression and return performance metrics
train_lm_model <- function(predictors, data) {
  # Train Linear Model
  model <- lm(success ~ ., data = data[, c(predictors, "success")])

  # Get predicted probabilities
  prob <- predict(model, type = "response")

  # Convert probabilities to class labels (Success = 1, No Success = 0)
```

```r
  pred_labels <- ifelse(prob > 0.5, 1, 0)

  # Calculate Accuracy
  accuracy <- mean(pred_labels == data$success)

  # Return Accuracy
  return(accuracy)
}

# Train and evaluate models with different predictor sets for Random Forest
rf_auc_1 <- train_rf_model(predictor_set_1, df_for_class)
rf_auc_2 <- train_rf_model(predictor_set_2, df_for_class)
rf_auc_3 <- train_rf_model(predictor_set_3, df_for_class)
rf_auc_4 <- train_rf_model(predictor_set_4, df_for_class)

# Train and evaluate models with different predictor sets for Multiple Linear Regression
lm_accuracy_1 <- train_lm_model(predictor_set_1, df_for_class)
lm_accuracy_2 <- train_lm_model(predictor_set_2, df_for_class)
lm_accuracy_3 <- train_lm_model(predictor_set_3, df_for_class)
lm_accuracy_4 <- train_lm_model(predictor_set_4, df_for_class)

# Create a summary of model performance
performance_comparison <- data.frame(
  Model = c("Random Forest (Set 1)", "Random Forest (Set 2)", "Random Forest (Set 3)", "Random Forest (S
            "Linear Regression (Set 1)", "Linear Regression (Set 2)", "Linear Regression (Set 3)", "Lin
  AUC_or_Accuracy = c(rf_auc_1, rf_auc_2, rf_auc_3, rf_auc_4,
                      lm_accuracy_1, lm_accuracy_2, lm_accuracy_3, lm_accuracy_4)
)

# Print performance comparison
print(performance_comparison)
```

Plotting Performance Comparison

```r
# Plot the comparison
library(ggplot2)

ggplot(performance_comparison, aes(x = Model, y = AUC_or_Accuracy, fill = Model)) +
  geom_bar(stat = "identity", show.legend = FALSE) +
  coord_flip() +
  theme_minimal() +
  ggtitle("Comparison of Model Performance across Different Predictor Sets") +
  xlab("Model and Predictor Set") +
  ylab("AUC (RF) / Accuracy (LM)")
```

Adding Interaction Terms

We can test whether interaction terms improve model performance.

```r
interaction_model <- glm(success ~ budget * popularity + runtime, data = df_for_class, family = binomial
interaction_probs <- predict(interaction_model, type = "response")
interaction_preds <- ifelse(interaction_probs > 0.5, 1, 0)
interaction_accuracy <- mean(interaction_preds == df_for_class$success)
cat("Interaction Model Accuracy: ", interaction_accuracy, "\n")
```

Cross-Validation to Compare Models

Use cross-validation to evaluate the generalizability of each predictor set.

```
# Base Model
base_cv <- train(success ~ budget + release_year + runtime + popularity, data = df_for_class, method =
print(base_cv)

# Reduced Model
reduced_cv <- train(success ~ budget + popularity, data = df_for_class, method = "glm", family = binomia
print(reduced_cv)
```

Visualize Feature Importance

Effect Plots for Logistic Regression

Visualize the effect of individual predictors on the probability of success.

```
install.packages("effects")
library(effects)
```

```
effect_plot <- allEffects(reduced_model)
plot(effect_plot)
```

Partial Dependence Plots

Use this for tree based models like Random Forest.

```
install.packages("pdp")
library(pdp)
```

```
# Partial dependence for "budget"
pd_budget <- partial(rf_model, pred.var = "budget")
plotPartial(pd_budget)
```

```
pd_popularity <- partial(rf_model, pred.var = "popularity")
plotPartial(pd_popularity)
```