IMDb Movie Rating Prediction – Complete & Easy-to-Understand Jupyter Notebook

This notebook is a **fully rewritten, step-by-step, beginner-friendly version** of your original Python script.

✅ **Nothing is removed** from your original code logic
✅ Every step is explained in **simple terms**
✅ Written exactly as you would use in a **Jupyter Notebook**

---

1. Import All Required Libraries

We first import all Python libraries needed for: - Data handling (pandas, numpy) - Visualization (matplotlib, seaborn) - Machine learning (scikit-learn, xgboost) - Saving the trained model (joblib)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import re
import joblib
from datetime import datetime

warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split, cross_val_score, KFold,
learning_curve
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor,
VotingRegressor
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from xgboost import XGBRegressor
```

---

2. Load the Dataset and Inspect It

We load the IMDb India movies dataset and check: - Number of rows and columns - Column names - Data types - Missing values

```
df = pd.read_csv("IMDb_Movies_India.csv", encoding="latin-1")

print(df.shape)
df.head()
```

```
df.info()
df.isnull().sum()
```

---

3. Data Cleaning

3.1 Clean the Year Column

Some years are written like "(2019)" or "2018–2020".
We extract only the valid **4-digit year**.

```
def extract_year(value):
    if pd.isna(value):
        return np.nan
    years = re.findall(r'\b\d{4}\b', str(value))
    if years:
        year = int(years[-1])
        if 1900 <= year <= datetime.now().year:
            return year
    return np.nan


df['Year'] = df['Year'].apply(extract_year)
```

3.2 Clean Duration Column

Duration values look like "142 min".
We extract only the number.

```python
def clean_duration(value):
    if pd.isna(value):
        return np.nan
    return float(re.findall(r'\d+', str(value))[0])


df['Duration'] = df['Duration'].apply(clean_duration)
```

3.3 Clean Votes Column

Votes may contain commas like "1,23,456".
We remove commas and convert to number.

```python
df['Votes'] = df['Votes'].astype(str).str.replace(',', '')
df['Votes'] = pd.to_numeric(df['Votes'], errors='coerce')
```

3.4 Clean Rating Column

Convert rating to numeric and remove invalid values.

```python
df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')
```

3.5 Handle Text Columns

For categorical columns, missing values are replaced with "Unknown".

```python
text_columns = ['Genre', 'Director', 'Actor 1', 'Actor 2', 'Actor 3']
for col in text_columns:
    df[col] = df[col].fillna('Unknown')
```

---

4. Handle Missing Values

4.1 Remove Rows Without Rating (Target Variable)

We cannot train without ratings, so we drop them.

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

4.2 Fill Remaining Missing Values

We fill numerical missing values using **median**.

```
for col in ['Duration', 'Year', 'Votes']:
    df[col].fillna(df[col].median(), inplace=True)
```

---

5. Feature Engineering

We create new meaningful features to improve prediction accuracy.

5.1 Number of Actors

```
df['Num_Actors'] = df[['Actor 1','Actor 2','Actor 3']].apply(
    lambda x: sum(x != 'Unknown'), axis=1
)
```

5.2 Genre Count

```
df['Genre_Count'] = df['Genre'].apply(lambda x: len(x.split(',')))
```

5.3 Movie Age

```
df['Movie_Age'] = datetime.now().year - df['Year']
```

5.4 Director Experience

```
df['Director_Exp'] = df['Director'].map(df['Director'].value_counts())
```

5.5 Actor Popularity

```
actors = pd.concat([df['Actor 1'], df['Actor 2'], df['Actor 3']])
actor_counts = actors.value_counts()
```

```python
df['Actor1_Pop'] = df['Actor 1'].map(actor_counts)
df['Actor2_Pop'] = df['Actor 2'].map(actor_counts)
df['Actor3_Pop'] = df['Actor 3'].map(actor_counts)
```

---

6. Outlier Handling (IQR Method)

Extreme values are capped using the Interquartile Range method.

```python
def cap_outliers(series):
    Q1, Q3 = series.quantile([0.25, 0.75])
    IQR = Q3 - Q1
    return series.clip(Q1 - 1.5*IQR, Q3 + 1.5*IQR)

for col in df.select_dtypes(include=np.number).columns:
    df[col] = cap_outliers(df[col])
```

---

7. Exploratory Data Analysis (EDA)

```python
plt.figure(figsize=(12,8))
sns.heatmap(df.corr(), cmap='coolwarm')
plt.show()
```

---

8. Train-Test Split

```python
X = df.drop(['Rating','Name'], axis=1, errors='ignore')
y = df['Rating']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42,
    stratify=pd.qcut(y, 5, duplicates='drop')
)
```

---

9. Target Mean Encoding for Categorical Features

```
cat_cols = X_train.select_dtypes(include='object').columns
encoding_maps = {}

global_mean = y_train.mean()

for col in cat_cols:
    mapping = y_train.groupby(X_train[col]).mean().to_dict()
    encoding_maps[col] = mapping
    X_train[col+'_encoded'] = X_train[col].map(mapping).fillna(global_mean)
    X_test[col+'_encoded'] = X_test[col].map(mapping).fillna(global_mean)
    X_train.drop(col, axis=1, inplace=True)
    X_test.drop(col, axis=1, inplace=True)
```

---

10. Imputation and Feature Scaling

```
imputer = SimpleImputer(strategy='median')
scaler = StandardScaler()

X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

---

11. Train Multiple Models

```
models = {
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor(n_estimators=100, random_state=42),
    'Gradient Boosting': GradientBoostingRegressor(),
    'XGBoost': XGBRegressor(verbosity=0),
    'SVR': SVR(),
    'Neural Network': MLPRegressor(max_iter=500)
}

results = []

for name, model in models.items():
```

```
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    results.append([
        name,
        r2_score(y_test, preds),
        np.sqrt(mean_squared_error(y_test, preds)),
        mean_absolute_error(y_test, preds)
    ])

results_df = pd.DataFrame(results, columns=['Model','R2','RMSE','MAE'])
results_df.sort_values('R2', ascending=False)
```

---

12. Ensemble Model (Voting Regressor)

```
top_models = results_df.sort_values('R2', ascending=False).head(3)['Model']
ensemble = VotingRegressor([(m, models[m]) for m in top_models])
ensemble.fit(X_train, y_train)
```

---

13. Save Model

```
joblib.dump({
    'model': ensemble,
    'scaler': scaler,
    'imputer': imputer,
    'encoding_maps': encoding_maps
}, 'movie_rating_model.pkl')
```

---

14. Prediction Function

```
def predict_movie_rating(movie):
    assets = joblib.load('movie_rating_model.pkl')
    model = assets['model']
    scaler = assets['scaler']
    imputer = assets['imputer']
    enc_maps = assets['encoding_maps']

    df_new = pd.DataFrame([movie])
    for col, mp in enc_maps.items():
```

```
        df_new[col+'_encoded'] =
df_new[col].map(mp).fillna(np.mean(list(mp.values())))
        df_new.drop(col, axis=1, inplace=True)

    df_new = scaler.transform(imputer.transform(df_new))
    return round(model.predict(df_new)[0], 2)
```

---

15. Example Prediction

```
predict_movie_rating({
    'Duration':150,
    'Year':2019,
    'Votes':250000,
    'Genre':'Action, Drama',
    'Director':'Rajkumar Hirani',
    'Actor 1':'Aamir Khan',
    'Actor 2':'Kareena Kapoor',
    'Actor 3':'Anil Kapoor',
    'Num_Actors':3,
    'Genre_Count':2,
    'Movie_Age':5,
    'Director_Exp':10,
    'Actor1_Pop':50,
    'Actor2_Pop':45,
    'Actor3_Pop':40
})
```

✅ Final Summary

- Full ML pipeline implemented
- Multiple models + ensemble
- Model saved to disk
- Prediction function ready

🔝 **This notebook is suitable for exams, projects, viva, and production use.**