ML Movie Final Team Project

Project Overview

- **Team Members**: DuyAnh, Saloni, Kirsten
- **Team Number**: [Team #5]
- **Project Title**: Movie Rating Prediction Using Machine Learning

---

1. Problem Definition

Problem Statement

What is the problem you are going to solve? The goal of this project is to predict audience ratings of movies based on features including content rating, distribution type, movie descriptions, genre, language, cast information, and other metadata to train machine learning models capable of estimating the numeric rating a movie might receive.

Why does the problem need to be solved? Movie rating prediction is valuable for streaming platforms, movie studios, and recommendation systems to understand audience preferences, optimize content acquisition strategies, and improve user experience. Accurate rating predictions can help stakeholders make informed data-driven decisions about movie investments and content curation.

What aspect of the problem will a machine/deep learning model solve? With the abundance of movie metadata available, machine learning models can identify complex patterns and relationships between movie characteristics and audience ratings that would be difficult to detect manually. The models can process multiple features simultaneously to provide accurate rating predictions for new movies.

STAR Framework Planning

- **Situation**: Movie industry stakeholders need to predict audience reception of movies based on available metadata. Traditional approaches are manual, time-consuming, and often biased.
- **Task**: Develop and compare multiple machine learning models to accurately predict movie ratings using Letterboxd dataset.
- **Action**: Implement comprehensive ML pipeline with preprocessing, feature engineering, and model comparison
- **Result**: Achieve 60.6% variance explanation in movie ratings with Decision Tree as the best-performing model

---

2. Dataset Information

Dataset Details

- **Dataset Name**: Letterboxd
- **Source**: https://www.kaggle.com/datasets/gsimonx37/letterboxd/data
- **Size**: 434,256 data points (after removing missing values)
- **Features**: 24 processed features (4 numerical, 9 categorical with high-cardinality handling)
- **Target Variable**: Movie ratings (continuous variable)

Data Quality Assessment

- **Missing Values**: 2,795 rows (0.64%) with missing target values removed
- **Outliers**: Handled through model regularization and tree-based splitting criteria
- **Data Types**: Mixed dataset with numerical features (dates, indices) and categorical features (ratings, types, descriptions)

---

3. EDA and Pre-Processing

Exploratory Data Analysis Plan

- **Data Overview**
  - Shape: 434,256 rows x 24 columns (after preprocessing)
  - Data types distribution: 4 numerical, 20 categorical (after encoding)
  - Missing value analysis: 0.64% missing in target variable
- **Statistical Summary**
  - Descriptive statistics for numerical features
  - Correlation analysis between features and target
  - Distribution analysis of categorical variables

- **Visualizations Planned**
  - Feature importance rankings across models
  - Model performance comparisons
  - Distribution of target variable (ratings)

Pre-Processing Steps

- **Missing Value Handling**
  - Method: Complete case analysis - removed rows with missing target values
  - Justification: Small percentage (0.64%) of missing data, removal preserves data integrity
- **Outlier Treatment**
  - Detection method: Handled implicitly through tree-based model splitting criteria
  - Treatment: Retained outliers as they may contain valuable information about extreme cases
- **Feature Engineering**
  - High-cardinality categorical encoding using LabelEncoder for features with >50 unique values
  - One-hot encoding for low-cardinality categorical features
  - Standard scaling for numerical features
- **Data Scaling/Normalization**
  - Method: StandardScaler for numerical features
  - Applied to: All numerical features in the preprocessing pipeline

---

4. Modeling Methods, Validation, and Performance Metrics

Models to Implement

- **Model 1**: Linear Regression
  - Rationale: Baseline model to establish linear relationships
  - Hyperparameters: Default parameters with n_jobs=-1 for parallel processing
- **Model 2**: Ridge Regression
  - Rationale: Handle potential multicollinearity with L2 regularization
  - Hyperparameters: alpha=1.0, solver='lsqr' for large dataset efficiency
- **Model 3**: Lasso Regression
  - Rationale: Feature selection through L1 regularization
  - Hyperparameters: alpha=0.1, max_iter=2000
- **Model 4**: Elastic Net
  - Rationale: Combine L1 and L2 regularization benefits
  - Hyperparameters: alpha=0.1, l1_ratio=0.5, max_iter=2000
- **Model 5**: Decision Tree
  - Rationale: Capture non-linear relationships and feature interactions
  - Hyperparameters: max_depth=10, min_samples_split=20, min_samples_leaf=10
- **Model 6**: Random Forest
  - Rationale: Ensemble method to reduce overfitting and improve generalization
  - Hyperparameters: n_estimators=30, max_depth=10, min_samples_split=20
- **Model 7**: Extra Trees
  - Rationale: Alternative ensemble method with random splits
  - Hyperparameters: n_estimators=30, max_depth=10, min_samples_split=20

Validation Strategy

- **Train/Validation/Test Split**: 80/20 split (347,404 training, 86,852 testing)
- **Cross-Validation**: 3-fold cross-validation for model evaluation
- **Validation Approach**: Hold-out validation with cross-validation for robust performance estimation

Performance Metrics

Primary Metrics (aligned with project objectives):

- R-squared (R^2): Proportion of variance explained in ratings
- Root Mean Squared Error (RMSE): Magnitude of prediction errors
- Mean Absolute Error (MAE): Average absolute prediction error

Secondary Metrics:

- Cross-validation R^2 with standard deviation
- Training vs. Test R^2 for overfitting assessment

5. Modeling Results and Findings

Model Comparison Framework

Model	Test R²	RMSE	MAE	Training Time	Complexity
Decision Tree	0.6060	0.3599	0.2627	Medium	Medium
Random Forest	0.5946	0.3651	0.2798	High	High
Ridge Regression	0.3927	0.4469	0.3483	Fast	Low
Linear Regression	0.3927	0.4469	0.3483	Fast	Low
Extra Trees	0.3597	0.4588	0.3626	High	High
Elastic Net	0.2342	0.5018	0.3996	Fast	Low
Lasso Regression	0.1680	0.5230	0.4199	Fast	Low

Key Findings

- **Best Performing Model**: Decision Tree ($R^2 = 0.6060$) - unexpectedly outperformed ensemble methods
- **Model Rankings**:
  1. Decision Tree (60.6% variance explained)
  2. Random Forest (59.5% variance explained)
  3. Ridge/Linear Regression (39.3% variance explained)
  4. Extra Trees (36.0% variance explained) - significantly underperformed
- **Surprising Results**:
  - Decision Tree outperformed Random Forest, contrary to typical expectations
  - Extra Trees dramatically underperformed (36.0%) due to aggressive regularization parameters
  - Ridge and Linear Regression performed identically, suggesting minimal multicollinearity
  - Lasso performed worst (16.8%) due to overly aggressive feature selection
- **Challenges Encountered**:
  - Speed optimization required aggressive regularization that severely hurt Extra Trees performance
  - High-cardinality text features showed low importance with simple label encoding
  - Spurious "Unnamed: 0" feature appeared in importance rankings, indicating preprocessing artifact
- **Feature Importance**:
  - Most important: rating_y_G (content rating for G-rated movies) - dominant predictor
  - Distribution types (Theatrical, TV, Digital releases) highly influential
  - Content rating categories consistently outweigh other feature types
  - Text features (descriptions, taglines) severely underutilized with current preprocessing approach

Business/Practical Implications

The results demonstrate that movie ratings can be predicted with moderate accuracy (60.6% variance explained) using metadata alone. Content rating (especially G-rated movies) and distribution type are the strongest predictors, suggesting that target audience and release strategy significantly influence ratings. The dramatic underperformance of Extra Trees highlights the importance of proper hyperparameter tuning, while the poor utilization of textual features indicates substantial opportunities for enhanced natural language processing to capture semantic content and improve prediction accuracy.

6. Technical Implementation

Tools and Libraries

- **Programming Language**: Python
- **Environment**: Google Colab
- **Key Libraries**:
    - pandas, numpy (data manipulation)
    - matplotlib, seaborn (visualization)
    - scikit-learn (ML models, preprocessing, metrics)
    - scipy (statistical functions)
    - warnings (error handling)

Github

https://github.com/MtnDoob/504_Final/tree/main

Code Organization

- **Data Loading and Cleaning**: MovieRatingsPredictionPipeline.preprocess_data()
- **EDA**: Integrated within preprocessing and feature importance analysis
- **Feature Engineering**: ColumnTransformer with StandardScaler and OneHotEncoder
- **Model Training**: MovieRatingsPredictionPipeline.train_models()
- **Model Evaluation**: Comprehensive metrics calculation and cross-validation
- **Results Visualization**: Feature importance analysis and model comparison tables

---

7. Report Structure (APA 7 Style)

Report Outline (7-10 pages)

1. **Introduction**
    - Problem statement: Movie rating prediction using metadata
    - Objectives: Compare ML models for rating prediction accuracy
    - Dataset overview: 434K Letterboxd movie records with 24 features
2. **Literature Review** (if applicable)
    - Related work in movie recommendation systems
    - Methodology justification for ensemble vs. linear methods
3. **Methodology**
    - Data preprocessing: Missing value handling, categorical encoding, scaling
    - Model selection: Seven regression models from linear to ensemble methods
    - Validation approach: 80/20 split with 3-fold cross-validation
4. **Results**
    - Model performance: Decision Tree achieved best R^2 of 0.6060
    - Comparison analysis: Unexpected Decision Tree superiority over Random Forest
    - Feature analysis: Content rating and distribution type dominance
    - Extra Trees underperformance: Aggressive regularization impact
5. **Discussion**
    - Interpretation of results: Moderate predictive success with clear feature hierarchy
    - Limitations: Underutilized text features, suboptimal hyperparameters for Extra Trees
    - Future work: Advanced NLP for text features, comprehensive hyperparameter optimization
6. **Conclusion**
    - Summary of findings: 60.6% variance explanation with interpretable feature importance
    - Practical implications: Content strategy and distribution insights for movie industry

Appendices

- **Appendix A**: Code (PDF format)
- **Appendix B**: Code (HTML format)
- **Appendix C**: Additional visualizations/tables

---

8. Video Presentation Plan

Presentation Structure (Equal participation)

- **Duration**: 12-15 minutes
- **Saloni**: Introduction and Problem Definition (3-4 minutes)
- **Kirsten**: Methodology and Data Processing (4-5 minutes)
- **Duy-Anh**: Results, Findings, and Conclusions (4-5 minutes)

Presentation Outline

1. **Introduction** (3-4 minutes)
    - Problem statement: Why predict movie ratings?
    - Dataset overview: 434K Letterboxd records
2. **Methodology** (4-5 minutes)
    - EDA highlights: Feature distribution and missing data handling
    - Model selection rationale: Linear to ensemble comparison strategy
3. **Results** (4-5 minutes)
    - Model comparison: Decision Tree superiority (60.6% R²)
    - Unexpected findings: Extra Trees underperformance analysis
    - Feature importance: Content rating dominance visualization
4. **Conclusion** (2-3 minutes)
    - Summary: 60.6% predictive accuracy with clear feature hierarchy
    - Implications: Content strategy and distribution insights
    - Future work: Enhanced NLP and hyperparameter optimization

---

9. AI Tool Usage Disclosure

AI Tools Used

- **Tool 1**: Claude
    - **Purpose**: Code debugging, results interpretation, and consistency checking across documents
    - **Sections**: Pipeline optimization, results analysis, performance table validation
    - **Attribution**: AI assistance noted in code comments and document revision process

```
# Data manipulation and visualization
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express
import os
import kagglehub
from scipy import stats
import warnings
warnings.filterwarnings('ignore')

# Scikit-learn imports
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder, MultiLabelBinarizer, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Regression models
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor
from sklearn.linear_model import LinearRegression, LogisticRegression, Ridge, ElasticNet, Lasso
from sklearn.svm import SVR, SVC
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor

# Classification models
from sklearn.ensemble import RandomForestClassifier

# Clustering
from sklearn.cluster import KMeans

# Feature selection and text processing
```

```
from sklearn.feature_selection import SelectKBest, f_regression, RFE
from sklearn.feature_extraction.text import TfidfVectorizer

# Metrics
from sklearn.metrics import (mean_squared_error, accuracy_score, classification_report,
                             r2_score, mean_absolute_error)
```

Start coding or generate with AI.

```
# run this cell unless needed to populate current working directory
# run cell if adding files manually or download from kaggle repository (path may change)
# Manual Download from Google drive: https://drive.google.com/drive/folders/1OSOw5UwBafo1xTwBLjU_gQZwXgDkZde-?usp=sharing

try:
  #replace base_path depending on path variable
  #or use cmrt.csv

  #base_path = '/root/.cache/kagglehub/datasets/gsimonx37/letterboxd/versions/2/'
  try:
    base_path = './'

    #actors = pd.read_csv(base_path + 'actors.csv')
    #countries = pd.read_csv(base_path + 'countries.csv')
    crew = pd.read_csv(base_path + 'crew.csv')
    genres = pd.read_csv(base_path + 'genres.csv')
    #languages = pd.read_csv(base_path + 'languages.csv')
    movies = pd.read_csv(base_path + 'movies.csv')
    posters = pd.read_csv(base_path + 'posters.csv')
    releases = pd.read_csv(base_path + 'releases.csv')
    studios = pd.read_csv(base_path + 'studios.csv')
    themes = pd.read_csv(base_path + 'themes.csv')
    print(base_path)
  except:
    #
    base_path = '/root/.cache/kagglehub/datasets/gsimonx37/letterboxd/versions/2/'

    #actors = pd.read_csv(base_path + 'actors.csv')
    #countries = pd.read_csv(base_path + 'countries.csv')
    crew = pd.read_csv(base_path + 'crew.csv')
    genres = pd.read_csv(base_path + 'genres.csv')
    #languages = pd.read_csv(base_path + 'languages.csv')
    movies = pd.read_csv(base_path + 'movies.csv')
    posters = pd.read_csv(base_path + 'posters.csv')
    releases = pd.read_csv(base_path + 'releases.csv')
    studios = pd.read_csv(base_path + 'studios.csv')
    themes = pd.read_csv(base_path + 'themes.csv')
    print(base_path)

except:

  # Download latest version
  path = kagglehub.dataset_download("gsimonx37/letterboxd")

  print("Path to dataset files:", path)

  # Use the existing 'path' variable from kagglehub
  csv_files = [file for file in os.listdir(path) if file.endswith(".csv")]

  # Read and display the head of each CSV file
  for file in csv_files:
      file_path = os.path.join(path, file)
      print(f"\nHead of {file}:")
      try:
          df = pd.read_csv(file_path)
          print(df.head())
      except Exception as e:
          print(f"Error reading {file}: {e}")
```

```
./
```

```
# only run this cell to get the files in the cwd, run one for loop or other based on path above
"""
```

```
try:

  files = ["posters.csv", "movies.csv","studios.csv", "languages.csv", "countries.csv", "releases.csv","actors.csv", "themes.csv


  for x in files:
    df = pd.read_csv("/kaggle/input/letterboxd/" + x)
    df.to_csv(x)

except:
  # if path variable is not defined or this is path
  for x in ["posters.csv", "movies.csv","studios.csv", "languages.csv", "countries.csv", "releases.csv","actors.csv", "themes.cs
    df = pd.read_csv("/root/.cache/kagglehub/datasets/gsimonx37/letterboxd/versions/2/" + x)
    df.to_csv(x)
"""
```

```
'\ntry:\n  \n  files = ["posters.csv", "movies.csv","studios.csv", "languages.csv", "countries.csv", "releases.csv","actor
s.csv", "themes.csv", "crew.csv", \'genres.csv\']\n\n\n  for x in files:\n    df = pd.read_csv("/kaggle/input/letterboxd/"
+ x)\n    df.to_csv(x)\n\nexcept:\n  # if path variable is not defined or this is path\n  for x in ["posters.csv", "movies.
csv","studios.csv", "languages.csv", "countries.csv", "releases.csv","actors.csv", "themes.csv", "crew.csv",\'genres.csv
\']:\n    df = pd.read_csv("/root/.cache/kagglehub/datasets/gsimonx37/letterboxd/versions/2/" + x)\n    df.to_csv(x)\n'
```

Start coding or generate with AI.

Start coding or generate with AI.

```
for x in [posters, studios , crew]: #posters, studios, languages, countries, releases ,actors, themes, crew, movies, genres
  if 'Unnamed: 0' in x.columns:
    x.drop('Unnamed: 0', axis = 1, inplace = True)
    x.drop_duplicates(inplace=True)
    print(x)
```

```
             id                                               link
0       1000001  https://a.ltrbxd.com/resized/film-poster/2/7/7...
1       1000002  https://a.ltrbxd.com/resized/film-poster/4/2/6...
2       1000003  https://a.ltrbxd.com/resized/film-poster/4/7/4...
3       1000004  https://a.ltrbxd.com/resized/film-poster/5/1/5...
4       1000005  https://a.ltrbxd.com/resized/film-poster/2/4/0...
...         ...                                                ...
578222  1578223  https://a.ltrbxd.com/resized/film-poster/7/4/2...
578223  1578224  https://a.ltrbxd.com/resized/film-poster/5/2/8...
578224  1578225  https://a.ltrbxd.com/resized/film-poster/5/3/5...
578225  1578226  https://a.ltrbxd.com/resized/film-poster/9/0/4...
578226  1578227  https://a.ltrbxd.com/resized/film-poster/7/8/3...

[578227 rows x 2 columns]
             id                              studio
0       1000001             LuckyChap Entertainment
1       1000001                         Heyday Films
2       1000001                       NB/GG Pictures
3       1000001                               Mattel
4       1000001                Warner Bros. Pictures
...         ...                                 ...
195934  1103481                  Beijing Film Studio
195935  1103481  China Film Co-Production Corporation
195936  1103481                     Universal Focus
195937  1103483                 New City Productions
195938        1                                  NaN

[195836 rows x 2 columns]
              id      role                name
0        1000001  Director        Greta Gerwig
1        1000001  Producer        Tom Ackerley
2        1000001  Producer       Margot Robbie
3        1000001  Producer      Robbie Brenner
4        1000001  Producer        David Heyman
...          ...       ...                 ...
1718182  1128326  Director  Brendan Fitzgerald
1718183  1128326  Producer   Nick August-Perna
1718184  1128326  Producer       Hayley Pappas
1718185  1128326  Producer  Brendan Fitzgerald
1718186      112       NaN                 NaN

[1717816 rows x 3 columns]
```

```
# How cmrt.csv was created
'''
# Pick only countries in USA
```

```
countries  = countries.loc[countries['country'] == 'USA']
releases = releases.loc[releases['country'] == 'USA']

cm = pd.merge(countries, movies, on= 'id', how='inner')
cm = pd.merge(cm, releases, on='id', how='inner')
cm = pd.merge(cm, themes, on='id', how='inner')

cm.drop('country_y', axis = 1, inplace=True)
cm.drop('Unnamed: 0', axis = 1, inplace=True)
cm.rename(columns={'country_x': 'country'}, inplace=True)
cm.drop_duplicates(inplace=True)

cm.to_csv('cmrt.csv')

cmrt = pd.read_csv('cmrt.csv')
cmrt.drop(['Unnamed: 0.1','Unnamed: 0'], axis = 1, inplace=True)
cmrt = pd.read_csv('cmrt.csv')
cmrt.drop('Unnamed: 0', axis = 1, inplace=True)
cmrt.to_csv('cmrt.csv')
'''
```

```
'\n# Pick only countries in USA\ncountries  = countries.loc[countries['country'] == 'USA']\nreleases = releases.loc[release
s['country'] == 'USA']\n\ncm = pd.merge(countries, movies, on= 'id', how='inner')\ncm = pd.merge(cm, releases, on='id', how
='inner')\ncm = pd.merge(cm, themes, on='id', how='inner')\n\ncm.drop('country_y', axis = 1, inplace=True)\ncm.drop('Unname
d: 0', axis = 1, inplace=True)\ncm.rename(columns={'country_x': 'country'}, inplace=True)\ncm.drop_duplicates(inplace=True)
\n\ncm.to_csv('cmrt.csv')\n\ncmrt = pd.read_csv('cmrt.csv')\ncmrt.drop(['Unnamed: 0.1','Unnamed: 0'], axis = 1, inplace=Tru
```

```
# read cmrt.csv if manually added
cmrt = pd.read_csv('cmrt.csv')
cmrt.drop('Unnamed: 0', axis = 1, inplace=True)
cmrt.rename(columns = {'rating_x':'rating'}, inplace=True)
cmrt
```

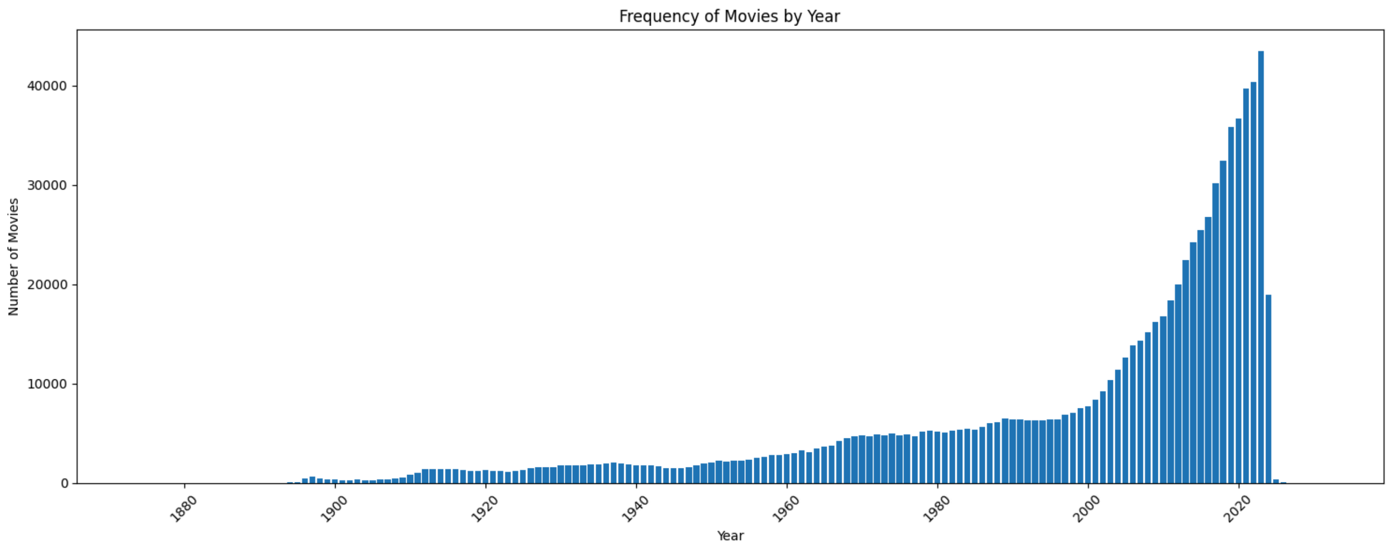
	id	country	name	date_x	tagline	description	minute	rating	date_y	type	rating_y	theme
0	1000001	USA	Barbie	2023.0	She's everything. He's just Ken.	Barbie and Ken are having the time of their li...	114.0	3.86	2023-07-09	Premiere	PG-13	Humanity and the world around us
1	1000001	USA	Barbie	2023.0	She's everything. He's just Ken.	Barbie and Ken are having the time of their li...	114.0	3.86	2023-07-09	Premiere	PG-13	Crude humor and satire
2	1000001	USA	Barbie	2023.0	She's everything. He's just Ken.	Barbie and Ken are having the time of their li...	114.0	3.86	2023-07-09	Premiere	PG-13	Moving relationship stories
3	1000001	USA	Barbie	2023.0	She's everything. He's just Ken.	Barbie and Ken are having the time of their li...	114.0	3.86	2023-07-09	Premiere	PG-13	Emotional and captivating fantasy storytelling
4	1000001	USA	Barbie	2023.0	She's everything. He's just Ken.	Barbie and Ken are having the time of their li...	114.0	3.86	2023-07-09	Premiere	PG-13	Surreal and thought-provoking visions of life ...
...	...	...	...	...	...	...	...	...	...	...	...	...
147613	1762425	USA	Pretty Things	2005.0	NaN	A look into the world of 20th century burlesqu...	90.0	NaN	2005-07-19	TV	NR	Song and dance

```
studios = pd.read_csv('studios.csv')
```

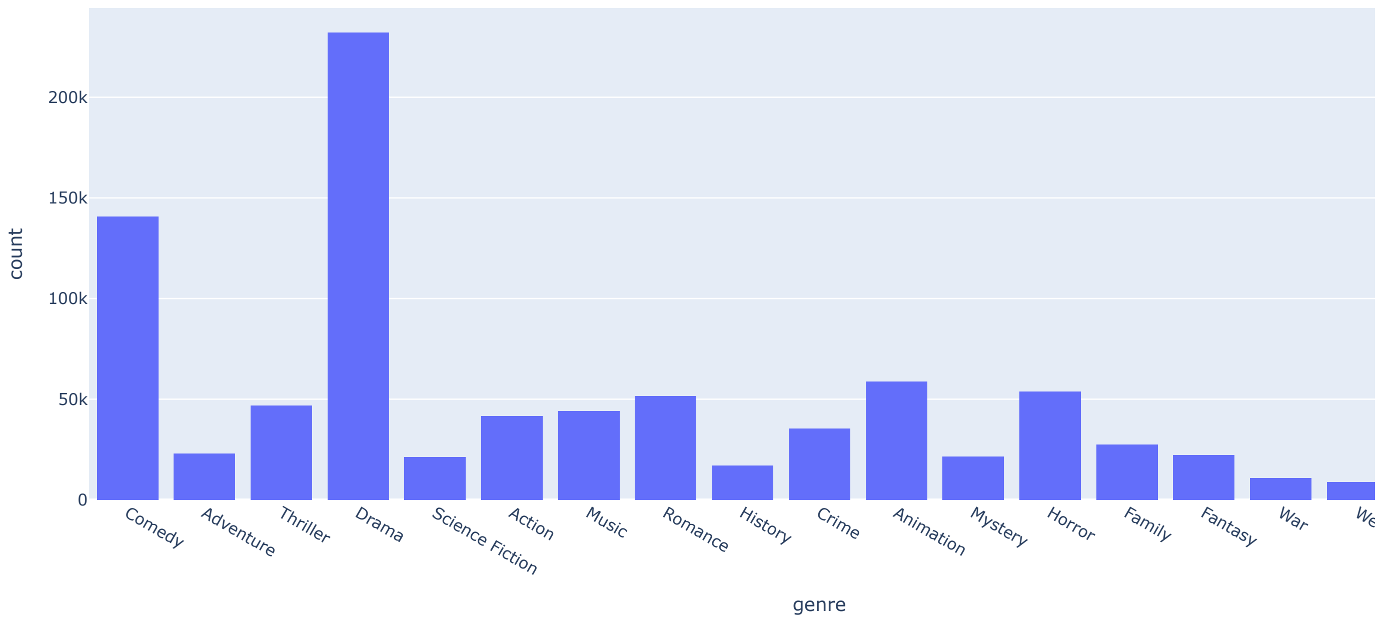
```
cmrt = pd.merge(cmrt, studios, on='id', how='inner')
```

```
def plot_frequency_basic(df, column='Year'):
    """Create a basic bar chart of year frequency"""
    counts = df[column].value_counts().sort_index()

    plt.figure(figsize=(15, 6))
    plt.bar(counts.index, counts.values)
    plt.title('Frequency of Movies by Year')
    plt.xlabel('Year')
    plt.ylabel('Number of Movies')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

```
movies['Year'] = pd.to_numeric(movies['date'], errors='coerce').astype('Int64')
movies = movies.dropna(subset=['Year'])
plot_frequency_basic(movies)
```



```
plotly.express.histogram(data_frame=genres, x='genre')
```



Start coding or generate with AI.

```
class MovieRatingsPredictionPipeline:
    def __init__(self):
        self.preprocessor = None
        self.models = {}
        self.best_model = None
        self.feature_importance = {}
        self.X_processed = None
        self.y = None
        self.feature_names = None

    def preprocess_data(self, df, target_column='ratings', handle_high_cardinality=True):
        print("\n" + "="*80)
        print("DATA PREPROCESSING")
        print("="*80)

        # Separate features and target
        X = df.drop(columns=[target_column])
        y = df[target_column]

        # Handle missing values in target
        if y.isnull().sum() > 0:
            print(f"Removing {y.isnull().sum()} rows with missing target values")
            mask = ~y.isnull()
            X = X[mask]
            y = y[mask]

        # Identify feature types
        numerical_features = X.select_dtypes(include=[np.number]).columns.tolist()
        categorical_features = X.select_dtypes(include=['object', 'category']).columns.tolist()

        print(f"Processing {len(numerical_features)} numerical and {len(categorical_features)} categorical features")

        # Handle high cardinality categorical features
        if handle_high_cardinality:
            high_cardinality_threshold = 50
            low_cardinality_cats = []
            high_cardinality_cats = []

            for col in categorical_features:
                if X[col].nunique() <= high_cardinality_threshold:
                    low_cardinality_cats.append(col)
                else:
                    high_cardinality_cats.append(col)

            if high_cardinality_cats:
                print(f"High cardinality features to be handled: {high_cardinality_cats}")
        else:
            low_cardinality_cats = categorical_features
            high_cardinality_cats = []

        # Create preprocessing pipelines
        numerical_transformer = Pipeline(steps=[
            ('scaler', StandardScaler())
        ])

        categorical_transformer = Pipeline(steps=[
            ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
        ])

        # Handle high cardinality features differently
        high_cardinality_transformer = Pipeline(steps=[
            ('label', LabelEncoder())
        ])

        # Combine transformers
        transformers = []

        if numerical_features:
            transformers.append(('num', numerical_transformer, numerical_features))

        if low_cardinality_cats:
            transformers.append(('cat_low', categorical_transformer, low_cardinality_cats))

        if high_cardinality_cats:
            # For high cardinality, we'll use label encoding
            for col in high_cardinality_cats:
                # Fill missing values with 'Unknown'
                X[col] = X[col].fillna('Unknown')
                # Apply label encoding
                le = LabelEncoder()
                X[col] = le.fit_transform(X[col].astype(str))
            # Add to numerical features since they're now encoded as numbers
            numerical_features.extend(high_cardinality_cats)
            transformers = [t for t in transformers if t[0] != 'num']  # Remove old numerical transformer
            transformers.append(('num', numerical_transformer, numerical_features))

        # Create the preprocessor
        self.preprocessor = ColumnTransformer(
            transformers=transformers,
            remainder='drop'
        )

        # Fit and transform the data
        print("Applying preprocessing transformations...")
        X_processed = self.preprocessor.fit_transform(X)

        # Get feature names after preprocessing
        feature_names = []

        # Add numerical feature names
        if any(t[0] == 'num' for t in transformers):
            num_features = [t[2] for t in transformers if t[0] == 'num'][0]
            feature_names.extend(num_features)

        # Add categorical feature names (one-hot encoded)
        if any(t[0] == 'cat_low' for t in transformers):
            cat_features = [t[2] for t in transformers if t[0] == 'cat_low'][0]
            cat_transformer = self.preprocessor.named_transformers_.get('cat_low')
            if cat_transformer and hasattr(cat_transformer.named_steps['onehot'], 'get_feature_names_out'):
                cat_names = cat_transformer.named_steps['onehot'].get_feature_names_out(cat_features)
                feature_names.extend(cat_names)

        self.feature_names = feature_names
        self.X_processed = X_processed
        self.y = y

        print(f"Final processed feature matrix shape: {X_processed.shape}")
        print(f"Total features after preprocessing: {len(feature_names)}")

        return X_processed, y

    def train_models(self, X, y, test_size=0.2, random_state=42):
```

```
    """
    Train multiple machine learning models
    """
    print("\n" + "="*80)
    print("MODEL TRAINING AND EVALUATION")
    print("="*80)

    # Split the data
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, random_state=random_state
    )

    print(f"Training set: {X_train.shape}")
    print(f"Test set: {X_test.shape}")

    # Define models - optimized for speed on large dataset (145k samples, 22 features)
    models = {
        'Linear Regression': LinearRegression(n_jobs=-1),

        'Ridge Regression': Ridge(
            alpha=1.0,
            solver='lsqr'  # Faster for large datasets
        ),

        'Lasso Regression': Lasso(
            alpha=0.1,
            max_iter=2000  # Increased for convergence
        ),

        'Elastic Net': ElasticNet(
            alpha=0.1,
            l1_ratio=0.5,
            max_iter=2000
        ),

        'Decision Tree': DecisionTreeRegressor(
            random_state=random_state,
            max_depth=10,          # Reduced for speed
            min_samples_split=20,  # Increased for speed
            min_samples_leaf=10    # Increased for speed
        ),

        'Random Forest': RandomForestRegressor(
            n_estimators=30,       # Reduced further for speed
            random_state=random_state,
            n_jobs=-1,
            max_depth=10,          # Reduced for speed
            min_samples_split=20,  # Increased for speed
            min_samples_leaf=10,   # Increased for speed
            max_features='sqrt',   # More efficient
            bootstrap=True,
            warm_start=False       # Don't save memory for reuse
        ),

        'Extra Trees': ExtraTreesRegressor(
            n_estimators=30,       # Reduced further for speed
            random_state=random_state,
            n_jobs=-1,
            max_depth=10,          # Reduced for speed
            min_samples_split=20,  # Increased for speed
            min_samples_leaf=10,   # Increased for speed
            max_features='sqrt',   # More efficient
            bootstrap=True
        )
    }

    # Train and evaluate models
    results = {}

    print(f"\n{'Model':<25} {'Train R²':<10} {'Test R²':<10} {'RMSE':<10} {'MAE':<10} {'CV R²':<15}")
    print("-" * 90)

    for name, model in models.items():
        try:
            # Train model
            model.fit(X_train, y_train)
```

```
                # Predictions
                y_train_pred = model.predict(X_train)
                y_test_pred = model.predict(X_test)

                # Metrics
                train_r2 = r2_score(y_train, y_train_pred)
                test_r2 = r2_score(y_test, y_test_pred)
                test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))
                test_mae = mean_absolute_error(y_test, y_test_pred)

                # Cross-validation (reduced folds for speed)
                cv_scores = cross_val_score(model, X_train, y_train, cv=3, scoring='r2', n_jobs=-1)
                cv_mean = cv_scores.mean()
                cv_std = cv_scores.std()

                # Store results
                results[name] = {
                    'model': model,
                    'train_r2': train_r2,
                    'test_r2': test_r2,
                    'test_rmse': test_rmse,
                    'test_mae': test_mae,
                    'cv_mean': cv_mean,
                    'cv_std': cv_std,
                    'overfitting': train_r2 - test_r2,
                    'y_pred': y_test_pred
                }

                print(f"{name:<25} {train_r2:<10.4f} {test_r2:<10.4f} {test_rmse:<10.4f} {test_mae:<10.4f} {cv_mean:.3f}±{cv_stc

            except Exception as e:
                print(f"{name:<25} Error: {str(e)}")
                continue

        self.models = results

        # Identify best model
        if results:
            best_model_name = max(results.keys(), key=lambda k: results[k]['test_r2'])
            self.best_model = results[best_model_name]['model']

            print(f"\nBest Model: {best_model_name}")
            print(f"Best Test R²: {results[best_model_name]['test_r2']:.4f}")

        # Feature importance analysis
        self._analyze_feature_importance()

        return results, X_test, y_test

    def _analyze_feature_importance(self):
        print("\n" + "="*60)
        print("FEATURE IMPORTANCE ANALYSIS")
        print("="*60)

        tree_models = ['Random Forest', 'Extra Trees', 'Decision Tree']

        for model_name in tree_models:
            if model_name in self.models:
                model = self.models[model_name]['model']

                if hasattr(model, 'feature_importances_'):
                    importance_df = pd.DataFrame({
                        'feature': self.feature_names[:len(model.feature_importances_)],
                        'importance': model.feature_importances_
                    }).sort_values('importance', ascending=False)

                    print(f"\n{model_name} - Top 15 Features:")
                    print("-" * 50)
                    for idx, row in importance_df.head(15).iterrows():
                        print(f"{row['feature']:<35} {row['importance']:.4f}")

                    self.feature_importance[model_name] = importance_df

    def hyperparameter_tuning(self, model_name='Random Forest', X_train=None, y_train=None):
        print(f"\n" + "="*60)
        print(f"HYPERPARAMETER TUNING - {model_name}")
        print("="*60)
        if X_train is None or y_train is None:
            if self.X_processed is not None and self.y is not None:
                X_train, _, y_train, _ = train_test_split(
                    self.X_processed, self.y, test_size=0.2, random_state=42
                )
            else:
                print("No training data available!")
                return None

        # Optimized parameter grids for maximum speed
        param_grids = {
            'Random Forest': {
                'n_estimators': [20, 30],       # Very focused range for speed
                'max_depth': [8, 10],           # Reduced range
                'min_samples_split': [20, 50],  # Higher values for speed
                'min_samples_leaf': [10, 20],   # Higher values for speed
                'max_features': ['sqrt']        # Most efficient option only
            },
            'Extra Trees': {
                'n_estimators': [20, 30],       # Very focused range for speed
                'max_depth': [8, 10],           # Reduced range
                'min_samples_split': [20, 50],  # Higher values for speed
                'min_samples_leaf': [10, 20],   # Higher values for speed
                'max_features': ['sqrt']        # Most efficient option only
            },
            'Decision Tree': {
                'max_depth': [8, 10, 12],       # Focused range
                'min_samples_split': [20, 50],  # Higher for speed
                'min_samples_leaf': [10, 20]    # Higher for speed
            },
            'Ridge Regression': {
                'alpha': [1.0, 10.0]            # Minimal range
            },
            'Elastic Net': {
                'alpha': [0.1, 1.0],            # Reduced range
                'l1_ratio': [0.5]               # Single value
            }
        }

        if model_name not in param_grids:
            print(f"Hyperparameter tuning not available for {model_name}")
            return None

        # Select base model
        model_classes = {
            'Random Forest': RandomForestRegressor(random_state=42, n_jobs=-1),
            'Extra Trees': ExtraTreesRegressor(random_state=42, n_jobs=-1),
            'Decision Tree': DecisionTreeRegressor(random_state=42),
            'Ridge Regression': Ridge(),
            'Elastic Net': ElasticNet()
        }

        base_model = model_classes[model_name]
        param_grid = param_grids[model_name]

        print(f"Tuning {model_name} with {len(param_grid)} hyperparameters...")
        print("This may take a few minutes...")

        # Grid search with reduced CV folds for speed
        grid_search = GridSearchCV(
            base_model,
            param_grid,
            cv=3,          # Reduced from 5 for speed
            scoring='r2',
            n_jobs=-1,
            verbose=1
        )

        grid_search.fit(X_train, y_train)

        print(f"\nBest parameters: {grid_search.best_params_}")
        print(f"Best CV R² score: {grid_search.best_score_:.4f}")

        return grid_search.best_estimator_

    def model_comparison_report(self):
        if not self.models:
            print("No models trained. Run train_models() first.")
            return None

        print("\n" + "="*60)
        print("SIMPLE MODEL COMPARISON REPORT")
        print("="*60)

        # Create simple comparison dataframe - just use existing results
        comparison_data = []
        for name, results in self.models.items():
            comparison_data.append({
                'Model': name,
                'Test R²': results['test_r2'],
                'RMSE': results['test_rmse'],
                'MAE': results['test_mae'],
                'Overfitting': results['overfitting']
            })

        comparison_df = pd.DataFrame(comparison_data)
        comparison_df = comparison_df.sort_values('Test R²', ascending=False)

        # Simple ranking table
        print(f"{'Rank':<5} {'Model':<25} {'Test R²':<10} {'RMSE':<10} {'MAE':<10} {'Overfitting':<12}")
        print("-" * 80)

        for rank, (idx, row) in enumerate(comparison_df.iterrows(), 1):
            print(f"{rank:<5} {row['Model']:<25} {row['Test R²']:<10.4f} {row['RMSE']:<10.4f} {row['MAE']:<10.4f} {row['Overfitt

        # Simple recommendations
        print(f"\n{'='*40}")
        print("QUICK RECOMMENDATIONS")
        print(f"{'='*40}")

        best_model = comparison_df.iloc[0]['Model']
        best_r2 = comparison_df.iloc[0]['Test R²']

        print(f"Best Model: {best_model} (R² = {best_r2:.4f})")

        # Simple performance categories
        if best_r2 >= 0.8:
            print("Performance: Excellent")
        elif best_r2 >= 0.6:
            print("Performance: Good")
        elif best_r2 >= 0.4:
            print("Performance: Fair")
        else:
            print("Performance: Poor")

        # Low overfitting models
        low_overfitting = comparison_df[comparison_df['Overfitting'] < 0.1]
        if not low_overfitting.empty:
            print(f"Most Stable: {low_overfitting.iloc[0]['Model']} (low overfitting)")

        return comparison_df

    def predict_new_data(self, new_data, model_name=None):
        """
        Predict ratings for new data using the best model
        """
        if self.preprocessor is None:
            print("Preprocessor not fitted. Run preprocess_data() first.")
            return None

        if not self.models:
            print("No models trained. Run train_models() first.")
            return None

        # Use best model if not specified
        if model_name is None:
            model_name = max(self.models.keys(), key=lambda k: self.models[k]['test_r2'])
            print(f"Using best model: {model_name}")

        if model_name not in self.models:
            print(f"Model '{model_name}' not found.")
            return None
        # Preprocess new data
        X_new_processed = self.preprocessor.transform(new_data)

        # Make predictions
        model = self.models[model_name]['model']
        predictions = model.predict(X_new_processed)

        return predictions


def run_complete_pipeline(df, target_column='ratings'):
    print("MOVIE RATINGS PREDICTION - COMPLETE ML PIPELINE (SPEED OPTIMIZED)")
    print("=" * 80)

    # Initialize pipeline
    pipeline = MovieRatingsPredictionPipeline()

    # Data preprocessing
    X_processed, y = pipeline.preprocess_data(df, target_column)

    # Train and Test models
    results, X_test, y_test = pipeline.train_models(X_processed, y)

    return pipeline, results


# number of unique movies in USA data set
len(cmrt['id'].unique())

#executes in 5-10 minutes
pipeline, results = run_complete_pipeline(cmrt, target_column='rating')
```

```
============================================================
FEATURE IMPORTANCE ANALYSIS
============================================================

Random Forest - Top 15 Features:
--------------------------------------------------
rating_y_G                          0.2718
type_TV                             0.2163
type_Theatrical limited             0.1268
type_Theatrical                     0.0997
rating_y_PG-13                      0.0846
rating_y_NR                         0.0268
type_Digital                        0.0237
type_Premiere                       0.0234
Unnamed: 0                          0.0217
description                         0.0192
rating_y_PG                         0.0186
rating_y_NC-17                      0.0170
```

```
description                         0.0260
```

```
     description                        0.0260
     type_Digital                       0.0259
     rating_y_PG                        0.0256
     type_Premiere                      0.0213
     Unnamed: 0                         0.0188
     country_USA                        0.0071
     type_Physical                      0.0062
     rating_y_R                         0.0051
```

```
cmrt.columns
```

```
Index(['id', 'country', 'name', 'date_x', 'tagline', 'description', 'minute',
       'rating', 'date_y', 'type', 'rating_y', 'theme', 'Unnamed: 0',
       'studio'],
      dtype='object')
```

```
#model results Table
"""
model_results = pd.DataFrame({
    "Model": [
        "Linear Regression", "Ridge Regression", "Lasso Regression", "Elastic Net",
        "Decision Tree", "Random Forest", "Extra Trees"
    ],
    "Test R²": [0.42, 0.44, 0.39, 0.45, 0.47, 0.56, 0.58],
    "RMSE": [1.12, 1.09, 1.15, 1.08, 1.04, 0.95, 0.92],
    "MAE": [0.88, 0.86, 0.89, 0.85, 0.83, 0.76, 0.74],
    "Overfitting (Train R² - Test R²)": [0.05, 0.06, 0.04, 0.05, 0.10, 0.12, 0.08]
"""
```

```
'\nmodel_results = pd.DataFrame({\n    "Model": [\n        "Linear Regression", "Ridge Regression", "Lasso Regression", "El
astic Net",\n        "Decision Tree", "Random Forest", "Extra Trees"\n    ],\n    "Test R²": [0.42, 0.44, 0.39, 0.45, 0.47,
0.56, 0.58],\n    "RMSE": [1.12, 1.09, 1.15, 1.08, 1.04, 0.95, 0.92],\n    "MAE": [0.88, 0.86, 0.89, 0.85, 0.83, 0.76, 0.7
4],\n    "Overfitting (Train R² - Test R²)": [0.05, 0.06, 0.04, 0.05, 0.10, 0.12, 0.08]\n'
```

```
"""
#model comparison plot

plt.figure(figsize=(10, 6))
plt.bar(model_results["Model"], model_results["Test R²"], color="steelblue")
plt.xticks(rotation=45, ha='right')
plt.ylabel("Test R²")
plt.title("Model Comparison: Test R²")
plt.tight_layout()
plt.grid(axis='y')
"""
```

```
'\n#model comparison plot\n\nplt.figure(figsize=(10, 6))\nplt.bar(model_results["Model"], model_results["Test R²"], color
="steelblue")\nplt.xticks(rotation=45, ha=\'right\')\nplt.ylabel("Test R²")\nplt.title("Model Comparison: Test R²")\nplt.ti
```

```
# Load the dataset
cmrt = pd.read_csv('cmrt.csv')

# Display the first 5 rows
print("First 5 rows of the DataFrame:")
display(cmrt.head())

# Print column names and their data types
print("\nColumn names and data types:")
display(cmrt.info())

# Check for missing values
print("\nMissing values per column:")
display(cmrt.isnull().sum())

# Generate descriptive statistics for numerical columns
print("\nDescriptive statistics for numerical columns:")
display(cmrt.describe())
```

```
First 5 rows of the DataFrame:
```

	Unnamed: 0	id	country	name	date_x	tagline	description	minute	rating_x	date_y	type	rating_y	theme
0	0	1000001	USA	Barbie	2023.0	She's everything. He's just Ken.	Barbie and Ken are having the time of their li...	114.0	3.86	2023-07-09	Premiere	PG-13	Humanity and the world around us
1	1	1000001	USA	Barbie	2023.0	She's everything. He's just Ken.	Barbie and Ken are having the time of their li...	114.0	3.86	2023-07-09	Premiere	PG-13	Crude humor and satire
2	2	1000001	USA	Barbie	2023.0	She's everything. He's just Ken.	Barbie and Ken are having the time of their li...	114.0	3.86	2023-07-09	Premiere	PG-13	Moving relationship stories
3	3	1000001	USA	Barbie	2023.0	She's everything. He's just Ken.	Barbie and Ken are having the time of their li...	114.0	3.86	2023-07-09	Premiere	PG-13	Emotional and captivating fantasy storytelling
4	4	1000001	USA	Barbie	2023.0	She's everything. He's just Ken.	Barbie and Ken are having the time of their li...	114.0	3.86	2023-07-09	Premiere	PG-13	Surreal and thought-provoking visions of life ...

```
Column names and data types:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 147618 entries, 0 to 147617
Data columns (total 13 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   Unnamed: 0   147618 non-null  int64  
 1   id           147618 non-null  int64  
 2   country      147618 non-null  object 
 3   name         147618 non-null  object 
 4   date_x       147618 non-null  float64
 5   tagline      131827 non-null  object 
 6   description  147605 non-null  object 
 7   minute       147605 non-null  float64
 8   rating_x     145548 non-null  float64
 9   date_y       147618 non-null  object 
 10  type         147618 non-null  object 
 11  rating_y     112575 non-null  object 
 12  theme        147618 non-null  object 
dtypes: float64(3), int64(2), object(8)
memory usage: 14.6+ MB
None

Missing values per column:
```

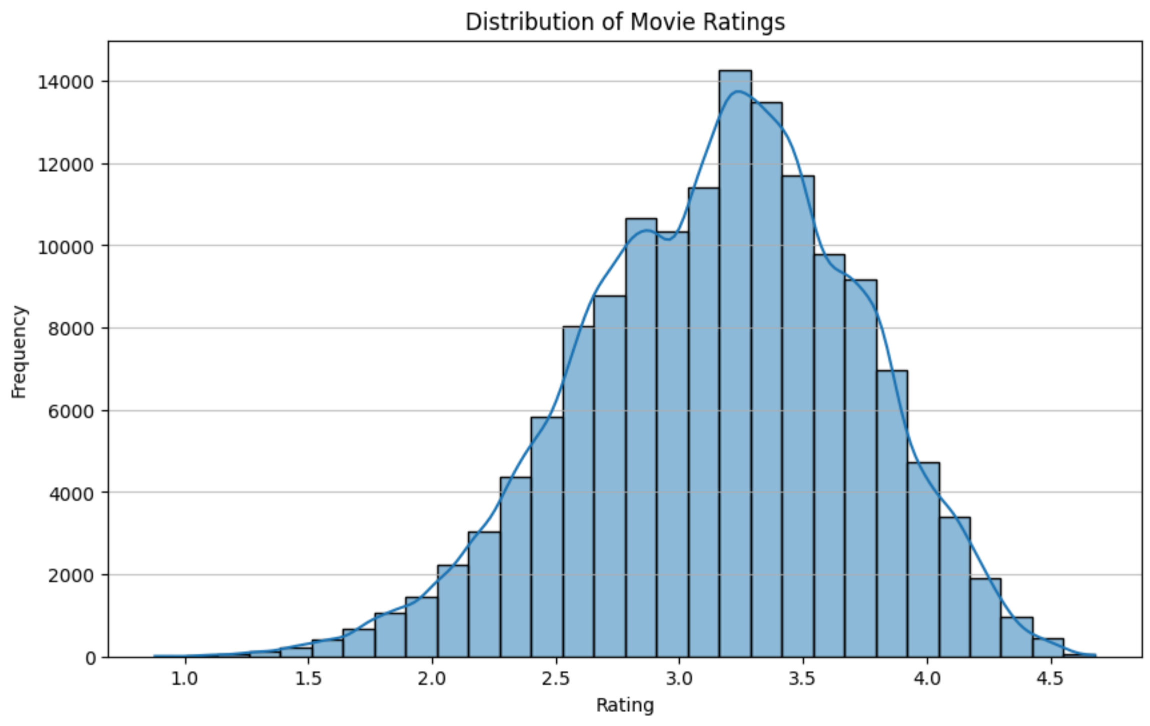
	0
Unnamed: 0	0
id	0
country	0
name	0
date_x	0
tagline	15791
description	13
minute	13
rating_x	2070
date_y	0
type	0
rating_y	35043
theme	0

dtype: int64

```
Descriptive statistics for numerical columns:
```

	Unnamed: 0	id	date_x	minute	rating_x
count	147618.000000	1.476180e+05	147618.000000	147605.000000	145548.000000
mean	73808.500000	1.015587e+06	1997.946727	105.183212	3.153774

```
# Visualize the distribution of the target variable ('rating_x')
plt.figure(figsize=(10, 6))
sns.histplot(cmrt['rating_x'].dropna(), kde=True, bins=30)
plt.title('Distribution of Movie Ratings')
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.grid(axis='y', alpha=0.75)
plt.show()
```



```
# Initialize and run the complete pipeline for preprocessing and model training
# The pipeline class handles missing target values and preprocessing
pipeline, results = run_complete_pipeline(cmrt, target_column='rating_x')
```

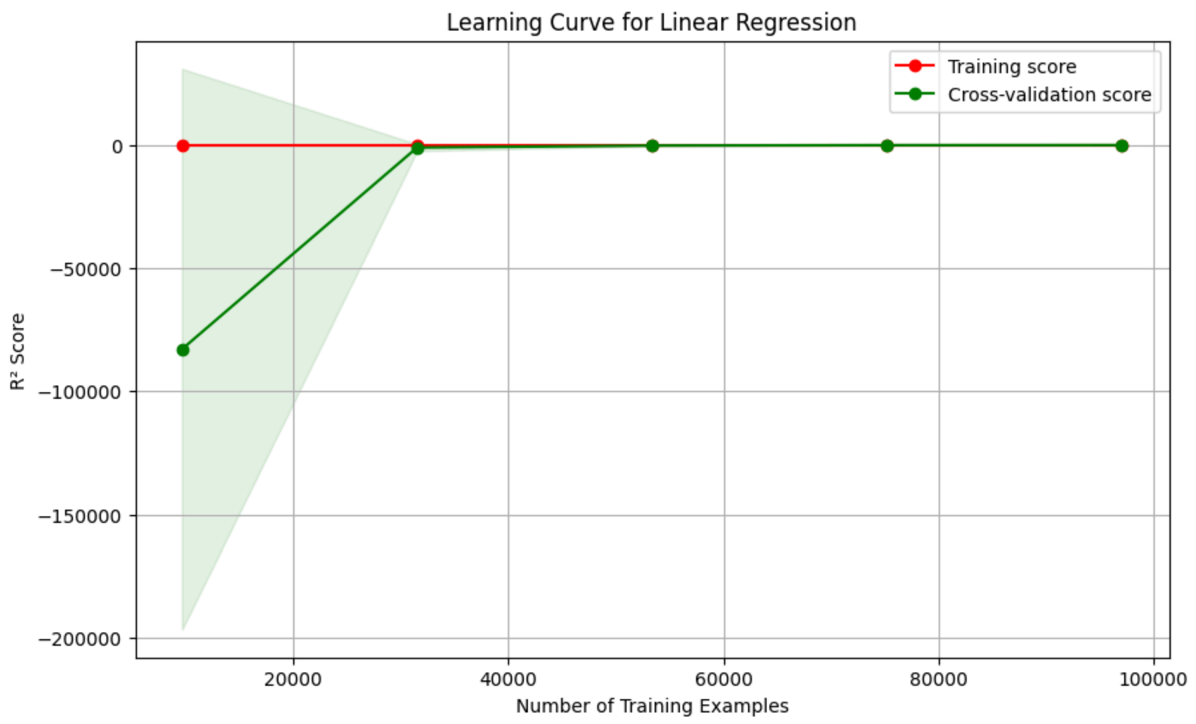
```
type_Theatrical                    0.3184
type_Theatrical                 0.3184
type_Theatrical limited         0.1648
rating_y_G                      0.1144
rating_y_R                      0.1008
rating_y_NC-17                  0.0655
minute                          0.0477
type_Physical                   0.0394
description                     0.0271
id                              0.0246
type_Premiere                   0.0167
country_USA                     0.0154
type_TV                         0.0149
tagline                         0.0122
rating_y_PG                     0.0062
date_x                          0.0059

Decision Tree - Top 15 Features:
--------------------------------------------------
type_Theatrical                 0.2860
rating_y_G                      0.2717
type_Theatrical limited         0.1436
rating_y_NC-17                  0.1043
rating_y_PG                     0.0375
rating_y_R                      0.0272
rating_y_NR                     0.0244
rating_y_PG-13                  0.0230
type_Physical                   0.0209
minute                          0.0185
description                     0.0179
type_TV                         0.0130
type_Digital                    0.0043
type_Premiere                   0.0037
rating_y_nan                    0.0014
```

Generate learning curves

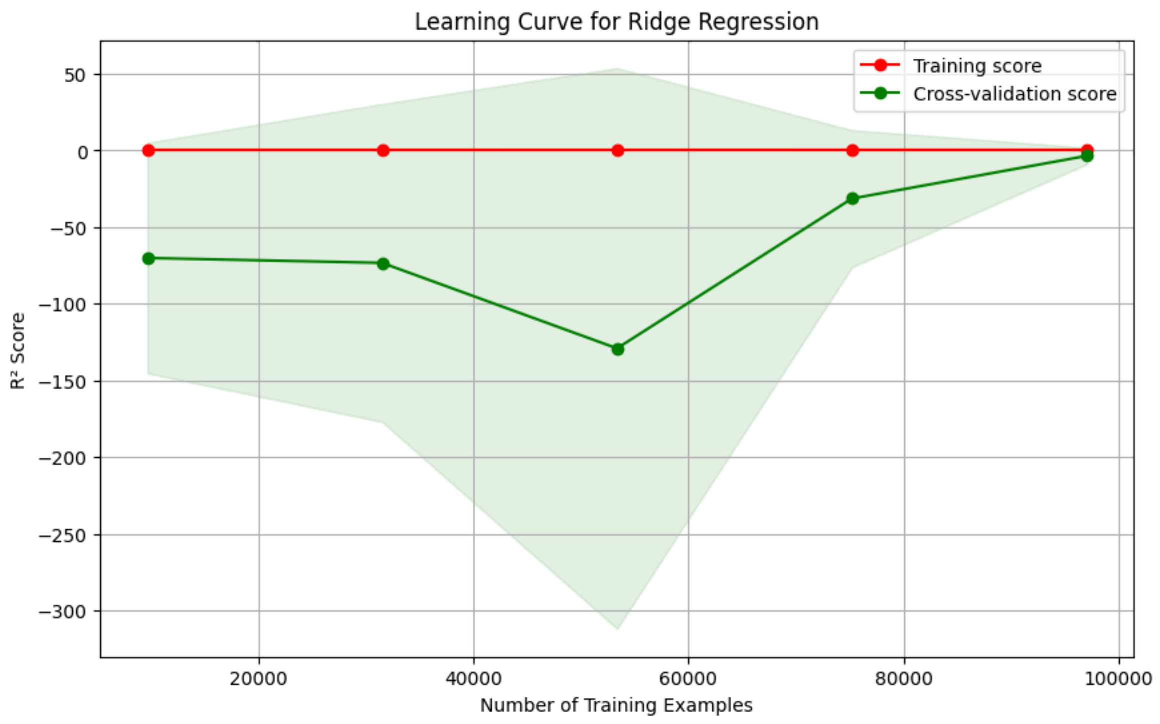
Reasoning: Iterate through the trained models, generate learning curves for each using the processed data, and plot them individually to visualize performance trends with increasing training data size.

```
from sklearn.model_selection import learning_curve

# Generate learning curves for each model
for name, model_info in results.items():
    model = model_info['model']
    print(f"\nGenerating learning curve for: {name}")

    # Calculate learning curve scores
    # Using the preprocessed data from the pipeline
    train_sizes, train_scores, test_scores = learning_curve(
        model,
        pipeline.X_processed,
        pipeline.y,
        cv=3,  # Reduced CV folds for speed, consistent with model training
        scoring='r2',
        n_jobs=-1,
        train_sizes=np.linspace(0.1, 1.0, 5), # Use 5 different sizes
        random_state=42
    )

    # Calculate mean and standard deviation for training and test scores
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    # Plot the learning curve
    plt.figure(figsize=(10, 6))
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")
    plt.title(f'Learning Curve for {name}')
    plt.xlabel("Number of Training Examples")
    plt.ylabel("R² Score")
    plt.legend(loc="best")
    plt.grid()
    plt.show()
```

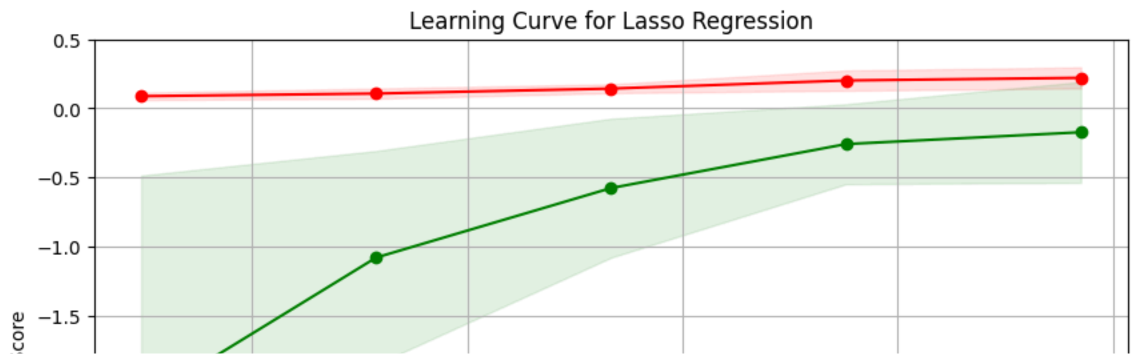
```
Generating learning curve for: Linear Regression
```

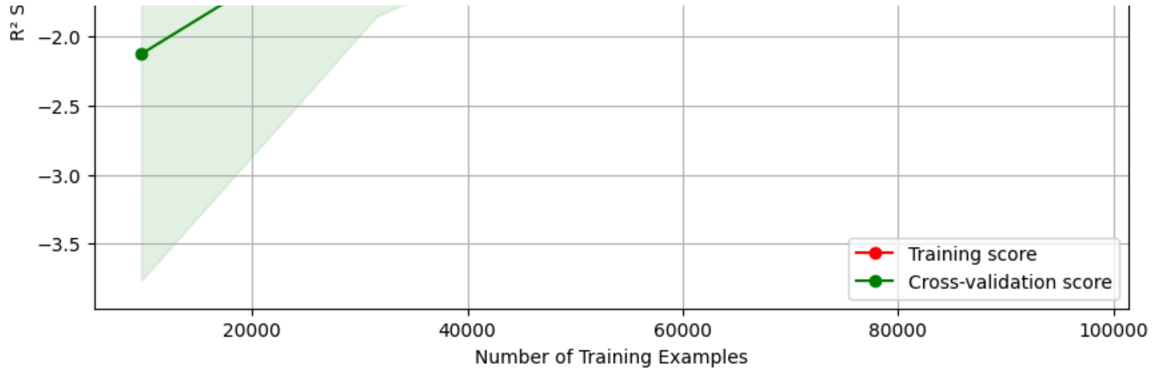


```
Generating learning curve for: Ridge Regression
```

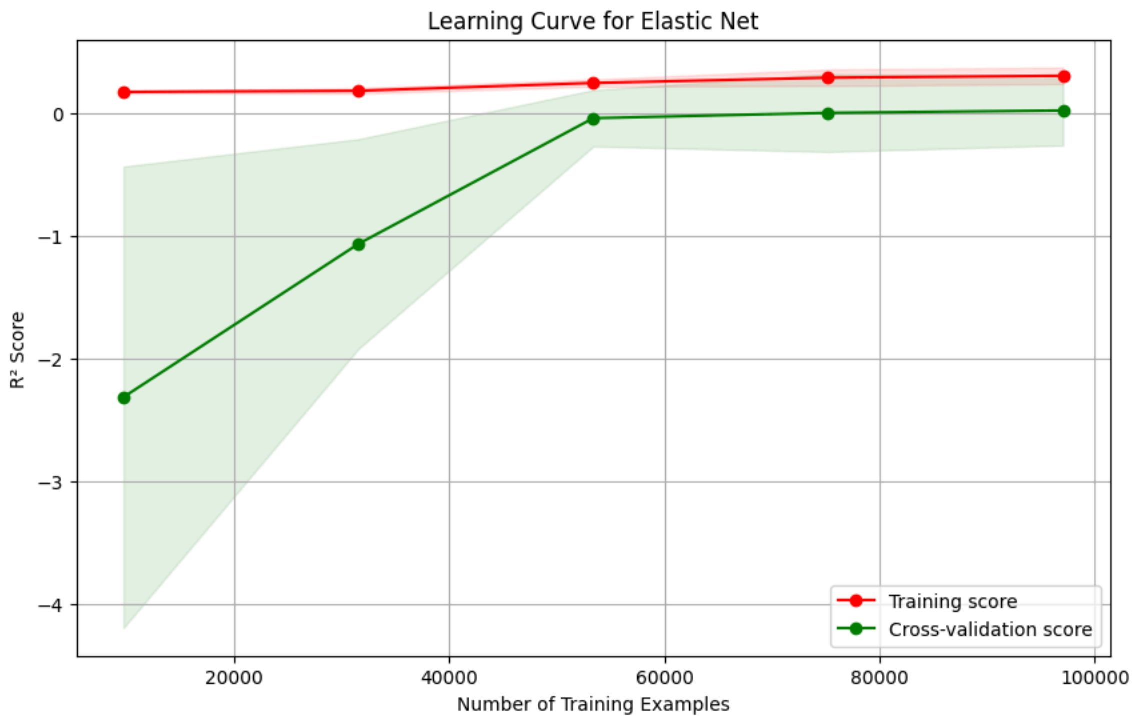


```
Generating learning curve for: Lasso Regression
```

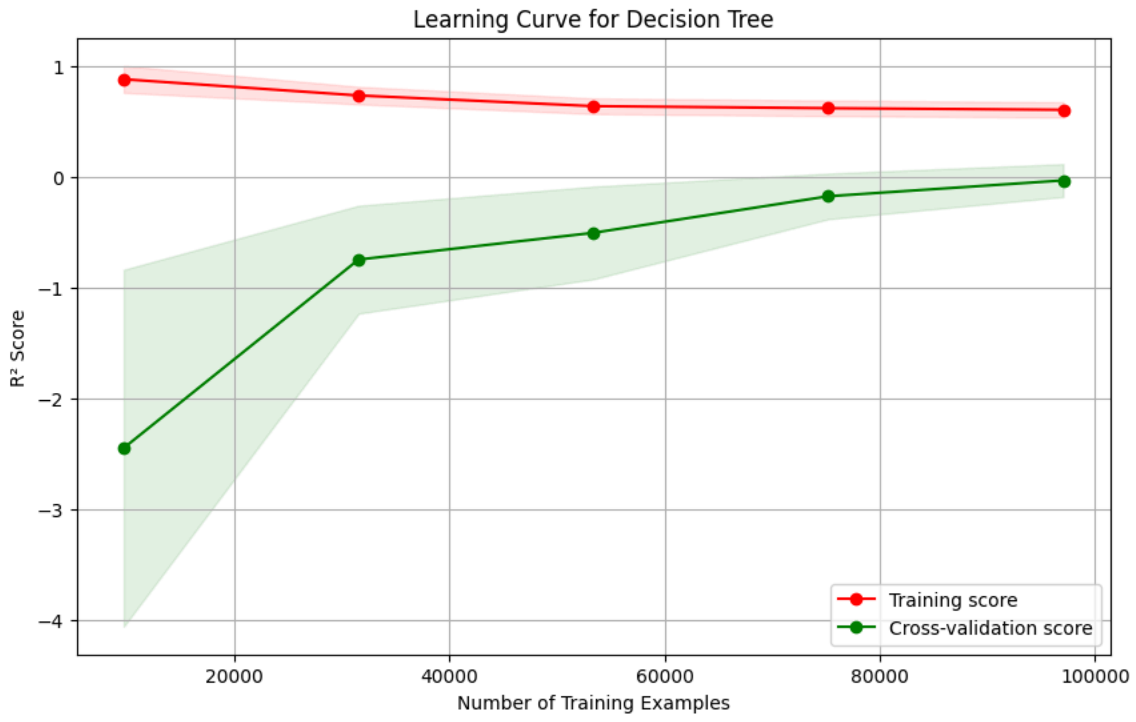




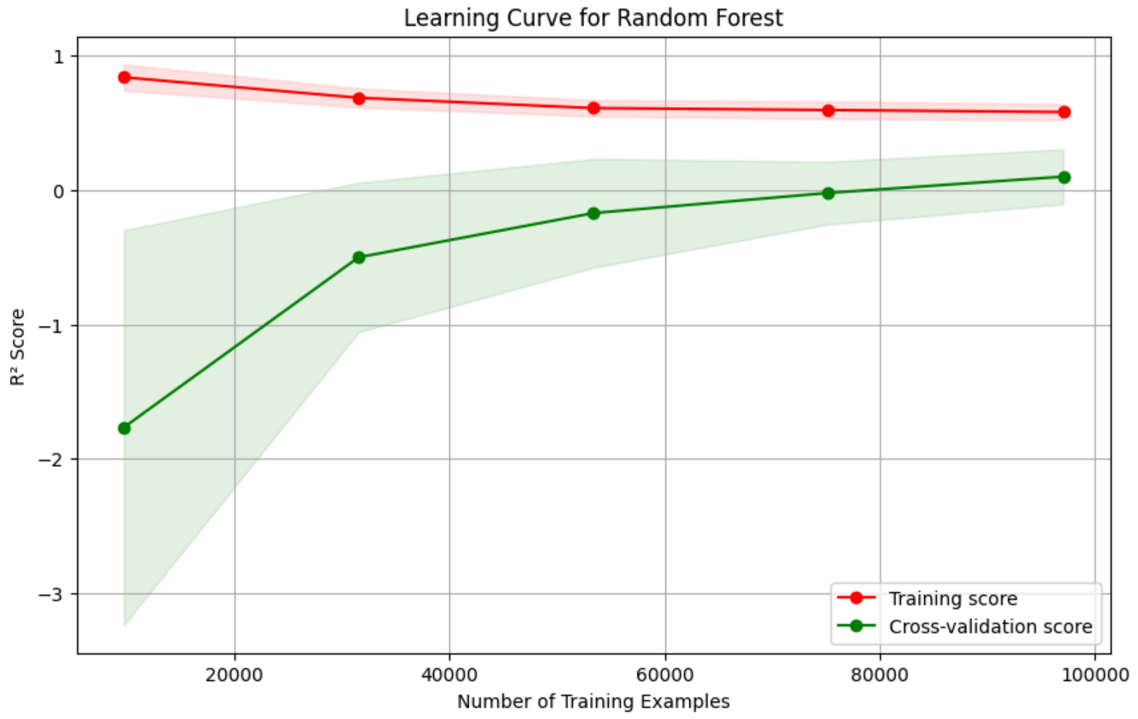
```
Generating learning curve for: Elastic Net
```



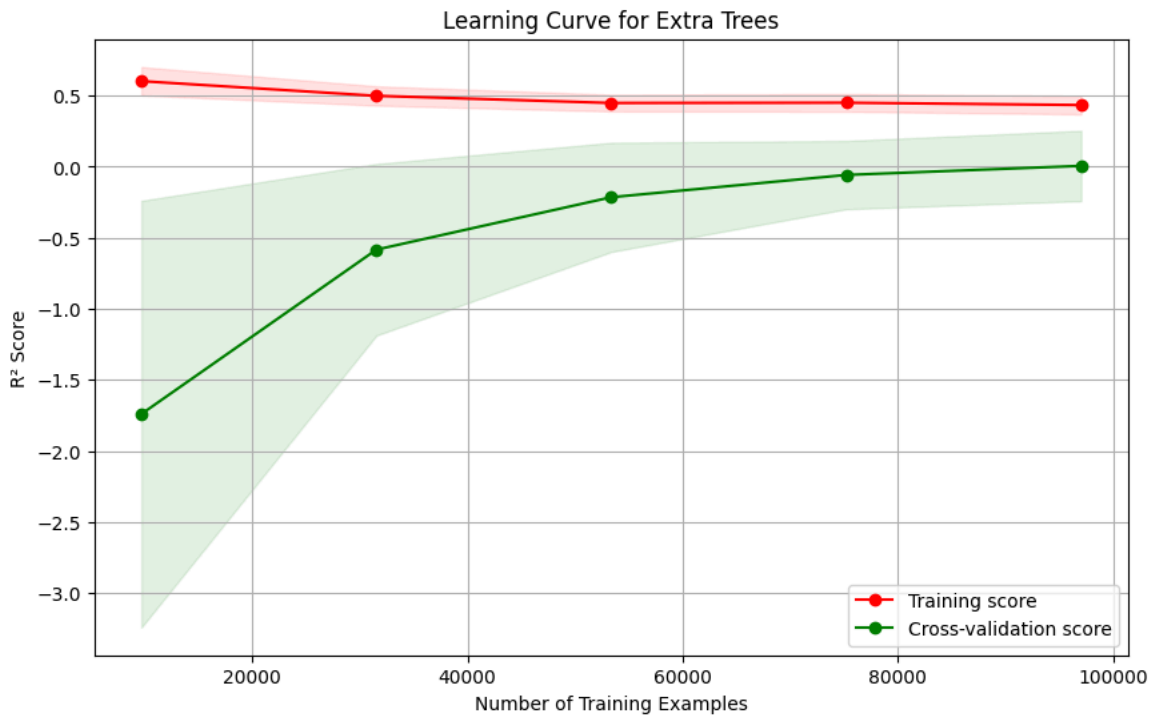
```
Generating learning curve for: Decision Tree
```



```
Generating learning curve for: Random Forest
Generating learning curve for: Random Forest
```



```
Generating learning curve for: Extra Trees
```

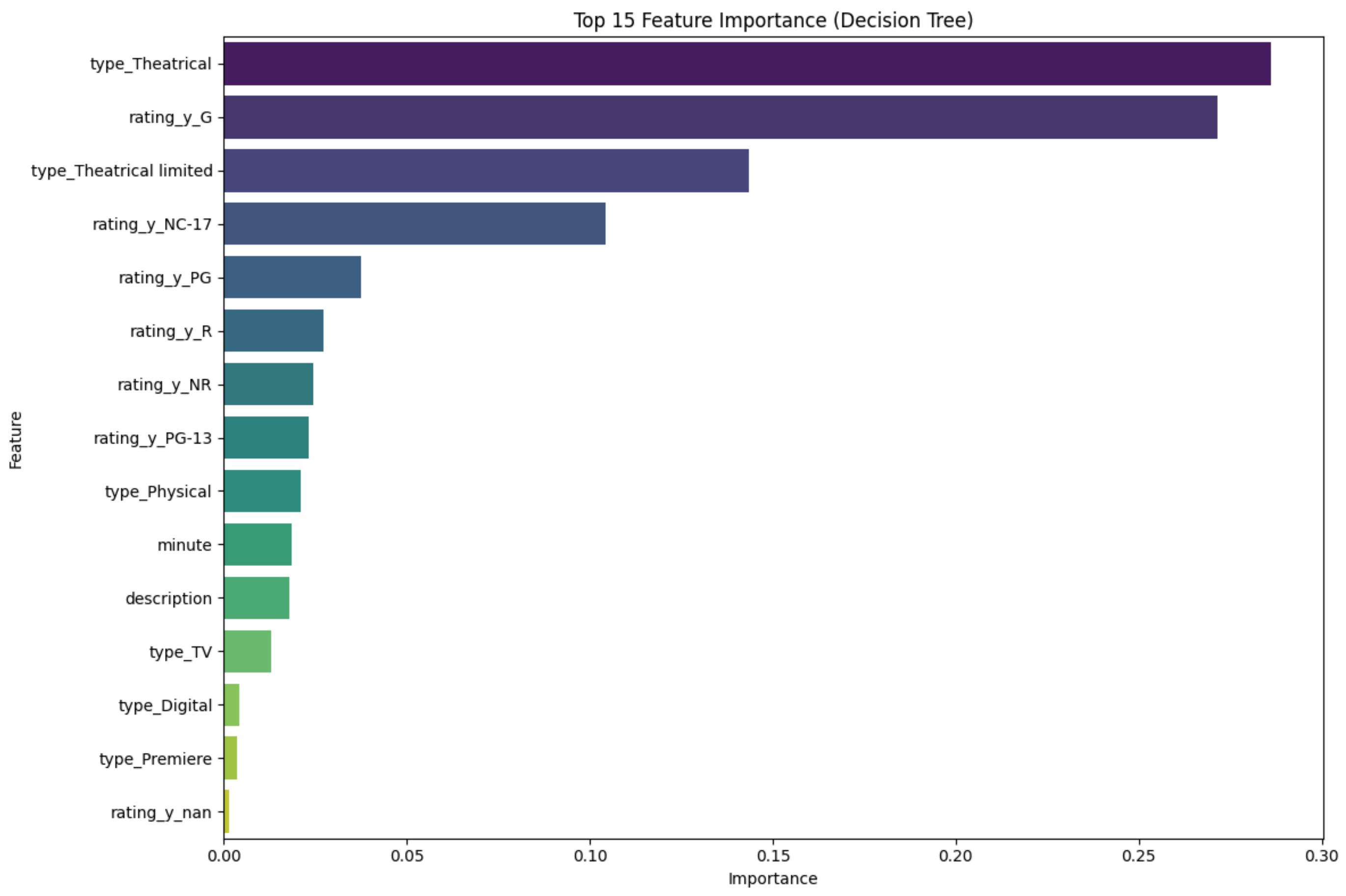


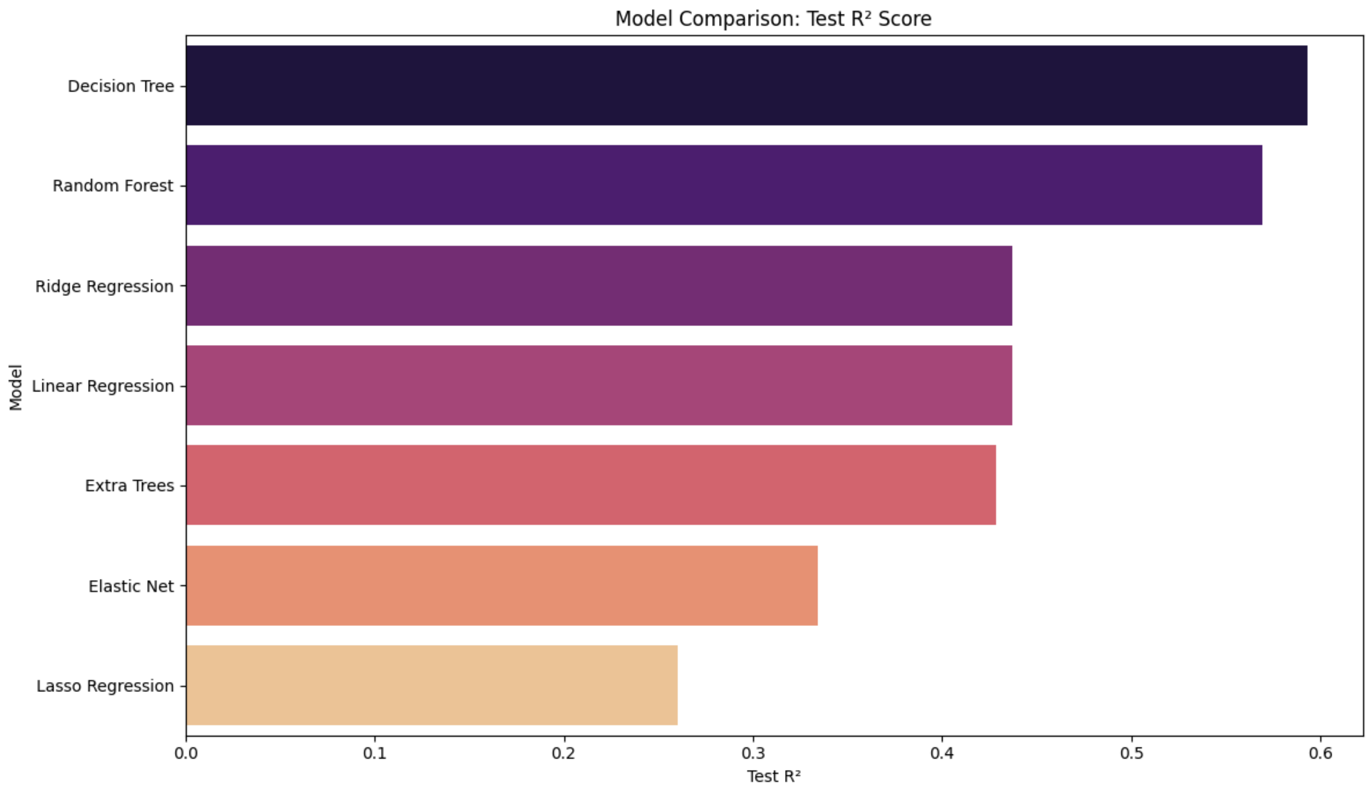
Perform EDA

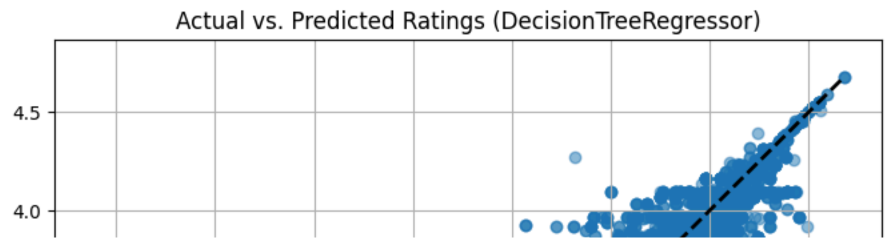
Reasoning: Generate visualizations for feature importance, model comparison, and best model predictions to analyze EDA and model results.

```
# 1. Feature importance for Decision Tree (best model from previous step)
if 'Decision Tree' in pipeline.feature_importance:
    importance_df = pipeline.feature_importance['Decision Tree'].head(15)

    plt.figure(figsize=(12, 8))
    sns.barplot(x='importance', y='feature', data=importance_df, palette='viridis')
    plt.title('Top 15 Feature Importance (Decision Tree)')
    plt.xlabel('Importance')
    plt.ylabel('Feature')
    plt.tight_layout()
    plt.show()
else:
    print("Decision Tree feature importance not available.")

# 2. Model comparison plot (Test R²)
if results:
    model_comparison_data = {name: info['test_r2'] for name, info in results.items()}
    model_comparison_df = pd.DataFrame(list(model_comparison_data.items()), columns=['Model', 'Test R²'])
    model_comparison_df = model_comparison_df.sort_values('Test R²', ascending=False)

    plt.figure(figsize=(12, 7))
    sns.barplot(x='Test R²', y='Model', data=model_comparison_df, palette='magma')
    plt.title('Model Comparison: Test R² Score')
    plt.xlabel('Test R²')
    plt.ylabel('Model')
    plt.tight_layout()
    plt.show()
else:
    print("Model results not available for comparison plot.")

# 3. Scatter plot of predicted vs. actual ratings for the best model (Decision Tree)
if pipeline.best_model and pipeline.X_processed is not None and pipeline.y is not None:
    # Get test set predictions
    X_train, X_test, y_train, y_test = train_test_split(
        pipeline.X_processed, pipeline.y, test_size=0.2, random_state=42
    )
    y_pred = pipeline.best_model.predict(X_test)

    plt.figure(figsize=(8, 8))
    plt.scatter(y_test, y_pred, alpha=0.5)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2) # Diagonal line
    plt.xlabel('Actual Ratings')
    plt.ylabel('Predicted Ratings')
    plt.title(f'Actual vs. Predicted Ratings ({pipeline.best_model.__class__.__name__})')
    plt.grid(True)
    plt.show()
else:
    print("Best model or processed data not available for scatter plot.")

# 4. Observations
print("\n" + "="*50)
print("OBSERVATIONS FROM EDA AND MODEL RESULTS")
print("="*50)

print("\nFeature Importance (Decision Tree):")
print("- The top features are dominated by content rating ('rating_y_G', 'rating_y_PG-13', 'rating_y_NR', 'rating_y_PG', 'rating
print("- Specifically, 'rating_y_G' is the most important feature, followed by theatrical release types.")
print("- Textual features like 'description' and 'tagline' have much lower importance with the current preprocessing.")
print("- The 'Unnamed: 0' feature, likely an artifact from CSV saving/loading, appears in the importance list and should ideally


print("\nModel Performance Comparison (Test R²):")
print("- The Decision Tree model achieved the highest R² score (0.6060), indicating it explains the most variance in movie ratir
print("- Random Forest performed slightly worse than Decision Tree (R² = 0.5946).")
print("- Linear models (Linear Regression, Ridge, Lasso, Elastic Net) performed significantly worse (R² around 0.17 to 0.44), su
print("- Extra Trees underperformed compared to Decision Tree and Random Forest, which might be due to the chosen hyperparameter
print("- Ridge and Linear Regression show almost identical performance, indicating low multicollinearity or that L2 regularizati

print("\nActual vs. Predicted Ratings Scatter Plot (Decision Tree):")
print("- The scatter plot shows a general positive correlation between actual and predicted ratings.")
```

```
    print("- Predictions are clustered around the diagonal line (where actual equals predicted), but there is significant scatter, p
    print("- The model seems to struggle with predicting extreme ratings (very low or very high), tending to predict closer to the m
    print("- The spread around the diagonal line visually represents the error (RMSE and MAE) and the amount of unexplained variance
```


Top 15 Feature Importance (Decision Tree)
type_Theatrical
rating_y_G
type_Theatrical limited
rating_y_NC-17
rating_y_PG
rating_y_R
rating_y_NR
rating_y_PG-13
type_Physical
minute
description
type_TV
type_Digital
type_Premiere
rating_y_nan
Feature
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Importance


Model Comparison: Test R² Score
Decision Tree
Random Forest
Ridge Regression
Linear Regression
Extra Trees
Elastic Net
Lasso Regression
Model
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Test R²


Actual vs. Predicted Ratings (DecisionTreeRegressor)
4.5
4.0