

# SLHW2

Ava Exelbirt, Sam Reade

2024-12-01

```
# 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")
# install.packages("gbm")
# install.packages("fastshap")
#install.packages("effects")

# 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

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

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
library(GGally)

## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg      ggplot2
library(caret)

## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##   lift
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.com
library(MASS)

##
## Attaching package: 'MASS'
##
## The following object is masked from 'package:dplyr':
##
##   select
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
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
library(fastshap)

##
## Attaching package: 'fastshap'
##
## The following object is masked from 'package:dplyr':
##
##   explain
library(effects)

## Loading required package: carData

## Warning in check_dep_version(): ABI version mismatch:
## lme4 was built with Matrix ABI version 1
## Current Matrix ABI version is 0
## Please re-install lme4 from source or restore original 'Matrix' package

## Use the command
##   lattice::trellis.par.set(effectsTheme())
## to customize lattice options for effects plots.
## See ?effectsTheme for details.
```

## About the Data

### Import Data

```
#tuesdata <- tidyuesdayR::tt_load('2022-11-01')
tuesdata <- tidyuesdayR::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"
horror <- tuesdata$horror_movies

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> "/5GA3vV1aWWHTSD05eno8V5zDo8r.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

```
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.

```
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
```

```
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
```

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

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

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

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

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

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

```
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`.

```
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.

```
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.

```

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:

```

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

```

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

```

## Making data frame with no 0s

```

# Create a new data frame without rows where revenue and budget 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>
```

```
dim(df_for_class)
```

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

## 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.

```
set.seed(123)
```

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

```
nrow(training)
```

```
## [1] 659
```

```
nrow(testing)
```

```
## [1] 439
```

```
summary(training)
```

```
## original_title      title      original_language  overview
## Length:659          Length:659          en      :560    Length:659
## Class :character     Class :character     ja      : 16      Class :character
## Mode  :character     Mode  :character     es      : 14      Mode  :character
##                               hi      : 14
##                               ko      : 10
##                               de      :  8
##                               (Other): 37
## tagline              release_date      popularity      vote_count
## Length:659          Min.   :1953-06-05  Min.   :  0.600  Min.   :  0.0
## Class :character     1st Qu.:1994-12-12  1st Qu.:  7.186  1st Qu.:  98.5
## Mode  :character     Median :2007-01-19  Median : 15.516  Median : 543.0
##                               Mean   :2003-08-09  Mean   : 27.011  Mean   :1206.6
##                               3rd Qu.:2015-01-28  3rd Qu.: 31.030  3rd Qu.:1544.5
##                               Max.   :2022-09-29  Max.   :1071.398  Max.   :16900.0
##
## vote_average         budget         revenue         runtime
## Min.   : 0.000       Min.   : 1       Min.   : 1       Min.   : 0.00
## 1st Qu.: 5.300       1st Qu.: 1000000  1st Qu.: 675326  1st Qu.: 87.50
## Median : 6.000       Median : 5000000  Median : 11642254 Median : 95.00
## Mean   : 5.797       Mean   : 12769148 Mean   : 39044327 Mean   : 91.09
## 3rd Qu.: 6.600       3rd Qu.: 15000000 3rd Qu.: 45023606 3rd Qu.:103.00
## Max.   :10.000       Max.   :200000000 Max.   :701842551 Max.   :153.00
##
## status      adult      genre_names
## In Production : 0      Mode :logical Horror, Thriller      :103
```

```
## Planned      : 0   FALSE:659      Horror      : 99
## Post Production: 0      Horror, Mystery, Thriller: 52
## Released      :659      Comedy, Horror      : 36
##              Horror, Science Fiction : 29
##              Drama, Horror, Thriller : 27
##              (Other)                  :313
## release_year  budget_category      profit      success
## Min.   :1953   Length:659      Min.   :-194775779   No Success:212
## 1st Qu.:1994   Class :character  1st Qu.:  -99162    Success :447
## Median :2007   Mode  :character  Median :   3400000
## Mean   :2003                      Mean   :  26275179
## 3rd Qu.:2015                      3rd Qu.: 30288153
## Max.   :2022                      Max.   : 666842551
##
```

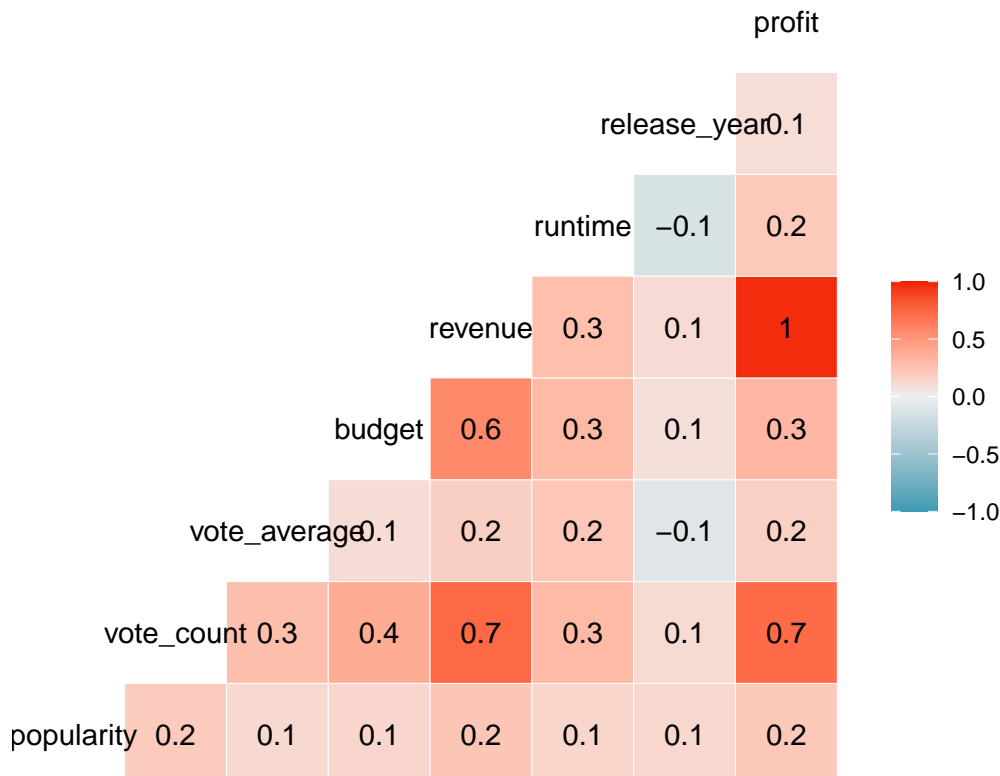
### Distribution of Target Variable

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

```
##
## No Success      Success
## 0.3216995      0.6783005
```

### Correlation Analysis

```
ggcorr(df_for_class[, apply(df_for_class, is.numeric)], label = TRUE)
```

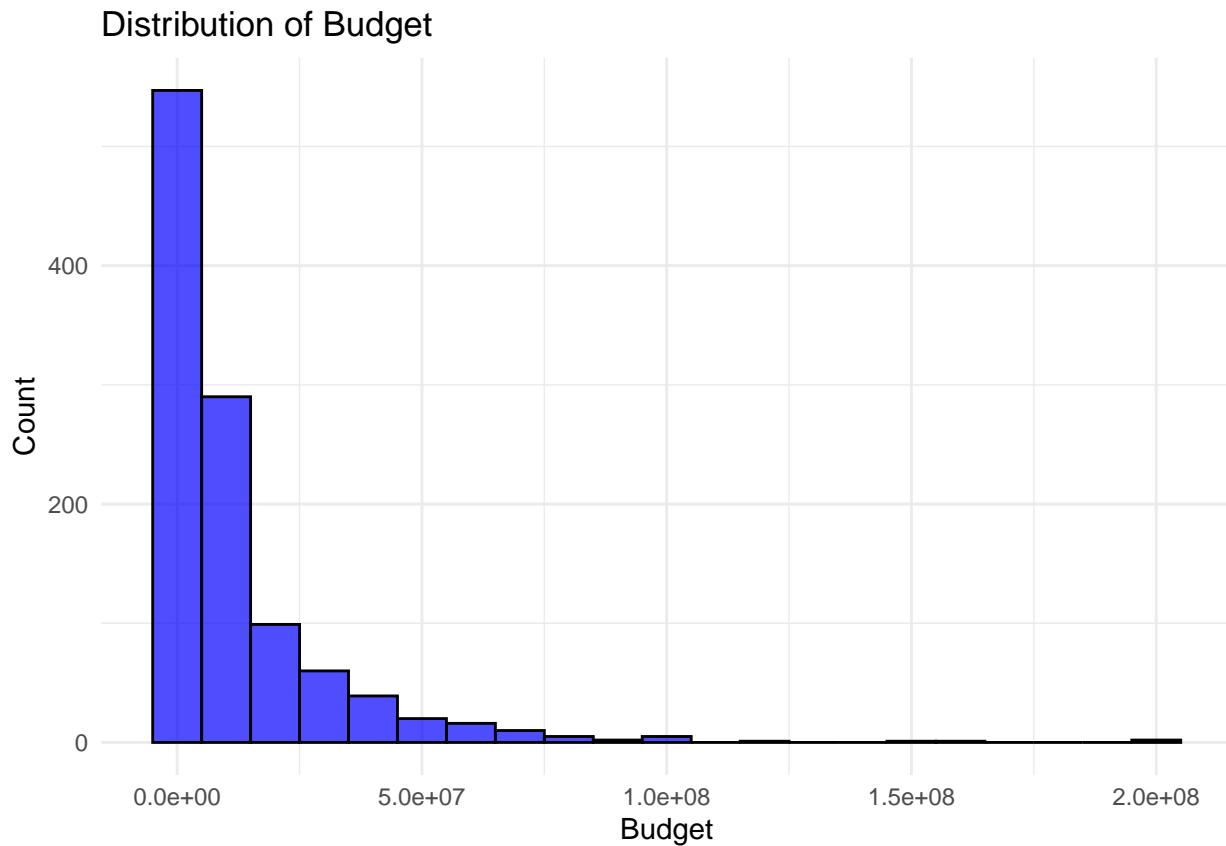


## Visualization Tools

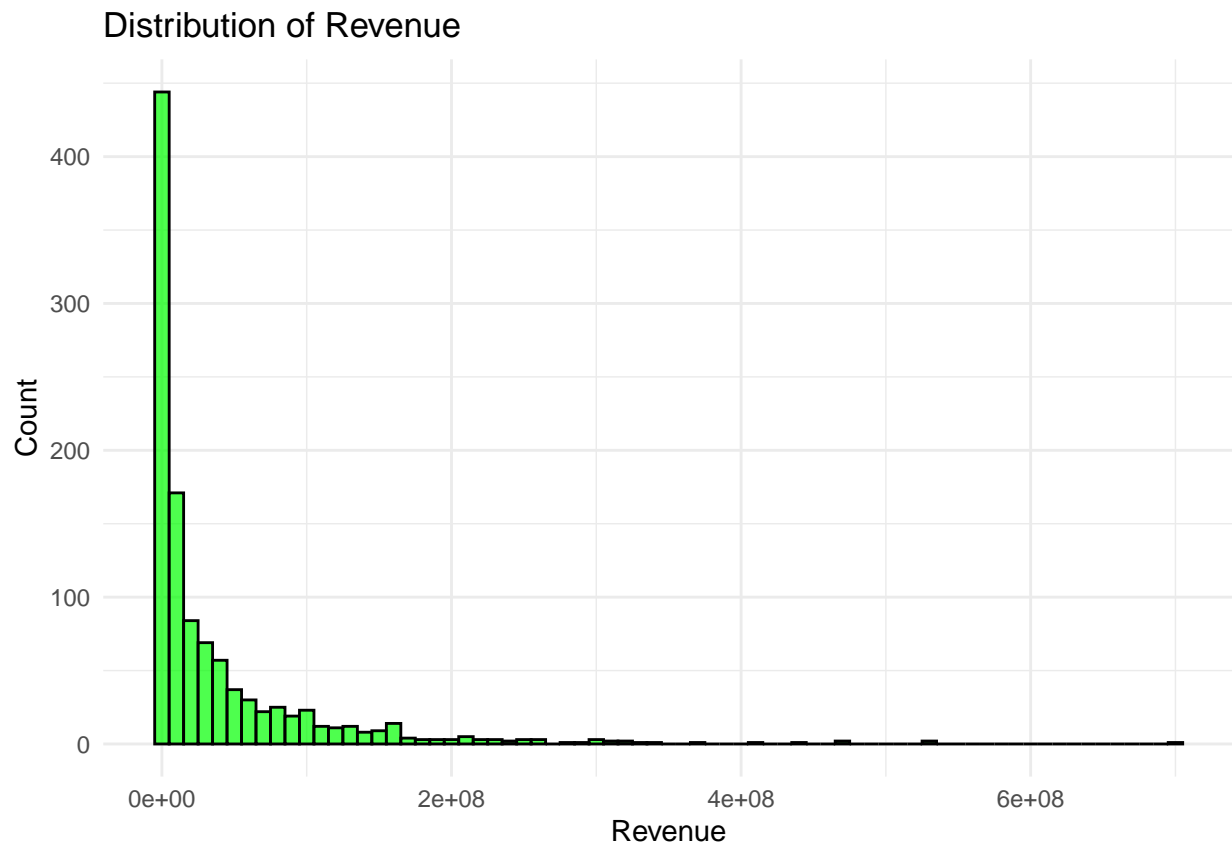
### Distribution of Numeric Features

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

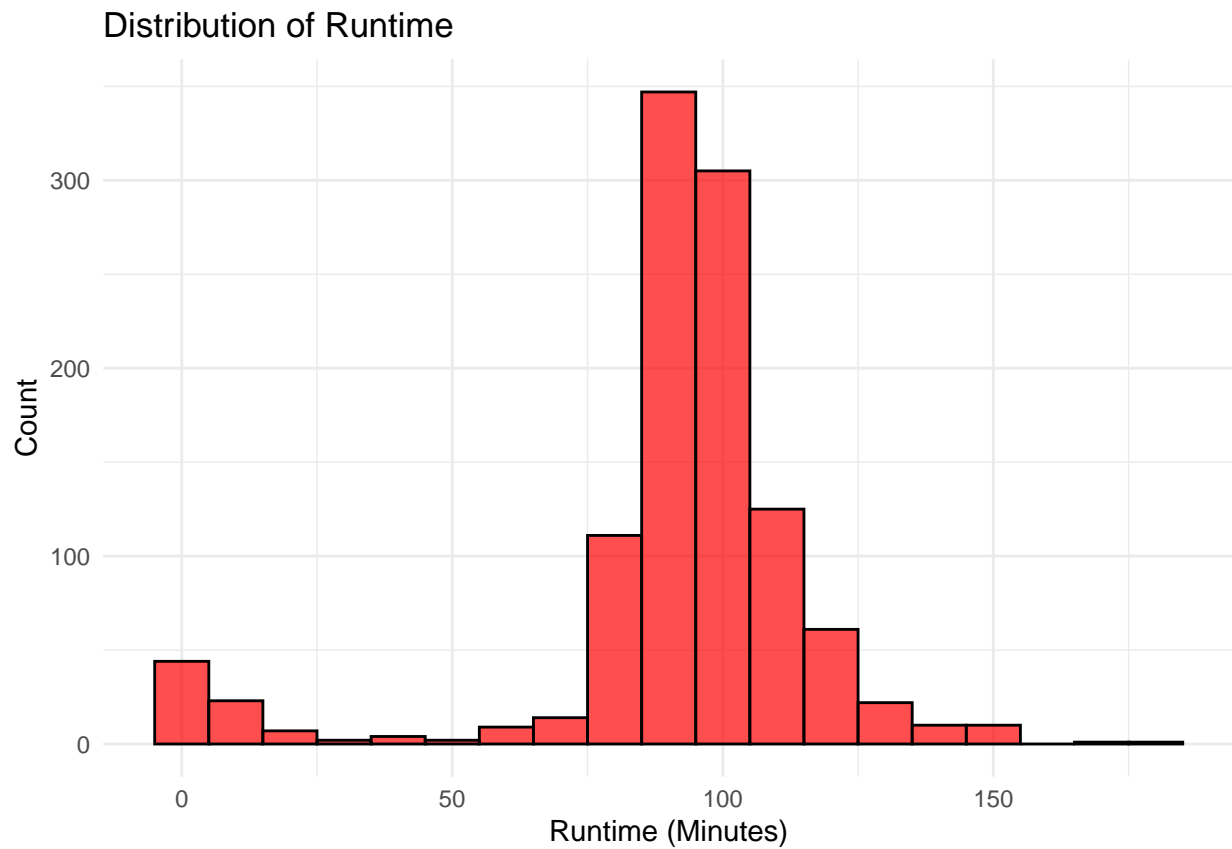
```
ggplot(df_for_class, aes(x = budget)) +  
  geom_histogram(binwidth = 1e7, fill = "blue", color = "black", alpha = 0.7) +  
  labs(title = "Distribution of Budget", x = "Budget", y = "Count") +  
  theme_minimal()
```



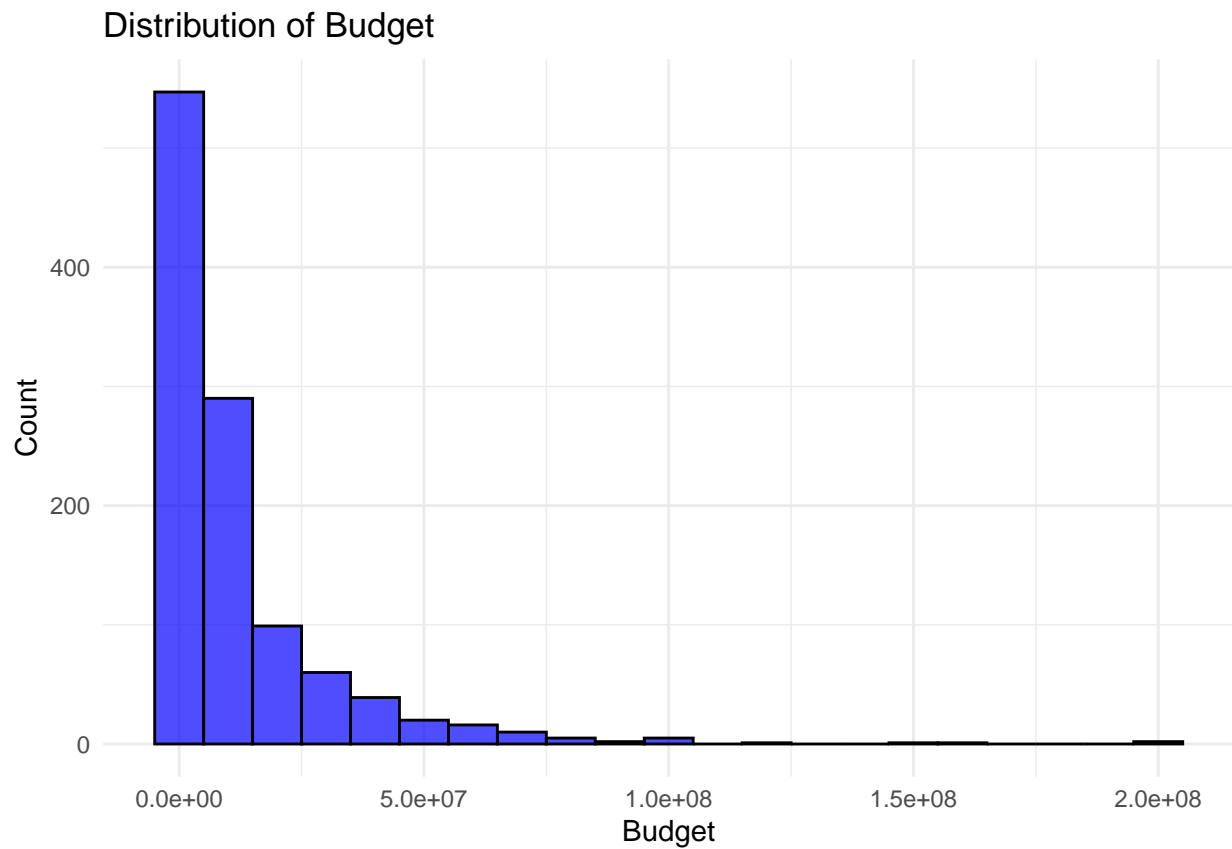
```
ggplot(df_for_class, aes(x = revenue)) +  
  geom_histogram(binwidth = 1e7, fill = "green", color = "black", alpha = 0.7) +  
  labs(title = "Distribution of Revenue", x = "Revenue", y = "Count") +  
  theme_minimal()
```



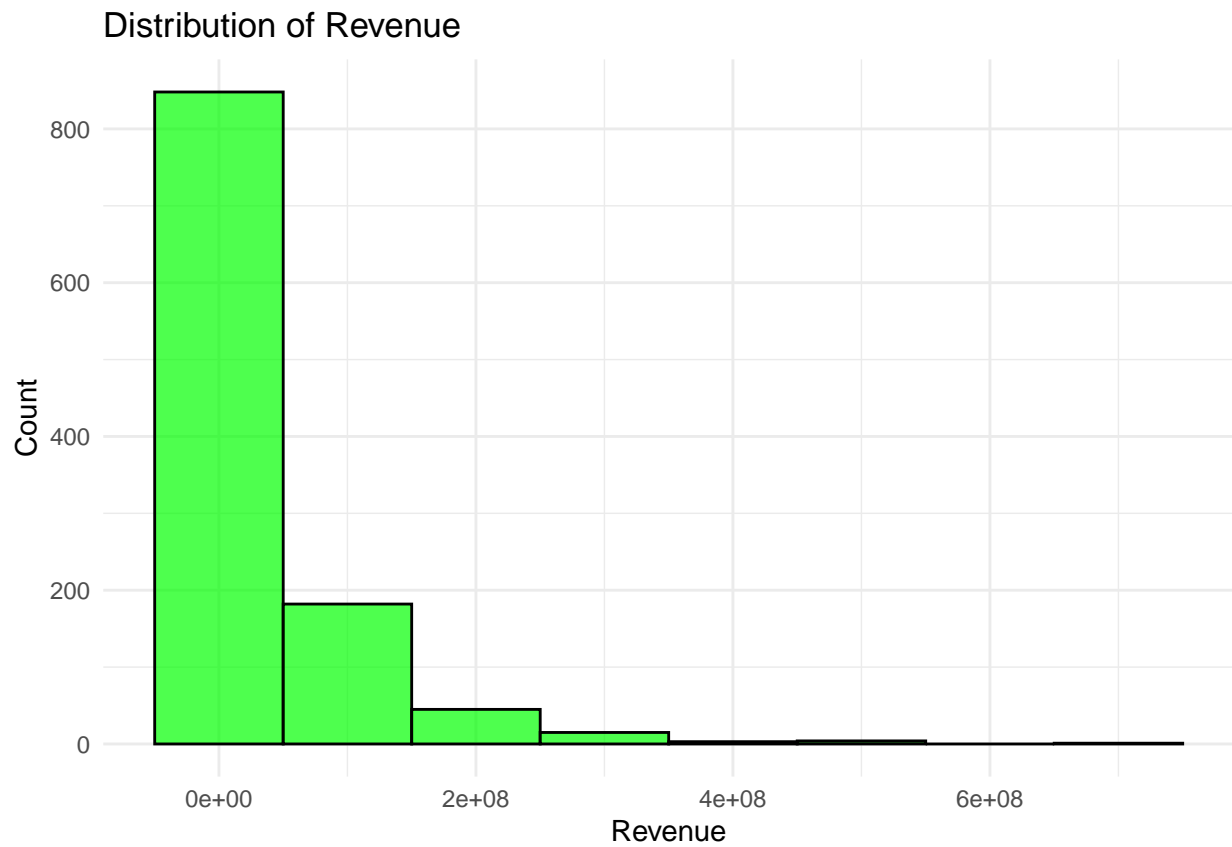
```
ggplot(df_for_class, aes(x = runtime)) +  
  geom_histogram(binwidth = 10, fill = "red", color = "black", alpha = 0.7) +  
  labs(title = "Distribution of Runtime", x = "Runtime (Minutes)", y = "Count") +  
  theme_minimal()
```



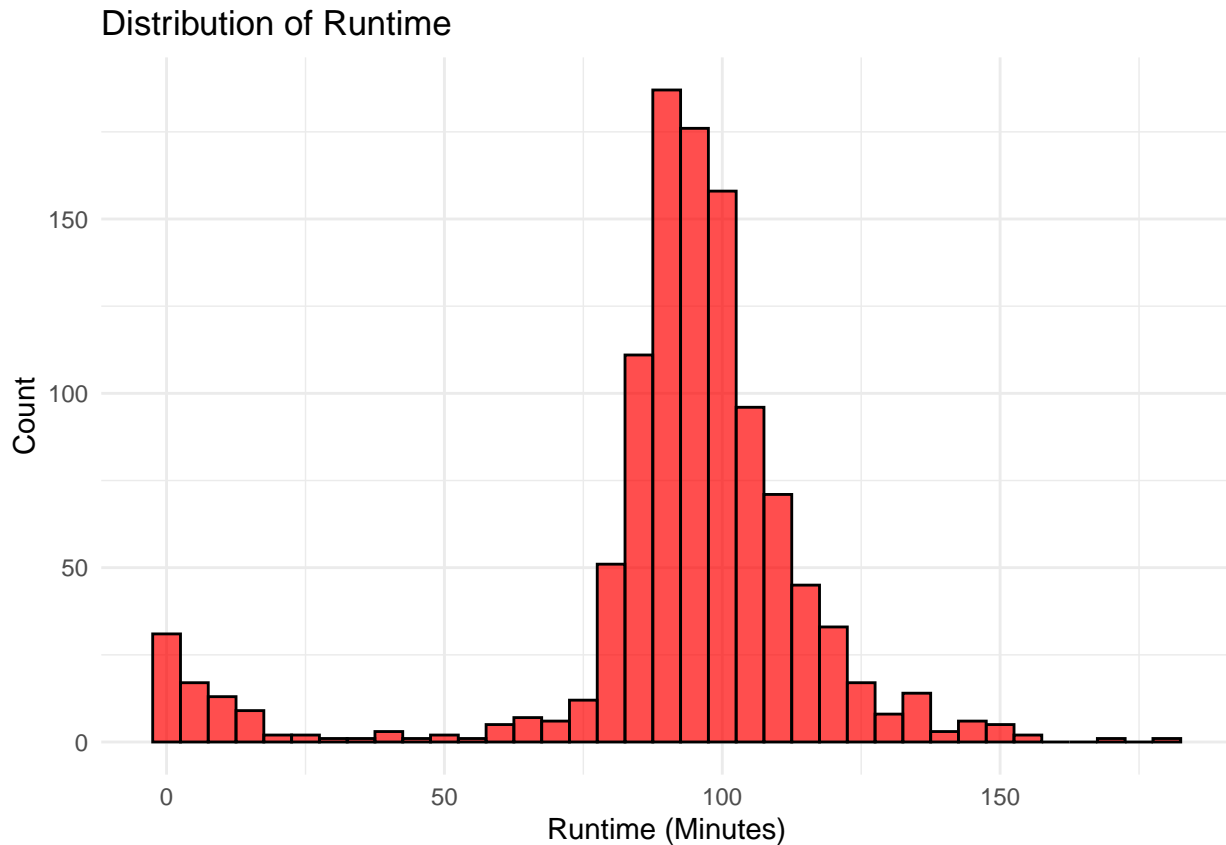
```
# For Budget (which has large values)
ggplot(df_for_class, aes(x = budget)) +
  geom_histogram(binwidth = 1e7, fill = "blue", color = "black", alpha = 0.7) + # Increase binwidth
  labs(title = "Distribution of Budget", x = "Budget", y = "Count") +
  theme_minimal()
```



```
# For Revenue (which also has large values)
ggplot(df_for_class, 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()
```



```
# For Runtime (which typically has smaller values)  
ggplot(df_for_class, 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.

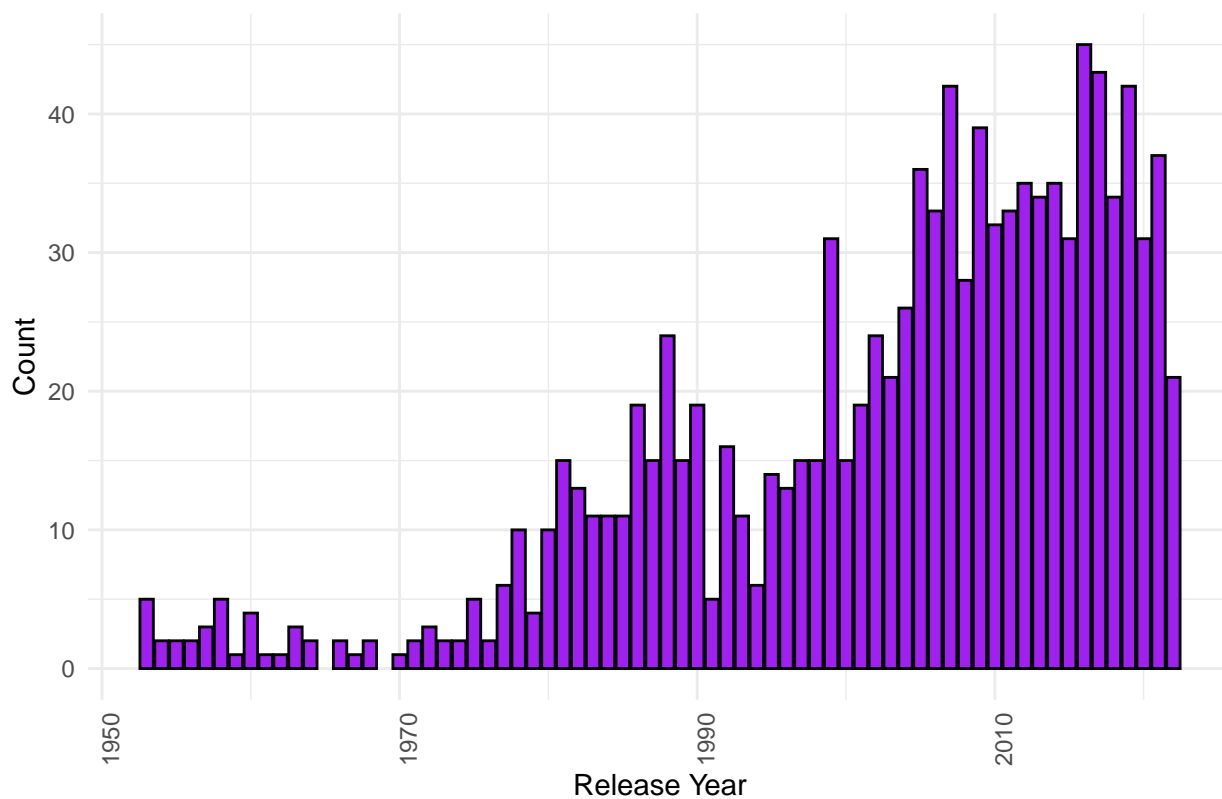
### Distribution of Some Categorical Features

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

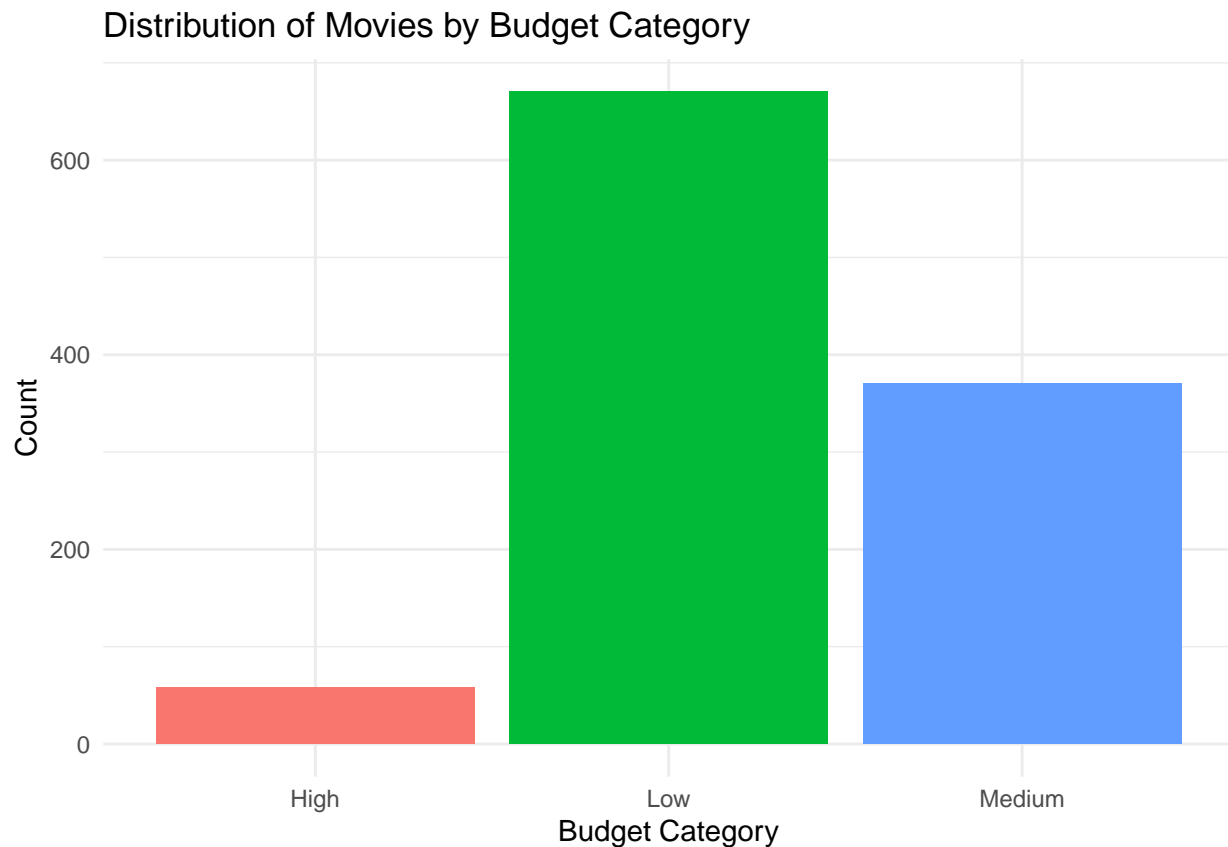
```
ggplot(df_for_class, aes(x = release_year)) +  
  geom_bar(fill = "purple", color = "black") +  
  labs(title = "Distribution of Movies by Release Year", x = "Release Year", y = "Count") +  
  theme_minimal() +  
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



Distribution of Movies by Release Year



```
ggplot(df_for_class, 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`.

```
sum(df_for_class$success == "Success")
```

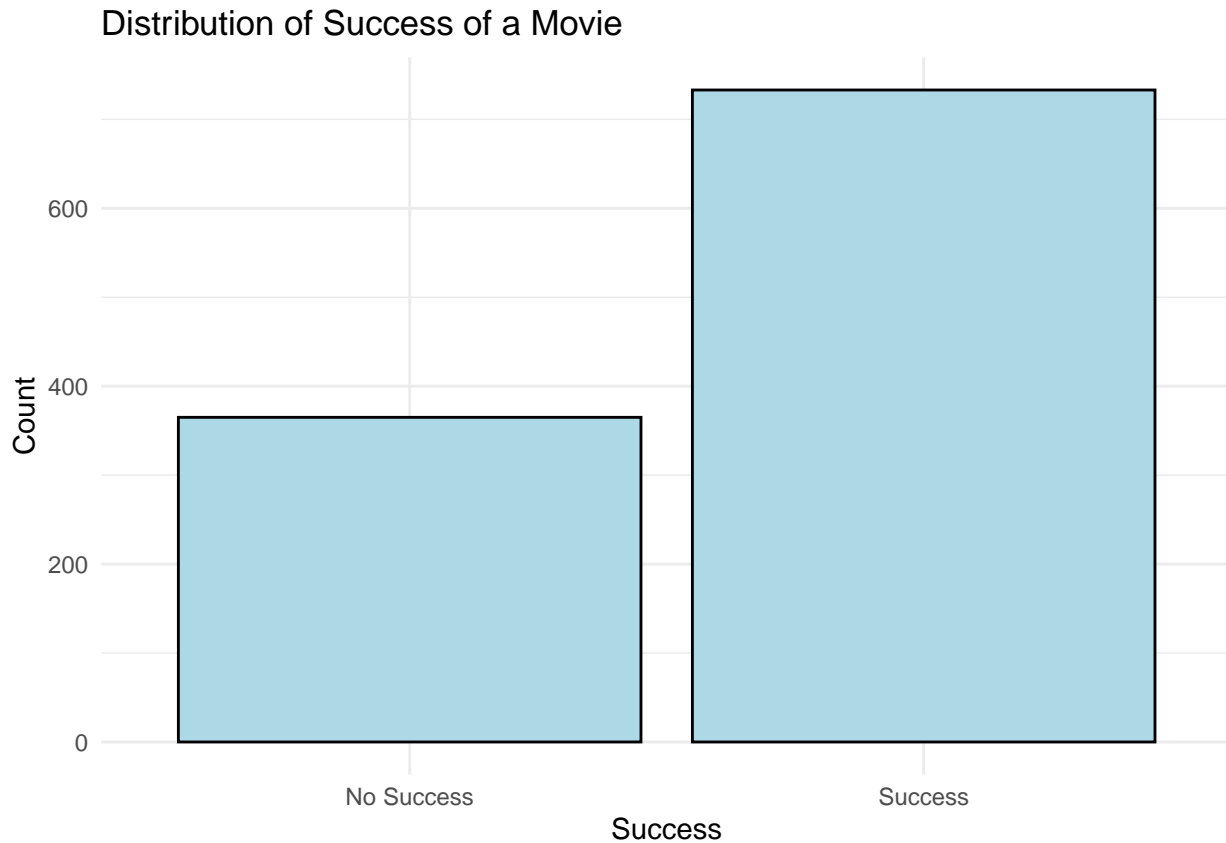
```
## [1] 733
```

```
sum(df_for_class$success == "No Success")
```

```
## [1] 365
```

66.7% successful movies, 33.3% unsuccessful movies: unbalanced dataset

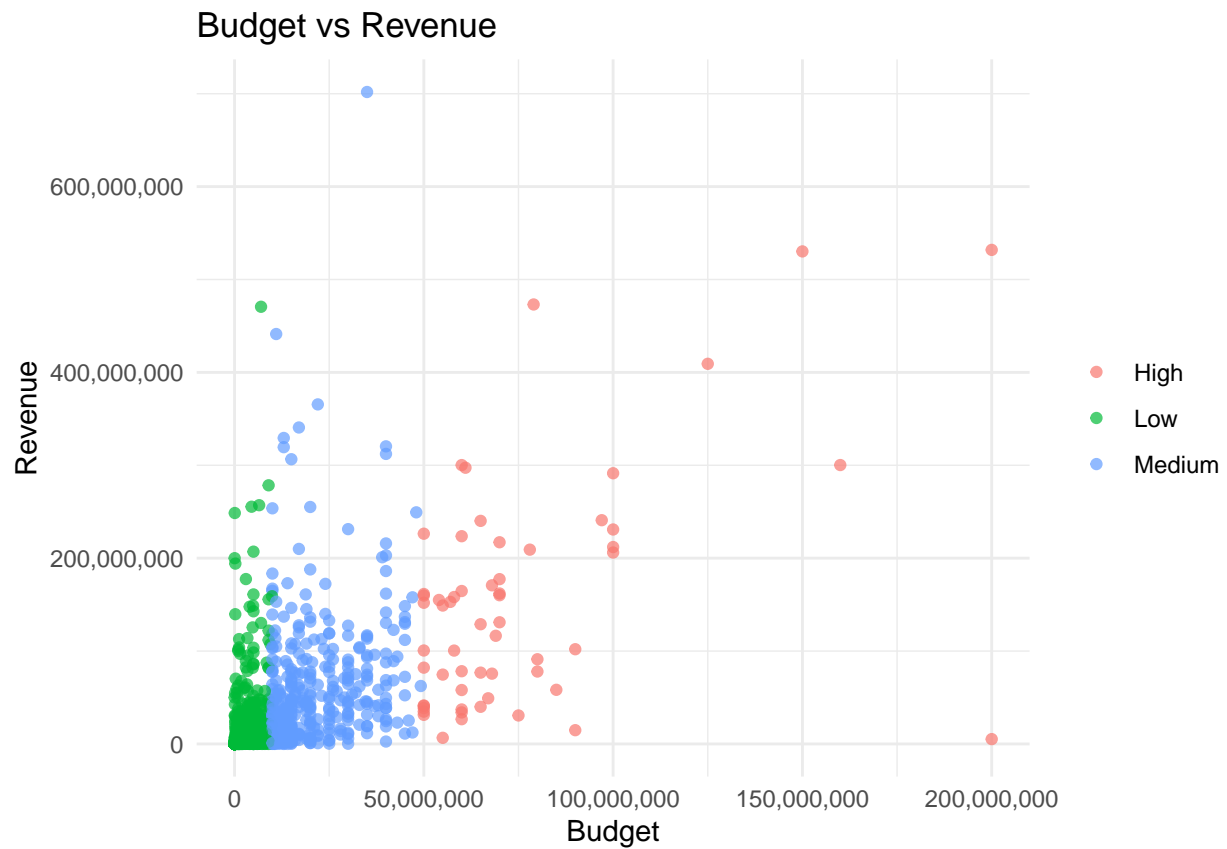
```
ggplot(df_for_class, aes(x = success)) +  
  geom_bar(fill = "lightblue", color = "black") +  
  labs(title = "Distribution of Success of a Movie", x = "Success", y = "Count") +  
  theme_minimal()
```



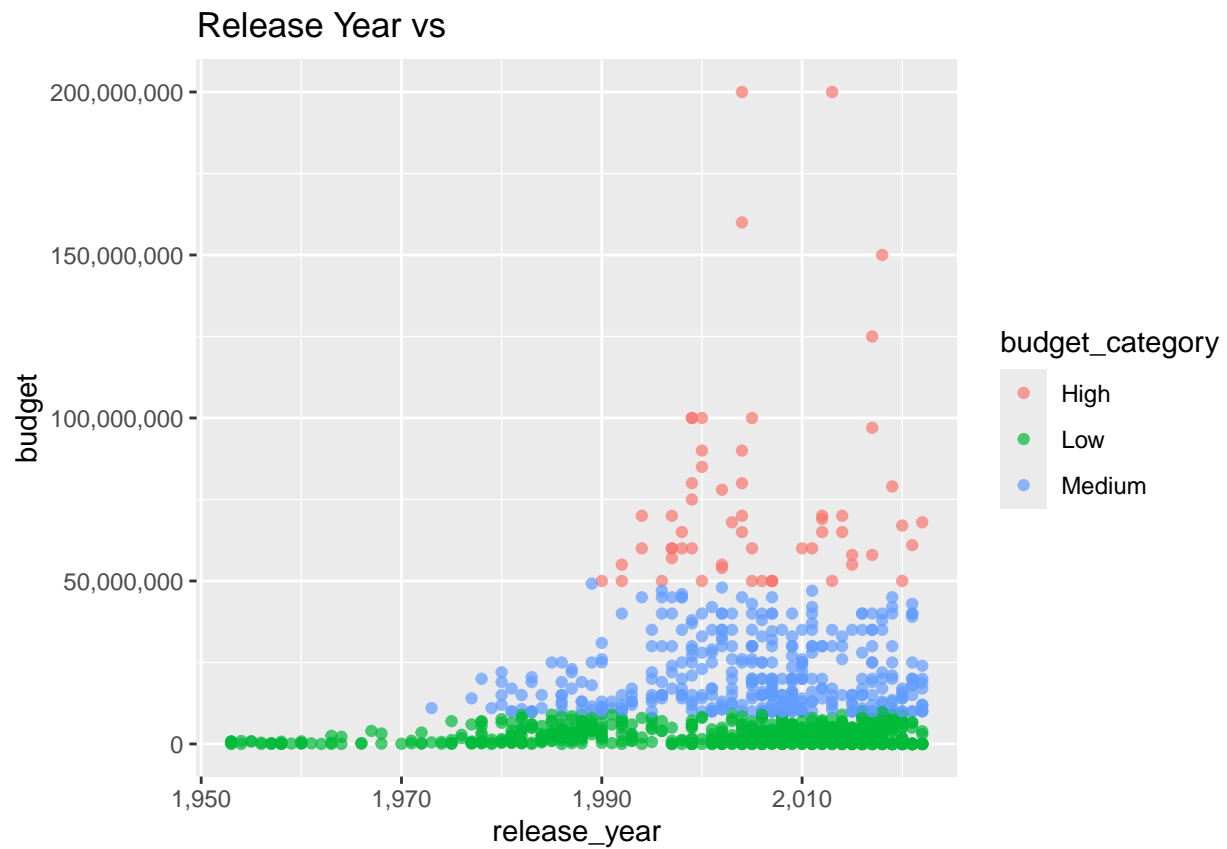
### Variable Relationships

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.

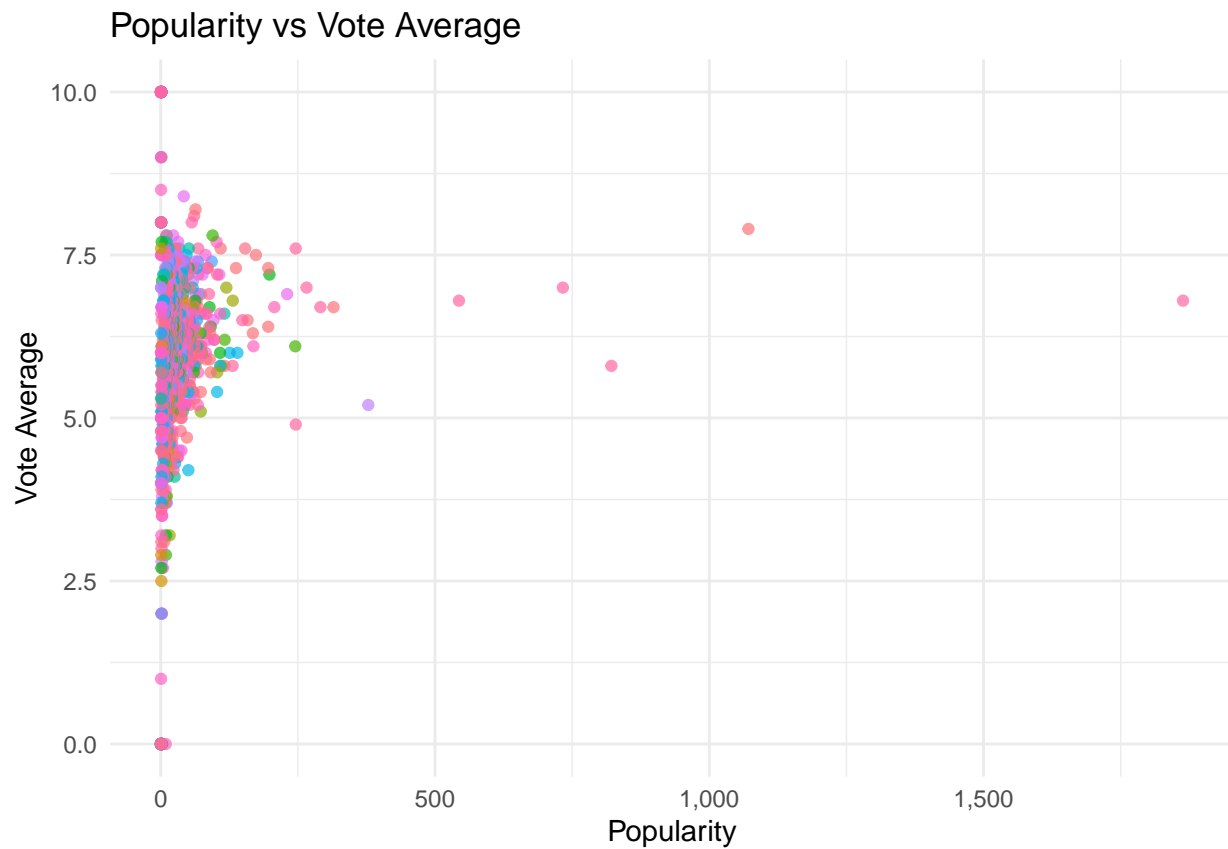
```
ggplot(df_for_class, aes(x = budget, y = revenue)) +
  geom_point(aes(color = budget_category), alpha = 0.7) +
  scale_x_continuous(labels = scales::comma) +
  scale_y_continuous(labels = scales::comma) +
  labs(title = "Budget vs Revenue", x = "Budget", y = "Revenue") +
  theme_minimal() +
  theme(legend.title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



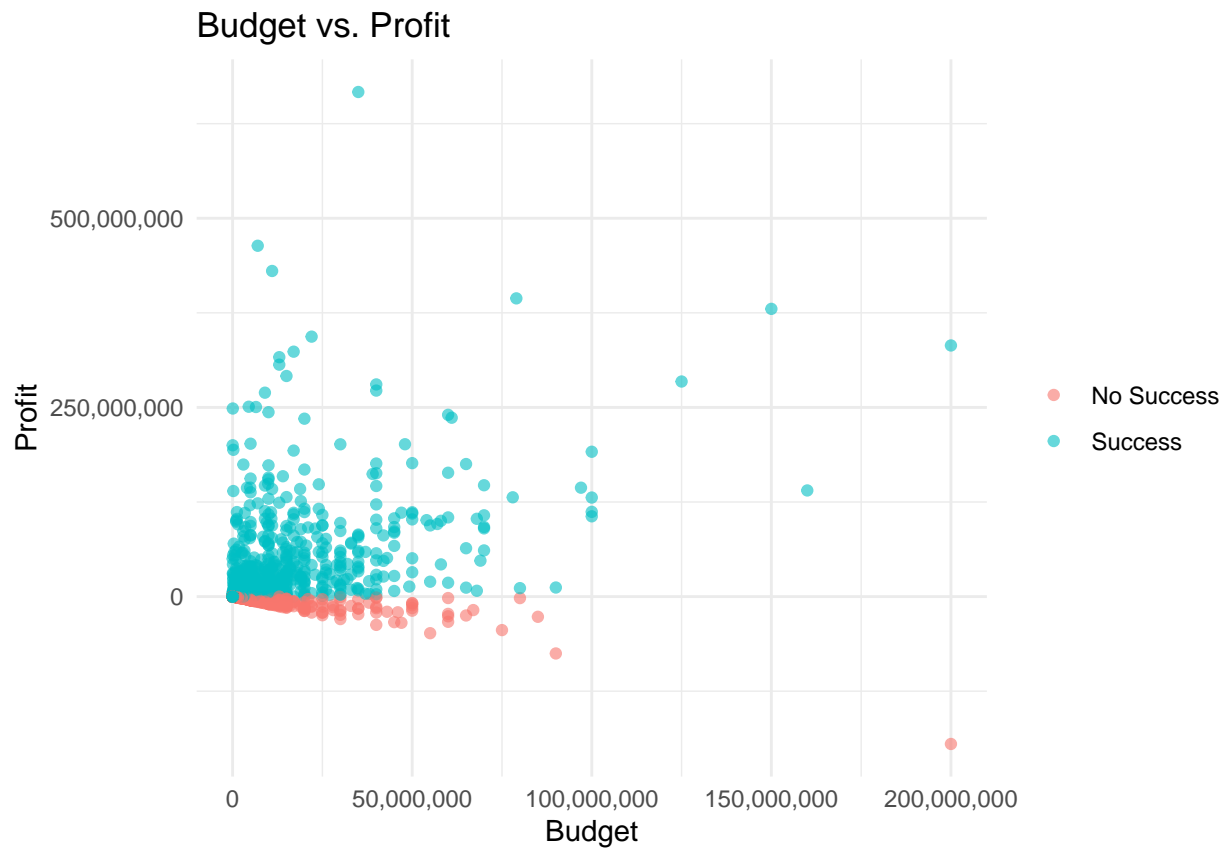
```
ggplot(df_for_class, aes(x = release_year, y = budget)) +  
  geom_point(aes(color = budget_category), alpha = 0.7) +  
  scale_x_continuous(labels = scales::comma) +  
  scale_y_continuous(labels = scales::comma) +  
  labs(title = "Release Year vs")
```



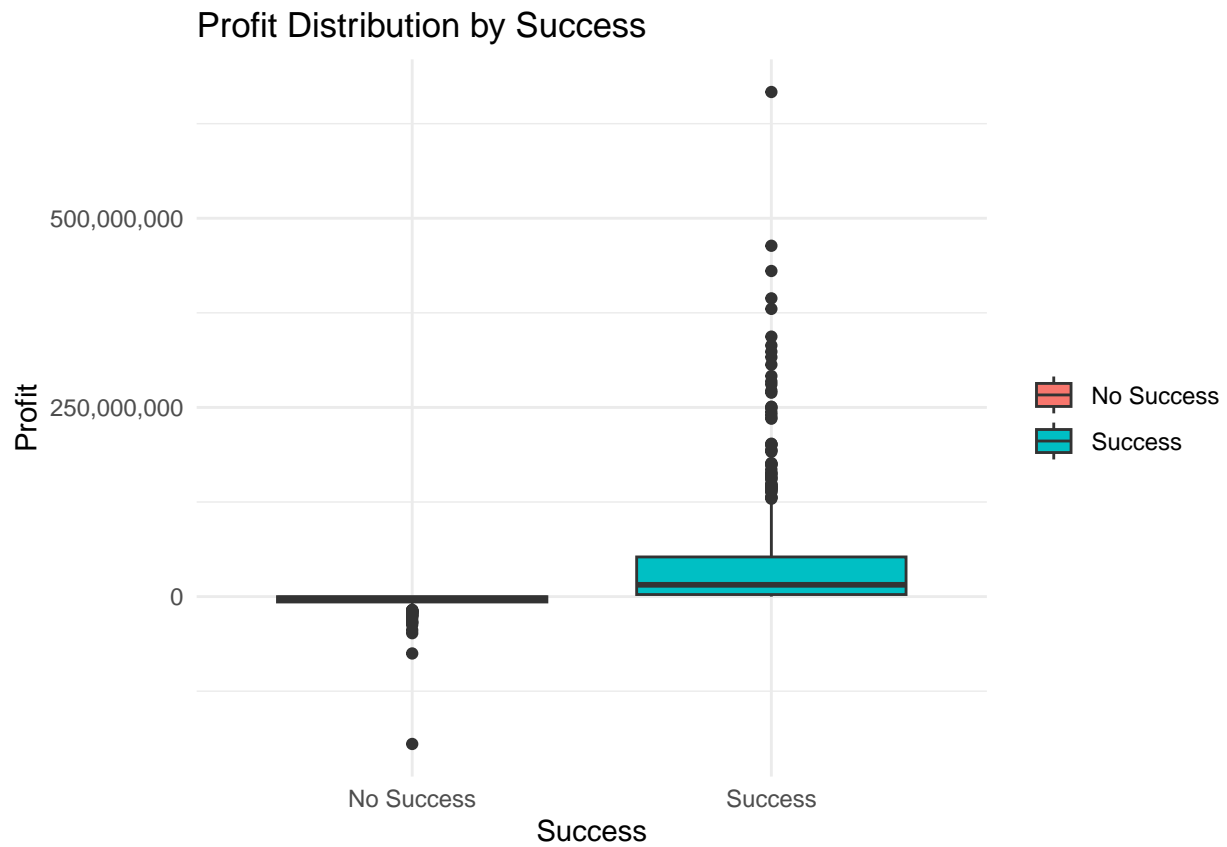
```
ggplot(df_for_class, aes(x = popularity, y = vote_average)) +
  geom_point(aes(color = genre_names), alpha = 0.7) +
  scale_x_continuous(labels = scales::comma) +
  labs(title = "Popularity vs Vote Average", x = "Popularity", y = "Vote Average") +
  theme_minimal() +
  theme(legend.position = "none")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



```
ggplot(df_for_class, 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())
```



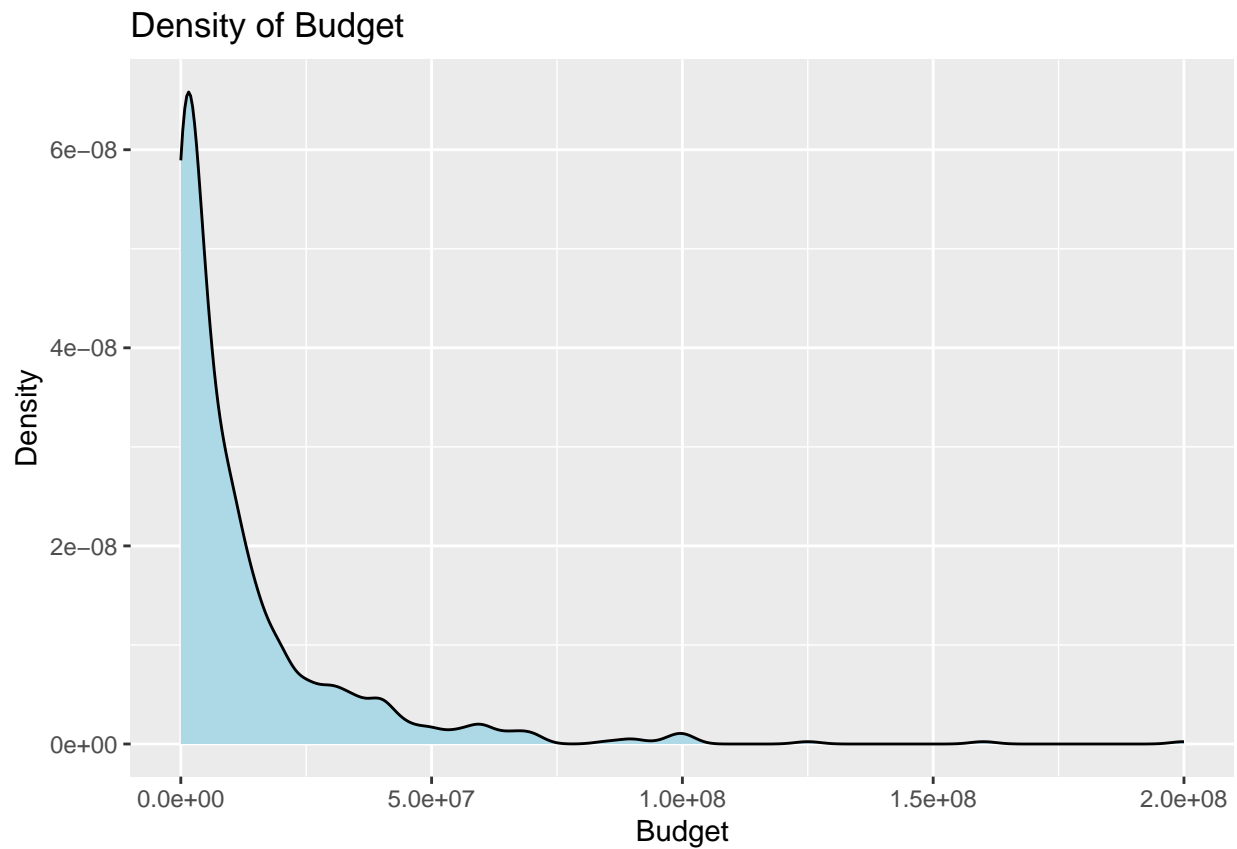
```
ggplot(df_for_class, aes(x = success, y = profit, fill = success)) +  
  geom_boxplot() +  
  scale_y_continuous(labels = scales::comma) +  
  labs(title = "Profit Distribution by Success", x = "Success", y = "Profit") +  
  theme_minimal() +  
  theme(legend.title = element_blank())
```



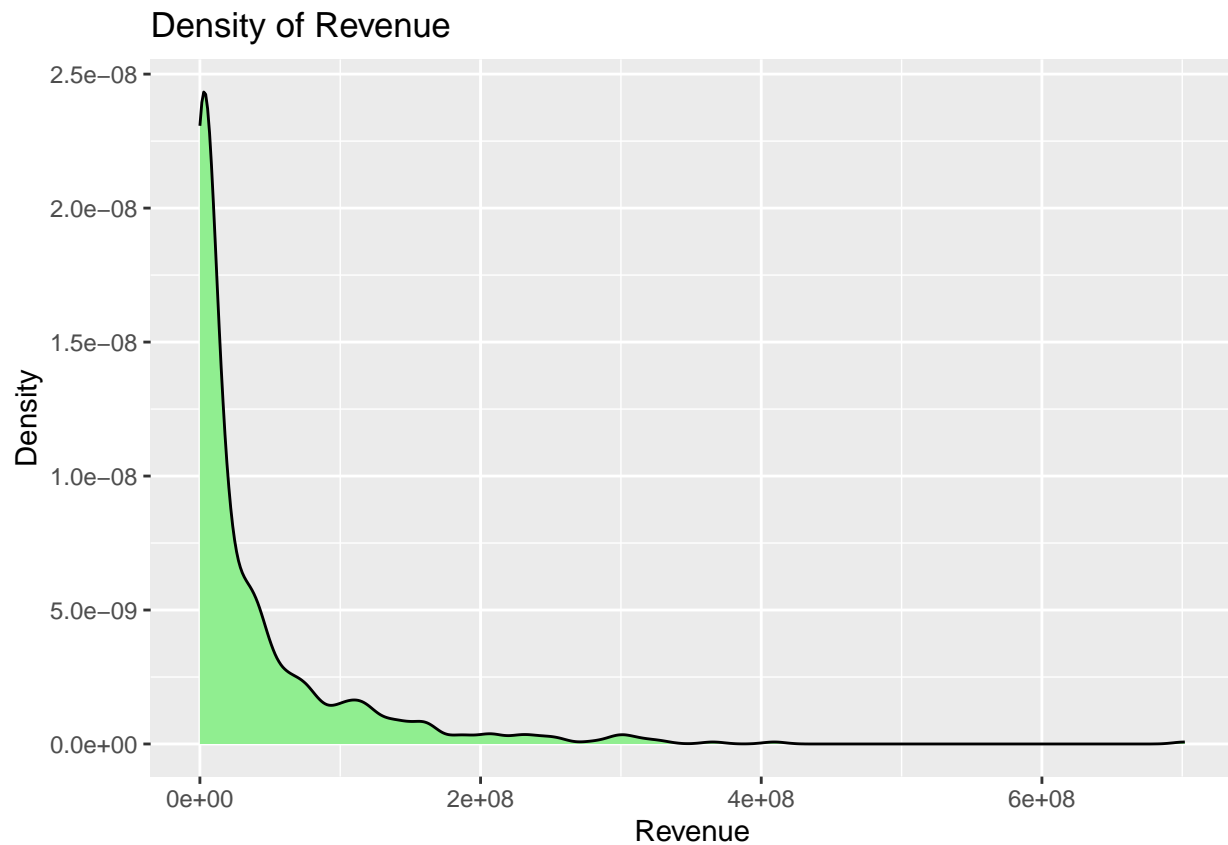
#### Training Visuals

```
# 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")
```

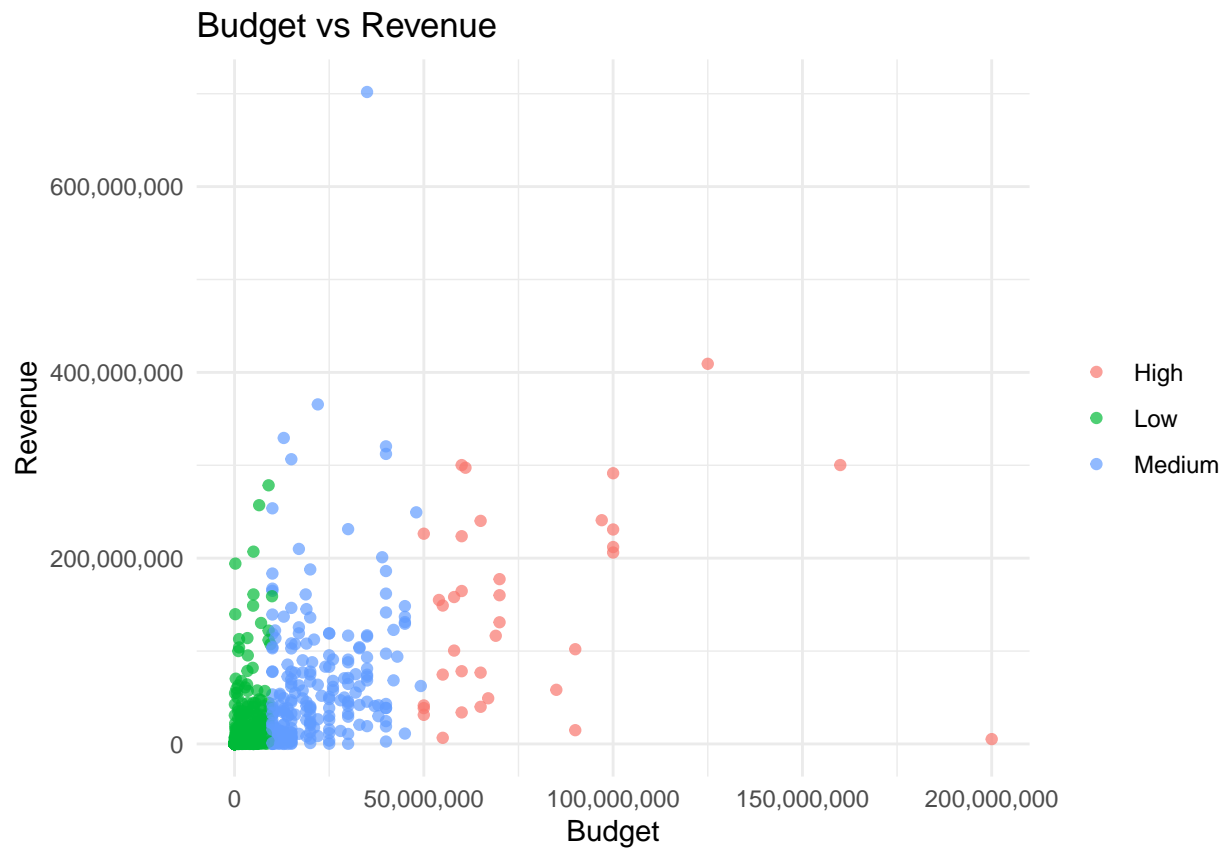




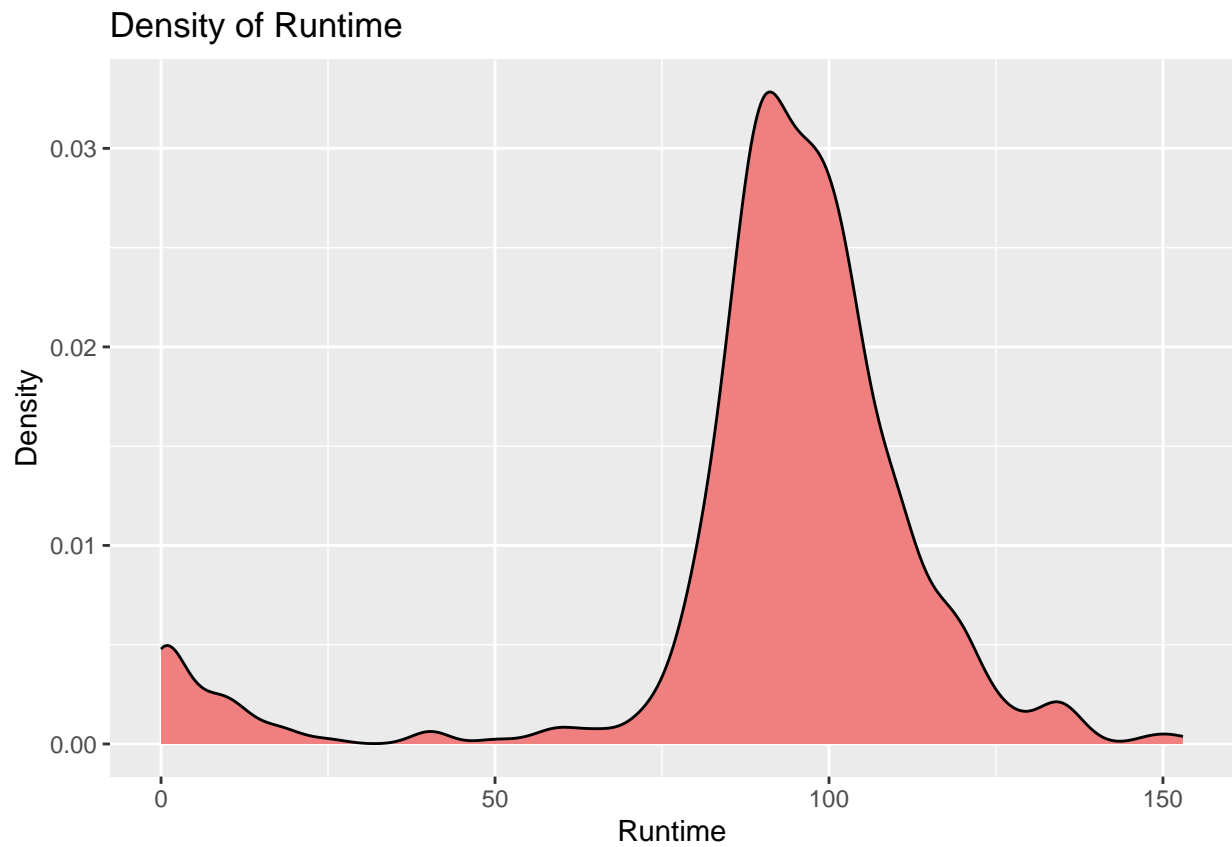
```
# Visualizing the distribution of 'revenue' in training data  
ggplot(training, aes(x = revenue)) +  
  geom_density(fill = "lightgreen") +  
  labs(title = "Density of Revenue", x = "Revenue", y = "Density")
```



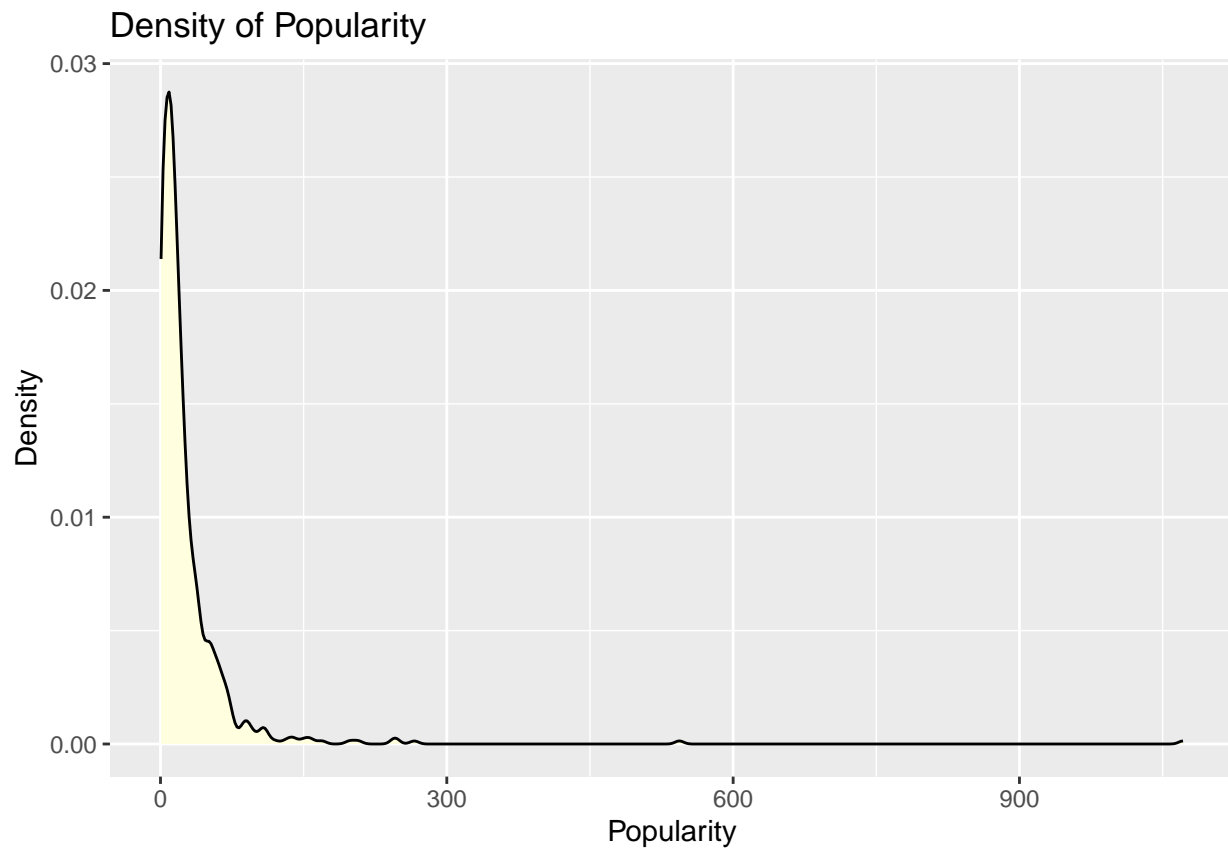
```
# Visualizing 'budget' against 'revenue'  
ggplot(training, aes(x = budget, y = revenue)) +  
  geom_point(aes(color = budget_category), alpha = 0.7) +  
  scale_x_continuous(labels = scales::comma) +  
  scale_y_continuous(labels = scales::comma) +  
  labs(title = "Budget vs Revenue", x = "Budget",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



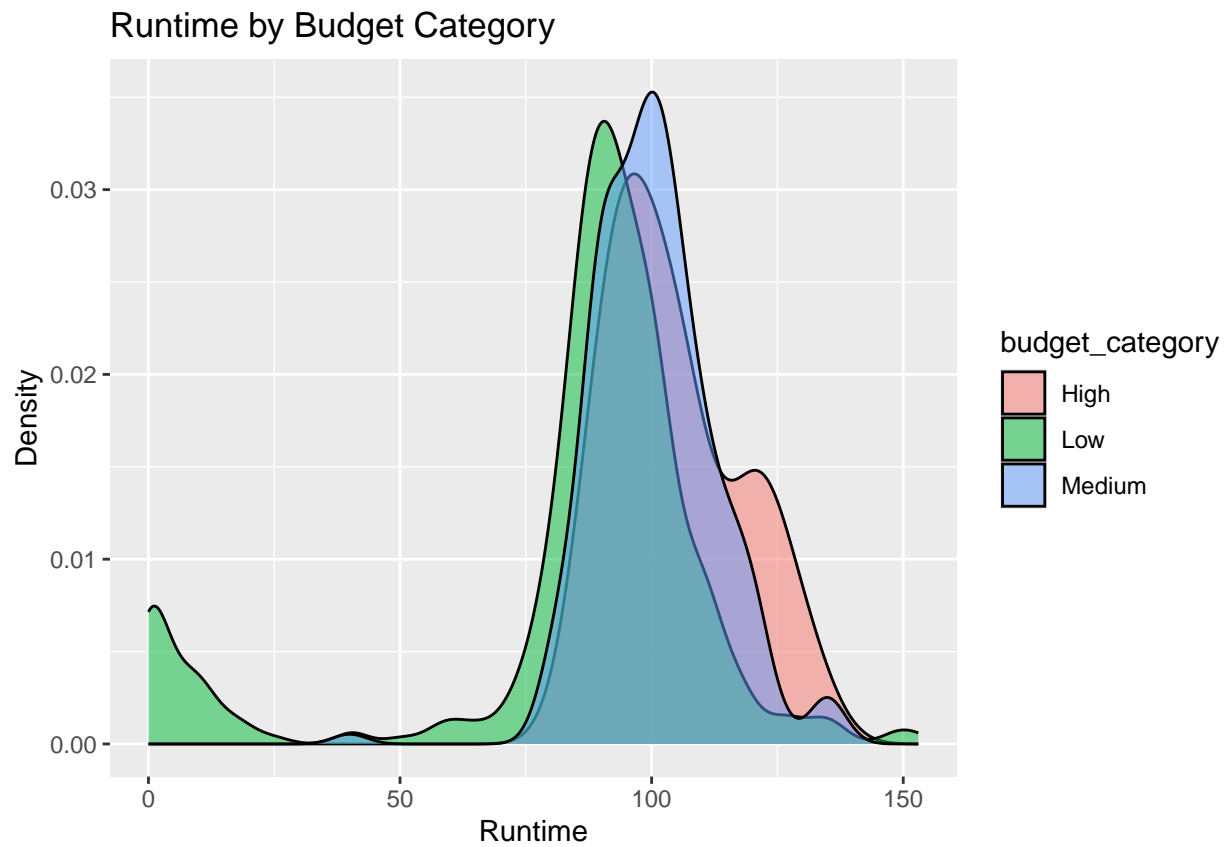
```
# Exploring 'runtime' distribution  
ggplot(training, aes(x = runtime)) +  
  geom_density(fill = "lightcoral") +  
  labs(title = "Density of Runtime", x = "Runtime", y = "Density")
```



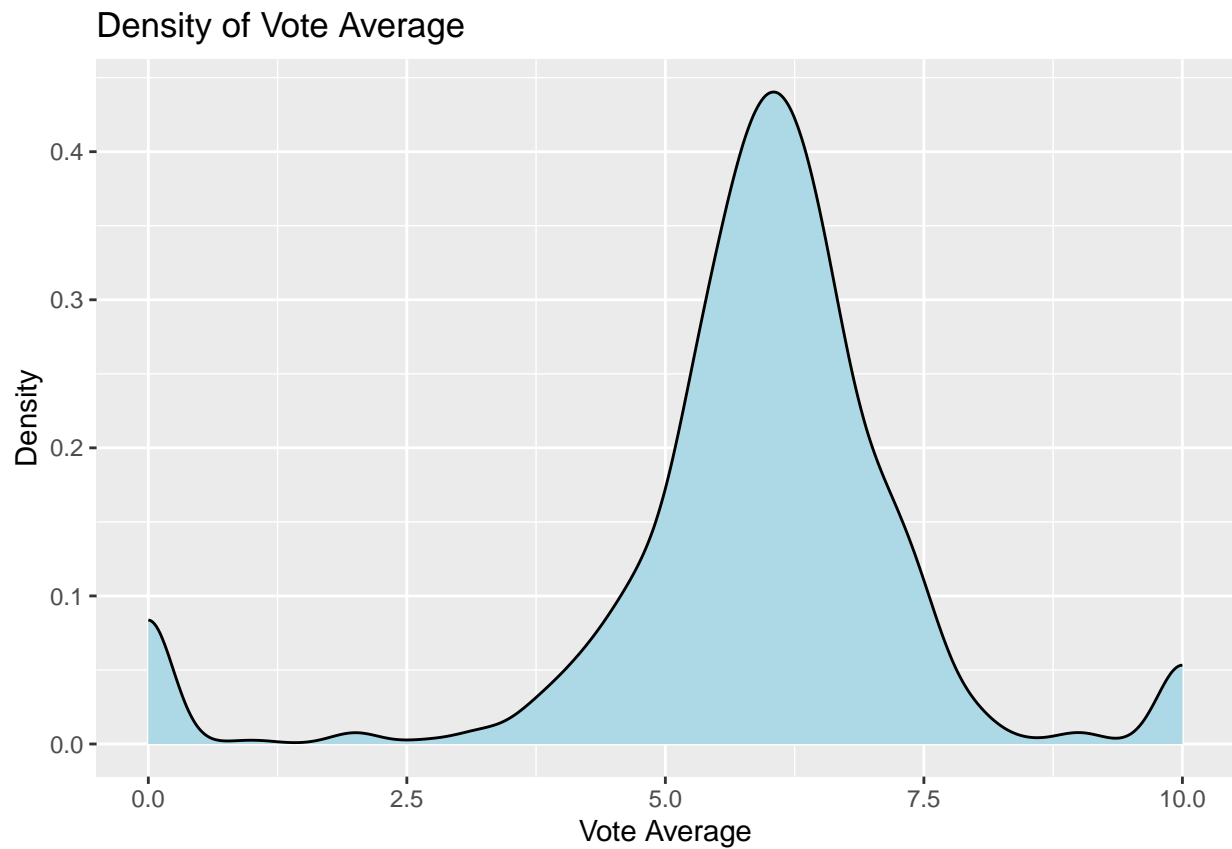
```
# Exploring 'popularity' distribution  
ggplot(training, aes(x = popularity)) +  
  geom_density(fill = "lightyellow") +  
  labs(title = "Density of Popularity", x = "Popularity", y = "Density")
```



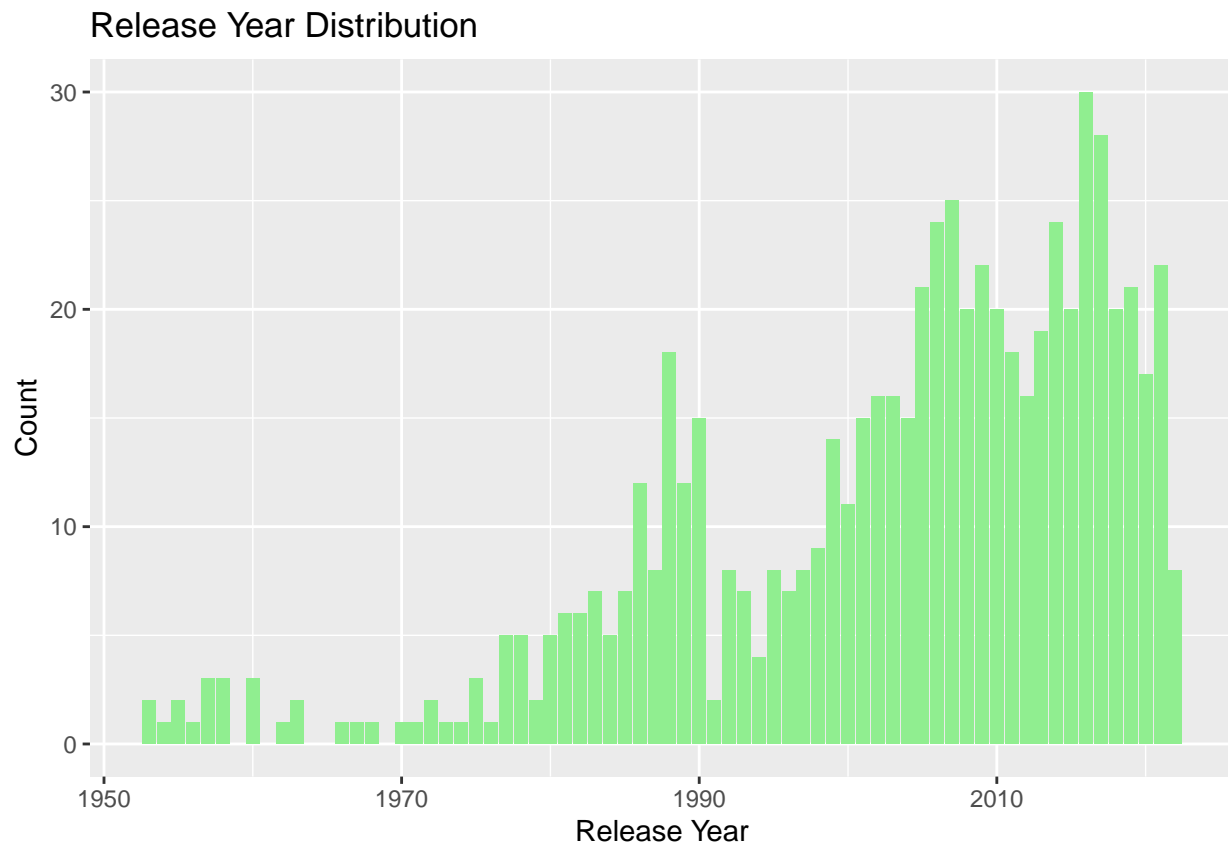
```
# Decomposing runtime by 'budget_category'  
ggplot(training, aes(x = runtime, fill = budget_category)) +  
  geom_density(alpha = 0.5) +  
  labs(title = "Runtime by Budget Category", x = "Runtime", y = "Density")
```



```
# 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")
```

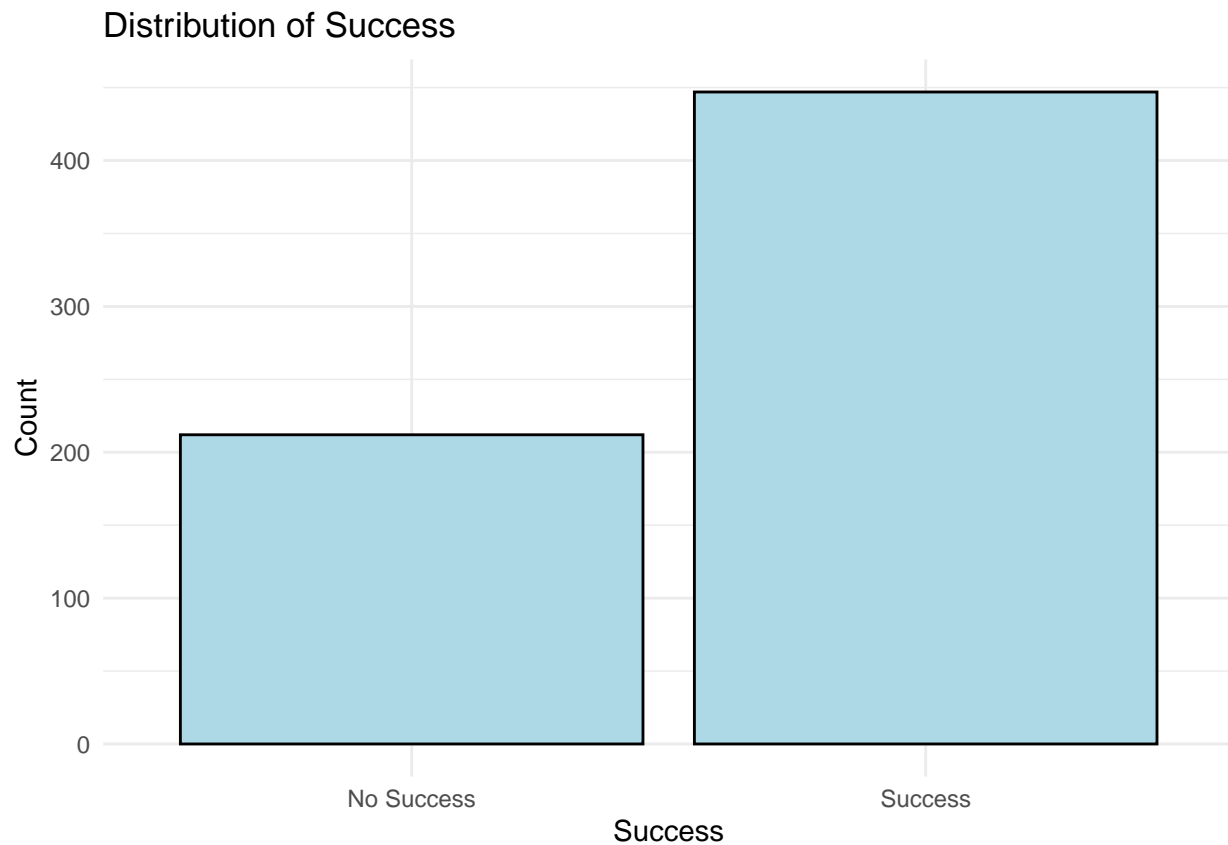


```
# Visualizing 'release_year' distribution  
ggplot(training, aes(x = release_year)) +  
  geom_bar(fill = "lightgreen") +  
  labs(title = "Release Year Distribution", x = "Release Year", y = "Count")
```



```
# 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
6. Predicting whether a movie was a success or not is our main objective.

## Normalizing the data

```
# Columns to normalize
columns_to_normalize <- c("budget", "release_year", "runtime", "popularity")

# Min-max normalization function
min_max_normalize <- function(x) {
  (x - min(x, na.rm = TRUE)) / (max(x, na.rm = TRUE) - min(x, na.rm = TRUE))
}

# Create a copy of the original data frame
df_normal <- df_for_class
```

```

# Normalize only the specified columns
df_normal[columns_to_normalize] <- lapply(df_for_class[columns_to_normalize], min_max_normalize)

# Check the structure to confirm normalization
str(df_normal)

## tibble [1,098 x 19] (S3: tbl_df/tbl/data.frame)
## $ original_title   : chr [1:1098] "Smile" "The Black Phone" "Jeepers Creepers: Reborn" "Nope" ...
## $ title            : chr [1:1098] "Smile" "The Black Phone" "Jeepers Creepers: Reborn" "Nope" ...
## $ original_language: Factor w/ 97 levels "af","am","ar",...: 19 19 19 19 19 19 19 19 19 19 ...
## $ overview         : chr [1:1098] "After witnessing a bizarre, traumatic incident involving a patient" ...
## $ tagline          : chr [1:1098] "Once you see it, it's too late." "Never talk to strangers." "Evil" ...
## $ release_date      : Date[1:1098], format: "2022-09-23" "2022-06-22" ...
## $ popularity        : num [1:1098] 1 0.575 0.441 0.393 0.291 ...
## $ vote_count        : num [1:1098] 114 2736 125 1684 1035 ...
## $ vote_average      : num [1:1098] 6.8 7.9 5.8 7 6.8 5.2 6.7 6.7 7 4.9 ...
## $ budget            : num [1:1098] 0.085 0.094 0.1 0.34 0.05 ...
## $ revenue           : num [1:1098] 4.50e+07 1.61e+08 2.89e+06 1.71e+08 1.43e+07 ...
## $ runtime           : num [1:1098] 0.642 0.575 0.492 0.726 0.592 ...
## $ status            : Factor w/ 4 levels "In Production",...: 4 4 4 4 4 4 4 4 4 4 ...
## $ adult             : logi [1:1098] FALSE FALSE FALSE FALSE FALSE FALSE ...
## $ genre_names        : Factor w/ 772 levels "Action, Adventure, Animation, Comedy, Drama, Fantasy, Horror, ...
## $ release_year       : num [1:1098] 1 1 1 1 1 ...
## $ budget_category    : chr [1:1098] "Medium" "Medium" "Medium" "High" ...
## $ profit             : num [1:1098] 28000000 142200000 -17107406 102800000 4257609 ...
## $ success            : Factor w/ 2 levels "No Success","Success": 2 2 1 2 2 1 2 2 2 1 ...

```

## Splitting training and testing for normalized data

```

in_train_n <- createDataPartition(df_normal$budget_category, p = 0.6, list = FALSE)
training_n <- df_normal[in_train, ]
testing_n <- df_normal[-in_train, ]

```

```
nrow(training_n)
```

```
## [1] 659
```

```
nrow(testing_n)
```

```
## [1] 439
```

```
summary(training_n)
```

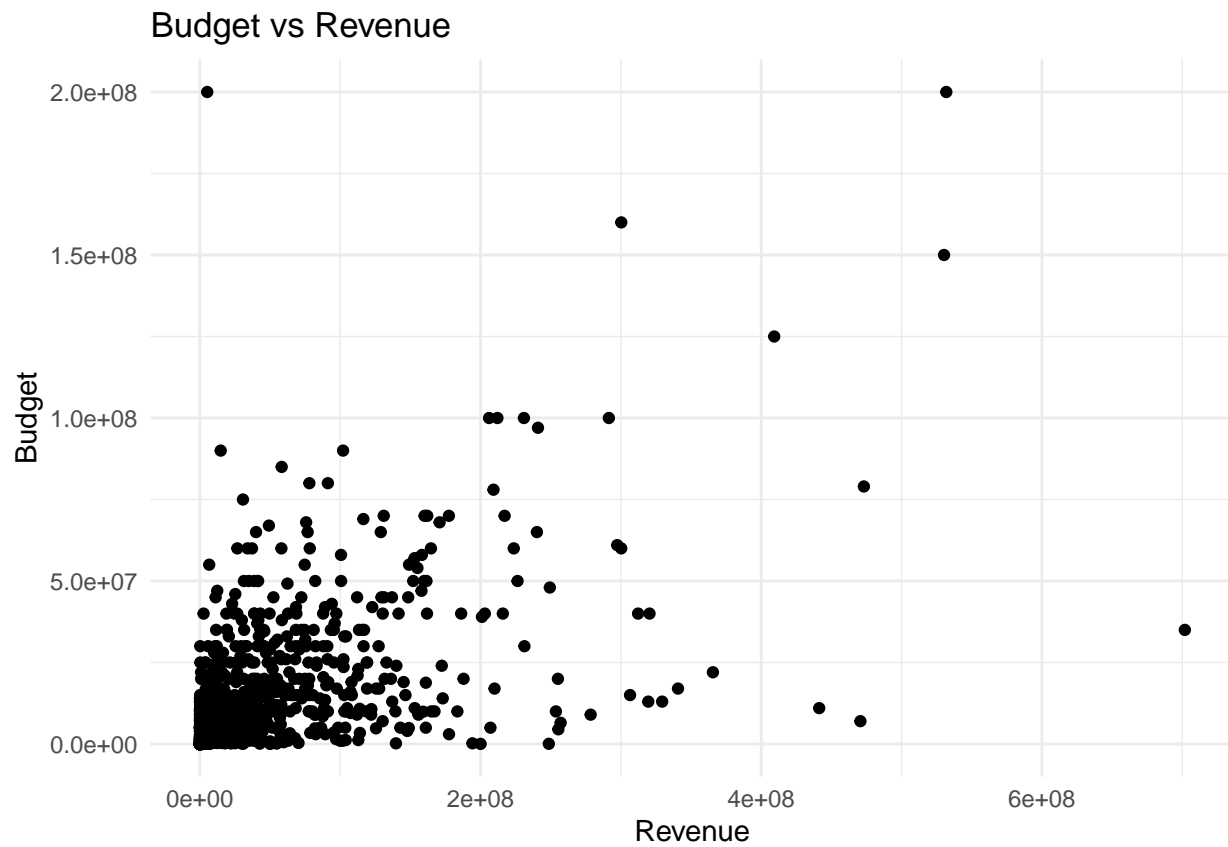
```
## original_title      title      original_language  overview
## Length:659          Length:659      en      :560      Length:659
## Class :character     Class :character  ja      : 16      Class :character
## Mode  :character     Mode  :character  es      : 14      Mode  :character
##                      hi      : 14
##                      ko      : 10
##                      de      :  8
##                      (Other): 37
## tagline              release_date      popularity      vote_count
## Length:659          Min.      :1953-06-05      Min.      :0.000000      Min.      :  0.0
## Class :character     1st Qu.:1994-12-12      1st Qu.:0.003535      1st Qu.:  98.5
## Mode  :character     Median :2007-01-19      Median :0.008006      Median : 543.0
```

```
##          Mean    :2003-08-09    Mean    :0.014176    Mean    : 1206.6
##          3rd Qu.:2015-01-28    3rd Qu.:0.016334    3rd Qu.: 1544.5
##          Max.    :2022-09-29    Max.    :0.574762    Max.    :16900.0
##
##   vote_average    budget    revenue    runtime
##   Min.    : 0.000    Min.    :0.00000    Min.    :      1    Min.    :0.0000
##   1st Qu.: 5.300    1st Qu.:0.00500    1st Qu.: 675326    1st Qu.:0.4888
##   Median : 6.000    Median :0.02500    Median : 11642254    Median :0.5307
##   Mean    : 5.797    Mean    :0.06385    Mean    : 39044327    Mean    :0.5089
##   3rd Qu.: 6.600    3rd Qu.:0.07500    3rd Qu.: 45023606    3rd Qu.:0.5754
##   Max.    :10.000    Max.    :1.00000    Max.    :701842551    Max.    :0.8547
##
##           status    adult    genre_names
##   In Production : 0    Mode :logical    Horror, Thriller    :103
##   Planned       : 0    FALSE:659    Horror            : 99
##   Post Production: 0    Horror, Mystery, Thriller: 52
##   Released      :659    Comedy, Horror     : 36
##               Horror, Science Fiction : 29
##               Drama, Horror, Thriller : 27
##               (Other)                 :313
##   release_year    budget_category    profit    success
##   Min.    :0.0000    Length:659    Min.    : -194775779    No Success:212
##   1st Qu.:0.6014    Class :character    1st Qu.:   -99162    Success   :447
##   Median :0.7826    Mode  :character    Median :    3400000
##   Mean    :0.7259    Mean    : 26275179
##   3rd Qu.:0.8986    3rd Qu.: 30288153
##   Max.    :1.0000    Max.    : 666842551
##
```

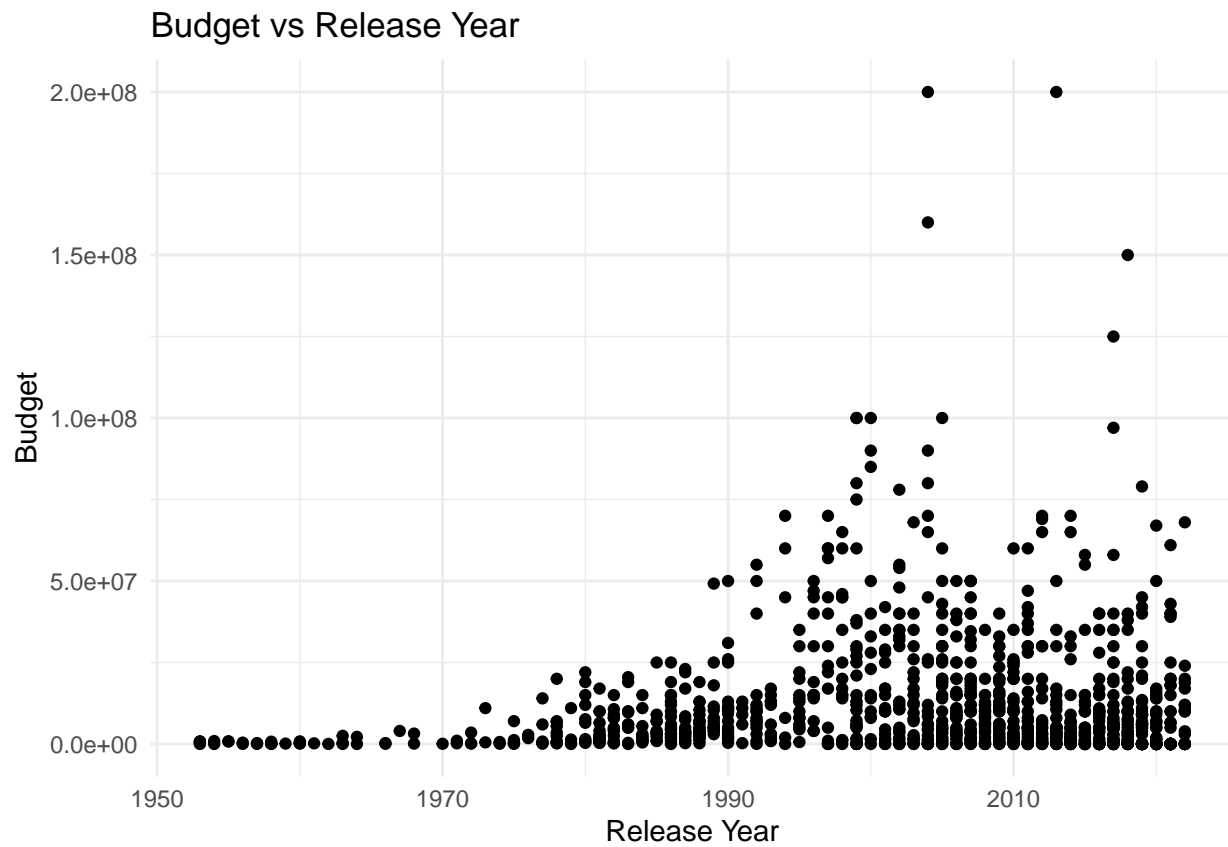
## Visualizing the Relationships

```
library(ggplot2)

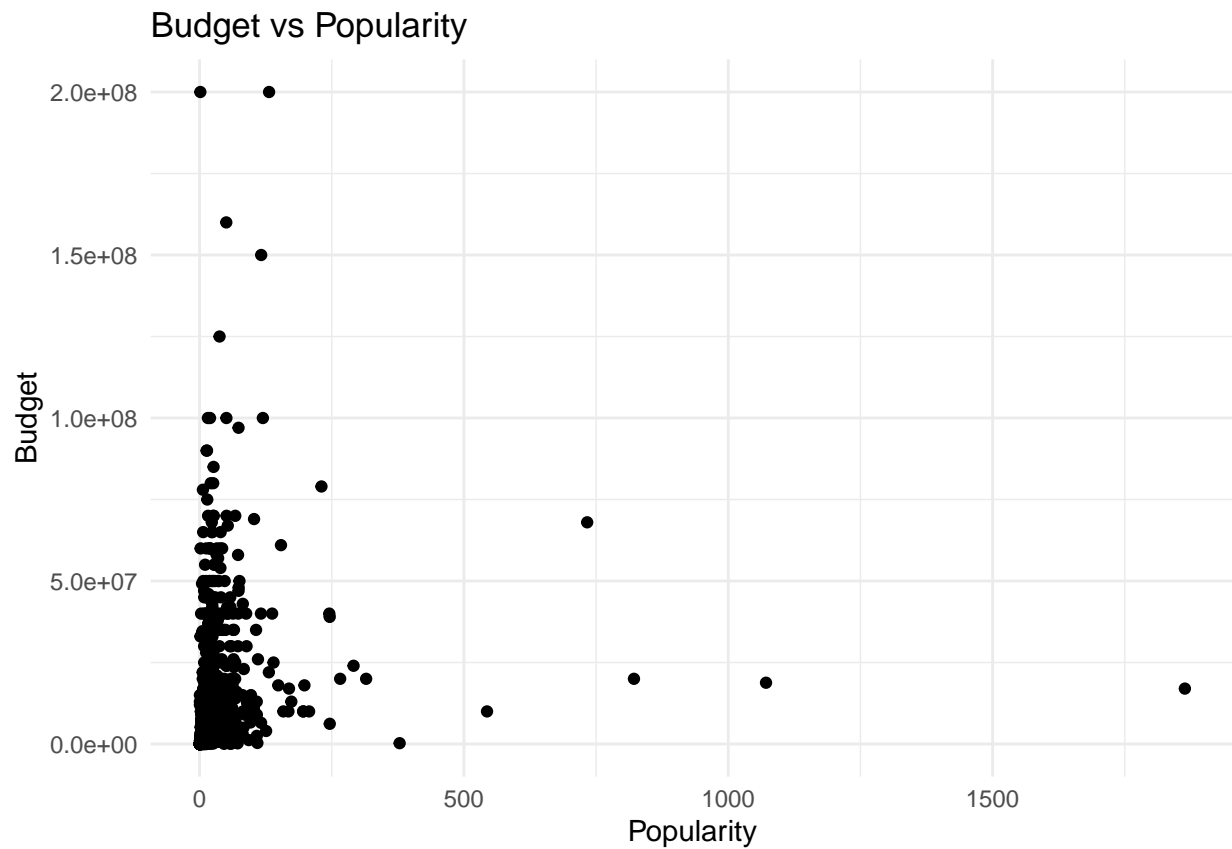
# Scatter plots to see the relationships
ggplot(df_for_class, aes(x = revenue, y = budget)) +
  geom_point() +
  labs(title = "Budget vs Revenue", x = "Revenue", y = "Budget") +
  theme_minimal()
```



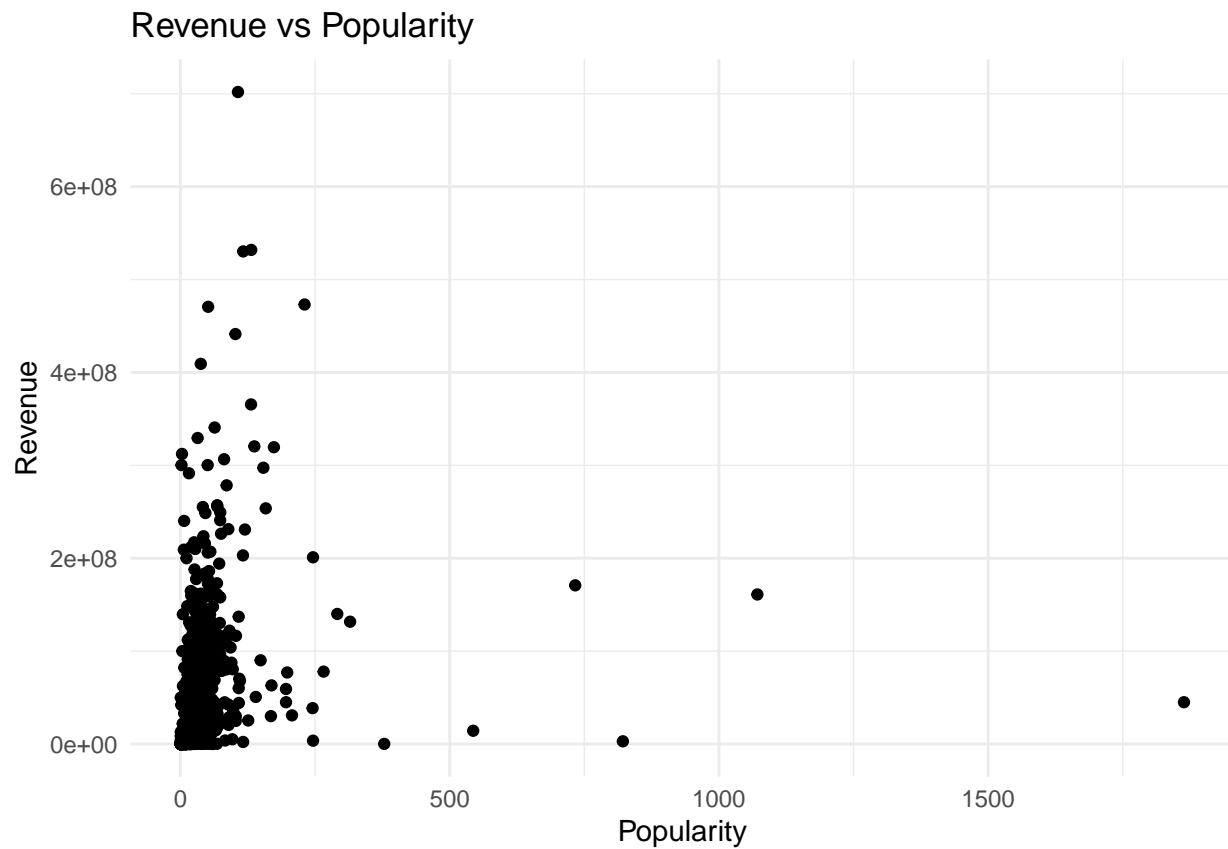
```
ggplot(df_for_class, aes(x = release_year, y = budget)) +  
  geom_point() +  
  labs(title = "Budget vs Release Year", x = "Release Year", y = "Budget") +  
  theme_minimal()
```



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

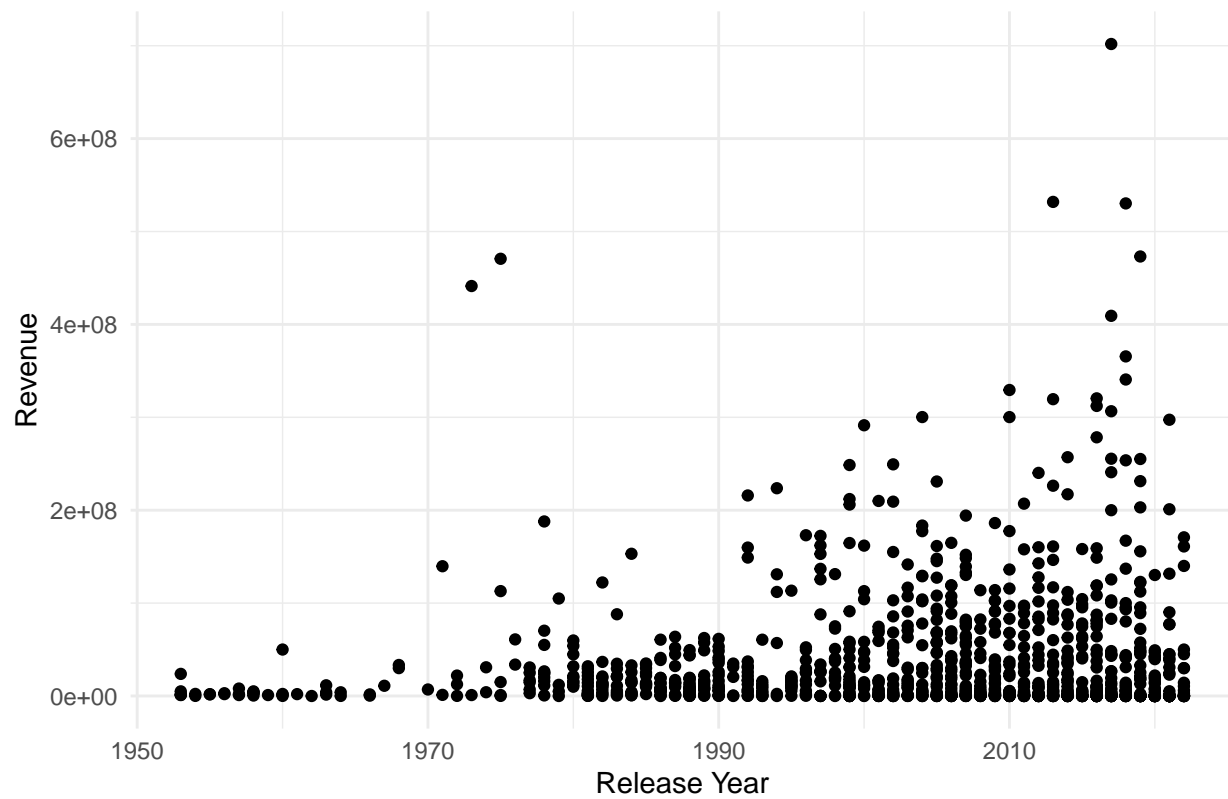


```
ggplot(df_for_class, aes(x = popularity, y = revenue)) +  
  geom_point() +  
  labs(title = "Revenue vs Popularity", x = "Popularity", y = "Revenue") +  
  theme_minimal()
```



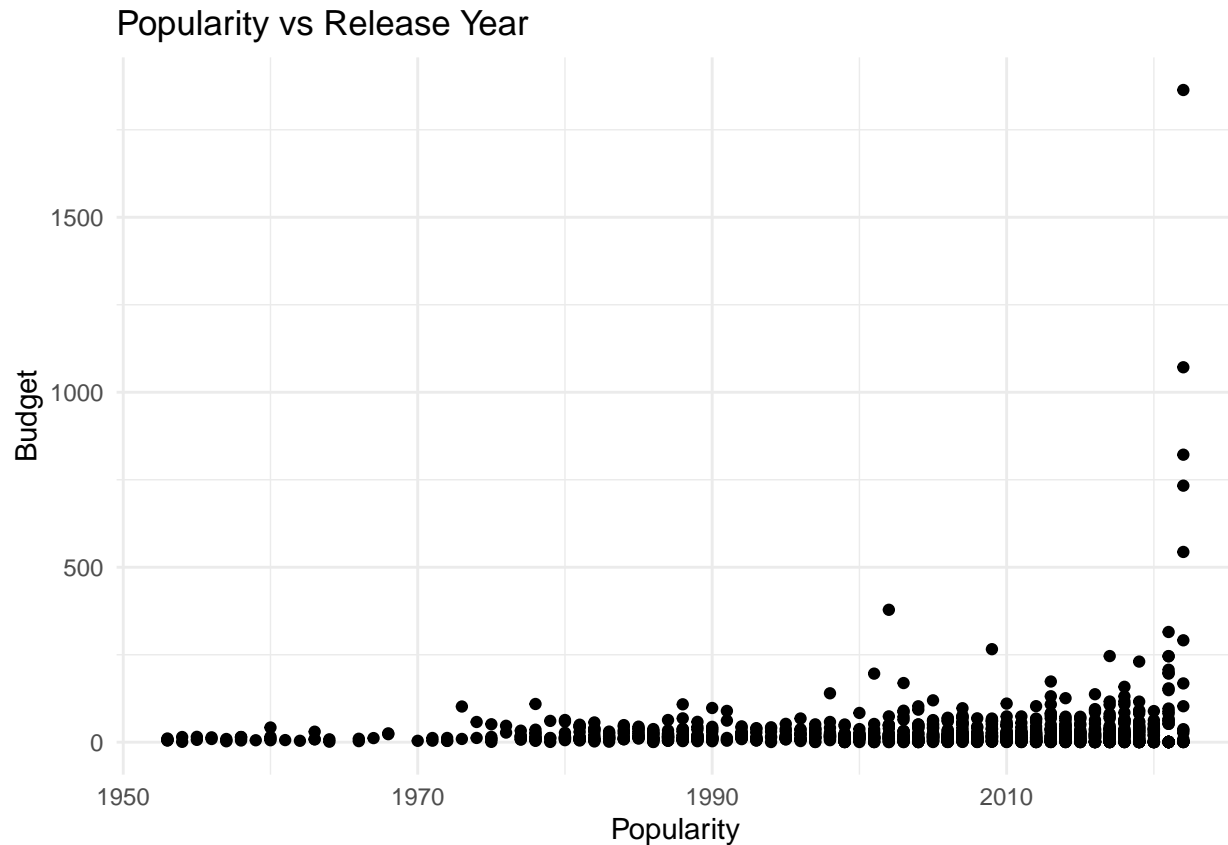
```
ggplot(df_for_class, aes(x = release_year, y = revenue)) +  
  geom_point() +  
  labs(title = "Revenue vs Release Year", x = "Release Year", y = "Revenue") +  
  theme_minimal()
```

Revenue vs Release Year



```
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 = training_n)
```

```
lda_model
```

```
## Call:
## lda(success ~ budget + release_year + runtime + popularity, data = training_n)
##
## Prior probabilities of groups:
## No Success    Success
##  0.3216995    0.6783005
##
## Group means:
##               budget release_year  runtime  popularity
## No Success  0.05370571    0.7686628 0.4738326 0.006290148
## Success     0.06865487    0.7056058 0.5254771 0.017916457
##
## Coefficients of linear discriminants:
##               LD1
## budget          1.005574
## release_year   -2.732860
## runtime         2.633095
## popularity     23.712963
```

## Predicting

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

```
lda_predictions <- predict(lda_model, newdata = testing_n)$class

prediction_counts_lda <- table(lda_predictions)
print(prediction_counts_lda)
```

```
## lda_predictions
## No Success      Success
##           35          404
```

## 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_predictions, Actual = testing_n$success)
print(confusion_matrix_lda)
```

```
##           Actual
## Predicted   No Success Success
## No Success         19        16
## Success          134        270
```

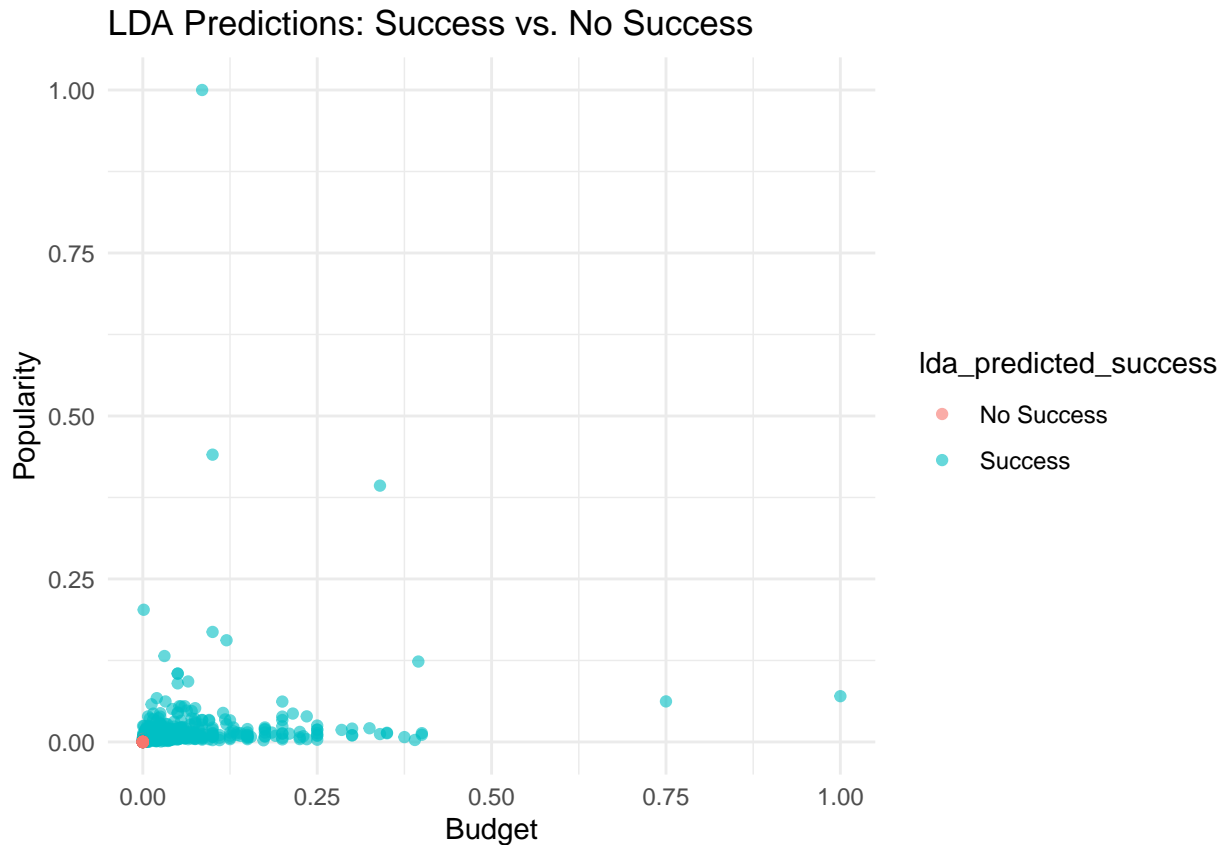
```
accuracy_lda <- sum(lda_predictions == testing_n$success) / length(lda_predictions)
print(paste("Accuracy: ", round(accuracy_lda * 100, 4), "%"))
```

```
## [1] "Accuracy: 65.8314 %"
```

## Visualizing Decision Boundaries

```
testing_n$lda_predicted_success <- lda_predictions

library(ggplot2)
ggplot(testing_n, aes(x = budget, y = popularity, color = lda_predicted_success)) +
  geom_point(alpha = 0.6) +
  labs(title = "LDA Predictions: Success vs. No Success", x = "Budget", y = "Popularity") +
  theme_minimal()
```



## PDP

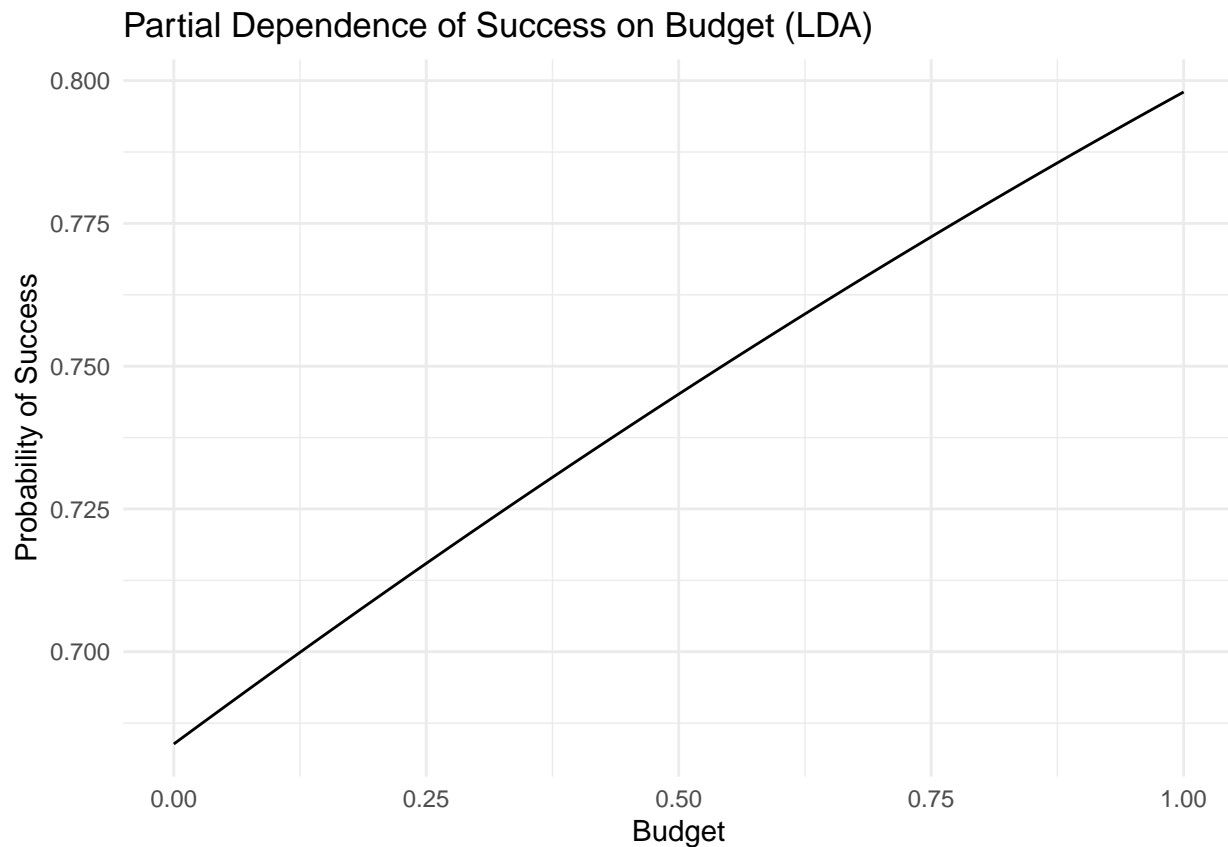
```
budget_range <- seq(from = min(training_n$budget, na.rm = TRUE),
                    to = max(training_n$budget, na.rm = TRUE),
                    length.out = 100)

pdp_data <- data.frame(
  budget = budget_range,
  release_year = rep(mean(training_n$release_year, na.rm = TRUE), 100),
  runtime = rep(mean(training_n$runtime, na.rm = TRUE), 100),
  popularity = rep(mean(training_n$popularity, na.rm = TRUE), 100)
)

pred_probs <- predict(lda_model, newdata = pdp_data, type = "response")$posterior[,2]

pdp_data$success_probability <- pred_probs

ggplot(pdp_data, aes(x = budget, y = success_probability)) +
  geom_line() +
  labs(title = "Partial Dependence of Success on Budget (LDA)",
       x = "Budget",
       y = "Probability of Success") +
  theme_minimal()
```



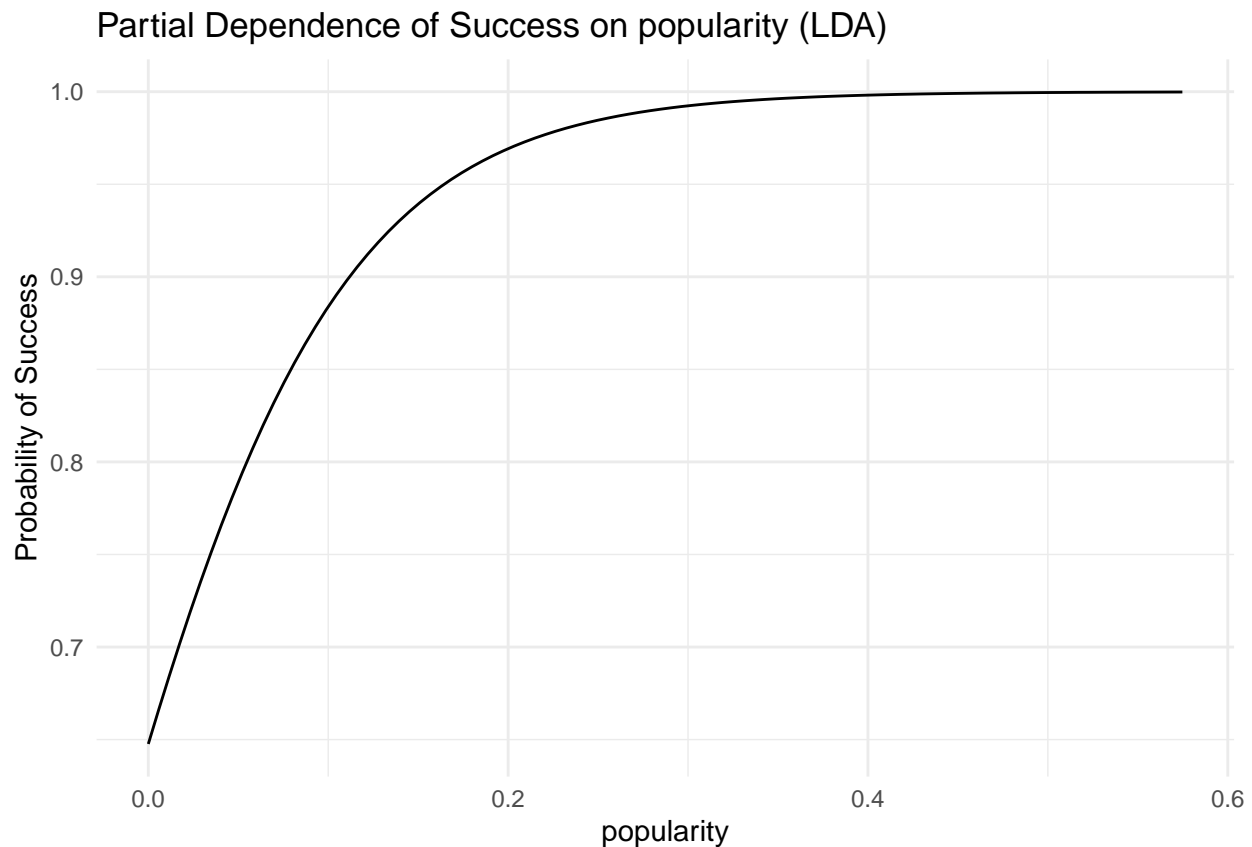
```
pop_range <- seq(from = min(training_n$popularity, na.rm = TRUE),
                 to = max(training_n$popularity, na.rm = TRUE),
                 length.out = 100)

pdp_data2 <- data.frame(
  popularity = pop_range,
  release_year = rep(mean(training_n$release_year, na.rm = TRUE), 100),
  runtime = rep(mean(training_n$runtime, na.rm = TRUE), 100),
  budget = rep(mean(training_n$budget, na.rm = TRUE), 100)
)

pred_probs2 <- predict(lda_model, newdata = pdp_data2, type = "response")$posterior[,2]

pdp_data2$success_probability <- pred_probs2

ggplot(pdp_data2, aes(x = popularity, y = success_probability)) +
  geom_line() +
  labs(title = "Partial Dependence of Success on popularity (LDA)",
       x = "popularity",
       y = "Probability of Success") +
  theme_minimal()
```



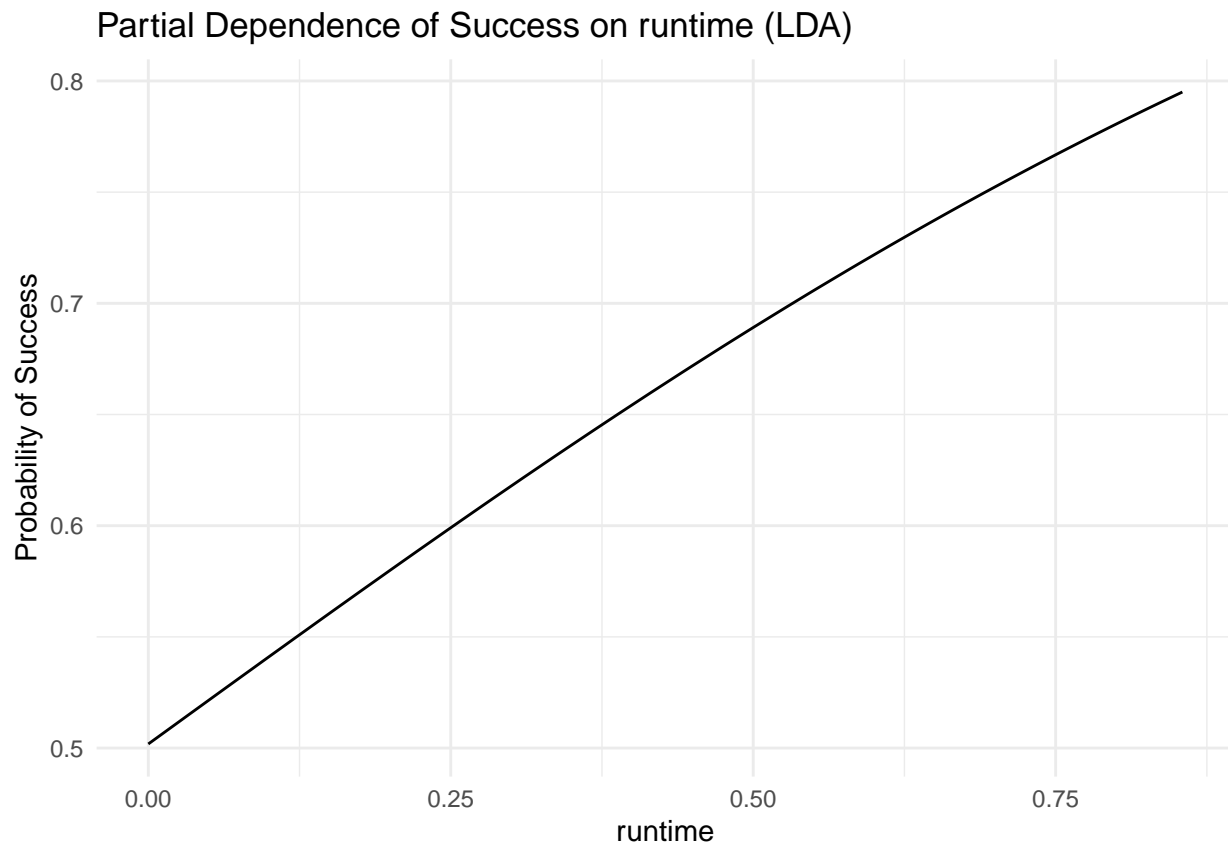
```
rt_range <- seq(from = min(training_n$runtime, na.rm = TRUE),
               to = max(training_n$runtime, na.rm = TRUE),
               length.out = 100)

pdp_data3 <- data.frame(
  runtime = rt_range,
  release_year = rep(mean(training_n$release_year, na.rm = TRUE), 100),
  popularity = rep(mean(training_n$popularity, na.rm = TRUE), 100),
  budget = rep(mean(training_n$budget, na.rm = TRUE), 100)
)

pred_probs3 <- predict(lda_model, newdata = pdp_data3, type = "response")$posterior[,2]

pdp_data3$success_probability <- pred_probs3

ggplot(pdp_data3, aes(x = runtime, y = success_probability)) +
  geom_line() +
  labs(title = "Partial Dependence of Success on runtime (LDA)",
       x = "runtime",
       y = "Probability of Success") +
  theme_minimal()
```



```
ry_range <- seq(from = min(training_n$release_year, na.rm = TRUE),
               to = max(training_n$release_year, na.rm = TRUE),
               length.out = 100)

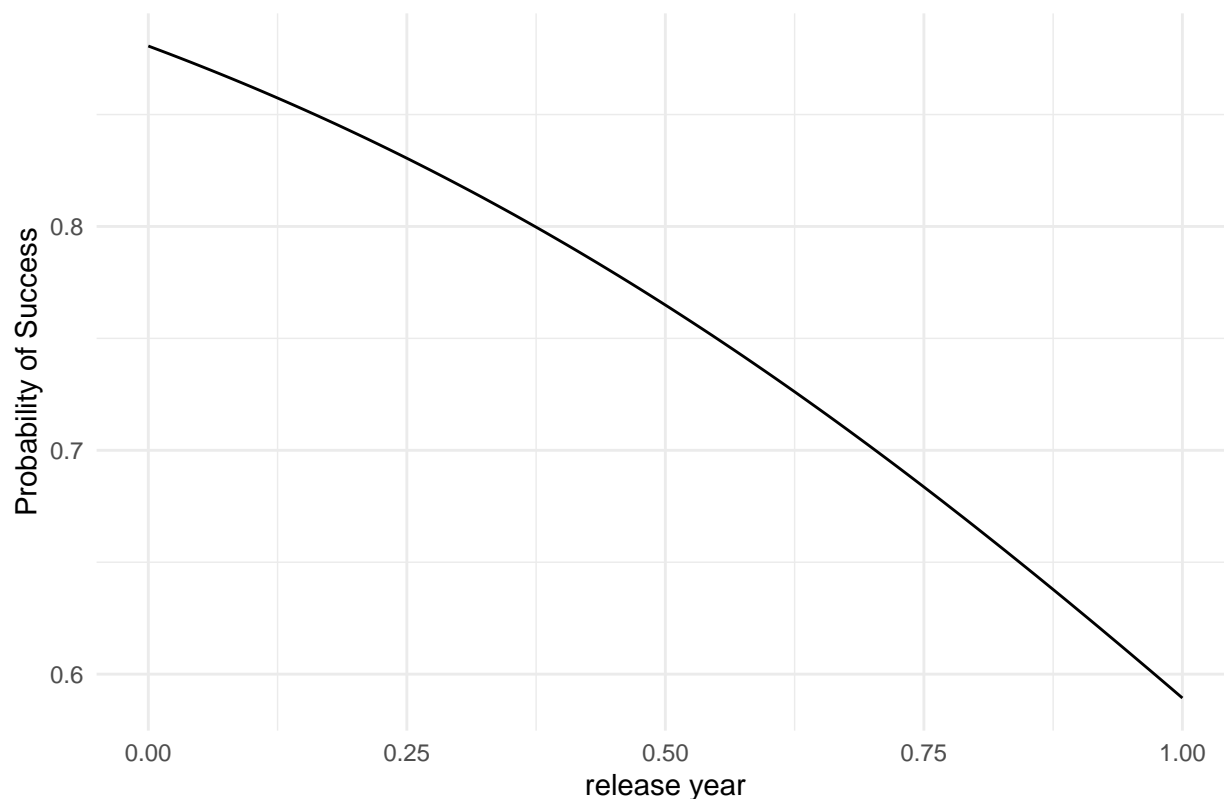
pdp_data4 <- data.frame(
  release_year = ry_range,
  runtime = rep(mean(training_n$runtime, na.rm = TRUE), 100),
  popularity = rep(mean(training_n$popularity, na.rm = TRUE), 100),
  budget = rep(mean(training_n$budget, na.rm = TRUE), 100)
)

pred_probs4 <- predict(lda_model, newdata = pdp_data4, type = "response")$posterior[,2]

pdp_data4$success_probability <- pred_probs4

ggplot(pdp_data4, aes(x = release_year, y = success_probability)) +
  geom_line() +
  labs(title = "Partial Dependence of Success on release year (LDA)",
       x = "release year",
       y = "Probability of Success") +
  theme_minimal()
```

## Partial Dependence of Success on release year (LDA)



## Classifying Success with QDA

### Training the model

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

```
qda_model
```

```
## Call:
```

```
## qda(success ~ budget + release_year + runtime + popularity, data = training_n)
```

```
##
```

```
## Prior probabilities of groups:
```

```
## No Success    Success
```

```
## 0.3216995 0.6783005
```

```
##
```

```
## Group means:
```

```
##          budget release_year  runtime  popularity
```

```
## No Success 0.05370571    0.7686628 0.4738326 0.006290148
```

```
## Success   0.06865487    0.7056058 0.5254771 0.017916457
```

### Predicting

```
qda_predictions = predict(qda_model, newdata = testing_n)$class
```

```
prediction_counts_qda <- table(qda_predictions)
```

```
print(prediction_counts_qda)
```

```
## qda_predictions
## No Success      Success
##           246          193
```

## Evaluating the Model

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

```
confusion_matrix_qda <- table(Predicted = qda_predictions, Actual = testing_n$success)
print(confusion_matrix_qda)
```

```
##           Actual
## Predicted   No Success Success
## No Success      117      129
## Success         36      157
```

```
accuracy_qda <- sum(qda_predictions == testing_n$success) / length(qda_predictions)
print(paste("Accuracy: ", round(accuracy_qda * 100, 4), "%"))
```

```
## [1] "Accuracy: 62.4146 %"
```

## PDP

```
budget_range_qda <- seq(from = min(training_n$budget, na.rm = TRUE),
                        to = max(training_n$budget, na.rm = TRUE),
                        length.out = 100)

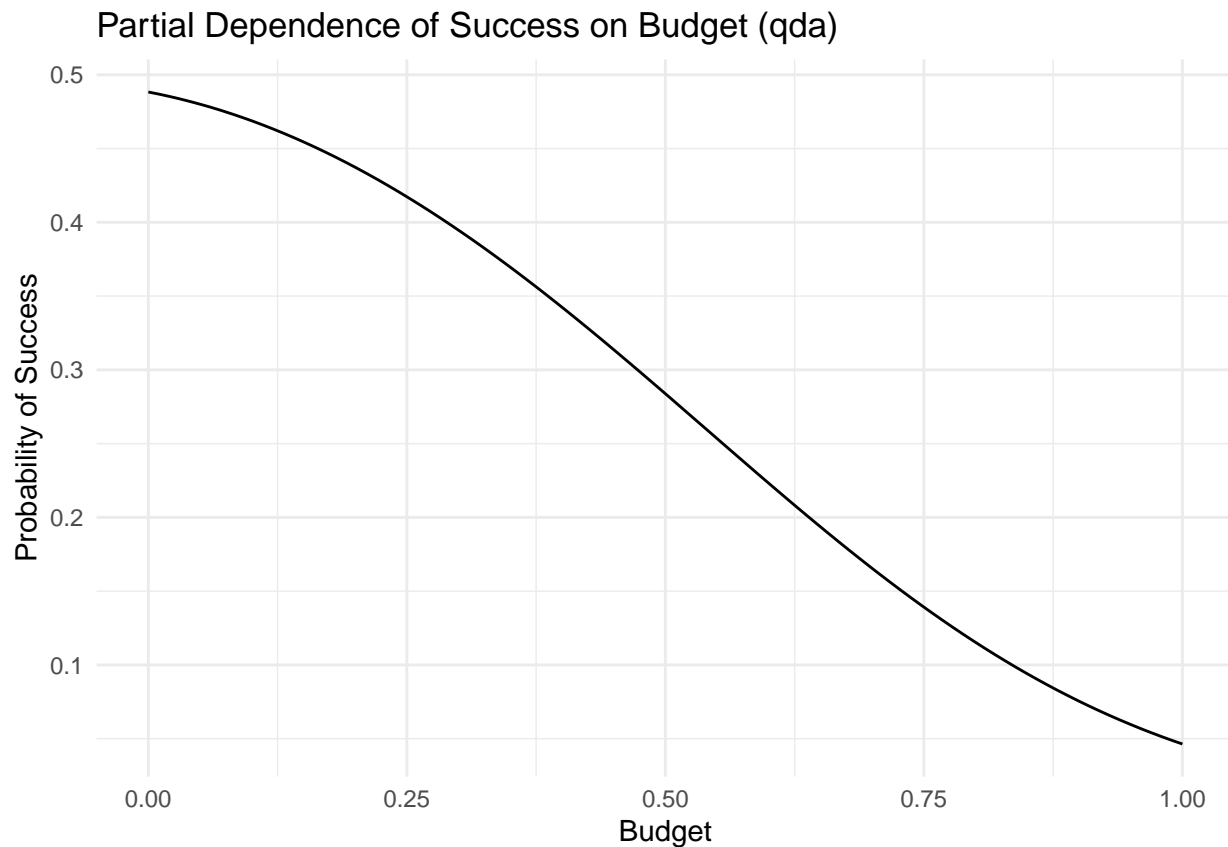
pdp_data_qda <- data.frame(
  budget = budget_range_qda,
  release_year = rep(mean(training_n$release_year, na.rm = TRUE), 100),
  runtime = rep(mean(training_n$runtime, na.rm = TRUE), 100),
  popularity = rep(mean(training_n$popularity, na.rm = TRUE), 100)
)

pred_probs_qda <- predict(qda_model, newdata = pdp_data_qda, type = "response")$posterior[,2]

pdp_data_qda$success_probability <- pred_probs_qda

ggplot(pdp_data_qda, aes(x = budget, y = success_probability)) +
  geom_line() +
  labs(title = "Partial Dependence of Success on Budget (qda)",
       x = "Budget",
       y = "Probability of Success") +
  theme_minimal()
```





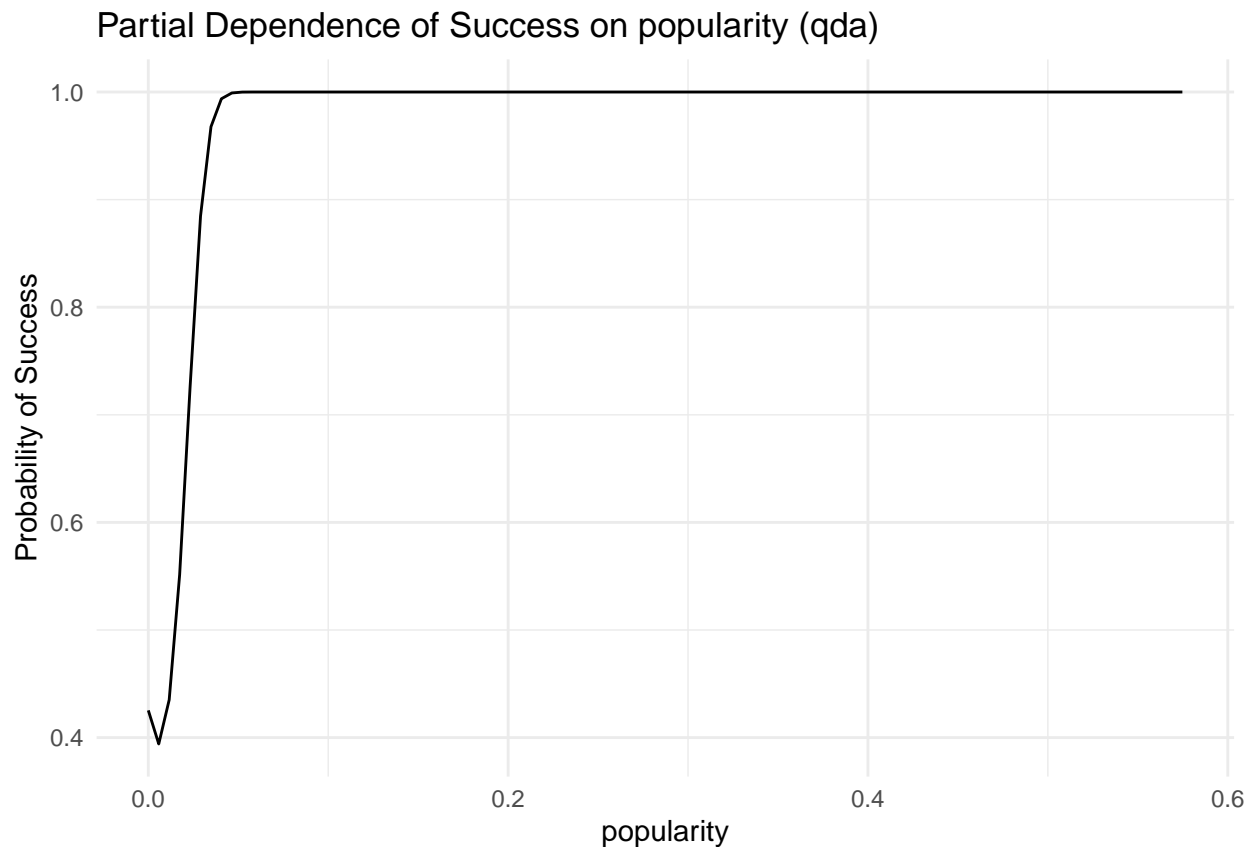
```
pop_range_qda <- seq(from = min(training_n$popularity, na.rm = TRUE),
                     to = max(training_n$popularity, na.rm = TRUE),
                     length.out = 100)

pdp_data2_qda <- data.frame(
  popularity = pop_range_qda,
  release_year = rep(mean(training_n$release_year, na.rm = TRUE), 100),
  runtime = rep(mean(training_n$runtime, na.rm = TRUE), 100),
  budget = rep(mean(training_n$budget, na.rm = TRUE), 100)
)

pred_probs2_qda <- predict(qda_model, newdata = pdp_data2_qda, type = "response")$posterior[,2]

pdp_data2_qda$success_probability <- pred_probs2_qda

ggplot(pdp_data2_qda, aes(x = popularity, y = success_probability)) +
  geom_line() +
  labs(title = "Partial Dependence of Success on popularity (qda)",
       x = "popularity",
       y = "Probability of Success") +
  theme_minimal()
```



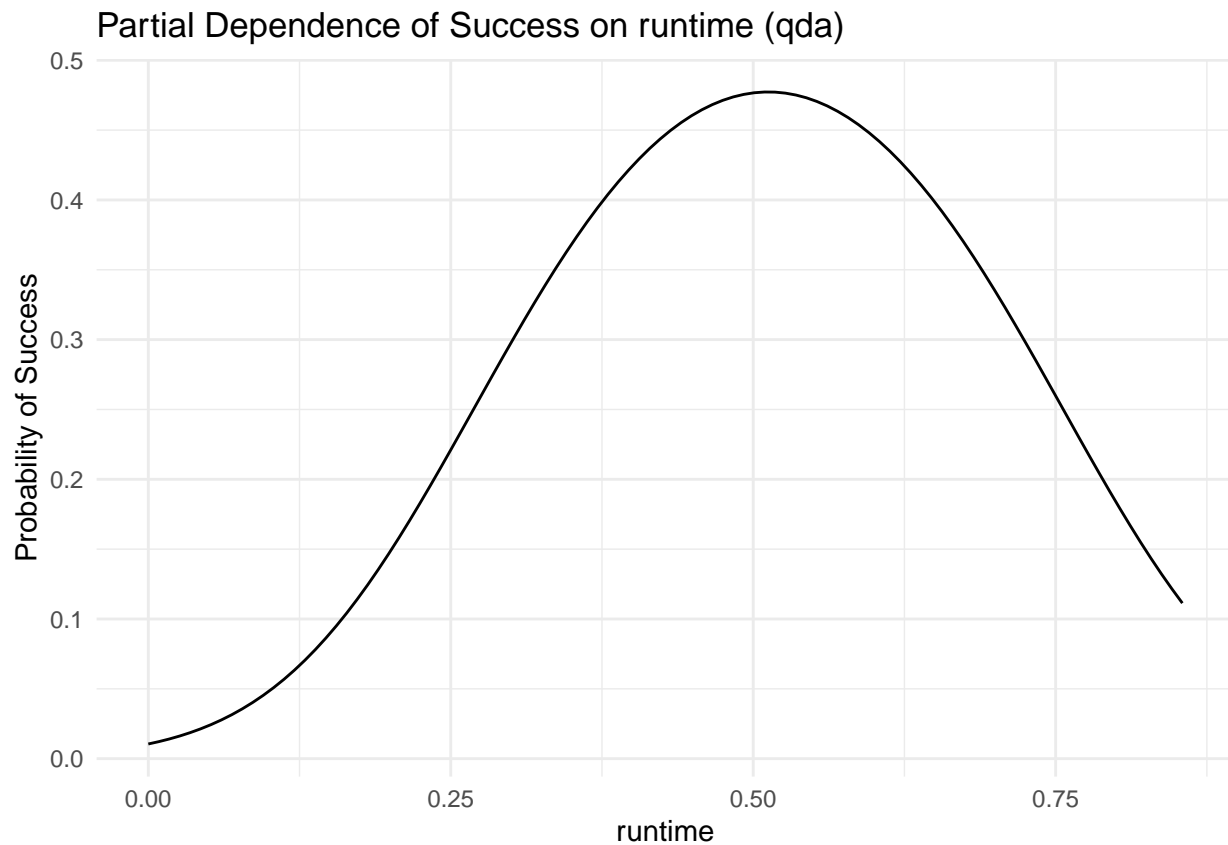
```
rt_range_qda <- seq(from = min(training_n$runtime, na.rm = TRUE),
                    to = max(training_n$runtime, na.rm = TRUE),
                    length.out = 100)

pdp_data3_qda <- data.frame(
  runtime = rt_range_qda,
  release_year = rep(mean(training_n$release_year, na.rm = TRUE), 100),
  popularity = rep(mean(training_n$popularity, na.rm = TRUE), 100),
  budget = rep(mean(training_n$budget, na.rm = TRUE), 100)
)

pred_probs3_qda <- predict(qda_model, newdata = pdp_data3_qda, type = "response")$posterior[,2]

pdp_data3_qda$success_probability <- pred_probs3_qda

ggplot(pdp_data3_qda, aes(x = runtime, y = success_probability)) +
  geom_line() +
  labs(title = "Partial Dependence of Success on runtime (qda)",
       x = "runtime",
       y = "Probability of Success") +
  theme_minimal()
```



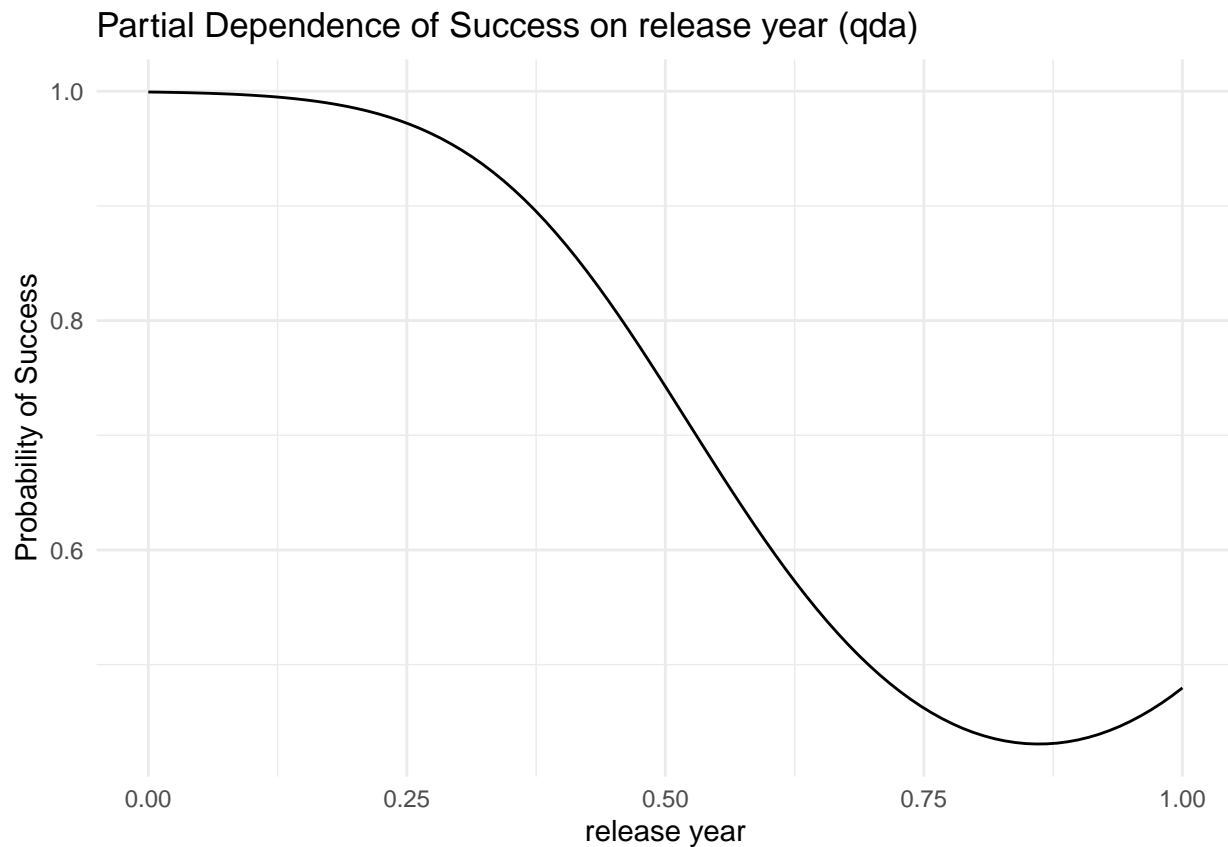
```
ry_range_qda <- seq(from = min(training_n$release_year, na.rm = TRUE),
                    to = max(training_n$release_year, na.rm = TRUE),
                    length.out = 100)

pdp_data4_qda <- data.frame(
  release_year = ry_range_qda,
  runtime = rep(mean(training_n$runtime, na.rm = TRUE), 100),
  popularity = rep(mean(training_n$popularity, na.rm = TRUE), 100),
  budget = rep(mean(training_n$budget, na.rm = TRUE), 100)
)

pred_probs4_qda <- predict(qda_model, newdata = pdp_data4, type = "response")$posterior[,2]

pdp_data4_qda$success_probability <- pred_probs4_qda

ggplot(pdp_data4_qda, aes(x = release_year, y = success_probability)) +
  geom_line() +
  labs(title = "Partial Dependence of Success on release year (qda)",
       x = "release year",
       y = "Probability of Success") +
  theme_minimal()
```



## 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

```
library(e1071)

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

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
## 0.3216995    0.6783005
##
## Conditional probabilities:
##                budget
## Y                [,1]    [,2]
```

```
## No Success 10741143 19918339
## Success 13730975 19930261
##
## release_year
## Y [,1] [,2]
## No Success 2006.038 11.47751
## Success 2001.687 15.63314
##
## runtime
## Y [,1] [,2]
## No Success 84.81604 32.49614
## Success 94.06040 22.54887
##
## popularity
## Y [,1] [,2]
## No Success 12.31872 20.38792
## Success 33.97886 64.34544
```

## Predicting

```
predictions_nb <- predict(nb_model, testing)

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

```
## predictions_nb
## No Success Success
## 223 216
```

## Evaluating the Model

We will use a confusion matrix to show model accuracy.

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

```
## Actual
## Predicted_NB No Success Success
## No Success 107 116
## Success 46 170
```

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

```
## [1] "Accuracy: 63.0979 %"
```

## PDP

```
budget_range_nb <- seq(from = min(training_n$budget, na.rm = TRUE),
                        to = max(training_n$budget, na.rm = TRUE),
                        length.out = 100)

pdp_data_nb <- data.frame(
  budget = budget_range_nb,
  release_year = rep(mean(training_n$release_year, na.rm = TRUE), 100),
```

```

runtime = rep(mean(training_n$runtime, na.rm = TRUE), 100),
popularity = rep(mean(training_n$popularity, na.rm = TRUE), 100)
)

pred_probs_nb <- predict(nb_model, newdata = pdp_data_nb, type = "raw")

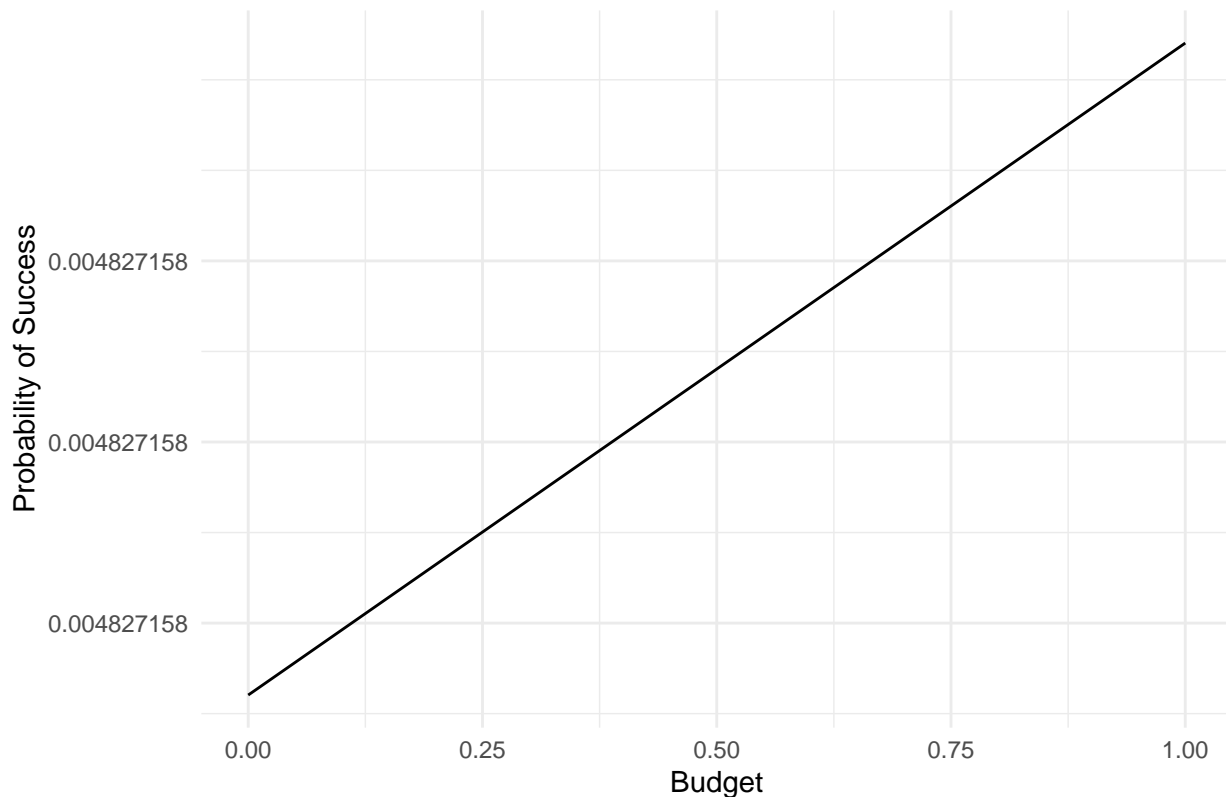
success_probability <- pred_probs_nb[, 2]

pdp_data_nb$success_probability <- success_probability

ggplot(pdp_data_nb, aes(x = budget, y = success_probability)) +
  geom_line() +
  labs(title = "Partial Dependence of Success on Budget (nb)",
       x = "Budget",
       y = "Probability of Success") +
  theme_minimal()

```

Partial Dependence of Success on Budget (nb)



```

pop_range_nb <- seq(from = min(training_n$popularity, na.rm = TRUE),
                    to = max(training_n$popularity, na.rm = TRUE),
                    length.out = 100)

pdp_data2_nb <- data.frame(
  popularity = pop_range_nb,
  release_year = rep(mean(training_n$release_year, na.rm = TRUE), 100),
  runtime = rep(mean(training_n$runtime, na.rm = TRUE), 100),
  budget = rep(mean(training_n$budget, na.rm = TRUE), 100)
)

```

```

pred_probs_nb2 <- predict(nb_model, newdata = pdp_data2_nb, type = "raw")

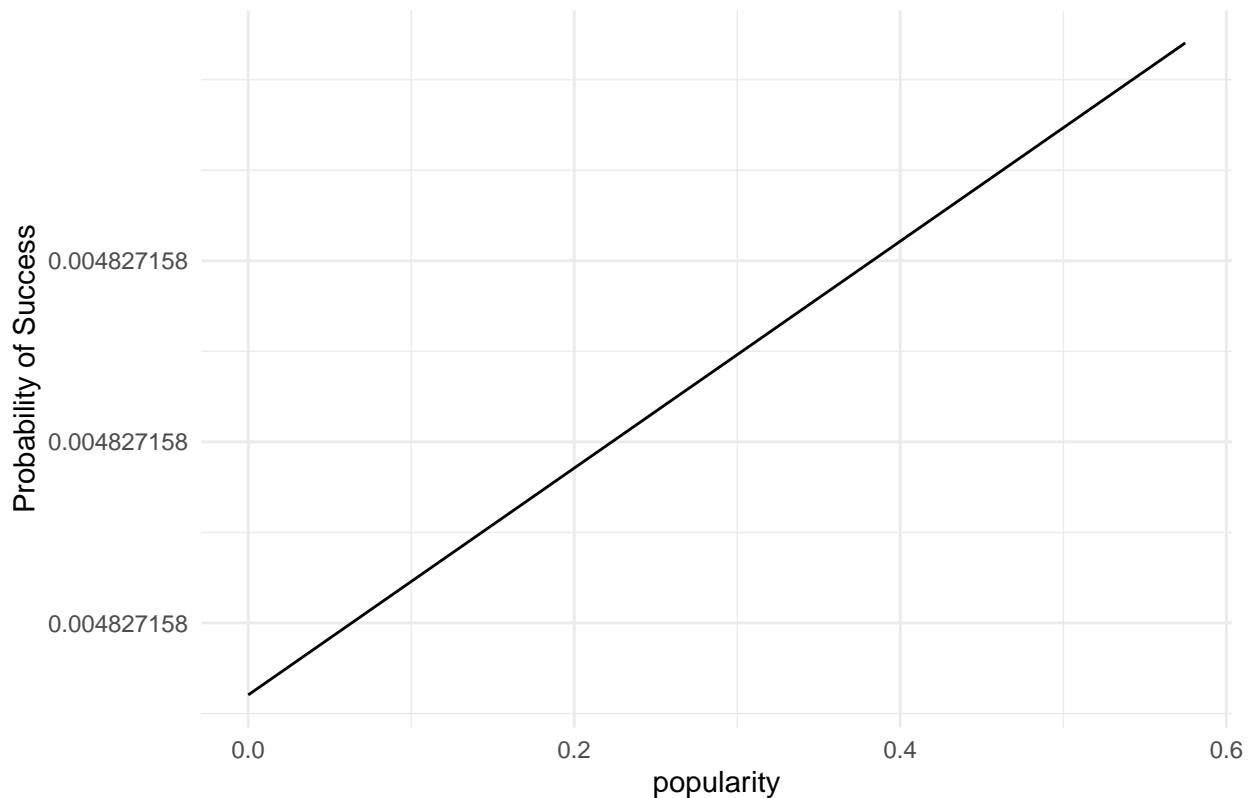
success_probability2 <- pred_probs_nb[, 2]

pdp_data2_nb$success_probability2 <- success_probability2

ggplot(pdp_data2_nb, aes(x = popularity, y = success_probability2)) +
  geom_line() +
  labs(title = "Partial Dependence of Success on popularity (nb)",
       x = "popularity",
       y = "Probability of Success") +
  theme_minimal()

```

Partial Dependence of Success on popularity (nb)



```

rt_range_nb <- seq(from = min(training_n$runtime, na.rm = TRUE),
                  to = max(training_n$runtime, na.rm = TRUE),
                  length.out = 100)

pdp_data3_nb <- data.frame(
  runtime = rt_range_nb,
  release_year = rep(mean(training_n$release_year, na.rm = TRUE), 100),
  popularity = rep(mean(training_n$popularity, na.rm = TRUE), 100),
  budget = rep(mean(training_n$budget, na.rm = TRUE), 100)
)

pred_probs3_nb <- predict(nb_model, newdata = pdp_data3_nb, type = "raw")

```

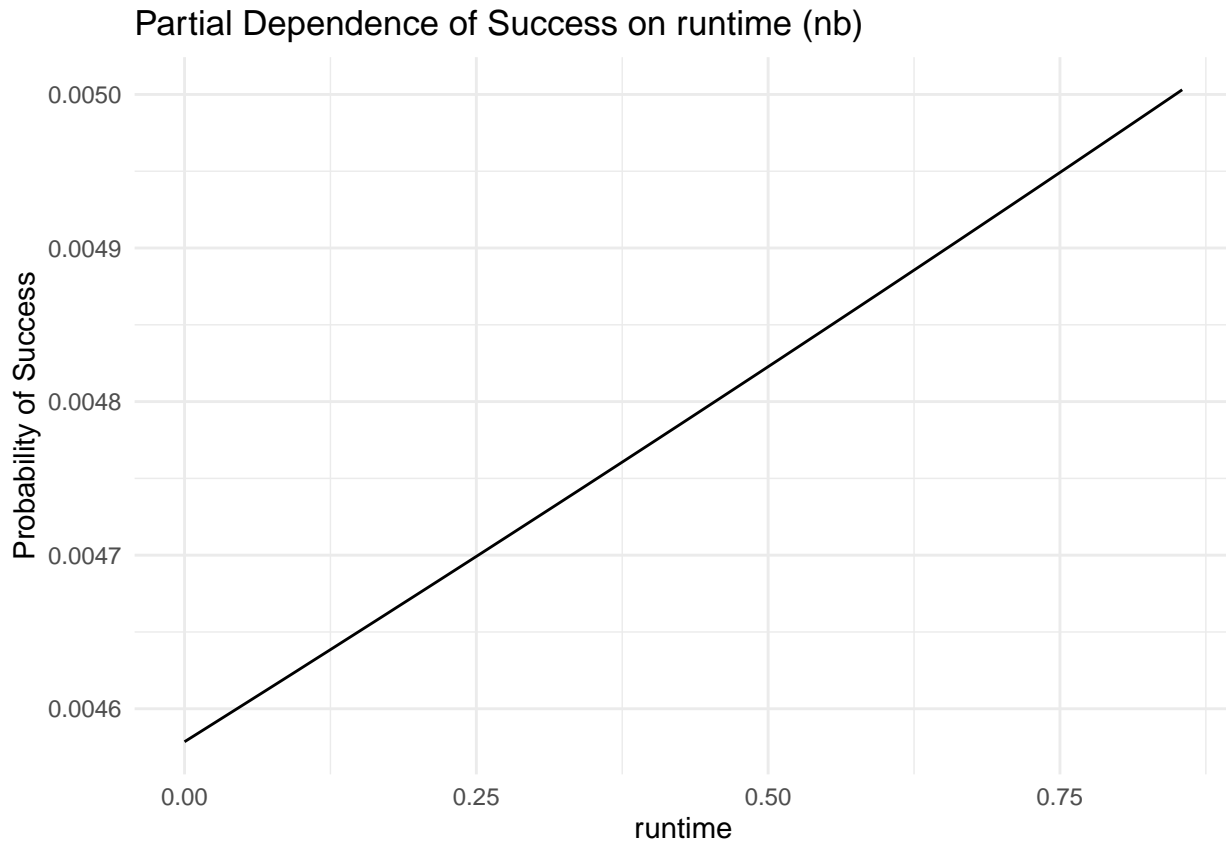
```

success_probability3 <- pred_probs3_nb[, 2]

pdp_data3_nb$success_probability3 <- success_probability3

ggplot(pdp_data3_nb, aes(x = runtime, y = success_probability3)) +
  geom_line() +
  labs(title = "Partial Dependence of Success on runtime (nb)",
       x = "runtime",
       y = "Probability of Success") +
  theme_minimal()

```



```

ry_range_nb <- seq(from = min(training_n$release_year, na.rm = TRUE),
                  to = max(training_n$release_year, na.rm = TRUE),
                  length.out = 100)

pdp_data4_nb <- data.frame(
  release_year = ry_range_nb,
  runtime = rep(mean(training_n$runtime, na.rm = TRUE), 100),
  popularity = rep(mean(training_n$popularity, na.rm = TRUE), 100),
  budget = rep(mean(training_n$budget, na.rm = TRUE), 100)
)

pred_probs4_nb <- predict(nb_model, newdata = pdp_data4_nb, type = "raw")

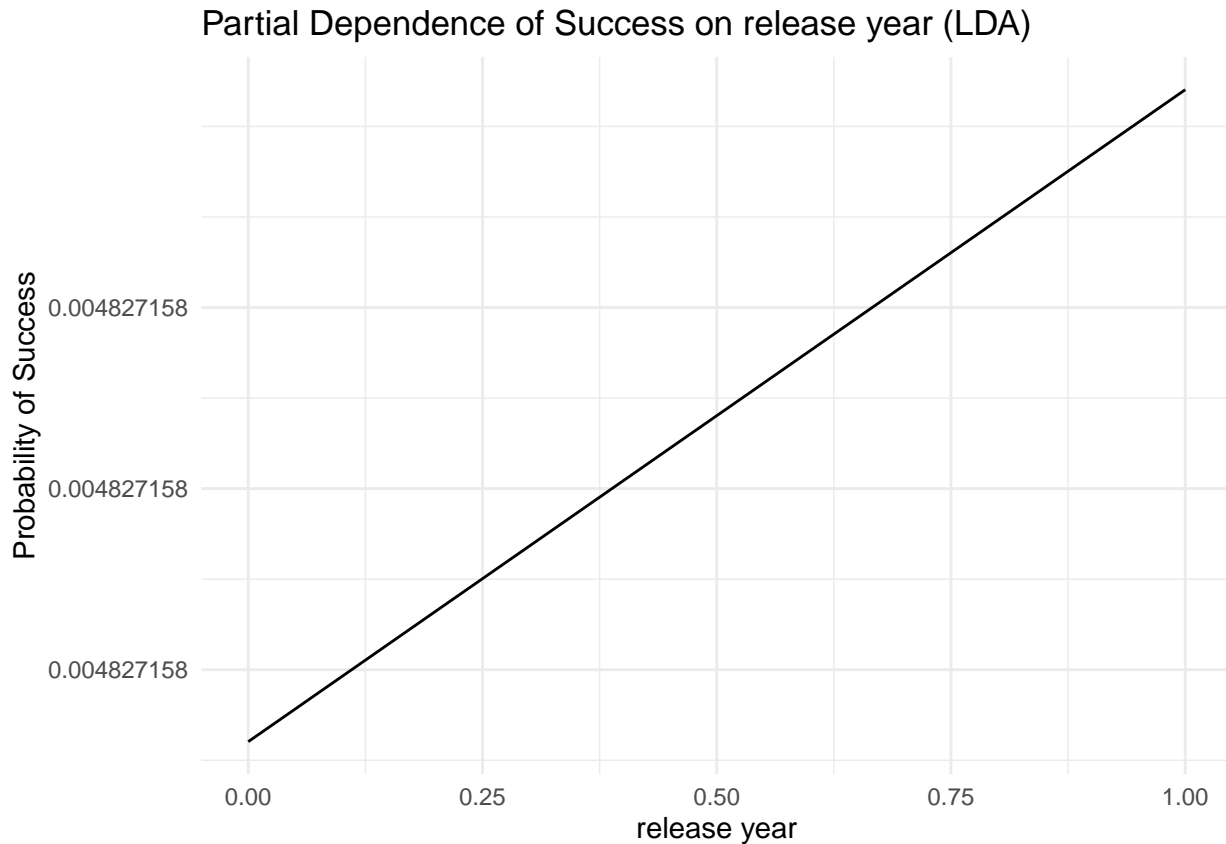
success_probability4 <- pred_probs_nb[, 2]

pdp_data4_nb$success_probability4 <- success_probability4

```



```
ggplot(pdp_data4_nb, aes(x = release_year, y = success_probability4)) +
  geom_line() +
  labs(title = "Partial Dependence of Success on release year (LDA)",
       x = "release year",
       y = "Probability of Success") +
  theme_minimal()
```



## 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 = training_n)[, -1]
y <- ifelse(training_n$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: 0.00857571602237266"

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
## 70 589
```

**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  40  30
##           1 172 417

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

## [1] "Accuracy of Ridge Regression: 69.3475 %"

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

##              s0
## (Intercept) 1.1558322
## budget      -0.3046535
## release_year -1.7238692
```

```
## runtime      0.6027949
## popularity    53.7195835
```

## 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.

```
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:  0.000679610828887"

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.

```
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
## 116 543
```

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

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

##           Actual
## Predicted    0    1
##           0  60  56
##           1 152 391

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

## [1] "Accuracy of Lasso Regression:  68.437 %"

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

```
##           s0
## (Intercept)  1.34153592
## budget      -0.89243201
## release_year -1.94555545
## runtime      0.06609342
## popularity   87.50579005
```

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

```
testing_lg = testing_n
training_lg = training_n

testing_lg$success <- ifelse(testing_lg$success == "Success", 1, 0)
training_lg$success <- ifelse(training_lg$success == "Success", 1, 0)

lg_model <- glm(success ~ budget + release_year + runtime + popularity,
               data = training_lg,
               family = "binomial")

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

### Predicting:

```
probabilities_lg <- predict(lg_model, newdata = testing_lg, type = "response")

predictions_lg <- ifelse(probabilities_lg > 0.5, 1, 0)

prediction_lg_counts <- table(predictions_lg)
print(prediction_lg_counts)

## predictions_lg
##    0    1
## 78 361
```

### Evaluating the model

```
confusion_matrix_lg <- table(Predicted = predictions_lg, Actual = testing_lg$success)
print(confusion_matrix_lg)

##           Actual
## Predicted    0    1
##           0  39  39
##           1 114 247

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

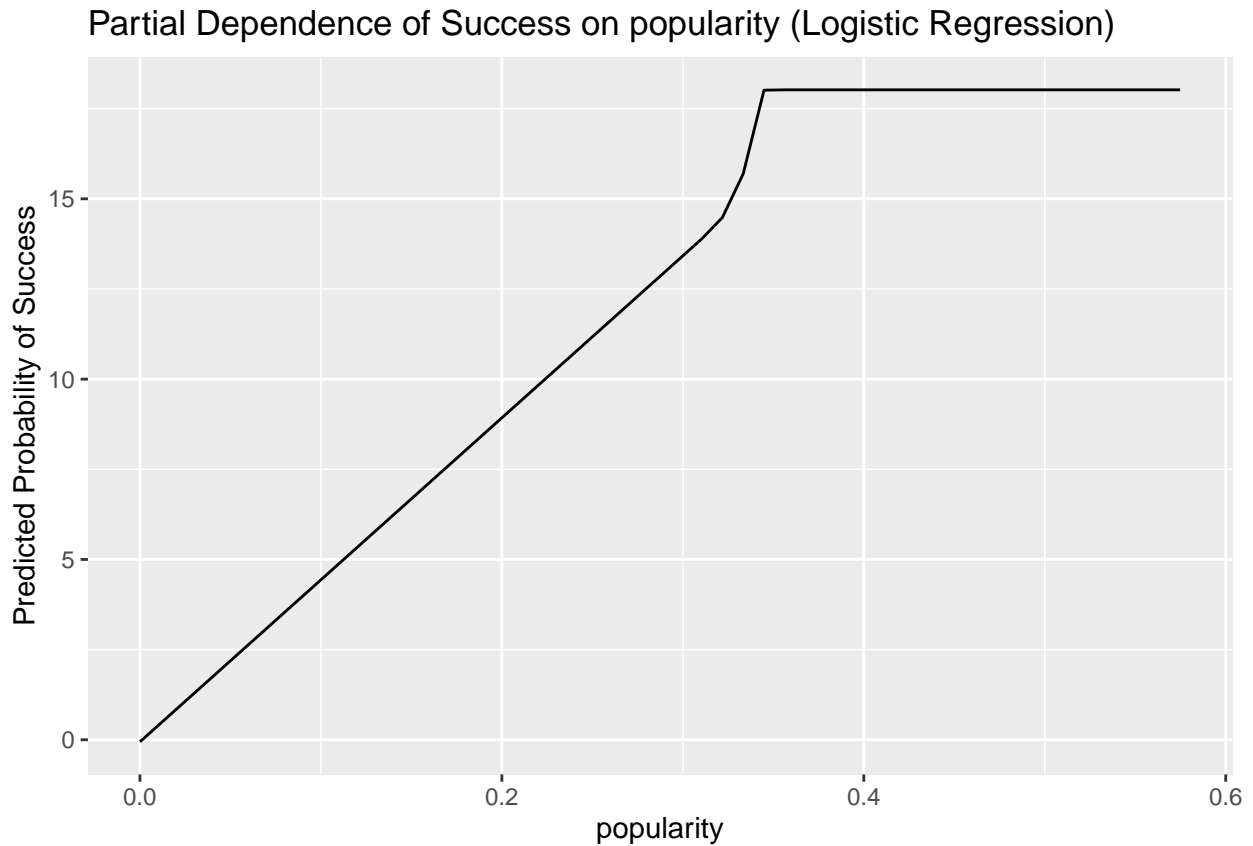
## [1] "Accuracy:  65.1481 %"
```

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.

## PDP

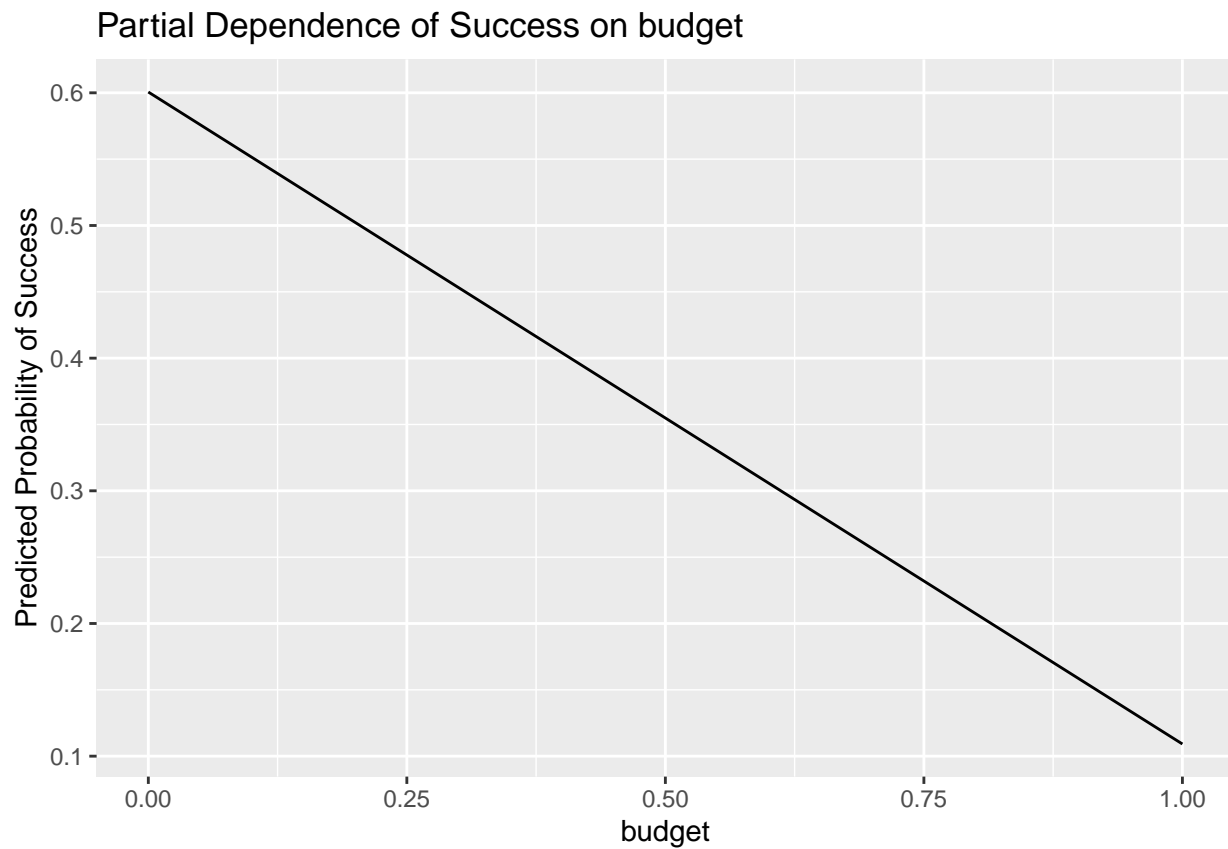
```
pdp_budget_lg <- pdp::partial(lg_model, pred.var = "popularity")

ggplot(pdp_budget_lg, aes(x = popularity, y = yhat)) +
  geom_line() +
  ggtitle("Partial Dependence of Success on popularity (Logistic Regression)") +
  xlab("popularity") +
  ylab("Predicted Probability of Success")
```



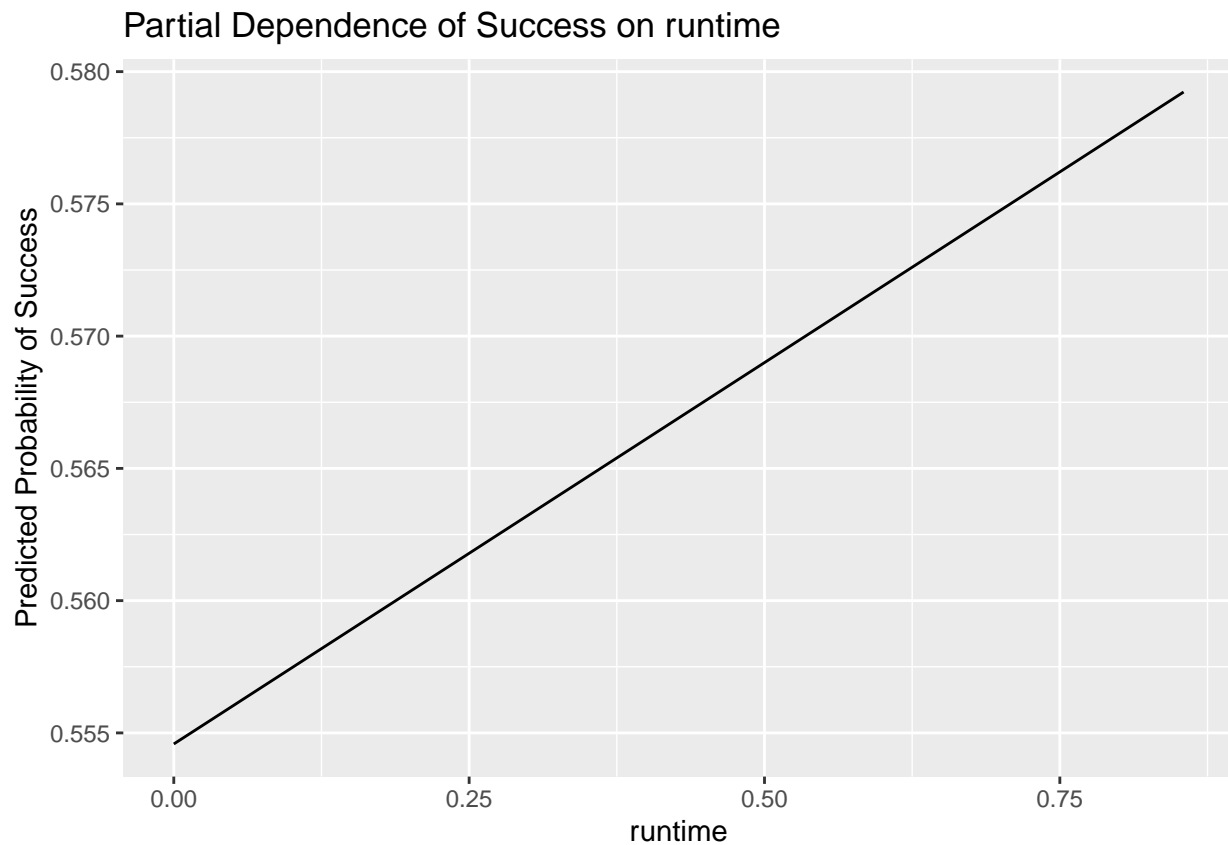
```
pdp_budget_lg2 <- pdp::partial(lg_model, pred.var = "budget")

ggplot(pdp_budget_lg2, aes(x = budget, y = yhat)) +
  geom_line() +
  ggtitle("Partial Dependence of Success on budget") +
  xlab("budget") +
  ylab("Predicted Probability of Success")
```



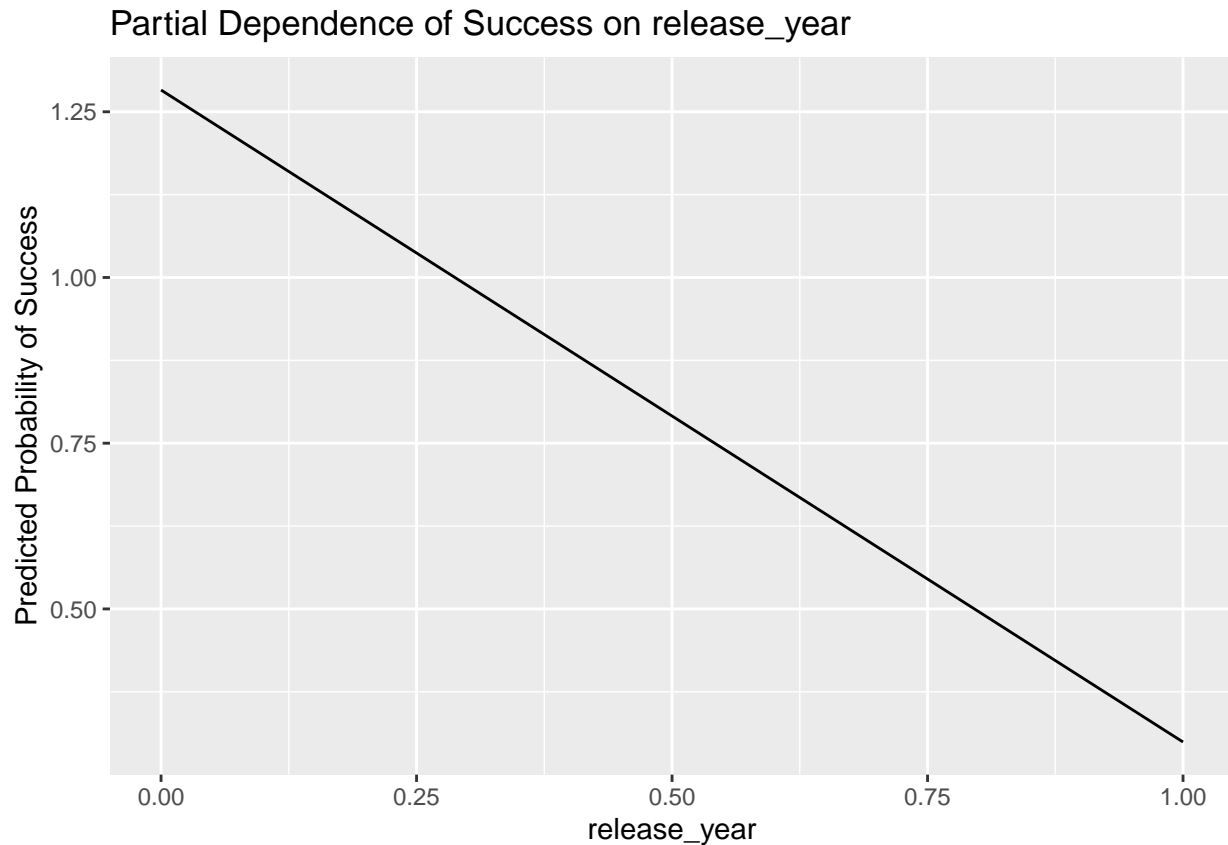
```
pdp_budget_lg3 <- pdp::partial(lg_model, pred.var = "runtime")

ggplot(pdp_budget_lg3, aes(x = runtime, y = yhat)) +
  geom_line() +
  ggtitle("Partial Dependence of Success on runtime") +
  xlab("runtime") +
  ylab("Predicted Probability of Success")
```



```
pdp_budget_lg4 <- pdp::partial(lg_model, pred.var = "release_year")

ggplot(pdp_budget_lg4, aes(x = release_year, y = yhat)) +
  geom_line() +
  ggtitle("Partial Dependence of Success on release_year") +
  xlab("release_year") +
  ylab("Predicted Probability of Success")
```



Classifying Budget into four predicted groups with multinomial logistic regression:

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

Preparing target

```
budget_quartiles <- quantile(training_n$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
training_n$budget_category <- cut(
  training_n$budget,
  breaks = budget_quartiles,
  labels = c("Low", "Medium", "High", "Very High"),
  include.lowest = TRUE
)

table(training_n$budget_category)
```

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



```
##          173          161          167          158
```

### Fitting the model

```
training_n$budget_category <- as.factor(training_n$budget_category)

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

## # weights: 20 (12 variable)
## initial value 913.567984
## iter 10 value 799.926103
## iter 10 value 799.926103
## iter 20 value 790.287990
## iter 30 value 789.269648
## final value 789.267820
## converged
```

### Check the model summary

```
summary(multinom_model)

## Call:
## multinom(formula = budget_category ~ revenue + release_year +
##   popularity, data = training_n)
##
## Coefficients:
##           (Intercept)          revenue release_year popularity
## Medium      -0.2282491  3.319914e-08    -0.3397700  0.003404545
## High        -0.6738019  5.083066e-08    -0.2738802  0.014257640
## Very High   -1.1782284  6.173187e-08    -0.5745488  0.026366845
##
## Std. Errors:
##           (Intercept)          revenue release_year popularity
## Medium    2.204654e-16  6.207521e-09  1.501721e-16  3.472101e-18
## High      2.082177e-16  5.953165e-09  1.438903e-16  3.366093e-18
## Very High 1.549093e-16  5.959676e-09  1.077308e-16  2.733015e-18
##
## Residual Deviance: 1578.536
## AIC: 1602.536
```

### Predict classifications

```
predicted_categories = predict(multinom_model, newdata = testing_n)

category_counts = table(predicted_categories)
print(category_counts)

## predicted_categories
##      Low      Medium      High Very High
##      239       41       81       78
```

## Evaluating model with confusion matrix and calculating accuracy

```
table(Predicted = predicted_categories, Actual = testing_n$budget_category)
```

```
##           Actual
## Predicted  High Low Medium
##    Low           0 199   40
##    Medium        0  26   15
##    High          6  23   52
##   Very High     17  20   41
```

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

```
## Accuracy: 50.1139 %
```

## Classification with Emphasis on Interpretation

### Classifying Success with Decision Trees

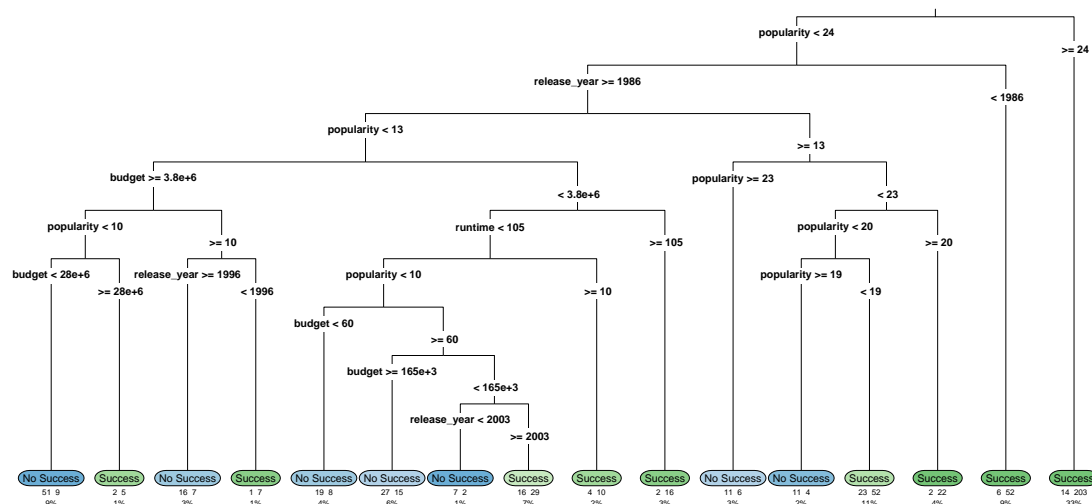
The decision tree structure shows how features like budget and popularity split the data to classify movies. The path from root to leaf highlights the decision rules. This easily identifies the most important features based on the splits.

### Fit the Model

```
tree_model <- rpart(success ~ budget + release_year + runtime + popularity, data = training, method = "glmnet")
```

### Visualize the Trees

```
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 = testing, type = "class")
```

## Evaluating the Model

Use the confusion matrix to evaluate the model.

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

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

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

```
print(tree_confusion)
```

```
##           Actual
## Predicted   No Success Success
## No Success      57      44
## Success        96     242
```

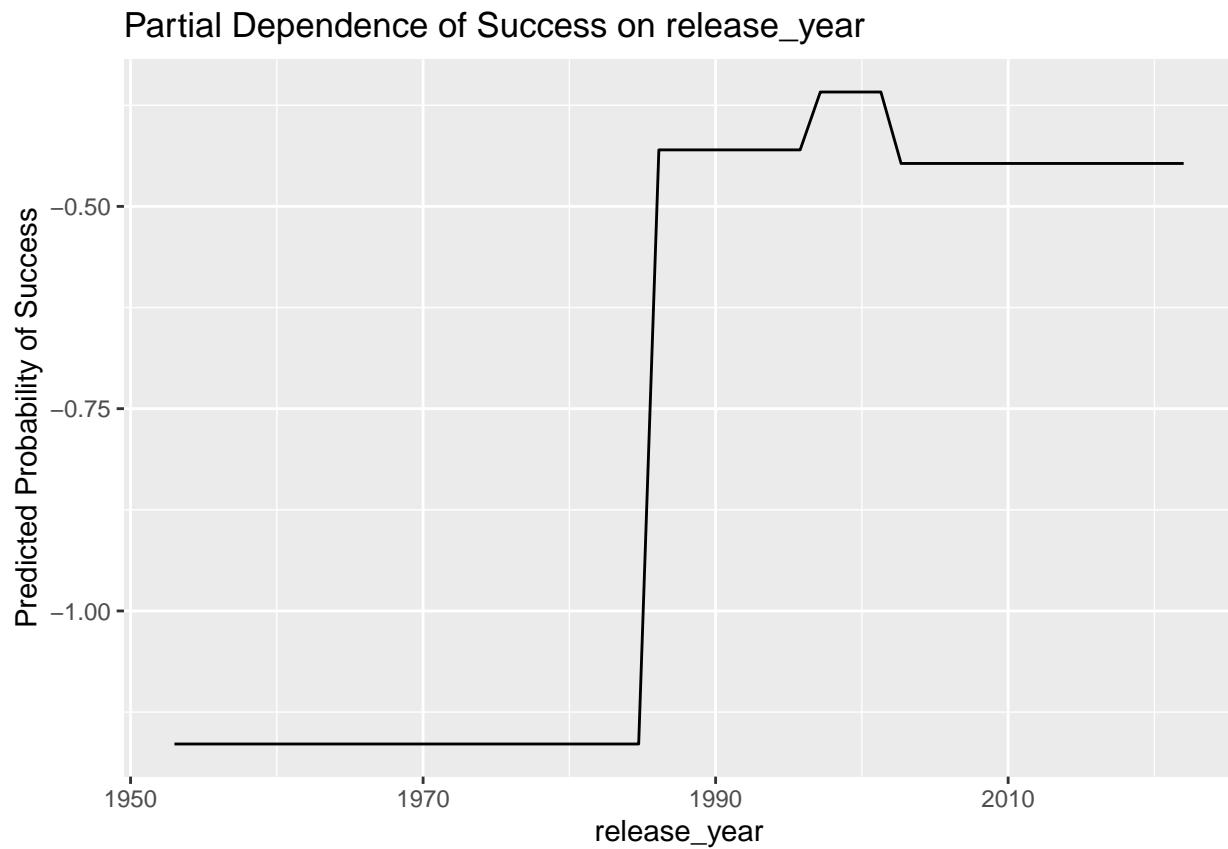
```
cat("Decision Tree Accuracy: ", round(accuracy_tree * 100, 4), "%\n")
```

```
## Decision Tree Accuracy: 68.1093 %
```

## PDP

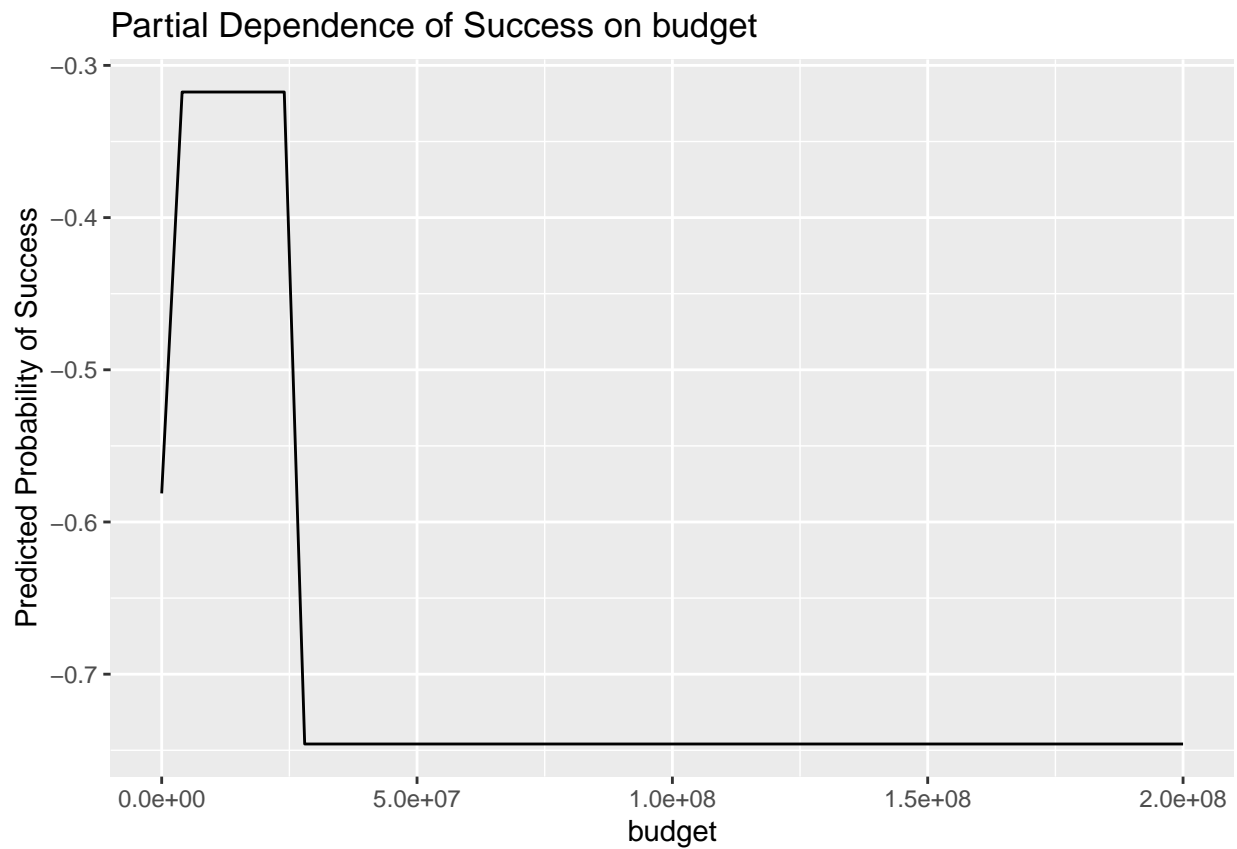
```
pdp_budget_tree <- pdp::partial(tree_model, pred.var = "release_year")
```

```
ggplot(pdp_budget_tree, aes(x = release_year, y = yhat)) +
  geom_line() +
  ggtitle("Partial Dependence of Success on release_year") +
  xlab("release_year") +
  ylab("Predicted Probability of Success")
```



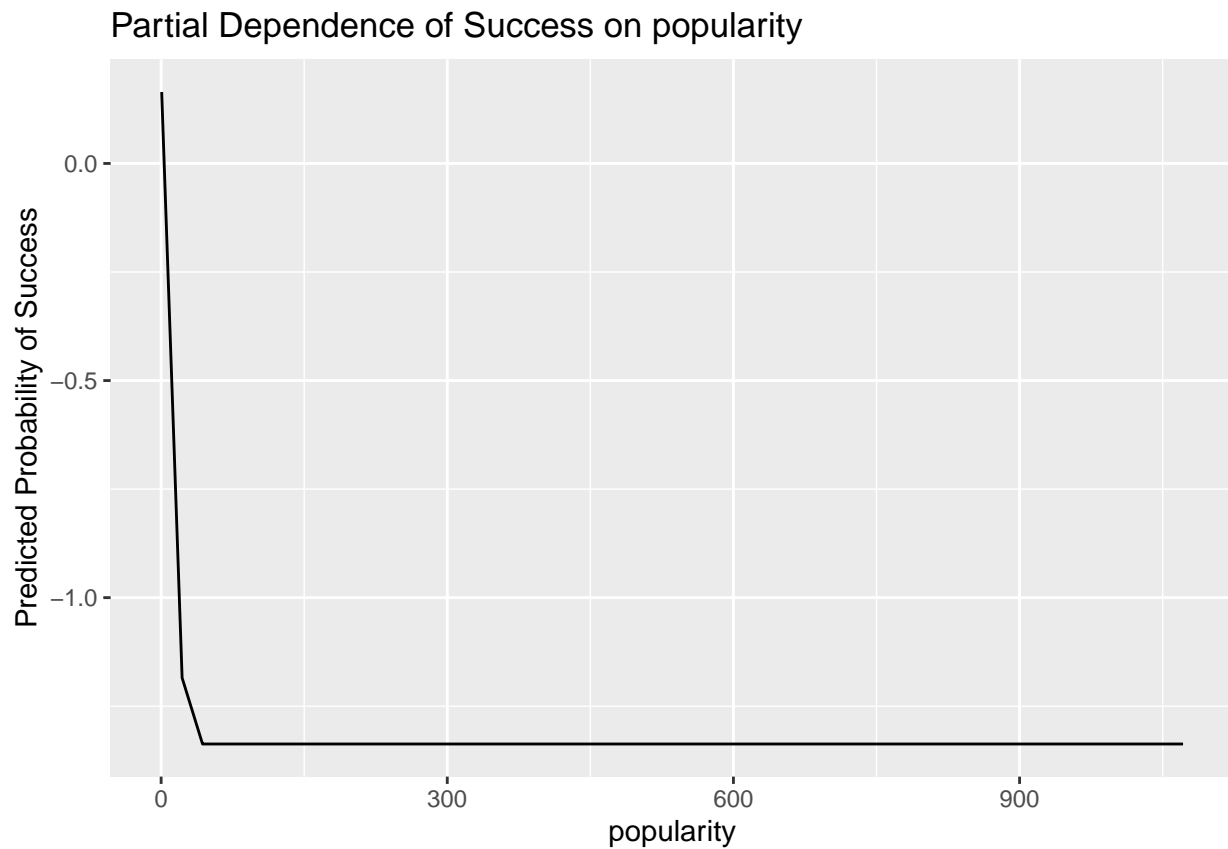
```
pdp_budget_tree2 <- pdp::partial(tree_model, pred.var = "budget")

ggplot(pdp_budget_tree2, aes(x = budget, y = yhat)) +
  geom_line() +
  ggtitle("Partial Dependence of Success on budget") +
  xlab("budget") +
  ylab("Predicted Probability of Success")
```



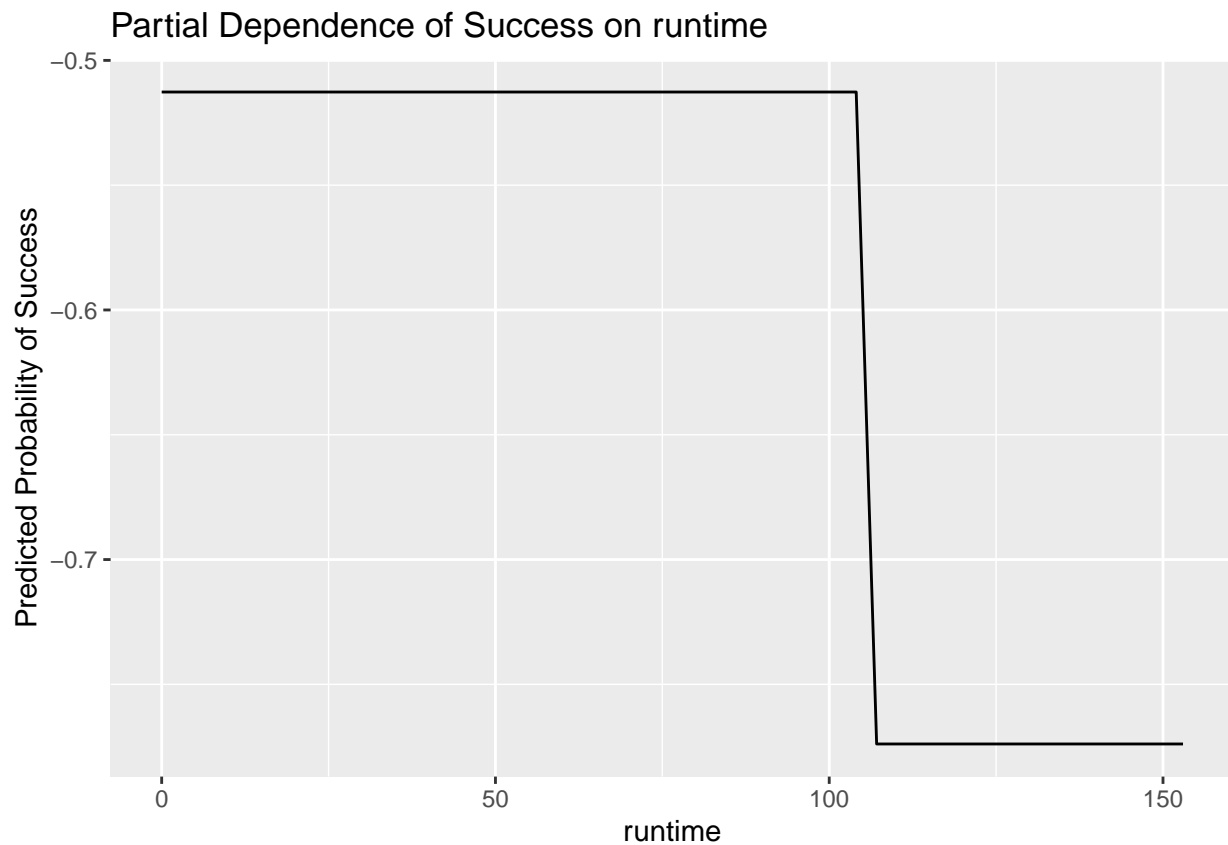
```
pdp_budget_tree3 <- pdp::partial(tree_model, pred.var = "popularity")

ggplot(pdp_budget_tree3, aes(x = popularity, y = yhat)) +
  geom_line() +
  ggtitle("Partial Dependence of Success on popularity") +
  xlab("popularity") +
  ylab("Predicted Probability of Success")
```



```
pdp_budget_tree4 <- pdp::partial(tree_model, pred.var = "runtime")

ggplot(pdp_budget_tree4, aes(x = runtime, y = yhat)) +
  geom_line() +
  ggtitle("Partial Dependence of Success on runtime") +
  xlab("runtime") +
  ylab("Predicted Probability of Success")
```



## Classifying Success with Random Forest (with Feature Importance)

### Fit the Model

```
library(randomForest)

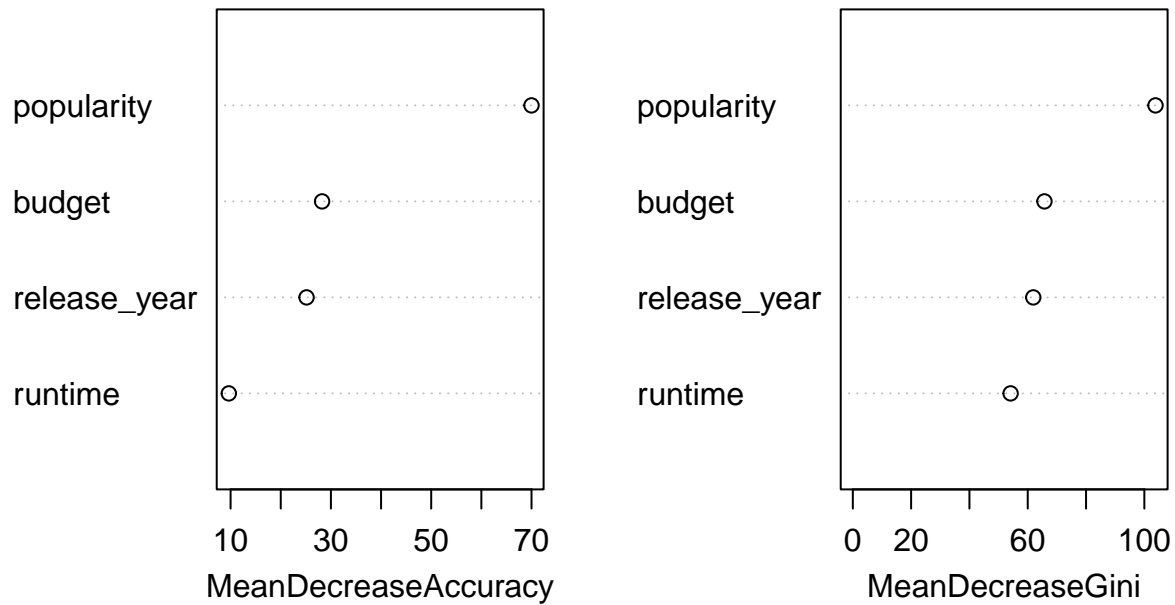
rf_model <- randomForest(success ~ budget + release_year + runtime + popularity, data = training, impor
```

### View Variable Importance

This shows which features contribute most to the model.

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

## rf\_model



### Prediction

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

### Evaluating the Model

Use a confusion matrix to evaluate the model.

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

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

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

```
print(rf_confusion)
```

```
##           Actual
## Predicted  No Success Success
## No Success      57      41
## Success        96     245
```

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

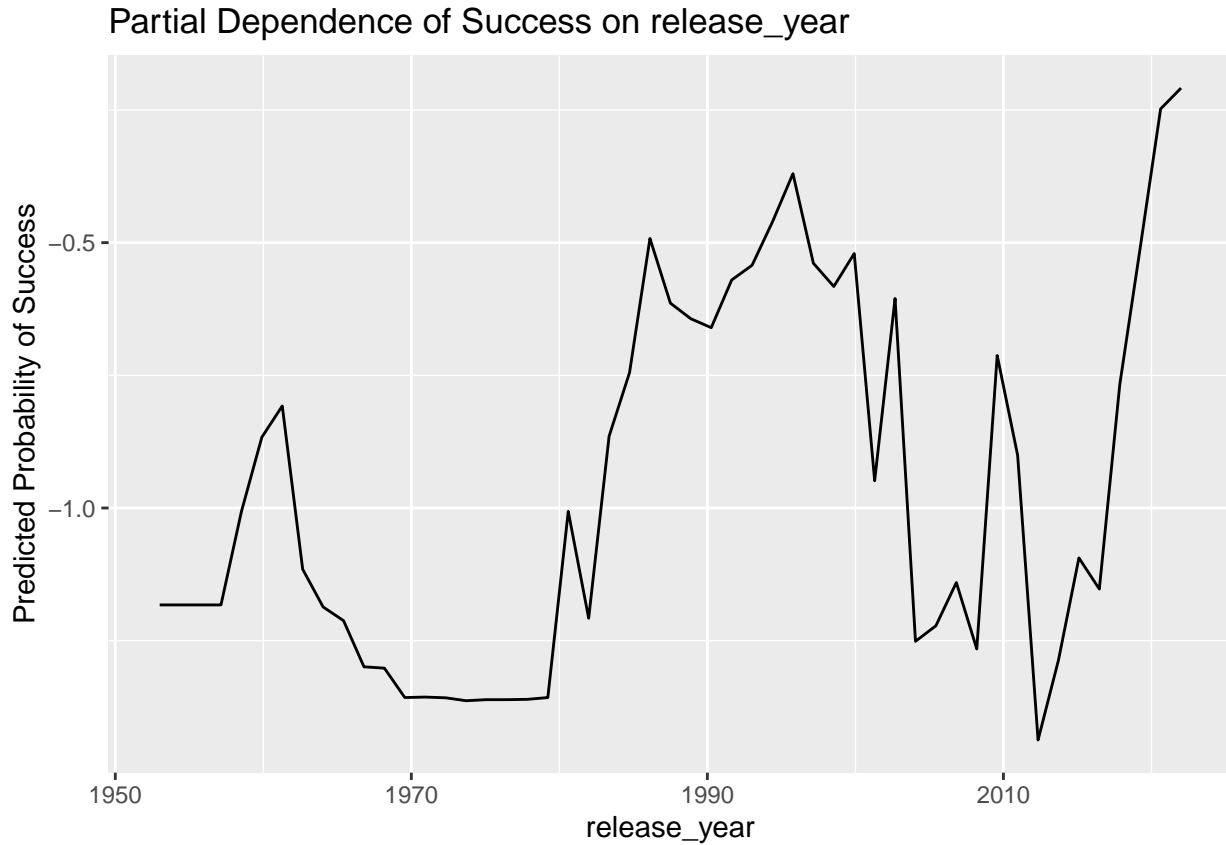
```
## Random Forest Accuracy:  68.7927 %
```

### PDP

```
pdp_budget_rf <- pdp::partial(rf_model, pred.var = "release_year")
```



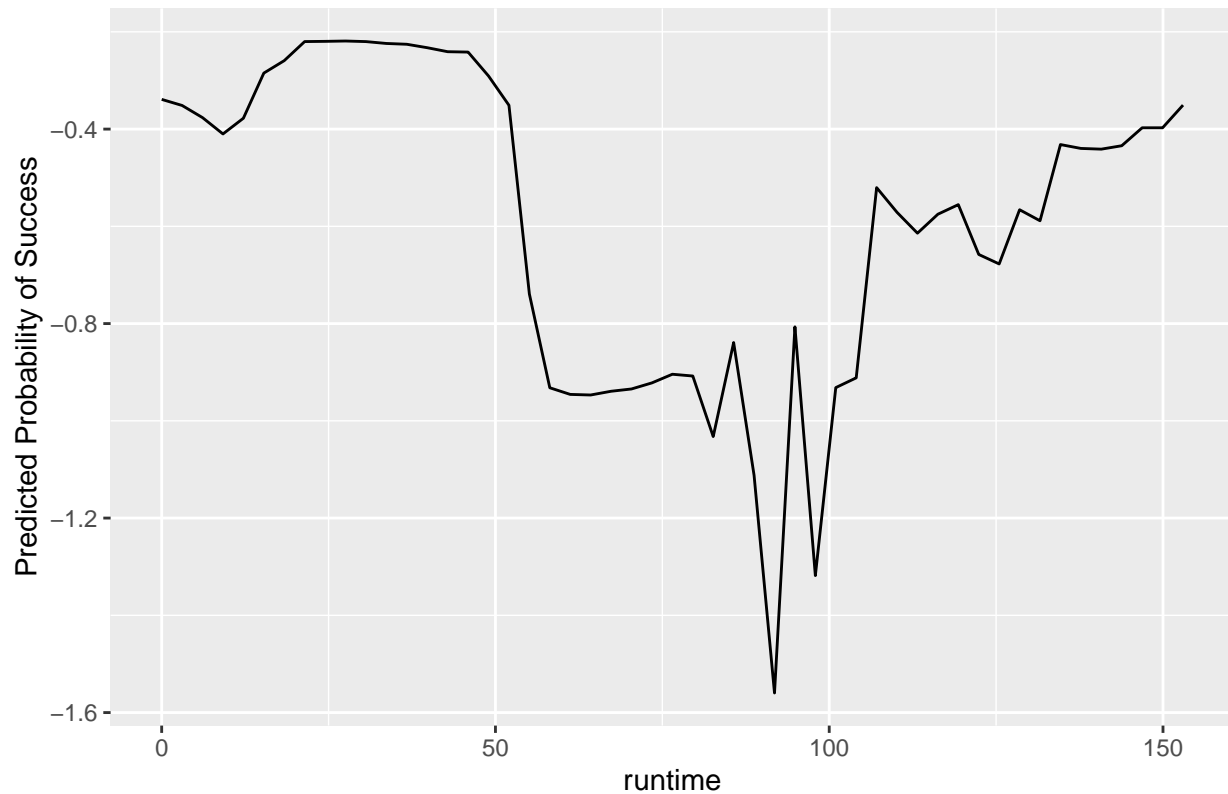
```
ggplot(pdp_budget_rf, aes(x = release_year, y = yhat)) +
  geom_line() +
  ggtitle("Partial Dependence of Success on release_year") +
  xlab("release_year") +
  ylab("Predicted Probability of Success")
```



```
pdp_budget_rf2 <- pdp::partial(rf_model, pred.var = "runtime")

ggplot(pdp_budget_rf2, aes(x = runtime, y = yhat)) +
  geom_line() +
  ggtitle("Partial Dependence of Success on runtime") +
  xlab("runtime") +
  ylab("Predicted Probability of Success")
```

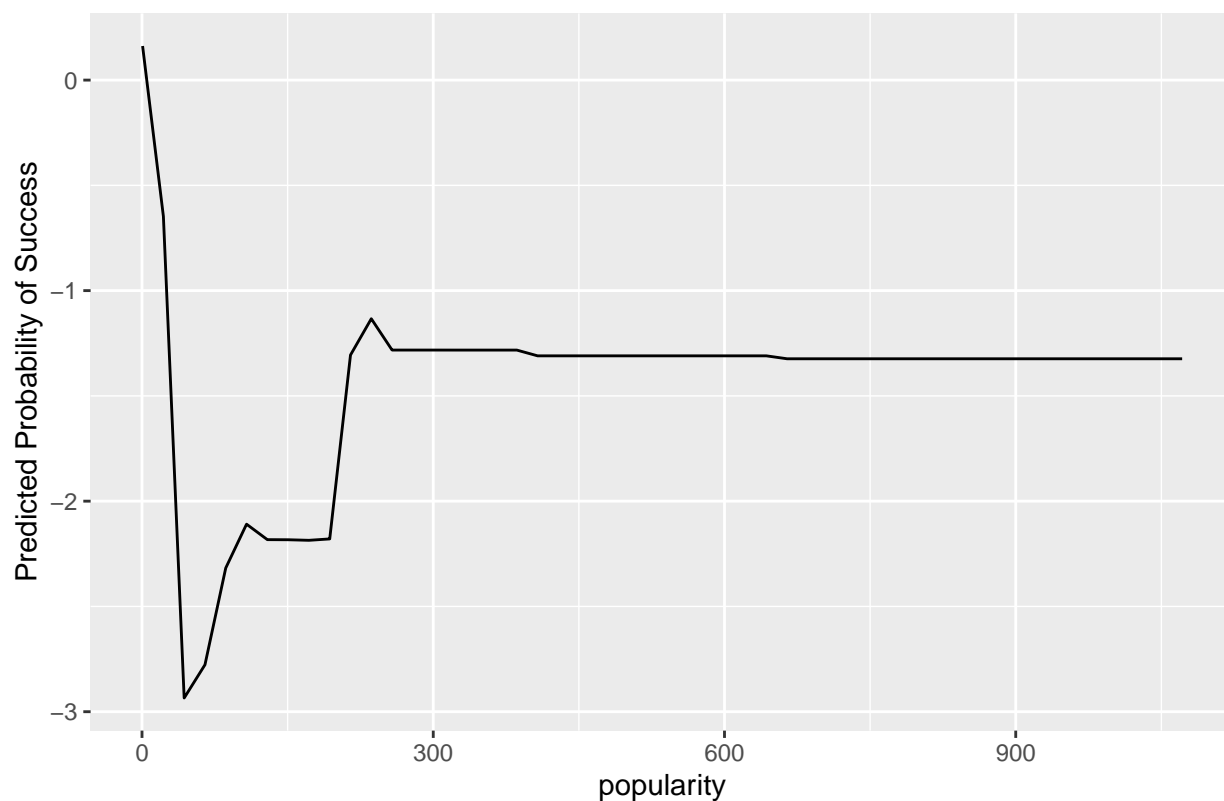
Partial Dependence of Success on runtime



```
pdp_budget_rf3 <- pdp::partial(rf_model, pred.var = "popularity")

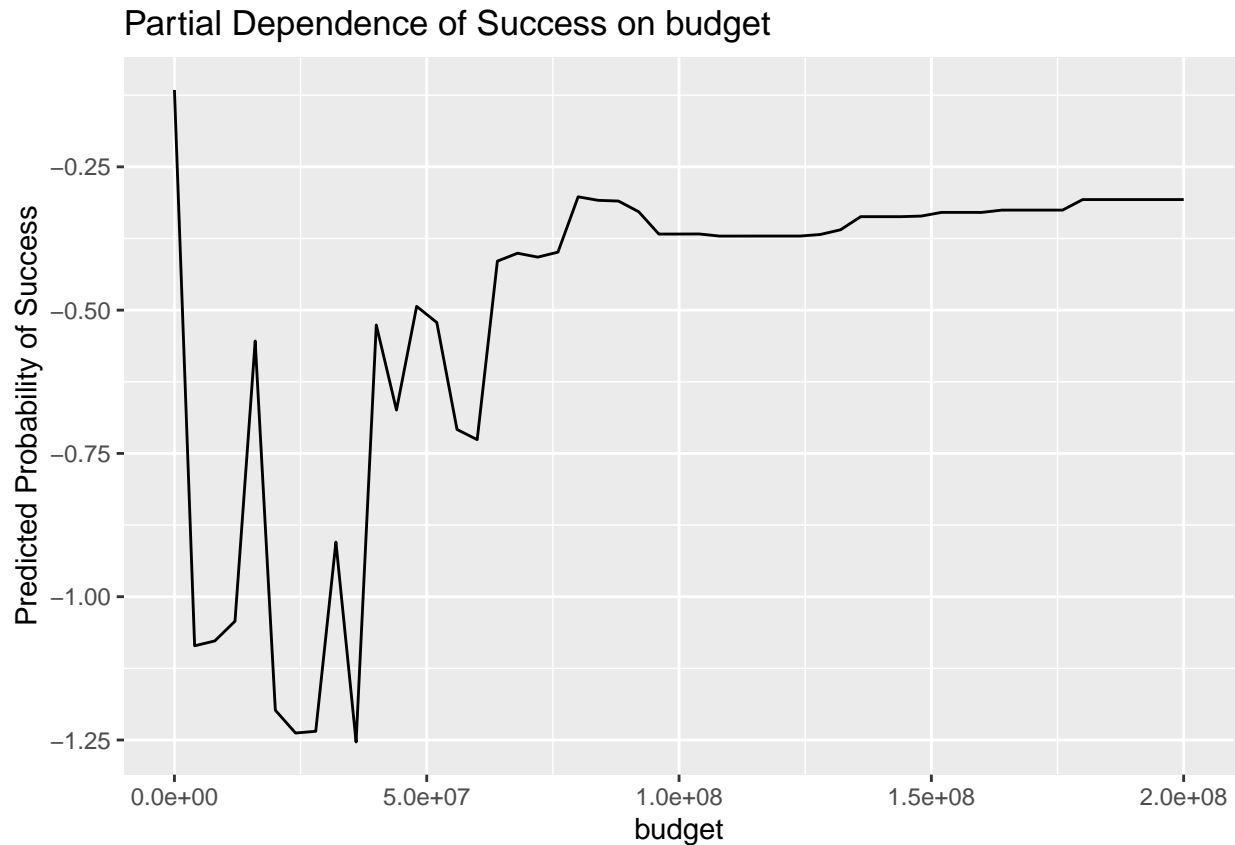
ggplot(pdp_budget_rf3, aes(x = popularity, y = yhat)) +
  geom_line() +
  ggtitle("Partial Dependence of Success on popularity") +
  xlab("popularity") +
  ylab("Predicted Probability of Success")
```

Partial Dependence of Success on popularity



```
pdp_budget_rf4 <- pdp::partial(rf_model, pred.var = "budget")

ggplot(pdp_budget_rf4, aes(x = budget, y = yhat)) +
  geom_line() +
  ggtitle("Partial Dependence of Success on budget") +
  xlab("budget") +
  ylab("Predicted Probability of Success")
```



## 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

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

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

```
## # weights: 31
## initial value 677.159206
## iter 10 value 532.657456
## iter 20 value 502.388653
## iter 30 value 492.785405
## iter 40 value 487.961334
## iter 50 value 485.487925
## iter 60 value 484.018116
```

```
## iter 70 value 482.830700
## iter 80 value 479.955217
## iter 90 value 479.556221
## iter 100 value 479.419115
## iter 110 value 479.329064
## iter 120 value 479.318553
## final value 479.317125
## converged

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
## -1.65 7.13 7.40 -1.76 0.47
## b->h2 i1->h2 i2->h2 i3->h2 i4->h2
## 0.24 0.03 -0.02 0.31 0.19
## b->h3 i1->h3 i2->h3 i3->h3 i4->h3
## -5.54 -1.22 4.72 1.61 10.24
## b->h4 i1->h4 i2->h4 i3->h4 i4->h4
## -2.48 0.09 2.97 1.52 27.35
## b->h5 i1->h5 i2->h5 i3->h5 i4->h5
## -5.37 2.02 5.78 5.31 0.61
## b->o h1->o h2->o h3->o h4->o h5->o
## 0.73 -8.41 0.24 -8.35 22.40 -6.59
```

## Prediction

```
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      32      25
## Success        41     121
```

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

```
## [1] "Accuracy: 69.863 %"
```

## 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.

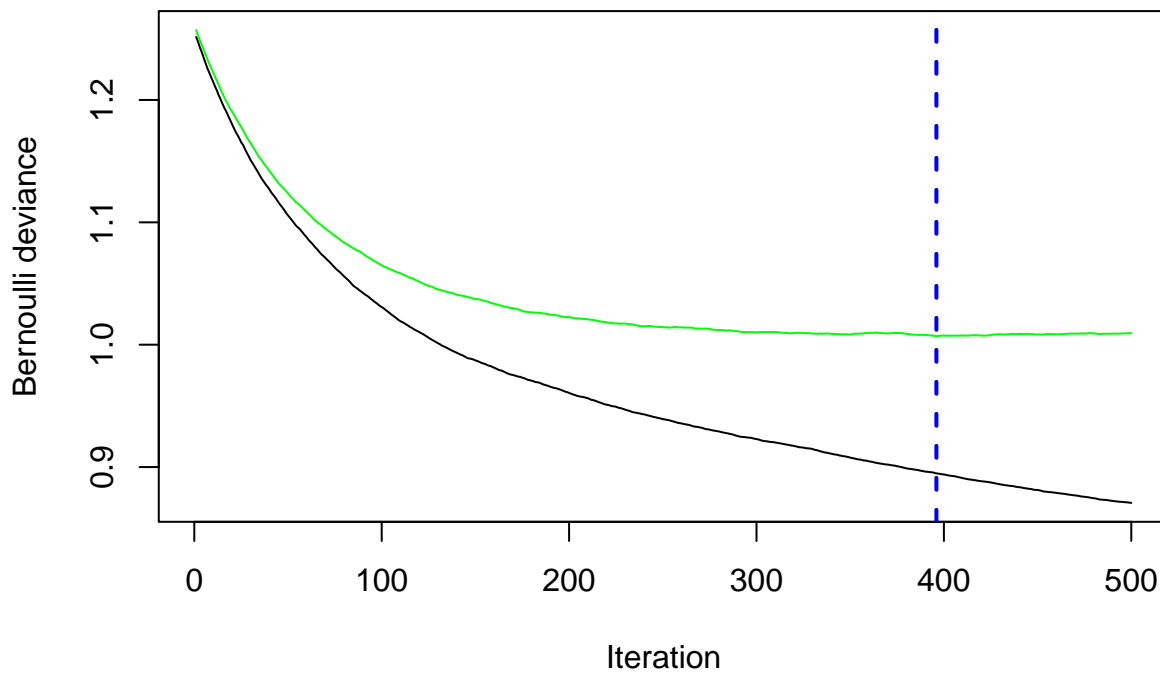
```
training$success <- ifelse(training$success == "Success", 1, 0)
```

```
set.seed(123)
gbm_model <- gbm(success ~ budget + release_year + runtime + popularity,
  data = training,
  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 = testing, 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 = testing$success)
print(confusion_matrix_gbm)
```

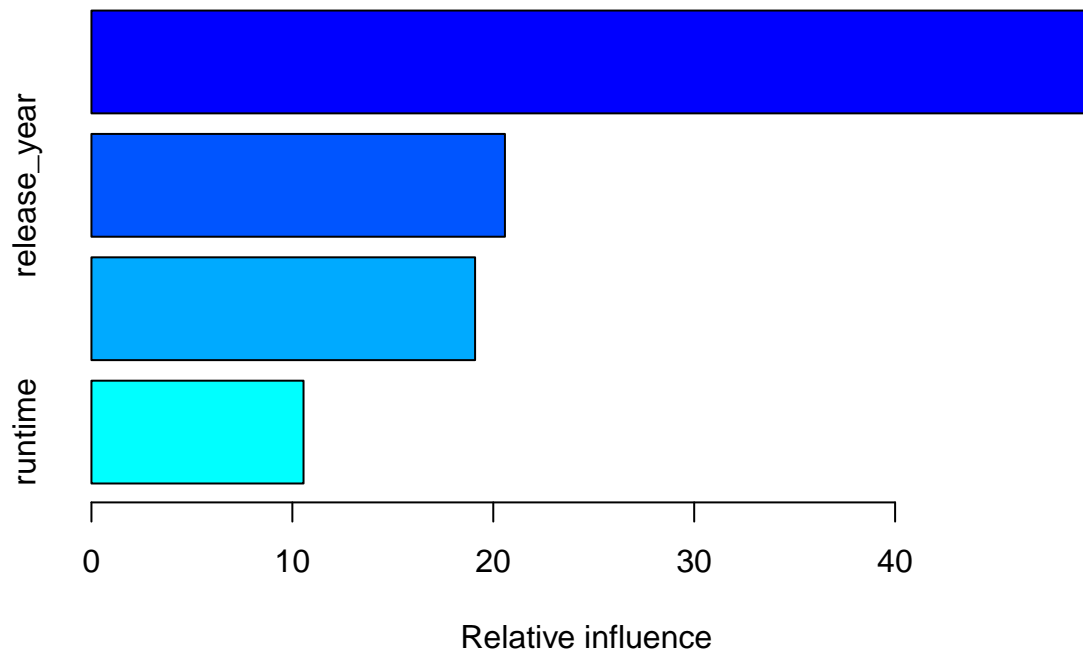
```
##           Actual
## Predicted   No Success Success
## No Success      54       45
## Success        99      241
```

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

```
## [1] "Accuracy: 67.1982 %"
```

## Feature Importance

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



```
print(importance)
```

```
##           var  rel.inf
## popularity  popularity 49.77012
## release_year release_year 20.57941
## budget      budget 19.09468
## runtime     runtime 10.55579
```

**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 boundaries.

## Feature Selection and Comparison of Predictor Sets

### Feature Importance Analysis

#### Correlation Analysis for Continuous Variables

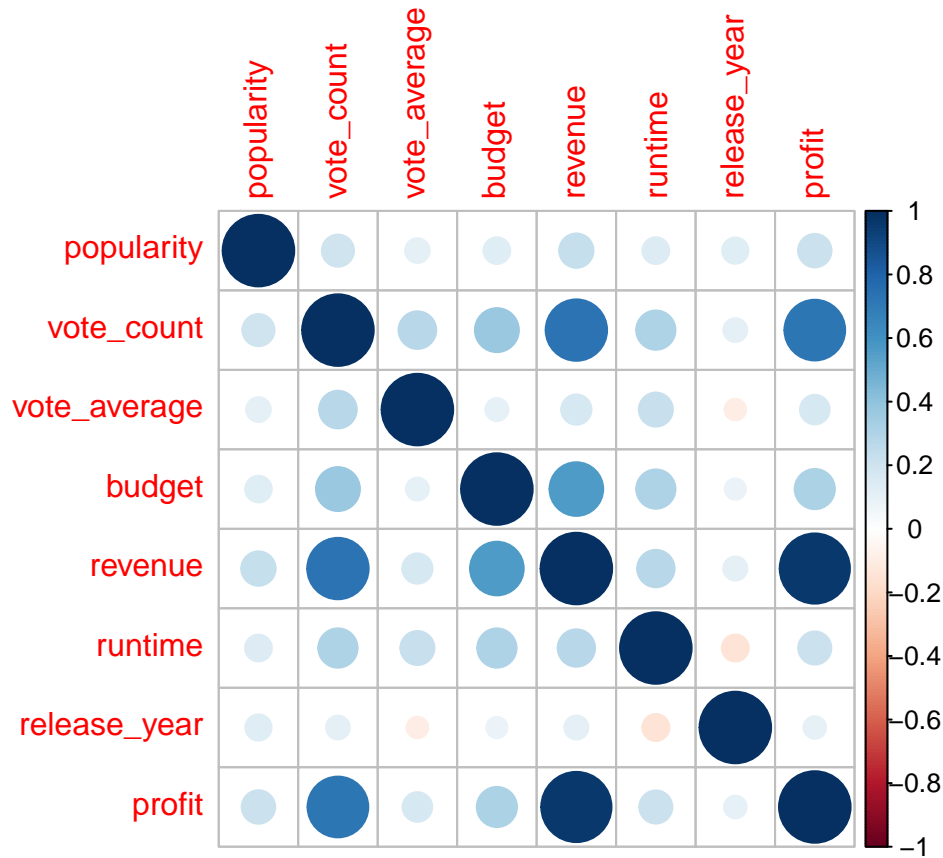
We will analyze the correlation between predictors and the target variable, success. We will identify multicollinearity among predictors to avoid redundancy. We will do so by looking at a correlation matrix for numeric predictors.

```
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  $\pm 1$ ) between a predictor and **success** suggest high relevance.
- Avoid using highly correlated predictors simultaneously to prevent redundancy.

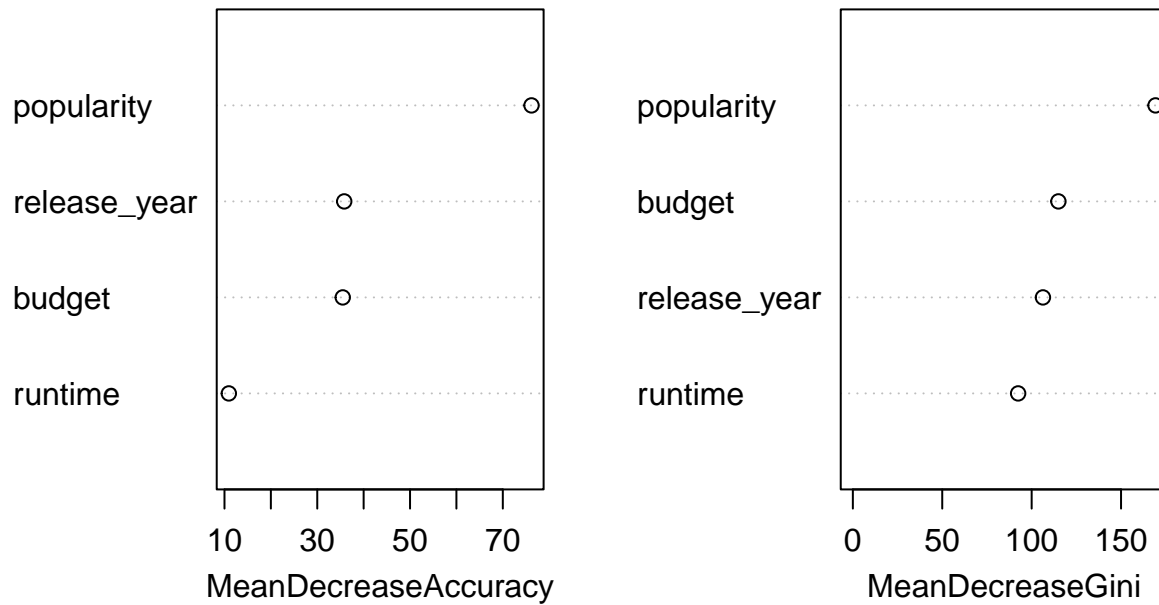
### Variable Importance from Random Forest

Random Forest provides a direct measure of feature importance.

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



## rf\_model

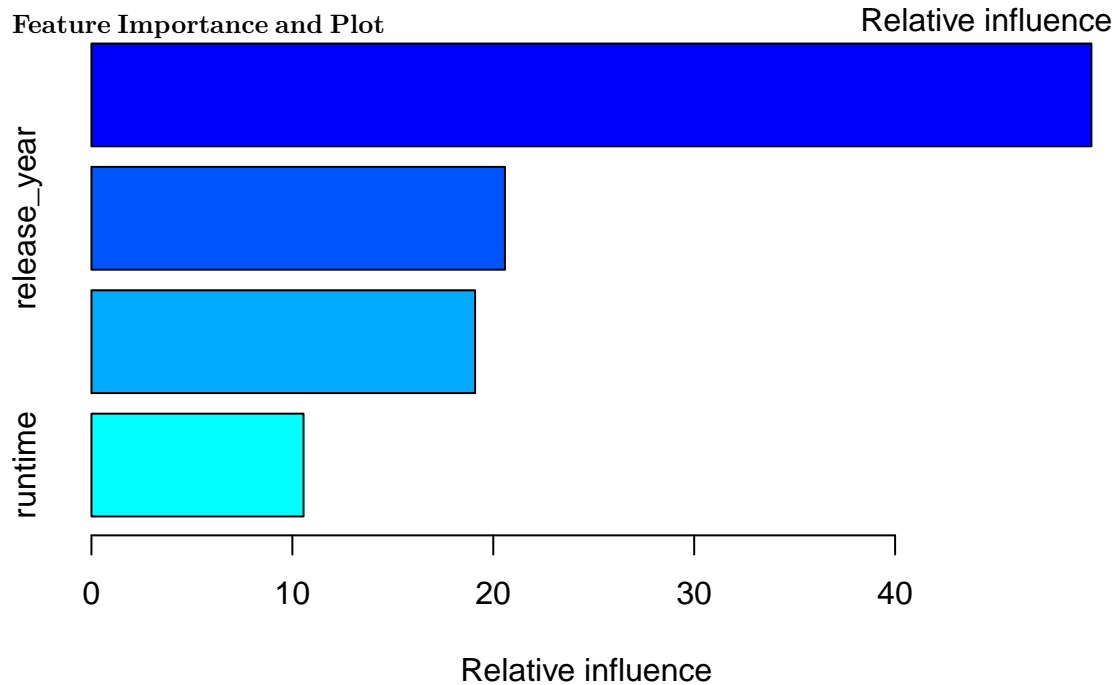
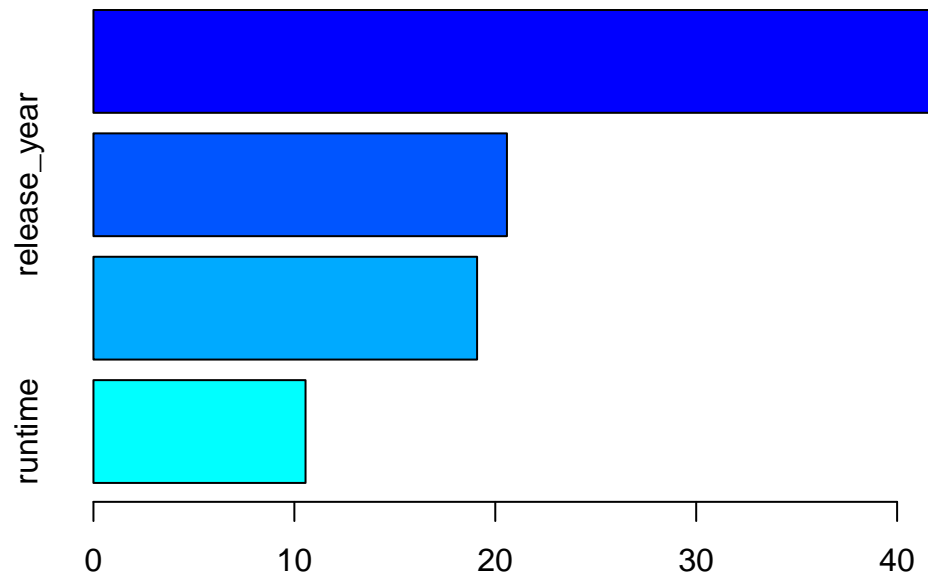


Interpretation: (ADD with context)

- 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.

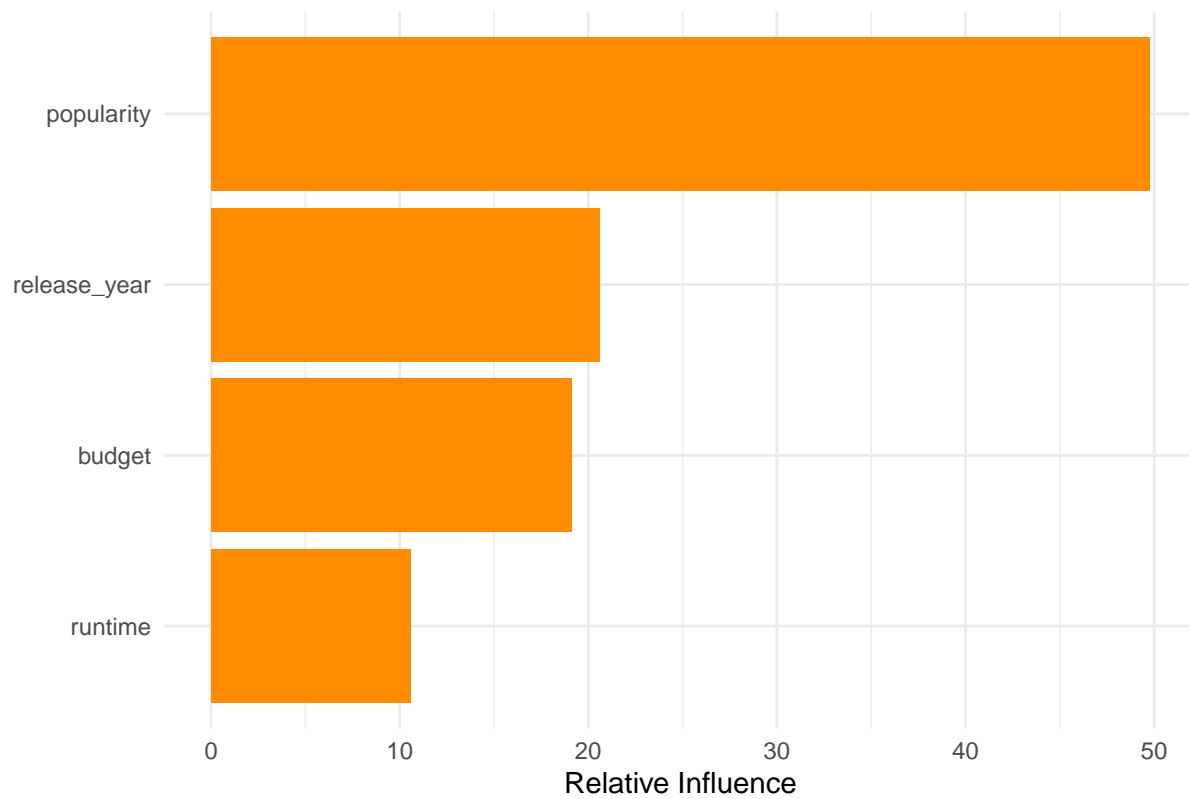
### Gradient Boosting Feature Importance and Partial Dependence

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



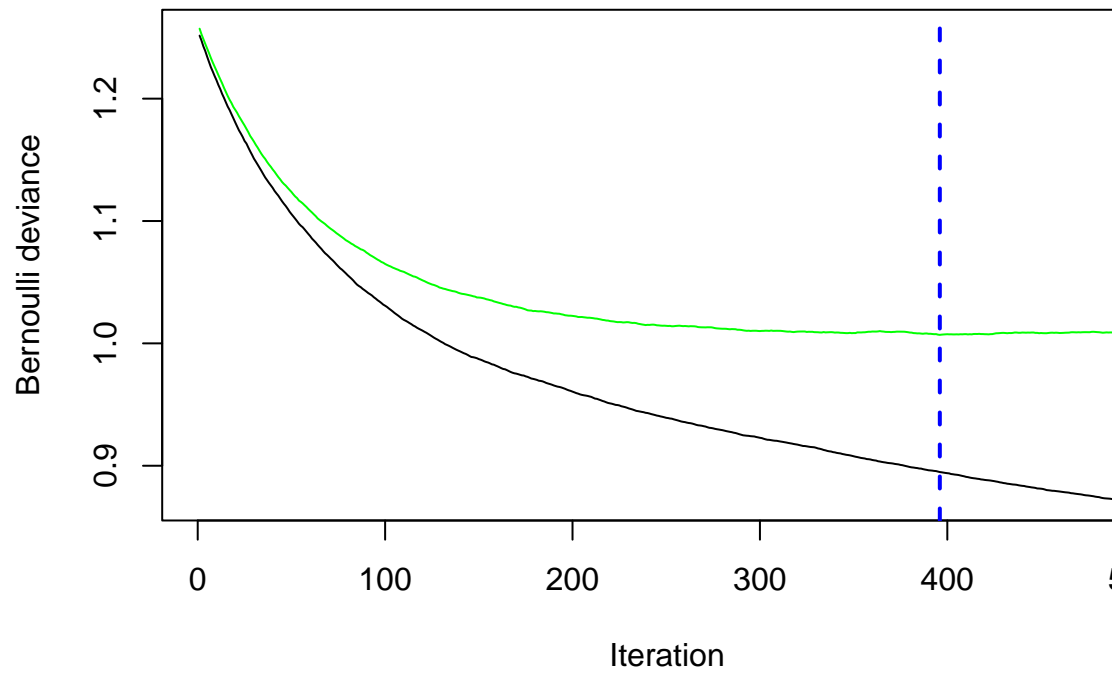
```
# Plot Feature Importances
ggplot(importance_df_gbm, aes(x = reorder(Feature, Importance), y = Importance)) +
  geom_bar(stat = "identity", fill = "darkorange") +
  coord_flip() +
  ggtitle("Feature Importances from Gradient Boosting") +
  xlab("") +
  ylab("Relative Influence") +
  theme_minimal()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

## Feature Importances from Gradient Boosting



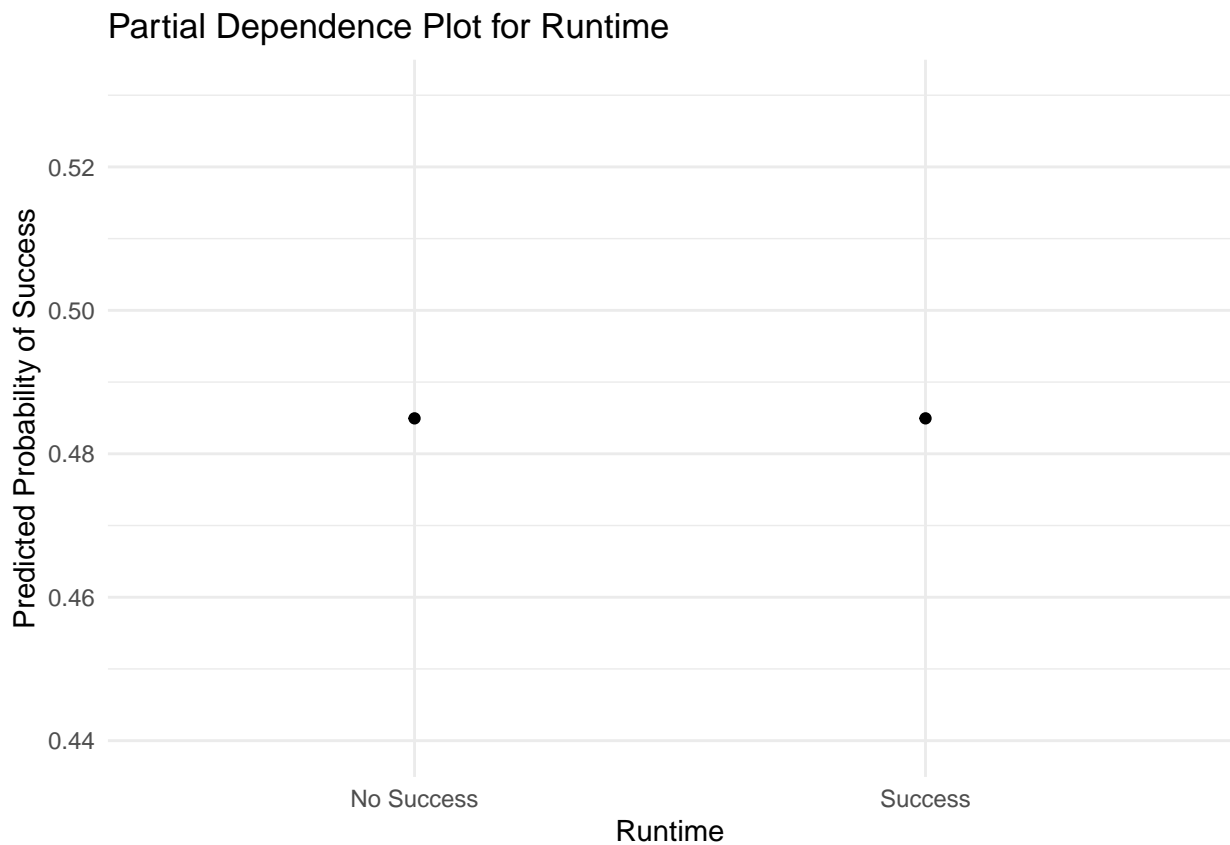
ADD Interpretations

```
pdp_success <- partial(gbm_model,
  pred.var = "success",
  n.trees = gbm.perf(gbm_model, method = "cv"),
  train = trainData) # Ensure the training data is passed
```



Partial Dependence Plot

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



## Stepwise Selection

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

```
full_model <- glm(success ~ budget + release_year + runtime + popularity, data = df_for_class, family =  
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred  
step_model <- stepAIC(full_model, direction = "both")  
  
## Start:  AIC=1313.79  
## success ~ budget + release_year + runtime + popularity  
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred  
  
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred  
##  
##           Df Deviance    AIC  
## - budget      1   1303.8 1311.8  
## <none>         1303.8 1313.8  
## - runtime      1   1306.9 1314.9  
## - release_year 1   1338.3 1346.3  
## - popularity   1   1348.3 1356.3  
  
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred  
##  
## Step:  AIC=1311.8  
## success ~ release_year + runtime + popularity  
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred  
  
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred  
  
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred  
##  
##           Df Deviance    AIC  
## <none>         1303.8 1311.8  
## - runtime      1   1307.0 1313.0  
## + budget       1   1303.8 1313.8  
## - release_year 1   1338.5 1344.5  
## - popularity   1   1350.1 1356.1  
  
summary(step_model)  
  
##  
## Call:  
## glm(formula = success ~ release_year + runtime + popularity,  
##      family = binomial, data = df_for_class)  
##  
## Coefficients:  
##           Estimate Std. Error z value Pr(>|z|)  
## (Intercept)  57.087853  10.267083   5.560 2.69e-08 ***  
## release_year -0.028542   0.005102  -5.595 2.21e-08 ***  
## runtime      0.004466   0.002492   1.792  0.0731 .  
## popularity   0.017716   0.003308   5.355 8.54e-08 ***  
## ---  
## 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 1094 degrees of freedom
## AIC: 1311.8
##
## Number of Fisher Scoring iterations: 6
```

Interpretation: (ADD with context)

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

## SHAP Values for Model Interpretability

```
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

```
# Define the prediction function for SHAP
predict_function <- function(object, newdata) {
  predict(object, newdata = newdata, type = "prob")[, "Success"] # Get probabilities for 'Success' class
}

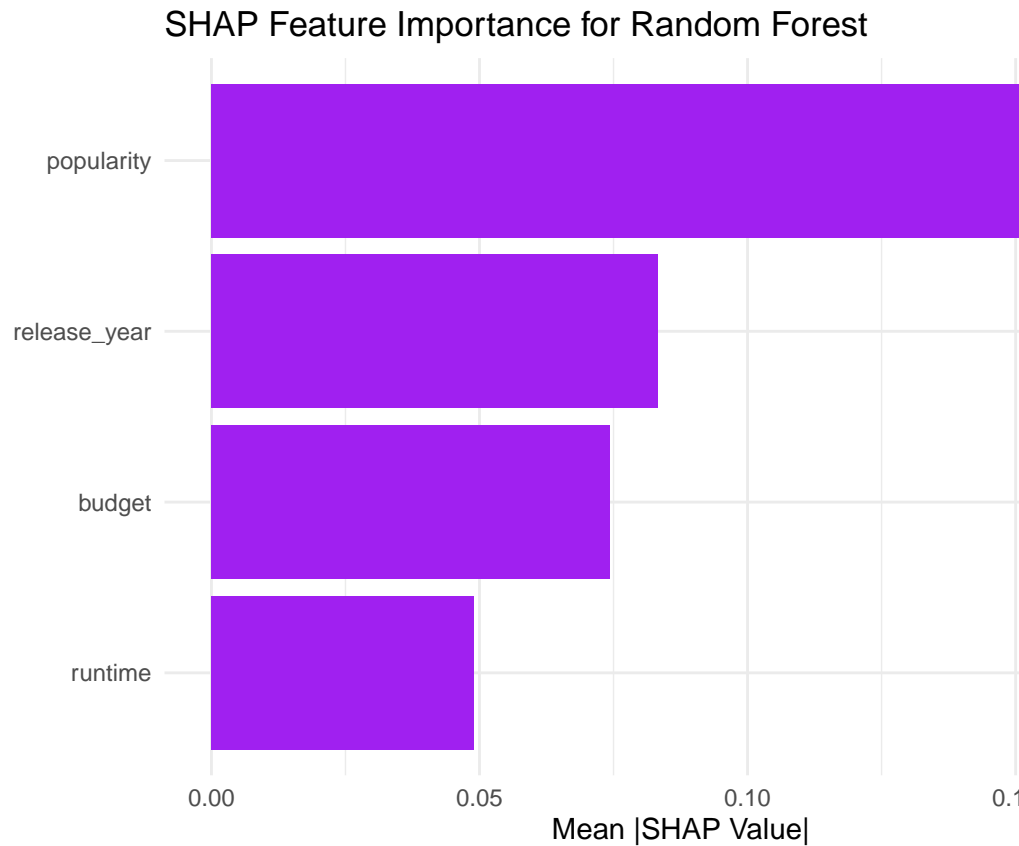
# Compute SHAP values
set.seed(123)
shap_values <- fastshap::explain(
  object = rf_model,
  X = df_for_class[, c("budget", "release_year", "runtime", "popularity")], # Use the feature columns
  pred_wrapper = predict_function,
  nsim = 50,
  adjust = TRUE
)

# 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

```
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()
```

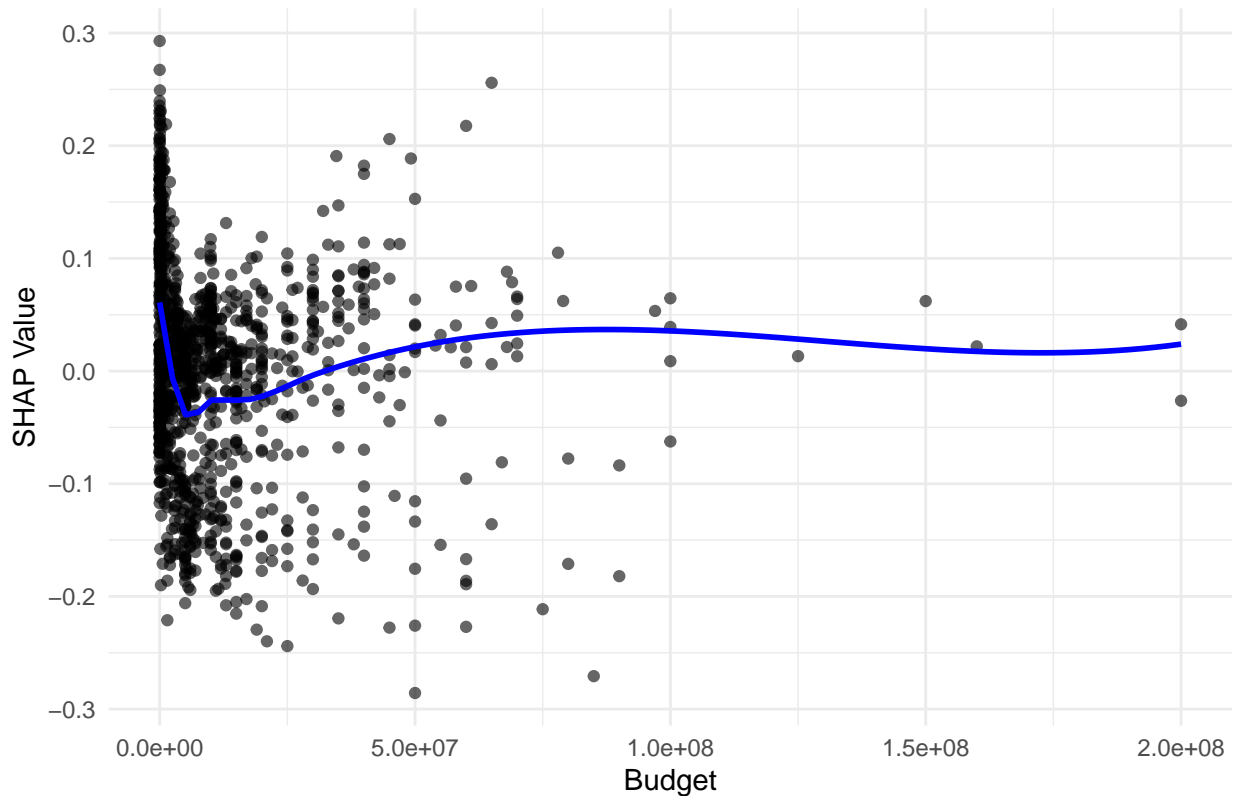


```
shap_values_budget <- shap_values[, "budget"]  
  
ggplot(data = data.frame(  
  SHAP_value = shap_values_budget,  
  Feature_value = df_for_class$budget # Original feature values for 'budget'  
) , aes(x = Feature_value, y = SHAP_value)) +  
  geom_point(alpha = 0.6) +  
  geom_smooth(method = "loess", se = FALSE, color = "blue") +  
  ggtitle("SHAP Dependence Plot for Budget") +  
  xlab("Budget") +  
  ylab("SHAP Value") +  
  theme_minimal()
```

### SHAP Dependence Plot

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

SHAP Dependence Plot for Budget



### Multiple Regression with t-values for Variable Importance

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

```
##
## Call:
## glm(formula = success ~ budget + release_year + runtime + popularity,
##      family = "binomial", data = training_lg)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   1.35118    0.50679   2.666  0.00767 **
## budget       -0.98439    1.00381  -0.981  0.32676
## release_year -1.97016    0.47477  -4.150 3.33e-05 ***
## runtime        0.05776    0.62504   0.092  0.92637
## popularity   89.83537   13.20727   6.802 1.03e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 827.89  on 658  degrees of freedom
## Residual deviance: 720.31  on 654  degrees of freedom
## AIC: 730.31
##
## Number of Fisher Scoring iterations: 6
```



```
# 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.

## Comparing Predictor Sets

### Base Set of Predictors

We will use all available predictors: budget, release\_year, runtime, and popularity.

```
# Fit logistic regression model
base_model <- glm(success ~ budget + release_year + runtime + popularity,
                  data = df_normal,
                  family = binomial)
```

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

```
# Predict probabilities and classify
base_probs <- predict(base_model, type = "response")
base_preds <- ifelse(base_probs > 0.5, 1, 0)
```

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

```
## Base Model Accuracy: 0
```

### Reduced Set of Predictors

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

```
#EXAMPLE BUT CHANGE WHEN FIND PREDICTORS TO USE
reduced_model <- glm(success ~ budget + popularity, data = df_normal, family = binomial)
```

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

```
reduced_probs <- predict(reduced_model, type = "response")
reduced_preds <- ifelse(reduced_probs > 0.5, 1, 0)
reduced_accuracy <- mean(reduced_preds == df_normal$success)
cat("Reduced Model Accuracy: ", reduced_accuracy, "\n")
```

```
## Reduced Model Accuracy: 0
```

### Comparing Models with Different Predictor Sets

```
table(df_for_class$success)
```

```
##
## No Success    Success
##           365      733
```

```
# Remove rows with missing values
df_for_class <- na.omit(df_for_class)
```

```

# 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)
  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)

## Warning in model.response(mf, "numeric"): using type = "numeric" with a factor
## response will be ignored

## Warning in Ops.factor(y, z$residuals): '-' not meaningful for factors
lm_accuracy_2 <- train_lm_model(predictor_set_2, df_for_class)

## Warning in model.response(mf, "numeric"): using type = "numeric" with a factor
## response will be ignored

## Warning in model.response(mf, "numeric"): '-' not meaningful for factors
lm_accuracy_3 <- train_lm_model(predictor_set_3, df_for_class)

## Warning in model.response(mf, "numeric"): using type = "numeric" with a factor
## response will be ignored

## Warning in model.response(mf, "numeric"): '-' not meaningful for factors
lm_accuracy_4 <- train_lm_model(predictor_set_4, df_for_class)

## Warning in model.response(mf, "numeric"): using type = "numeric" with a factor
## response will be ignored

## Warning in model.response(mf, "numeric"): '-' not meaningful for factors
# 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 (Set 4)",
            "Linear Regression (Set 1)", "Linear Regression (Set 2)", "Linear Regression (Set 3)", "Linear Regression (Set 4)"),
  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)

##           Model AUC_or_Accuracy
## 1 Random Forest (Set 1)      0.5950606
## 2 Random Forest (Set 2)      0.7651946
## 3 Random Forest (Set 3)      0.7138444
## 4 Random Forest (Set 4)      0.7731260
## 5 Linear Regression (Set 1)  0.0000000
## 6 Linear Regression (Set 2)  0.0000000
## 7 Linear Regression (Set 3)  0.0000000
## 8 Linear Regression (Set 4)  0.0000000

```

## Plotting Performance Comparison

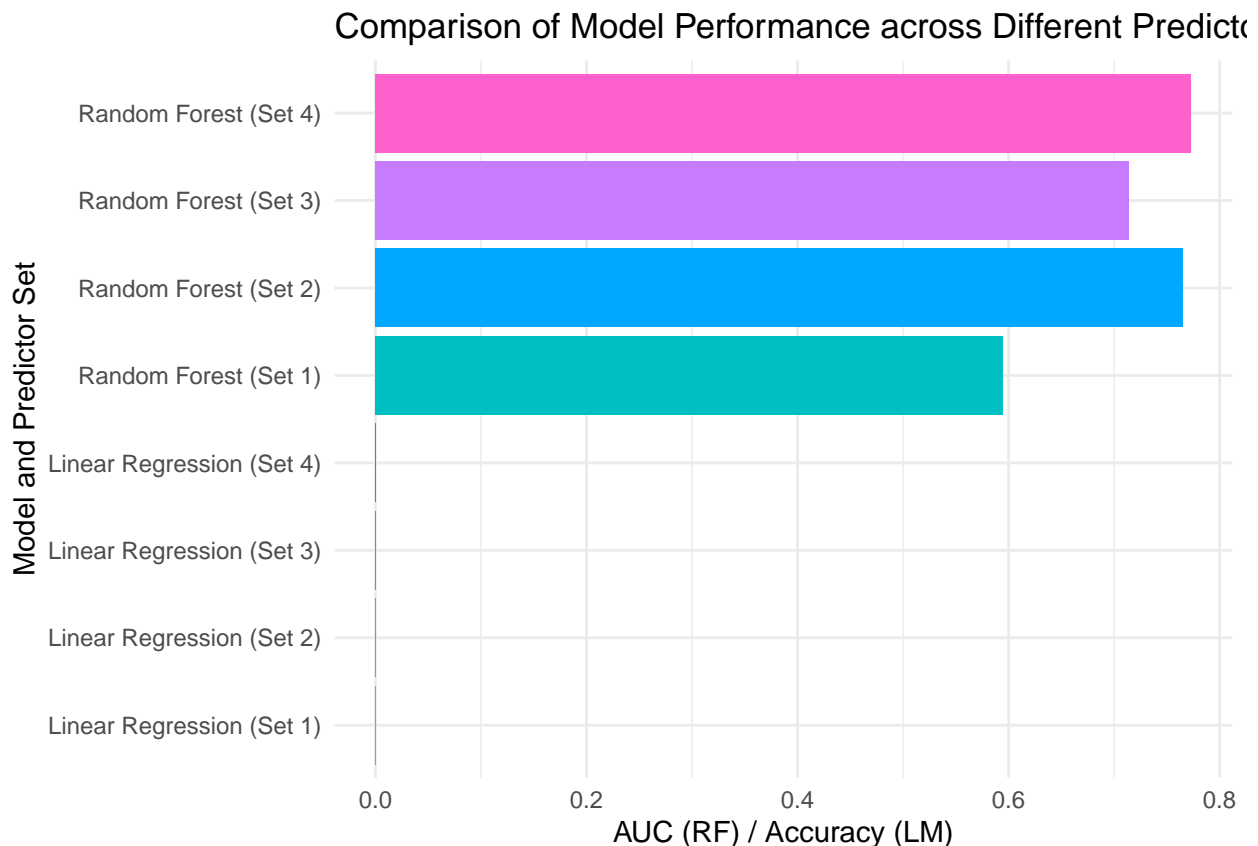
```

# 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.

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

```
## Interaction Model Accuracy: 0
```

### 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 =

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

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

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

```

print(base_cv)

## Generalized Linear Model
##
## 1098 samples
##    4 predictor
##    2 classes: 'No Success', 'Success'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 879, 878, 879, 878, 878
## Resampling results:
##
##   Accuracy   Kappa
##   0.6748485  0.1087814

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

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

print(reduced_cv)

## Generalized Linear Model
##
## 1098 samples
##    2 predictor
##    2 classes: 'No Success', 'Success'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 879, 878, 878, 878, 879
## Resampling results:
##
##   Accuracy   Kappa
##   0.6675758  0

```

## Visualize Feature Importance

### Effect Plots for Logistic Regression

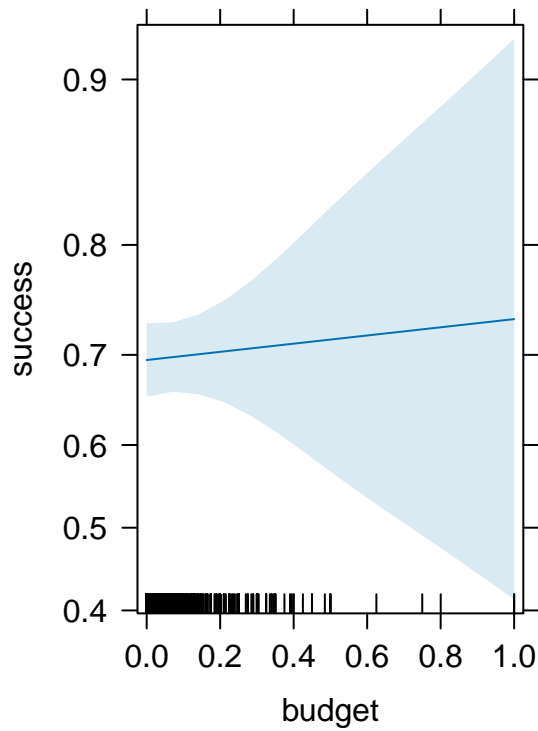
Visualize the effect of individual predictors on the probability of success.

```

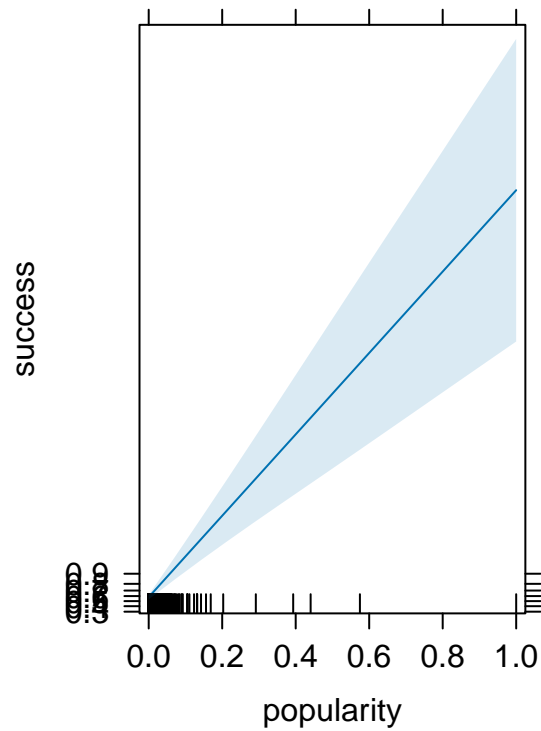
effect_plot <- allEffects(reduced_model)
plot(effect_plot)

```

**budget effect plot**



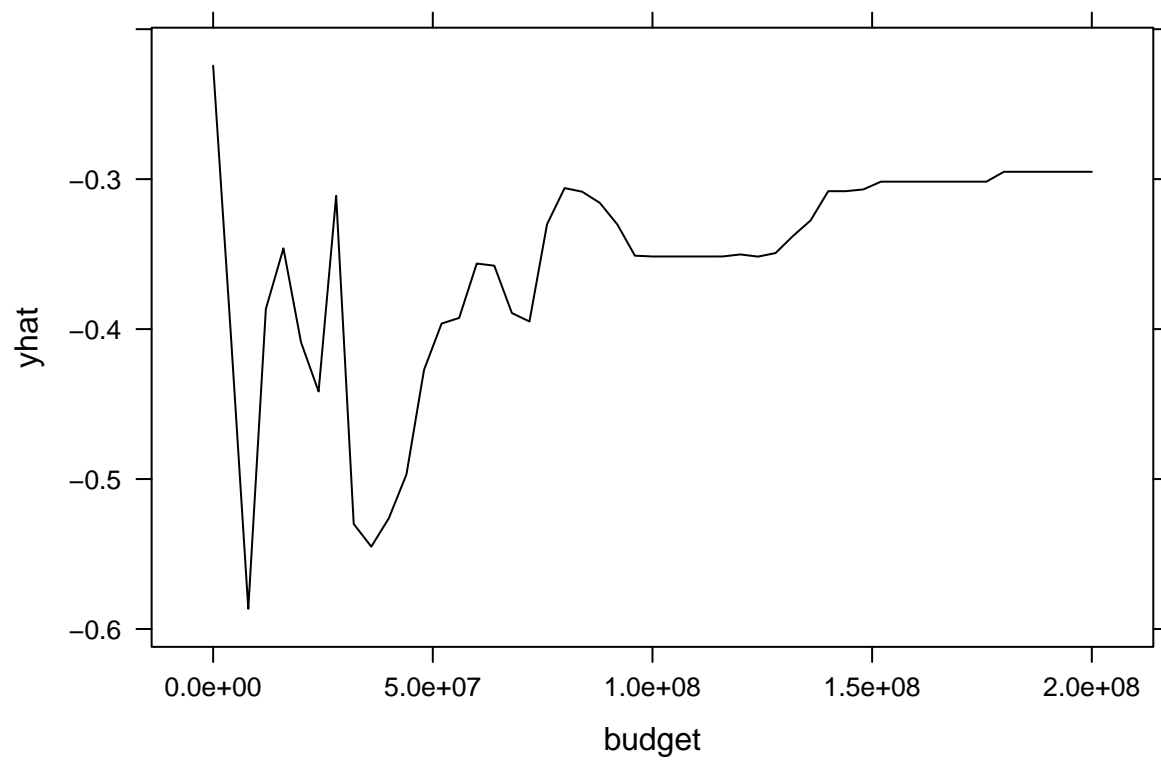
**popularity effect plot**



### Partial Dependence Plots

Use this for tree based models like Random Forest.

```
# 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)
```

