Identifying Potential for High IMDB Ratings in Netflix Originals Using Pre-release Features

Data Visualization Group Project

Using Machine Learning to Predict a Show's popularity potential

NETFLIX

Arya Mehta

Jane Heng

Prajwal Dambalkar

Vedika Sumbli

CONTENTS

Research Problem

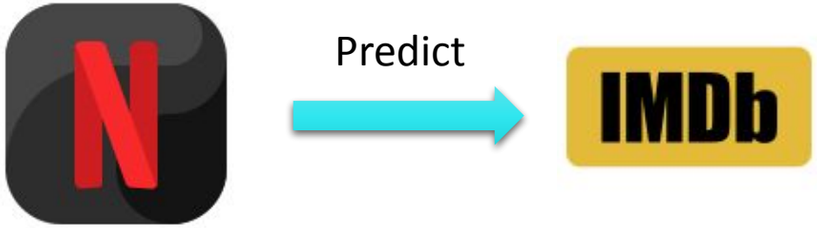
Research Problem

In the increasingly competitive landscape of streaming platforms like Netflix, predicting the performance of a show or movie has become critically important. In particular, **IMDb ratings**, as a widely referenced user evaluation metric, play a significant role in influencing long-term content reach, platform reputation, and future investment decisions.

Traditional Rating Prediction Method

- user ratings
- review counts
- social media sentiment

Sources : Audience behavior and feedback
Time : Only available after release



Early planning stage's meta data

- Director
- Cast
- Description
- …

7.5?
4.8?

Question:

But what if production teams want to make data-driven decisions during the early planning stages?

Research Problem

Key Question

Q1. **Develop an ML model** to forecast whether a piece of Netflix content will receive a high or low IMDb rating **using only pre-release metadata** such as title, genre, director, cast, runtime, and natural language processing (NLP) features derived from content descriptions。

Q2. **Which features have the greatest impact on IMDb rating predictions**?

Specific Requirements:

High level:

IMDB rating > threshold

IMDB vote count > threshold

Objective:
To predict whether a Netflix show will get high IMDB score (can be customized) before it is released, using metadata that is available during the pre-production or planning stage.

Dataset Overview

Datasets Overview

Dataset: Netflix data(Kaggle)

Dataset Contains Fields:

Type, Title, Director, Cast, Country, date_added, Release Year, Rating, Duration, Listed_in, Description

Original Dataset：Netflix data(Kaggle)

show_id	type	title	director	cast	country	date_added	release_year	rating	duration	listed_in	description
s6	TV Show	Midnight Mass	Mike Flanagan	Kate Siegel, Zach Gilford, Hamish Linklater, Henry Thomas, Kristin Lehman, Samantha Sloyan, Igby Rigney, Rahul Kohli, Annarah Cymone, Annabeth Gish, Alex Essoe, Rahul Abburi, Matt Biedel, Michael Trucco, Crystal Balint, Louis Oliver		24-Sep-21	2021	TV-MA	1 Season	TV Dramas, TV Horror, TV Mysteries	The arrival of a charismatic young priest brings glorious miracles, ominous mysteries and renewed religious fervor to a dying town desperate to believe.
s7	Movie	My Little Pony: A New Generation	Robert Cullen, JosÃ© Luis Ucha	Vanessa Hudgens, Kimiko Glenn, James Marsden, Sofia Carson, Liza Koshy, Ken Jeong, Elizabeth Perkins, Jane Krakowski, Michael McKean, Phil LaMarr		24-Sep-21	2021	PG	91 min	Children & Family Movies	Equestria's divided. But a bright-eyed hero believes Earth Ponies, Pegasi and Unicorns should be pals â€” and, hoof to heart, sheâ€™s determined to prove it.
s8	Movie	Sankofa	Haile Gerima	Kofi Ghanaba, Oyafunmike Ogunlano, Alexandra Duah, Nick Medley, Mutabaruka, Afemo Omilami, Reggie Carter, Mzuri	United States, Ghana, Burkina Faso, United Kingdom, Germany, Ethiopia	24-Sep-21	1993	TV-MA	125 min	Dramas, Independent Movies, International Movies	On a photo shoot in Ghana, an American model slips back in time, becomes enslaved on a plantation and bears witness to the agony of her ancestral past.

Data sources: Kaggle url: https://www.kaggle.com/datasets/shivamb/netflix-shows

Datasets Overview

Raw Dataset: missing values, data type transformation, different data scales, mixed case formatting, Long text fields contain special characters, spaces, and commas.

Column	Variable Type	Data Type	Detail	Preprocessing	Missing values
show_id	Categorical	String	Unique identifier; can be ignored or used as index		0
type	Categorical	String	Movie / TV Show	Encoding	0
Title	Categorical	String	Title of the show	Lowercase	0
Director	Categorical	String		deriving features like director popularity	748
Cast	Categorical	String	Long text	deriving features like cast popularity	316
Country	Categorical	String	Country of origin	Focus on U.S.	0
Date_added	Categorical	String			0
Release_year	Numerical	String		Narrow to after year 2000	0
rating	Categorical	String		Encoding	0
duration	Numerical	String		Split into year, month, weekday, season.	0
listed_in	Categorical	String		Encoding	0
description	Categorical	String	Long text	NLP	0

Table: Original Dataset Summary

Datasets Exploration

Datasets Exploration

Data Combine & Data Enrich

- **Narrow data range** **('country =U.S' and 'release_year' after 2000)**
- **Budget** (production + marketing spend) **(Web scraping, use mean fill missing values)**
- **Director Success**

  – average IMDb rating of their past titles
- **Production-Company Track Record**

  – average past ratings of their shows
- **Cast Popularity**

  – aggregate social-media mentions & past IMDb performance
- **Genre Trend**

  – relative popularity of each genre over time
- **Release Season**

  – e.g., Summer, Holiday, Awards Season
- **IMDb Rating & Vote Count** **(Supplement Dataset)**

  – pulled via IMDb API; used to define our "popular" label

Calculated + Extract

Supplied IMDb ratings and vote count (imdb API) dataset.

These fields simulate what a studio knows pre-release

```python
def get_show_budget(show_title, show_type, release_year):
    """
    Get budget information from multiple sources
    """
    budget = None

    # 1. Try to get from Wikipedia
    try:
        # Format title for Wikipedia URL
        wiki_title = show_title.replace(' ', '_')
        url = f"https://en.wikipedia.org/wiki/{wiki_title}"
        response = requests.get(url)
        soup = BeautifulSoup(response.text, 'html.parser')

        # Look for budget information in infobox
        infobox = soup.find('table', {'class': 'infobox'})
        if infobox:
            for row in infobox.find_all('tr'):
                if 'Budget' in row.text:
                    budget_text = row.find('td').text
                    # Extract numeric value
                    budget = float(''.join(filter(str.isdigit, budget_text)))
                    break
    except:
        pass

    # 2. If Wikipedia fails, use industry averages based on type and year
    if budget is None:
        if show_type == 'TV Show':
            # TV Show budget estimates based on industry standards
            if release_year >= 2020:
                budget = 5_000_000  # Modern high-budget TV show
            elif release_year >= 2015:
                budget = 3_000_000  # Mid-budget TV show
            else:
                budget = 2_000_000  # Standard TV show
        else:
            # Movie budget estimates based on industry standards
            if release_year >= 2020:
                budget = 20_000_000  # Modern movie
            elif release_year >= 2015:
                budget = 15_000_000  # Mid-era movie
            else:
                budget = 10_000_000  # Older movie

    return budget
```

Datasets Exploration

Dataset Pre-processing

Data Cleaning

1. Convert 'date_added' to datetime
2. Extract 'duration_minutes’
3. Convert 'imdbrating’ and ‘imdbvotes’ to numeric

Missing Values

1. Check and handle the missing values
    - Fill missing duration values with mode value
    - Drop rows where 'imdbrating' or 'imdbvotes' are missing
2. Check and drop duplicate rows

Encoding

1. Encode the category data (e.g. type, release_season)

Normalization & Standardization

1. Standardize column names: ‘Date added’ -> ‘date_added’
2. Normalize text columns: lowercase and strip spaces
3. Normalize and standardize numeric fields to ensure consistent scale (marketing_budget)

Datasets Exploration

NLP Feature Extraction: Tokenization & TF-IDF

1. Text Cleaning & Tokenization

- Lowercase & strip punctuation
- Split each description into word tokens

2. Stop-Word Removal & Stemming

- Remove common English stop-words (e.g., “the,” “and,” “of”)
- Apply Porter stemmer to reduce words to their root forms

3. TF-IDF Vectorization

- Fit TfidfVectorizer on cleaned descriptions (max_features=200, ngram_range=(1,2))
- Convert each description into a 200-dimensional numeric vector

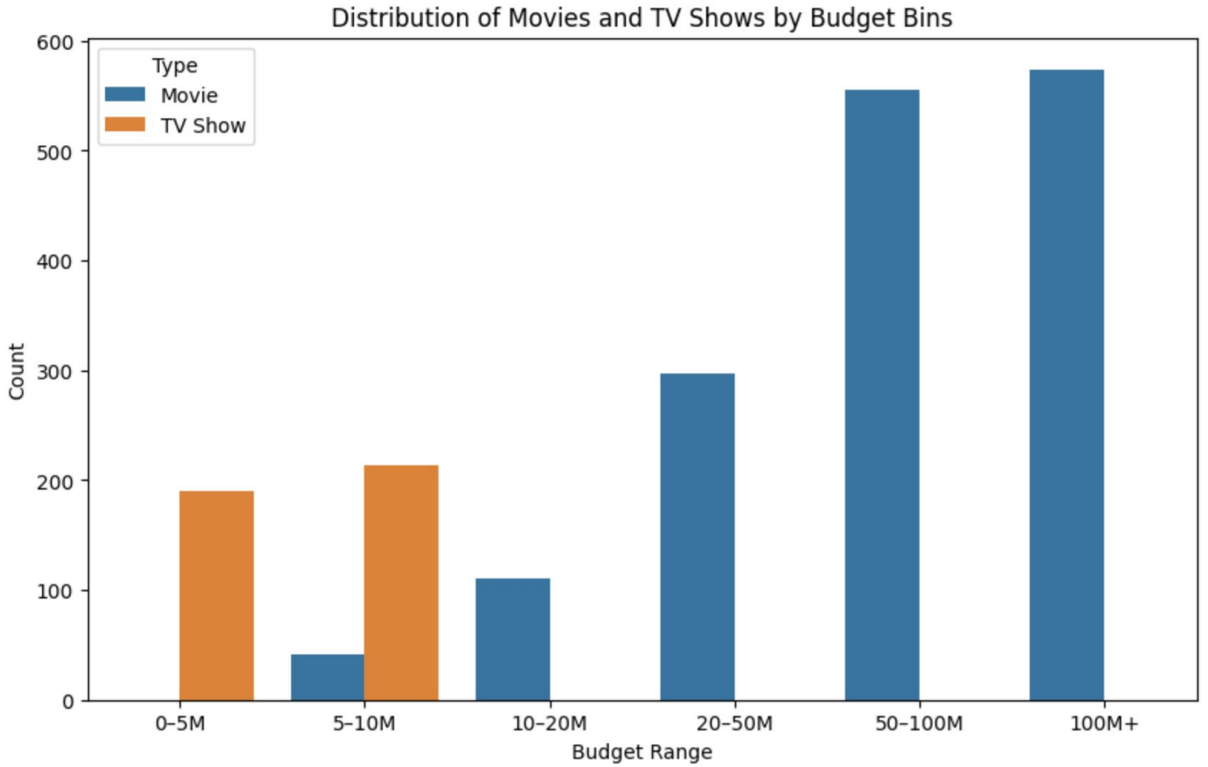
4. Keyword Extraction

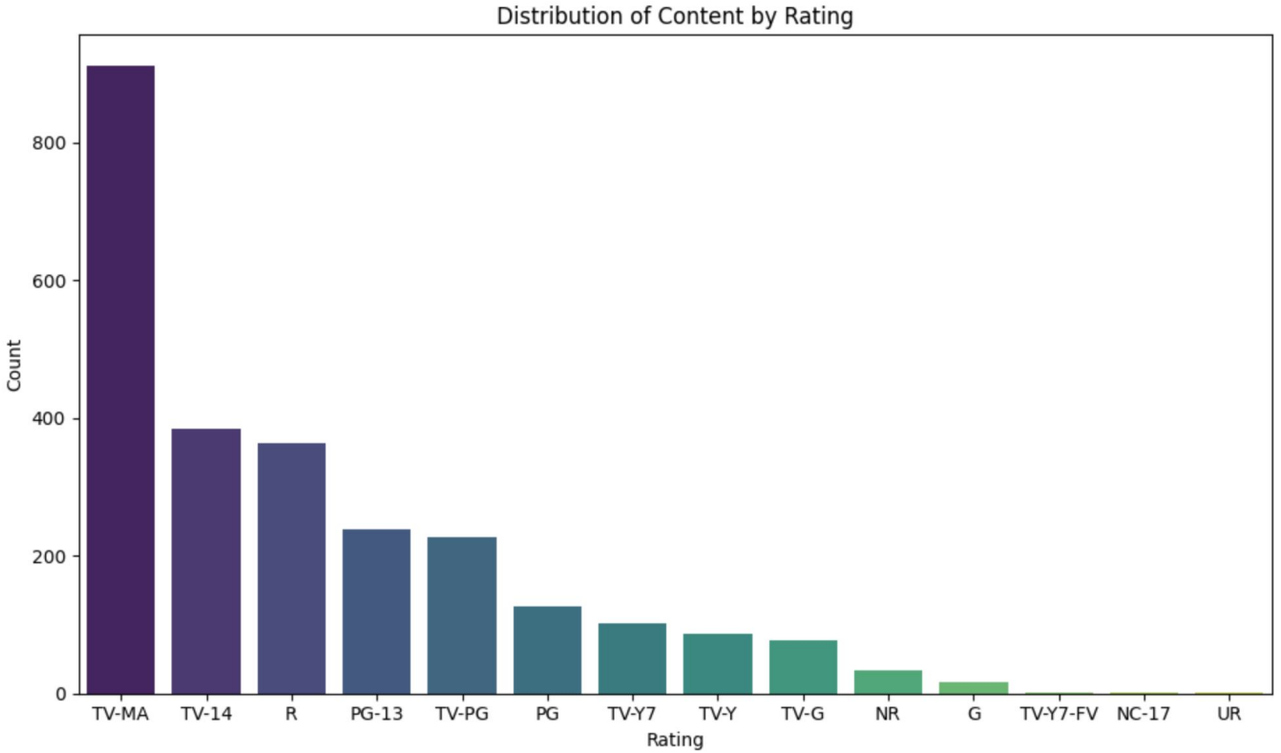
- Select top-10 highest TF-IDF terms per show
- Store as the new keywords column for modeling

```
"""
Use TF-IDF Vectorizer to extrac 5-10 key words from description and add a new column called keywords.
"""
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize the TF-IDF Vectorizer with a higher feature limit
tfidf = TfidfVectorizer(max_features=100, stop_words='english')  # More features for flexibility
tfidf_matrix = tfidf.fit_transform(df['description'].dropna())  # Transform non-null descriptions
feature_names = tfidf.get_feature_names_out()

# Function to extract at least 5 and at most 10 keywords
def extract_keywords(row):
    if pd.isna(row):  # Return empty list if the row is NaN
        return []
    tfidf_scores = tfidf.transform([row]).toarray().flatten()  # Get TF-IDF scores
    sorted_indices = tfidf_scores.argsort()[::-1]  # Indices sorted in descending order of scores
    # Select the top-scoring keywords
    keywords = [feature_names[i] for i in sorted_indices if tfidf_scores[i] > 0][:10]
    # Ensure there are at least 5 keywords by adding low-scoring ones if necessary
    if len(keywords) < 5:
        keywords = [feature_names[i] for i in sorted_indices[:5]]
    return keywords

# Apply the extraction function to the 'description' column
df['keywords'] = df['description'].apply(extract_keywords)
df.head()
```

Datasets Exploration

Budget Distribution Analysis





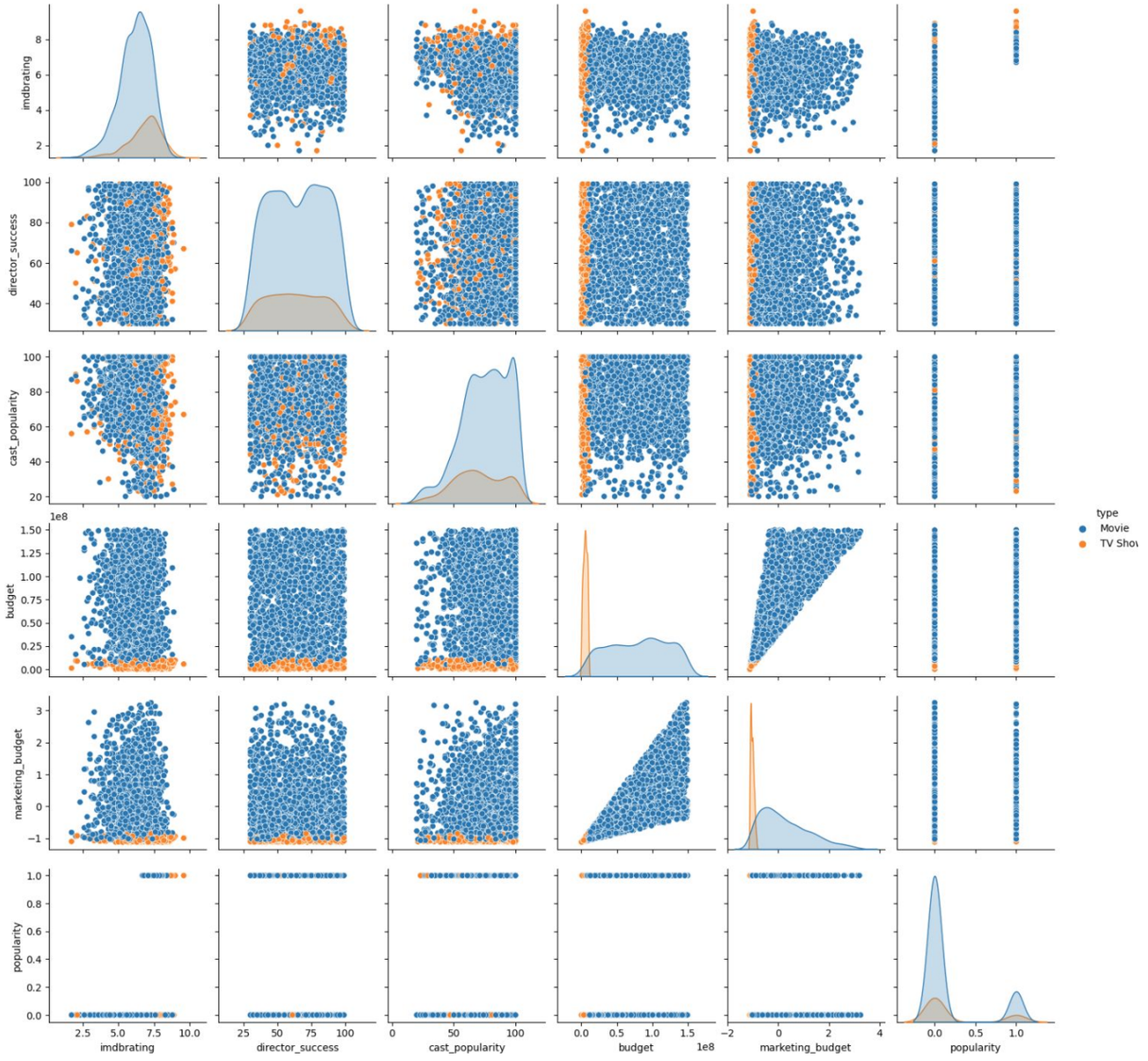
Insight:

- **TV shows** (orange) are mostly concentrated in **lower budget ranges (0-10M)**.
- **Movies** (blue) dominate the **higher budget ranges (20M+)**, especially **50-100M** and **100M+**.
- Very few TV shows exceed a **10M budget**, reinforcing their lower production budgets compared to movies.

Mostly shows are of TV-MA rating, followed by TV-14 & R

Datasets Exploration

Variable Pair Plot

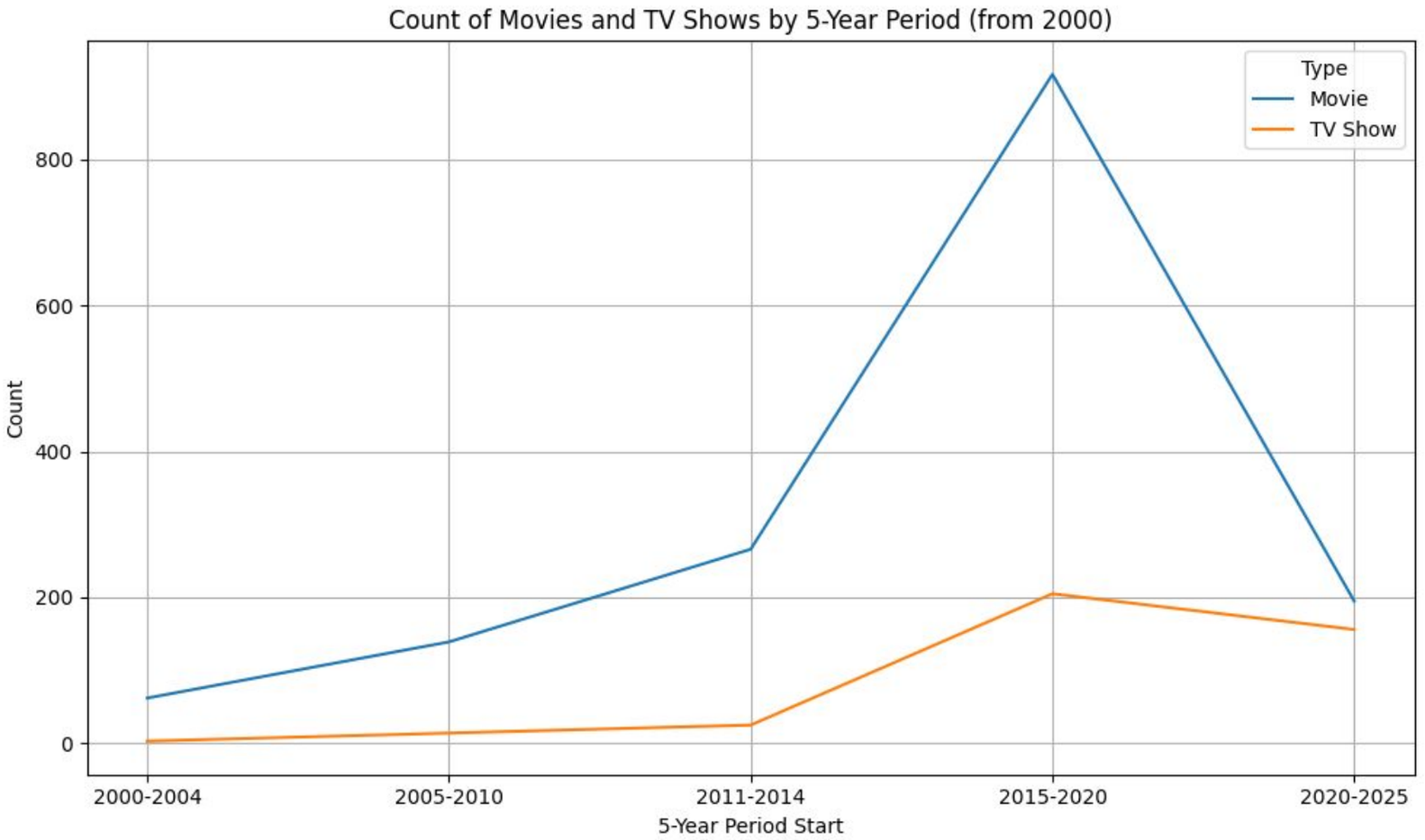


Variables Pair Plot Analysis

Insights:

- TV shows are clustered at lower budget and marketing values, while movies are more spread out.
- There's clear separation in budget and marketing budget between the two types.
- **IMDB rating** tends to be slightly higher for TV shows.
- **Budget and marketing budget** are highly correlated, shown by the tight diagonal line in their scatter plot.

Datasets Exploration



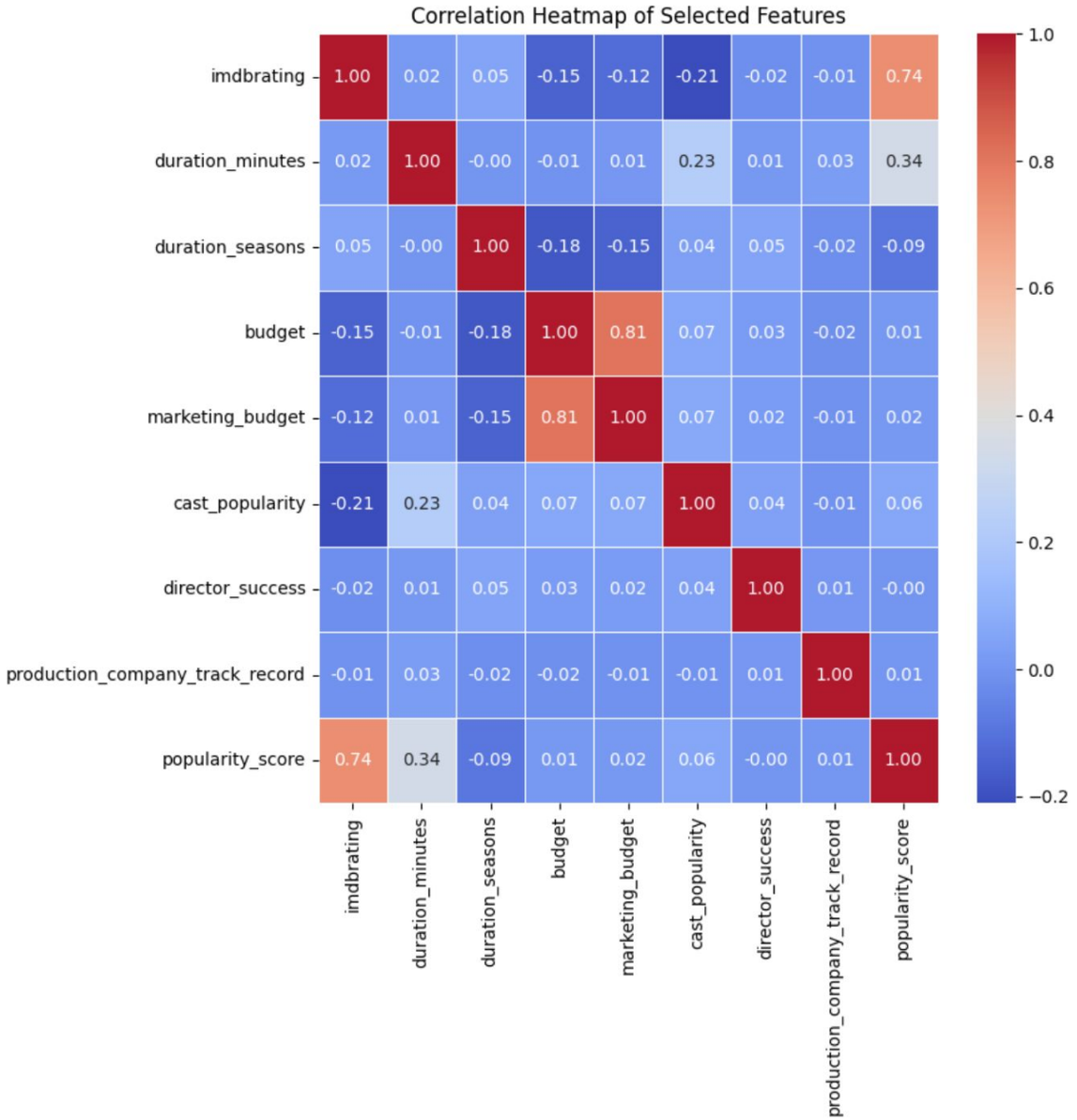
Time Trend Analysis

Insights:

- Movies have shown a **sharp decline post 2020** and TV shows have shown a **steady increase.**
- Till the recent data available, TV shows and Movies over the platform are at par with each other.

Datasets Exploration

Correlation Heatmap Analysis – Redundant Variables Reduce



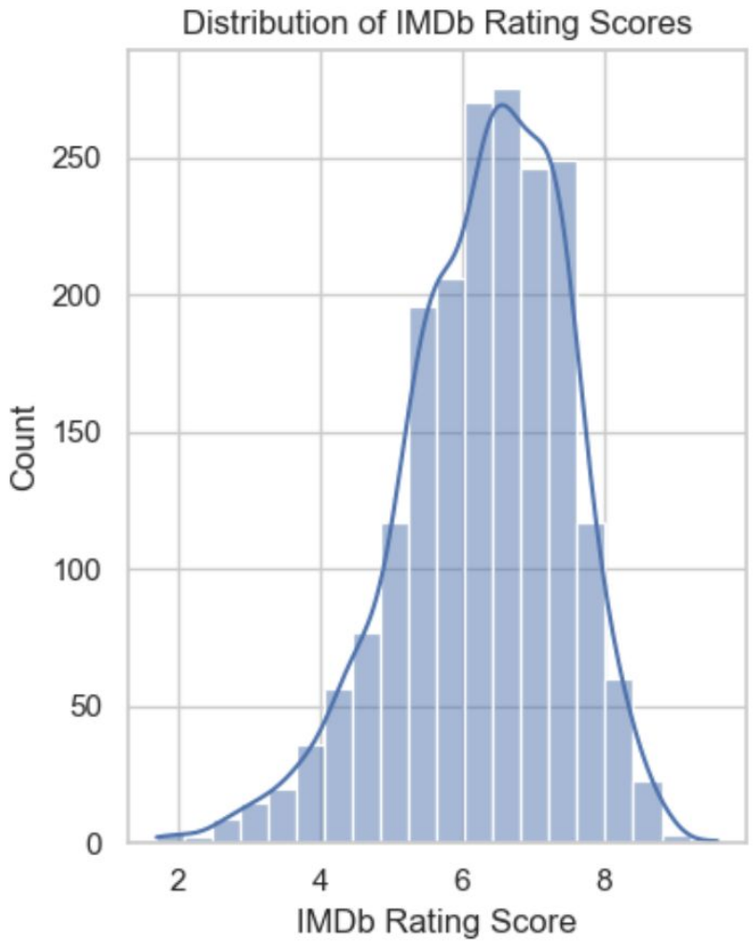
Insights:

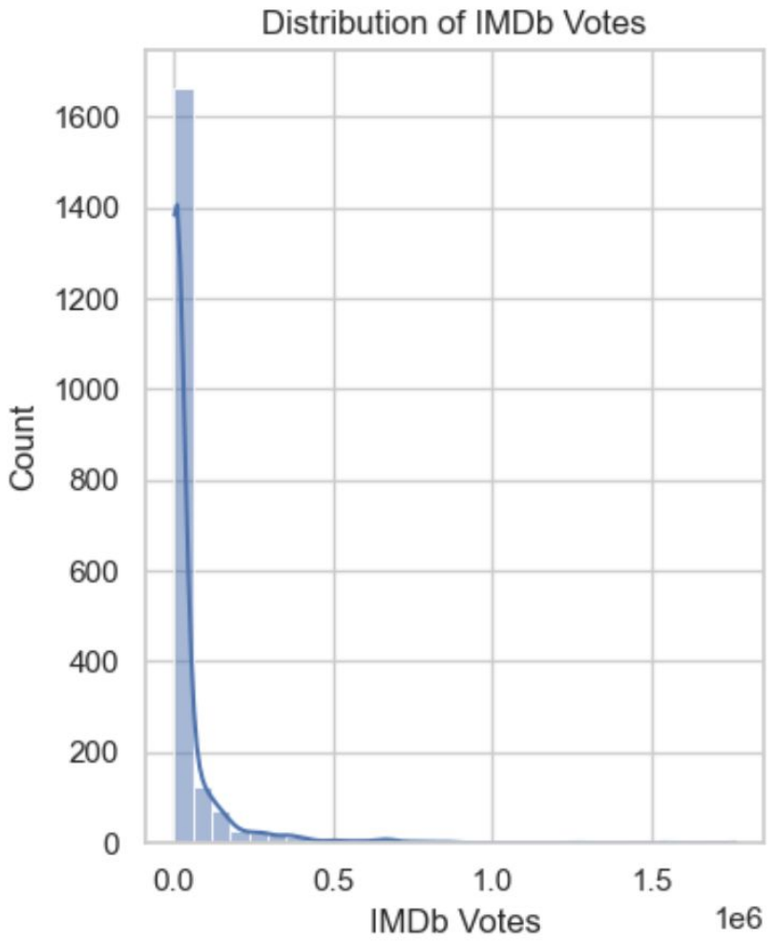
- **Budget and marketing budget** are **highly correlated (0.81)**, indicating they grow together.
- **IMDB rating has a slight negative correlation with budget (-0.15)**, suggesting bigger budgets don't necessarily guarantee higher ratings.
- **Duration in minutes** shows weak correlations with other variables, except a mild positive correlation with popularity (0.34).
- Marketing Budget and budget are redundant variables, can choose one.
- The remaining variables **have low correlation** with each other and can be used in modeling process.

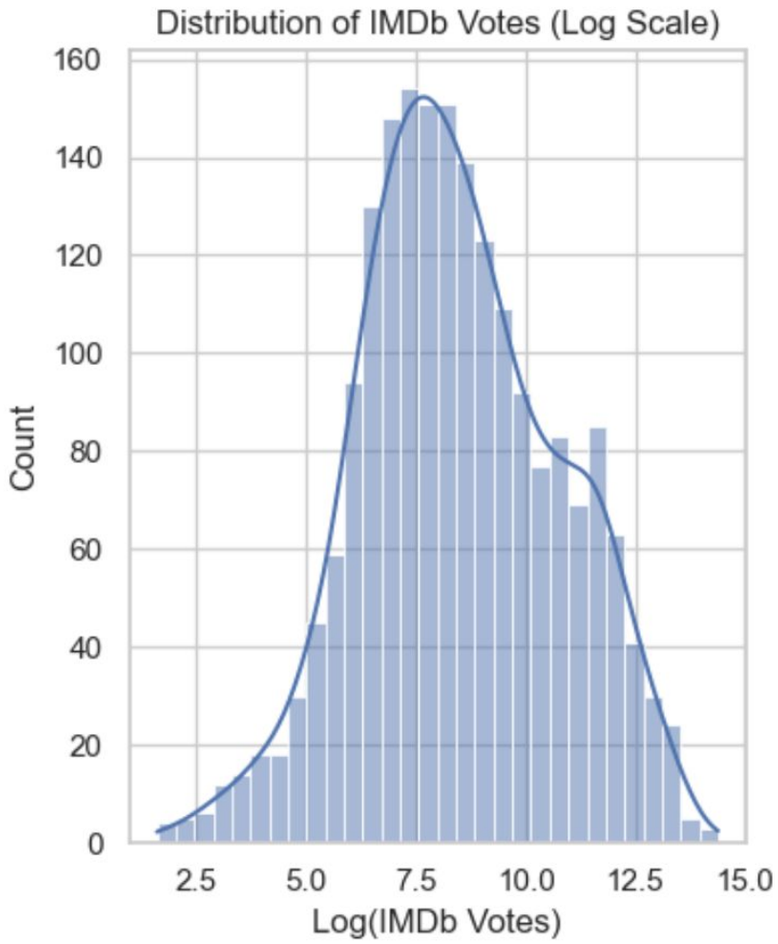
Parameters Selection

Datasets Exploration

Model Target: IMDb Rating and Votes: add a target “popularity” (1=high score / 0=low score)



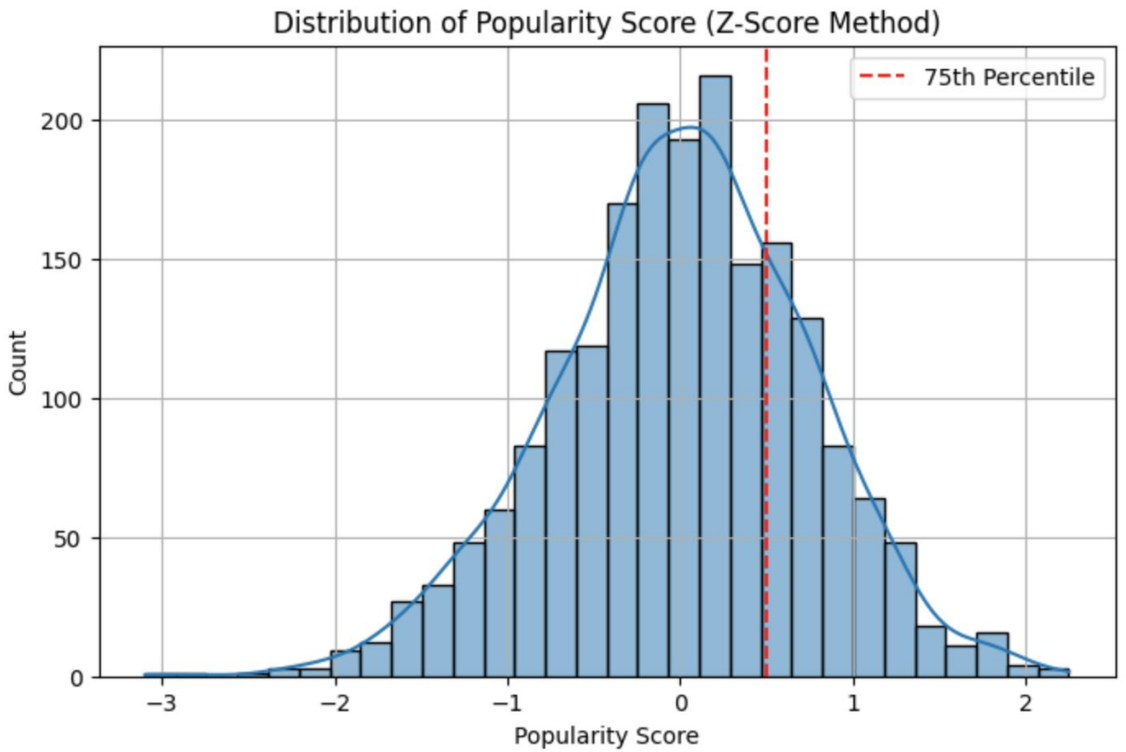


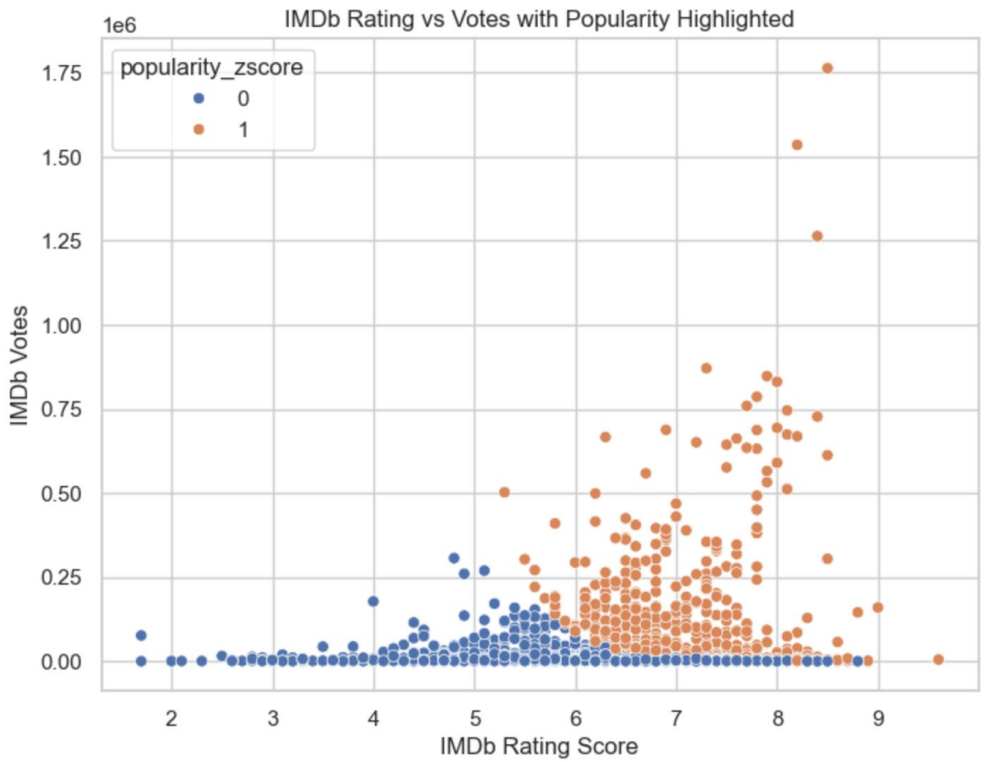


Insights: Most IMDb rating are between 4 and 8, and median of IMDb rating scores is around 6.8. The distribution of IMDb Ratings shows skewness. And The IMDb Votes shows extreme skewness.

Parameter Selection

Z-Score Combined Variables Method





Z-Score Combined Mehtod

This code calculates a "popularity score" for each title based on its **IMDb rating and number of votes**, and uses Z-score normalization to identify the top 25% most popular items.

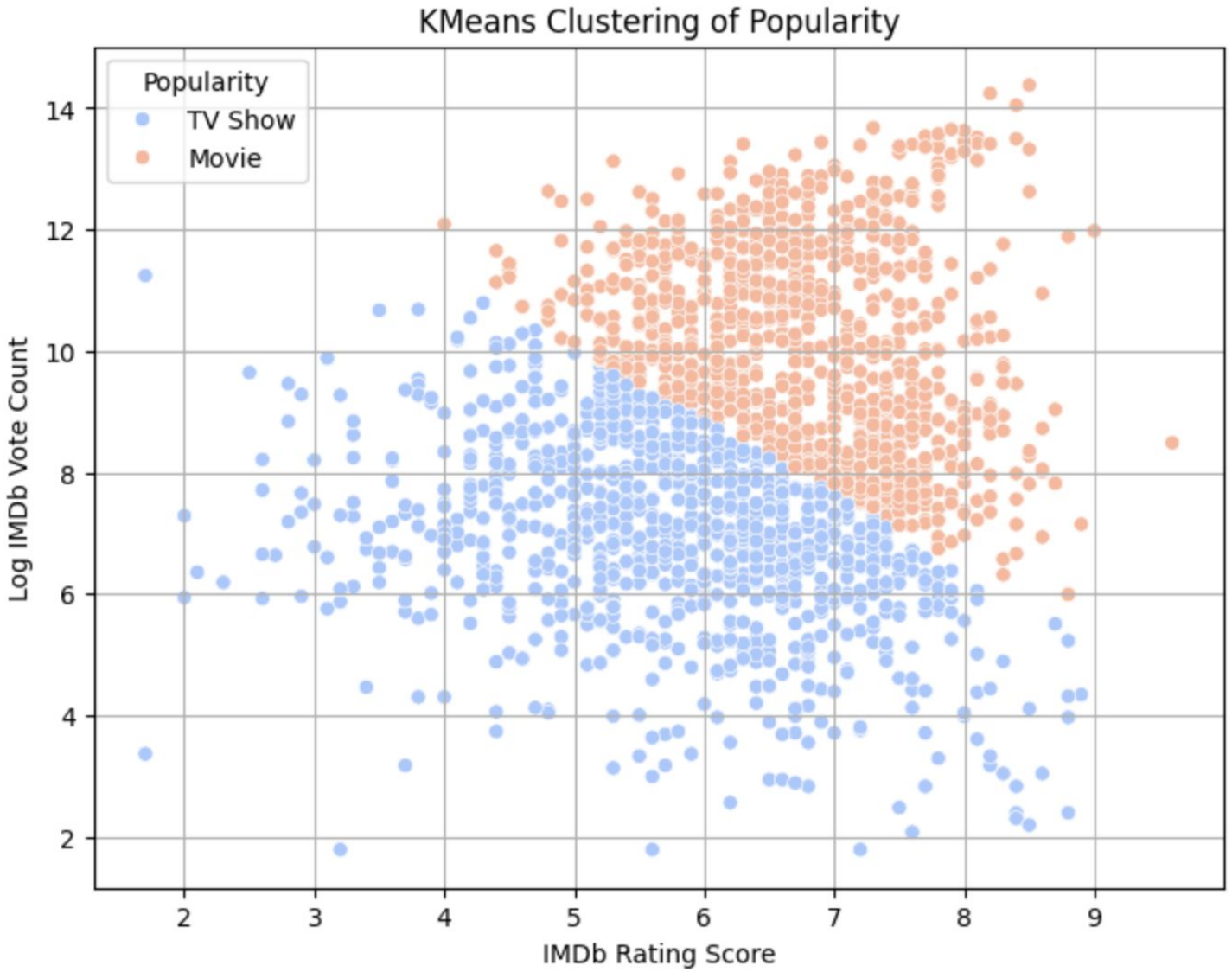
Weight: 0.5 and 0.5

Threshold: 75th quantile

```
# Composite score (weights can be adjusted, here both are 0.5)
df['popularity_score'] = 0.5 * df['rating_z'] + 0.5 * df['votes_z']
```

Most Netflix shows have IMDb ratings and vote counts below average, with low ratings indicating that viewers consider them average or poor.

Parameter Selection



TV Shows and Movies have distinct patterns in ratings and votes:

- TV shows tend to have higher ratings and receive more votes.
- Movies generally have lower ratings and fewer votes.

The features **imdbrating** and **log(votes)** behave differently across the two content types, indicating heterogeneous data distributions.

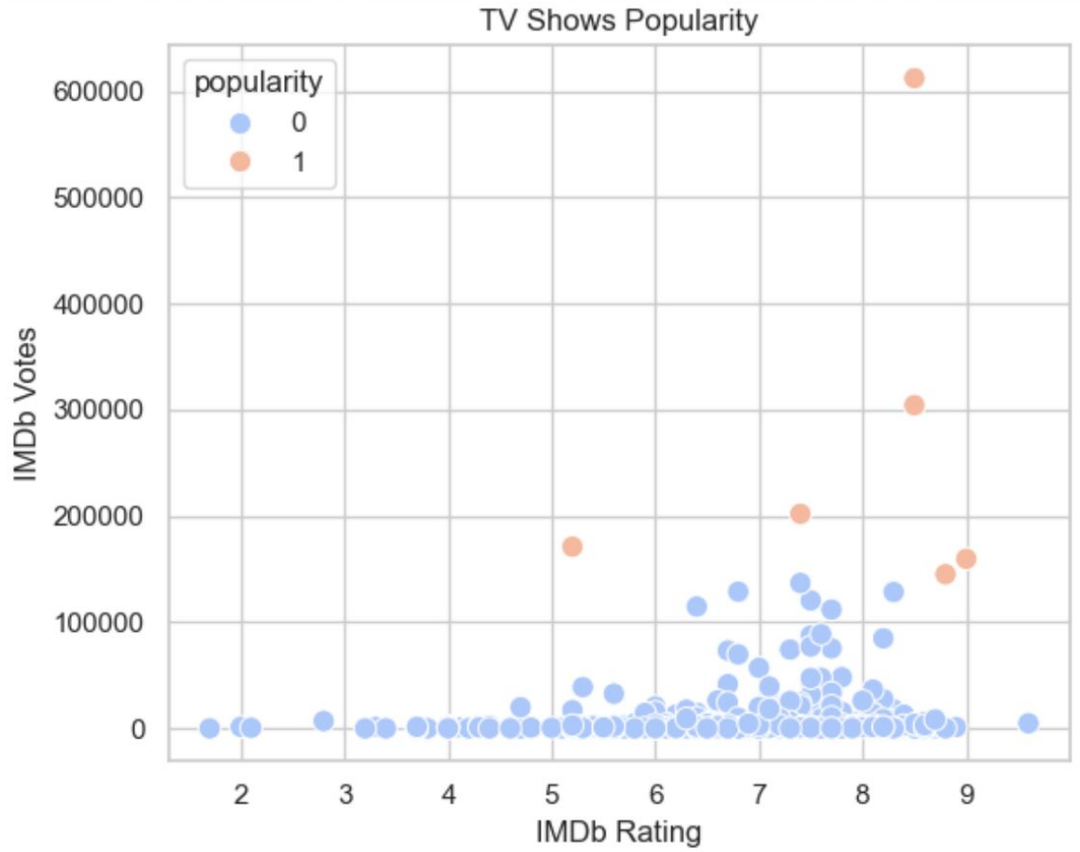
Model Strategy

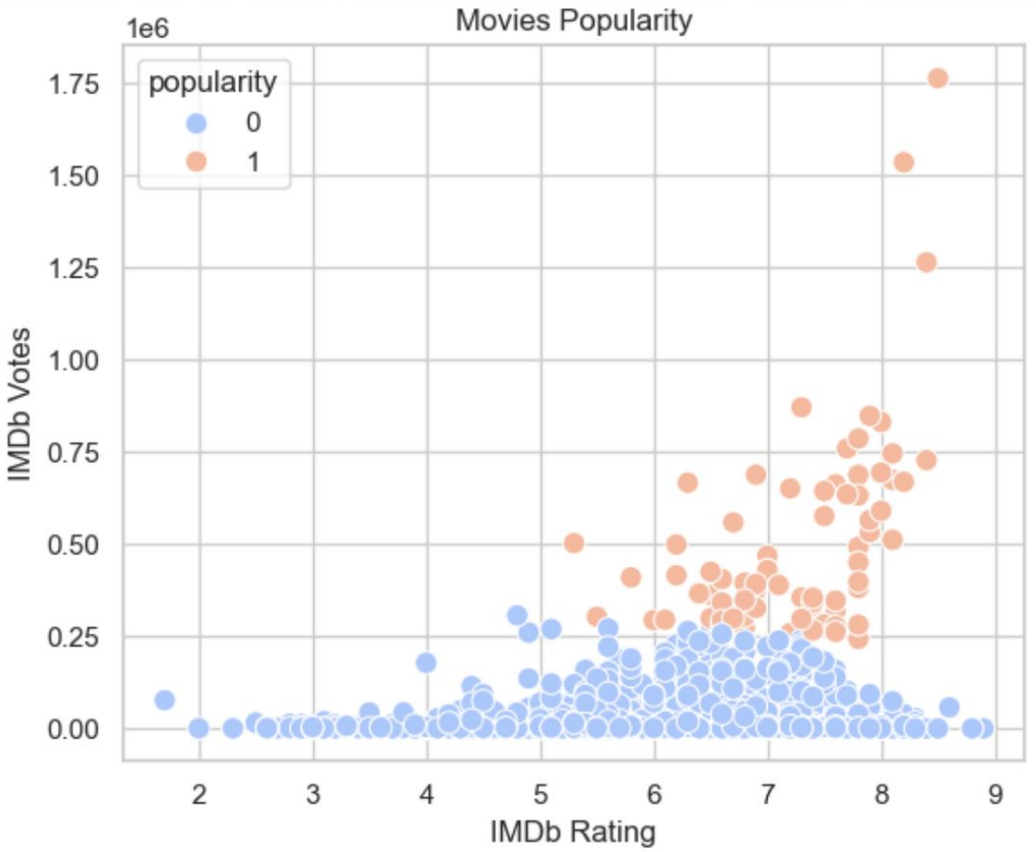
Choose thresholds of TV Shows and Movies' popularity separately

Separate thresholds may better capture the unique characteristics and patterns within each group.

Parameter Selection

K-means Clustering Method – To Naturally Classify



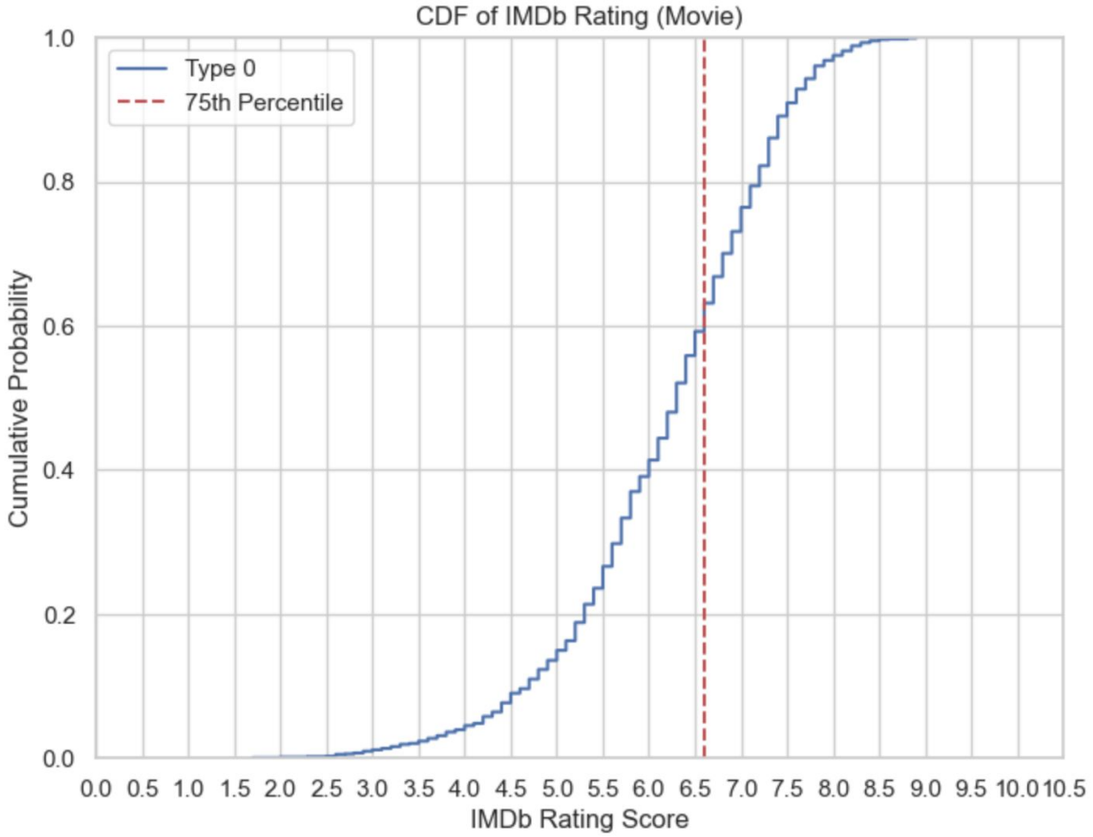


K-means clustering method amplifies the impact of votes, but our main focus is on obtaining high ratings. Votes are only used as an auxiliary indicator to ensure that high ratings are not from a small group of people or artificially inflated.

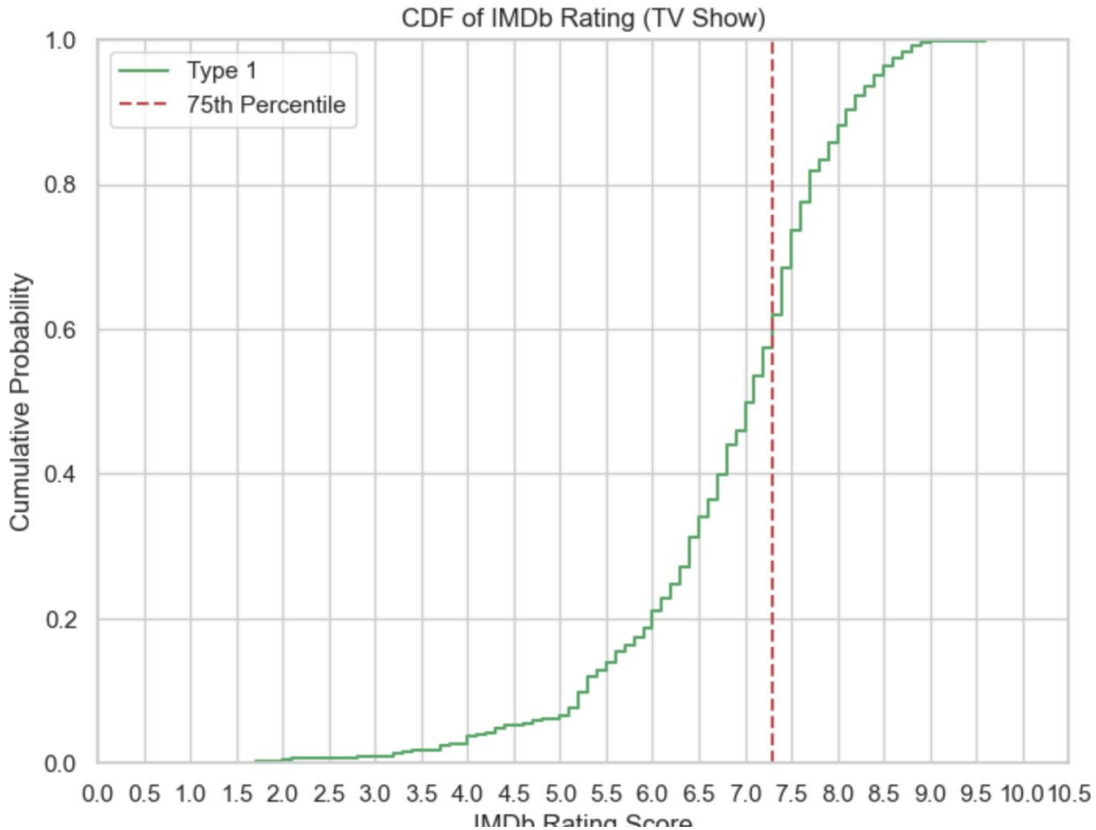
continue examining the CDF curve to see if I can find a natural inflection point to choose the parameters.

Parameter Selection

The distribution is relatively "uniform" or "linear," with no extreme concentration or abrupt changes. There is no obvious region of "high ratings concentration" or "low ratings concentration.” Ratings are fairly evenly distributed across the entire rating range and there is no natural "boundary" to distinguish between "popular" and "non-popular."

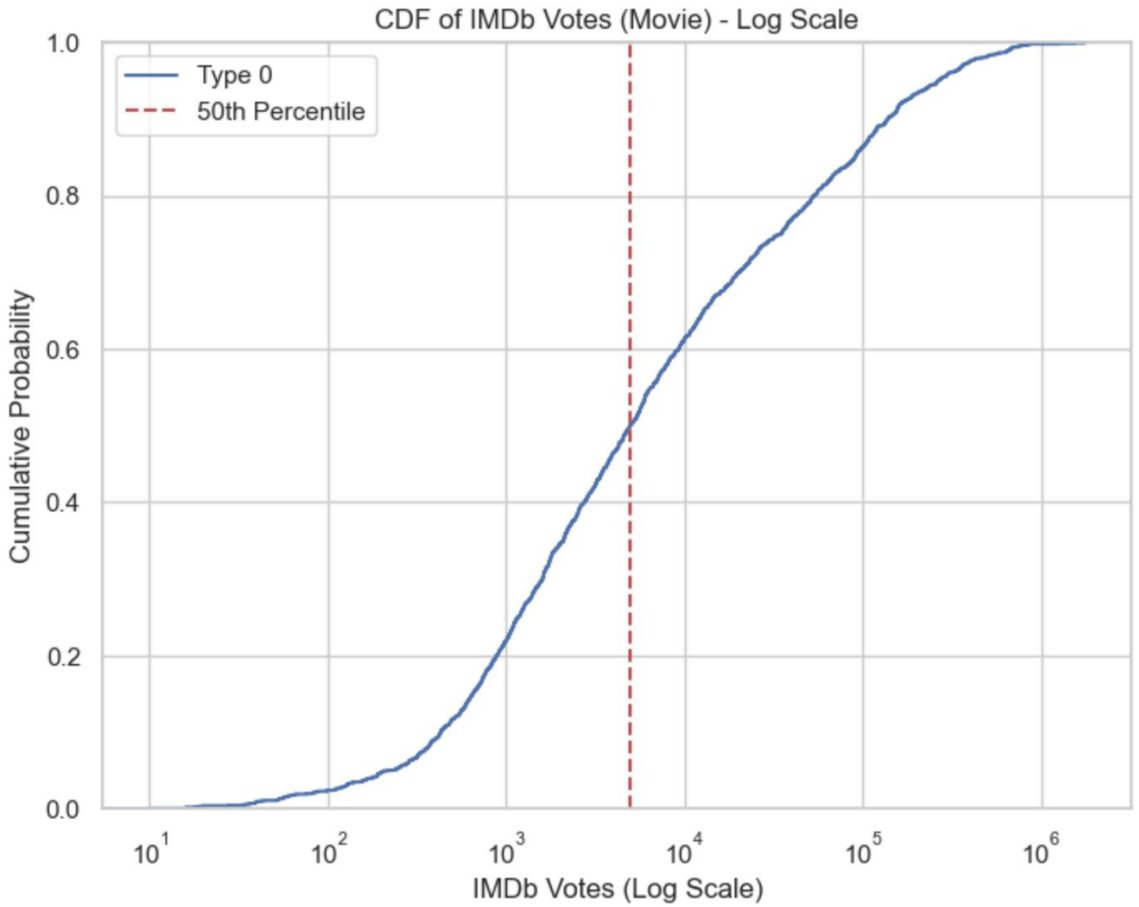


60th quantile of Movie’sIMDb ratings > 6.6

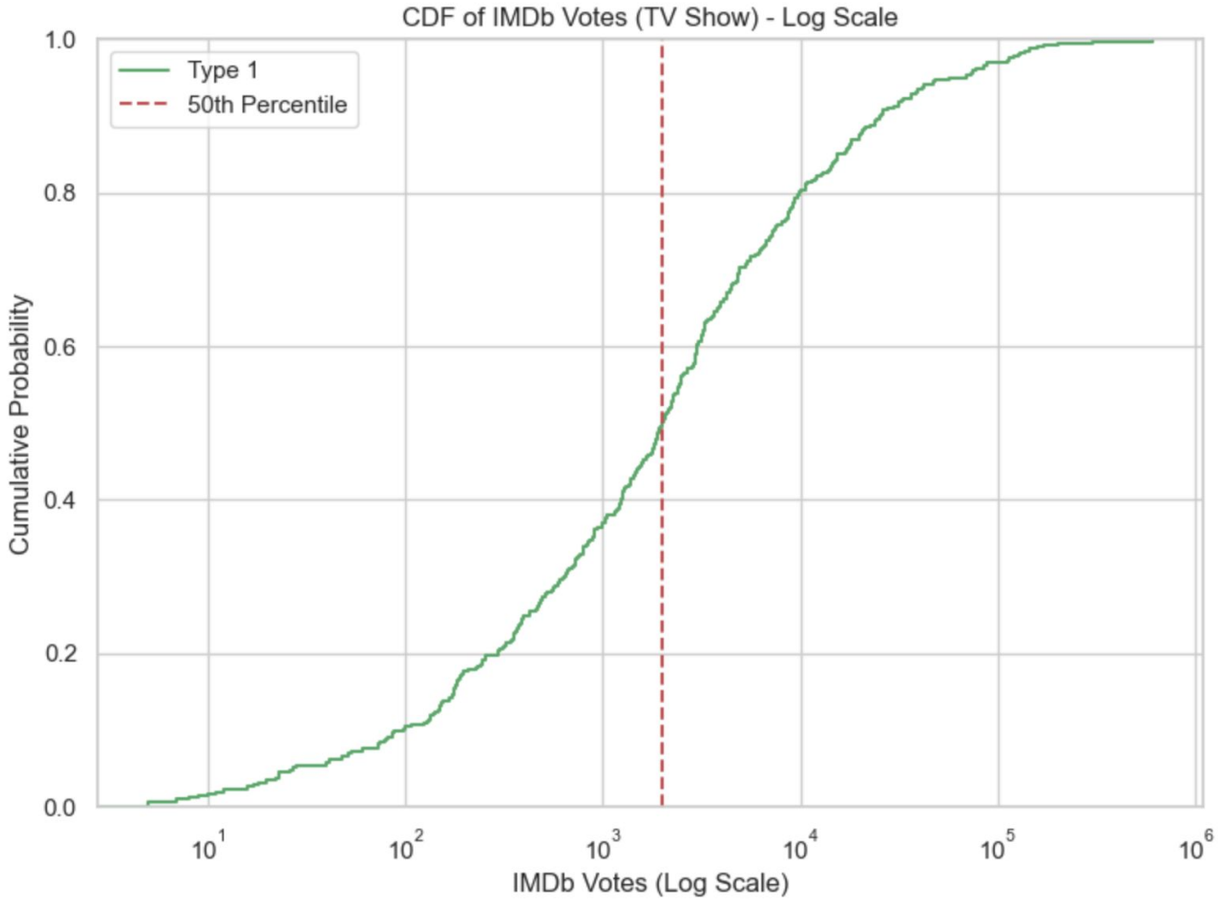


60th quantile of IMDb ratings > 7.3

Parameter Selection



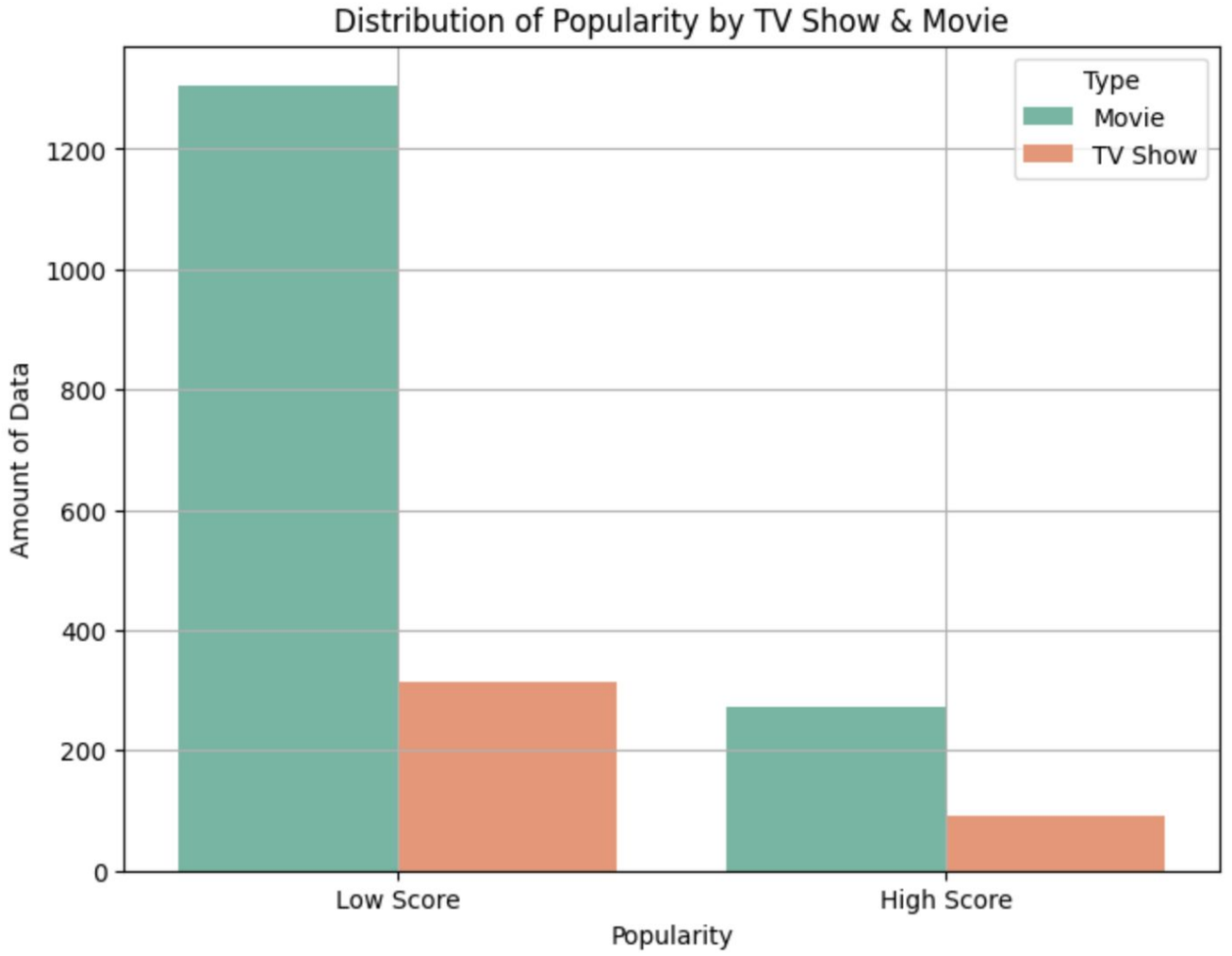
50% Movie's IMDb votes > 4993



50% TV show's IMDb votes > 2013

Parameter Selection

Quantile-based Method



- IMDb scores have a **clear definition**
- Threshold is based on the **actual data distribution** (e.g., “top 25% most popular” or “scores above a certain value”)
- Allowing **flexibility in adjusting the score threshold**.
- The proportion of items labeled as popular (popular = 1) shouldn’t be too few (e.g. should fall between 10% and 30%).

Final Parameters:

Vote threshold for movies: 4993.0

Vote threshold for TV shows: 2013.0

Rating threshold for movies (60th percentile): 6.6

Rating threshold for TV shows (60th percentile): 7.3

Target for model: popularity
0 for low score vs 1 for high score
Allow adjusting score thresholds

05

Prediction Model

Machine-Learning Modeling Approach

To predict a Netflix content's potential of getting high IMDB score

Our Challenge:

Predict which Netflix content will become popular based on various features which includes numerical data and text unstructured data.

Machine Learning Objective

Develop a robust binary classification model.

Target Variable (popularity):

1: Popular

0: Not Popular

Key Evaluation Metrics:

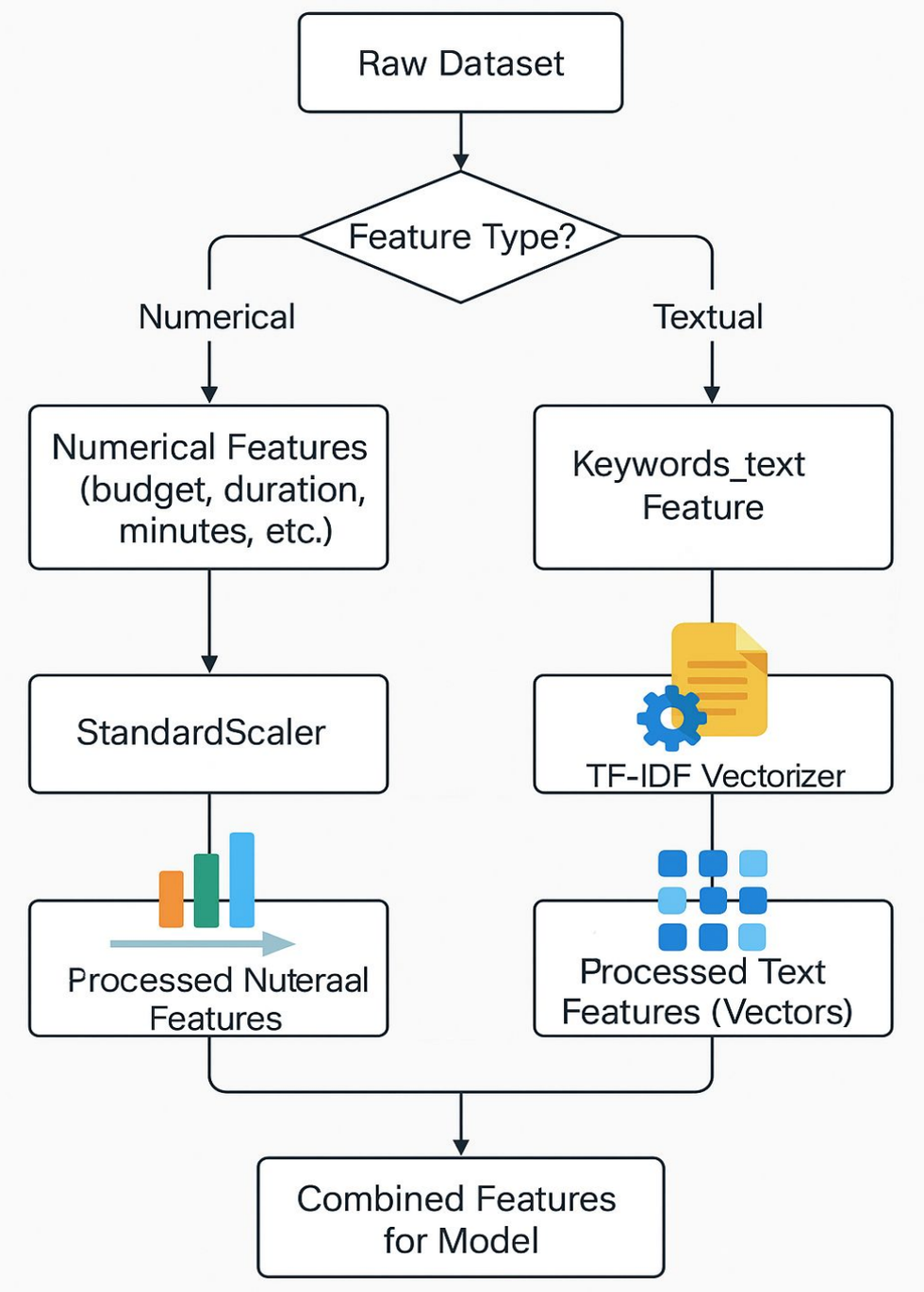
- Accuracy
- Precision (especially for the 'Popular' class)
- Recall (especially for the 'Popular' class)
- F1-Score (balances Precision and Recall, crucial for imbalanced datasets)

Overall Approach:

A data-driven modeling strategy, combining:

Structured metadata (e.g., budget, duration, type).

Natural Language Processing (NLP) features from keywords.



Feature Engineering & Data Preparation

1. Feature Set Overview

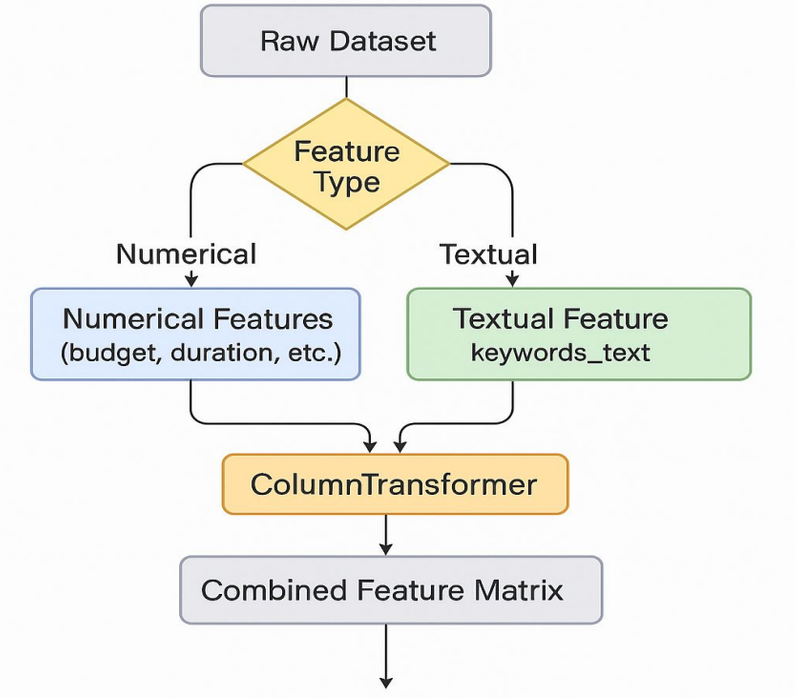
- **Numerical Features:** budget, production_company_track_record, director_success, cast_popularity, marketing_budget, release_year, duration_minutes, duration_seasons, and pre-encoded categoricals like type, rating, release_season, genre_trend. **(12 Dimensions)**
- **Textual Feature for NLP:** keywords_text (derived from content keywords). **(200+ Dimensions)**

2. NLP for Keywords (keywords_text)

- The keywords column (potentially string representations of lists) was processed.
- ast.literal_eval used to safely convert string-lists to Python lists.
- Keywords joined into a single text string per title for vectorization.
- **TF-IDF Vectorization:** TfidfVectorizer(max_features=200, stop_words='english')
- converted these keyword strings into 200 **numerical features**, emphasizing important terms.

3. Preprocessing with Column Transformer

- Numerical Features: Scaled using StandardScaler() to have zero mean and unit variance. This helps algorithms sensitive to feature magnitudes.
- Text Feature (keywords_text): Processed by the TfidfVectorizer as described above.
- Benefit: ColumnTransformer applies these distinct preprocessing steps efficiently to the correct feature subsets.



Handling Class Imbalance & Modeling Pipeline Structure

Addressing Class Imbalance

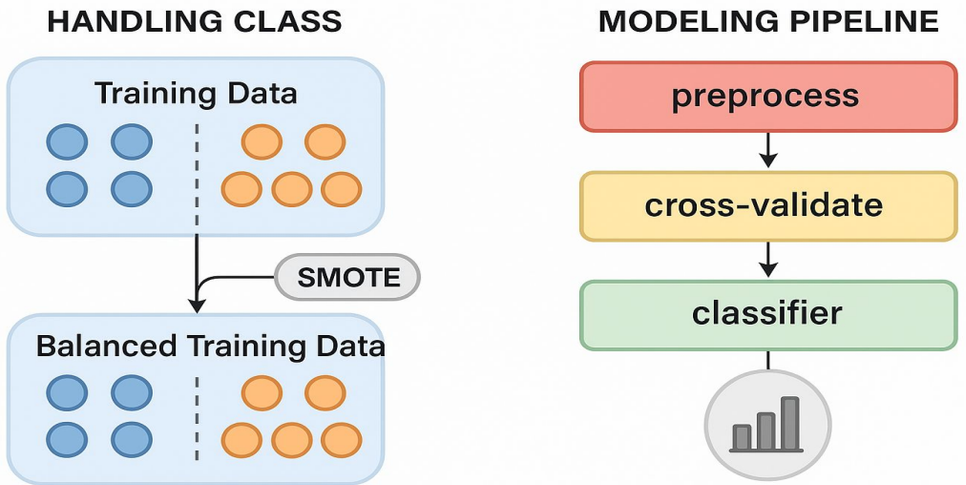
- Initial analysis of the target variable popularity in the training set showed an imbalance: [X_train popular %]% Popular vs. [X_train not popular %]% Not Popular.
- **SMOTE** (Synthetic Minority Over-sampling Technique): Applied to the training data to create synthetic samples for the minority class, balancing the dataset.
- **class_weight='balanced':** Used in both models to further adjust for class imbalance by giving more weight to the minority class during training.

Integrated Modeling Pipeline (ImbPipeline from imblearn)

This pipeline ensures a correct workflow:

- **preprocess:** Applies ColumnTransformer (Standardization for numerical, TF-IDF for text).
- **smote:** Performs SMOTE oversampling only on the training data within each cross-validationfold or during final training. This prevents data leakage.
- **classifier:** Trains the specified machine learning model (Logistic Regression or Random Forest).

```python
# --- Define Models ---
# Model 1: Logistic Regression
# Using ImbPipeline to include SMOTE step specifically for training data
lr_pipeline = ImbPipeline([
    ('preprocess', preprocessor),
    ('smote', SMOTE(random_state=42)), # Apply SMOTE to handle imbalance
    ('classifier', LogisticRegression(class_weight='balanced', # Also use class_weight for robustness
                                       max_iter=1000,
                                       random_state=42,
                                       solver='liblinear')) # Good solver for smaller datasets
])

# Model 2: Random Forest
rf_pipeline = ImbPipeline([
    ('preprocess', preprocessor),
    ('smote', SMOTE(random_state=42)), # Apply SMOTE
    ('classifier', RandomForestClassifier(class_weight='balanced', # Use class_weight
                                           random_state=42,
                                           n_estimators=150, # Example: Increase estimators
                                           max_depth=10,     # Example: Limit tree depth
                                           min_samples_leaf=5)) # Example: Ensure leaves have min samples
])
```



Models Implemented & Configuration

1. Logistic Regression Model

A linear model that predicts the probability of an outcome.
Configuration from script:

class_weight='balanced'
max_iter=1000
solver='liblinear' (suitable for this dataset size and binary classification)
random_state=42

2. Random Forest Classifier Model

An ensemble model using multiple decision trees for robust predictions.
Configuration from script:

class_weight='balanced'
n_estimators=150 (number of trees)
max_depth=10 (limits tree complexity to prevent overfitting)
min_samples_leaf=5 (minimum samples at a leaf node)
random_state=42

Why Two Models? Benchmarking & Diverse Approaches:

- **Logistic Regression (Baseline):** Serves as a strong, interpretable baseline. It helps understand linear relationships between features and popularity.
- **Random Forest (Complex Relationships):** Chosen for its ability to capture more complex, non-linear patterns and interactions between features that a linear model might miss.
- Comparing these two helps us understand if the added complexity of Random Forest provides a significant performance benefit for this specific problem.

Models Implemented & Configuration

Model Architecture

- Dual-model approach: Logistic Regression vs. Random Forest
- Integrated preprocessing and sampling within model pipeline
- Class weighting to handle imbalanced data

Table: Model Perfomance Comparison

Model Type	Logistic Regression	Random Forest
Accuracy	0.68	0.81
Precision (Popularity)	0.34	0.52
Recall (Popularity)	0.65	0.37
F1-score (Popular):	0.45	0.43

Key Insight:

Random Forest Model delivered higher overall accuracy **(81%)**, highlighting the complex non-linear relationships in content popularity factors

06

Result & Model Evaluation

Result: Feature Importance

Uncovering the Drivers of Content Popularity

Discoveries

- **Content characteristics** far outweigh production metrics in predictive power
- **Duration** is the single strongest predictor - optimal length appears critical for engagement
- **Specific thematic elements** (represented by keywords) have significant influence
- **Traditional industry metrics** (budgets, cast popularity) have lower predictive value than expected

Business Application

- Content acquisition teams should prioritize assessment of **duration, themes,** and **rating over production company reputation or star power**

Key Features !

```
--- Example Predictions ---
                    title  actual_popularity  predicted_popularity (RF)  predicted_probability (RF)
0    aliens ate my homework                  0                          0                    0.352644
1    code name: the cleaner                  0                          0                    0.467038
2               get shorty                   1                          0                    0.489744
3              sweet tooth                   1                          0                    0.322005
4  honey: rise up and dance                  0                          0                    0.425367

--- Random Forest Feature Importance ---
Top 20 Important Features:
                            feature  importance
6                  duration_minutes    0.109432
38                            finds    0.084408
9                            rating    0.077790
47                             help    0.074108
110                           young    0.055227
5                     release_year    0.054101
39                           follows    0.045968
37                              film    0.044853
46                             group    0.027255
1    production_company_track_record    0.027218
2                  director_success    0.026721
0                            budget    0.026174
4                 marketing_budget    0.026119
10                  release_season    0.024165
3                  cast_popularity    0.021595
54                             life    0.021384
98                             true    0.019050
7                  duration_seasons    0.012253
65                              new    0.012032
```

Business Impact

KEY INSIGHTS

- Duration metrics outweigh production indicators in predicting popularity
- Theme highly impacts content performance

APPLICATIONS

- Assess content acquisition with predictive factors
- Guide script elements and duration in pre-release
- Prioritize promotional strategies

Model Training & Evaluation Strategy

Data Split:
Data was split **into 80% Training and 20% Testing sets**.
stratify=y was used to ensure similar class proportions in both sets.
Used cross-validation (CV=5) for reliability

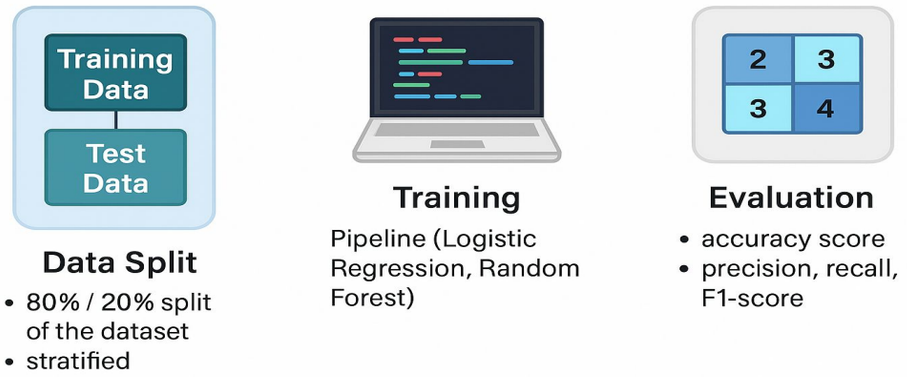
Training:
Each ImbPipeline (Logistic Regression and Random Forest) was trained using pipeline.fit(X_train, y_train).

Evaluation (on Test Set):
Predictions made using pipeline.predict(X_test).

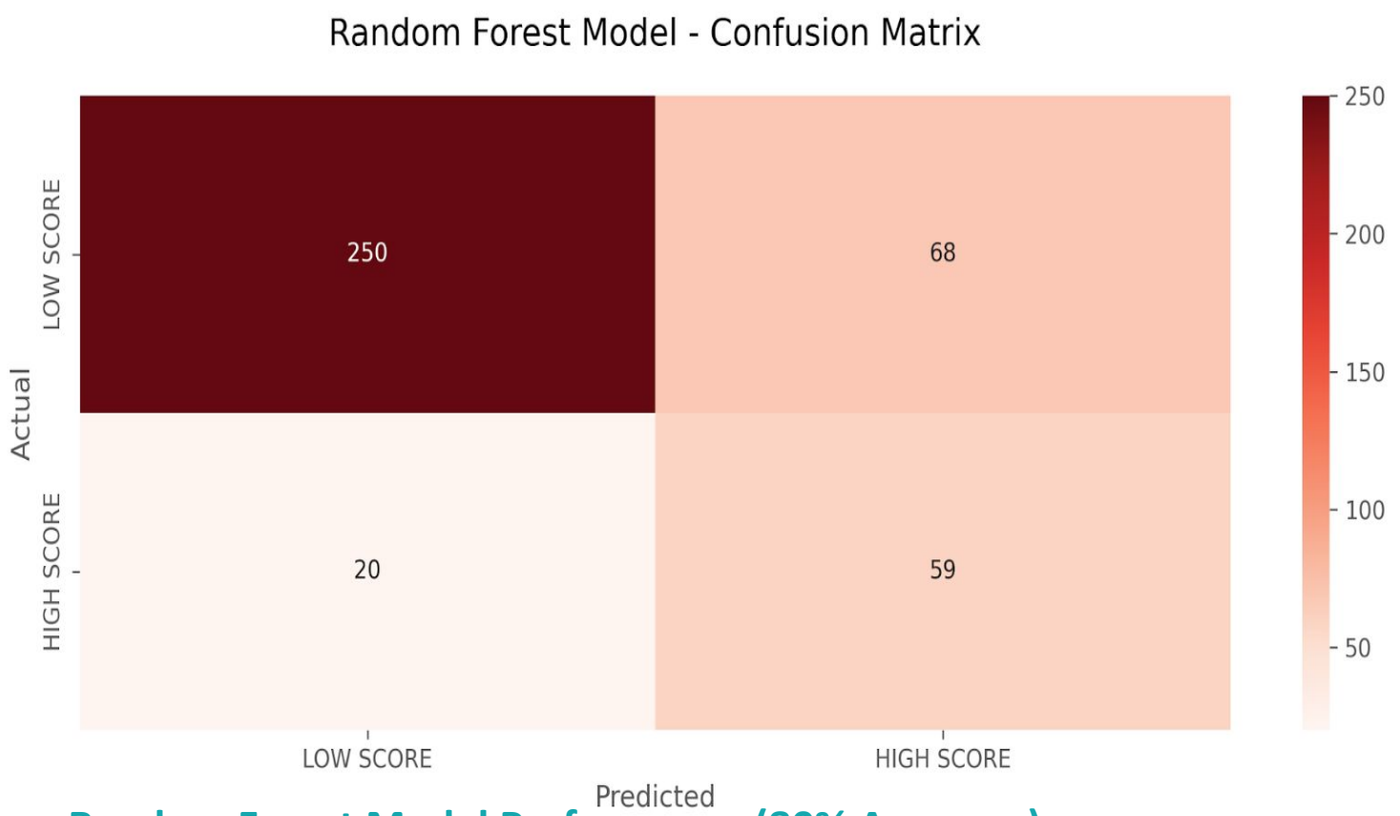
Metrics calculated:

- accuracy_score
- Classification report (Precision, Recall, F1-score per class) confusion_matrix

```python
models = {
    "Logistic Regression": lr_pipeline,
    "Random Forest": rf_pipeline
}

for name, pipeline in models.items():
    print(f"\n--- Training and Evaluating {name} ---")

    # Train the model
    pipeline.fit(X_train, y_train)

    # Make predictions on the TEST set
    y_pred = pipeline.predict(X_test)
    y_pred_proba = pipeline.predict_proba(X_test)[:, 1] # Probability of class 1 (popular)

    # Evaluate
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy: {accuracy:.4f}")
    print("\nClassification Report:")
    # Use target_names for better report readability
    print(classification_report(y_test, y_pred, target_names=['Not Popular (0)', 'Popular (1)']))
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_test, y_pred))

# --- 7. Example Prediction (Optional) ---
```



Model Evaluation




```
--- Training and Evaluating Logistic Regression ---
Accuracy: 0.6801

Classification Report:
                precision    recall  f1-score   support

Not Popular (0)       0.89      0.69      0.78       318
    Popular (1)       0.34      0.65      0.45        79

       accuracy                           0.68       397
      macro avg       0.61      0.67      0.61       397
   weighted avg       0.78      0.68      0.71       397


Confusion Matrix:
[[219  99]
 [ 28  51]]

--- Training and Evaluating Random Forest ---
Accuracy: 0.8060

Classification Report:
                precision    recall  f1-score   support

Not Popular (0)       0.85      0.92      0.88       318
    Popular (1)       0.52      0.37      0.43        79

       accuracy                           0.81       397
      macro avg       0.69      0.64      0.66       397
   weighted avg       0.79      0.81      0.79       397


Confusion Matrix:
[[291  27]
 [ 50  29]]
```

Random Forest Model Performance (80% Accuracy)

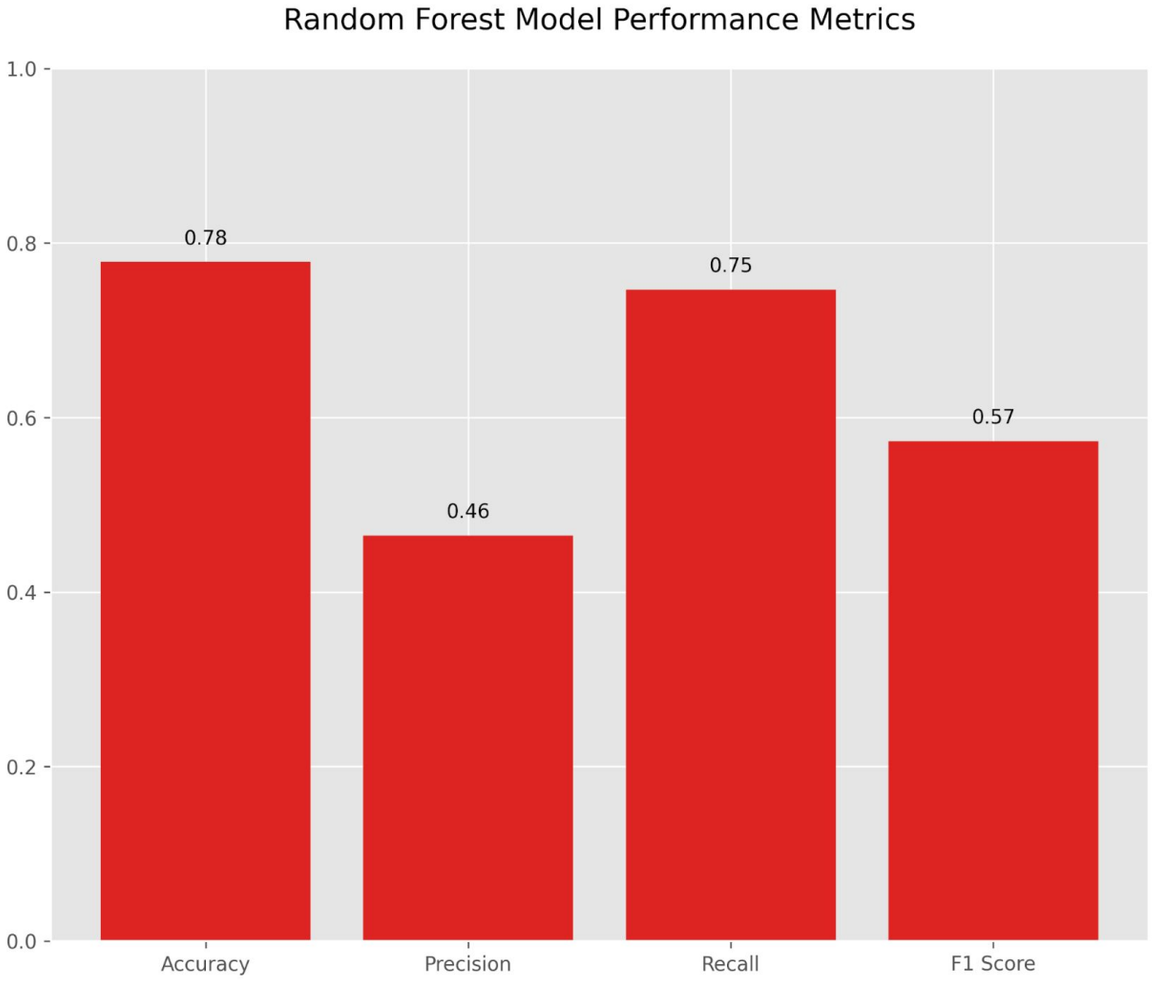
True Negatives (250): Correctly predicted LOW SCORE shows **False Positives (68):** Incorrectly predicted HIGH SCORE shows

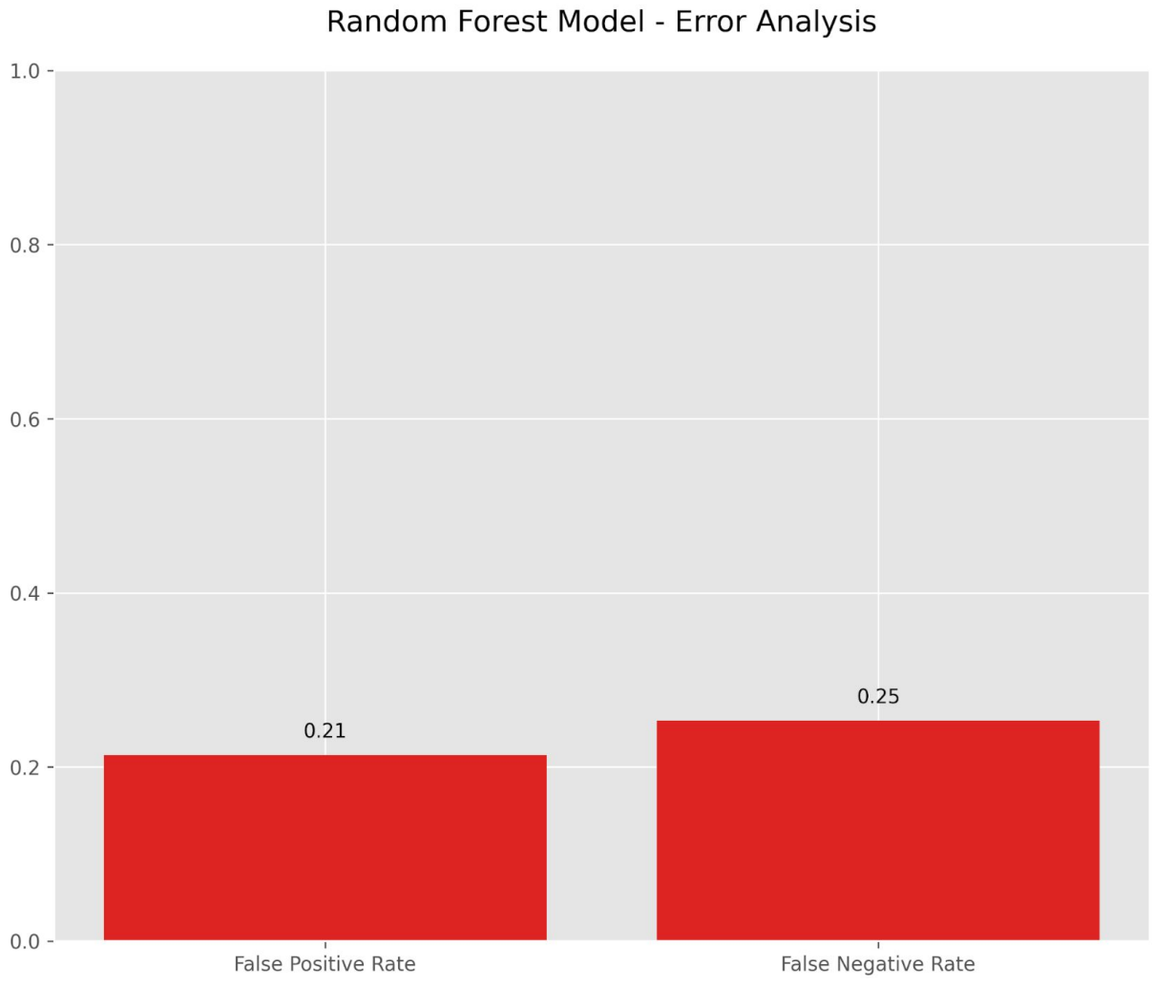
False Negatives (20): Incorrectly predicted LOW SCORE shows **True Positives (59):** Correctly predicted HIGH SCORE shows

Total Shows: 397

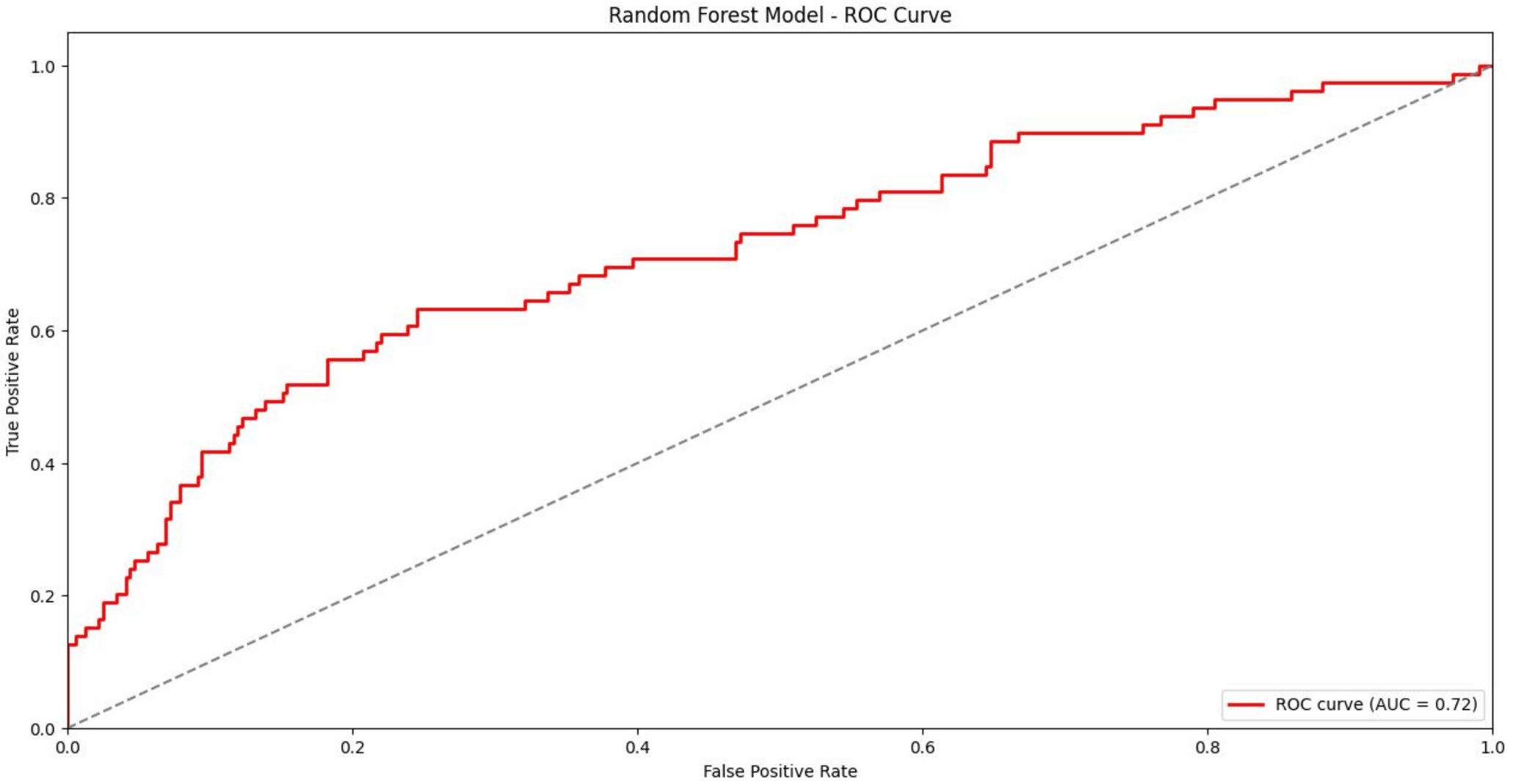
Accuracy: 80.10%

Model Evaluation





Model Evaluation



Conclusion & Future Work

Conclusion

- Predicts show if a TV show or Movie can get high score in IMDB or not before release using only **production-side metadata** — no audience feedback, no user ratings is possible.
- Random Forest model achieves **81% accuracy** despite challenging class imbalance.
- **Content characteristics** (duration, themes) outweigh industry metrics (cast popularity, budgets), Duration, themes, rating over production company reputation and star power are top features has higher influence in industry features.
- Model correctly identified **92% of non-popular content** (high specificity). **(We can accurately predict if a show will get low score!)**
- Opportunity to improve prediction of popular content (currently 37% recall).
- Our model empowers more confident greenlighting and acquisition decisions by focusing on **core content drivers** like optimal duration and impactful themes, rather than just budget or star power, to de-risk investments.

Business Application

This predictive model is a powerful tool for business-wise decision-making, especially for producers in early development. It offers a data-driven way to assess potential popularity using pre-release metadata, guiding choices on script elements and content structure before major resources are committed.

Future Work

Potential Enhancements

1. **NLP Further Analysis:**
    - o Show description embeddings
    - o Genre keyword analysis
    - o Sentiment analysis from tweets or related news
2. **Model Improvements:**
    - o Ensemble methods (Random Forest, XGBoost)
    - o Hyperparameter tuning with cross-validation
3. **Advanced Feature Engineering:**
    - o Temporal trends (seasonality, release date)
    - o Social media metrics
    - o Content similarity clustering

Next Steps

- Experiment with deep learning for better keyword representation.
- Incorporate additional features: viewer demographics, social media sentiment.
- Develop separate models for movies vs. series (different success patterns).
- Create interactive prediction tool for content acquisition team.