INDEX

1. INTRODUCTION

IMDb ratings have become a significant measure of a movie's quality, as perceived by a global audience. These ratings aggregate individual scores and reviews from millions of users, providing a reliable indicator of a movie's reception. High IMDb ratings often correlate with critical acclaim and popular success, making them an invaluable resource for moviegoers seeking quality content.

The dataset used in this project includes key features that influence movie ratings, such as:

- **Poster_Link** - Link of the poster that imdb using
- **Series_Title** = Name of the movie
- **Released_Year** - Year at which that movie released
- **Certificate** - Certificate earned by that movie
- **Runtime** - Total runtime of the movie
- **Genre** - Genre of the movie
- **IMDB_Rating** - Rating of the movie at IMDB site
- **Overview** - mini story/ summary
- **Meta_score** - Score earned by the movie
- **Director** - Name of the Director
- **Star1,Star2,Star3,Star4** - Name of the Stars
- **No_of_votes** - Total number of votes
- **Gross** - Money earned by that movie

By accurately classifying movies as highly rated or not, our project aims to help audiences quickly identify the best movies even among the top ones, thus saving time and enhancing their viewing experience. This classification can streamline the decision-making process, allowing users to bypass less favourable options and focus on the most critically acclaimed and popular films.

The analysis is conducted using R, a versatile statistical programming language that provides a wide array of tools for data manipulation, visualization, and machine learning. We begin by preprocessing the data and applying Logistic Regression to establish a performance baseline. Following this, we implement the KNN classifier, rigorously examining different kkk values to optimize model performance.

This study not only demonstrates the practical application of Logistic Regression and KNN algorithms but also provides valuable insights into their effectiveness in classifying IMDb movies based on ratings. By conducting a comprehensive comparative analysis, we contribute to a deeper understanding of the strengths and limitations of these algorithms in binary classification tasks, particularly in the domain of movie ratings. The findings of this project are expected to enhance our knowledge and guide future endevours in similar classification challenges.

2. DATA PREPROCESSING:

- We store runtime as numeric from string excluding the word 'minutes'
- We find a released year given as "PG" using unique() ,find the corresponding row and change it to it's correct released year, and store the released year column as a number instead of a string.
- We remove the first column Poster_link since it is not useful for our analysis.
- Gross income has to be adjusted for all the years to year 2019 using CPI for valid analysis or else discarded entirely. Since films in our dataset are from different countries, we have to account for their origin country, and how exactly the gross income was calculated, accounting for inflation till current year for each country, which is a project in itself. Hence we choose to discard it from our analysis.
- We find the columns which have 'NA' values and count them,we find meta score and gross is missing in 157 and 169 rows respectively. We subset the dataset into those entries only where none of our predictor variables (Released_Year,Runtime ,Meta_score ,No_of_Votes & Meta_score) have missing value.
- Hence we arrive at our final dataset of 843 observations which is now ready for further analysis.

Our Score index: IMDB Rating:

IMDB ratings are determined by the votes of registered IMDb users. The process can be broken down into several key steps:

1. **User Votes**: Registered IMDb users rate movies and TV shows on a scale from 1 to 10.
2. **Weighted Average**: IMDb uses a weighted average to calculate the final rating. This means that not all votes have the same influence. IMDb does not disclose the exact method for how the votes are weighted, but it is known that votes from more regular and reliable voters carry more weight.
3. **Bayesian Estimate**: To avoid bias from a small number of votes, IMDb employs a Bayesian estimate. This method combines the average rating of the item with the average rating of all items on IMDb to get a more balanced score. The formula used is:

$$\text{Weighted Rating} = \frac{v}{v+m} \times R + \frac{m}{v+m} \times C$$

Where:

- R = average rating for the item
- v = number of votes for the item
- m = minimum votes required to be listed in the Top Rated list (currently 25,000)
- C = the mean vote across the entire report (mean of all IMDb votes)

4. **Filtering**: IMDb has mechanisms in place to detect and prevent manipulation of ratings. This includes identifying and filtering out unusual voting patterns and fake accounts.
5. **Final Rating**: The final IMDb rating displayed on a movie or TV show's page is the result of this process. It is updated regularly to reflect the most current votes.

We find the 8th decile of IMDB Ratings, which comes out to be 8.1 . We use this IMDB rating of 8.1 to classify the movies as Highly Rated (1) or Not (0) and store this binary classification in a new column called High_Rating.

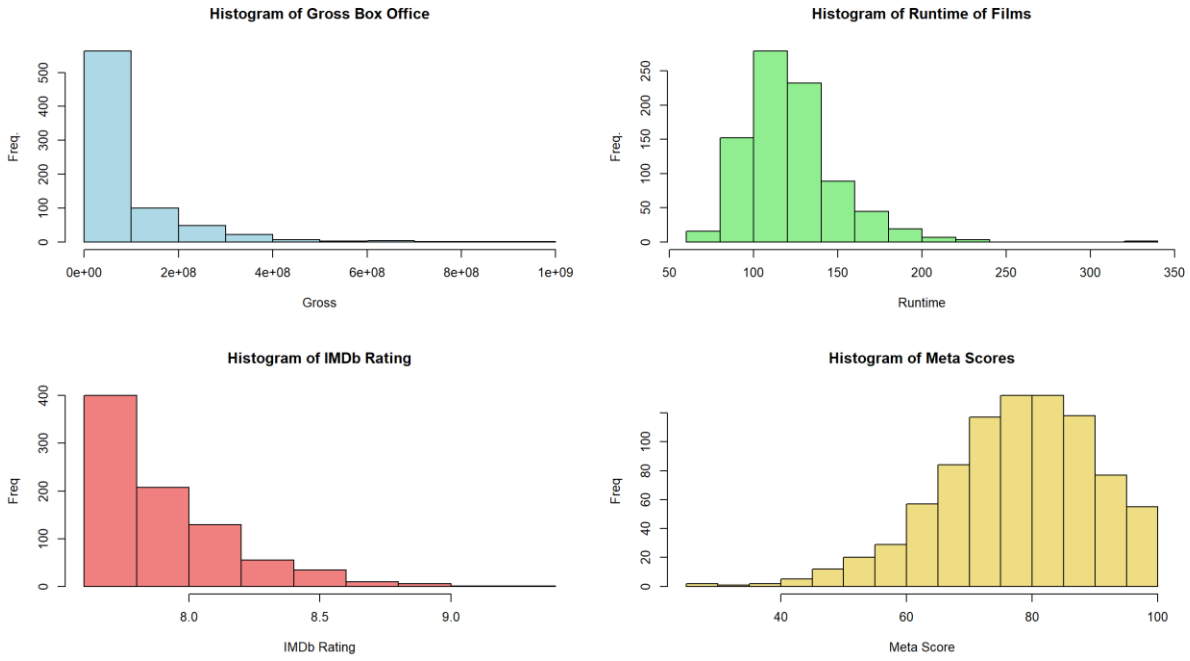
High_Rating = 1 , if IMDB_Rating >=8.1

0, otherwise

3. DATA VISUALISATION:

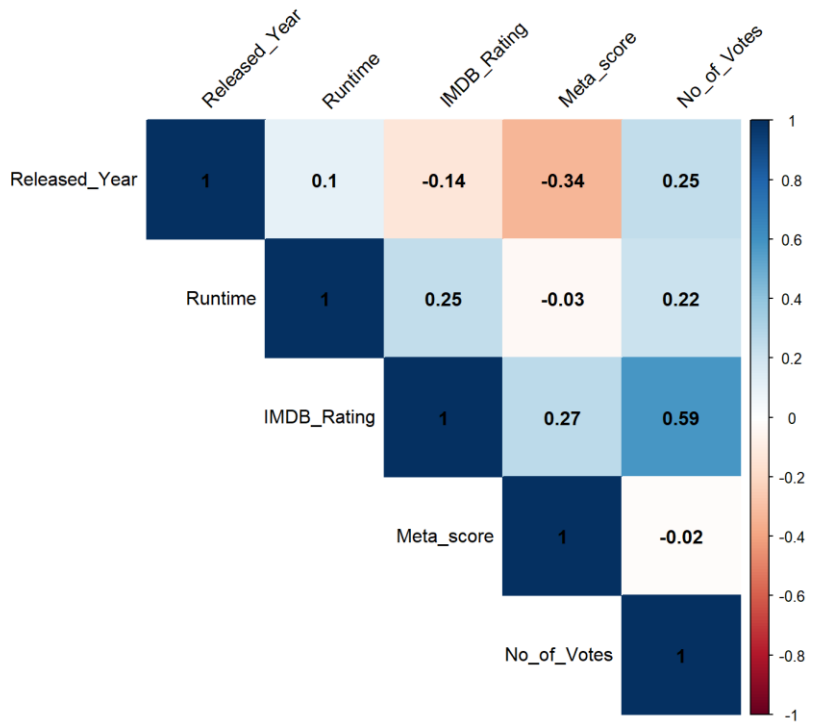
- We plot histograms of Gross,Runtime,IMDb_Rating and Meta scores to visualize their distributions.

```
> library(ggplot2)
> # Set up a 2x2 grid for plotting
> par(mfrow=c(2, 2))
> # Plot the histograms
> hist(imdb_data$Gross,
+      xlab="Gross",
+      ylab="Freq.",
+      main="Histogram of Gross Box Office",
+      col="lightblue",
+      border="black")
> hist(imdb_data$Runtime,
+      xlab="Runtime",
+      ylab="Freq.",
+      main="Histogram of Runtime of Films",
+      col="lightgreen",
+      border="black")
> hist(imdb_data$IMDB_Rating,
+      xlab="IMDb Rating",
+      ylab="Freq",
+      main="Histogram of IMDb Rating",
+      col="lightcoral",
+      border="black")
> hist(imdb_data$Meta_score,
+      xlab="Meta Score",
+      ylab="Freq",
+      main="Histogram of Meta Scores",
+      col="lightgoldenrod",
+      border="black")
> # Reset layout to default
> par(mfrow=c(1, 1))
```



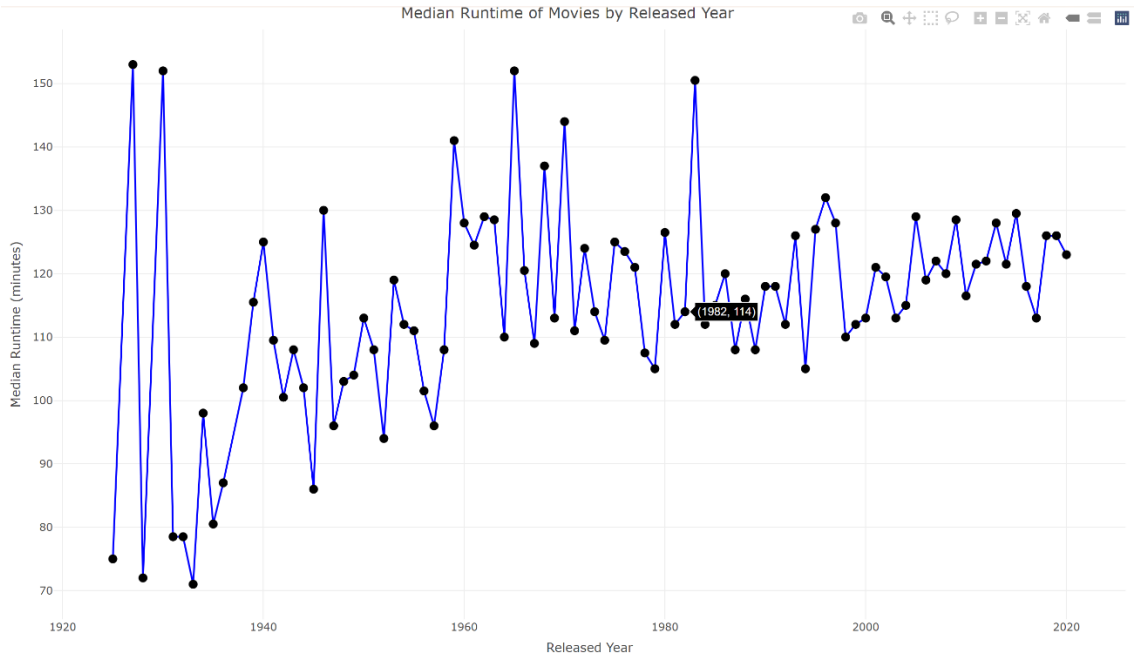
- We plot correlation matrix of the numeric variables Released_Year,Runtime,IMDB_Rating,Meta_score,No_of_Votes,Gross to study extent of linear association between each pair of these variables.

```
> library(corrplot)
> # Subset numeric variables
> numeric_vars <- imdb_data[, sapply(imdb_data, is.numeric) & names(imdb_data) != "Gross"]
> # Compute correlation matrix
> correlation_matrix <- cor(numeric_vars)
> # Plot correlation heatmap using corrplot
> c=corrplot(correlation_matrix, method = "color", type = "upper",
+            addCoef.col = "black", tl.col = "black", tl.srt = 45)
```



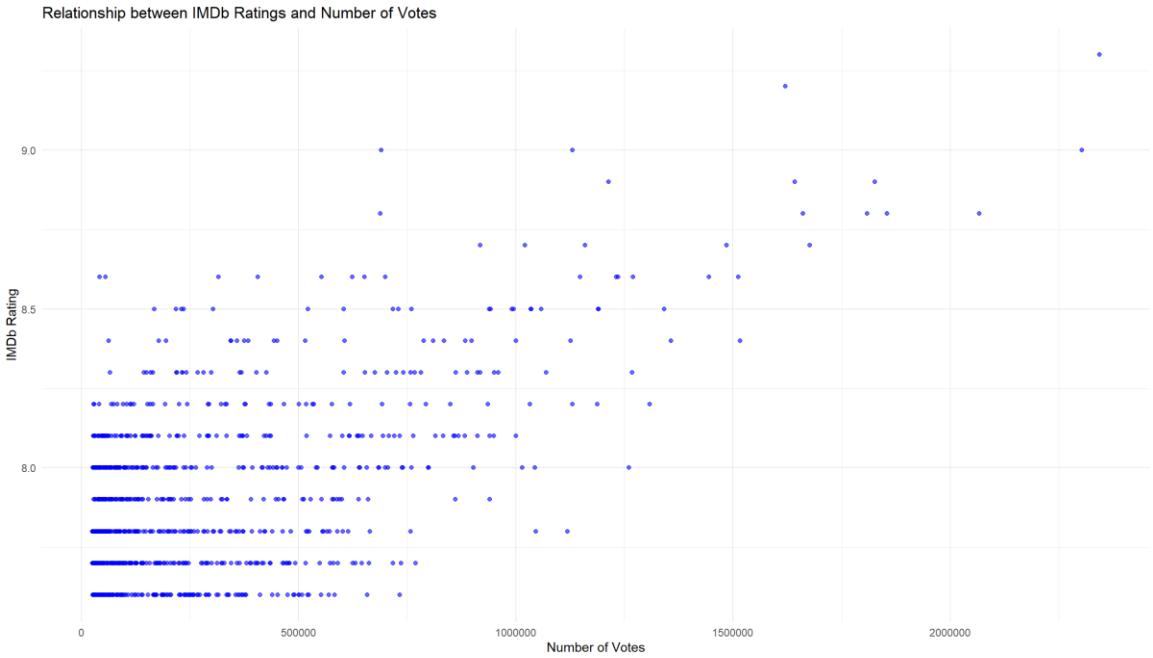
- We create an interactive plot of median imdb rating of each year right from 1925 to 2019.

```
> # Calculate median runtime by year
> med_runtime_by_year <- imdb_data %>%
+   group_by(Released_Year) %>%
+   summarise(Med_Runtime = median(Runtime, na.rm = TRUE))
> # Create interactive plot with plot_ly
> library(plotly)
> plot_ly(data = med_runtime_by_year, x = ~Released_Year, y = ~Med_Runtime,
+         type = 'scatter', mode = 'lines+markers',
+         marker = list(color = 'black', size = 10),
+         line = list(color = 'blue', width = 2)) %>%
+   layout(title = "Median Runtime of Movies by Released Year",
+          xaxis = list(title = "Released Year"),
+          yaxis = list(title = "Median Runtime (minutes)"),
+          hovermode = "closest")
```



- Scatter plot to visualize relationship between IMDb ratings and No_of_Votes :

```
> # Scatter plot to visualize relationship between IMDb ratings and No_of_Votes
> ggplot(imdb_data, aes(x = No_of_Votes, y = IMDB_Rating)) +
+   geom_point(color = "blue", alpha = 0.6) +  # Points with transparency
+   labs(title = "Relationship between IMDb Ratings and Number of Votes",
+        x = "Number of Votes",
+        y = "IMDb Rating") +
+   theme_minimal()
```



- In order to use the variable gross ,we have to adjust the gross income of every film to current year (using cost price index of origin country of the film or otherwise) for our analysis to be meaningful. Due to lack of additional information, we cannot use gross income.

4. SOME TERMS FREQUENTLY USED IN CLASSIFICATION TASKS :

Confusion Matrix

A confusion matrix is a vital tool in machine learning and statistics for evaluating the performance of a classification algorithm. It is a table that summarizes the outcomes of a prediction model by comparing the actual and predicted classifications. The matrix consists of four key components:

1. *True Positives* (TP): The number of correctly predicted positive instances.
2. *True Negatives* (TN): The number of correctly predicted negative instances.
3. *False Positives* (FP): The number of negative instances incorrectly predicted as positive (also known as Type I errors).
4. *False Negatives* (FN): The number of positive instances incorrectly predicted as negative (also known as Type II errors).

The confusion matrix provides a comprehensive view of how well the model distinguishes between different classes. From this matrix, various performance metrics can be derived, such as accuracy, precision, recall, and the F1 score, which help in assessing and improving the model's effectiveness. Understanding the confusion matrix is crucial for identifying the strengths and weaknesses of a classifier, especially when dealing with imbalanced datasets or aiming to optimize specific aspects of model performance.

The False Positive Rate (FPR) is directly related to the Type 1 Error because both metrics focus on the rate of incorrect positive classifications relative to all instances that should be negative according to the true labels or hypotheses.

- **Sensitivity (Recall)**: Proportion of actual positives correctly identified.

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **Specificity**: Proportion of actual negatives correctly identified.

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

- **F1 Score**: Harmonic mean of precision and sensitivity.

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Sensitivity}}{\text{Precision} + \text{Sensitivity}}$$

where

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

FPR = 1-Specificity TPR = Sensitivity

ROC :

The ROC (Receiver Operating Characteristic) curve is a graphical representation of a binary classification model's performance. It plots the true positive rate (sensitivity) against the false positive rate (1 - specificity) at various threshold settings. The area under the ROC curve (AUC) quantifies the model's ability to distinguish between classes: an AUC close to 1 indicates excellent performance, while 0.5 suggests random guessing. ROC curves help evaluate and compare models, especially in scenarios where class imbalance exists or different thresholds are considered. In ROC analysis, the threshold values are typically the unique predicted probabilities that separate the positive and negative classes in your test set. The `roc` function in the `pROC` package automatically determines these thresholds based on your predicted probabilities and the true class labels.Here's a detailed explanation of how these threshold values are chosen:

1. **Unique Predicted Probabilities**:
    - The `roc` function takes the predicted probabilities from your logistic regression model and identifies the unique values. These values are potential thresholds where the decision boundary changes from predicting one class to another.
2. **Evaluating Performance**:
    - For each unique threshold, the ROC function evaluates the sensitivity (True Positive Rate) and specificity (True Negative Rate) by applying the threshold to the predicted probabilities and comparing the resulting binary classification to the true class labels.
3. **Order of Thresholds**:
    - The thresholds are typically sorted in ascending order. The first threshold might be set to `-Inf` to ensure that all predicted probabilities are considered initially.
4. **Threshold Points on ROC Curve**:
    - Each threshold value corresponds to a point on the ROC curve. The curve is constructed by plotting sensitivity against 1-specificity for each threshold, thus providing a visual representation of the trade-off between True Positive Rate and False Positive Rate across different thresholds.

4. Analysis :

We split the pre-processed IMDB dataset of 843 observations into training set (85%) and testing set (15%) for our Logistic regression model, randomly choosing the rows for the training set to develop the model, and keeping aside the remaining unseen data as our test set for model validation.

○ Logistic Regression

Considering the event of a movie being highly rated (High_Rating=1) as success.

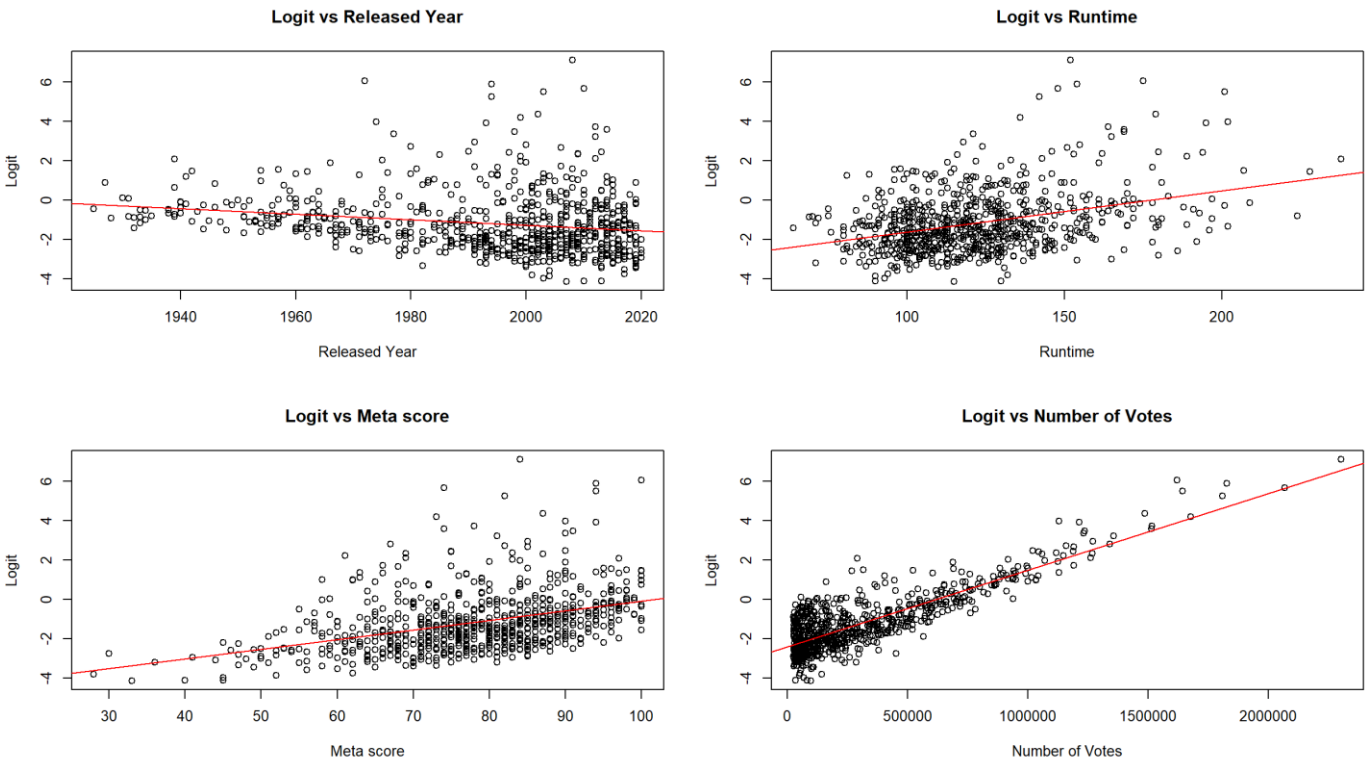
We train a Logistic regression model using Released_Year,Runtime,Meta_score, No_of_Votes to model success probability using the training data.

In summary, this output provides a detailed overview of the logistic regression model, including the estimated coefficients for each predictor, their statistical significance, and overall model fit statistics.

- Assumptions of Logistic regression :

I. Binary outcome: 1 if the movie is 'High_rated' , 0 otherwise.

II. Linearity of Logits : Calculate the logit values and create scatterplots against the continuous predictors.



By visual inspection of scatter plots, the predictor variables seem to have more or less a linear relationship with the logit values.

III. Multicollinearity-Check the variance inflation factor (VIF) for each predictor.

In logistic regression, the VIF (Variance Inflation Factor) is calculated similarly to linear regression but considering the logistic regression model's context:

1. **Fit a logistic regression model**: Fit a logistic regression model with the predictor variable of interest (say X_j) as the response and all other predictors (say X_{-j}) as predictors:

   $$\text{logit}(P(Y = 1)) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_{j-1} X_{j-1} + \beta_{j+1} X_{j+1} + \ldots + \beta_p X_p$$

   Here, $P(Y = 1)$ is the probability of the binary outcome Y being 1.

2. **Calculate VIF**: For each predictor X_j, calculate the VIF using the formula:

   $$\text{VIF}_j = \frac{1}{1 - R_j^2}$$

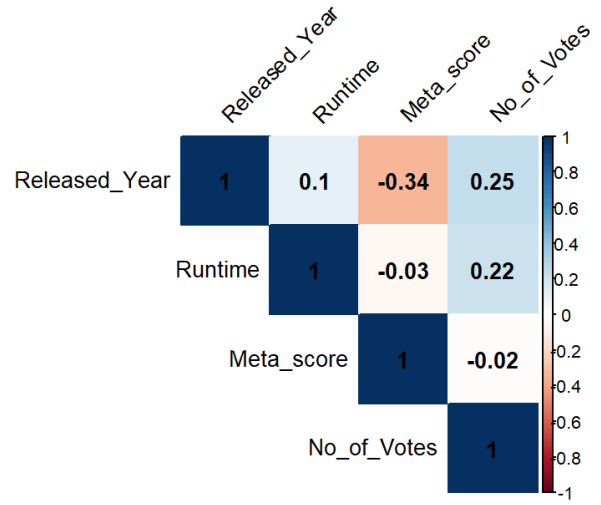
   where R_j^2 is the R^2 value of the logistic regression model from step 1. R_j^2 measures how well X_j is explained by the other predictors.

3. **Interpretation**:

   - **VIF = 1**: No multicollinearity. X_j is not correlated with any other predictors.
   - **VIF > 1**: Some correlation between X_j and other predictors.
   - **VIF > 5**: Moderate multicollinearity. The variance of β_j is inflated by more than 5 times due to correlation with other predictors.
   - **VIF > 10**: High multicollinearity. The variance of β_j is inflated by more than 10 times due to correlation with other predictors.

```
> vif_values <- vif(final_model)
> print(vif_values)
Released_Year       Runtime    Meta_score   No_of_Votes
     1.438549      1.033951      1.191727      1.265848
```

In general, VIF values close to 1 indicate minimal multicollinearity, while values above 5 or 10 suggest a problematic level of multicollinearity. Your results show that all VIF values are well below 5, which is typically considered acceptable. This suggests that your predictor variables are reasonably independent of each other in explaining the variation in the response variable (`High_Rating`).




```
> summary(log_reg_model$finalModel)

Call:
NULL

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)    4.111e+01  1.002e+01   4.104 4.05e-05 ***
Released_Year -2.402e-02  4.946e-03  -4.856 1.20e-06 ***
Runtime        1.174e-02  3.753e-03   3.127  0.00177 **
Meta_score     3.691e-02  9.378e-03   3.936 8.28e-05 ***
No_of_Votes    4.058e-06  3.916e-07  10.362  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 850.75  on 716  degrees of freedom
Residual deviance: 635.85  on 712  degrees of freedom
AIC: 645.85

Number of Fisher Scoring iterations: 5
```

Coefficients

This section lists the coefficients for each predictor variable in the model, including the intercept. Each row provides the following information:

1. **Variable Name**: The name of the predictor variable (e.g., `Released_Year`, `Runtime`, etc.).
2. **Estimate**: The estimated coefficient for the predictor variable. This value represents the change in the log-odds of the outcome for a one-unit change in the predictor, holding all other predictors constant. For example, the estimate for `Released_Year` is -2.402e-02, meaning each additional year decreases the log-odds of the outcome by 0.02402, holding other variables constant.
3. **Std. Error**: The standard error of the estimated coefficient. This value measures the variability of the coefficient estimate.
4. **z value**: The test statistic for the hypothesis test that the coefficient is zero. It is calculated as the estimate divided by the standard error.
5. **Pr(>|z|)**: The p-value for the hypothesis test that the coefficient is zero. A smaller p-value indicates stronger evidence against the null hypothesis (that the coefficient is zero). For example, a p-value of 1.20e-06 for `Released_Year` indicates strong evidence that the coefficient is not zero.
6. **Signif. codes**: Symbols indicating the significance level of the p-values. Typically, `***` denotes p-values less than 0.001, `**` less than 0.01, `*` less than 0.05, and so on.

Model Fit Statistics

This section provides information about the overall fit of the model:

1. **(Dispersion parameter for binomial family taken to be 1)**: Indicates the dispersion parameter used in the model, which is 1 for binomial logistic regression.
2. **Null deviance**: The deviance of the model with no predictors (only the intercept). It measures the fit of a model that only includes the intercept.
3. **Residual deviance**: The deviance of the fitted model. It measures the fit of the model with the predictors included.
4. **Degrees of freedom**: The degrees of freedom for the null and residual deviance. The null deviance has `n - 1` degrees of freedom, where `n` is the number of observations. The residual deviance has `n - p - 1` degrees of freedom, where `p` is the number of predictors.
5. **AIC (Akaike Information Criterion)**: A measure of the relative quality of the model. Lower AIC values indicate a better fit to the data, considering both the model's complexity and goodness of fit.

Iterations

This line indicates the number of Fisher Scoring iterations used to fit the model. Fisher Scoring is an iterative method used to estimate the model parameters.

Based on the reduction in deviance and the AIC, the logistic regression model appears to fit the data reasonably well, indicating that the predictors provide meaningful information. However, for a comprehensive evaluation, it's important to consider additional performance metrics and possibly compare with alternative models.

Now, we calculate McFadden's R^2

```
[1] "McFadden's R^2: 0.489098797140802"
```

Understanding McFadden's R-squared

McFadden's R-squared is a measure of the goodness of fit of a logistic regression model. It quantifies how well the model explains the variance in the dependent variable (in this case, `High_Rating`) relative to a model with no predictors. Here's what the value of 0.489098797140802 implies:

1. **Interpretation**: McFadden's R-squared ranges between 0 and 1. A value closer to 1 indicates that the model fits the data well, meaning the included predictors explain a significant portion of the variance in the outcome.
2. **Your Value (0.489098797140802)**:
    - This indicates that the predictors (`Released_Year`, `Runtime`, `Meta_score`, `No_of_Votes`) collectively explain approximately 48.91% of the variance in the `High_Rating` variable.
    - In other words, your logistic regression model, which includes these predictors, explains about 48.91% of the variability in whether a movie is rated as "High_Rating" or not.
3. **Assessment**:
    - McFadden's R-squared is a relative measure, so its interpretation depends on the context and what constitutes a "good" fit. In general, values above 0.2 or 0.3 are often considered indicative of a reasonably good fit, but this can vary depending on the field and specific application.
4. **Comparative Analysis**:
    - To further interpret the quality of your model, compare McFadden's R-squared with other models or benchmarks in your domain. Higher values indicate better explanatory power, but it's essential to consider the complexity and purpose of the model.

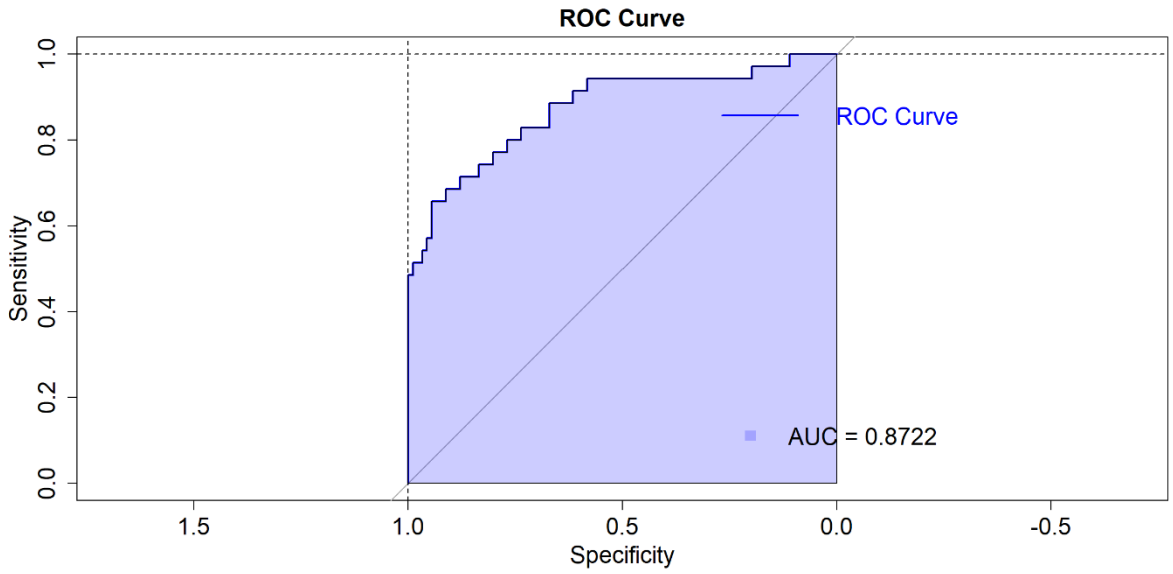
In summary, McFadden's R-squared of 0.489098797140802 suggests that your logistic regression model using `Released_Year`, `Runtime`, `Meta_score`, and `No_of_Votes` as predictors provides a moderately good fit to the data, explaining nearly 49% of the variability in predicting whether a movie is rated as "High_Rating". This statistic provides valuable insight into the effectiveness of your model in explaining and predicting the outcome variable based on the included predictors.

After training our model, we use it to predict the success probabilities of the movies in the test dataset which is unseen to it.

Now, we plot the ROC curve of the logistic regression to visualize the trade-off between Sensitivity and Specificity for various threshold values.

Of course, when Threshold value =1, Sensitivity=0 & Specificity=1

Threshold value = 0, Sensitivity=1 & Specificity=0.



1. **Plotting the Curve**: By plotting Sensitivity (True Positive Rate) on the y-axis and 1 - Specificity (False Positive Rate) on the x-axis for different threshold values, the ROC curve illustrates how the model's performance varies across the entire range of possible thresholds.
2. **Interpretation**: A curve that hugs the upper left corner of the plot indicates a model with high sensitivity and specificity across a range of thresholds, reflecting strong overall performance. The area under the ROC curve (AUC) quantifies the model's discriminatory ability: higher AUC values (closer to 1) indicate better discrimination between classes.

In summary, ROC curves are valuable tools for visualizing and evaluating the trade-offs between sensitivity and specificity in binary classification models, providing insights into model performance across different decision thresholds.

An AUC (Area Under the Curve) value of ROC of almost 0.8722 indicates that the model has strong discriminatory ability, effectively distinguishing between high-rated and not high-rated movies based on the chosen predictors. A value above 0.5 suggests the model performs better than random chance, with higher values indicating better performance, typically exceeding 0.8 for strong models.

Thus, we have the predicted success probabilities of the test data. Our task at hand is to settle for a final threshold point now.

If predicted probability>= threshold, model predicts 1 (ie 'Highly Rated') otherwise 0.

Note that the classes ‘High Rated’ and ‘ Not High Rated’ for the movies are not balanced. That is, there is a significant disparity in the number of instances (movies) belonging to each class within the training dataset. Thus, simply looking at the accuracy of the classification task for different threshold points would not suffice. We have to account for the trade-off between Sensitivity and Specificity.

The Sensitivity and Specificity of this classification task at hand for some selected threshold values are as follows:

Threshold	Sensitivity	Specificity
0.20	0.8571429	0.6703297
0.21	0.8285714	0.6813187
0.22	0.8285714	0.7142857
0.23	0.8285714	0.7252747
0.24	0.8000000	0.7472527
0.25	0.7714286	0.7802198
0.26	0.7714286	0.7802198
0.27	0.7714286	0.7802198
0.28	0.7428571	0.8131868
0.29	0.7142857	0.8351648
0.30	0.7142857	0.8461538
0.31	0.7142857	0.8571429
0.32	0.6857143	0.8791209
0.33	0.6857143	0.8901099
0.34	0.6857143	0.9120879
0.35	0.6857143	0.9120879
0.36	0.6857143	0.9120879
0.37	0.6857143	0.9120879
0.38	0.6857143	0.9120879
0.39	0.6571429	0.9120879
0.40	0.6571429	0.9120879
0.41	0.6571429	0.9230769
0.42	0.6571429	0.9230769
0.43	0.6285714	0.9450549
0.44	0.6000000	0.9450549
0.45	0.5428571	0.9560440
0.46	0.5428571	0.9560440
0.47	0.5428571	0.9670330
0.48	0.5428571	0.9670330
0.49	0.5142857	0.9670330
0.50	0.5142857	0.9670330

To account for this tradeoff, we make use of two methods to arrive at two threshold points. We would then compare their confusion matrices to arrive at our final threshold point which can be used to rate new movies.

1. *Maximizing Sensitivity while Ensuring Specificity ≥ a certain level.*

Initially, we identify all threshold values where the specificity is 0.9 or higher(i.e. FPR<=0.1).

```
> # Find thresholds where specificity is at least 0.90
> valid_thresholds <- thresholds[specificities >= 0.90]
> valid_thresholds
 [1] 0.3358403 0.3618246 0.3958286 0.4143399 0.4268818 0.4293750
 [7] 0.4319425 0.4372693 0.4431820 0.4478197 0.4589338 0.4781645
[13] 0.4960307 0.5057416 0.5107767 0.5146729 0.5174110 0.5273278
[19] 0.5386376 0.5515720 0.5651112 0.6044777 0.6600752 0.6881578
[25] 0.7200997 0.7510018 0.7657562 0.7905937 0.8651706 0.9366595
[31] 0.9704760 0.9929008 0.9971305       Inf
```

Then from this subset, we select the threshold that maximizes sensitivity and call it optimal threshold point.

```
> print(paste("Optimal Threshold:", optimal_threshold1))
[1] "Optimal Threshold: 0.335840308481836"
> #Print the sensitivity for optimal threshold
> print(max(valid_sensitivities))
[1] 0.6857143
```

Improvements Over Using a Fixed Threshold of 0.5:

1. **Prioritizes Specificity**: By ensuring that specificity is at least 0.9, the method significantly reduces the rate of false positives. This is especially important in applications where false positives have severe consequences.
2. **Optimizes Sensitivity**: Within the subset of thresholds that meet the specificity criterion, the method selects the one that maximizes sensitivity. This ensures that the model identifies as many true positives as possible, balancing the trade-off between sensitivity and specificity effectively.
3. **Handles Imbalanced Classes**: In imbalanced datasets, a threshold of 0.5 may not be optimal, as it can lead to poor performance in identifying the minority class. This method tailors the threshold to the specific characteristics of the dataset, improving the model's ability to correctly classify both classes.
4. **Improved Overall Performance**: By carefully selecting a threshold that balances both sensitivity and specificity, the method enhances the overall performance of the model, making it more reliable and effective for practical use.

Comparison:

- **Fixed Threshold (0.5)**: Assumes an equal trade-off between false positives and false negatives, which may not be appropriate for all applications, especially with imbalanced data.
- **Proposed Method**: Adapts to the dataset's characteristics, ensuring high specificity and maximizing sensitivity, leading to a more nuanced and effective classification model.

In summary, the proposed method offers a more refined and context-aware approach to threshold selection, improving the model's ability to make accurate predictions, especially in scenarios with imbalanced classes and specific performance requirements.

The confusion matrix for the optimal threshold on the test data :

```
Confusion Matrix and Statistics

          Reference
Prediction  1  0
         1 24  9
         0 11 82

               Accuracy : 0.8413
                 95% CI : (0.7656, 0.9003)
    No Information Rate : 0.7222
    P-Value [Acc > NIR] : 0.001239

                  Kappa : 0.5973

 Mcnemar's Test P-Value : 0.823063

            Sensitivity : 0.6857
            Specificity : 0.9011
         Pos Pred Value : 0.7273
         Neg Pred Value : 0.8817
             Prevalence : 0.2778
         Detection Rate : 0.1905
   Detection Prevalence : 0.2619
      Balanced Accuracy : 0.7934

       'Positive' Class : 1
```

Overall Metrics

1. **Accuracy**:
    - **Value**: 0.8413
    - **95% CI**: (0.7656, 0.9003)
    - **Explanation**: Accuracy is the proportion of correctly classified instances (both true positives and true negatives) out of the total instances. The 95% confidence interval provides a range in which the true accuracy is expected to fall 95% of the time.
2. **No Information Rate (NIR)**:
    - **Value**: 0.7222
    - **Explanation**: The NIR is the accuracy that would be achieved by always predicting the majority class. In this case, it's 72.22%.
3. **P-Value [Acc > NIR]**:
    - **Value**: 0.001239
    - **Explanation**: This p-value tests whether the model's accuracy is significantly greater than the NIR. A p-value of 0.001239 suggests that the model's accuracy is significantly better than just predicting the majority class.

Agreement Metrics

4. **Kappa**:
    - **Value**: 0.5973
    - **Explanation**: Kappa is a measure of agreement between the observed accuracy and the accuracy expected by chance. A Kappa of 0.5973 indicates moderate agreement beyond chance.
5. **McNemar's Test P-Value**:
    - **Value**: 0.823063
    - **Explanation**: McNemar's test checks for significant differences in the paired proportions of misclassifications. A p-value of 0.823063 suggests no significant difference, indicating that there isn’t a significant discrepancy between the types of errors made by the model.

Class-Specific Metrics

6. **Sensitivity (Recall)**:
    - **Value**: 0.6857
    - **Explanation**: Sensitivity (Recall) is the proportion of actual positives correctly identified by the model. Here, it means that 68.57% of the positive instances are correctly identified.
7. **Specificity**:
    - **Value**: 0.9011
    - **Explanation**: Specificity is the proportion of actual negatives correctly identified by the model. Here, it means that 90.11% of the negative instances are correctly identified.
8. **Positive Predictive Value (PPV or Precision)**:
    - **Value**: 0.7273
    - **Explanation**: PPV (Precision) is the proportion of predicted positives that are actual positives. Here, 72.73% of the instances predicted as positive are actually positive.

9. **Negative Predictive Value (NPV)**:
    - **Value**: 0.8817
    - **Explanation**: NPV is the proportion of predicted negatives that are actual negatives. Here, 88.17% of the instances predicted as negative are actually negative.

Prevalence and Detection Rates

10. **Prevalence**:
    - **Value**: 0.2778
    - **Explanation**: Prevalence is the proportion of actual positive instances in the dataset. Here, 27.78% of the instances are positive.
11. **Detection Rate**:
    - **Value**: 0.1905
    - **Explanation**: Detection rate is the proportion of the total instances that are true positives. It's calculated as Sensitivity * Prevalence. Here, 19.05% of the total instances are true positives.
12. **Detection Prevalence**:
    - **Value**: 0.2619
    - **Explanation**: Detection prevalence is the proportion of instances predicted to be positive. Here, 26.19% of the instances are predicted to be positive.

Balanced Accuracy

13. **Balanced Accuracy**:
    - **Value**: 0.7934
    - **Explanation**: Balanced Accuracy is the average of Sensitivity and Specificity. It's useful for evaluating models on imbalanced datasets. Here, it's 79.34%.

Positive Class

14. **Positive Class**:
    - **Value**: 1
    - **Explanation**: This indicates which class is considered the positive class in binary classification. Here, class '1' is treated as positive.

Summary:

- **Accuracy (84.13%)** and its confidence interval suggest good overall performance.
- **Sensitivity (68.57%)** and **Specificity (90.11%)** indicate the model is better at identifying negatives than positives.
- **Precision (72.73%)** indicates that most of the positive predictions are correct.
- **Balanced Accuracy (79.34%)** provides a more balanced view, especially useful for imbalanced classes.
- **Kappa (0.5973)** and the **p-value for McNemar's test (0.823063)** indicate moderate agreement and no significant difference in the types of errors.

This comprehensive set of metrics helps in understanding the model's performance across different aspects, particularly important in the context of imbalanced classes.

2. *Using F1 score*

The F1 score is a metric that combines precision and recall into a single value, providing a balance between the two. It is particularly useful when dealing with imbalanced datasets, where the positive class is much less frequent than the negative class.

Calculating the F1 Score

The F1 score is calculated using the formula:

F1=2 * Precision * Recall/(Precision + Recall) where,

- **Precision (Positive Predictive Value)**: The proportion of predicted positive instances that are actually positive.
- **Recall (Sensitivity)**: The proportion of actual positive instances that are correctly predicted as positive.

We choose the threshold point which has maximum F1 score.

```
> print(paste("Optimal Threshold:", optimal_threshold))
[1] "Optimal Threshold: 0.187807694583138"
> print(paste("Maximum F1 Score:", max_f1_score))
[1] "Maximum F1 Score: 0.799603174603175"
```

The confusion matrix for the optimal threshold on the test data :

```
Confusion Matrix and Statistics

          Reference
Prediction  1  0
         1 31 30
         0  4 61

               Accuracy : 0.7302
                 95% CI : (0.6438, 0.8053)
    No Information Rate : 0.7222
    P-Value [Acc > NIR] : 0.4662

                  Kappa : 0.4526

 Mcnemar's Test P-Value : 1.807e-05

            Sensitivity : 0.8857
            Specificity : 0.6703
         Pos Pred Value : 0.5082
         Neg Pred Value : 0.9385
             Prevalence : 0.2778
         Detection Rate : 0.2460
   Detection Prevalence : 0.4841
      Balanced Accuracy : 0.7780

       'Positive' Class : 1
```

Now, the two threshold points obtained using the above two methods have more or less balanced Sensitivity and Specificity. Thus, to arrive at our final threshold point, we can compare their accuracy directly.

As seen from the two confusion matrices, accuracy for the threshold obtained by the first method (0.335840308481836) = 84.13% while that of the threshold obtained by the second method (0.187807694583138) = 73.02%.

Hence, we choose the first threshold point for further classification of movies.
Final selected threshold point = 0.3358 (rounded up to 4 decimal places).
For future movies containing information on the predictor variables used, we find p, and make our decision accordingly.

Calculating the Success Probability p

In logistic regression, the success probability p for a binary outcome (highly rated or not highly rated) is computed using the logistic function:

$$p = \frac{1}{1+e^{-(\beta_0+\beta_1 \cdot \text{Released\_Year}+\beta_2 \cdot \text{Runtime}+\beta_3 \cdot \text{Meta\_score}+\beta_4 \cdot \text{No\_of\_Votes})}}$$

Where:

- $\beta_0, \beta_1, \beta_2, \beta_3, \beta_4$ are the coefficients estimated from the logistic regression model.
- $\text{Released\_Year}, \text{Runtime}, \text{Meta\_score}, \text{No\_of\_Votes}$ are the predictor variables for a specific movie.

Steps to Compute p:

1. **Calculate the Logit**: Compute the linear combination of the intercept and coefficients multiplied by the predictor variables:

   $$\text{logit} = \beta_0 + \beta_1 \cdot \text{Released\_Year} + \beta_2 \cdot \text{Runtime} + \beta_3 \cdot \text{Meta\_score} + \beta_4 \cdot \text{No\_of\_Votes}$$

2. **Apply the Logistic Function**: Use the logistic function to transform the logit into a probability p:

   $$p = \frac{1}{1+e^{-\text{logit}}}$$

3. **Interpret the Result**: p represents the estimated probability that the movie is highly rated (class 1).

Decision Rule: Based on the chosen threshold (0.3358 in this case), if $p \geq 0.3358$, the movie is predicted to be highly rated; otherwise, it is predicted not to be highly rated.

This approach leverages the logistic regression model's coefficients to provide a probabilistic assessment of movie ratings based on their attributes.

K-Nearest Neighbours (KNN) Classifier

Introduction

The K-Nearest Neighbors (KNN) classifier is a simple, yet powerful, non-parametric algorithm used for classification and regression tasks. KNN operates on the assumption that similar data points are likely to belong to similar classes. This makes it particularly intuitive and easy to understand.

How KNN Works

1. **Parameter Selection**:
   - Choose the number of nearest neighbors (kkk) to consider for determining the class of a data point.
2. **Distance Metric**:
   - Select a distance metric to measure similarity between data points. Common choices include Euclidean distance, Manhattan distance, and Minkowski distance.
3. **Find Neighbours**:
   - For a given data point, calculate the distance to all other data points in the training set using the chosen metric.
4. **Sort Neighbours**:
   - Sort these distances in ascending order and select the k closest neighbours.
5. **Class Assignment**:
   - Assign the class that is most frequent among the k nearest neighbours to the data point.

Application to IMDb Movie Ratings

In this project, we applied the KNN classifier to classify IMDb movies as either high-rated (1) or not high-rated (0) based on the following predictors: Released_Year, Runtime, Meta_score, and No_of_Votes. Here's a detailed methodology of our approach:

Data Preparation

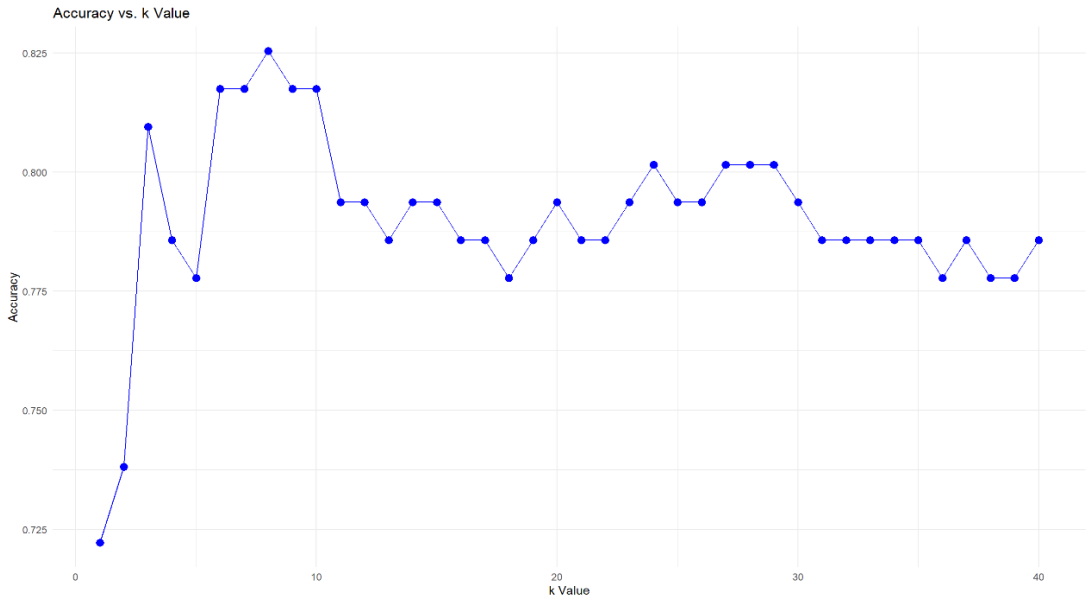
1. **Selection of Predictor and Target Variables**:
   - We selected Released_Year, Runtime, Meta_score, and No_of_Votes as the predictor variables.
   - The target variable was High_Rating, which indicates whether a movie is high-rated (1) or not (0).
2. **Scaling the Predictors**:
   - To ensure that each feature contributes equally to the distance calculations, we scaled the predictor variables. This standardization process transforms the data to have a mean of zero and a standard deviation of one.

Train-Test Split

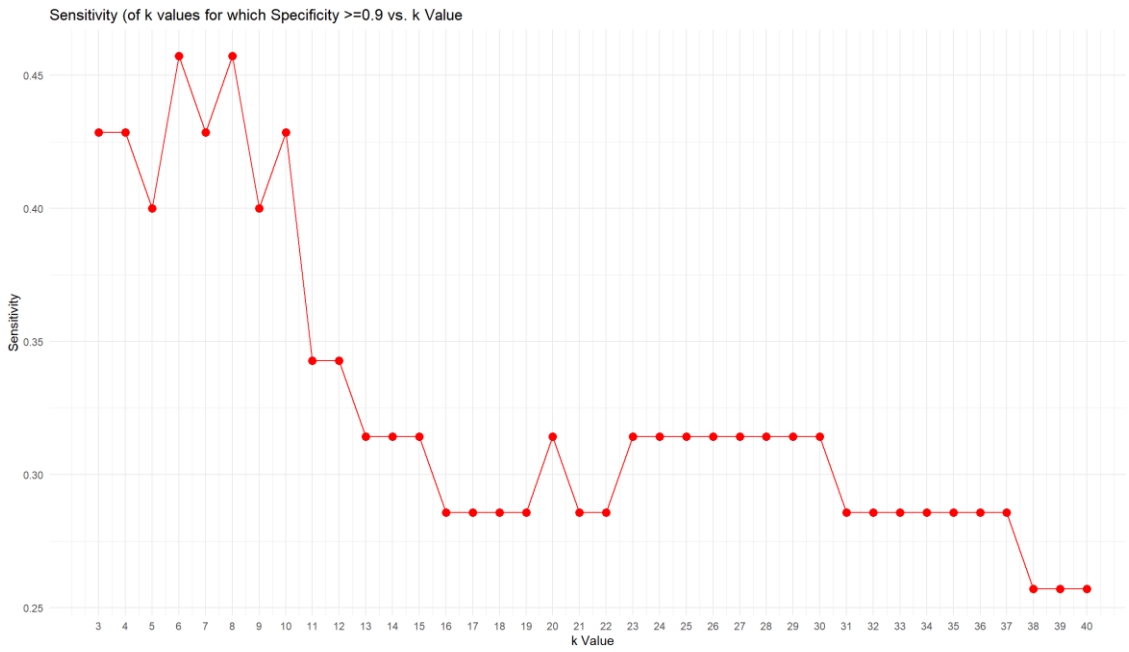
- We split the dataset into training and testing sets using an 85-15% split. This ensures that the model is trained on a substantial portion of the data while being validated on a separate, unseen subset. [We use the same datasets for training and testing as for the logistic regression]

Model Training and Evaluation

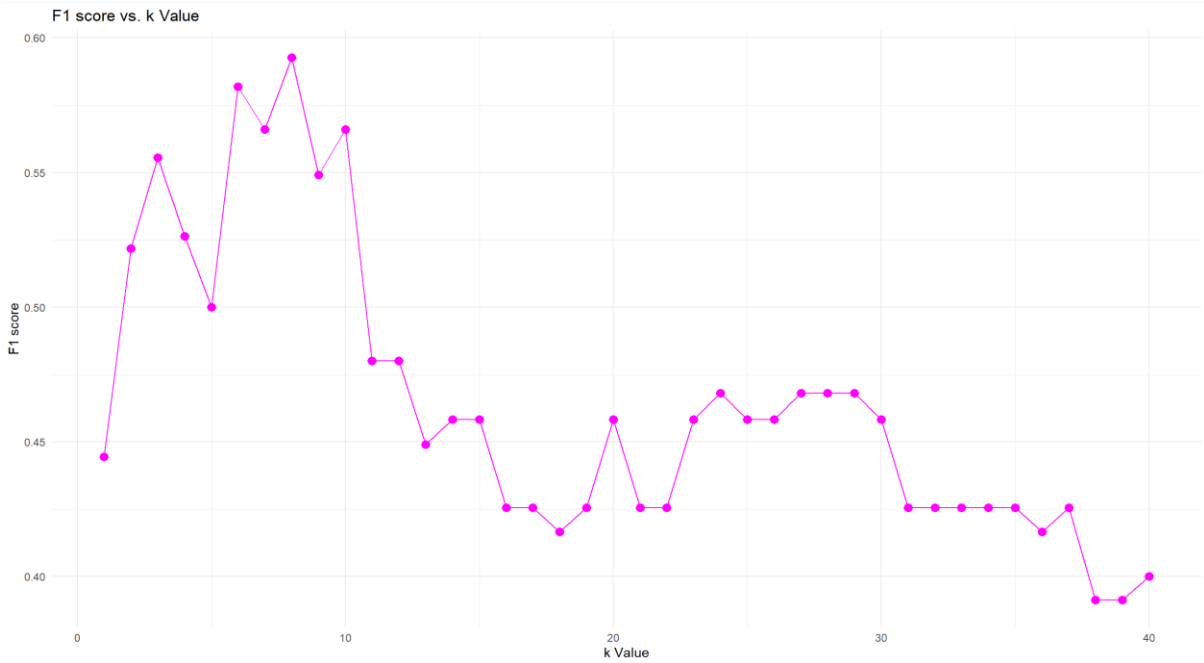
1. **Training the KNN Model**:
   - We trained the KNN model with different values of k ranging from 1 to 40.
   - For each k, we evaluated the model's performance using various metrics: accuracy, specificity, sensitivity, and F1 score.
2. **Choosing the Optimal k**:
   - **Maximum Accuracy**: The k value that resulted in the highest accuracy was 8, achieving an accuracy of 81.75%.



   - **Trade-off Between Sensitivity and Specificity**: Given the imbalance in the class labels (Highly_rated 1 and 0), we evaluated the trade-off between sensitivity and specificity. Among classifiers with specificity ≥0.9, k=6 had the highest sensitivity.



- **Maximum F1 Score**: The k value with the highest F1 score was 8.



Confusion matrix for k=6:

```
[1] "Confusion Matrix for Optimal k (sensitivity when specificity>=0.9):"
> print(final_cm2)
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 85 19
         1  6 16

               Accuracy : 0.8016
                 95% CI : (0.7212, 0.8673)
    No Information Rate : 0.7222
    P-Value [Acc > NIR] : 0.0265

                  Kappa : 0.4417

 Mcnemar's Test P-Value : 0.0164

            Sensitivity : 0.4571
            Specificity : 0.9341
         Pos Pred Value : 0.7273
         Neg Pred Value : 0.8173
             Prevalence : 0.2778
         Detection Rate : 0.1270
   Detection Prevalence : 0.1746
      Balanced Accuracy : 0.6956

       'Positive' Class : 1
```

Confusion matrix for k=8:

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 89 21
         1  2 14

               Accuracy : 0.8175
                 95% CI : (0.7388, 0.8806)
    No Information Rate : 0.7222
    P-Value [Acc > NIR] : 0.0090212

                  Kappa : 0.4538

 Mcnemar's Test P-Value : 0.0001746

            Sensitivity : 0.4000
            Specificity : 0.9780
         Pos Pred Value : 0.8750
         Neg Pred Value : 0.8091
             Prevalence : 0.2778
         Detection Rate : 0.1111
   Detection Prevalence : 0.1270
      Balanced Accuracy : 0.6890

       'Positive' Class : 1
```

- We examined the confusion matrices for k=6 and k=8. The accuracy for k=6 was 80.16%, while the accuracy for k=8 was 81.75%.

3. **Conclusion**:
   - Although k=6 offered a balanced sensitivity and specificity, its overall accuracy was slightly lower than that of k=8.
   - Thus, we selected k=8 as the optimal value for our final KNN classifier model due to its higher accuracy and balanced performance metrics.
4. **Final Choice**: Despite k=6 having balanced sensitivity and specificity, k=8 provided the best overall accuracy and balanced metrics (keeping F1 score in account). Therefore, we selected k=8 as the optimal value for our KNN classifier.
5. **Future Predictions**:
   - For future movie data points with information on all four predictor variables (Released_Year, Runtime, Meta_score, and No_of_Votes), the KNN classifier with k=8 will operate as follows:
     - **Distance Calculation**: Calculate the Euclidean distance from the new data point to all points in the training set.
     - **Identify Nearest Neighbours**: Identify the 8 nearest neighbours to the new data point based on the Euclidean distance.
     - **Class Prediction**: Assign the class label (high-rated or not high-rated) that is most common among these 8 neighbours.
     - **Tie** : In case of tie, class is randomly allotted.

5. Conclusion

Logistic regression efficiently utilized independent predictors such as Released Year, Runtime, Meta_score, and No_of_Votes to model the probability of a movie's rating category. This ability to handle independent predictors aligns well with our dataset characteristics and contributed significantly to its superior performance over KNN.Furthermore, logistic regression demonstrated robustness in handling imbalanced data and adjusting decision thresholds, offering flexibility to optimize classification outcomes based on specific project objectives. This adaptability is crucial for minimizing classification errors or prioritizing particular types of misclassifications as per project requirements.In contrast, while KNN is known for its versatility and simplicity, its performance in this project was hindered by the dataset's limited number of predictors and potential noise, which impacted its ability to generalize effectively compared to logistic regression.

Logistic regression achieved an accuracy of 84.13% with a threshold value set at 0.3358, surpassing KNN's accuracy of 81.75% with k=8 on the same test set. This higher accuracy underscores logistic regression's capability to effectively predict whether a movie belongs to the highly rated (1) or not highly rated (0) category.

Therefore, based on our thorough comparative analysis, logistic regression stands out as the preferred choice for accurately classifying highly rated movies among the top IMDb listed films. Beyond its superior accuracy, logistic regression's interpretability through coefficient analysis further underscores its suitability for similar classification tasks in the domain of movie ratings and beyond.