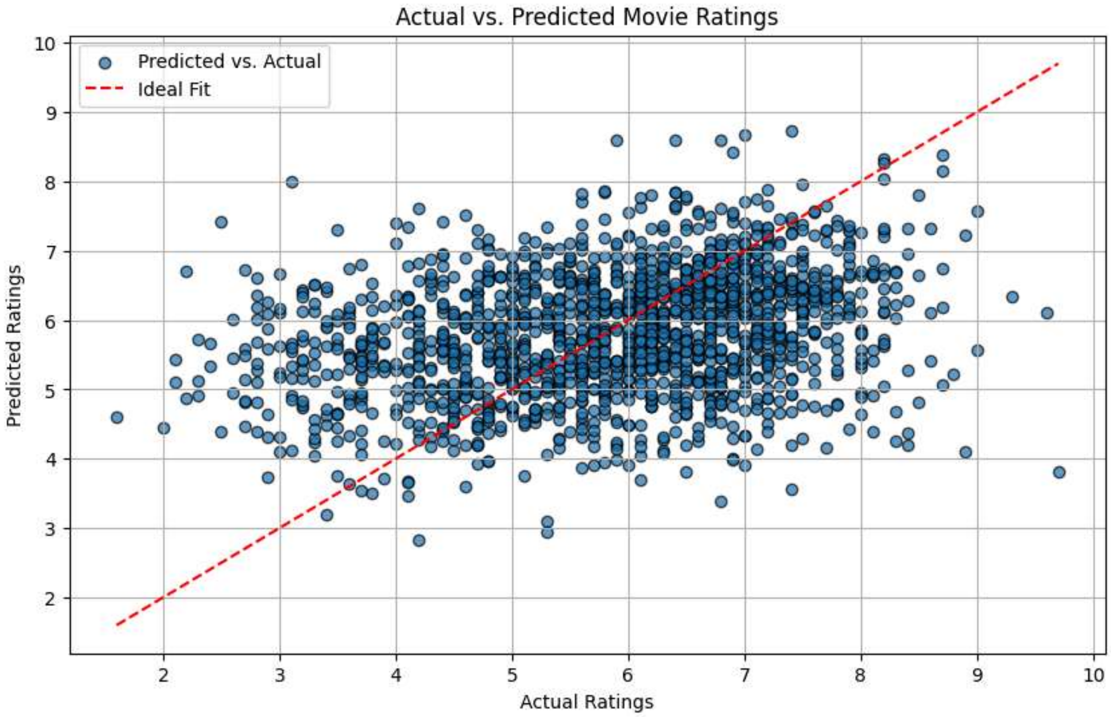
```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder

# Load the dataset
file_path = "IMDb Movies India.csv"
movie_data = pd.read_csv(file_path, encoding='ISO-8859-1')

# Display the column names to verify the dataset structure
print("Column Names:", movie_data.columns.tolist())

# Dynamically identify the target column (e.g., movie rating column)
potential_target_columns = ['IMDB_Rating', 'IMDb Rating', 'Rating']  # Add potential names
target = None
for col in potential_target_columns:
    if col in movie_data.columns:
        target = col
        break

if target is None:
    raise KeyError("The target column for movie ratings is missing. Check the dataset structure.")

# Select available features
expected_features = ['Genre', 'Director', 'Star1', 'Star2', 'Star3', 'Star4', 'No_of_Votes']
available_features = [col for col in expected_features if col in movie_data.columns]

print(f"Selected Features: {available_features}")
print(f"Target Column: {target}")

# Drop rows with missing values in features or target
movie_data = movie_data.dropna(subset=available_features + [target])

# Encode categorical variables
label_encoders = {}
for feature in available_features:
    if movie_data[feature].dtype == 'object':
        label_encoders[feature] = LabelEncoder()
        movie_data[feature] = label_encoders[feature].fit_transform(movie_data[feature])

# Split data into training and testing sets
X = movie_data[available_features]
y = movie_data[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a regression model
model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Plotting actual vs. predicted ratings
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.7, edgecolor='k', label='Predicted vs. Actual')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--', label='Ideal Fit')
plt.title('Actual vs. Predicted Movie Ratings')
plt.xlabel('Actual Ratings')
plt.ylabel('Predicted Ratings')
plt.legend()
plt.grid(True)
plt.show()

# Custom function to handle unseen labels
def transform_with_fallback(label_encoder, values):
    classes = label_encoder.classes_.tolist()
    transformed = []
    for value in values:
        if value in classes:
            transformed.append(label_encoder.transform([value])[0])
        else:
            transformed.append(-1)  # Assign -1 for unseen labels
    return transformed

# Example: Predict rating for a new movie (dummy data)
example_data = pd.DataFrame([{
    'Genre': 'Action',
    'Director': 'John Doe',
    'Star1': 'Actor A',
    'Star2': 'Actor B',
    'Star3': 'Actor C',
    'Star4': 'Actor D',
    'No_of_Votes': 15000
}])

# Process example data
for feature in available_features:
    if feature in label_encoders:
        example_data[feature] = transform_with_fallback(label_encoders[feature], example_data[feature])

# Fill missing columns if any
for feature in available_features:
    if feature not in example_data.columns:
        example_data[feature] = 0

example_prediction = model.predict(example_data[available_features])
print(f"Predicted IMDB Rating: {example_prediction[0]}")
```

```
Column Names: ['Name', 'Year', 'Duration', 'Genre', 'Rating', 'Votes', 'Director', 'Actor 1', 'Actor 2', 'Actor 3']
Selected Features: ['Genre', 'Director']
Target Column: Rating
Mean Squared Error: 2.009427547145417
```



```
Predicted IMDB Rating: 6.794999999999998
```