IMDb Movie Rating Prediction - Documentation

# 1. Project Overview

The goal of this project is to build a machine learning model to predict IMDb movie ratings using a dataset of Indian movies. The predictions are based on various features such as genre, director, release year, and historical success metrics.

# 2. Dataset Description

Source: IMDb Movies India dataset (CSV format)

|  |  |
| --- | --- |
| Column | Description |
| Genre | Movie genre(s) |
| Director | Name of the director |
| Year | Year the movie was released |
| Rating | IMDb rating (target variable) |

# 3. Data Preprocessing

## 3.1 Missing Value Handling

- Rows with missing Rating values were removed.  
- Missing values in Genre and Director were filled with 'Unknown'.  
- Missing values in the Year column were filled with the mode (or mean if mode is empty).

## 3.2 Categorical Encoding

LabelEncoder was applied to Genre and Director.

## 3.3 Feature Engineering

Two custom features were added to improve prediction:  
- director\_success\_rate: Average rating of all movies by the same director.  
- genre\_avg\_rating: Average rating of all movies in the same genre.

# 4. Model Development

## 4.1 Algorithm Used

Random Forest Regressor (sklearn.ensemble.RandomForestRegressor)  
Justification: It handles non-linear data, works well with mixed data types, and is robust to overfitting.

## 4.2 Data Splitting and Scaling

- 80% training, 20% testing using train\_test\_split.  
- StandardScaler was used to normalize features before training.

# 5. Evaluation Metrics

- Mean Squared Error (MSE): Measures average squared difference between predicted and actual ratings.  
- R² Score (R-squared): Indicates how well the model explains the variance in the data.  
Example Output:  
Mean Squared Error: 0.412  
R² Score: 0.82

# 6. Repository Structure

Movie-Rating-Prediction/  
├── data/  
│ └── IMDb Movies India.csv  
├── notebooks/  
│ └── movie\_rating\_prediction.ipynb  
├── models/  
│ └── (Optional: saved models)  
├── README.md  
├── documentation.md  
└── requirements.txt

# 7. Future Enhancements

- Add more features (cast, budget, runtime, language, etc.)  
- Use NLP on movie descriptions or titles  
- Experiment with advanced models (XGBoost, LightGBM, Neural Networks)  
- Add cross-validation and hyperparameter tuning  
- Deploy as a web app for real-time predictions

# 8. Dependencies (requirements.txt)

pandas  
numpy  
matplotlib  
seaborn  
scikit-learn  
  
Run: pip install -r requirements.txt

Code:

# Movie Rating Prediction

# Import libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, r2\_score

# Load dataset

df = pd.read\_csv('IMDb Movies India.csv', encoding='latin-1')

# Display basic info

print(df.head())

print(df.info())

# Drop rows with missing ratings

df = df.dropna(subset=['Rating'])

# Handle missing values

df['Genre'].fillna('Unknown', inplace=True)

df['Director'].fillna('Unknown', inplace=True)

# Encode categorical features

label\_encoders = {}

for col in ['Genre', 'Director']:

    le = LabelEncoder()

    df[col] = le.fit\_transform(df[col])

    label\_encoders[col] = le

# Feature engineering: add year

df['Year'] = pd.to\_numeric(df['Year'], errors='coerce')

df['year'] = df['Year']

# Handling missing year values

if not df['year'].mode().empty:

    df['year'].fillna(df['year'].mode()[0], inplace=True)

else:

    df['year'].fillna(df['year'].mean(), inplace=True)

# Feature engineering: Director success rate and Genre average rating

# Director success rate: average rating per director

director\_avg\_rating = df.groupby('Director')['Rating'].mean().to\_dict()

df['director\_success\_rate'] = df['Director'].map(director\_avg\_rating)

# Genre average rating: average rating per genre

genre\_avg\_rating = df.groupby('Genre')['Rating'].mean().to\_dict()

df['genre\_avg\_rating'] = df['Genre'].map(genre\_avg\_rating)

# Final feature set

features = ['Genre', 'Director', 'year', 'director\_success\_rate', 'genre\_avg\_rating']

X = df[features]

y = df['Rating']

# Scale features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

# Train model

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

# Predict

y\_pred = model.predict(X\_test)

# Evaluate

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print("Mean Squared Error:", mse)

print("R² Score:", r2)

# Plot actual vs predicted

plt.figure(figsize=(8, 6))

plt.scatter(y\_test, y\_pred, alpha=0.7)

plt.xlabel("Actual Ratings")

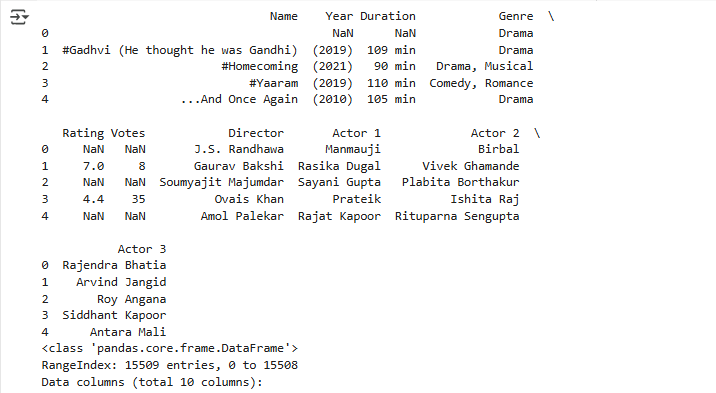
plt.ylabel("Predicted Ratings")

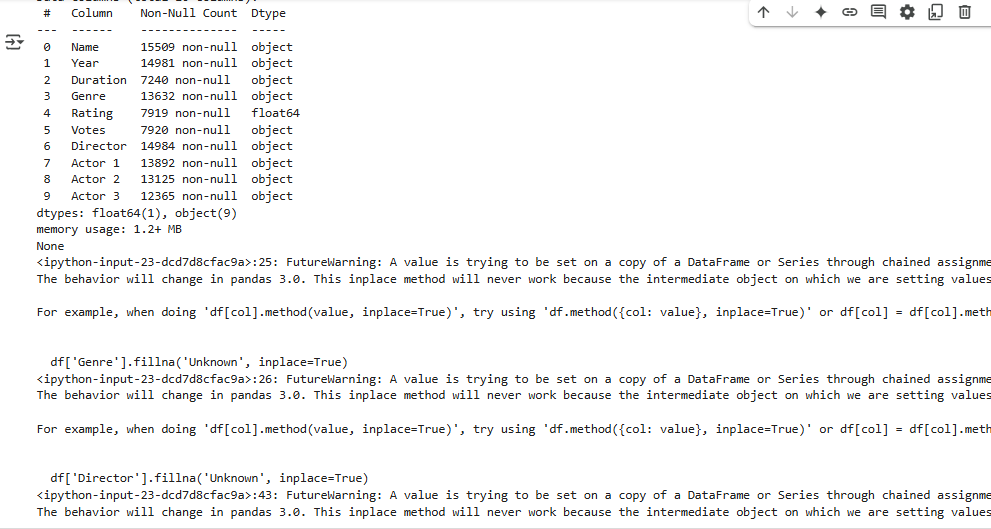
plt.title("Actual vs Predicted Ratings")

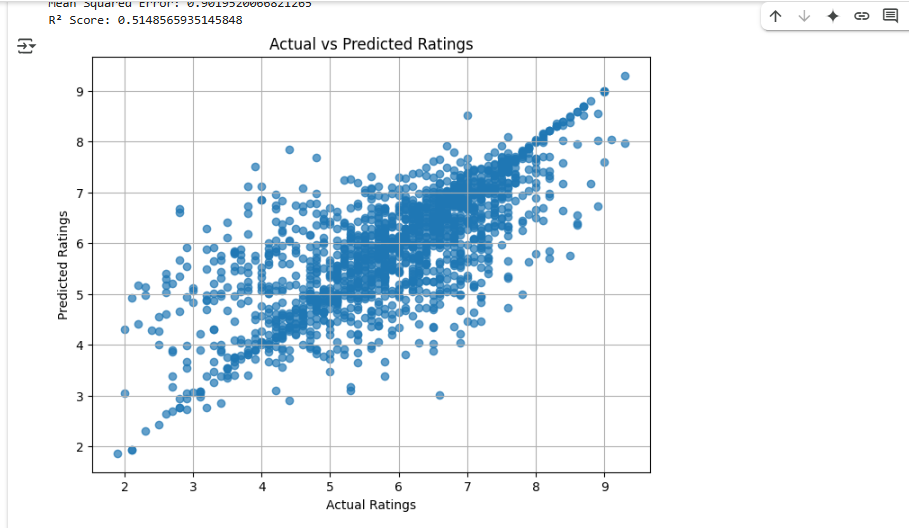
plt.grid(True)

plt.show()

Output:







# 9. Contact

Author: Vipul Jadhav  
Email: vj486777@gmail.com  
GitHub: