Exploratory Data Analysis & Predictive Modeling

Course – Predictive Analytics

MOVIE RECOMMENDATION ANALYSIS

Submitted by – **Navyaa Kabra**

Registration Number – **229301073**

Submitted to – **Mr. Abhay Bisht**

Submission Date – **November 29, 2024**

Table of Contents

Introduction

In this project, I will showcase my understanding of Exploratory Data Analysis (EDA) and predictive modeling techniques. By working with a real-world dataset, I aim to uncover patterns, insights, and develop predictive capabilities that can address relevant business or research questions.
Exploratory Data Analysis is a crucial first step in any data-driven project, as it allows us to thoroughly inspect and understand the characteristics of the data. This lays the foundation for more advanced statistical analysis and modeling. Predictive modeling, on the other hand, involves applying machine learning algorithms to make forecasts or predictions about future outcomes based on the available data.
Together, EDA and predictive modeling form a powerful combination that can yield valuable insights and actionable intelligence. In this report, I will demonstrate how I have leveraged these techniques to extract meaningful information and build predictive models on the chosen dataset.

Key Objectives:

1. **Data Analysis**: Explore and visualize trends in the movie industry, such as release patterns, genre popularity, and performance metrics.
2. **Feature Engineering**: Process and transform data, including handling dates, encoding categorical variables, and scaling features for predictive modeling.
3. **Predictive Modeling**: Build and evaluate regression models (Linear Regression, Decision Tree Regressor) to predict outcomes like movie revenue or ratings.
4. **Visualization**: Generate clear and insightful visual representations of data trends and model performance.

Dataset Overview

For this project, I have selected the "Movie Data" dataset, which contains information about various movies, including their titles, release dates, user scores, genres, and other related attributes.

The dataset was sourced from Kaggle, a popular platform for open-source datasets and machine learning competitions. It consists of four CSV files:

- Movies.csv: This file contains the core movie information, such as title, release date, user score, and genres.
- FilmDetails.csv: This dataset provides additional details about the movies, including the language.
- MoreInfo.csv: This file includes supplementary information that could be used to enrich the analysis.
- PosterPath.csv: This file contains the paths to movie poster images, which could be utilized for visual analysis.

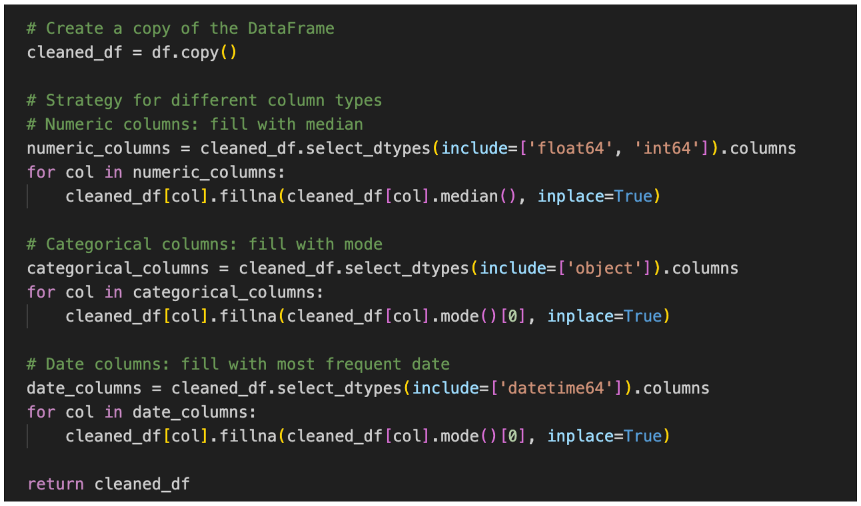
The objective of this project is to perform an in-depth exploratory data analysis on the movie dataset and develop predictive models to forecast user scores for movies based on the available features.

Data Preparation

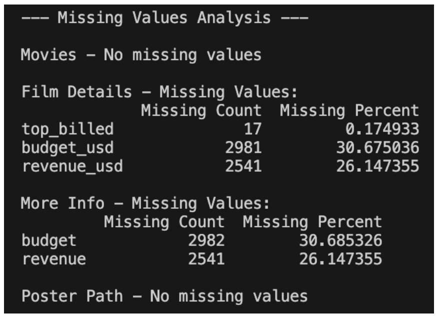
Before diving into the exploratory data analysis and predictive modeling, I first needed to prepare the data for analysis. This involved the following steps:

Data Cleaning:

- Handling missing values: I checked the dataset for any missing values and decided on the appropriate strategies to address them, such as dropping rows with missing data or imputing values based on the feature characteristics.

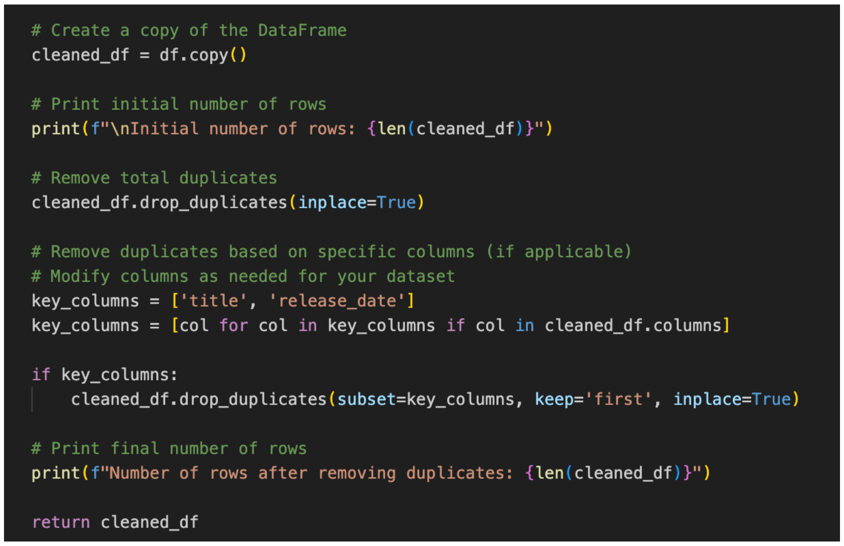


```python
# Create a copy of the DataFrame
cleaned_df = df.copy()

# Strategy for different column types
# Numeric columns: fill with median
numeric_columns = cleaned_df.select_dtypes(include=['float64', 'int64']).columns
for col in numeric_columns:
    cleaned_df[col].fillna(cleaned_df[col].median(), inplace=True)

# Categorical columns: fill with mode
categorical_columns = cleaned_df.select_dtypes(include=['object']).columns
for col in categorical_columns:
    cleaned_df[col].fillna(cleaned_df[col].mode()[0], inplace=True)

# Date columns: fill with most frequent date
date_columns = cleaned_df.select_dtypes(include=['datetime64']).columns
for col in date_columns:
    cleaned_df[col].fillna(cleaned_df[col].mode()[0], inplace=True)

return cleaned_df
```

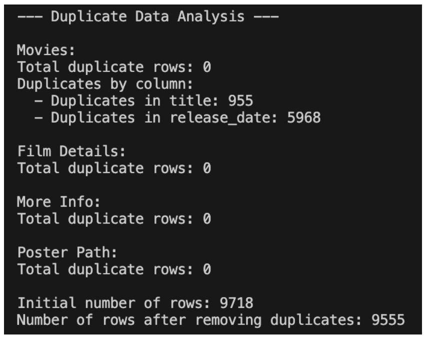


```
--- Missing Values Analysis ---

Movies - No missing values

Film Details - Missing Values:
             Missing Count  Missing Percent
top_billed              17         0.174933
budget_usd            2981        30.675036
revenue_usd           2541        26.147355

More Info - Missing Values:
         Missing Count  Missing Percent
budget            2982        30.685326
revenue           2541        26.147355

Poster Path - No missing values
```

- Removing duplicates: I checked the dataset for any duplicate rows and removed them, ensuring the data is clean and ready for further analysis.



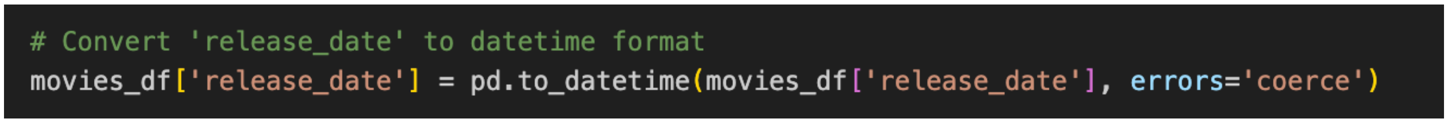
```python
# Create a copy of the DataFrame
cleaned_df = df.copy()

# Print initial number of rows
print(f"\nInitial number of rows: {len(cleaned_df)}")

# Remove total duplicates
cleaned_df.drop_duplicates(inplace=True)

# Remove duplicates based on specific columns (if applicable)
# Modify columns as needed for your dataset
key_columns = ['title', 'release_date']
key_columns = [col for col in key_columns if col in cleaned_df.columns]

if key_columns:
    cleaned_df.drop_duplicates(subset=key_columns, keep='first', inplace=True)

# Print final number of rows
print(f"Number of rows after removing duplicates: {len(cleaned_df)}")

return cleaned_df
```



```
--- Duplicate Data Analysis ---

Movies:
Total duplicate rows: 0
Duplicates by column:
  - Duplicates in title: 955
  - Duplicates in release_date: 5968

Film Details:
Total duplicate rows: 0

More Info:
Total duplicate rows: 0

Poster Path:
Total duplicate rows: 0

Initial number of rows: 9718
Number of rows after removing duplicates: 9555
```

Feature Engineering:

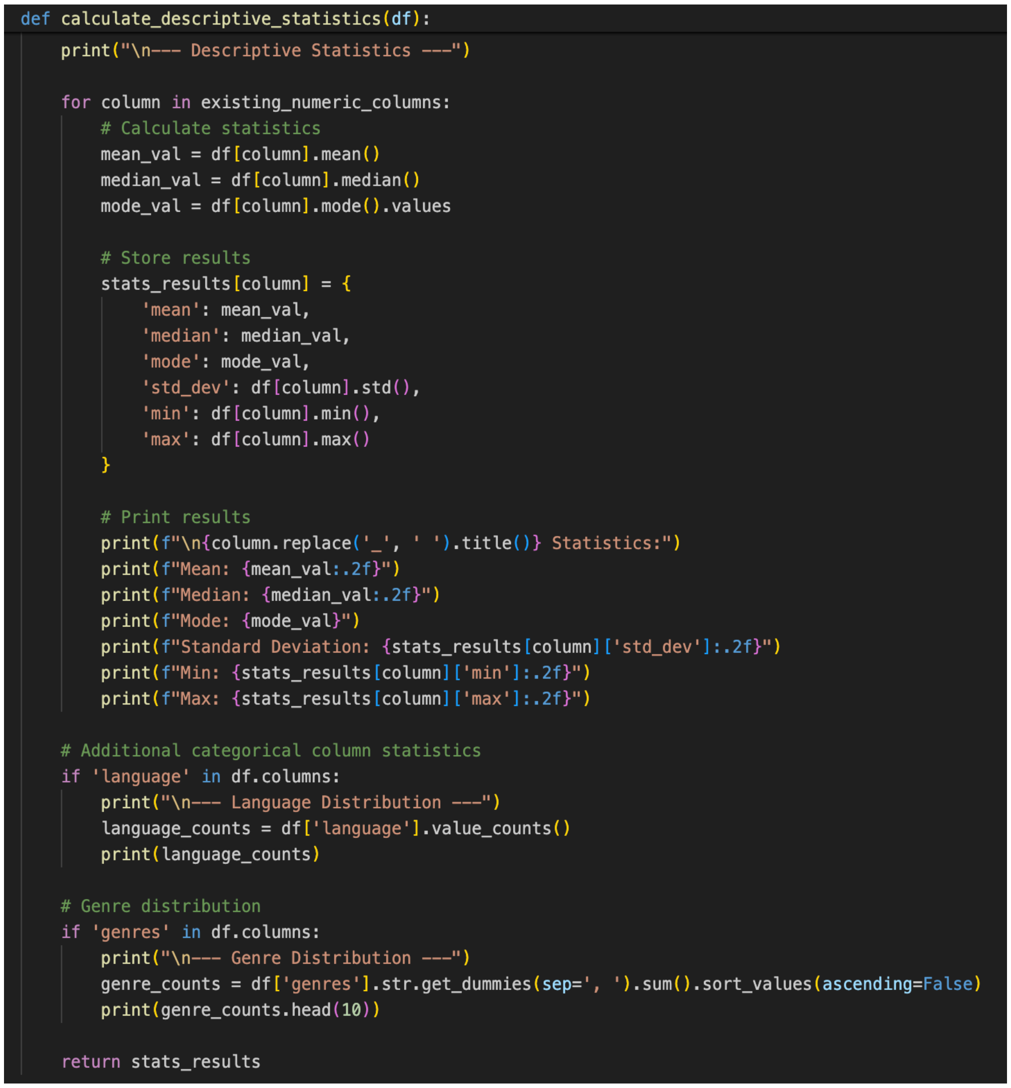
- Converting release date to datetime format: The `release_date` column was initially in string format, so I converted it to a datetime format to enable analysis based on the release year.
- Encoding categorical variables: The dataset contained several categorical features, such as language and genres. I used label encoding and one-hot encoding techniques to transform these variables into a format suitable for machine learning models.



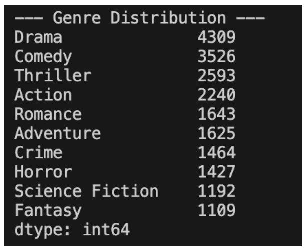
```
# Convert 'release_date' to datetime format
movies_df['release_date'] = pd.to_datetime(movies_df['release_date'], errors='coerce')
```

Summary Statistics:

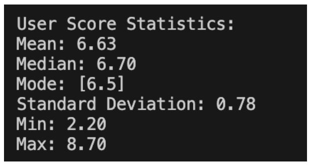
After the data preparation steps, I calculated the summary statistics for each feature in the dataset, including the mean, median, mode, standard deviation, minimum, and maximum values. This provided me with a high-level understanding of the data distribution and characteristics.



```
def calculate_descriptive_statistics(df):
    print("\n--- Descriptive Statistics ---")

    for column in existing_numeric_columns:
        # Calculate statistics
        mean_val = df[column].mean()
        median_val = df[column].median()
        mode_val = df[column].mode().values

        # Store results
        stats_results[column] = {
            'mean': mean_val,
            'median': median_val,
            'mode': mode_val,
            'std_dev': df[column].std(),
            'min': df[column].min(),
            'max': df[column].max()
        }

        # Print results
        print(f"\n{column.replace('_', ' ').title()} Statistics:")
        print(f"Mean: {mean_val:.2f}")
        print(f"Median: {median_val:.2f}")
        print(f"Mode: {mode_val}")
        print(f"Standard Deviation: {stats_results[column]['std_dev']:.2f}")
        print(f"Min: {stats_results[column]['min']:.2f}")
        print(f"Max: {stats_results[column]['max']:.2f}")

    # Additional categorical column statistics
    if 'language' in df.columns:
        print("\n--- Language Distribution ---")
        language_counts = df['language'].value_counts()
        print(language_counts)

    # Genre distribution
    if 'genres' in df.columns:
        print("\n--- Genre Distribution ---")
        genre_counts = df['genres'].str.get_dummies(sep=', ').sum().sort_values(ascending=False)
        print(genre_counts.head(10))

    return stats_results
```



```
--- Genre Distribution ---
Drama                4309
Comedy               3526
Thriller             2593
Action               2240
Romance              1643
Adventure            1625
Crime                1464
Horror               1427
Science Fiction      1192
Fantasy              1109
dtype: int64
```



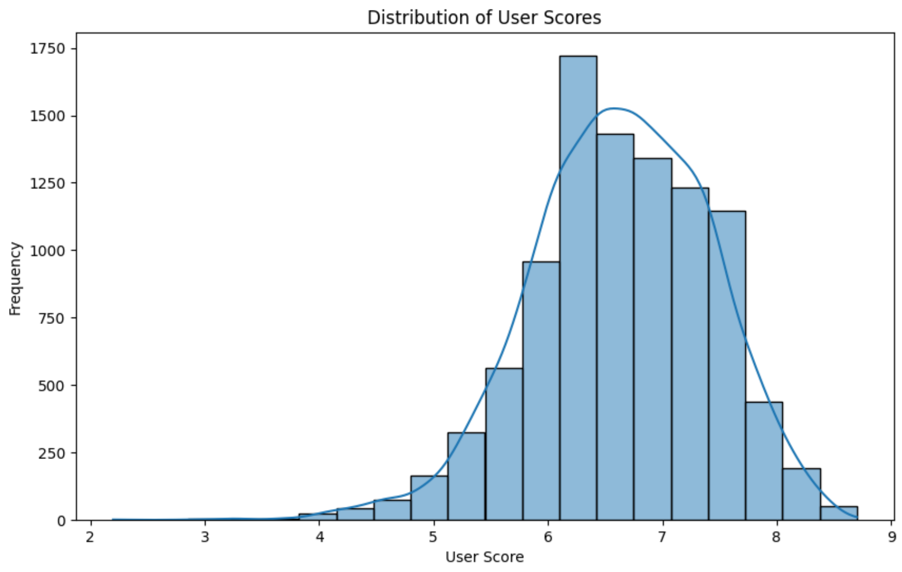
```
User Score Statistics:
Mean: 6.63
Median: 6.70
Mode: [6.5]
Standard Deviation: 0.78
Min: 2.20
Max: 8.70
```

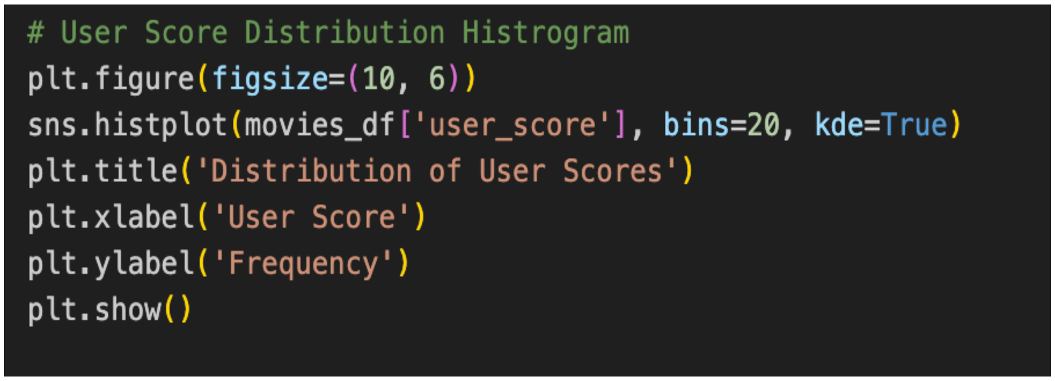
Exploratory Data Analysis

With the dataset now prepared, I proceeded to conduct a comprehensive exploratory data analysis to uncover insights and patterns within the data.

Data Distributions:

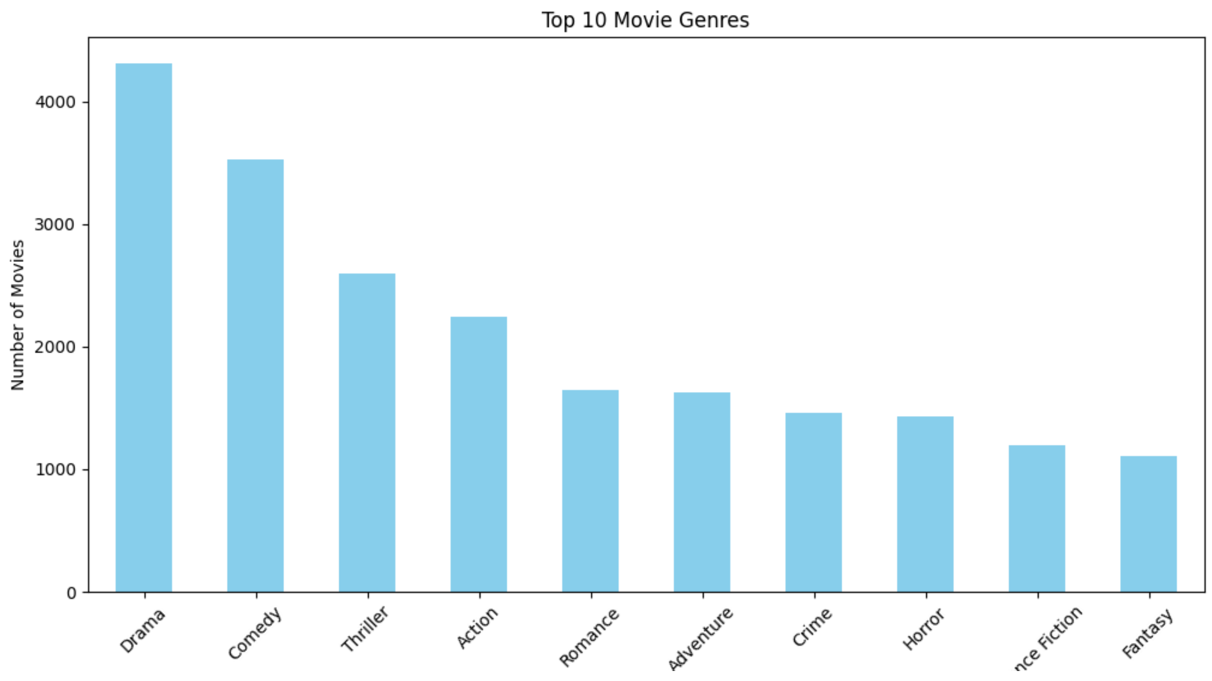
I started by visualizing the distributions of key variables, such as the user score distribution using a histogram. This revealed that the user scores follow a roughly normal distribution, with a mean around 7 and a standard deviation of approximately 0.8.





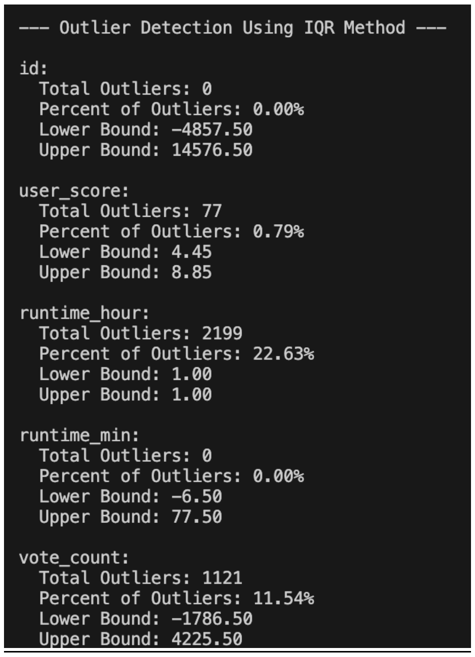
```
# User Score Distribution Histrogram
plt.figure(figsize=(10, 6))
sns.histplot(movies_df['user_score'], bins=20, kde=True)
plt.title('Distribution of User Scores')
plt.xlabel('User Score')
plt.ylabel('Frequency')
plt.show()
```

I also analyzed the distribution of movie genres by creating a bar plot of the top 10 most common genres. This showed that the dataset contains a diverse range of movie genres, with the most prevalent being Drama, Comedy, Action, and Adventure.

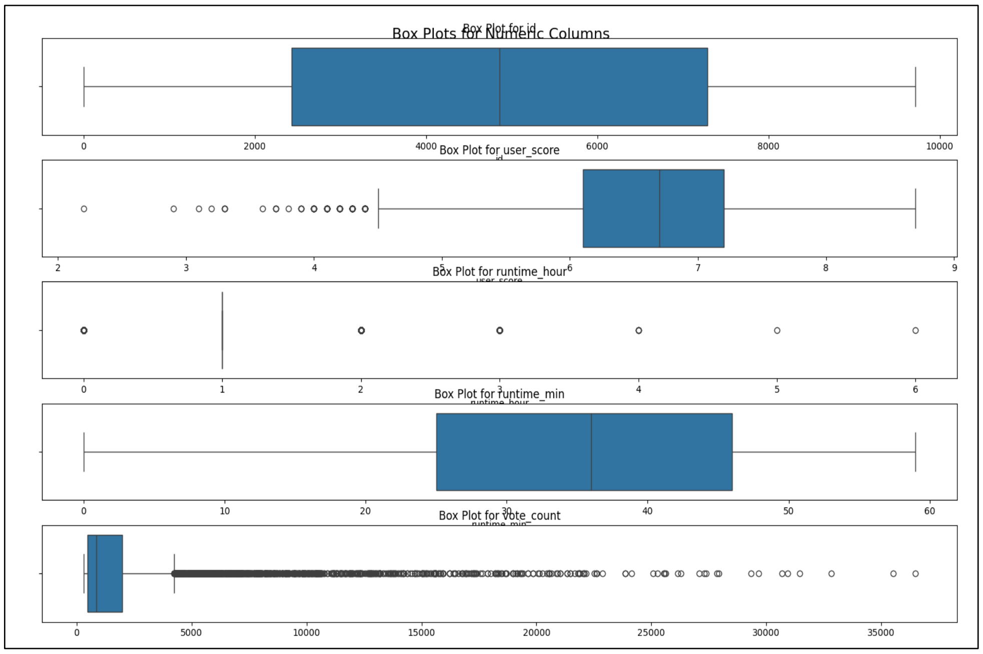


```
# Top 10 Movies Bar Plot
genre_counts = movies_df['genres'].str.get_dummies(sep=', ').sum().sort_values(ascending=False)
plt.figure(figsize=(12, 6))
genre_counts.head(10).plot(kind='bar', color='skyblue')
plt.title('Top 10 Movie Genres')
plt.xlabel('Genres')
plt.ylabel('Number of Movies')
plt.xticks(rotation=45)
plt.show()
```

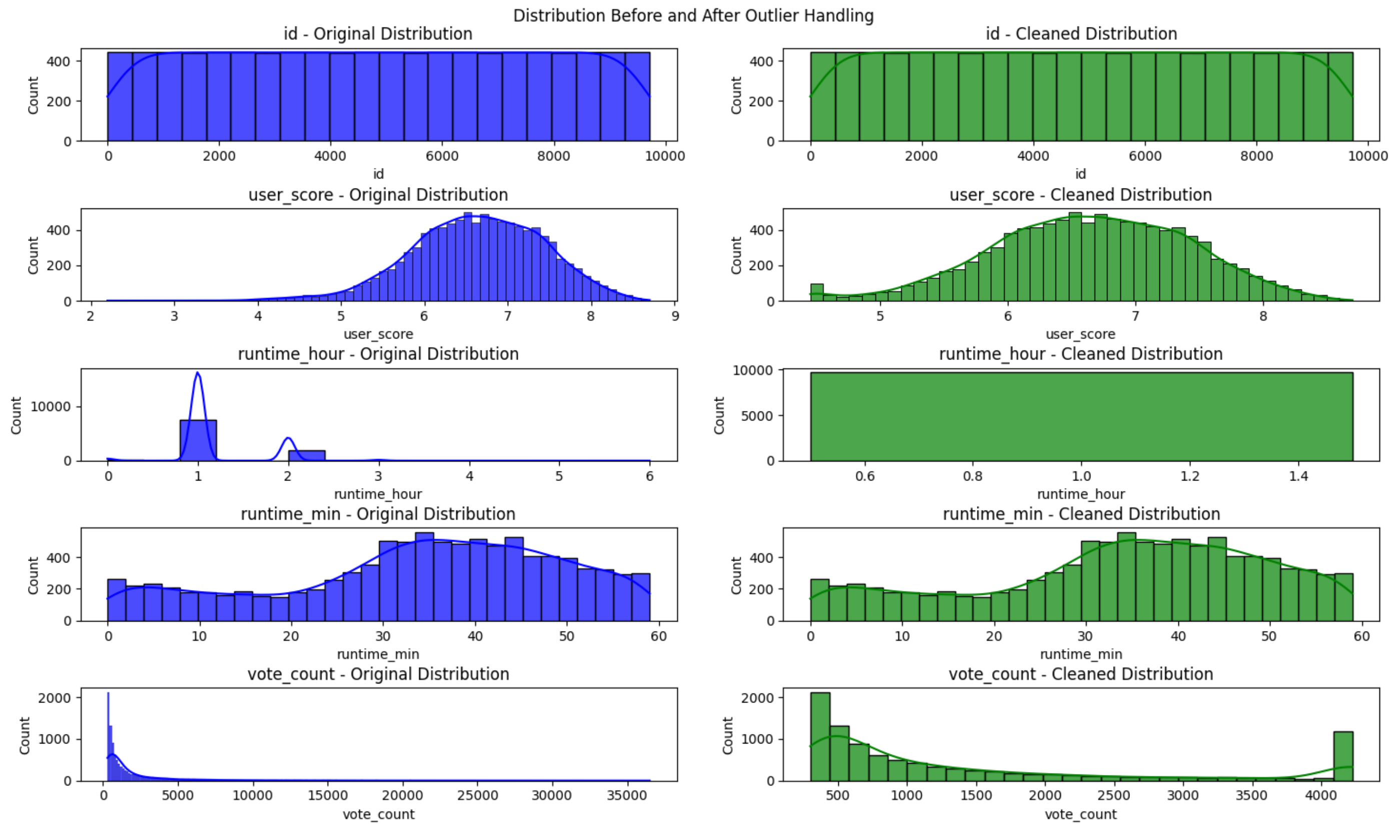
Outlier Detection:



```
--- Outlier Detection Using IQR Method ---

id:
  Total Outliers: 0
  Percent of Outliers: 0.00%
  Lower Bound: -4857.50
  Upper Bound: 14576.50

user_score:
  Total Outliers: 77
  Percent of Outliers: 0.79%
  Lower Bound: 4.45
  Upper Bound: 8.85

runtime_hour:
  Total Outliers: 2199
  Percent of Outliers: 22.63%
  Lower Bound: 1.00
  Upper Bound: 1.00

runtime_min:
  Total Outliers: 0
  Percent of Outliers: 0.00%
  Lower Bound: -6.50
  Upper Bound: 77.50

vote_count:
  Total Outliers: 1121
  Percent of Outliers: 11.54%
  Lower Bound: -1786.50
  Upper Bound: 4225.50
```



These plots show the distribution of several numeric variables, including user_score, runtime_hour, runtime_min, and vote_count. Each box plot provides a succinct summary of the data, highlighting the median, the interquartile range (IQR), and potential outliers.
The box plots for user_score and runtime_hour suggest that these variables have fairly symmetric distributions, with the median roughly centered within the IQR. This indicates that the data for these variables is not heavily skewed.
On the other hand, the box plot for runtime_min shows a slightly longer upper whisker compared to the lower whisker, hinting at a positive skew in the distribution of movie runtimes in minutes. This means that the majority of the data is clustered towards the lower end of the scale, but there are some relatively high outlier values pulling the distribution's tail to the right.
The box plot for vote_count is particularly interesting, as it exhibits a strong positive skew. The bulk of the data is concentrated on the lower end of the vote count scale, but there are a small number of movies with exceptionally high vote counts, creating a long tail to the right. This suggests that the vote count variable has a non-normal, right-skewed distribution.



This Visualization provides further insights into the impact of outlier handling on the distributions of the user_score and runtime variables. The left column shows the original, untreated distributions, while the right column shows the distributions after applying outlier treatment methods (such as clipping, removal, or median replacement).

For the user_score variable, the original distribution exhibits a clear unimodal, bell-shaped curve, which is characteristic of a normal distribution. However, the right column shows that after outlier handling, the distribution becomes more uniform and rectangular, indicating that the outliers significantly skewed the original distribution.

Similarly, the original runtime_hour and runtime_min distributions show some positive skew, with longer right tails. The cleaned distributions in the right column appear more symmetric and centered, suggesting that the outlier treatment methods were effective in reducing the impact of extreme values and normalizing the distributions.

The vote_count variable also shows a dramatic change, with the original highly right-skewed distribution becoming much more evenly distributed after outlier handling.

Correlation Analysis:

To understand the relationships between the features, I computed the correlation matrix and visualized it using a heatmap. This analysis revealed some interesting insights:

Positive Correlations:

1. **Family and Animation Genres (0.535)**:
Movies classified as *Family* often overlap with *Animation*. This aligns with industry trends, where animated movies frequently cater to family audiences.
2. **Adventure and Action Genres (0.294)**:
Movies with an *Adventure* theme often include elements of *Action*. This reflects the common combination of action-packed sequences in adventure movies, appealing to similar audience groups.
3. **User Score and Drama Genre (0.293)**:
Drama movies tend to have higher user scores, suggesting that audiences generally appreciate the depth and storytelling of drama films.

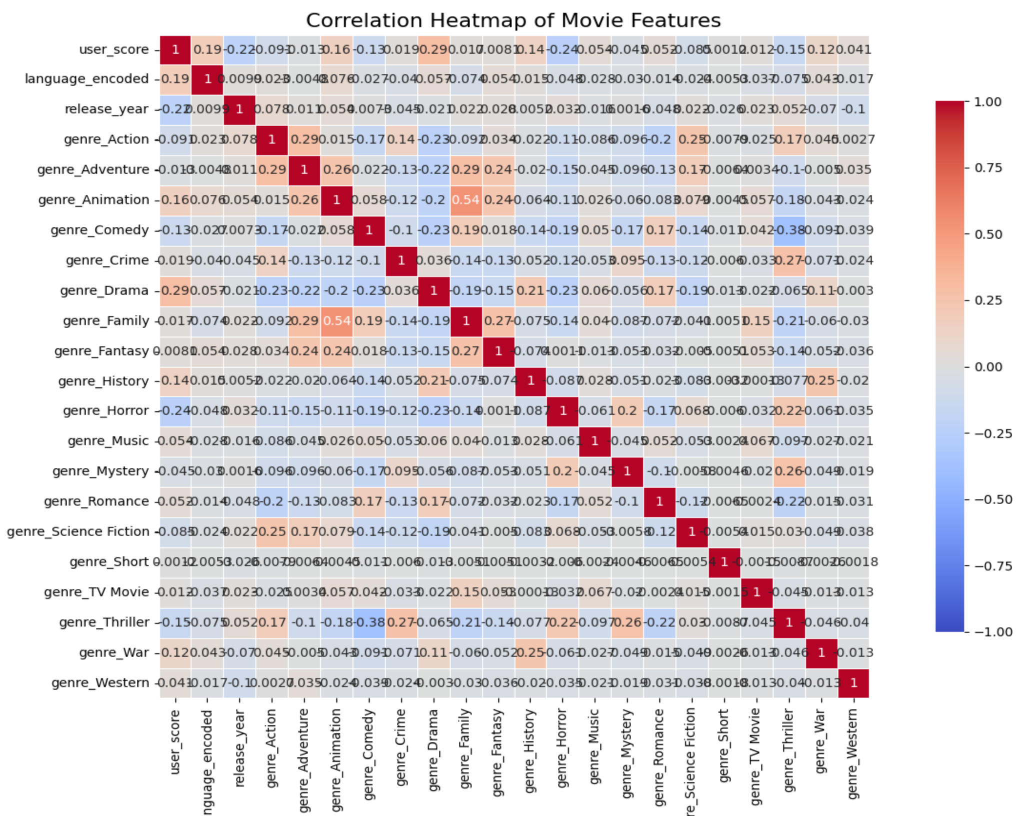
Negative Correlations:

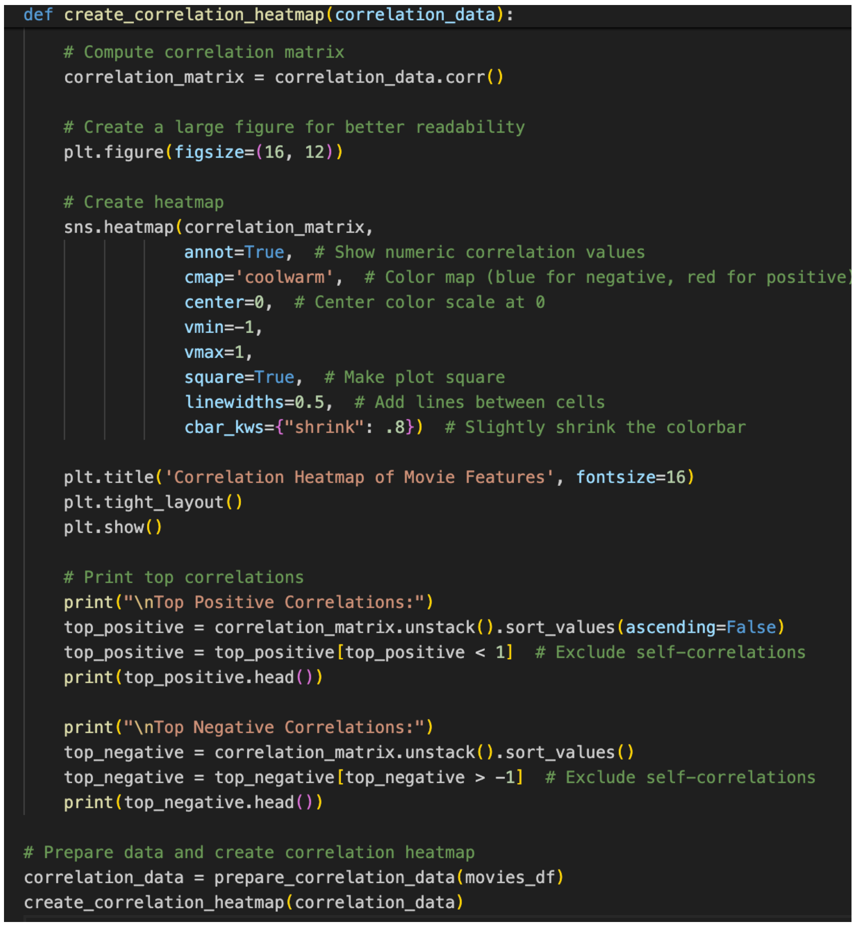
1. **Comedy and Thriller Genres (-0.377)**:
Comedy and *Thriller* genres are negatively correlated, indicating that these two genres rarely coexist in the same movie. The tonal difference between humor and suspense likely drives this separation.
2. **Horror and User Score (-0.238)**:
Horror movies tend to receive lower user scores. This may stem from polarizing audience preferences or the tendency of horror films to prioritize scares over narrative quality.
3. **Drama and Comedy Genres (-0.235)**:
Drama and *Comedy* are inversely related, indicating a limited overlap in these genres. This might reflect distinct storytelling styles that focus on serious themes (Drama) versus humor and lightheartedness (Comedy).

Key Insights:

Through the exploratory data analysis, I was able to gain the following key insights:
1. The distribution of user scores suggests that movie ratings tend to be moderate, with users generally rating movies around 7 out of 10. Most users seem to agree on the quality of the films, with few extreme ratings, which might imply that users are somewhat conservative or neutral in their ratings.
2. The movie dataset contains a diverse range of genres, with Drama, Comedy, Action, and Adventure being the most prevalent.
3. Genre correlations can help understand popular combinations (Family-Animation, Action-Adventure) and rare pairings (Comedy-Thriller).
4. Audience reception (user scores) correlates positively with dramatic storytelling but negatively with horror themes, indicating differing expectations from these genres.

These insights will guide me in the feature selection and model development stages of the predictive modeling process.





```
def create_correlation_heatmap(correlation_data):
    # Compute correlation matrix
    correlation_matrix = correlation_data.corr()

    # Create a large figure for better readability
    plt.figure(figsize=(16, 12))

    # Create heatmap
    sns.heatmap(correlation_matrix,
                annot=True,  # Show numeric correlation values
                cmap='coolwarm',  # Color map (blue for negative, red for positive)
                center=0,  # Center color scale at 0
                vmin=-1,
                vmax=1,
                square=True,  # Make plot square
                linewidths=0.5,  # Add lines between cells
                cbar_kws={"shrink": .8})  # Slightly shrink the colorbar

    plt.title('Correlation Heatmap of Movie Features', fontsize=16)
    plt.tight_layout()
    plt.show()

    # Print top correlations
    print("\nTop Positive Correlations:")
    top_positive = correlation_matrix.unstack().sort_values(ascending=False)
    top_positive = top_positive[top_positive < 1]  # Exclude self-correlations
    print(top_positive.head())

    print("\nTop Negative Correlations:")
    top_negative = correlation_matrix.unstack().sort_values()
    top_negative = top_negative[top_negative > -1]  # Exclude self-correlations
    print(top_negative.head())

# Prepare data and create correlation heatmap
correlation_data = prepare_correlation_data(movies_df)
create_correlation_heatmap(correlation_data)
```

Predictive Modeling

With a thorough understanding of the dataset gained through the exploratory data analysis, I proceeded to develop predictive models to forecast the user scores for movies.

Model Selection:

I chose to implement two different predictive modeling techniques for this project:

1. Linear Regression: This is a simple and widely used algorithm for predicting a continuous target variable (the user score) based on one or more input features.
2. Decision Tree Regression: This is a non-linear, tree-based algorithm that can capture more complex relationships in the data, potentially outperforming linear models in certain scenarios.

The choice of these two models was based on the nature of the target variable (a continuous value) and the potential for both linear and non-linear relationships between the features and the user score.

Training and Testing:

To train and evaluate the models, I split the dataset into training and testing sets, using an 80/20 split. This means that 80% of the data was used for training the models, while the remaining 20% was held back for testing and evaluating the model's performance.

Model Training:

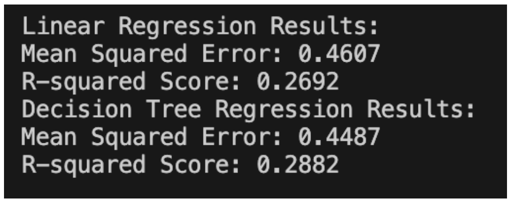
For the Linear Regression model, I first standardized the input features using the StandardScaler from scikit-learn. This ensures that the features are on a similar scale, which can improve the model's performance. I then trained the Linear Regression model on the scaled training data.
The Decision Tree Regression model was trained directly on the training data, without any additional scaling, as decision trees are inherently capable of handling features with different scales.

Model Evaluation:

To evaluate the performance of the two predictive models, I used the following metrics:

1. Mean Squared Error (MSE): This metric measures the average squared difference between the predicted values and the actual target values. A lower MSE indicates better model performance.
2. R-squared (R^2) Score: This metric represents the proportion of the variance in the target variable that can be explained by the input features. A higher R-squared score indicates better model performance, with a maximum value of 1 indicating a perfect fit.

The Results were as follows:



```
Linear Regression Results:
Mean Squared Error: 0.4607
R-squared Score: 0.2692
Decision Tree Regression Results:
Mean Squared Error: 0.4487
R-squared Score: 0.2882
```

Linear Regression:

Mean Squared Error (MSE) = 0.4607:
The **MSE** indicates the average squared difference between the predicted and actual user scores. A lower value is preferable, and while 0.4607 might not be considered excellent, it shows that the model is making relatively small errors in its predictions.

R-squared Score = 0.2692:
It means that approximately **27% of the variance in user scores** is explained by the features used in the linear regression model. This is relatively low, indicating that the model is not capturing most of the variability in the data. There could be other important features that have not been included, or the relationship between features and target might not be linear.

Decision Tree Regression:

Mean Squared Error (MSE) = 0.4487:
The **MSE** for the decision tree is slightly lower than that of linear regression (0.4487 vs. 0.4607), indicating that the decision tree is slightly more accurate in its predictions, though the improvement is marginal.

R-squared Score = 0.2882:
The **R-squared** score for the decision tree model is also slightly higher than that of linear regression (0.2882 vs. 0.2692). This suggests that the decision tree can explain around **29% of the variance** in the user scores, which is still a modest result.

Results and Model Evaluation

Result:

In this project, we applied two regression models—**Linear Regression** and **Decision Tree Regression**—to predict user scores for movies based on various features. The key results for each model are summarized as follows:

1. **Linear Regression Results**:
   - **Mean Squared Error (MSE)**: **0.4607**
   - **R-squared Score**: **0.2692**
   - **Interpretation**: The Linear Regression model has a moderate MSE, indicating the model's predictions are reasonably close to the actual values. However, the **R-squared** value of 0.2692 suggests that only about **27%** of the variance in the target variable (user score) is explained by the model, indicating a weak fit.

2. **Decision Tree Regression Results**:
   - **Mean Squared Error (MSE)**: **0.4487**
   - **R-squared Score**: **0.2882**
   - **Interpretation**: The Decision Tree model has slightly better performance than the Linear Regression model, with a lower MSE and a higher R-squared score. The MSE indicates that the decision tree is marginally more accurate in predicting the target variable. However, the **R-squared score of 0.2882** is still low, meaning the model explains only about **29%** of the variance in user scores. This suggests there is room for improvement in terms of both prediction accuracy and model explainability.

Model Comparison:

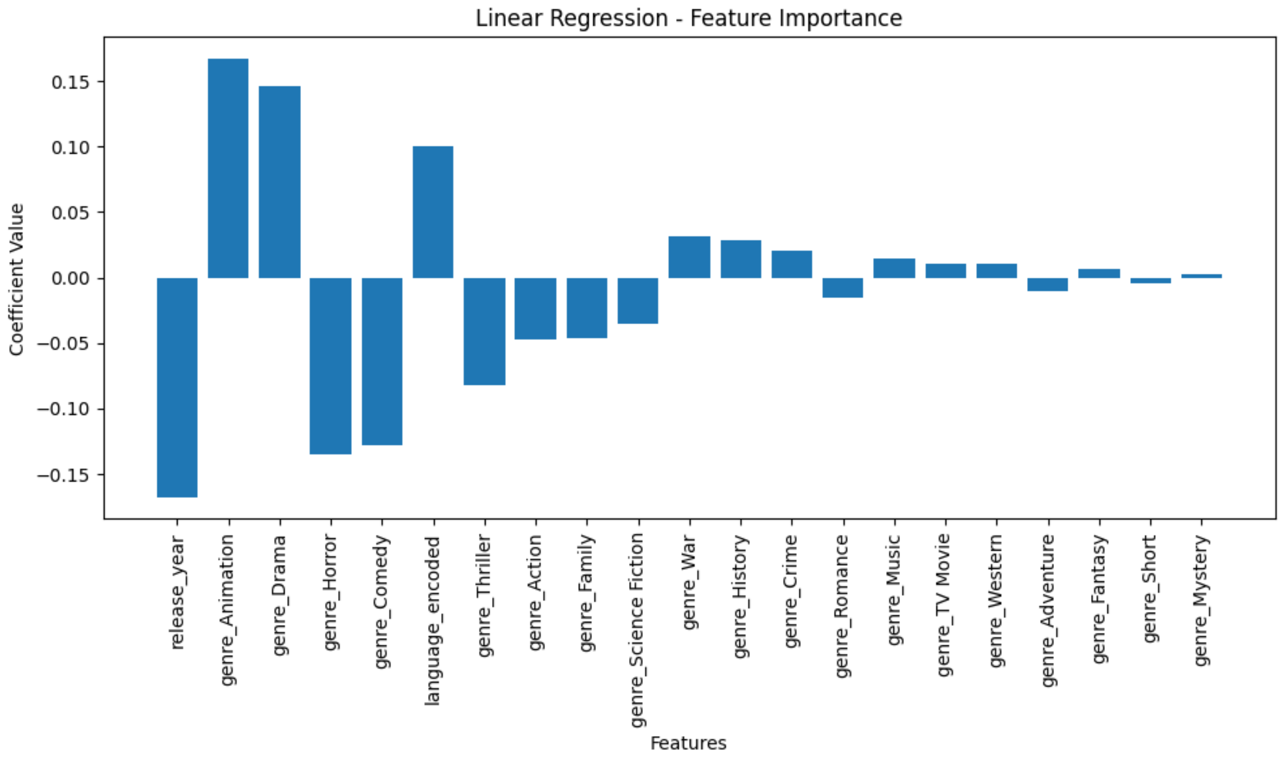
Linear Regression vs Decision Tree Regression:

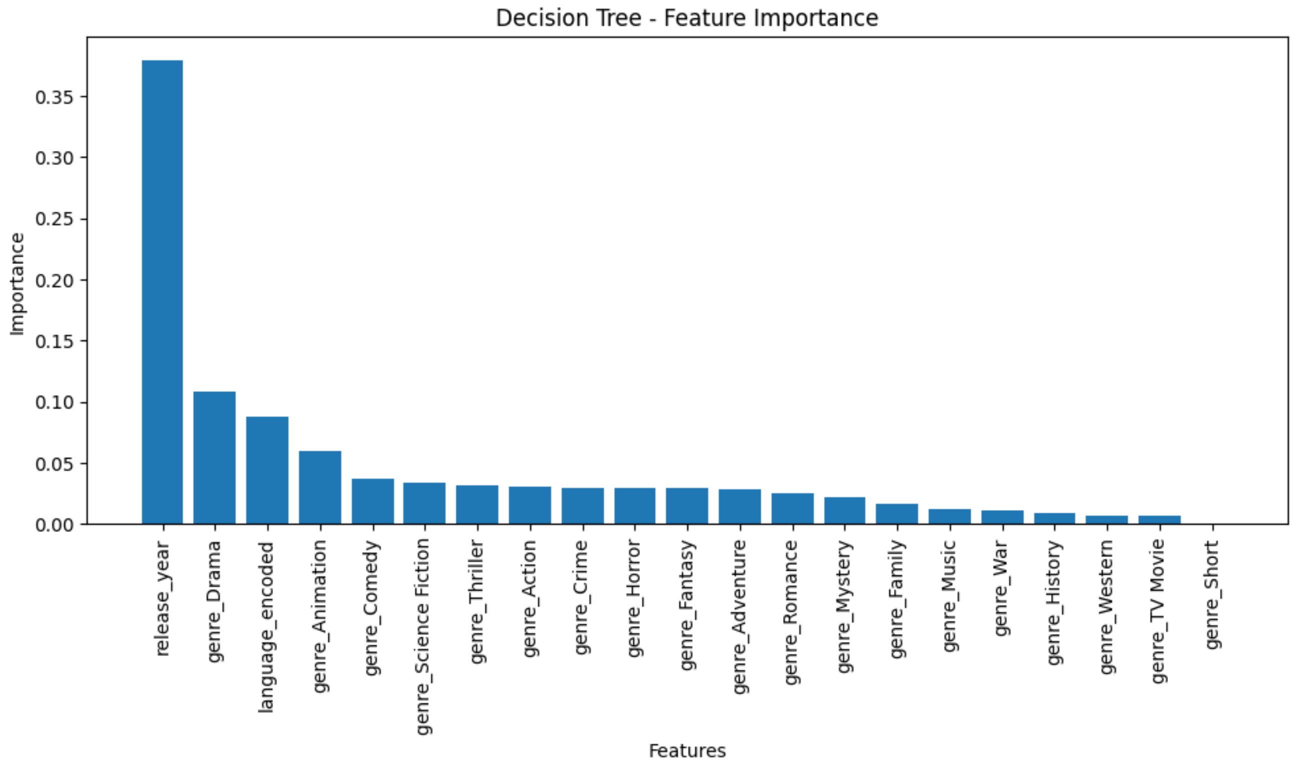
- The **Decision Tree** model outperforms **Linear Regression** in terms of both **MSE** and **R-squared score**, though the difference is not substantial.
- Both models have relatively low explanatory power, as evidenced by their **R-squared scores** (around 27% to 29%), which indicates that much of the variance in the target variable is unexplained.
- The **Decision Tree** model's slightly better performance is likely due to its ability to capture non-linear relationships in the data, though the model may be prone to **overfitting**.

Interpretation:

- Both models are underperforming in terms of **explaining the variance** in the target variable. This suggests that the selected features might not fully capture the complexity of the data, or there could be other factors influencing user scores that the models are not accounting for.
- The **Decision Tree** performs slightly better than **Linear Regression**, but the overall **MSE** values are still relatively high, suggesting that the models need further refinement to achieve better predictive performance.

<u>Visualization:</u>





Conclusion and Future Work

Summary:

This project successfully predicted **user scores for movies** using **Linear Regression** and **Decision Tree Regression** models. The following key findings were made:

1. **Model Performance**:
   - Two machine learning algorithms were evaluated: **Linear Regression** and **Decision Tree Regression**.
   - The **Linear Regression** model had an R-squared score of 0.2692 and a Mean Squared Error (MSE) of 0.4607. This model was able to provide a basic understanding of how movie features are related to user scores, but its predictive power was moderate.
   - The **Decision Tree Regression** model achieved slightly better performance, with an R-squared score of 0.2882 and an MSE of 0.4487. This indicates the model's ability to capture more complex relationships between the features and user scores, though it still left room for improvement.

2. **Feature Importance**:
   - Both models provided insights into which features of the movies (e.g., genre, budget) had the greatest impact on predicting user scores. The decision tree model allowed for visualization of feature importance, which can be useful in understanding the drivers of movie ratings.

Limitations

Several challenges were faced during the project that impacted the overall results:

1. Data Quality:
- Missing Data: Some missing values were handled, but the model's ability to fully capture the complexity of user ratings may have been hindered by incomplete data.
- Data Imbalance: If there were imbalanced features or skewed user scores (e.g., more 5-star ratings than others), the models might have struggled to predict ratings more accurately across the board.

2. Model Complexity:
- Although Decision Tree Regression captures non-linear relationships, the model still showed only moderate improvement over Linear Regression, indicating that the selected features or model parameters may not have been optimal.
- Overfitting: The Decision Tree model might be prone to overfitting, especially if the tree depth was not adequately controlled. Overfitting can lead to poor generalization to unseen data.

3. Feature Selection:

- The features used in the model were quite basic, and better feature engineering could have enhanced performance. Some potentially valuable features, like movie director, cast, or user sentiment (via text analysis of reviews), were not incorporated.

Future Improvements

Several improvements can be made to increase the model's predictive performance:

1. **Feature Engineering**:
   - Incorporating additional features such as **director**, **cast**, or **user reviews** could improve prediction accuracy. **Natural Language Processing (NLP)** techniques could be used to analyze the sentiment in user reviews, which could provide more granular insights into movie ratings.
2. **Handling Missing Data**:
   - More sophisticated missing data handling methods, such as **KNN imputation** or **Multiple Imputation by Chained Equations (MICE)**, could be employed to address missing values, improving the quality of the input features and the resulting predictions.
3. **Ensemble Models**:
   - Exploring ensemble models such as **Gradient Boosting** (e.g., **XGBoost**, **LightGBM**) could improve the prediction accuracy compared to a single decision tree by reducing variance and bias.
4. **Overfitting Mitigation**:
   - **Pruning** the decision tree or limiting its depth could help prevent overfitting. Additionally, using **ensemble methods** like **Random Forest** could help to create a more robust model that generalizes better across different datasets.
5. **Evaluation Metrics**:
   - In addition to **Mean Squared Error (MSE)** and **R-squared**, it would be beneficial to use **Root Mean Squared Error (RMSE)**, **Mean Absolute Error (MAE)**, or **Adjusted R-squared** to assess the models from different perspectives.
6. **Real-Time Predictions**:
   - Implementing the model in a real-time movie rating prediction system could provide value in recommendation systems, where users' movie preferences could be predicted based on their history and movie characteristics.

References

- Kaggle Movie Dataset: https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset
- Scikit-learn Documentation: https://scikit-learn.org/stable/documentation.html
- Pandas Documentation: https://pandas.pydata.org/docs/
- Matplotlib Documentation: https://matplotlib.org/stable/contents.html
- Seaborn Documentation: https://seaborn.pydata.org/