Movie Rating Prediction with Python

Project Overview

The Movie Rating Prediction project delves into the world of movie ratings, aiming to predict audience reception through machine learning. By analyzing historical data, the project seeks valuable insights to aid filmmakers in understanding what factors contribute to a movie's success.

Data Source:

- Kaggle dataset: "IMDb India Movies" - This dataset provides a comprehensive look at Indian movies on IMDb, including details like genre, director, actors, and ratings.

Project Methodology:

1. **Data Preprocessing:** The project begins by cleaning and preparing the data from the chosen dataset for further analysis.
2. **Exploratory Data Analysis (EDA):** Through visualizations, the project explores the data to identify patterns and trends that might influence movie ratings.
3. **Feature Importance:** This step focuses on determining which factors (genre, director, actors, etc.) have the most significant impact on how audiences rate movies.
4. **Predictive Modeling:** Using machine learning techniques, a model is built to predict movie ratings based on the identified influential features.
5. **Model Evaluation:** The project then assesses the model's performance to gauge its prediction accuracy.
6. **Interpretation of Results:** Finally, the project dives into understanding the model's predictions and how different factors affect movie ratings, providing valuable insights.

Technical Environment:

- Google Colab Notebook is the platform used for data analysis, visualizations, and project development.

This project utilizes the power of machine learning for predicting movie ratings. By understanding the factors that influence audience reception, filmmakers can gain valuable knowledge to create movies that resonate with viewers.

Movie Rating Prediction with Python

Introduction:

This project explores predicting movie ratings using machine learning. Here I will analyze a dataset of Indian movies from Kaggle to identify factors influencing audience reception and build a model to predict movie ratings. By understanding these factors, filmmakers can gain valuable insights for creating successful movies.

Objective:

The objective of this project can be two-fold, focusing on both prediction and understanding:

1. **Prediction:** To develop a machine learning model that can accurately forecast movie ratings for Indian films. This allows stakeholders like filmmakers, studios, and distributors to anticipate audience reception even before a movie's release.
2. **Understanding:** To identify the key ingredients that drive audience ratings for Indian movies. This involves uncovering which aspects, like genre, director, or actors, have the strongest influence on viewers. By understanding these factors, filmmakers can make informed decisions while crafting movies that resonate with the target audience.

Scope:

The scope of this project centers on predicting movie ratings for Indian films using machine learning and analyzing the factors that influence those ratings.

Focus:

- **Prediction:** Building a machine learning model to estimate user ratings for Indian movies based on available data points.

Data Source:

- Leveraging the "IMDb India Movies" dataset from Kaggle, which includes details like genre, director, actors, and ratings.

Analysis Techniques:

- Exploratory Data Analysis (EDA) to uncover trends and patterns within the data.

- Feature engineering to identify the most impactful factors on movie ratings.

Model Development:

- Constructing a machine learning model using techniques suitable for rating prediction.

Evaluation and Interpretation:

- Rigorously testing the model's accuracy in predicting movie ratings.
- Analyzing the model's predictions to understand the influence of different factors on ratings.

Deliverables:

- A functional machine learning model for predicting movie ratings.
- Insights into the key factors that drive movie ratings in the Indian market.

Limitations:

- The model's accuracy will be dependent on the quality and completeness of the data used.
- The project focuses on Indian movies and might not be generalizable to other film industries.

Requirement Analysis:

This analysis outlines the essential elements needed to successfully complete the Movie Rating Prediction Project.

1. Data Requirements:

- **Data Source:** The project will utilize the "IMDb India Movies" dataset from Kaggle. This dataset should encompass information about Indian movies, including:
    - Movie title
    - Year of release
    - Duration (length of the movie)
    - Genre(s)
    - User rating

    - Director name
    - Actor names
- **Data Quality:** The data should be inspected for missing values, inconsistencies, and outliers. Techniques for data cleaning and preprocessing will be necessary.

2. Software Requirements:

- **Programming Language:** Python is a popular choice for machine learning projects due to its extensive libraries and community support. Libraries like pandas, NumPy, scikit-learn, and TensorFlow/PyTorch might be required for data manipulation, analysis, and model building.
- **Development Environment:** A platform like Google Colab Notebook facilitates data analysis, visualization, and model development in the cloud.

3. Hardware Requirements:

- While the project can potentially run on a personal computer, sufficient RAM and processing power are recommended for efficient data processing and model training, especially if dealing with large datasets. Cloud platforms like Google Colab can alleviate hardware limitations.

4. Model Requirements:

- **Model Type:** The project will explore machine learning models suitable for rating prediction tasks. Regression algorithms like Linear Regression or Random Forest Regression are potential candidates.

5. Evaluation Requirements:

- **Metrics:** Metrics like Mean Squared Error (MSE) or Root Mean Squared Error (RMSE) will be used to assess the model's accuracy in predicting movie ratings.

6. Reporting Requirements:

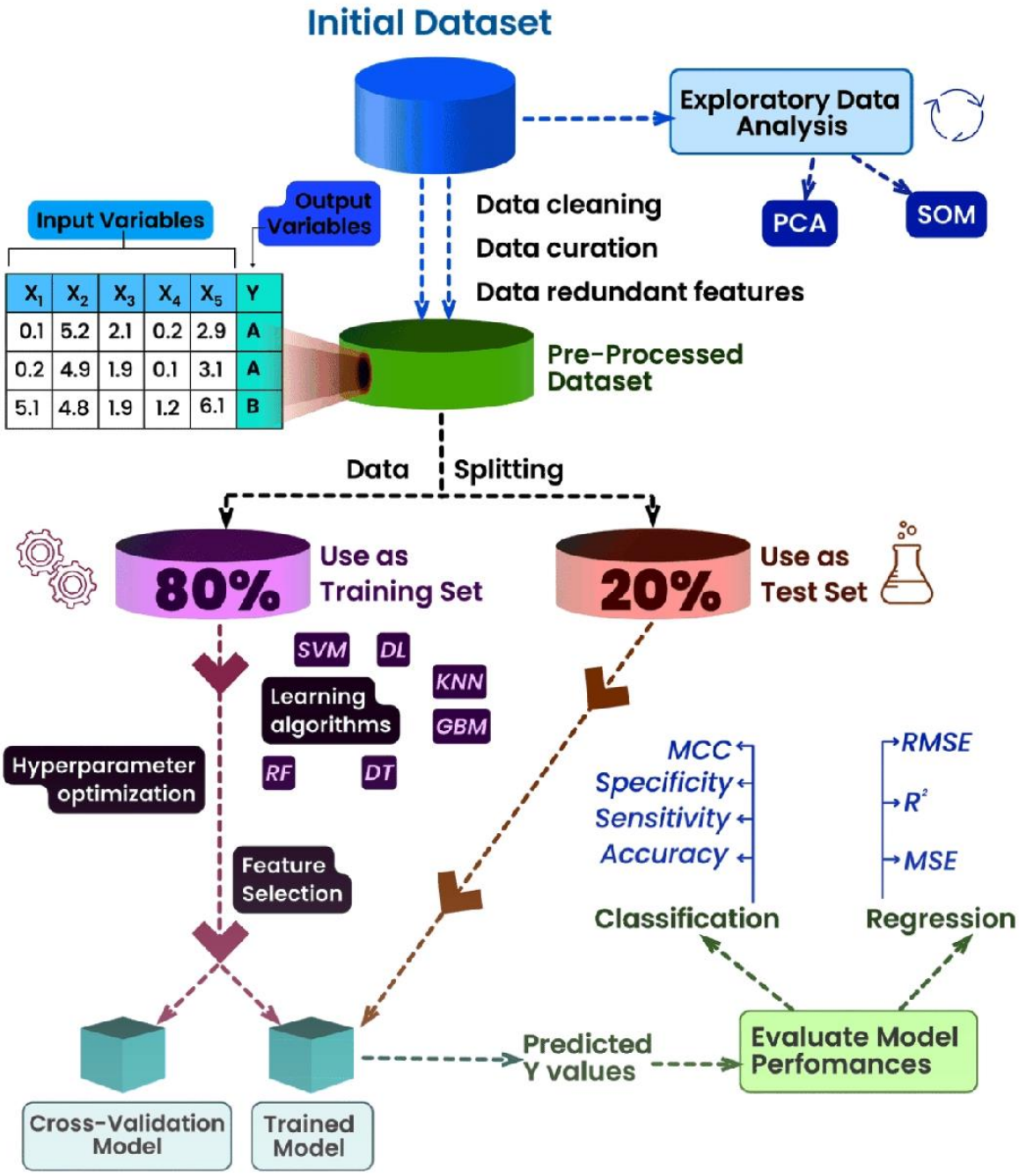
- **Visualization:** Data visualizations will be crucial for exploratory data analysis and presenting insights. Libraries like Matplotlib or Seaborn can be used for creating informative charts and graphs.

7. Additional Considerations:

- **Feature Engineering:** Techniques to transform and create new features from existing data might be necessary to enhance model performance.

This requirement analysis provides a roadmap for the project, ensuring all necessary elements are considered for a successful movie rating prediction model and analysis.

<u>Architecture Diagram</u>:



X_1	X_2	X_3	X_4	X_5	Y
0.1	5.2	2.1	0.2	2.9	A
0.2	4.9	1.9	0.1	3.1	A
5.1	4.8	1.9	1.2	6.1	B

The machine learning model in this project learns from movie data (genre, director, etc.) and their ratings.

1. **Data Preparation:** Clean and format the movie data.
2. **Train the Model:** The model learns the connection between movie features and ratings using training data.
3. **Evaluation:** Test the model's accuracy on unseen data.
4. **Prediction:** Use the trained model to predict ratings for new movies.

List of Modules used:

I. Data Retrieval

- Importing Required Libraries
- Loading Data

II. Data Preparation

- Data Cleaning
  - Handling Missing Values
- Data Transformation
  - Reducing Number of variables

III. Data Exploration

- Descriptive Statistical Analysis
- Visualizations

IV. Data Modeling

- Model Selection and Execution
- Model Diagnostic and Comparison

V. Automation and Presentation

- Automation & Presentation

Module Description:

I. Data Acquisition

- **Importing Required Libraries:**
    - pandas: For data manipulation (loading, cleaning, transforming)
    - NumPy: For numerical computations and array operations
    - matplotlib and seaborn: For creating informative data visualizations

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
from joblib import dump, load
```

- **Loading Data:**
    - It uses pandas.read_csv() to read the "IMDb India Movies" dataset.
    - Then it explores the initial data to understand its structure (number of rows, columns, data types) and identify potential issues.

```
IMDb = pd.read_csv('/content/drive/MyDrive/IMDb Movies India.csv',
encoding='latin1')
IMDb.head()
```

	Name	Year	Duration	Genre	Rating	Votes	Director	Actor 1	Actor 2	Actor 3
0		NaN	NaN	Drama	NaN	NaN	J.S. Randhawa	Manmauji	Birbal	Rajendra Bhatia
1	#Gadhvi (He thought he was Gandhi)	(2019)	109 min	Drama	7.0	8	Gaurav Bakshi	Rasika Dugal	Vivek Ghamande	Arvind Jangid
2	#Homecoming	(2021)	90 min	Drama, Musical	NaN	NaN	Soumyajit Majumdar	Sayani Gupta	Plabita Borthakur	Roy Angana
3	#Yaaram	(2019)	110 min	Comedy, Romance	4.4	35	Ovais Khan	Prateik	Ishita Raj	Siddhant Kapoor
4	...And Once Again	(2010)	105 min	Drama	NaN	NaN	Amol Palekar	Rajat Kapoor	Rituparna Sengupta	Antara Mali

```
IMDb.columns
```

```
Index(['Name', 'Year', 'Duration', 'Genre', 'Rating', 'Votes', 'Director',
       'Actor 1', 'Actor 2', 'Actor 3'],
      dtype='object')
```

II. Data Understanding & Preprocessing

- **Understanding Data:**

1. **Name:** The name or title of the movie.
2. **Year:** The year when the movie was released.
3. **Duration:** The duration of the movie in minutes.
4. **Genre:** The genre or category to which the movie belongs (Action, Drama, Comedy).
5. **Rating:** The rating assigned to the movie, indicating its overall quality or reception.
6. **Votes:** The number of votes or ratings that the movie has received.
7. **Director:** The name of the director who directed the movie.
8. **Actor 1:** The name of the first actor associated with the movie.
9. **Actor 2:** The name of the second actor associated with the movie.
10. **Actor 3:** The name of the third actor associated with the movie.

- **Data Cleaning:**
    - Addressing missing values using imputation techniques (filling in missing entries with appropriate strategies) or deletion if necessary.
    - Let us check the total number of missing values in each column.

```
# Checking for missing values in the entire dataset
print(IMDb.isnull().sum())
```

```
Name          0
Year        528
Duration   8269
Genre      1877
Rating     7590
Votes      7589
Director    525
Actor 1    1617
Actor 2    2384
Actor 3    3144
dtype: int64
```

- Let us handle the missing values with the measures of central tendency [Mean, Median, Mode]
- Mean and Median for Numerical attributes and Mode for Categorical attributes
- Let us remove parenthesis surrounded by *Year* attribute and 'min' string present in *Duration* attribute and then convert them into a interger datatype.
- Let us rename the 'Duration' column name to 'Duration_in_min' just to indicate that they are in Minutes (Time Duration of movie)

```
for feature in ['Year', 'Duration']:
    IMDb[feature] = IMDb[feature].replace(r'\D', '', regex=True)

    # Convert the 'Year' and 'Duration' column to numeric type (Casted
as a floating-point variable)
    IMDb[feature] = pd.to_numeric(IMDb[feature], errors='coerce')

    # Converting columns to integer type
    IMDb[feature] = IMDb[feature].astype('Int64')

    # Filling missing values in 'Year' and 'Duration' with the median
value
    IMDb[feature].fillna(IMDb[feature].median(), inplace=True)

IMDb.rename(columns={'Duration': 'Duration_in_min'}, inplace=True)
```

- Let us fill NA in 'Rating' and 'Votes' attribute with Mean Value

```
# Convert the 'Votes' column to numeric type
IMDb['Votes'] = pd.to_numeric(IMDb['Votes'], errors='coerce')

#Filling missing values present in 'Rating' and 'Votes' attribute with
Mean Value
for feature in ['Rating', 'Votes']:
  IMDb[feature].fillna(IMDb[feature].mean(), inplace=True)
```

- Filling most frequent Actor name for Actor 2 in missing fields. Dropping rows of 'Genre', 'Director' and 'Actor 1' where ever the null values are present.

```
IMDb['Actor 2'].fillna(IMDb['Actor 2'].mode()[0], inplace=True)
for feature in ['Genre', 'Director', 'Actor 1']:
  IMDb.dropna(subset=[feature], inplace=True)
```

```
print(IMDb.isnull().sum())
```

```
Name                0
Year                0
Duration_in_min     0
Genre               0
Rating              0
Votes               0
Director            0
Actor 1             0
Actor 2             0
Actor 3          1039
dtype: int64
```

Now, the data cleaning is over. Let's move on to the next step of data processing.

- **Data Transformation:**

  Reducing Number of variables

  - Let us remove the variables that are not needed for our analysis and making prediction. The following columns are not required:
    1. Actor 3 (It does not affect our analysis significantly)
    2. Name

```
# Droping the 'Actor 3' and 'Name' columns
IMDb.drop(['Actor 3', 'Name'], axis=1, inplace=True)
```

Removing Duplicate Entries

```
IMDb.drop_duplicates()
```

	Year	Duration_in_min	Genre	Rating	Votes	Director	Actor 1	Actor 2
0	1991	131	Drama	5.841621	120.839493	J.S. Randhawa	Manmauji	Birbal
1	2019	109	Drama	7.000000	8.000000	Gaurav Bakshi	Rasika Dugal	Vivek Ghamande
2	2021	90	Drama, Musical	5.841621	120.839493	Soumyajit Majumdar	Sayani Gupta	Plabita Borthakur
3	2019	110	Comedy, Romance	4.400000	35.000000	Ovais Khan	Prateik	Ishita Raj
4	2010	105	Drama	5.841621	120.839493	Amol Palekar	Rajat Kapoor	Rituparna Sengupta
...	...	...	...	...	...	...	...	...
15503	1989	125	Action, Crime, Drama	5.800000	44.000000	S.P. Muthuraman	Chiranjeevi	Jayamalini
15504	1988	131	Action	4.600000	11.000000	Mahendra Shah	Naseeruddin Shah	Sumeet Saigal
15505	1999	129	Action, Drama	4.500000	655.000000	Kuku Kohli	Akshay Kumar	Twinkle Khanna
15506	2005	131	Action	5.841621	120.839493	Kiran Thej	Sangeeta Tiwari	Rekha
15508	1998	130	Action, Drama	6.200000	20.000000	K.C. Bokadia	Dharmendra	Jaya Prada

12397 rows × 8 columns

III. Data Exploration (EDA)

- **Descriptive Statistical Analysis:**
    - Calculate and visualize summary statistics (mean, median, standard deviation, quartiles) for numerical features like rating, duration, and year of release. Identify potential skewness or outliers.

```
print(IMDb.describe())
```

	Year	Duration_in_min	Rating	Votes
count	12407.0	12407.0	12407.000000	12407.000000
mean	1992.005884	130.052873	5.830017	122.394984
std	21.458992	20.571998	1.082952	140.255562
min	1917.0	21.0	1.100000	5.000000
25%	1978.0	129.0	5.600000	35.000000
50%	1994.0	131.0	5.841621	120.839493
75%	2010.0	135.0	6.300000	120.839493
max	2022.0	321.0	10.000000	999.000000

    - Getting the count of movies directed by each Director

```
IMDb["Director"].value_counts()
```

```
Director
Kanti Shah              51
Mahesh Bhatt            47
David Dhawan            43
Hrishikesh Mukherjee    42
B.R. Ishara             40
                        ..
Jitendra Kumar Singh     1
Satyabhan Sinha          1
Partho Sen-Gupta         1
Sudarshan Babbar         1
Kiran Thej               1
Name: count, Length: 5230, dtype: int64
```

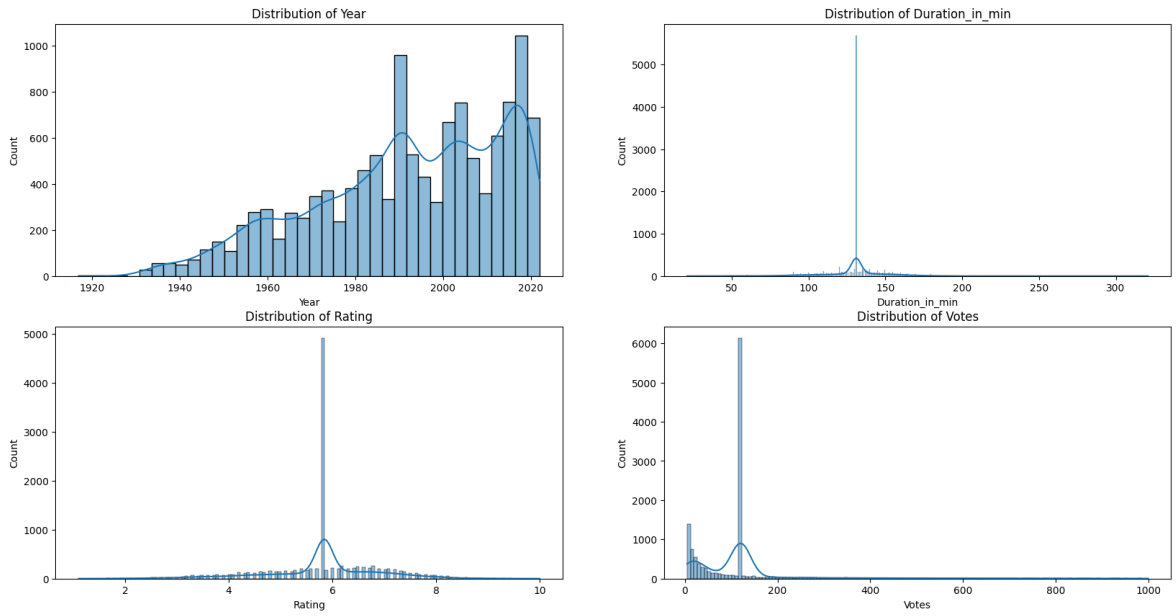
    - Creating visualizations (bar charts, pie charts) for categorical features to understand the distribution of genres, number of actors per movie, etc.

- **Visualizations:**
    - Create histograms to explore the distribution of movie ratings, year of release, and other numerical features. Identify potential skewness or outliers.
    - Use scatter plots to visualize relationships between features, such as rating vs. year, rating vs. genre, or rating vs. number of actors. Look for patterns or correlations.
    - Consider boxplots to compare the distribution of ratings across different genres, directors, or release years.

Let us plot histogram to find the distribution of numerical attributes in our IMDb dataset.
Numerical Columns:

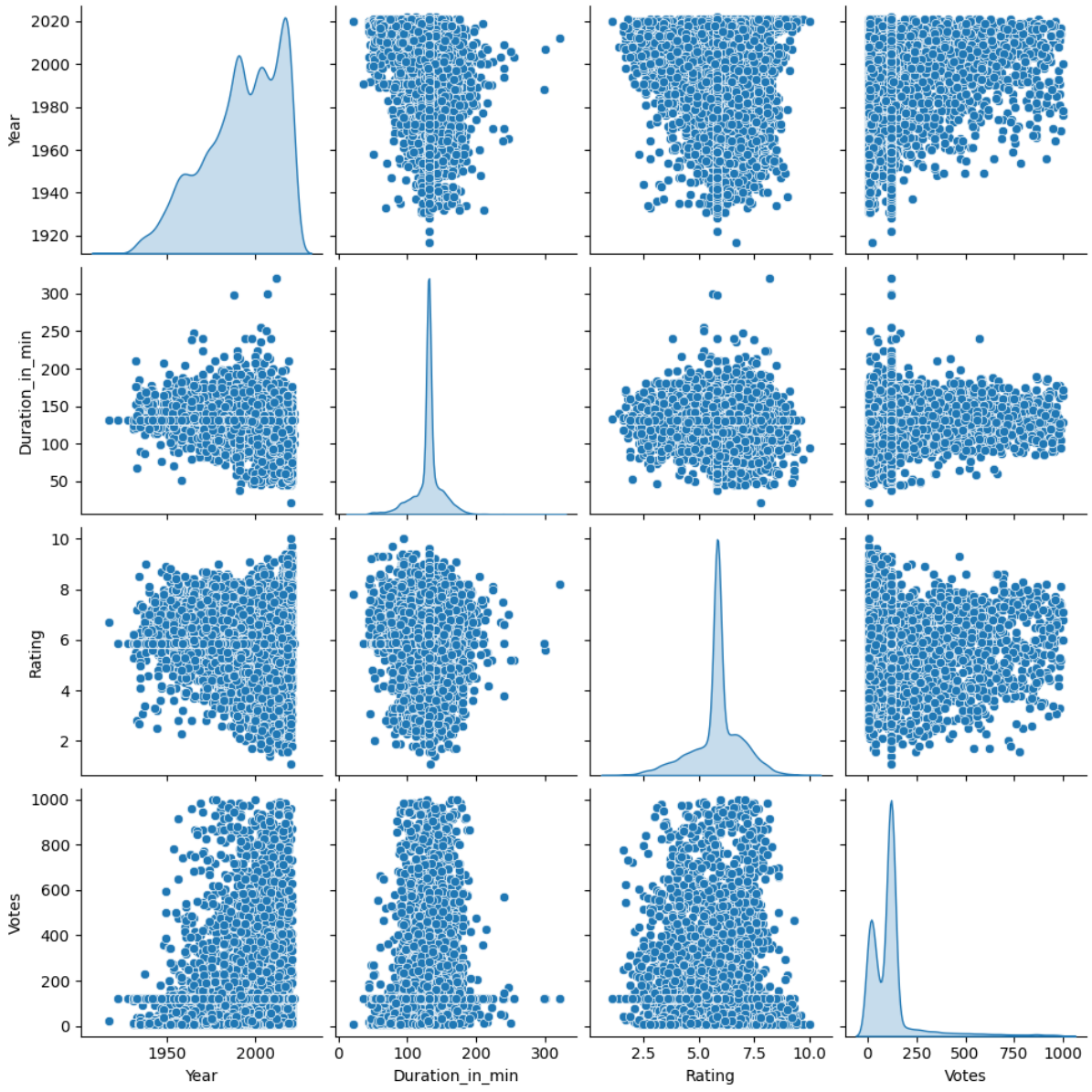
1. 'Year',
2. 'Duration_in_min',
3. 'Rating',
4. 'Votes'

```
plt.figure(figsize=(20, 10))
i = 1
for feature in ['Year', 'Duration_in_min', 'Rating', 'Votes']:
    plt.subplot(2, 2, i)
    sns.histplot(IMDb[feature], kde=True)
    plt.title(f'Distribution of {feature}')
    i += 1
plt.show()
```



- Pair plot to visualize the pair wise relationship between variables.

```
sns.pairplot(IMDb, diag_kind='kde')
plt.show()
```

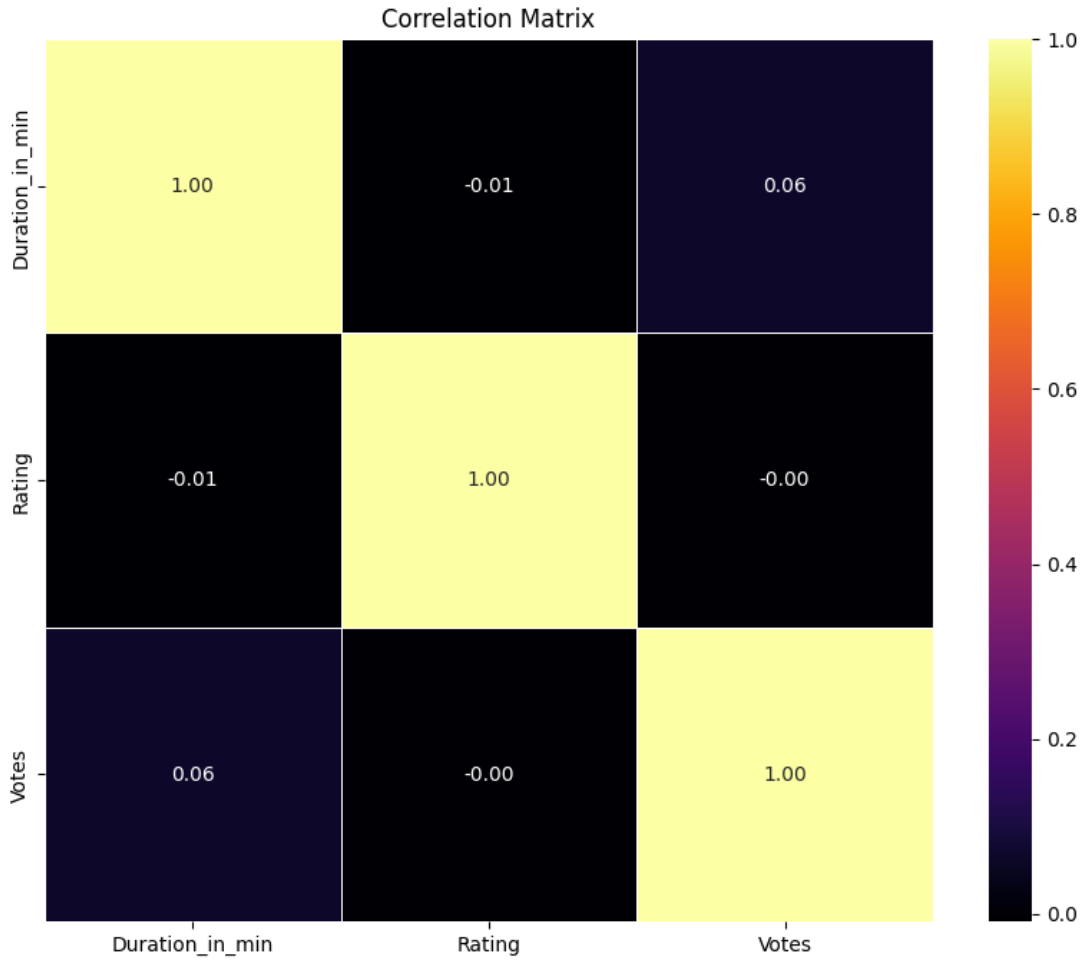


- To explore the correlation between features, a correlation matrix can be used which will give us an idea of how different features are related.

```
col = IMDb.columns.tolist()
li = [] #Numerical Columns
li.append(col[1])
li.append(col[3])
li.append(col[4])
correlation = IMDb[li].corr()
print(correlation)
```

	Duration_in_min	Rating	Votes
Duration_in_min	1.000000	-0.008747	0.063609
Rating	-0.008747	1.000000	-0.004276
Votes	0.063609	-0.004276	1.000000

```
# Creating a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation, annot=True, cmap='inferno', fmt=".2f",
linewidths=.5)
plt.title('Correlation Matrix')
plt.show()
```



Let us use linear regression to predict the movie rating based on historical data. But, we have some attributes with String datatype. We need to handle them separately by employing Feature Engineering.

Feature Engineering:

For Genre Mean Rating:

1. The following code snippet calculates the mean rating for each genre in the 'Genre' column.

2. The groupby('Genre')['Rating'].transform('mean') part groups the data by 'Genre' and computes the mean rating for each group.
3. The above step results with a new Series with the mean rating for each movie's respective genre.
4. This new feature is named 'Genre_mean_rating' and is added to the IMDb DataFrame.

These 4 steps are applicable for other attributes like Director, Actor 1, Actor 2. These new features aim to capture the average rating associated with certain genres, directors, and actors.

```
#Feature Engineering on IMDb dataset
def featureEngineering(IMDb):
  genre_mean_rating = IMDb.groupby('Genre')['Rating'].transform('mean')
  IMDb['Genre_mean_rating'] = genre_mean_rating

  director_mean_rating =
IMDb.groupby('Director')['Rating'].transform('mean')
  IMDb['Director_encoded'] = director_mean_rating

  actor1_mean_rating = IMDb.groupby('Actor
1')['Rating'].transform('mean')
  IMDb['Actor1_encoded'] = actor1_mean_rating

  actor2_mean_rating = IMDb.groupby('Actor
2')['Rating'].transform('mean')
  IMDb['Actor2_encoded'] = actor2_mean_rating

featureEngineering(IMDb)
```

IV. Data Modeling & Evaluation

The next few steps in this phase are crucial for building a predictive model and it involves preparing the data for model training, selecting a model, training the model, and evaluating its performance.

A. Model Selection & Training

Let X be the new dataframe which comprises of independent variables and y be the dependent or target variable of our IMDb.

Step 1. Train-Test Split:

- Divide the preprocessed data into two sets:
    - Training set (typically 70-80%): Used to train the model and learn the relationships between features and movie ratings.

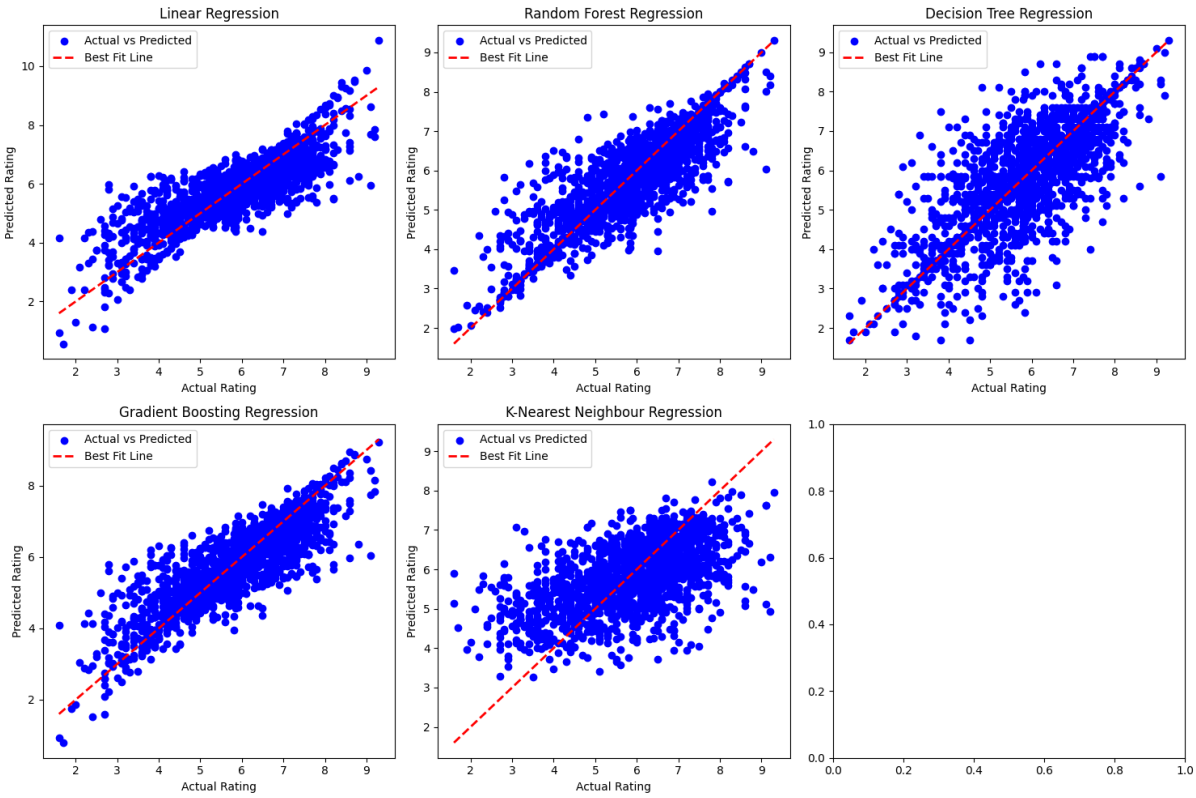
- Testing set (typically 20-30%): Used to evaluate the model's performance on unseen data, simulating real-world use cases.

```
# Defining features (X) and target variable (y)
ind_var = [ 'Year', 'Duration_in_min', 'Votes',
'Genre_mean_rating','Director_encoded','Actor1_encoded',
'Actor2_encoded']
X = IMDb[ind_var]
y = IMDb['Rating']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Step 2. Model Selection

- Choose a suitable machine learning regression algorithm based on the characteristics of the data and the project goals. I have trained my data with the following Machine Learning models and selected one among the best performing model.



1. **Multiple Linear Regression (MLR):**

    - Assuming a linear relationship between movie rating and features.

- Limitations: Might not capture complex non-linear relationships between features.

2. **Random Forest Regression:**
    - A powerful ensemble method that combines multiple decision trees, improving accuracy and generalization.
    - Handles non-linear relationships and feature interactions effectively.
    - Less interpretable than MLR, but feature importance can be analyzed to understand influential features.
3. **Decision Tree Regression:**
    - A tree-based model that splits the data into decision nodes based on feature values, predicting the movie rating.
    - Easy to visualize and understand, providing insights into the decision-making process.
    - Can be prone to overfitting if not carefully tuned with hyperparameters
4. **Gradient Boosting Regression:**
    - Another ensemble method that sequentially builds decision trees to improve prediction accuracy over individual trees.
    - Often outperforms other models on complex datasets due to its iterative learning approach.
    - Interpretability can be challenging, but feature importance analysis can provide some insights.
5. **K-Nearest Neighbors Regression (KNN):**
    - Predicts movie ratings based on the average rating of the k most similar movies in the training data.
    - Similarity is determined using a distance metric (Euclidean distance).

- Can handle complex data distributions where other models might struggle.
- Becomes computationally expensive for large datasets as it needs to compare new data points to all training points.

Among the above models trained with our IMDb movie data, I considered

1. Random Forest Regression &
2. Gradient Boosting Regression model as best performing models.

Random Forest Regression:

```
# Initializing and Training the RandomForest model
randomForest = RandomForestRegressor(n_estimators=100, random_state=42)
randomForest.fit(X_train, y_train)

# Making predictions on the test set
y_pred_random = randomForest.predict(X_test)

# Train set metrics
train_mse = mean_squared_error(y_train, randomForest.predict(X_train))
train_r2 = r2_score(y_train, randomForest.predict(X_train))

print('Model: Random Forest Regressor\n')

print(f'Training Mean Squared Error: {train_mse}')
print(f'Training R-squared: {train_r2}\n')

# Test set metrics
test_mse = mean_squared_error(y_test, y_pred_random)
test_r2 = r2_score(y_test, y_pred_random)

print(f'Test Mean Squared Error: {test_mse}')
print(f'Test R-squared: {test_r2}')
```

```
Model: Random Forest Regressor

Training Mean Squared Error: 0.04486238614318421
Training R-squared: 0.961840929757815

Test Mean Squared Error: 0.3285231406885322
Test R-squared: 0.7169718758050624
```

```
from sklearn.model_selection import learning_curve
```

```
train_sizes, train_scores, test_scores = learning_curve(randomForest,
X, y, cv=5, scoring='r2')

plt.plot(train_sizes, np.mean(train_scores, axis=1), label='Training
Data')
plt.plot(train_sizes, np.mean(test_scores, axis=1), label='Testing
Data')
plt.xlabel('Training Set Size')
plt.ylabel('R-squared Score')
plt.legend()
plt.show()
```

V. Automation and Presentation

- Among the two best performing models, I selected ***Random Forest Regression*** for automation and future prediction with unseen real world data.

```
def make_predictions(new_data):
    predictions = randomForest.predict(new_data)
    predictions = pd.DataFrame({
      'Random Forest': predictions
  })
    return predictions

# Sample data for testing

new_data = pd.DataFrame({
    'Year' : [2021, 2020, 2012, 2021, 2023] ,
    'Duration_in_min' : [130, 150, 120, 90, 180],
    'Genre' : ['Action, Drama, Comedy', "Drama", "Action", "Drama,
Musical", "Action, Drama"],
    'Rating' : [4.25, 5.86, 7.28, 6.63, 4.9],
    'Votes' : [120, 48, 29, 100, 67],
    'Director' : ["Amol Palekar","Ovais","Aarthi Agarwal","A.R.
Murugadoss","A. Shamsheer"],
    'Actor 1' : ["Ramesh", "Suresh", "Amitabh Bachchan", "Thuppaki",
"Deva"],
    'Actor 2' : ["Kajol", "Ishita Raj", "Trisha", "Kajal Aggarwal",
"Birbal"],
})

featureEngineering(new_data)

predictions = make_predictions(new_data[ind_var])
```

```
print("\nRating Result:")
print(predictions)
```

```
Rating Result:
   Random Forest
0       3.783000
1       5.990330
2       7.402416
3       6.665324
4       4.724665
```

```
# Save the model
dump(randomForest, 'MovieRatingPredictive_model.joblib')

# Load the model for predictions
loaded_model = load('MovieRatingPredictive_model.joblib')
new_predictions = loaded_model.predict(new_data[ind_var])
new_predictions
```

```
array([3.783     , 5.99032971, 7.40241621, 6.66532428, 4.72466486])
```

Conclusion:

In this project, I investigated the use of machine learning models to predict movie ratings on the IMDb India Movies dataset. I explored several models, including Random Forest Regression, and evaluated their performance on a training and testing set.

The Random Forest Regressor achieved a training mean squared error (MSE) of 0.0448 and a training R-squared of 0.9618, indicating a strong fit to the training data. However, the test MSE of 0.3285 and test R-squared of 0.7169 suggest that the model's generalizability to unseen data could be improved.

Key Findings:

- *Random Forest Regression* was the best performing model among those explored in this project.
- The model achieved a high R-squared on the training data, indicating a good fit.