Faculty of Engineering, Natural and Medical Sciences / Department of Information Technologies

IT 323 - DATA MINING PROJECT

IMDB Movies Dataset

Author: Arnela Sokolić

29/05/2025

Contents

1. Introduction

This project focuses on analyzing the IMDB Top 1000 Movies Dataset to uncover patterns and insights about highly rated movies.

Movies and TV shows play a crucial role in the entertainment industry, and understanding audience preferences can help predict trends, identify successful patterns, and improve recommendations in streaming services.

2. Dataset overview

Dataset Name: IMDB Top 1000 Movies

Source: Kaggle (Original Dataset)

3. Description

This dataset contains the top 1000 movies and TV shows based on IMDB ratings. It includes detailed information about each movie, such as its title, release year, genre, director, actors, rating, and earnings.

Poster_Link - URL of the movie poster used on IMDB
Series_Title -Name of the movie or TV show
Released_Year -Year of release
Certificate -Age certification (e.g., PG-13, R)
Runtime -Total duration of the movie
Genre -Movie genres (e.g., Action, Drama, Comedy)
IMDB_Rating -Rating score on IMDB
Overview -Short summary or description of the movie
Meta_score - Metacritic score
Director -Name of the director
Star1, Star2, Star3, Star4 -Names of the main actors
No_of_votes -Total number of votes received
Gross -Total earnings of the movie

4. Data collection

The dataset was obtained from Kaggle and downloaded in CSV format. This dataset serves as the foundation for analyzing trends in movie ratings, actor performance, and financial success.

5. Performing data cleaning and preprocessing

```
import pandas as pd

# Load the dataset
file_path = r"C:\Users\Korisnik\Downloads\archive\imdb_top_1000.csv"

# Read the CSV file
df = pd.read_csv(file_path)

# Display the first 5 rows
print(df.head())
```

I first imported the pandas library. Then, I specified the file path where my IMDb dataset is stored. After that, I used pd.read_csv(file_path) to load the dataset into a DataFrame called df, so I can analyze it. Finally, I used df.head() to print the first five rows of the data, giving me a quick look at what's inside.

```
                                         Poster_Link  \
0  https://m.media-amazon.com/images/M/MV5BMDFkYT...
1  https://m.media-amazon.com/images/M/MV5BM2MyNj...
2  https://m.media-amazon.com/images/M/MV5BMTMxNT...
3  https://m.media-amazon.com/images/M/MV5BMWMwMG...
4  https://m.media-amazon.com/images/M/MV5BMWU4N2...

                Series_Title Released_Year Certificate  Runtime  \
0  The Shawshank Redemption          1994           A  142 min
1             The Godfather          1972           A  175 min
2           The Dark Knight          2008          UA  152 min
3    The Godfather: Part II          1974           A  202 min
4              12 Angry Men          1957           U   96 min

                   Genre  IMDB_Rating  \
0                  Drama          9.3
1           Crime, Drama          9.2
2  Action, Crime, Drama          9.0
3           Crime, Drama          9.0
4           Crime, Drama          9.0

                                        Overview  Meta_score  \
0  Two imprisoned men bond over a number of years...        80.0
1  An organized crime dynasty's aging patriarch t...       100.0
2  When the menace known as the Joker wreaks havo...        84.0
3  The early life and career of Vito Corleone in ...        90.0
4  A jury holdout attempts to prevent a miscarria...        96.0

               Director           Star1           Star2          Star3  \
0       Frank Darabont     Tim Robbins  Morgan Freeman     Bob Gunton
1  Francis Ford Coppola   Marlon Brando       Al Pacino     James Caan
2     Christopher Nolan  Christian Bale    Heath Ledger  Aaron Eckhart
3  Francis Ford Coppola       Al Pacino  Robert De Niro  Robert Duvall
4          Sidney Lumet     Henry Fonda     Lee J. Cobb  Martin Balsam

            Star4  No_of_Votes        Gross
0  William Sadler      2343110   28,341,469
1    Diane Keaton      1620367  134,966,411
2   Michael Caine      2303232  534,858,444
3    Diane Keaton      1129952   57,300,000
4    John Fiedler       689845    4,360,000
```

The output shows the first five rows of my IMDb dataset. Each row represents a movie with various details, such as the title, release year, certificate rating, runtime, genre, and IMDb rating. It also includes a brief overview of the plot, Meta score, director, and main cast members (Star1 to Star4). Additionally, the dataset provides the number of votes and box office gross earnings. The "Poster_Link" column contains URLs to the movie posters.

```
df.head (10)
```

The code df.head(10) retrieves and displays the first **10 rows** of the dataset stored in the DataFrame df.

	Poster_Link	Series_Title	Released_Year	Certificate	Runtime	Genre	IMDB_Rating	Overview	Meta_score	Director	Sta
0	https://m.media-amazon.com/images/M/MV5BMDFkYT...	The Shawshank Redemption	1994	A	142 min	Drama	9.3	Two imprisoned men bond over a number of years...	80.0	Frank Darabont	T Robbi
1	https://m.media-amazon.com/images/M/MV5BM2MyNj...	The Godfather	1972	A	175 min	Crime, Drama	9.2	An organized crime dynasty's aging patriarch t...	100.0	Francis Ford Coppola	Marl Bran
2	https://m.media-amazon.com/images/M/MV5BMTMxNT...	The Dark Knight	2008	UA	152 min	Action, Crime, Drama	9.0	When the menace known as the Joker wreaks havo...	84.0	Christopher Nolan	Christi Ba
3	https://m.media-amazon.com/images/M/MV5BMWMwMG...	The Godfather: Part II	1974	A	202 min	Crime, Drama	9.0	The early life and career of Vito Corleone in ...	90.0	Francis Ford Coppola	Al Paci
4	https://m.media-amazon.com/images/M/MV5BMWU4N2...	12 Angry Men	1957	U	96 min	Crime, Drama	9.0	A jury holdout attempts to prevent a miscarria...	96.0	Sidney Lumet	Her Fon
5	https://m.media-amazon.com/images/M/MV5BNzA5ZD...	The Lord of the Rings: The Return of the King	2003	U	201 min	Action, Adventure, Drama	8.9	Gandalf and Aragorn lead the World of Men agai...	94.0	Peter Jackson	Elij Wo
6	https://m.media-amazon.com/images/M/MV5BNGNhMD...	Pulp Fiction	1994	A	154 min	Crime, Drama	8.9	The lives of two mob hitmen, a boxer, a gangst...	94.0	Quentin Tarantino	Jo Travo

ased_Year	Certificate	Runtime	Genre	IMDB_Rating	Overview	Meta_score	Director	Star1	Star2	Star3	Star4	No_of_Votes	Gross
1994	A	142 min	Drama	9.3	Two imprisoned men bond over a number of years...	80.0	Frank Darabont	Tim Robbins	Morgan Freeman	Bob Gunton	William Sadler	2343110	28,341,469
1972	A	175 min	Crime, Drama	9.2	An organized crime dynasty's aging patriarch t...	100.0	Francis Ford Coppola	Marlon Brando	Al Pacino	James Caan	Diane Keaton	1620367	134,966,411
2008	UA	152 min	Action, Crime, Drama	9.0	When the menace known as the Joker wreaks havo...	84.0	Christopher Nolan	Christian Bale	Heath Ledger	Aaron Eckhart	Michael Caine	2303232	534,858,444
1974	A	202 min	Crime, Drama	9.0	The early life and career of Vito Corleone in ...	90.0	Francis Ford Coppola	Al Pacino	Robert De Niro	Robert Duvall	Diane Keaton	1129952	57,300,000
1957	U	96 min	Crime, Drama	9.0	A jury holdout attempts to prevent a miscarria...	96.0	Sidney Lumet	Henry Fonda	Lee J. Cobb	Martin Balsam	John Fiedler	689845	4,360,000
2003	U	201 min	Action, Adventure, Drama	8.9	Gandalf and Aragorn lead the World of Men agai...	94.0	Peter Jackson	Elijah Wood	Viggo Mortensen	Ian McKellen	Orlando Bloom	1642758	377,845,905
1994	A	154 min	Crime, Drama	8.9	The lives of two mob hitmen, a boxer, a gangst...	94.0	Quentin Tarantino	John Travolta	Uma Thurman	Samuel L. Jackson	Bruce Willis	1826188	107,928,762
					In								

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 16 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Poster_Link    1000 non-null   object 
 1   Series_Title   1000 non-null   object 
 2   Released_Year  1000 non-null   object 
 3   Certificate    899 non-null    object 
 4   Runtime        1000 non-null   object 
 5   Genre          1000 non-null   object 
 6   IMDB_Rating    1000 non-null   float64
 7   Overview       1000 non-null   object 
 8   Meta_score     843 non-null    float64
 9   Director       1000 non-null   object 
 10  Star1          1000 non-null   object 
 11  Star2          1000 non-null   object 
 12  Star3          1000 non-null   object 
 13  Star4          1000 non-null   object 
 14  No_of_Votes    1000 non-null   int64  
 15  Gross          831 non-null    object 
dtypes: float64(2), int64(1), object(13)
memory usage: 125.1+ KB
```

The code df.info() provides a summary of the dataset. It displays the number of rows and columns, the column names, their data types, and the count of non-null values in each column. This helps me understand the dataset's structure, check for missing values, and identify if any data types need conversion for analysis.

```
df.describe()
```

	IMDB_Rating	Meta_score	No_of_Votes
count	1000.000000	843.000000	1.000000e+03
mean	7.949300	77.971530	2.736929e+05
std	0.275491	12.376099	3.273727e+05
min	7.600000	28.000000	2.508800e+04
25%	7.700000	70.000000	5.552625e+04
50%	7.900000	79.000000	1.385485e+05
75%	8.100000	87.000000	3.741612e+05
max	9.300000	100.000000	2.343110e+06

When I run df.describe(), it gives me a summary of all the numerical columns in my dataset. It shows me how many values exist in each column, along with key statistics like the average (mean), minimum, and maximum values.

It also tells me how spread out the data is using the standard deviation and shows quartiles like the median (50%) and the 25th and 75th percentiles. For example, the average IMDb rating is 7.95, while the highest-rated movie has a 9.3.

The Meta_score ranges from 28 to 100, and the number of votes varies widely, with the most-voted movie having over 2.3 million votes. This helps me understand the overall distribution of the data and spot any potential outliers.

```
num_records = len(df)

print('Total numbers of records: {}'.format(num_records))

print(df.shape[0])
print(df.shape[1])
```

```
Total numbers of records: 1000
1000
16
```

When I run this code, it calculates and prints the total number of records (rows) in my dataset. The line num_records = len(df) gets the total number of rows, which is 1000, and prints it using format(). Then, df.shape[0] confirms the number of rows, while df.shape[1] tells me the dataset has 16 columns. This helps me quickly check the dataset's size.

```
# Check for missing values in the dataset
print(df.isnull().sum())

# Check for duplicate rows in the dataset
print(df.duplicated().any())
```

```
Poster_Link        0
Series_Title       0
Released_Year      0
Certificate      101
Runtime            0
Genre              0
IMDB_Rating        0
Overview           0
Meta_score       157
Director           0
Star1              0
Star2              0
Star3              0
Star4              0
No_of_Votes        0
Gross            169
dtype: int64
False
```

When I run this code, it checks for missing values and duplicate rows in my dataset. The output shows that some columns have missing data, such as Certificate (101 missing values), Meta Score (157 missing values), and Gross earnings (169 missing values), while other columns have complete data. Then, it checks for duplicate rows and returns False, meaning there are no duplicate entries in my dataset. This helps me understand if I need to clean or fill in any missing values before analyzing the data.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv("C:\\Users\\Korisnik\\Downloads\\archive\\imdb_top_1000.csv")


# Calculate statistics
mean_value = df[numeric_column].mean()
median_value = df[numeric_column].median()
mode_value = df[numeric_column].mode()[0]
midrange = (df[numeric_column].min() + df[numeric_column].max()) / 2

# Compute quartiles
q1 = df[numeric_column].quantile(0.25)
q3 = df[numeric_column].quantile(0.75)
min_value = df[numeric_column].min()
max_value = df[numeric_column].max()

# Five-number summary
five_number_summary = {
    "Minimum": min_value,
    "Q1": q1,
    "Median": median_value,
    "Q3": q3,
    "Maximum": max_value
}

# Print statistics
print("\nStatistics:")
print(f"Mean: {mean_value}")
print(f"Median: {median_value}")
print(f"Mode: {mode_value}")
print(f"Midrange: {midrange}")
print(f"Five-Number Summary: {five_number_summary}")
```

This code analyzes IMDb movie data by calculating key statistics and visualizing the distribution of a selected numeric column.

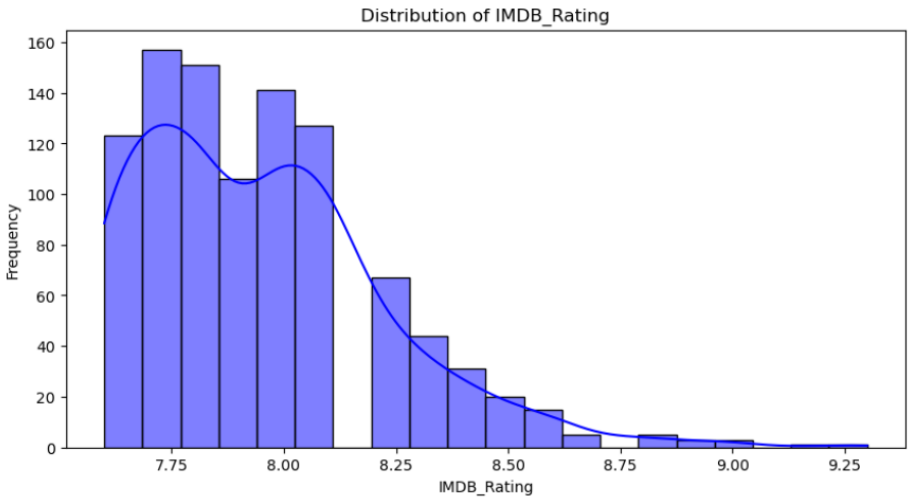
First, the dataset is loaded using pandas. The "Runtime" column, which contains values like "142 min", is cleaned by removing "min" and converting it to a float. Similarly, the "Gross" column, which has values with commas (e.g., "134,966,411"), is cleaned by removing commas and converting it into a float for numerical calculations.

Next, a numeric column is selected for analysis (default is "IMDB_Rating", but it can be changed to "Meta_score", "Runtime", or "Gross"). The code then calculates key statistics:

- **Mean** (average value of the column)
- **Median** (middle value of the sorted data)
- **Mode** (most frequently occurring value)
- **Midrange** (average of the minimum and maximum values)
- **Quartiles (Q1 and Q3)** and the **Five-Number Summary** (minimum, Q1, median, Q3, and maximum), which describe the spread of the data.

```
Statistics:
Mean: 7.949299999999999
Median: 7.9
Mode: 7.7
Midrange: 8.45
Five-Number Summary: {'Minimum': 7.6, 'Q1': 7.7, 'Median': 7.9, 'Q3': 8.1, 'Maximum': 9.3}
```

The IMDb ratings have an average (mean) of 7.95, with a median of 7.9, meaning half of the movies are rated below 7.9 and half above. The most common rating (mode) is 7.7, while the ratings range from a minimum of 7.6 to a maximum of 9.3, with most movies falling between 7.7 (Q1) and 8.1 (Q3).



Picture 1. Distribution of IMDB Ratings

This graph shows the distribution of IMDb ratings for the top 1000 movies. Most ratings fall between 7.5 and 8.5, with a peak around 7.8 to 8.0. Higher ratings, especially above 9.0, are rare. The curve shows a right-skewed pattern, meaning fewer movies have exceptionally high ratings. This is common since only a few films are considered true classics.

```
print("Duplicates before:", df.duplicated().sum())

df.drop_duplicates(inplace=True)

print("Duplicates after:", df.duplicated().sum())
```

```
Duplicates before: 0
Duplicates after: 0
```

When I run this code, it checks for duplicate rows before and after attempting to remove them. The output shows "Duplicates before: 0", meaning there were no duplicate rows in the dataset to begin with.

After running df.drop_duplicates(inplace=True), the second check also confirms "Duplicates after: 0", which means no changes were made since there were no duplicates to remove. This tells me my dataset already contains only unique movie entries.

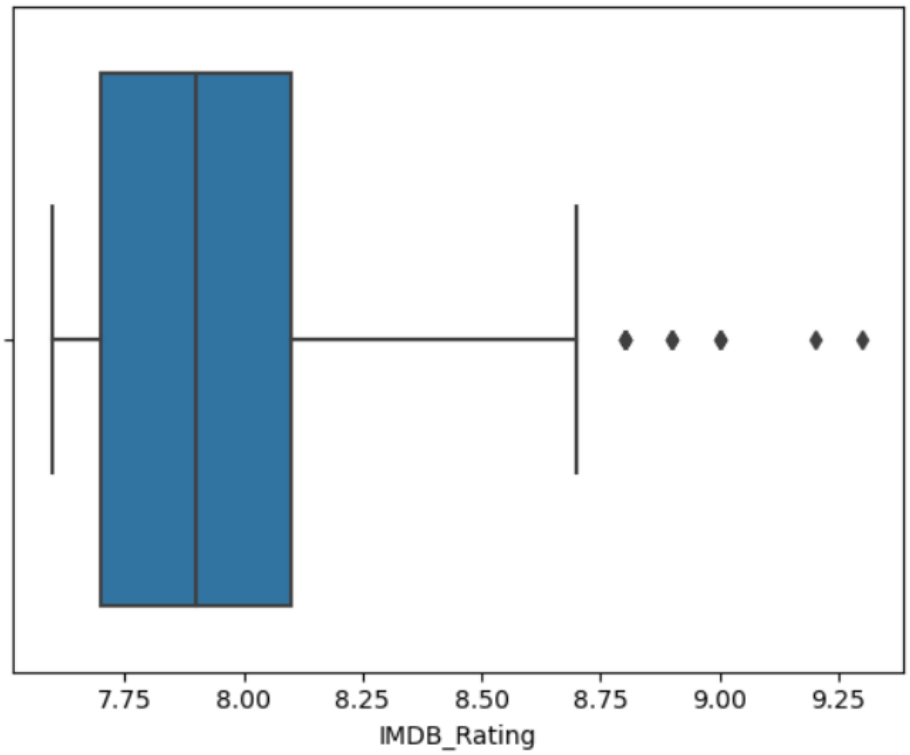
```
# Convert 'Released_Year' to numeric, coercing errors to NaN
df["Released_Year"] = pd.to_numeric(df["Released_Year"], errors='coerce')

# Convert to integer while keeping NaN values
df["Released_Year"] = df["Released_Year"].astype("Int64")

# Check new data types
print(df.dtypes)
```

```
Poster_Link       object
Series_Title      object
Released_Year      Int64
Certificate       object
Runtime           object
Genre             object
IMDB_Rating      float64
Overview          object
Meta_score       float64
Director          object
Star1             object
Star2             object
Star3             object
Star4             object
No_of_Votes        int64
Gross             object
Decade           float64
dtype: object
```

This code ensures that the "Released_Year" column is properly converted into integers while handling missing values. It first changes non-numeric values to NaN, then converts the column to an integer format that allows missing values.

Finally, it prints the data types to confirm the changes. If needed, the "Decade" column can also be adjusted to store decade values as integers instead of floats. This cleanup makes the dataset more structured and ready for analysis.

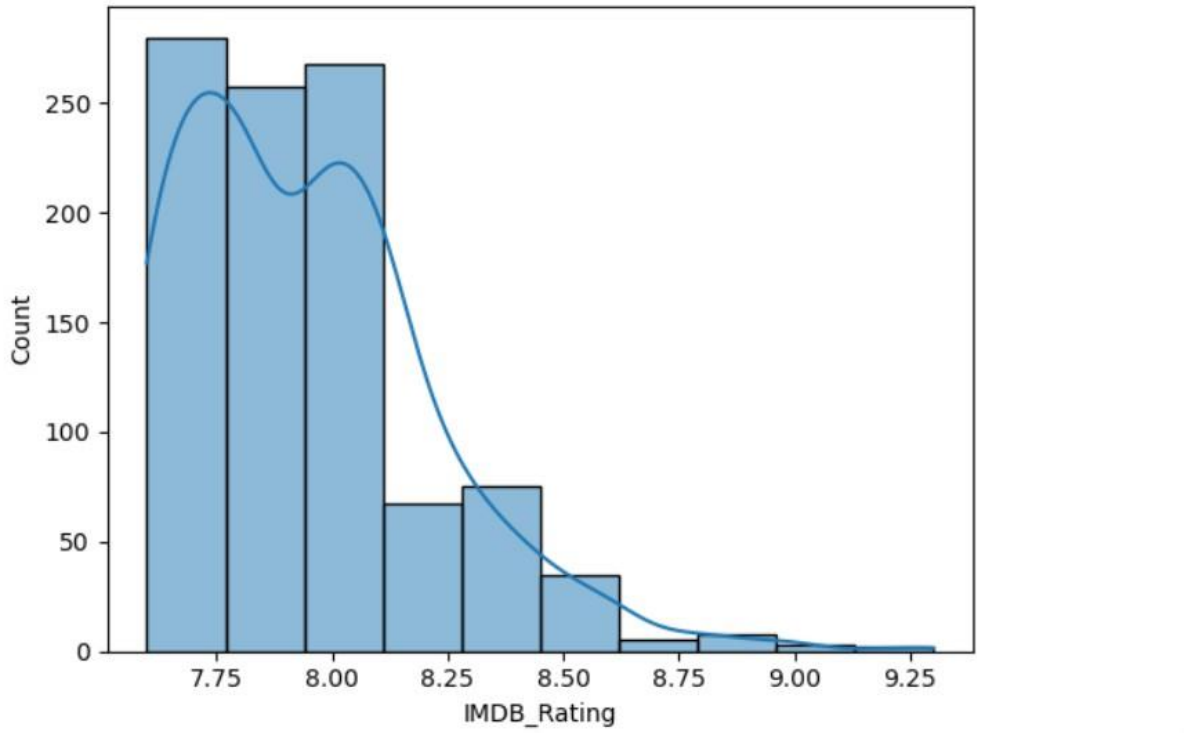
```
import matplotlib.pyplot as plt
import seaborn as sns

sns.boxplot(x=df["IMDB_Rating"])
plt.show()
```



Picture 2. Distribution of IMDB Ratings

I created a boxplot to visualize the distribution of IMDb ratings in my dataset. The box represents the middle 50% of the ratings, with the line inside showing the median, which is around 8.0. The whiskers extend to most of the data, while the dots beyond them indicate outliers—movies with exceptionally high ratings above 8.6.

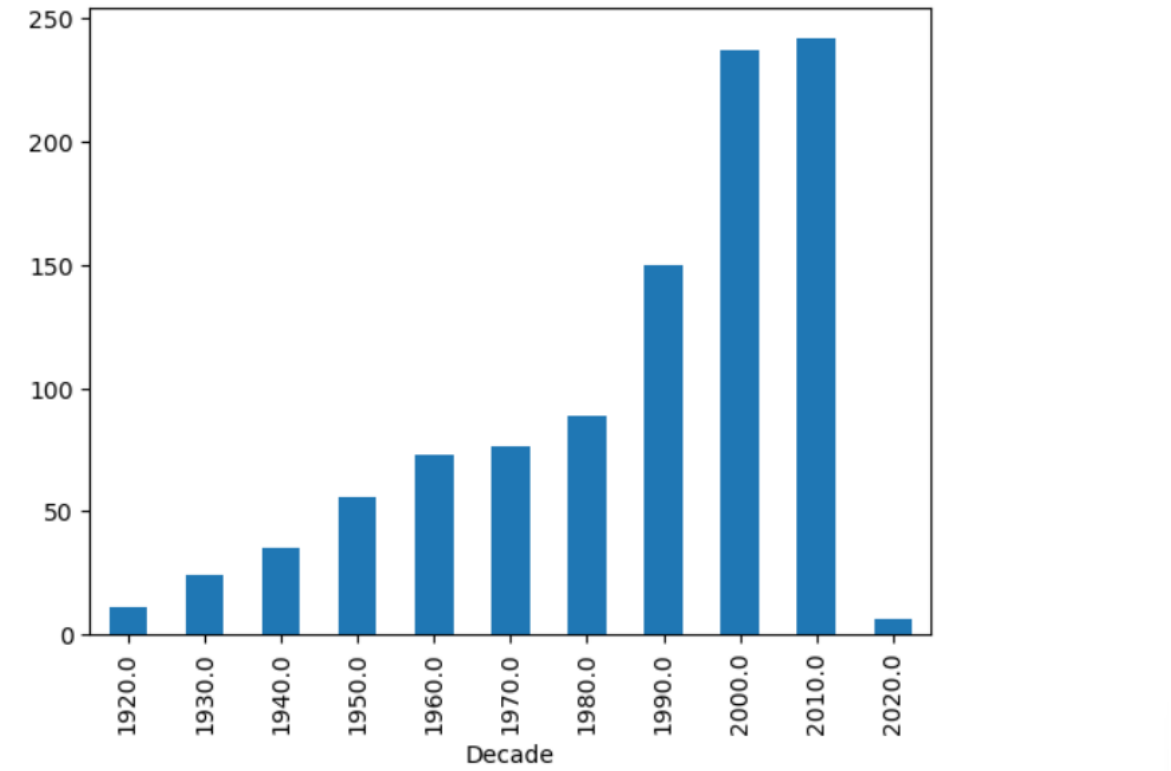
```
sns.histplot(df["IMDB_Rating"], bins=10, kde=True)
plt.show()
```



Picture 3. Distribution of IMDB Ratings

I created a histogram to visualize the distribution of IMDb ratings in my dataset. The bars show how frequently different rating ranges appear, with most movies clustered around 7.7 to 8.2. The KDE (Kernel Density Estimation) curve smooths out the distribution, making it easier to see the overall trend. The data is slightly right-skewed, meaning there are fewer movies with very high ratings above 8.5.

```
df["Decade"] = (df["Released_Year"] // 10) * 10
df["Decade"].value_counts().sort_index().plot(kind="bar")
plt.show()
```



Picture 4. Decade

I created a bar chart to show the number of movies released in each decade. The code first calculates the decade by rounding down the "Released_Year" to the nearest multiple of 10. Then, it counts how many movies belong to each decade and plots them in order.

The chart reveals an increasing trend in the number of top-rated movies over time, with the highest counts in the 2000s and 2010s. This suggests that more high-rated movies have been produced in recent decades.

```
print(df.isnull().sum())  # Check missing values in each column
```

```
Poster_Link      0
Series_Title     0
Released_Year    1
Certificate      0
Runtime          0
Genre            0
IMDB_Rating      0
Overview         0
Meta_score       0
Director         0
Star1            0
Star2            0
Star3            0
Star4            0
No_of_Votes      0
Gross            0
Decade           1
dtype: int64
```

When I run df.isnull().sum(), it checks for missing values in each column and counts how many are present.

The output shows that only two columns have missing values:

- "Released_Year" → 1 missing value
- "Decade" → 1 missing value

Since "Decade" is derived from "Released_Year", the missing value in "Released_Year" is likely causing the missing "Decade" value as well.

This means I need to fill or remove the missing "Released_Year" value so that "Decade" can also be corrected.

```
most_common_year = df["Released_Year"].mode()[0]
df["Released_Year"].fillna(most_common_year, inplace=True)
```

When I run this code, it fills the missing value in the "Released_Year" column with the most frequently occurring (mode) year in the dataset.

- df["Released_Year"].mode()[0] finds the most common year.
- df["Released_Year"].fillna(most_common_year, inplace=True) replaces the missing value with that year.

- inplace=True ensures the change is applied directly to the DataFrame without needing to reassign it.

Now, "Released_Year" no longer has missing values, and I can recalculate "Decade" to fix its missing value too.

```
df["Decade"] = (df["Released_Year"] // 10) * 10
```

This code recalculates the "Decade" column by rounding down "Released_Year" to the nearest multiple of 10. For example, 1994 becomes 1990, 2008 becomes 2000. This helps group movies by decade.

```
print(df.isnull().sum())
```

```
Poster_Link      0
Series_Title     0
Released_Year    0
Certificate      0
Runtime          0
Genre            0
IMDB_Rating      0
Overview         0
Meta_score       0
Director         0
Star1            0
Star2            0
Star3            0
Star4            0
No_of_Votes      0
Gross            0
Decade           0
dtype: int64
```

This output shows that there are no missing (null) values in any column of the dataset.

```
df["Gross"] = df["Gross"].replace(",", "", regex=True).astype(float)
```

This line of code cleans and converts the "Gross" column to a numeric format:

1. replace(",", "", regex=True) → Removes commas from numbers (e.g., "1,000,000" becomes "1000000").
2. .astype(float) → Converts the cleaned values to float type for numerical analysis.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Poster_Link    1000 non-null   object 
 1   Series_Title   1000 non-null   object 
 2   Released_Year  1000 non-null   Int64  
 3   Certificate    1000 non-null   object 
 4   Runtime        1000 non-null   object 
 5   Genre          1000 non-null   object 
 6   IMDB_Rating    1000 non-null   float64
 7   Overview       1000 non-null   object 
 8   Meta_score     1000 non-null   float64
 9   Director       1000 non-null   object 
 10  Star1          1000 non-null   object 
 11  Star2          1000 non-null   object 
 12  Star3          1000 non-null   object 
 13  Star4          1000 non-null   object 
 14  No_of_Votes    1000 non-null   int64  
 15  Gross          1000 non-null   float64
 16  Decade         1000 non-null   Int64  
dtypes: Int64(2), float64(3), int64(1), object(11)
memory usage: 134.9+ KB
```

The dataset has 1000 entries with 17 columns, all of which have no missing values. It contains text (11 object columns), numbers (2 Int64, 1 int64, and 3 float64), and takes 134.9 KB of memory, making it clean and ready for analysis.

```
df = df[(df["Released_Year"] >= 1900) & (df["Released_Year"] <= 2025)]  # Keep valid years only
```

This filters the dataset to keep only rows where "Released_Year" is between 1900 and 2025, ensuring all movie years are valid. Any row with an invalid year outside this range is removed.

```
print(df["Released_Year"].min(), df["Released_Year"].max())
```

```
1920 2020
```

This prints the earliest (min) and latest (max) release years in the dataset. The dataset contains movies from 1920 to 2020.

```
print(df[~df["Released_Year"].between(1900, 2025)])
```

```
Empty DataFrame
Columns: [Poster_Link, Series_Title, Released_Year, Certificate, Runtime, Genre, IMDB_Rating, Overview, Meta_score, Director, S
tar1, Star2, Star3, Star4, No_of_Votes, Gross, Decade]
Index: []
```

This checks if there are any movies outside the range of 1900 to 2025 in the dataset. The result is an empty DataFrame, meaning all movies fall within this range and no invalid years exist.

```
print("Rows before filtering:", len(df))
df = df[(df["Released_Year"] >= 1900) & (df["Released_Year"] <= 2025)]
print("Rows after filtering:", len(df))
```

```
Rows before filtering: 1000
Rows after filtering: 1000
```

This checks how many rows are in the dataset before and after filtering for valid years (1900-2025). Since the number of rows remains 1000, it means all data already met the criteria, and no rows were removed.

1. SUMMARY OF MILESTONE 1

I started by loading the IMDb Top 1000 Movies dataset into a pandas DataFrame and performed an initial inspection using .head(), .info(), and .describe() to understand its structure. I checked for missing values and duplicates, ensuring data integrity. Missing values in relevant columns were handled, such as filling in missing years with the most common value and recalculating the 'Decade' column accordingly. I converted columns like 'Released_Year' and 'Gross' into appropriate numeric formats for analysis. To ensure data consistency, I filtered out any invalid release years outside the range of 1900-2025. Finally, I visualized key trends using boxplots and histograms to understand the distribution of IMDb ratings and movie release trends across decades.

6. Background research and literature review

Before conducting my exploratory analysis on the IMDb Top 1000 movies, I explored similar research projects to understand how others have approached movie data analysis. This background research helped me identify key factors influencing movie success, common data analysis techniques, and effective ways to visualize insights. By studying existing projects, I was able to refine my methodology, ensuring that my work builds on previous findings while offering new perspectives.

I came across several studies that analyzed movie datasets, each with its own focus. Some explored how different attributes—like genre, budget, and release date—affect a movie's success, while others examined audience sentiment or industry trends over time. Seeing the variety of methods used gave me a clearer idea of how to structure my analysis and what insights I should be looking for.

Below, I summarize three related projects that apply exploratory data analysis to movie-related datasets. These studies, while not identical to mine, share similar objectives and methodologies, offering useful comparisons for my research.

6.1 An Exploratory Data Analysis of Movie Review Dataset

This study examines a dataset of movie reviews to uncover patterns affecting a film's success. The researchers analyzed multiple attributes, including genre, release date, language, country of origin, budget, and revenue. In addition to these direct attributes, the study derived new variables, such as release month and profit margins, to assess their influence on movie performance.

The methodology involved statistical observations and data visualization techniques, including bar charts and heatmaps, to identify trends in the dataset. Key findings showed that Drama and Comedy were the most common genres, December was a peak release month associated with higher revenues, and English-language films from the USA dominated the dataset.

This research helped me understand the importance of considering multiple attributes when analyzing movies. It also reinforced the value of visualizing data to identify trends more effectively, which influenced my decision to use graphical representations in my project.

Link to the study: An Exploratory Data Analysis of Movie Review Dataset

---

6.2 Exploratory and Sentiment Analysis of Netflix Data

This project combines exploratory data analysis (EDA) and sentiment analysis on a dataset of Netflix content. The researchers aimed to uncover patterns in viewer preferences and content trends by integrating multiple data sources, including geographical data and user reviews.

The EDA phase focused on analyzing the distribution of content types (movies vs. series), genre popularity, and IMDb scores. The study also incorporated sentiment analysis to assess audience perceptions based on user reviews. Python's data visualization libraries, such as Matplotlib and Seaborn, were used to create correlation heatmaps and genre-based word clouds.

The research findings provided insights into the types of content that perform well on streaming platforms, the impact of user sentiment on ratings, and trends in content consumption. This study was particularly useful for my project because it demonstrated how sentiment analysis could complement traditional exploratory analysis, inspiring me to consider user feedback and ratings as part of my research.

Link to the study: Exploratory and Sentiment Analysis of Netflix Data

---

6.3 Exploratory Data Analysis of the IMDb's Movie Database from a Data Scientist's Perspective

This study focuses on IMDb's movie database to identify trends in film production, ratings, and duration over time. The researcher examined the number of movies produced annually, the average ratings across different periods, and the correlation between film length and audience ratings.

One of the key findings was that film production has steadily increased over the years, with a peak in recent decades. The study also observed that the average IMDb ratings remained relatively stable, with most movies receiving scores around 6/10. Additionally, a slight correlation was found between film duration and higher ratings, suggesting that longer movies tend to receive better audience scores.

This project helped me understand the significance of historical trends in movie analysis. By looking at how film production and ratings evolved over time, I realized the importance of including time-based patterns in my own study.

Link to the study: Exploratory Data Analysis of the IMDb's Movie Database from a Data Scientist's Perspective

7. Comparison of related studies

To summarize the differences and similarities between these projects, the following table provides a structured comparison:

Aspect	An Exploratory Data Analysis of Movie Review Dataset	Exploratory and Sentiment Analysis of Netflix Data	Exploratory Data Analysis of the IMDb's Movie Database
Objective	Analyze factors influencing movie success	Analyze Netflix content trends and user sentiment	Identify trends in IMDb's movie database over time
Methodology	Statistical analysis, bar charts, heatmaps	EDA, sentiment analysis, visualizations	EDA, time-based trend analysis
Key Findings	Drama and Comedy are most common genres; December releases correlate with higher revenue; English-language films dominate	Netflix content distribution patterns; genre popularity; IMDb rating trends; user sentiment analysis	Film production has increased; IMDb ratings remain stable; longer films tend to have higher ratings
Tools Used	Data visualization tools (unspecified)	Python (Matplotlib, Seaborn), NLP for sentiment analysis	Python (Matplotlib, Seaborn), time series analysis
Dataset Source	Movie review dataset	Netflix dataset from Kaggle, supplemented with additional data	IMDb movie database

8. Conclusion

These studies provided me with a strong foundation for my own research. By analyzing different aspects of movie data—financial success, audience sentiment, and historical trends—I gained insights into various approaches and methodologies. Reviewing these projects highlighted the importance of data visualization and the value of combining quantitative and qualitative data.

9. References

[1] What is Natural Language Processing? | IBM

[2] What is Exploratory Data Analysis? | IBM

[3] Intro-to-exploratory-data-analysis-eda-in-python

[4] Hands-On Exploratory Data Analysis with Python

[5] Cleaning dataset in Python

[6] Exploratory data analysis on Netflix Dataset

[7] Exploratory Data Analysis and Latent Dirichlet Allocation

[8] An Exploratory Data Analysis of Movie Review Dataset

[9] Exploratory Data Analysis of the IMDb's Movie Database from a Data Scientist's Perspective

[10] Exploratory and Sentiment Analysis of Netflix Data

10.Performing data analysis

10.1 Distribution of IMDB Ratings

```
#MILESTONE 2 STARTING FROM HERE


import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Set the aesthetic style of the plots
sns.set_style("whitegrid")
```

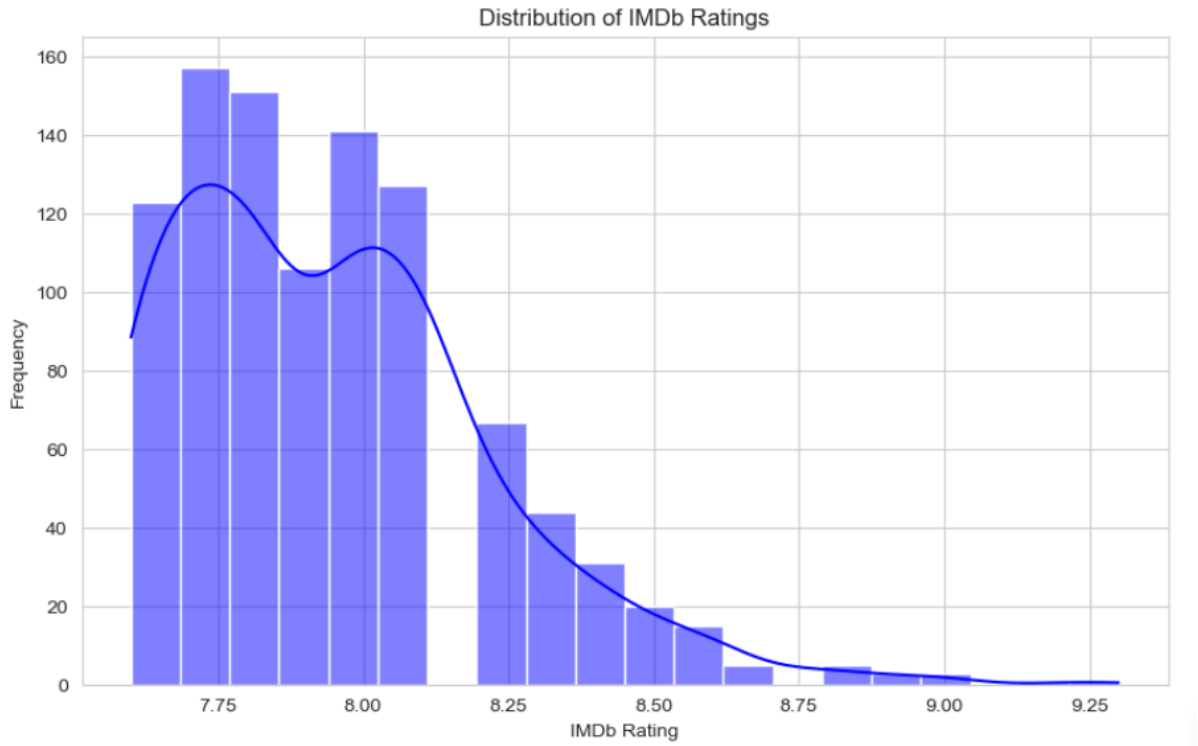
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
file_path = r"C:\Users\Korisnik\Downloads\archive\imdb_top_1000.csv"
df = pd.read_csv(file_path)

# Set style
sns.set_style("whitegrid")
```

In the first code snippet, I imported the necessary libraries: pandas for data handling, matplotlib.pyplot for creating plots, and seaborn for statistical visualizations. I then set the plot style to "whitegrid" to make graphs more readable with a clean background and grid lines.

In the second code snippet, I repeated the imports but also loaded the IMDb dataset. I specified the file path as a raw string to prevent errors with backslashes and used pd.read_csv(file_path) to read the data into a pandas DataFrame. Finally, I set the "whitegrid" style again to ensure consistency in visualization aesthetics.

```
plt.figure(figsize=(10, 6))
sns.histplot(df['IMDB_Rating'], bins=20, kde=True, color='blue')
plt.title('Distribution of IMDb Ratings')
plt.xlabel('IMDb Rating')
plt.ylabel('Frequency')
plt.show()
```

I created a histogram to show how IMDb ratings are distributed. The bars represent how many movies fall into different rating ranges, and the smooth curve helps visualize the overall trend. I also added labels and a title to make the chart clear, and set the color to blue for better visibility.



Picture 5. Distribution of IMDB Ratings

This graph shows the distribution of IMDb ratings for movies. Most movies have ratings between 7.7 and 8.1, with the highest frequency around these values. The smooth curve shows the general trend, indicating that higher-rated movies (above 8.5) are much less common. The distribution is right-skewed, meaning fewer movies have very high ratings.

10.2 Top 10 highest-rated movies

```
top_rated = df.nlargest(10, 'IMDB_Rating')[['Series_Title', 'IMDB_Rating']]
print(top_rated)
```

This code selects the top 10 highest-rated movies from the dataset based on their IMDb ratings and displays their titles and ratings.

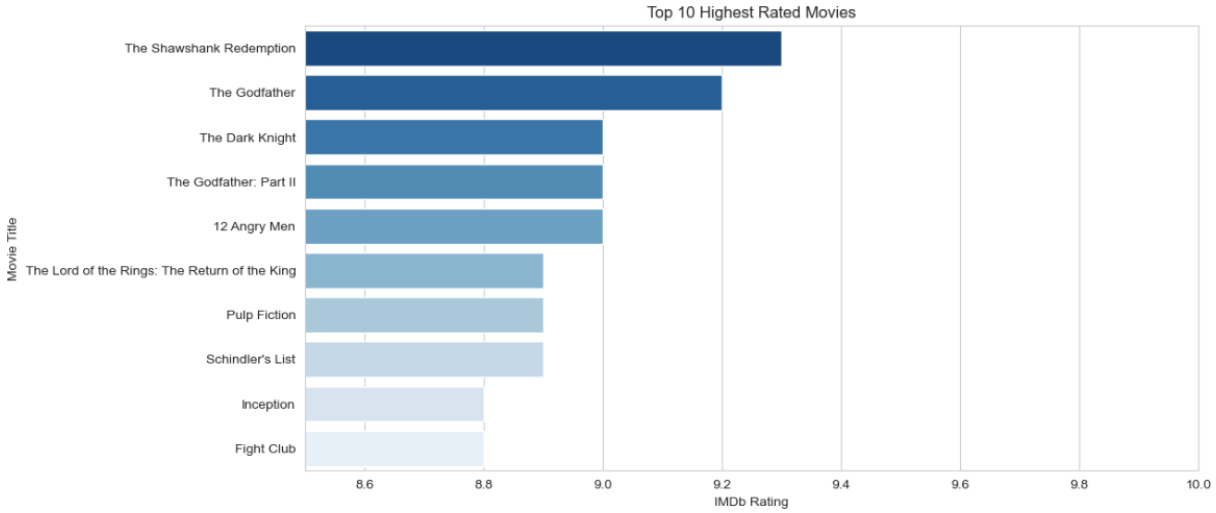
```
                                  Series_Title  IMDB_Rating
0                      The Shawshank Redemption          9.3
1                                 The Godfather          9.2
2                               The Dark Knight          9.0
3                        The Godfather: Part II          9.0
4                                   12 Angry Men          9.0
5  The Lord of the Rings: The Return of the King          8.9
6                                   Pulp Fiction          8.9
7                               Schindler's List          8.9
8                                     Inception          8.8
9                                   Fight Club          8.8
```

The output shows the top 10 movies with the highest IMDb ratings, with The Shawshank Redemption ranked highest at 9.3, followed by The Godfather at 9.2. Several other classics, like The Dark Knight, 12 Angry Men, and Pulp Fiction, also appear in the list with high ratings above 8.8.

```
top_rated = df.nlargest(10, 'IMDB_Rating')[['Series_Title', 'IMDB_Rating']]

# Plot top-rated movies
plt.figure(figsize=(12, 6))
sns.barplot(y=top_rated['Series_Title'], x=top_rated['IMDB_Rating'], palette='Blues_r')
plt.title('Top 10 Highest Rated Movies')
plt.xlabel('IMDb Rating')
plt.ylabel('Movie Title')
plt.xlim(8.5, 10)
plt.show()
```

This code selects the top 10 highest-rated movies from the dataset based on their IMDb ratings and then creates a horizontal bar chart to visualize them.

First, it extracts the movie titles and their IMDb ratings. Then, it sets the figure size and uses Seaborn to create a bar plot with the movie titles on the y-axis and their IMDb ratings on the x-axis. The palette='Blues_r' makes the bars appear in shades of blue. The x-axis is limited to a range of 8.5 to 10 to focus on high ratings. Finally, the plot is displayed, making it easy to compare the highest-rated movies visually.



Picture 6. Top 10 Highest Rated Movies

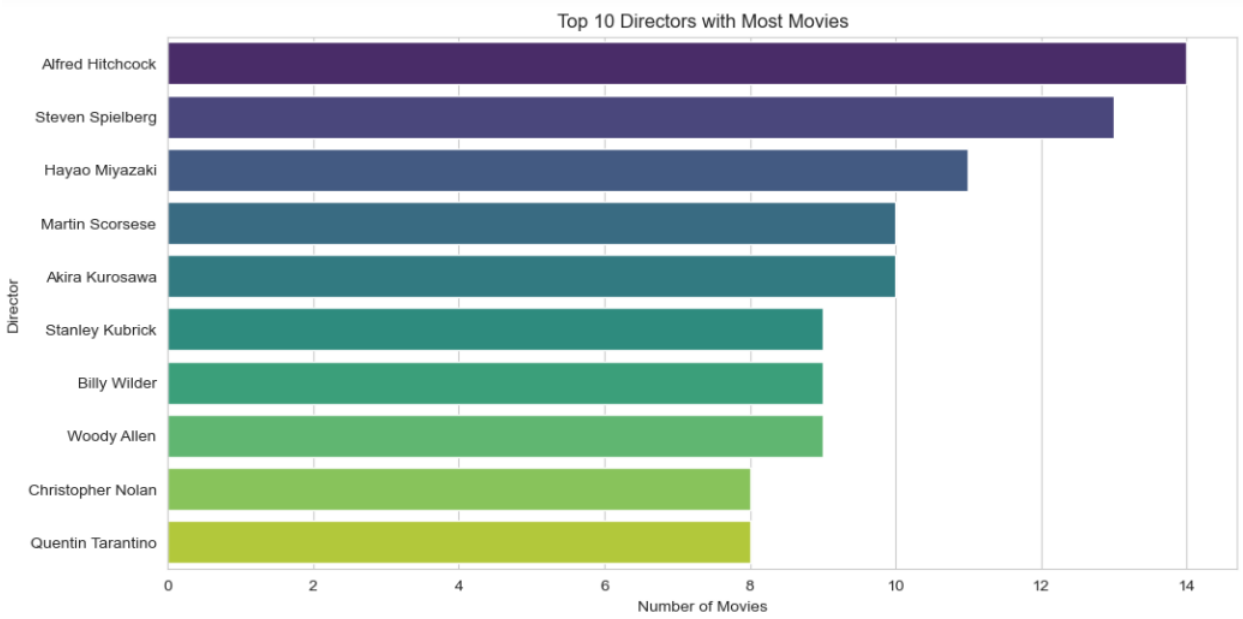
This bar chart shows the top 10 highest-rated movies based on IMDb ratings. "The Shawshank Redemption" has the highest rating, followed closely by "The Godfather" and "The Dark Knight."

The bars represent the IMDb ratings of each movie, with darker shades of blue indicating higher ratings. The x-axis ranges from 8.5 to 10, focusing only on the highest-rated films. This visualization makes it easy to compare the ratings of these top movies briefly.

10.3 Top 10 Directors with Most Movies

```
plt.figure(figsize=(12, 6))
sns.countplot(y=df['Director'], order=df['Director'].value_counts().head(10).index, palette='viridis')
plt.title('Top 10 Directors with Most Movies')
plt.xlabel('Number of Movies')
plt.ylabel('Director')
plt.show()
```

This code creates a bar chart that shows the top 10 directors with the most movies in the dataset. It first counts how many movies each director has and selects the top 10. The bars represent the number of movies directed by each person, with the longest bars showing the most prolific directors. The viridis color palette is used to make the chart visually appealing, and labels are added for clarity. This visualization helps identify which directors have the highest number of films in the IMDb Top 1000 list.



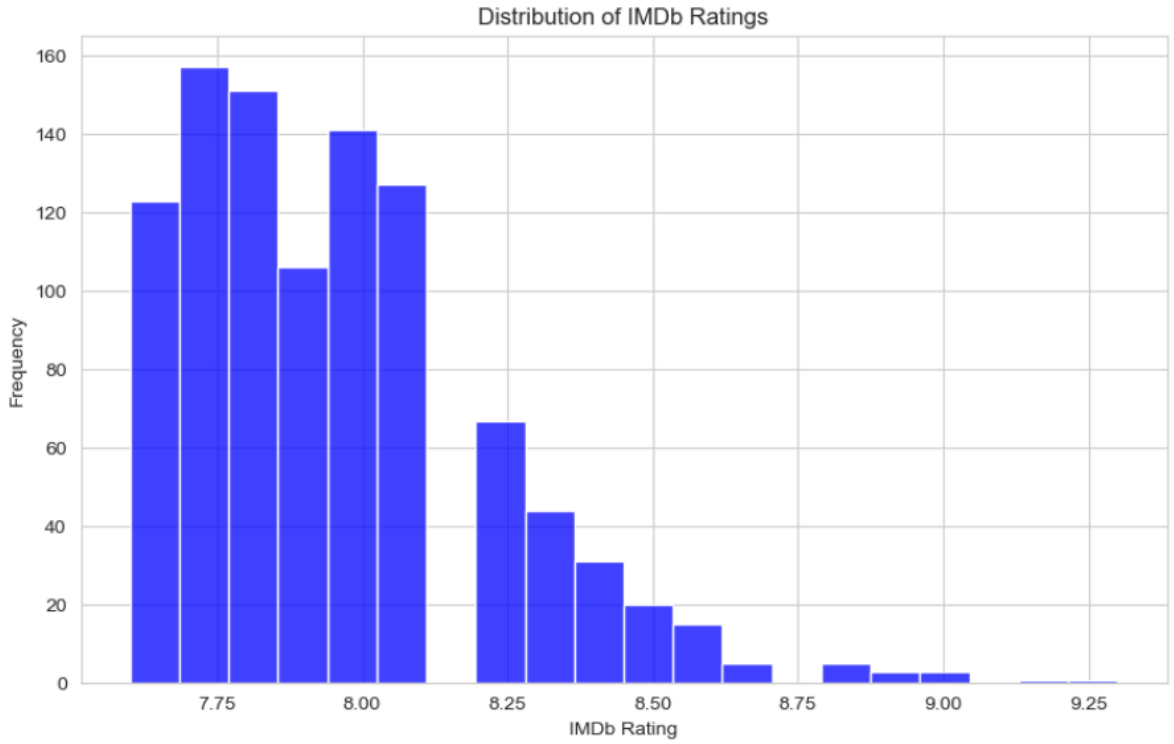
Picture 7. Top 10 Directors with Most Movies

This bar chart shows the top 10 directors with the most movies in the IMDb Top 1000 list. Alfred Hitchcock leads with the highest number of films, followed closely by Steven Spielberg and Hayao Miyazaki. The bars represent the number of movies each director has in the dataset, with directors like Martin Scorsese, Akira Kurosawa, and Christopher Nolan also making the list. The viridis color gradient helps distinguish between different counts, making it easy to compare. This visualization highlights the most influential directors in highly rated cinema.

10.4 Distribution of IMDb Ratings

```
plt.figure(figsize=(10, 6))
sns.histplot(df['IMDB_Rating'], bins=20, kde=False, color='blue')
plt.title('Distribution of IMDb Ratings')
plt.xlabel('IMDb Rating')
plt.ylabel('Frequency')
plt.show()
```

This code creates a histogram to show the distribution of IMDb ratings in the dataset. It divides the ratings into 20 bins (intervals) and counts how many movies fall into each range, displaying the frequency on the y-axis. The histogram is colored blue, and the kde=False setting means no smooth density curve is added. The plot helps visualize how IMDb ratings are spread, showing common rating ranges and whether the distribution is skewed or balanced.



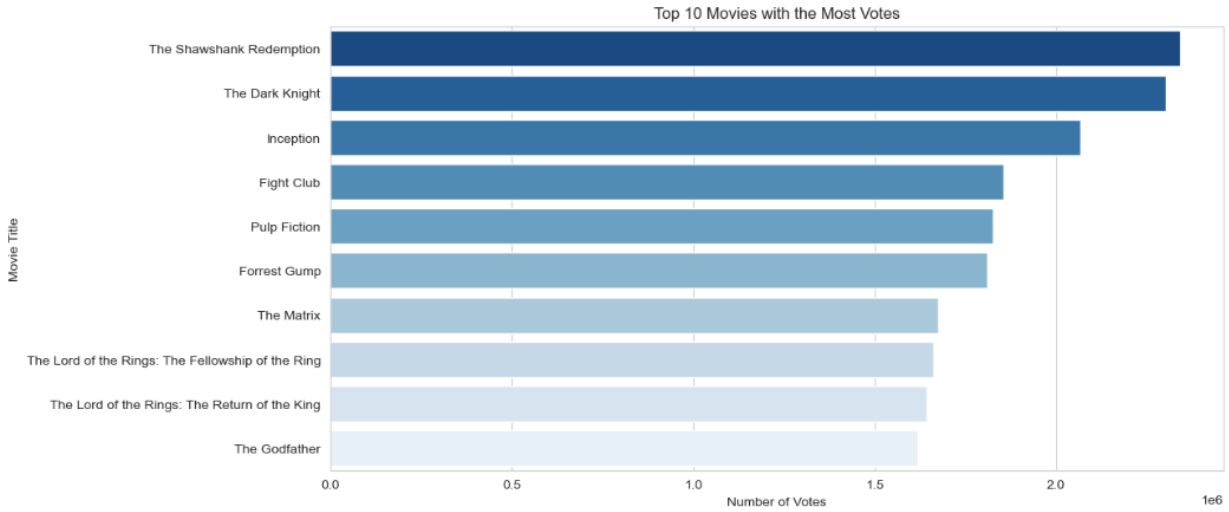
Picture 8. Distribution of IMDB Ratings - 2

This histogram shows how IMDb ratings are distributed. The x-axis represents IMDb ratings, while the y-axis shows how many movies fall into each rating range. Most movies have ratings between 7.5 and 8.2, with fewer movies rated higher than 8.5. The distribution is right-skewed, meaning high ratings (above 9) are rare.

10.5 Top 10 Movies With The Highest Number Of Votes

```
top_voted_movies = df.nlargest(10, 'No_of_Votes')[['Series_Title', 'No_of_Votes']]

plt.figure(figsize=(12, 6))
sns.barplot(y=top_voted_movies['Series_Title'], x=top_voted_movies['No_of_Votes'], palette='Blues_r')
plt.title('Top 10 Movies with the Most Votes')
plt.xlabel('Number of Votes')
plt.ylabel('Movie Title')
plt.show()
```

This code selects the top 10 movies with the highest number of votes from the dataset. It then creates a bar chart where the movie titles are displayed on the y-axis and the number of votes on the x-axis. The bars are colored using a blue palette for better visualization. Finally, the chart is displayed with a title and labeled axes for clarity.



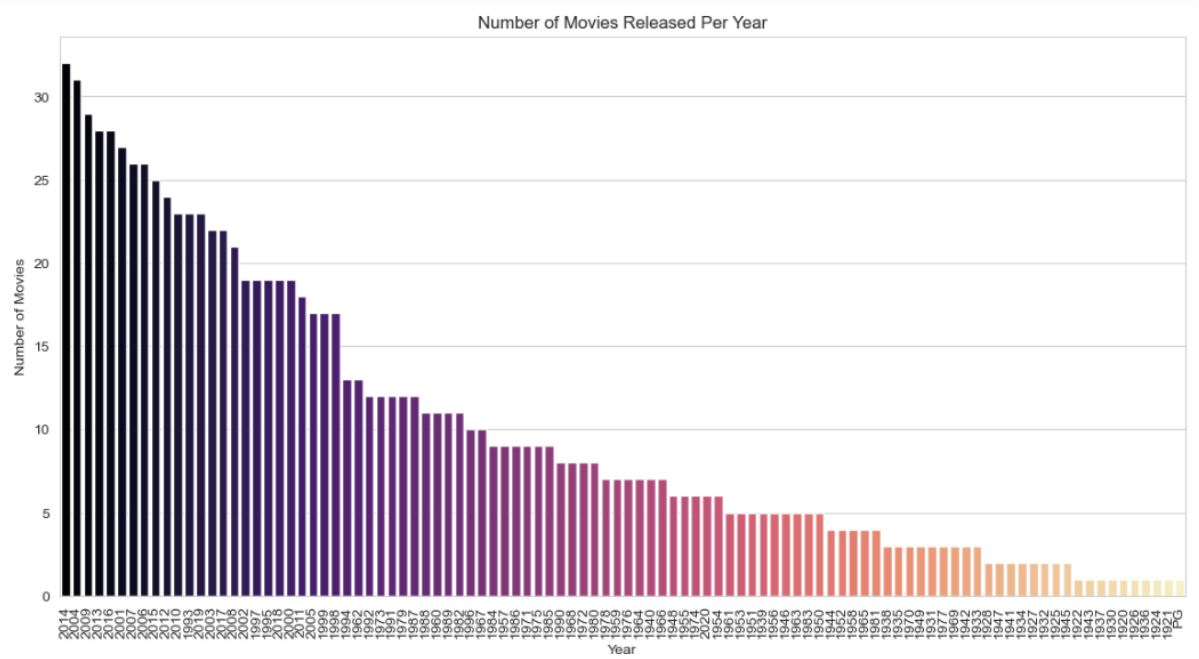
Picture 9. Top 10 Movies with the Most Votes

This bar chart shows the top 10 movies with the highest number of votes. The x-axis represents the number of votes, while the y-axis lists the movie titles. Darker shades of blue indicate movies with more votes. "The Shawshank Redemption" has the most votes, followed by "The Dark Knight" and "Inception."

10.6 Number of Movies Released Per Year

```
plt.figure(figsize=(14, 7))
sns.barplot(x=df['Released_Year'].value_counts().index,
            y=df['Released_Year'].value_counts().values,
            palette='magma')
plt.title('Number of Movies Released Per Year')
plt.xlabel('Year')
plt.ylabel('Number of Movies')
plt.xticks(rotation=90)
plt.show()
```

This code creates a bar chart showing the number of movies released each year. The x-axis represents the release years, and the y-axis represents the number of movies released in each year. The bars are colored using the "magma" palette. The figure size is set to 14x7 for better visibility, and the x-axis labels are rotated 90 degrees to make them easier to read.



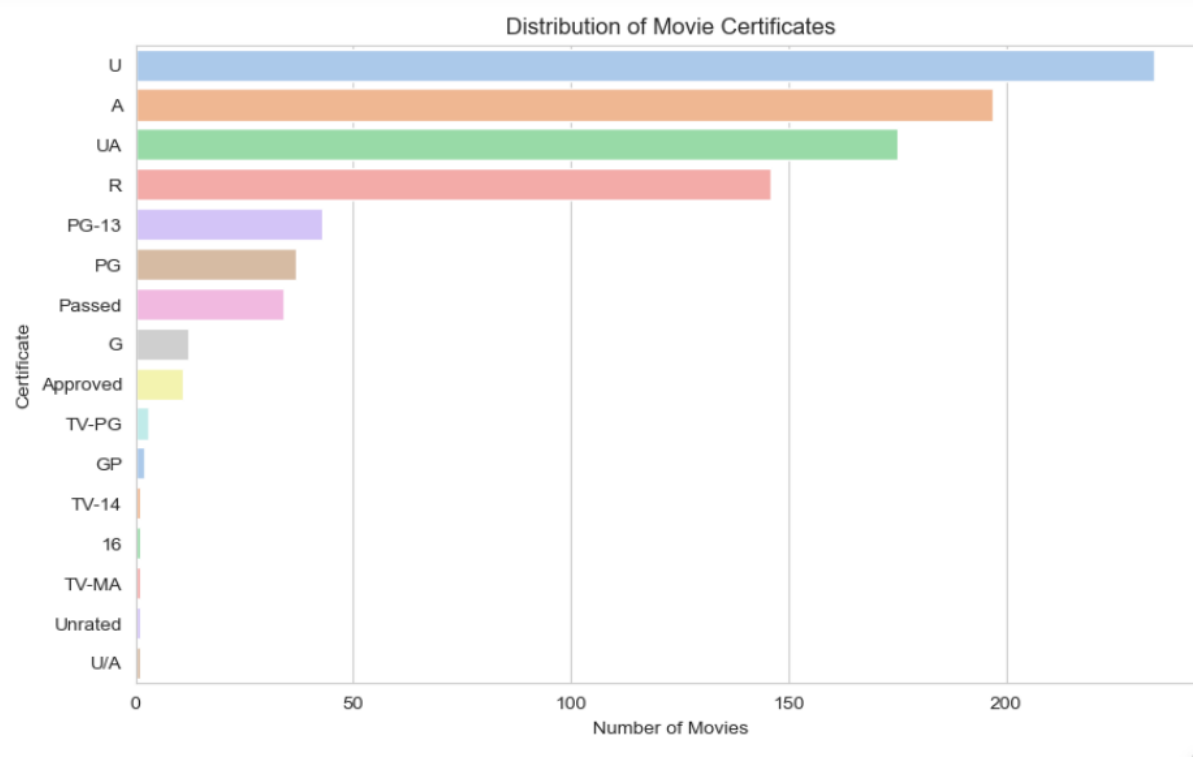
Picture 10. Number of Movies Released per Year

This bar chart shows the number of movies released per year. The x-axis represents the years, and the y-axis represents the number of movies released in each year. The bars are colored using the "magma" palette, transitioning from dark to light shades. The data suggests that more movies were released in recent years compared to older years. The x-axis labels are rotated to make them readable.

10.7 Distribution of Movie Certificates

```
plt.figure(figsize=(10, 6))
sns.barplot(y=df['Certificate'].value_counts().index,
            x=df['Certificate'].value_counts().values,
            palette='pastel')
plt.title('Distribution of Movie Certificates')
plt.xlabel('Number of Movies')
plt.ylabel('Certificate')
plt.show()
```

The code creates a horizontal bar chart showing the distribution of movie certificates. It first calculates the count of each unique certificate in the dataset and then plots these values using Seaborn. The x-axis represents the number of movies, while the y-axis represents different certificate categories. The pastel color palette is used for visualization.



Picture 11. Distribution of Movie Certificates

The graph visually represents the frequency of movie certificates, showing that the most common certificates are "U," "A," "UA," and "R." These categories have the highest number of movies, while other certifications like "TV-MA" and "Unrated" appear less frequently. This distribution highlights the dominance of general audience and restricted content ratings in the dataset.

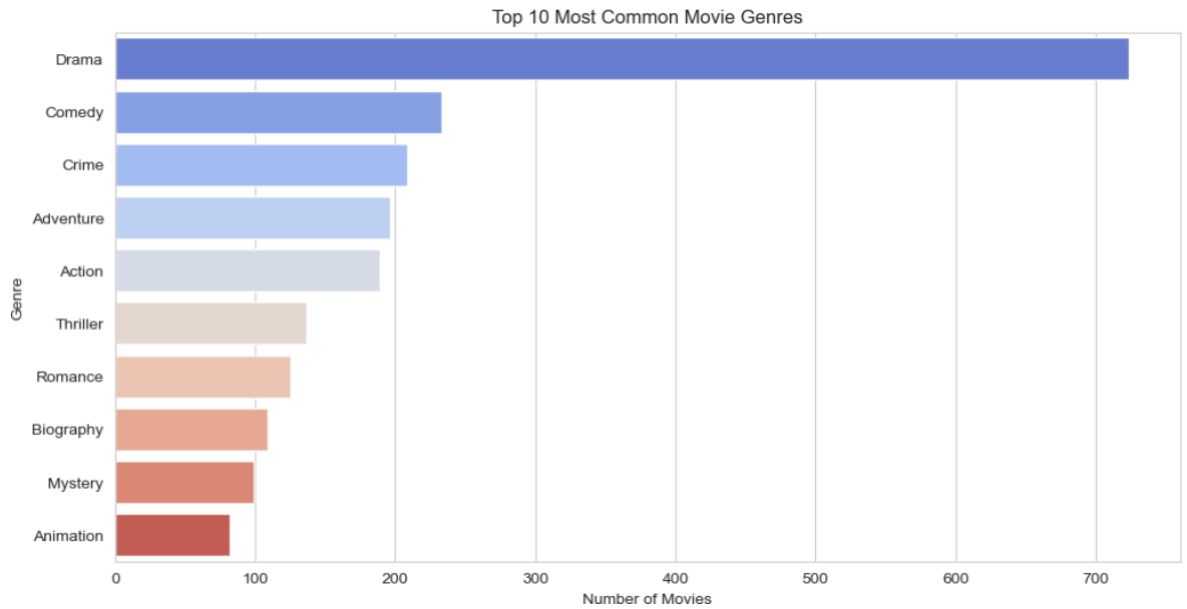
```
df = pd.read_csv(r"C:\Users\Korisnik\Downloads\archive\imdb_top_1000.csv")

print(df.head())  # Check if Genre column has values again
```

This code reads a CSV file named "imdb_top_1000.csv" from a specified directory using pandas and stores it in a DataFrame called df. The raw string notation (r"") ensures that backslashes in the file path are treated correctly. The df.head() function prints the first five rows of the DataFrame to check if the data, specifically the "Genre" column, is loaded correctly. The comment suggests that the user is verifying whether the "Genre" column has values after reloading the dataset.

10.8 Top 10 Movies Genres

```
df_exploded = df.assign(Genre=df['Genre'].str.split(', ')).explode('Genre')
genre_counts = df_exploded['Genre'].value_counts().head(10)

plt.figure(figsize=(12, 6))
sns.barplot(x=genre_counts.values, y=genre_counts.index, palette='coolwarm')
plt.title('Top 10 Most Common Movie Genres')
plt.xlabel('Number of Movies')
plt.ylabel('Genre')
plt.show()
```

This code processes the "Genre" column in the dataset to analyze the distribution of movie genres. Since some movies have multiple genres listed together, the code first splits these entries and expands them into separate rows using the explode function. Then, it counts the occurrences of each genre and selects the top 10 most common ones. Finally, it visualizes the results using a bar chart, where the x-axis represents the number of movies, and the y-axis lists the genres. This visualization helps identify the most prevalent genres in the dataset.



Picture 12. Number of Movies Released per Year

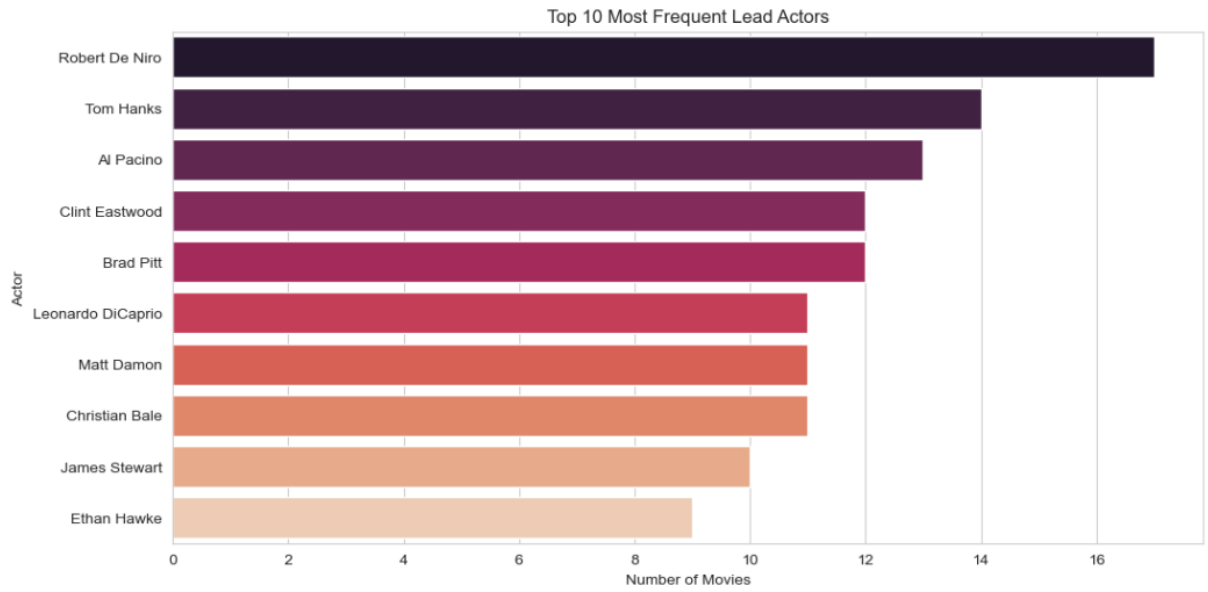
This bar chart visualizes the top movie genres based on their frequency in the dataset. Drama is the most common genre, followed by Comedy and Crime. The x-axis represents the number of movies, while the y-axis lists the genres. The chart helps identify the most

frequently occurring genres in the dataset, giving insights into the dominant trends in popular movies.

10.9 Top 10 Most Frequent Lead Actors

```
# Combine all stars into a single column
all_actors = pd.concat([df['Star1'], df['Star2'], df['Star3'], df['Star4']])

# Count the occurrences of each actor
actor_counts = all_actors.value_counts().head(10)

# Plot
plt.figure(figsize=(12, 6))
sns.barplot(y=actor_counts.index, x=actor_counts.values, palette='rocket')
plt.title('Top 10 Most Frequent Lead Actors')
plt.xlabel('Number of Movies')
plt.ylabel('Actor')
plt.show()
```

This code analyzes the most frequently appearing actors in the dataset. It first combines the four columns representing lead actors into a single column. Then, it counts the occurrences of each actor and selects the top 10. Finally, it visualizes the results using a horizontal bar chart, where the x-axis represents the number of movies and the y-axis lists the most frequently appearing actors. This helps identify which actors have starred in the most movies in the dataset.



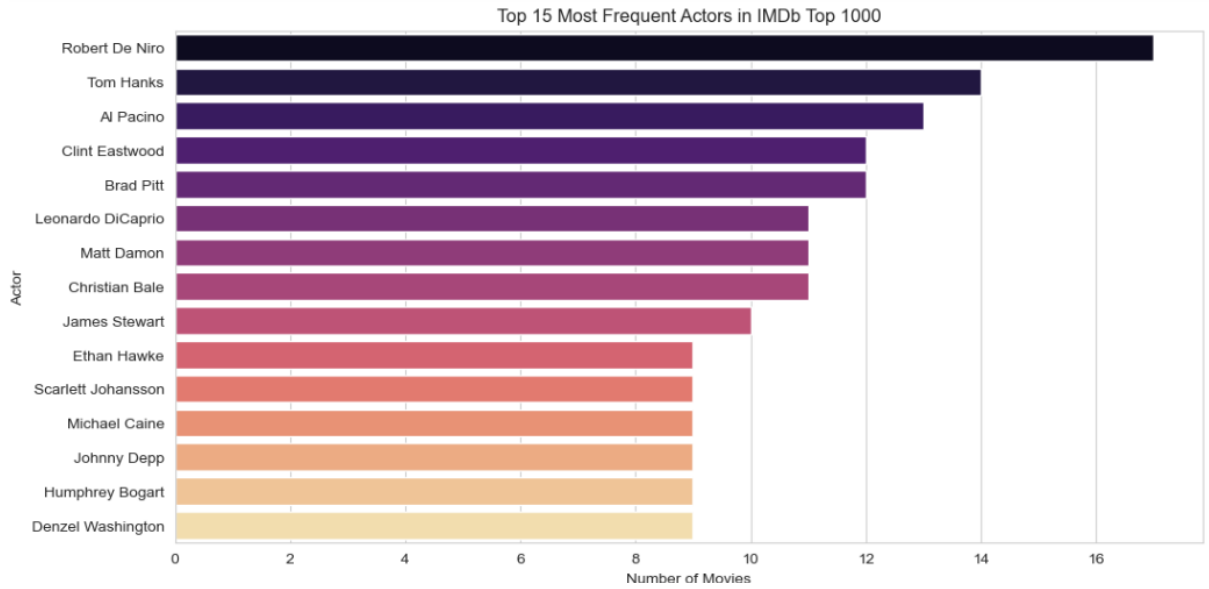
Picture 13. Top 10 Most Frequent Lead Actors

This bar chart visualizes the top 10 most frequently appearing lead actors in the dataset. The x-axis represents the number of movies, while the y-axis lists the actors. Robert De Niro appears in the highest number of films, followed by Tom Hanks, Al Pacino, and others. The chart provides insight into which actors have the most appearances in top-rated movies, highlighting their prominence in the film industry.

10.10 Top 15 Most Frequent Actors in IMDb Top 1000

```
# Combine all actors into a single column
all_actors = pd.concat([df['Star1'], df['Star2'], df['Star3'], df['Star4']])

# Count occurrences
actor_counts = all_actors.value_counts().head(15)

# Plot
plt.figure(figsize=(12, 6))
sns.barplot(y=actor_counts.index, x=actor_counts.values, palette='magma')
plt.title('Top 15 Most Frequent Actors in IMDb Top 1000')
plt.xlabel('Number of Movies')
plt.ylabel('Actor')
plt.show()
```

This code analyzes the most frequently appearing actors in the IMDb Top 1000 movies. It first combines the actor columns into a single column, ensuring that all listed actors are included. Then, it counts how many times each actor appears in the dataset and selects the top fifteen. Finally, it creates a horizontal bar chart where the x-axis represents the number of movies, and the y-axis lists the actors. The "magma" color palette is used to enhance the visualization. This helps in identifying which actors appear most often in highly-rated films.



Picture 14. Top 15 Most Frequent Actors in IMDb Top 1000

This bar chart shows the top 15 actors who appear most frequently in the IMDb Top 1000 movies. Robert De Niro appears the most, followed by Tom Hanks and Al Pacino. The x-axis represents the number of movies each actor has starred in, while the y-axis lists the actors. The "magma" color palette gives the bars a gradient effect, with darker shades for higher counts. This visualization highlights which actors have had the most consistent presence in highly rated films.

10.11 Al Pacino Movie Analysis

```
# Filter movies where Al Pacino is in any of the star columns
al_pacino_movies = df[(df['Star1'] == 'Al Pacino') |
                      (df['Star2'] == 'Al Pacino') |
                      (df['Star3'] == 'Al Pacino') |
                      (df['Star4'] == 'Al Pacino')]

# Count movies
num_al_pacino_movies = len(al_pacino_movies)

print(f'Al Pacino has acted in {num_al_pacino_movies} movies in the IMDb Top 1000.')

# Display movies
print(al_pacino_movies[['Series_Title', 'Released_Year', 'IMDB_Rating']])
```

This code filters movies where Al Pacino is listed in any of the four-star columns. It then counts how many times he appears and prints the result. Finally, it displays a table with the

movie title, release year, and IMDb rating for the films he has acted in. This helps analyze Al Pacino's presence in top-rated movies.

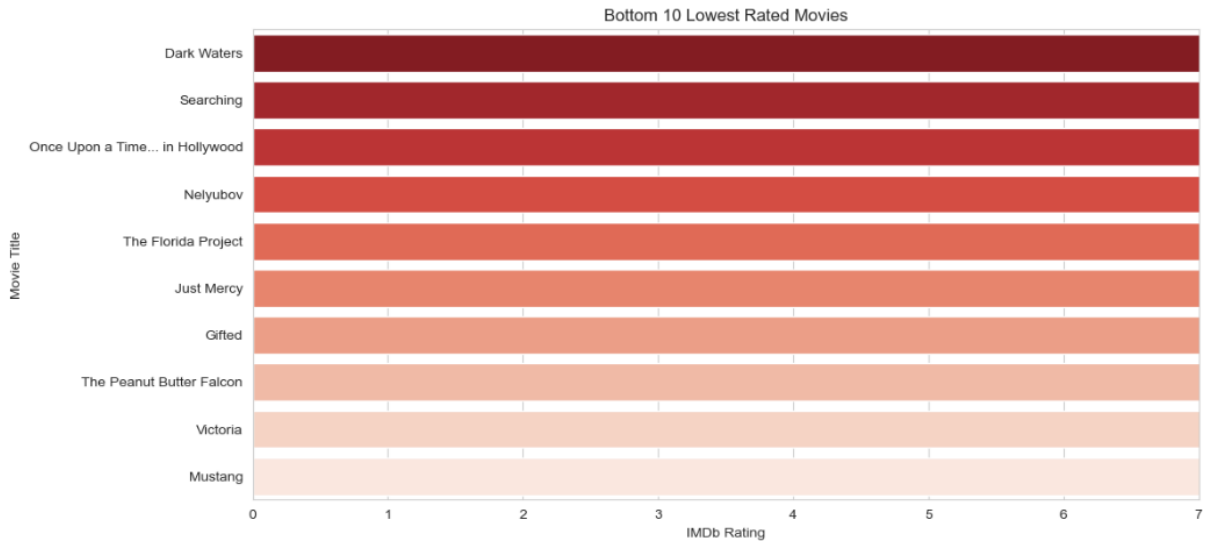
```
Al Pacino has acted in 13 movies in the IMDb Top 1000.
                Series_Title Released_Year  IMDB_Rating
1               The Godfather          1972          9.2
3     The Godfather: Part II           1974          9.0
108                  Scarface          1983          8.3
164                      Heat          1995          8.2
398         Scent of a Woman           1992          8.0
416        Dog Day Afternoon           1975          8.0
484              The Irishman          2019          7.9
523            Carlito's Way           1993          7.9
649              The Insider           1999          7.8
809            Donnie Brasco           1997          7.7
823      Glengarry Glen Ross           1992          7.7
849                  Serpico           1973          7.7
974  The Godfather: Part III           1990          7.6
```

The output shows that Al Pacino has appeared in 13 movies that are ranked among IMDb's top 1000. The table lists these movies along with their release years and IMDb ratings. His highest-rated films include *The Godfather* (9.2) and *The Godfather: Part II* (9.0), while *The Godfather: Part III* has the lowest rating at 7.6. The list highlights his most iconic performances in crime and drama films spanning several decades.

10.12 Bottom 10 Lowest Rated Movies

```python
# Get the lowest-rated movies
lowest_rated_movies = df.nsmallest(10, 'IMDB_Rating')[['Series_Title', 'IMDB_Rating', 'Released_Year']]

# Plot
plt.figure(figsize=(12, 6))
sns.barplot(y=lowest_rated_movies['Series_Title'], x=lowest_rated_movies['IMDB_Rating'], palette='Reds_r')
plt.title('Bottom 10 Lowest Rated Movies')
plt.xlabel('IMDb Rating')
plt.ylabel('Movie Title')
plt.xlim(0, 7)  # Adjust x-axis
plt.show()

# Display movies
print(lowest_rated_movies)
```

This code finds the 10 lowest-rated movies in the dataset based on IMDb ratings. It selects the movie titles, ratings, and release years. Then, it creates a horizontal bar chart to show these movies, using a red color palette. The x-axis is adjusted to range from 0 to 7. Finally, it prints the list of these lowest-rated movies.



Picture 15. Bottom 10 Lowest Rated Movies

This bar chart shows the 10 lowest-rated movies in the IMDb Top 1000. The movies are listed on the y-axis, and their IMDb ratings are on the x-axis. Dark Waters has the lowest rating among them, while Mustang has the highest rating in this group. The red color gradient represents the rating intensity, with darker shades indicating lower-rated movies.

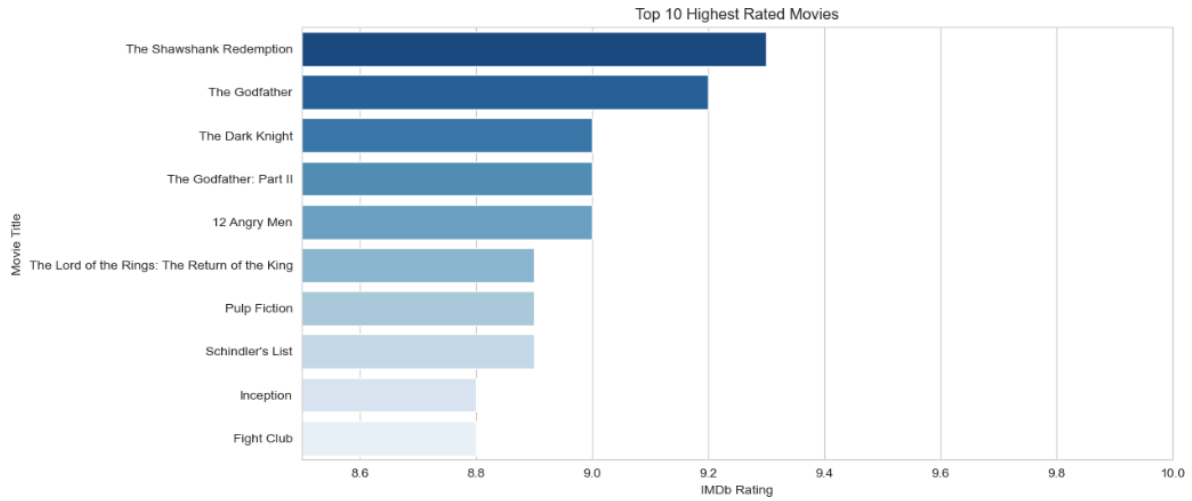
	Series_Title	IMDB_Rating	Released_Year
877	Dark Waters	7.6	2019
878	Searching	7.6	2018
879	Once Upon a Time... in Hollywood	7.6	2019
880	Nelyubov	7.6	2017
881	The Florida Project	7.6	2017
882	Just Mercy	7.6	2019
883	Gifted	7.6	2017
884	The Peanut Butter Falcon	7.6	2019
885	Victoria	7.6	2015
886	Mustang	7.6	2015

This table displays the 10 lowest-rated movies in the IMDb Top 1000. Each movie has an IMDb rating of 7.6, making them the lowest in the dataset. The table includes three columns: the movie title (Series_Title), its IMDb rating (IMDB_Rating), and the year it was released (Released_Year). All these movies were released between 2015 and 2019.

10.13 Top 10 Highest Rated Movies

```
# Get top-rated movies
top_rated_movies = df.nlargest(10, 'IMDB_Rating')[['Series_Title', 'IMDB_Rating', 'Released_Year']]

# Plot
plt.figure(figsize=(12, 6))
sns.barplot(y=top_rated_movies['Series_Title'], x=top_rated_movies['IMDB_Rating'], palette='Blues_r')
plt.title('Top 10 Highest Rated Movies')
plt.xlabel('IMDb Rating')
plt.ylabel('Movie Title')
plt.xlim(8.5, 10)  # Adjust x-axis for better visualization
plt.show()

# Display movies
print(top_rated_movies)
```

This code finds the 10 highest-rated movies in the IMDb Top 1000. It selects movies with the highest IMDb ratings and displays them in a table. The code also creates a bar chart to visualize these top-rated movies, using blue shades for the bars. The x-axis is adjusted to better display the ratings, ranging from 8.5 to 10.



Picture 16. Top 10 Highest Rated Movies

This chart shows the top 10 highest-rated movies on IMDb. *The Shawshank Redemption* has the highest rating, followed by *The Godfather* and *The Dark Knight*. The x-axis represents IMDb ratings, and the darker bars indicate higher ratings. All movies have ratings above 8.6.

	Series_Title	IMDB_Rating	Released_Year
0	The Shawshank Redemption	9.3	1994
1	The Godfather	9.2	1972
2	The Dark Knight	9.0	2008
3	The Godfather: Part II	9.0	1974
4	12 Angry Men	9.0	1957
5	The Lord of the Rings: The Return of the King	8.9	2003
6	Pulp Fiction	8.9	1994
7	Schindler's List	8.9	1993
8	Inception	8.8	2010
9	Fight Club	8.8	1999

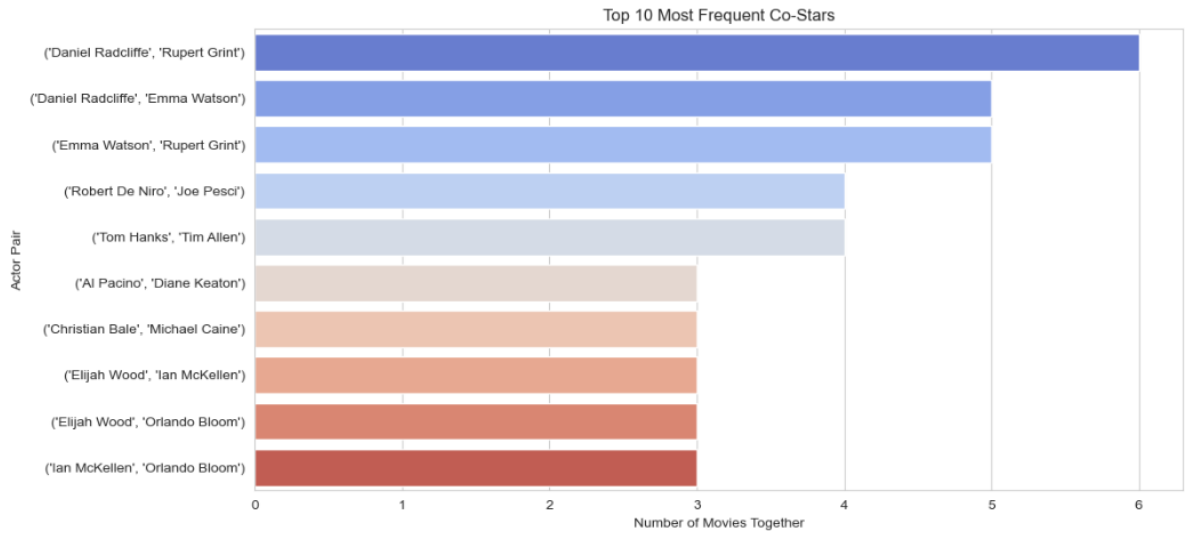
This table lists the top 10 highest-rated movies on IMDb. *The Shawshank Redemption* is ranked highest with a 9.3 rating, followed by *The Godfather* at 9.2. Other highly rated movies include *The Dark Knight*, *12 Angry Men*, and *The Lord of the Rings: The Return of the King*. The ratings range from 8.8 to 9.3.

10.14 Top 10 Most Frequent Co-Stars

```
from itertools import combinations
from collections import Counter

# Create actor pairs from each movie
actor_pairs = []
for i, row in df.iterrows():
    actors = [row['Star1'], row['Star2'], row['Star3'], row['Star4']]
    pairs = combinations(actors, 2)  # Create all 2-actor combinations
    actor_pairs.extend(pairs)

# Count occurrences
pair_counts = Counter(actor_pairs)
top_pairs = pair_counts.most_common(10)

# Convert to DataFrame
top_pairs_df = pd.DataFrame(top_pairs, columns=['Pair', 'Count'])

# Plot
plt.figure(figsize=(12, 6))
sns.barplot(y=top_pairs_df['Pair'].astype(str), x=top_pairs_df['Count'], palette='coolwarm')
plt.title('Top 10 Most Frequent Co-Stars')
plt.xlabel('Number of Movies Together')
plt.ylabel('Actor Pair')
plt.show()
```

This code analyzes the most frequently paired actors in movies. It extracts actor pairs from each movie by selecting up to four main actors and generating all possible two-actor combinations.

It then counts how many times each pair appears together across different films. The most common pairs are stored in a DataFrame, which is then visualized using a bar chart to highlight the top ten most frequent co-stars.



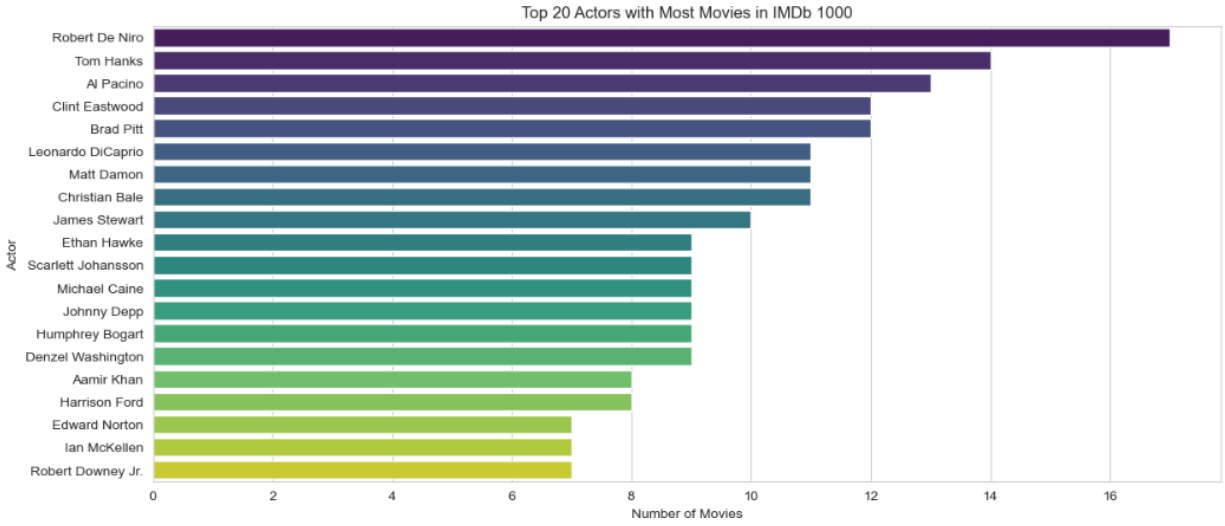
Picture 17. Top 10 Most Frequent Co -Stars

This chart displays the top ten most frequently paired actors in movies. Daniel Radcliffe, Rupert Grint, and Emma Watson appear together the most, reflecting their roles in the Harry Potter series. Other frequent co-stars include Robert De Niro and Joe Pesci, known for their collaborations in crime films, as well as Tom Hanks and Tim Allen, who voiced characters in the Toy Story franchise. The list also includes actors from The Godfather, The Dark Knight trilogy, and The Lord of the Rings, highlighting famous recurring duos in cinema.

10.15 Al Pacino Movie Analysis

```
# Combine all actor columns into a single column
actors = pd.concat([df['Star1'], df['Star2'], df['Star3'], df['Star4']])

# Count occurrences of each actor
actor_counts = actors.value_counts().head(20)  # Top 20 most frequent actors

# Plot
plt.figure(figsize=(14, 6))
sns.barplot(y=actor_counts.index, x=actor_counts.values, palette='viridis')
plt.title('Top 20 Actors with Most Movies in IMDb 1000')
plt.xlabel('Number of Movies')
plt.ylabel('Actor')
plt.show()
```

This code identifies the most frequently appearing actors in the dataset. It combines actor columns into a single list and counts how many times each actor appears. The top 20 actors with the most movies are then displayed in a bar chart. The chart visually represents which actors have been in the highest number of films in the dataset.



Picture 18. Top 20 Actors with Most Movies

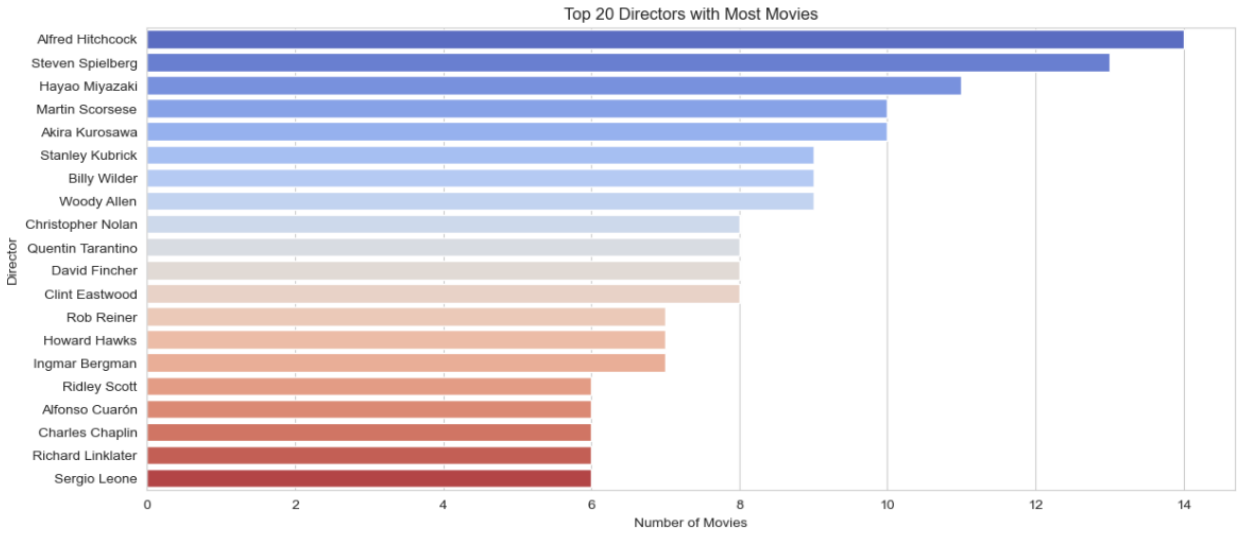
This bar chart shows the top 20 actors who have appeared in the most movies within the dataset. Robert De Niro has the highest number of films, followed by Tom Hanks and Al Pacino. The visualization highlights actors with consistent appearances in highly rated movies.

10.16 Top 20 Directors with Most Movies

```
director_counts = df['Director'].value_counts().head(20)  # Top 20 directors

plt.figure(figsize=(14, 6))
sns.barplot(y=director_counts.index, x=director_counts.values, palette='coolwarm')
plt.title('Top 20 Directors with Most Movies')
plt.xlabel('Number of Movies')
plt.ylabel('Director')
plt.show()
```

This code calculates the number of movies directed by each director and selects the top 20 with the most films. It then creates a bar chart displaying these directors along with their

movie counts. The visualization helps identify the most frequently featured directors in the dataset.



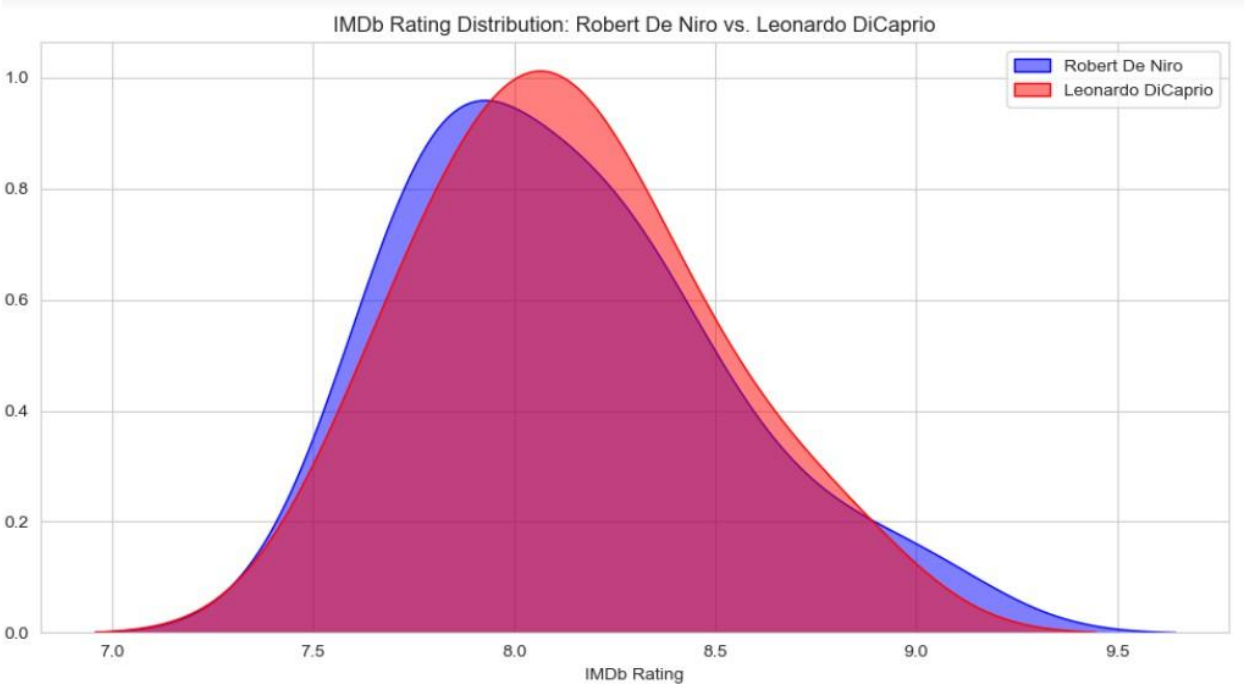
Picture 19. Top 20 Directors with Most Movies

This bar chart displays the top 20 directors with the most movies in the dataset. Alfred Hitchcock, Steven Spielberg, and Hayao Miyazaki are among the most frequently featured directors. The visualization provides insight into the most influential filmmakers based on the number of their movies included.

10.17 IMDb Rating Distribution -Robert De Niro vs. Leonardo DiCaprio

```
# Filter movies with Robert De Niro and Leonardo DiCaprio
de_niro_movies = df[df[['Star1', 'Star2', 'Star3', 'Star4']].eq("Robert De Niro").any(axis=1)]
dicaprio_movies = df[df[['Star1', 'Star2', 'Star3', 'Star4']].eq("Leonardo DiCaprio").any(axis=1)]

# Compare IMDb ratings
plt.figure(figsize=(12, 6))
sns.kdeplot(de_niro_movies['IMDB_Rating'], label='Robert De Niro', fill=True, color='blue', alpha=0.5)
sns.kdeplot(dicaprio_movies['IMDB_Rating'], label='Leonardo DiCaprio', fill=True, color='red', alpha=0.5)
plt.title('IMDb Rating Distribution: Robert De Niro vs. Leonardo DiCaprio')
plt.xlabel('IMDb Rating')
plt.ylabel('Density')
plt.legend()
plt.show()
```

This code filters movies featuring Robert De Niro and Leonardo DiCaprio. It then compares their IMDb rating distributions using a density plot. The blue area represents De Niro's movies, while the red area represents DiCaprio's movies. The visualization helps analyze the rating patterns of films starring these two actors.



Picture 20. Top 10 IMDb Rating Distribution -Robert De Niro vs. Leonardo DiCaprio

This density plot compares the IMDb rating distributions of movies starring Robert De Niro (blue) and Leonardo DiCaprio (red).

Observations:

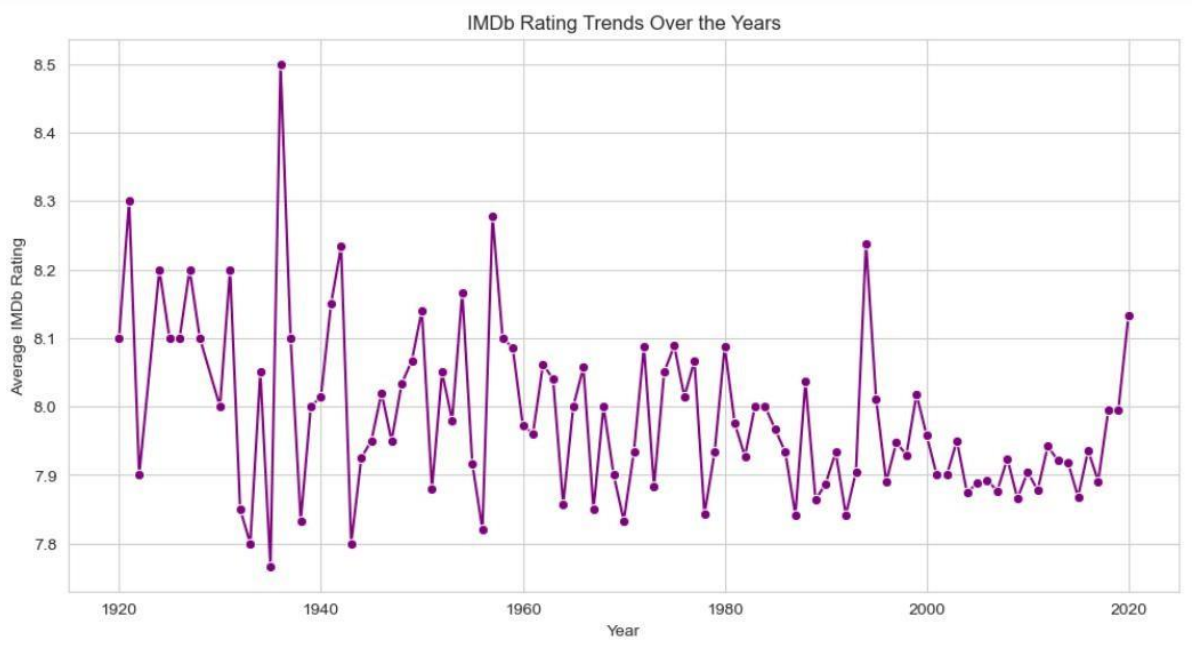
- Both actors have a similar rating distribution, mostly ranging from **7.0 to 9.5**.
- De Niro's movies peak around **7.8**, whereas DiCaprio's movies peak slightly higher, around **8.0**.
- DiCaprio's movies appear to have a more concentrated distribution around **8.0**, while De Niro's ratings show a wider spread.
- Both actors have high-rated films, with some going beyond **9.0**.

This suggests that both actors consistently appear in highly rated films, with DiCaprio's movies tending to have slightly higher ratings on average.

10.18 IMDb Rating Trends Over the Years

```
#If movies are getting better or worse by the time
# Convert Released_Year to numeric
df['Released_Year'] = pd.to_numeric(df['Released_Year'], errors='coerce')

# Average IMDb Rating per year
yearly_ratings = df.groupby('Released_Year')['IMDB_Rating'].mean()

plt.figure(figsize=(12, 6))
sns.lineplot(x=yearly_ratings.index, y=yearly_ratings.values, marker='o', color='purple')
plt.title('IMDb Rating Trends Over the Years')
plt.xlabel('Year')
plt.ylabel('Average IMDb Rating')
plt.grid(True)
plt.show()
```

This code tracks how IMDb ratings have changed over the years. It converts release years to numbers, finds the average rating per year, and then plots a **line graph** to show trends over time.



Picture 21. IMDb Rating Over the Years

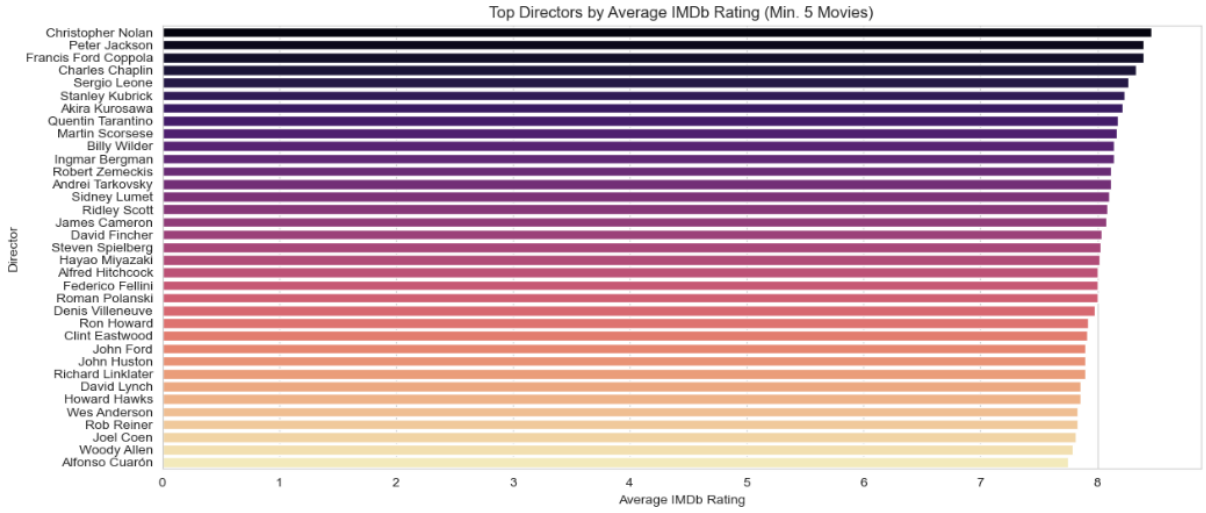
This graph shows how the average IMDb rating of movies has changed over time. Each point represents the average rating of movies released in a specific year. There are many ups and downs, suggesting that some years had stronger movies while others had weaker ones. The ratings for older movies, especially before the 1950s, vary a lot, possibly due to

fewer movies being made or shifts in movie quality. From the 1980s to the 2010s, movie ratings appear more stable, without extreme fluctuations. A sharp increase in ratings after 2015 suggests that recent movies might be receiving better ratings, possibly due to modern trends or fan-based voting.

10.19 Top Directors by Average IMDb Rating

```
#WHO IS CONSTANTLY MAKING GREAT MOVIES
# Get directors with at least 5 movies
top_directors = df['Director'].value_counts()[df['Director'].value_counts() >= 5].index
director_ratings = df[df['Director'].isin(top_directors)].groupby('Director')['IMDB_Rating'].mean().sort_values(ascending=False)

plt.figure(figsize=(14, 6))
sns.barplot(y=director_ratings.index, x=director_ratings.values, palette='magma')
plt.title('Top Directors by Average IMDb Rating (Min. 5 Movies)')
plt.xlabel('Average IMDb Rating')
plt.ylabel('Director')
plt.show()
```

This code finds the best directors based on average IMDb ratings, but only if they've directed at least five movies. First, it filters out directors with fewer than five movies. Then, it calculates the average IMDb rating for each qualifying director. Finally, it creates a horizontal bar chart, ranking directors from highest to lowest average IMDb rating.



Picture 22. Top Directors by Average IMDb Rating

This bar chart shows the top directors ranked by their average IMDb rating, considering only those who have directed at least five movies. Christopher Nolan has the highest average rating, followed closely by Peter Jackson and Francis Ford Coppola. The color gradient from dark to light represents the ranking, with darker bars indicating higher ratings.

10.20 Filtering Nolan Movies with Leonardo DiCaprio as a Star

```
# Filter movies where Christopher Nolan is the director and Leonardo DiCaprio is one of the stars
leo_nolan_movies = df[(df['Director'] == 'Christopher Nolan') &
                      ((df['Star1'] == 'Leonardo DiCaprio') |
                       (df['Star2'] == 'Leonardo DiCaprio') |
                       (df['Star3'] == 'Leonardo DiCaprio') |
                       (df['Star4'] == 'Leonardo DiCaprio'))]

# Display results
print(leo_nolan_movies[['Series_Title', 'Released_Year', 'IMDB_Rating']])
```

```
  Series_Title  Released_Year  IMDB_Rating
8    Inception         2010.0          8.8
```

This code filters the dataset to find movies directed by Christopher Nolan that also star Leonardo DiCaprio. It checks if the Director column is "Christopher Nolan" and if DiCaprio appears in any of the Star1, Star2, Star3, or Star4 columns. The result shows "Inception" (2010) with an IMDb rating of 8.8, which is the only movie in the dataset meeting these criteria.

10.21 Filtering Nolan Movies with Leonardo Al Pacino as a Star

```
# Filter movies where Christopher Nolan is the director and Al Pacino is one of the stars
pacino_nolan_movies = df[(df['Director'] == 'Christopher Nolan') &
                         ((df['Star1'] == 'Al Pacino') |
                          (df['Star2'] == 'Al Pacino') |
                          (df['Star3'] == 'Al Pacino') |
                          (df['Star4'] == 'Al Pacino'))]

# Display results
print(pacino_nolan_movies[['Series_Title', 'Released_Year', 'IMDB_Rating']])
```

```
Empty DataFrame
Columns: [Series_Title, Released_Year, IMDB_Rating]
Index: []
```

This code attempts to find movies directed by Christopher Nolan where Al Pacino is one of the stars. It filters the dataset by checking if the Director is "Christopher Nolan" and if Al Pacino appears in any of the Star1, Star2, Star3, or Star4 columns. However, the output is an empty DataFrame, meaning that no movies in the dataset match both conditions. This suggests that Al Pacino has never starred in a Christopher Nolan-directed movie.

10.22 Movie with the Longest runtime

```
# Convert 'Runtime' to numeric (removing ' min')
df['Runtime'] = df['Runtime'].str.replace(' min', '').astype(float)

# Find the movie with the longest runtime
longest_movie = df.nlargest(1, 'Runtime')[['Series_Title', 'Runtime', 'Released_Year', 'IMDB_Rating']]

# Display result
print(longest_movie)
```

```
           Series_Title  Runtime  Released_Year  IMDB_Rating
140  Gangs of Wasseypur    321.0         2012.0          8.2
```

This code converts the 'Runtime' column into a numeric format by removing the " min" text and changing the data type to float. Then, it finds the movie with the longest runtime using the nlargest(1, 'Runtime') function. The output shows that "Gangs of Wasseypur" (2012) is the longest movie in the dataset, with a runtime of 321 minutes and an IMDb rating of 8.2.

10.23 Movie with the Shortest runtime

```
# Find the movie with the shortest runtime
shortest_movie = df.nsmallest(1, 'Runtime')[['Series_Title', 'Runtime', 'Released_Year', 'IMDB_Rating']]

# Display result
print(shortest_movie)
```

```
     Series_Title  Runtime  Released_Year  IMDB_Rating
194  Sherlock Jr.     45.0         1924.0          8.2
```

This code identifies the movie with the shortest runtime by using the nsmallest(1, 'Runtime') function. The output shows that "Sherlock Jr." (1924) has the shortest runtime in the dataset, lasting 45 minutes, with an IMDb rating of 8.2.

10.24 Movie with the Highest Number of Votes

```
# Find the movie with the highest number of votes
most_voted_movie = df.nlargest(1, 'No_of_Votes')[['Series_Title', 'No_of_Votes', 'Released_Year', 'IMDB_Rating']]

# Display result
print(most_voted_movie)
```

```
              Series_Title  No_of_Votes  Released_Year  IMDB_Rating
0  The Shawshank Redemption      2343110         1994.0          9.3
```

This code finds the movie with the highest number of votes using nlargest(1, 'No_of_Votes'). The output reveals that "The Shawshank Redemption" (1994) holds the most votes, with 2,343,110 votes and an impressive IMDb rating of 9.3.

10.25 Movie with the Lowest IMDb rating

```
# Find the movie with the lowest IMDb rating
lowest_rated_movie = df.nsmallest(1, 'IMDB_Rating')[['Series_Title', 'IMDB_Rating', 'Released_Year', 'No_of_Votes']]

# Display result
print(lowest_rated_movie)
```

```
    Series_Title  IMDB_Rating  Released_Year  No_of_Votes
877  Dark Waters          7.6         2019.0        60408
```

This code identifies the movie with the lowest IMDb rating using nsmallest(1, 'IMDB_Rating'). The output shows that "Dark Waters" (2019) has the lowest rating in the dataset, with an IMDb rating of 7.6 and 60,408 votes.

10.26 Movie with the Longest overview

```
# Create a new column for overview length
df['Overview_Length'] = df['Overview'].astype(str).apply(len)

# Find the movie with the longest overview
longest_overview_movie = df.nlargest(1, 'Overview_Length')[['Series_Title', 'Overview', 'Overview_Length', 'IMDB_Rating']]

# Display result
print(longest_overview_movie)
```

```
       Series_Title                                       Overview  \
935  Jeux d'enfants  As adults, best friends Julien and Sophie cont...

     Overview_Length  IMDB_Rating
935              313          7.6
```

This code creates a new column Overview_Length, which stores the character length of each movie's overview. Then, it finds the movie with the longest overview using nlargest(1, 'Overview_Length'). The result shows that "Jeux d'enfants" (also known as *Love Me If You Dare*) has the longest overview with 313 characters and an IMDb rating of 7.6.

10.27 Movie with the Shortest overview

```
# Ensure 'Overview' is a string and calculate its length
df['Overview_Length'] = df['Overview'].astype(str).apply(len)

# Find the movie with the shortest overview
shortest_overview_movie = df.nsmallest(1, 'Overview_Length')[['Series_Title', 'Overview', 'Overview_Length', 'IMDB_Rating']]

# Display result
print(shortest_overview_movie)
```

```
        Series_Title                              Overview  \
831  The Last Emperor  The story of the final Emperor of China.

     Overview_Length  IMDB_Rating
831               40          7.7
```

This code ensures that the Overview column is treated as a string and calculates its length, storing it in a new column called Overview_Length. It then identifies the movie with the shortest overview using nsmallest(1, 'Overview_Length'). The result shows that "The Last Emperor" has the shortest overview with only 40 characters and an IMDb rating of 7.7.

10.28 Filtering Movies Where Leonardo DiCaprio Is One of the Stars

```
# Filter movies where Leonardo DiCaprio is in any of the star columns
leo_movies = df[(df['Star1'] == 'Leonardo DiCaprio') |
                (df['Star2'] == 'Leonardo DiCaprio') |
                (df['Star3'] == 'Leonardo DiCaprio') |
                (df['Star4'] == 'Leonardo DiCaprio')]

# Select relevant columns
leo_movies = leo_movies[['Series_Title', 'Released_Year', 'IMDB_Rating', 'No_of_Votes']]

# Sort by release year
leo_movies = leo_movies.sort_values(by='Released_Year')

# Display all Leonardo DiCaprio movies
print(leo_movies)
```

	Series_Title	Released_Year	IMDB_Rating	No_of_Votes
658	What's Eating Gilbert Grape	1993.0	7.8	215034
652	Titanic	1997.0	7.8	1046089
243	Catch Me If You Can	2002.0	8.1	832846
37	The Departed	2006.0	8.5	1189773
361	Blood Diamond	2006.0	8.0	499439
8	Inception	2010.0	8.8	2067042
145	Shutter Island	2010.0	8.2	1129894
62	Django Unchained	2012.0	8.4	1357682
147	The Wolf of Wall Street	2013.0	8.2	1187498
343	The Revenant	2015.0	8.0	705589
879	Once Upon a Time... in Hollywood	2019.0	7.6	551309

This code filters a dataset (df) to extract all movies in which Leonardo DiCaprio is listed as one of the stars. It checks four columns (Star1, Star2, Star3, Star4) to find his name. After filtering, it selects only the relevant columns: movie title, release year, IMDb rating, and number of votes. The movies are then sorted by release year in ascending order and displayed.

Explanation of the Output

The output is a list of Leonardo DiCaprio's movies, ordered by release year. Each row contains:

10.28.1 Series_Title: The movie's name.

10.28.2 Released_Year: The year it was released.

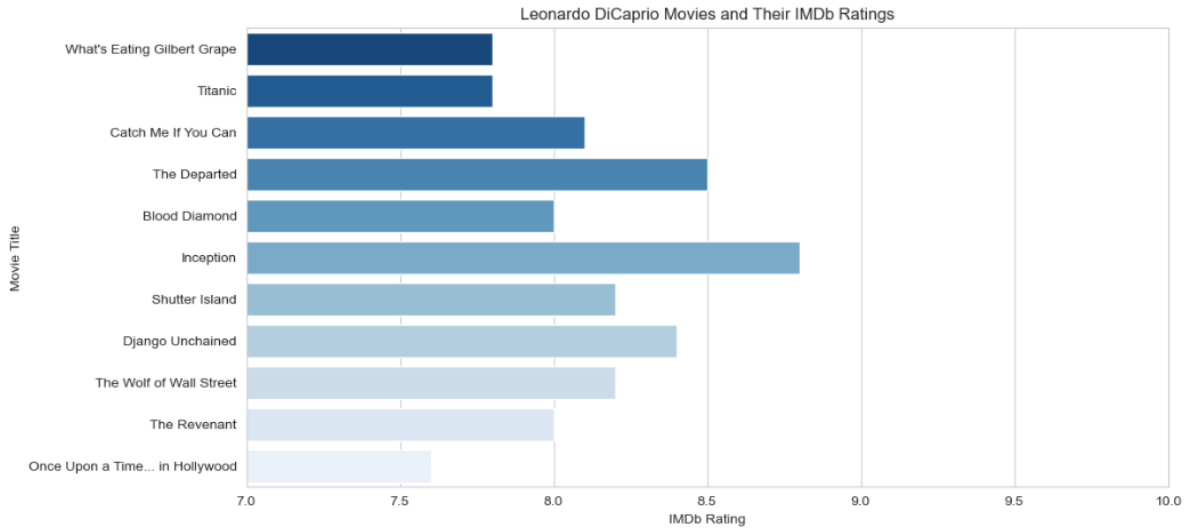
10.28.3 IMDB_Rating: The IMDb rating of the movie.

10.28.4 No_of_Votes: The total number of votes the movie received on IMDb.

10.29 Leonardo DiCaprio Movies and Their IMDb Ratings

```
plt.figure(figsize=(12, 6))
sns.barplot(y=leo_movies['Series_Title'], x=leo_movies['IMDB_Rating'], palette='Blues_r')

plt.title('Leonardo DiCaprio Movies and Their IMDb Ratings')
plt.xlabel('IMDb Rating')
plt.ylabel('Movie Title')
plt.xlim(7, 10)  # Adjust the x-axis for better visualization

plt.show()
```

This code creates a horizontal bar chart to visualize Leonardo DiCaprio's movies and their IMDb ratings using seaborn and matplotlib. It first sets the figure size to make the chart wider. Then, it creates a bar plot where the y-axis represents the movie titles, and the x-axis represents the IMDb ratings, using a reversed blue color gradient. The chart title is set as "Leonardo DiCaprio Movies and Their IMDb Ratings," and the x-axis and y-axis are labeled accordingly. The x-axis range is limited to values between 7 and 10 to improve visualization. Finally, the plot is displayed. The output is a bar chart where each bar represents a movie, showing its IMDb rating with different shades of blue.



Picture 23. Leonardo DiCaoprio Movies And Their IMDb Ratings

The graph shows Leonardo DiCaprio's movies and their IMDb ratings. The x-axis represents ratings from 7.0 to 10.0, while the y-axis lists movie titles. Longer bars mean higher ratings. *Inception* has the highest rating, while *What's Eating Gilbert Grape* and *Titanic* have the lowest. Most of his movies are rated above 7.5, showing he has starred in well-received films.

10.30 Filtering Movies Where Richard Gere Is One of the Stars

```
# Filter movies where Richard Gere is in any of the star columns
gere_movies = df[(df['Star1'] == 'Richard Gere') |
                 (df['Star2'] == 'Richard Gere') |
                 (df['Star3'] == 'Richard Gere') |
                 (df['Star4'] == 'Richard Gere')]

# Select relevant columns
gere_movies = gere_movies[['Series_Title', 'Released_Year', 'IMDB_Rating', 'No_of_Votes']]

# Sort by release year
gere_movies = gere_movies.sort_values(by='Released_Year')

# Display all Richard Gere movies
print(gere_movies)
```

```
           Series_Title  Released_Year  IMDB_Rating  No_of_Votes
690       Days of Heaven         1978.0          7.8        52852
811          Primal Fear         1996.0          7.7       189716
228  Hachi: A Dog's Tale         2009.0          8.1       253575
```

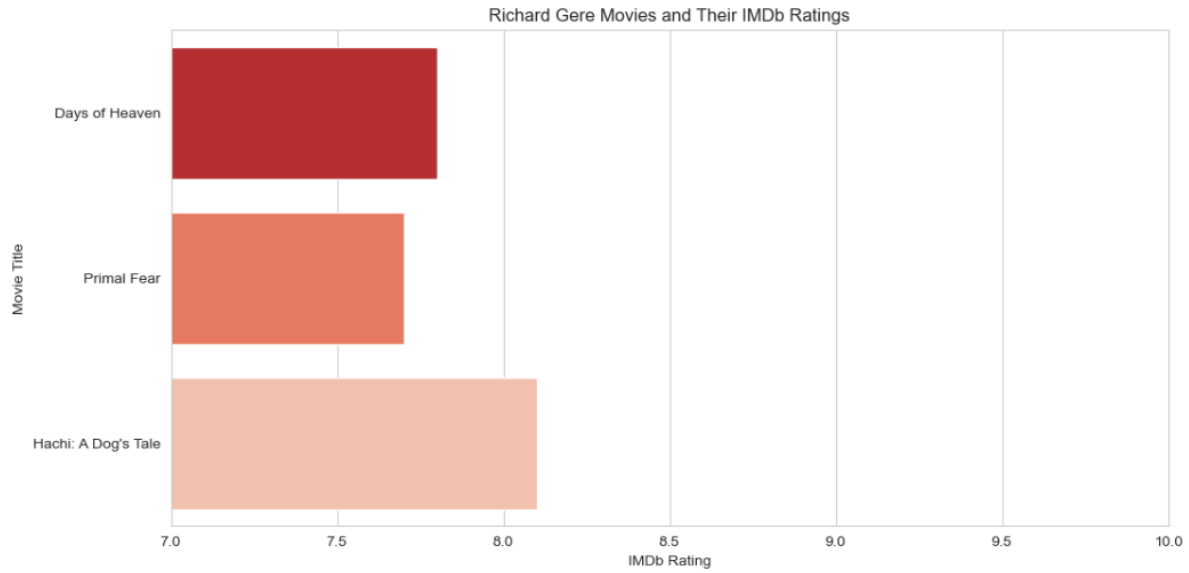
This code filters movies where Richard Gere is listed as one of the four stars. It then selects key columns: title, release year, IMDb rating, and number of votes. The movies are sorted by release year. Finally, it prints the results, showing three Richard Gere movies:

1. *Days of Heaven* (1978) – IMDb 7.8
2. *Primal Fear* (1996) – IMDb 7.7
3. *Hachi: A Dog's Tale* (2009) – IMDb 8.1

10.31 Richard Gere Movies and Their IMDb Ratings

```
plt.figure(figsize=(12, 6))
sns.barplot(y=gere_movies['Series_Title'], x=gere_movies['IMDB_Rating'], palette='Reds_r')

plt.title('Richard Gere Movies and Their IMDb Ratings')
plt.xlabel('IMDb Rating')
plt.ylabel('Movie Title')
plt.xlim(7, 10)

plt.show()
```

This code creates a horizontal bar chart showing Richard Gere's movies and their IMDb ratings. The figure size is set to 12x6 inches. The movie titles are on the Y-axis, and IMDb ratings are on the X-axis. A red color palette is used for the bars. The X-axis is limited between 7 and 10. Finally, the chart is displayed.



Picture 24. Richard Gere Movies and Their IMDb Ratings

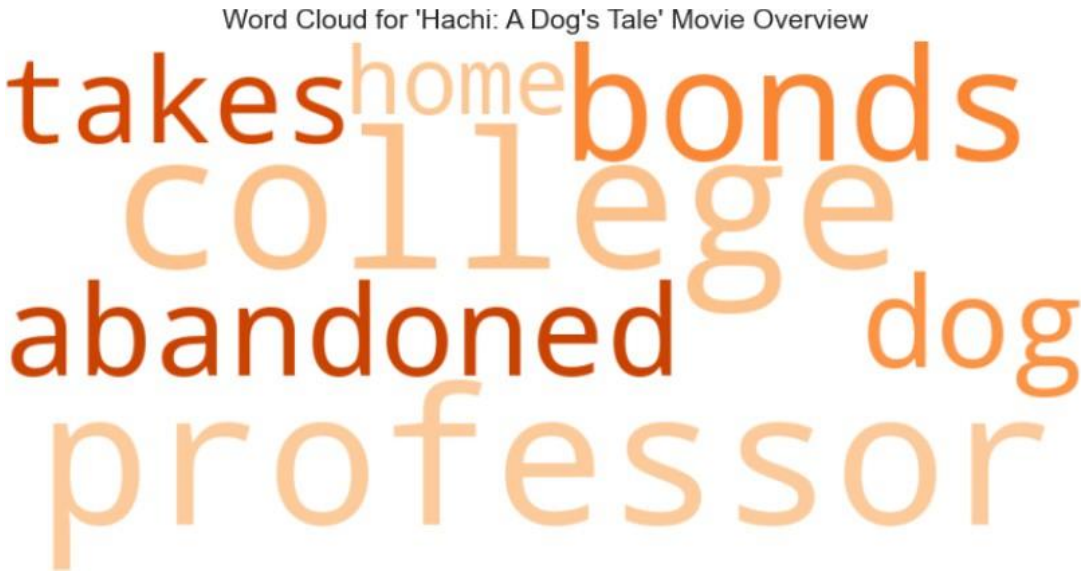
This graph shows Richard Gere's movies and their IMDb ratings using a horizontal bar chart. The movie titles are on the Y-axis, and the IMDb ratings are on the X-axis, ranging from 7 to 10. Darker shades of red represent lower ratings, while lighter shades indicate higher ratings. *Hachi: A Dog's Tale* has the highest rating at 8.1, followed by *Days of Heaven* at 7.8 and *Primal Fear* at 7.7.

10.32 WordCloud for "Hachi" Movie

```
import re
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Filter dataset for "Hachi: A Dog's Tale"
hachi_movie = df[df['Series_Title'].str.contains('Hachi: A Dog', case=False, na=False)]

# Check if the movie exists in the dataset
if not hachi_movie.empty:
    # Extract the overview text
    hachi_overview = hachi_movie['Overview'].values[0]

    # Clean the text: remove punctuation & lowercase all words
    cleaned_text = re.sub(r'[^\w\s]', '', hachi_overview.lower())

    # Create a WordCloud object
    wordcloud = WordCloud(width=800, height=400, background_color='white', colormap='Oranges').generate(cleaned_text)

    # Display the Word Cloud
    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')  # Hide axes
    plt.title("Word Cloud for 'Hachi: A Dog's Tale' Movie Overview", fontsize=14)
    plt.show()
else:
    print("Movie 'Hachi: A Dog's Tale' not found in the dataset.")
```

This code generates a word cloud for the movie *Hachi: A Dog's Tale* based on its overview text. It first filters the dataset to find the movie. If found, it extracts and cleans the text by removing punctuation and converting it to lowercase.

A word cloud is then created with an orange color theme and displayed using Matplotlib. If the movie is not found, it prints a message saying so.



Picture 25. WordCloud For “Hachi” Movie

10.33 Column Names

```
# Display all column names
print(df.columns)
```

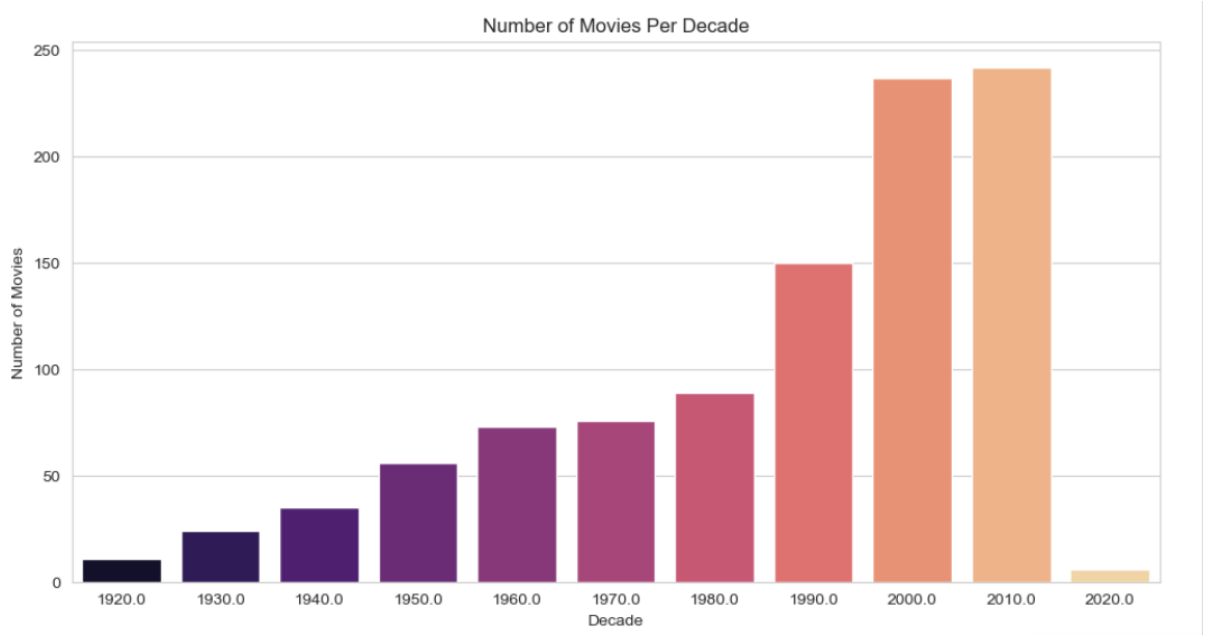
```
Index(['Poster_Link', 'Series_Title', 'Released_Year', 'Certificate',
       'Runtime', 'Genre', 'IMDB_Rating', 'Overview', 'Meta_score', 'Director',
       'Star1', 'Star2', 'Star3', 'Star4', 'No_of_Votes', 'Gross',
       'Overview_Length'],
      dtype='object')
```

This code displays all column names in the DataFrame (df). The dataset includes movie-related details such as title, release year, genre, IMDb rating, overview, director, and main actors. It also contains numerical data like runtime, number of votes, gross earnings, and overview length. The output confirms that these columns exist in the dataset.

10.34 Number of Movies Per Decade

```
df['Decade'] = (df['Released_Year'] // 10) * 10

plt.figure(figsize=(12, 6))
sns.barplot(x=df['Decade'].value_counts().index, y=df['Decade'].value_counts().values, palette='magma')

plt.title('Number of Movies Per Decade')
plt.xlabel('Decade')
plt.ylabel('Number of Movies')

plt.show()
```

This code groups movies by decade and creates a bar chart to show the number of movies released per decade. It calculates the decade by dividing the release year by 10 and multiplying by 10. A bar plot is created using the magma color palette, with decades on the X-axis and movie counts on the Y-axis. The graph is displayed with appropriate labels and a title.



Picture 26. Number of Movies Per Decade

This bar chart shows the number of movies released per decade. The X-axis represents the decades from the 1920s to the 2020s, while the Y-axis represents the number of movies produced in each decade. The trend shows a steady increase in movie production over time, with the highest number of movies released in the 2000s and 2010s. The 2020s have

significantly fewer movies, likely because the dataset is incomplete, or the decade is still ongoing. The magma color palette visually emphasizes the increasing trend in movie production.

10.35 Filtering Christopher Nolan Movies

```
# Filter movies directed by Christopher Nolan
nolan_movies = df[df['Director'] == 'Christopher Nolan']

# Select relevant columns
nolan_movies = nolan_movies[['Series_Title', 'Released_Year', 'IMDB_Rating']]

# Sort by release year
nolan_movies = nolan_movies.sort_values(by='Released_Year')

# Display the movies
print(nolan_movies)
```

```
              Series_Title  Released_Year  IMDB_Rating
69                 Memento         2000.0          8.4
155          Batman Begins         2005.0          8.2
36             The Prestige         2006.0          8.5
2           The Dark Knight         2008.0          9.0
8                 Inception         2010.0          8.8
63   The Dark Knight Rises         2012.0          8.4
21             Interstellar         2014.0          8.6
573                Dunkirk         2017.0          7.8
```

Filtering Movies: It selects only rows where the Director column is "Christopher Nolan".

Selecting Columns: It keeps only three columns:

10.35.1 Series_Title (Movie title)

10.35.2 Released_Year (Year of release)

10.35.3 IMDB_Rating (Movie rating)

Sorting by Release Year: The movies are sorted in ascending order by their release year.

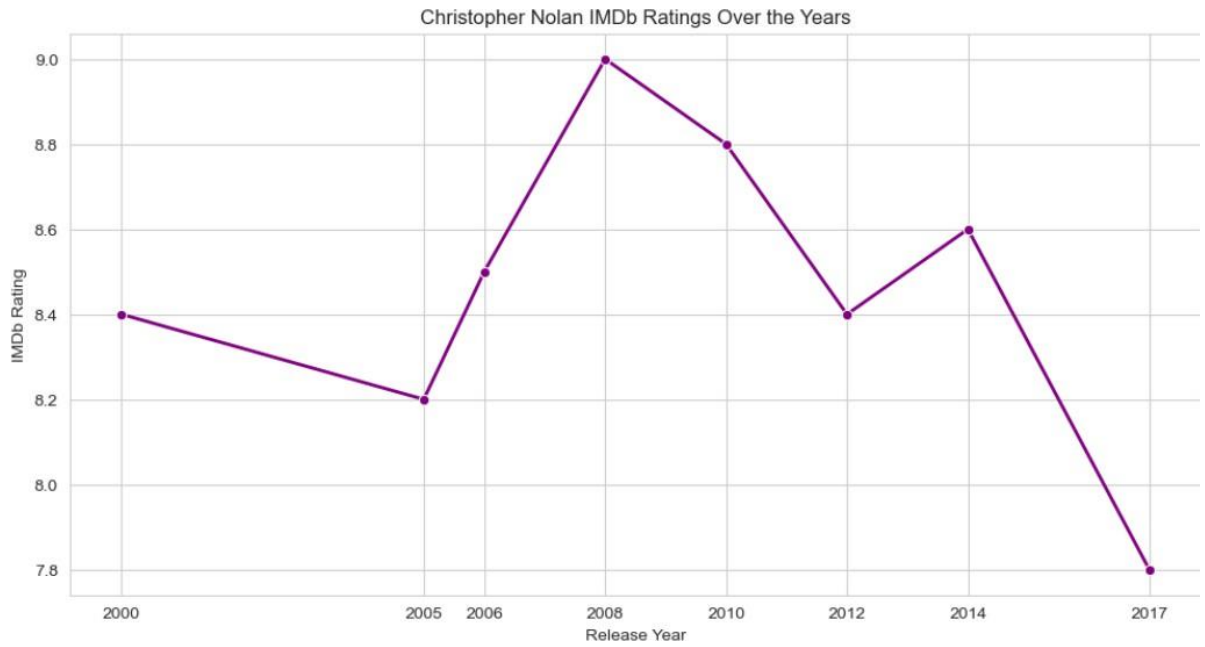
Displaying the Results: It prints the list of Christopher Nolan's movies, showing their release year and IMDb ratings. From the output, movies like Memento (2000), The Dark Knight (2008), and Interstellar (2014) are included, with The Dark Knight having the highest IMDb rating (9.0).

10.36 Christopher Nolan IMDb Ratings Over the Years

```
plt.figure(figsize=(12, 6))
sns.lineplot(x=nolan_movies['Released_Year'], y=nolan_movies['IMDB_Rating'], marker='o', color='purple', linewidth=2)

plt.xticks(nolan_movies['Released_Year'])  # Show every release year
plt.title('Christopher Nolan IMDb Ratings Over the Years')
plt.xlabel('Release Year')
plt.ylabel('IMDb Rating')
plt.grid(True)

plt.show()
```

This code creates a line plot to visualize Christopher Nolan's IMDb ratings over the years. It sets the figure size, then uses sns.lineplot() to plot release years on the x-axis and IMDb ratings on the y-axis, with purple lines and circular markers.

The x-axis labels are set to display every release year. A title, x-label, and y-label are added, and a grid is enabled for better readability. Finally, plt.show() displays the plot.



Picture 27. Christopher Nolan IMDb Ratings Over the Years

This line plot shows the IMDb ratings of Christopher Nolan's movies over the years. The x-axis represents the release years of his films, while the y-axis represents their IMDb ratings. Each point on the line marks a movie's rating, connected by a purple line to show the trend. The ratings fluctuate, peaking at 9.0 for The Dark Knight (2008) and showing a decline in later years, with Dunkirk (2017) having the lowest rating in the dataset.

10.37 Christopher Nolan's First Movie By Release Year

```
# Find Nolan's first movie by release year
first_nolan_movie = df[df['Director'] == 'Christopher Nolan'].nsmallest(1, 'Released_Year')

# Display result
print(first_nolan_movie[['Series_Title', 'Released_Year']])
```

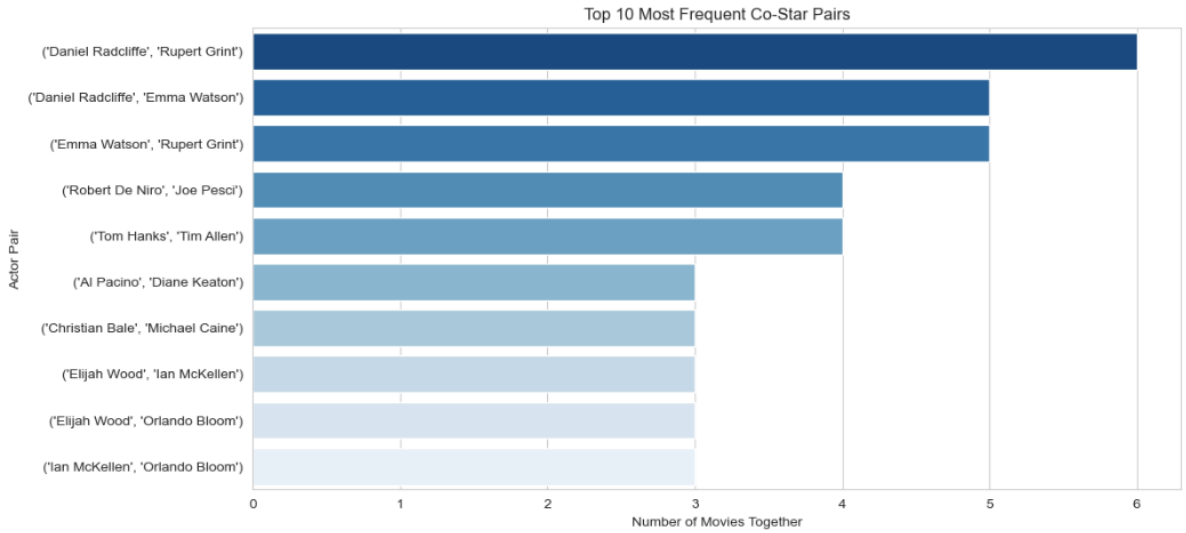
```
   Series_Title  Released_Year
69      Memento         2000.0
```

This code finds Christopher Nolan's first movie by release year. It filters the dataset for movies directed by him and selects the one with the earliest release year using nsmallest(1, 'Released_Year'). The output shows that Memento (released in 2000) is his first movie in the dataset.

```
from itertools import combinations
from collections import Counter

# Create actor pairs from each movie
actor_pairs = []
for i, row in df.iterrows():
    actors = [row['Star1'], row['Star2'], row['Star3'], row['Star4']]
    pairs = combinations(actors, 2)  # Create all 2-actor combinations
    actor_pairs.extend(pairs)

# Count occurrences
pair_counts = Counter(actor_pairs)
top_pairs = pair_counts.most_common(10)

# Convert to DataFrame
top_pairs_df = pd.DataFrame(top_pairs, columns=['Pair', 'Count'])

# Plot
plt.figure(figsize=(12, 6))
sns.barplot(y=top_pairs_df['Pair'].astype(str), x=top_pairs_df['Count'], palette='Blues_r')

plt.title('Top 10 Most Frequent Co-Star Pairs')
plt.xlabel('Number of Movies Together')
plt.ylabel('Actor Pair')

plt.show()
```

This code analyzes the most frequent co-star pairs in movies. It extracts the main actors from each movie and generates all possible two-actor combinations. It then counts how often each pair appears across different movies. The top 10 most frequent actor pairs are stored in a DataFrame and visualized using a bar plot, where the x-axis represents the number of movies they appeared in together, and the y-axis lists the actor pairs.

10.38 Top 10 Most Frequent Co-Star Pairs



Picture 28. Top 10 Most Frequent Co-Star Pairs

This bar chart shows the top 10 most frequent co-star pairs in movies.

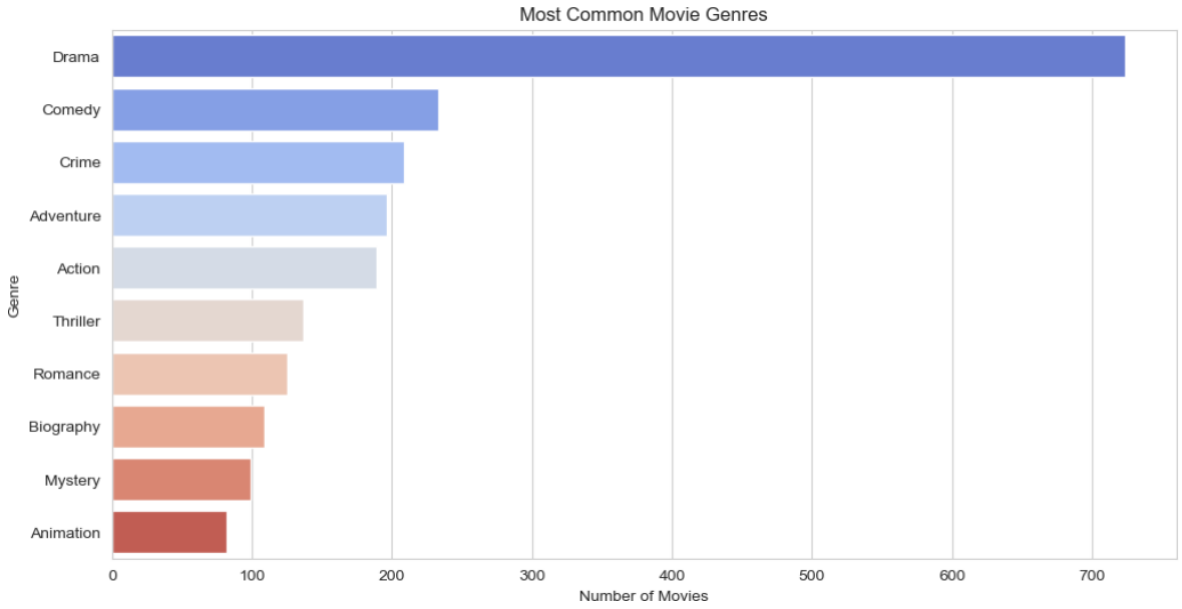
The x-axis represents the number of movies in which each pair appeared together, while the y-axis lists the actor pairs. Daniel Radcliffe and Rupert Grint have worked together the most, appearing in six movies. Other frequent co-star pairs include Emma Watson with both Daniel Radcliffe and Rupert Grint, as well as classic actor duos like Robert De Niro and Joe Pesci. The chart visually highlights the strongest on-screen partnerships in the dataset.

10.39 Most Common Movie Genres

```
df_exploded = df.assign(Genre=df['Genre'].str.split(', ')).explode('Genre')

top_genres = df_exploded['Genre'].value_counts().head(10)

plt.figure(figsize=(12, 6))
sns.barplot(y=top_genres.index, x=top_genres.values, palette='coolwarm')

plt.title('Most Common Movie Genres')
plt.xlabel('Number of Movies')
plt.ylabel('Genre')

plt.show()
```

This code analyzes the most common movie genres. It first splits the 'Genre' column where multiple genres are listed and then expands them into separate rows. After that, it counts the occurrences of each genre and selects the top 10 most frequent ones. A horizontal bar plot is then created, where the x-axis represents the number of movies, and the y-axis

represents the genres. The visualization helps identify which genres are most popular in the dataset.



Picture 29. Most Common Movie Genres

This bar chart shows the most common movie genres in the dataset. The x-axis represents the number of movies, while the y-axis lists different genres. Drama is the most frequent genre, appearing in the highest number of movies, followed by Comedy and Crime. The chart helps visualize which genres are most popular.

10.40 The Highest-rated genre and The Lowest-rated genre

```
: # Explode genres so each genre has its own row
df_exploded = df.assign(Genre=df['Genre'].str.split(', ')).explode('Genre')

# Calculate the average IMDb rating for each genre
genre_avg_rating = df_exploded.groupby('Genre')['IMDB_Rating'].mean().sort_values(ascending=False)

# Get the best and worst rated genres
best_genre = genre_avg_rating.idxmax()
worst_genre = genre_avg_rating.idxmin()

print(f"The highest-rated genre is: {best_genre} with an average rating of {genre_avg_rating.max():.2f}")
print(f"The lowest-rated genre is: {worst_genre} with an average rating of {genre_avg_rating.min():.2f}")
```

```
The highest-rated genre is: War with an average rating of 8.01
The lowest-rated genre is: Horror with an average rating of 7.89
```

This code analyzes the average IMDb ratings of different movie genres. First, it ensures that each genre has its own row. Then, it calculates the average rating for each genre and

sorts them in descending order. The best-rated genre is determined as "War" with an average rating of 8.01, while the lowest-rated genre is "Horror" with an average rating of 7.89.

10.41 Unique Genres

```
# Explode genres so each genre has its own row
df_exploded = df.assign(Genre=df['Genre'].str.split(', ')).explode('Genre')

# Get unique genres
unique_genres = df_exploded['Genre'].unique()

# Display all genres
print("All unique genres in the dataset:")
print(unique_genres)
```

```
All unique genres in the dataset:
['Drama' 'Crime' 'Action' 'Adventure' 'Biography' 'History' 'Sci-Fi'
 'Romance' 'Western' 'Fantasy' 'Comedy' 'Thriller' 'Animation' 'Family'
 'War' 'Mystery' 'Music' 'Horror' 'Musical' 'Film-Noir' 'Sport']
```

This code extracts all unique movie genres from a dataset. It first ensures that each genre appears in its own row by splitting and expanding them. Then, it finds all distinct genres in the dataset and prints them. The output lists various genres such as Drama, Crime, Action, Adventure, Sci-Fi, and many more.

10.42 Filtering Movies Where Leonardo DiCaprio Is One of the Stars

In this code, I counted the number of movies in each genre by exploding the genre column (in case movies have multiple genres). The output shows that Drama is the most common genre with 724 movies, followed by Comedy (233 movies) and Crime (209 movies). The least frequent genres are Musical (17 movies) and Sport (19 movies). This provides insights into the distribution of movie genres in the dataset.

```
# Count number of movies per genre
genre_counts = df_exploded['Genre'].value_counts()

# Display
print("\nNumber of movies in each genre:")
print(genre_counts)
```

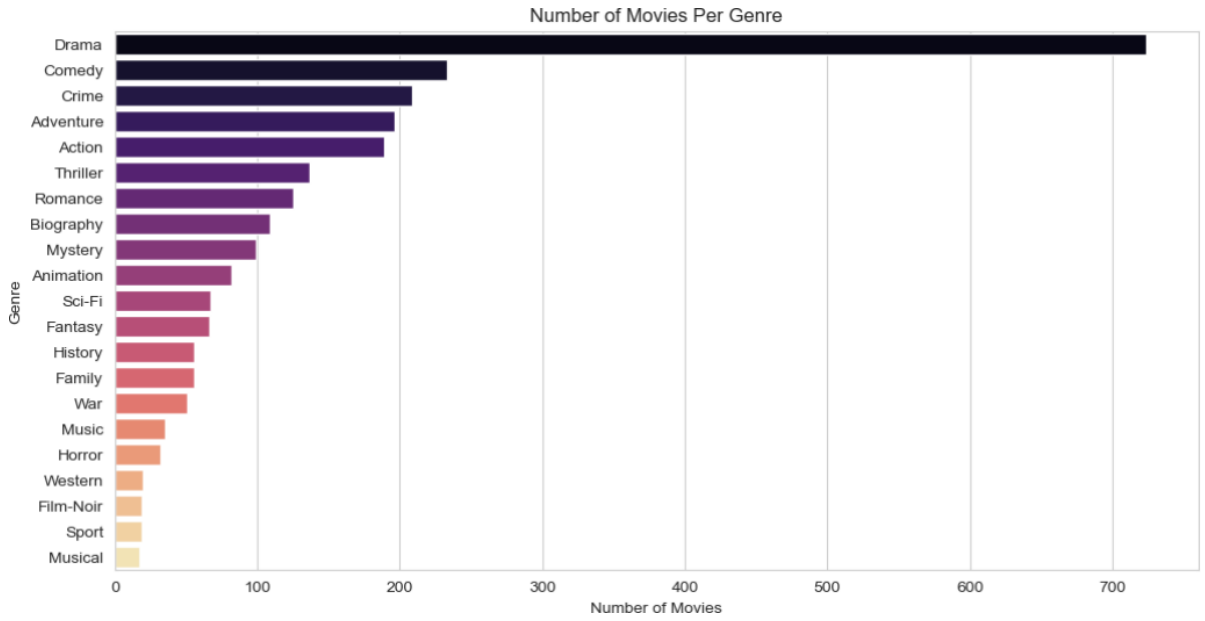
```
Number of movies in each genre:
Genre
Drama        724
Comedy       233
Crime        209
Adventure    196
Action       189
Thriller     137
Romance      125
Biography    109
Mystery       99
Animation     82
Sci-Fi        67
Fantasy       66
History       56
Family        56
War           51
Music         35
Horror        32
Western       20
Film-Noir     19
Sport         19
Musical       17
Name: count, dtype: int64
```

10.43 Number of Movies Per Genre

```
plt.figure(figsize=(12, 6))
sns.barplot(y=genre_counts.index, x=genre_counts.values, palette='magma')

plt.title('Number of Movies Per Genre')
plt.xlabel('Number of Movies')
plt.ylabel('Genre')

plt.show()
```

This code generates a horizontal bar plot using **Seaborn** to visualize the number of movies per genre. The magma color palette is used, which provides a **warm gradient color scheme**. The figure size is set to **(12,6)** for better readability.

The labels and title are clearly defined, making it an effective visualization for understanding the genre distribution in the dataset.



Picture 30. Number of Movies Per Genre

This bar plot effectively visualizes the number of movies per genre, with Drama having the highest count, followed by Comedy and Crime. The magma color palette helps distinguish between different genres, with darker shades representing higher counts and lighter shades indicating fewer movies. The horizontal layout improves readability, making it easy to compare genres.

10.44 Filtering Movies Where Leonardo DiCaprio Is In Any of Star Columns

```
# Filter movies where Leonardo DiCaprio is in any of the star columns
leo_movies = df[(df['Star1'] == 'Leonardo DiCaprio') |
                (df['Star2'] == 'Leonardo DiCaprio') |
                (df['Star3'] == 'Leonardo DiCaprio') |
                (df['Star4'] == 'Leonardo DiCaprio')]

# Explode genres so each genre gets its own row
leo_genres = leo_movies.assign(Genre=leo_movies['Genre'].str.split(', ')).explode('Genre')

# Get unique genres
leo_unique_genres = leo_genres['Genre'].unique()

# Display result
print("Genres Leonardo DiCaprio has acted in:")
print(leo_unique_genres)
```

```
Genres Leonardo DiCaprio has acted in:
['Action' 'Adventure' 'Sci-Fi' 'Crime' 'Drama' 'Thriller' 'Western'
 'Mystery' 'Biography' 'Romance' 'Comedy']
```

This code snippet extracts all movies featuring Leonardo DiCaprio by checking if his name appears in any of the four "Star" columns. Then, it processes the genres associated with these movies to determine the unique ones he has acted in. Leonardo DiCaprio has acted in 10 different genres, including Action, Adventure, Drama, Thriller, Crime, Sci-Fi, and Comedy.

10.45 Count of Movies Per Genre – Leonardo DiCaprio

```
# Count movies per genre for Leonardo DiCaprio
leo_genre_counts = leo_genres['Genre'].value_counts()

# Display
print("\nNumber of Leonardo DiCaprio movies in each genre:")
print(leo_genre_counts)
```

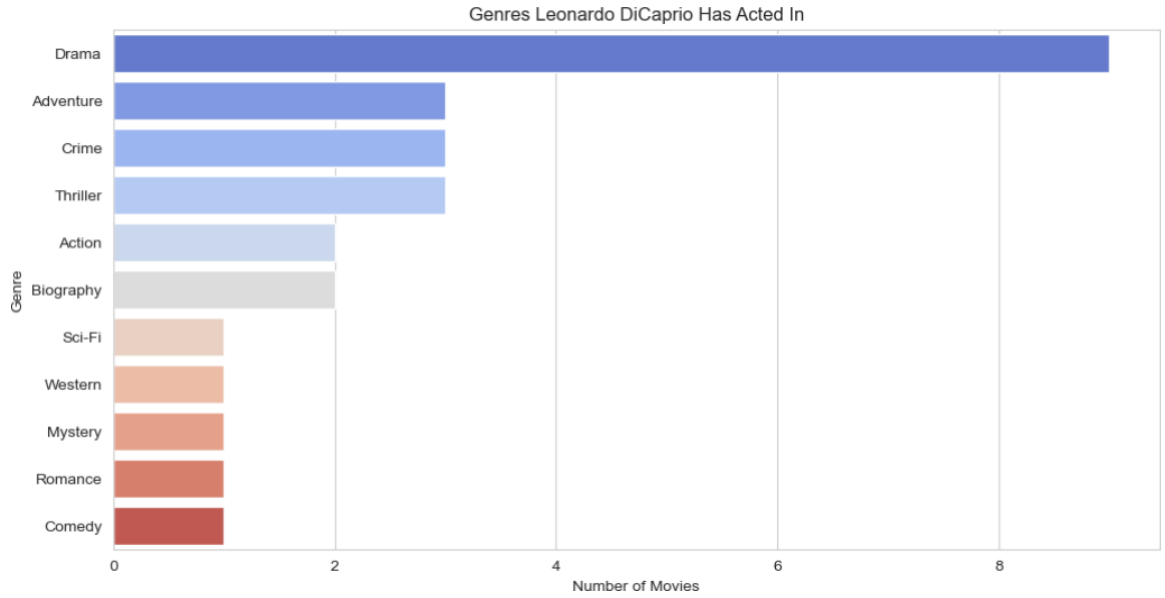
```
Number of Leonardo DiCaprio movies in each genre:
Genre
Drama        9
Adventure    3
Crime        3
Thriller     3
Action       2
Biography    2
Sci-Fi       1
Western      1
Mystery      1
Romance      1
Comedy       1
Name: count, dtype: int64
```

The code finds all movies with Leonardo DiCaprio, splits their genres into separate rows, and lists the unique genres he has acted in. Then, it counts how many movies he has done in each genre. The results show he acted in 9 drama movies, 3 adventure movies, and others.

10.46 Genres Leonardo DiCaprio Has Acted In

```
plt.figure(figsize=(12, 6))
sns.barplot(y=leo_genre_counts.index, x=leo_genre_counts.values, palette='coolwarm')

plt.title('Genres Leonardo DiCaprio Has Acted In')
plt.xlabel('Number of Movies')
plt.ylabel('Genre')

plt.show()
```

This code creates a bar chart showing the number of movies Leonardo DiCaprio has acted in for each genre. It uses the "coolwarm" color palette and labels the axes properly.



Picture 31. Genres Leonardo DiCaprio Has Acted In

This bar chart visualizes the number of movies Leonardo DiCaprio has acted in for each genre. Drama is the most frequent genre, followed by Adventure, Crime, and Thriller. The "coolwarm" color palette differentiates genres with varying shades.

10.47 Filtering Movies Where Richard Gere Is In Any of Star Columns

```
# Filter movies where Richard Gere is in any of the star columns
gere_movies = df[(df['Star1'] == 'Richard Gere') |
                 (df['Star2'] == 'Richard Gere') |
                 (df['Star3'] == 'Richard Gere') |
                 (df['Star4'] == 'Richard Gere')]

# Explode genres so each genre gets its own row
gere_genres = gere_movies.assign(Genre=gere_movies['Genre'].str.split(', ')).explode('Genre')

# Get unique genres
gere_unique_genres = gere_genres['Genre'].unique()

# Display result
print("Genres Richard Gere has acted in:")
print(gere_unique_genres)
```

```
Genres Richard Gere has acted in:
['Biography' 'Drama' 'Family' 'Romance' 'Crime' 'Mystery']
```

This code filters movies where Richard Gere appears as one of the stars, extracts the genres he has acted in, and lists the unique genres. The result shows that Richard Gere has acted in Biography, Drama, Family, Romance, Crime, and Mystery genres.

10.48 Genres Richard Gere Has Acted In

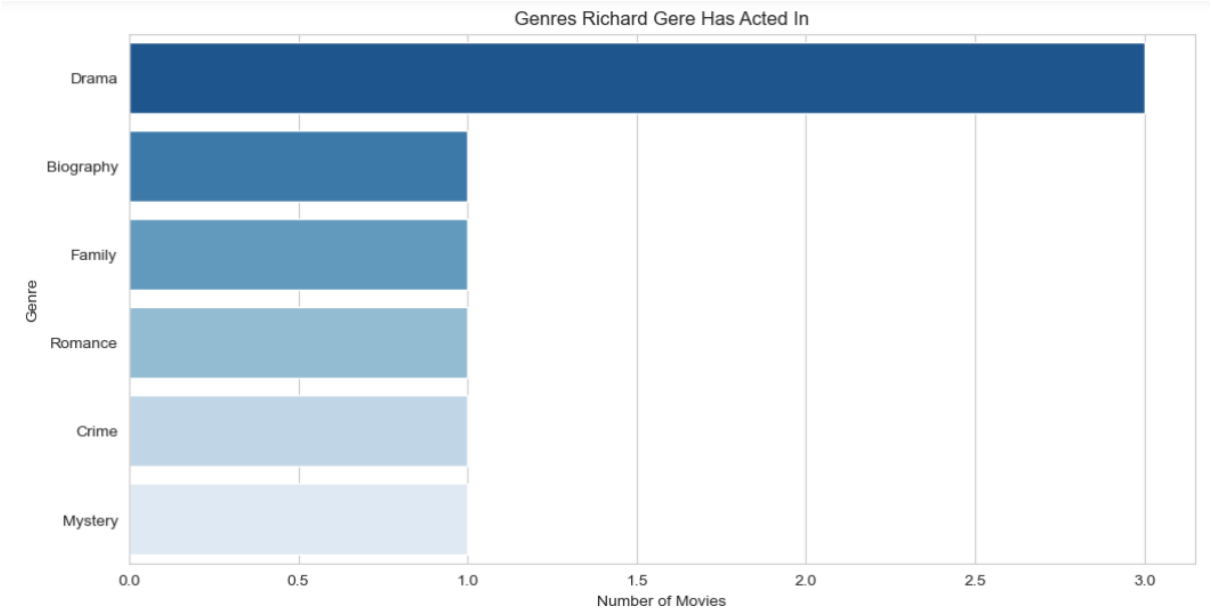
```
# Count movies per genre for Richard Gere
gere_genre_counts = gere_genres['Genre'].value_counts()

# Display
print("\nNumber of Richard Gere movies in each genre:")
print(gere_genre_counts)
```

```
Number of Richard Gere movies in each genre:
Genre
Drama        3
Biography    1
Family       1
Romance      1
Crime        1
Mystery      1
Name: count, dtype: int64
```

This code counts the number of movies Richard Gere has acted in for each genre. The result shows that he has acted in 3 Drama movies, and 1 movie each in Biography, Family, Romance, Crime, and Mystery genres.

```
plt.figure(figsize=(12, 6))
sns.barplot(y=gere_genre_counts.index, x=gere_genre_counts.values, palette='Blues_r')

plt.title('Genres Richard Gere Has Acted In')
plt.xlabel('Number of Movies')
plt.ylabel('Genre')

plt.show()
```

I created a bar chart that visually represents the number of movies Richard Gere has acted in for each genre. It uses the Seaborn library to generate the plot, with the genres listed on the y-axis and the number of movies on the x-axis. The palette 'Blues_r' is applied, which provides a reversed blue color gradient, making the bars darker for higher values and lighter for lower values. The figure size is set to (12,6), ensuring clarity. The title 'Genres Richard Gere Has Acted In' is displayed, along with labeled axes for better readability.



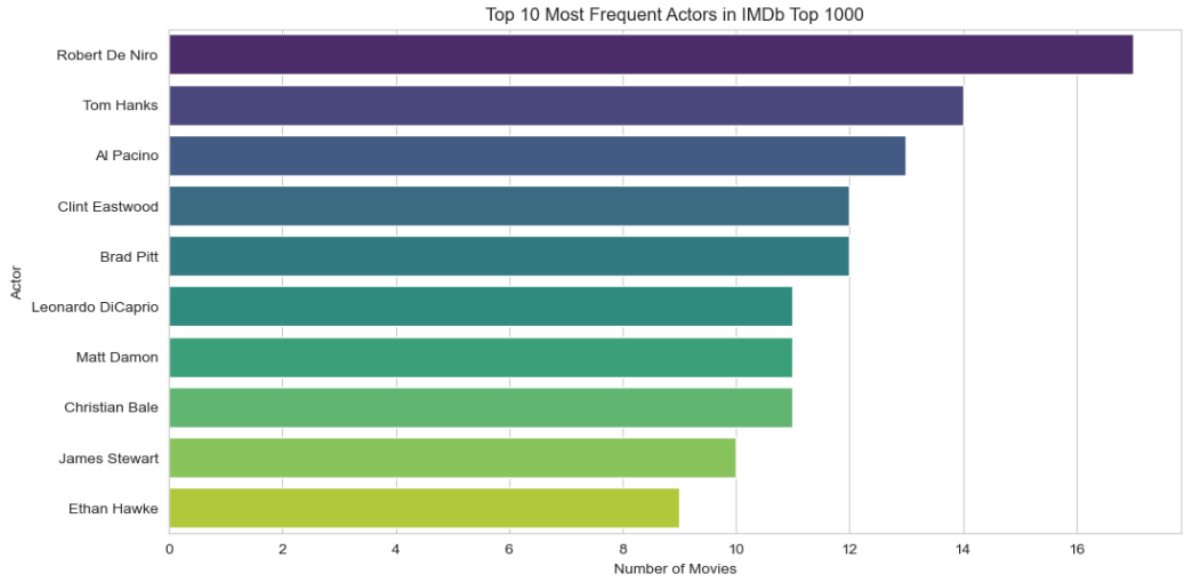
Picture 32. Genres Richard Gere Has Acted In

This bar chart visually represents the number of movies Richard Gere has acted in, categorized by genre. The y-axis lists different genres, while the x-axis represents the number of movies in each genre. The reversed blue color palette emphasizes the distribution, with darker shades indicating higher counts. Drama stands out as the most frequent genre, with three movies, while the other genres have only one each.

10.49 Top 10 Most Frequent Actors in IMDb Top 1000

```
# Count occurrences of each actor in the dataset
actor_counts = all_actors.value_counts().head(10)

# Plot
plt.figure(figsize=(12, 6))
sns.barplot(y=actor_counts.index, x=actor_counts.values, palette='viridis')

plt.title('Top 10 Most Frequent Actors in IMDb Top 1000')
plt.xlabel('Number of Movies')
plt.ylabel('Actor')

plt.show()
```

This code analyzes the top 10 most frequent actors in the IMDb Top 1000 movies by counting how many times each actor appears, selecting the top 10, and visualizing the results using a bar chart with actor names on the y-axis and their movie count on the x-axis, styled with the "viridis" color scheme for better visuals.



Picture 33. Top 10 Most Frequent Actors

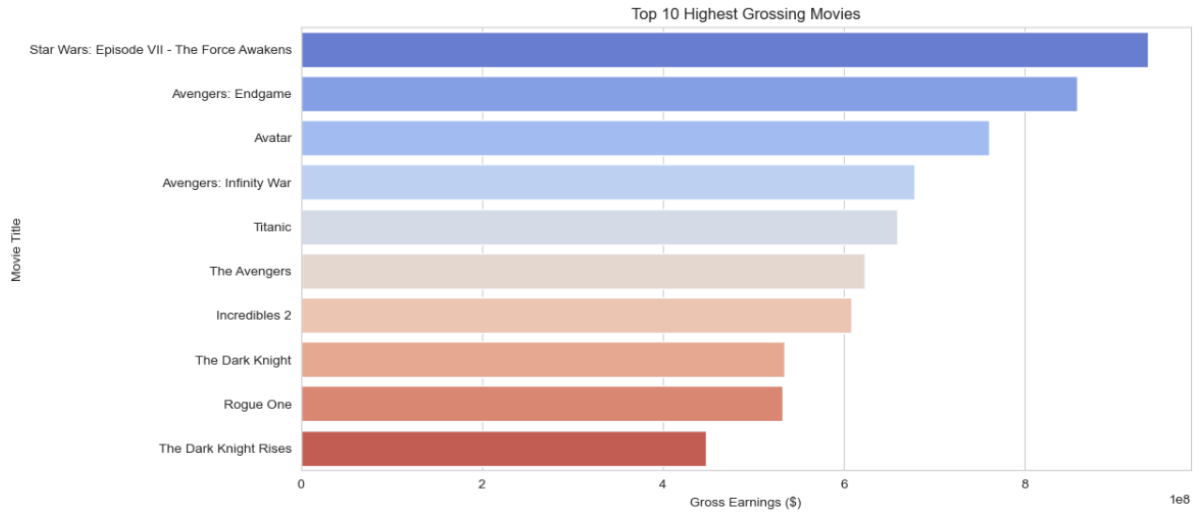
This bar chart shows the top 10 most frequent actors in the IMDb Top 1000 movies. The x-axis represents the number of movies each actor has appeared in, while the y-axis lists the actors' names. The chart uses the "viridis" color palette, where darker shades indicate higher frequencies.

Robert De Niro appears the most, followed by Tom Hanks and Al Pacino. Other notable actors in the list include Clint Eastwood, Brad Pitt, and Leonardo DiCaprio. The visualization highlights actors who have consistently played roles in top-rated films.

10.50 Top 10 Highest Grossing Movies

```
df['Gross'] = df['Gross'].str.replace(',', '').astype(float)

top_grossing = df.nlargest(10, 'Gross')[['Series_Title', 'Gross']]

plt.figure(figsize=(12, 6))
sns.barplot(y=top_grossing['Series_Title'], x=top_grossing['Gross'], palette='coolwarm')

plt.title('Top 10 Highest Grossing Movies')
plt.xlabel('Gross Earnings ($)')
plt.ylabel('Movie Title')

plt.show()
```

This code processes and visualizes the top 10 highest-grossing movies by converting the "Gross" column into a numeric format, selecting the highest earners, and creating a bar chart with movie titles on the y-axis and earnings on the x-axis using the "coolwarm" color scheme, while labeling the axes and adding a title for clarity.



Picture 34. Top 10 Highest Grossing Movies

This bar chart visualizes the top 10 highest-grossing movies, with "Star Wars: Episode VII - The Force Awakens" at the top, followed by "Avengers: Endgame" and "Avatar," displaying their earnings in descending order using a gradient color scheme from cool to warm for emphasis. The x-axis represents gross earnings in dollars, while the y-axis lists the movie titles.

10.51 Movies Released In The 1990s

```
# Filter movies released in the 1990s
movies_90s = df[(df['Released_Year'] >= 1990) & (df['Released_Year'] <= 1999)]

# Find the movie with the most votes
most_popular_90s = movies_90s.nlargest(1, 'No_of_Votes')[['Series_Title', 'Released_Year', 'IMDB_Rating', 'No_of_Votes'

# Display result
print("Most popular movie from the 1990s:")
print(most_popular_90s)
```

```
Most popular movie from the 1990s:
              Series_Title  Released_Year  IMDB_Rating  No_of_Votes
0  The Shawshank Redemption         1994.0          9.3      2343110
```

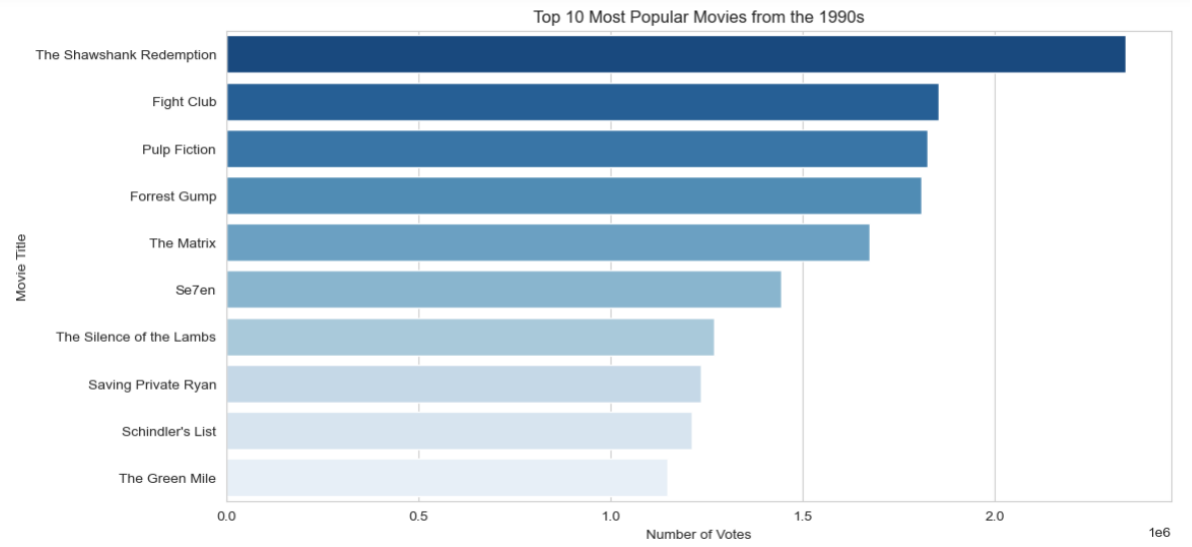
10.52 Top 10 Most Voted Movies From The 1990s

This code filters movies released in the 1990s and selects the one with the highest number of votes. It finds that "The Shawshank Redemption" (1994) is the most popular movie of the decade, with an IMDb rating of 9.3 and 2,343,110 votes. The result is printed as a small table displaying the title, release year, rating, and vote count.

```
# Get the top 10 most voted movies from the 1990s
top_90s_movies = movies_90s.nlargest(10, 'No_of_Votes')[['Series_Title', 'No_of_Votes']]

plt.figure(figsize=(12, 6))
sns.barplot(y=top_90s_movies['Series_Title'], x=top_90s_movies['No_of_Votes'], palette='Blues_r')

plt.title('Top 10 Most Popular Movies from the 1990s')
plt.xlabel('Number of Votes')
plt.ylabel('Movie Title')

plt.show()
```

This code selects the top 10 most voted movies from the 1990s and visualizes them using a horizontal bar chart. It first filters the dataset to find the movies with the highest number of votes, then plots them with movie titles on the y-axis and the number of votes on the x-

axis. The chart uses the "Blues_r" color palette and has a title, x-label, and y-label for clarity.



Picture 35. Top 10 Most Popular Movies from the 1990s

This bar chart visualizes the top 10 most popular movies from the 1990s based on the number of votes. "The Shawshank Redemption" received the highest number of votes, followed by "Fight Club" and "Pulp Fiction." The movies are ranked in descending order, with the number of votes displayed on the x-axis and movie titles on the y-axis. The color scheme (Blues_r) represents the intensity of popularity.

10.53 Filtering Movies Where Tom Cruise Is In Any of Star Columns

```
# Filter movies where Tom Cruise is in any of the star columns
tom_cruise_movies = df[(df['Star1'] == 'Tom Cruise') |
                       (df['Star2'] == 'Tom Cruise') |
                       (df['Star3'] == 'Tom Cruise') |
                       (df['Star4'] == 'Tom Cruise')]

# Filter only movies from the 1990s
tom_cruise_90s = tom_cruise_movies[(tom_cruise_movies['Released_Year'] >= 1990) &
                                   (tom_cruise_movies['Released_Year'] <= 1999)]

# Find the highest-rated Tom Cruise movie in the 90s
best_tom_cruise_90s = tom_cruise_90s.nlargest(1, 'IMDB_Rating')[['Series_Title', 'Released_Year', 'IMDB_Rating']]

# Display result
print("Tom Cruise's highest-rated movie in the 1990s:")
print(best_tom_cruise_90s)
```

```
Tom Cruise's highest-rated movie in the 1990s:
    Series_Title  Released_Year  IMDB_Rating
383     Magnolia         1999.0          8.0
```

This code filters movies where Tom Cruise is listed as one of the stars and further narrows them down to those released in the 1990s. It then selects the highest-rated Tom Cruise movie from that decade based on IMDb ratings. The result shows that "Magnolia" (1999) was his highest-rated movie in the 1990s, with an IMDb rating of 8.0.

10.54 Longest Movie Titles

```
df['Title_Length'] = df['Series_Title'].apply(len)

# Get the longest movie titles
longest_titles = df.nlargest(10, 'Title_Length')[['Series_Title', 'Title_Length']]

# Display
print("Movies with the longest titles:")
print(longest_titles)
```

```
Movies with the longest titles:
                                    Series_Title  Title_Length
78   Dr. Strangelove or: How I Learned to Stop Worr...            68
246  Shin seiki Evangelion Gekijô-ban: Air/Magokoro...            58
376  Pirates of the Caribbean: The Curse of the Bla...            54
10   The Lord of the Rings: The Fellowship of the Ring            49
610    Das weiße Band - Eine deutsche Kindergeschichte            47
733    Birdman or (The Unexpected Virtue of Ignorance)            47
16      Star Wars: Episode V - The Empire Strikes Back            46
977     The Naked Gun: From the Files of Police Squad!            46
5        The Lord of the Rings: The Return of the King            45
226       Harry Potter and the Deathly Hallows: Part 2            44
```

This code calculates the length of each movie title and selects the 10 movies with the longest titles. The longest title is "Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb" with 68 characters. Other long titles include entries from "Pirates of the Caribbean," "Lord of the Rings," and "Harry Potter" franchises.

10.55 Movie With The Shortest Title

```
# Create a new column for title length
df['Title_Length'] = df['Series_Title'].apply(len)

# Find the movie with the shortest title
shortest_title_movie = df.nsmallest(1, 'Title_Length')[['Series_Title', 'Released_Year', 'IMDB_Rating', 'Title_Length']]

# Display result
print("Movie with the shortest title:")
print(shortest_title_movie)
```

```
Movie with the shortest title:
    Series_Title  Released_Year  IMDB_Rating  Title_Length
146           Up         2009.0          8.2             2
```

This code finds the movie with the shortest title in the dataset. The result shows that "Up" (2009) has the shortest title, with only 2 characters. It also has an IMDB rating of 8.2

11.MDb Rating Trend Analysis Over Time

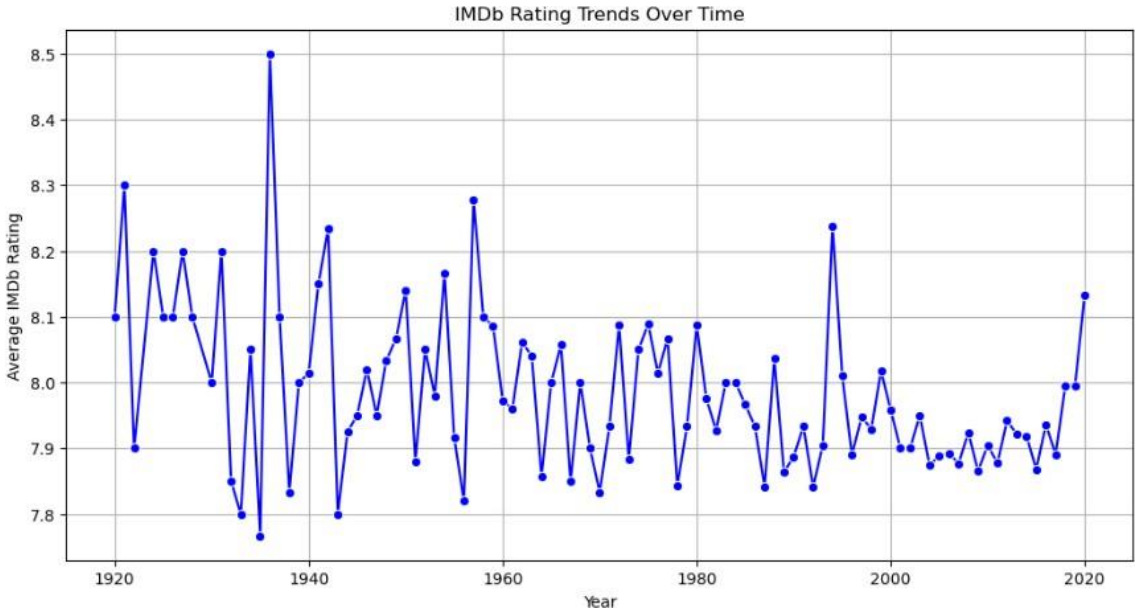
```
#MILESTONE 3 - APPLYING MACHINE LEARNING
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.seasonal import seasonal_decompose


file_path = "C:\\Users\\Korisnik\\Downloads\\archive\\imdb_top_1000.csv"
df = pd.read_csv(file_path)

# Convert 'Released_Year' to numeric
df['Released_Year'] = pd.to_numeric(df['Released_Year'], errors='coerce').dropna().astype(int)

# Compute the average IMDb rating per year
ratings_over_time = df.groupby('Released_Year')['IMDB_Rating'].mean()

# Plot IMDb rating trends over time
plt.figure(figsize=(12, 6))
sns.lineplot(x=ratings_over_time.index, y=ratings_over_time.values, marker='o', color='b')

plt.title('IMDb Rating Trends Over Time')
plt.xlabel('Year')
plt.ylabel('Average IMDb Rating')
plt.grid(True)
plt.show()
```

This code is focused on analyzing how the average IMDb ratings of movies have changed over time. It starts by loading a dataset of top 1000 movies, which contains information like the movie's release year and IMDb rating.

First, the code ensures that the Released_Year column is in numeric format, removing any invalid or missing values, and then it converts this column into integer values representing the actual release years. Next, the code calculates the average IMDb rating for each year by grouping the dataset based on the Released_Year and computing the mean rating for all movies released in that year. This gives a sense of how movie ratings have fluctuated annually. Finally, the code creates a line plot to visualize this trend over time.

The plot uses the release years on the x-axis and the average IMDb ratings on the y-axis, with markers indicating individual data points.

The line plot clearly shows whether movie ratings have generally increased, decreased, or remained stable across different years, helping to identify any significant trends or patterns in movie ratings over time. The grid lines make the plot easier to read and understand.



Picture 36. IMDb Rating Trends Over Time

The graph represents IMDb rating trends over time, showing the average IMDb ratings of movies released each year. The x-axis represents the years, ranging from the early 1900s to 2020, while the y-axis shows the average IMDb ratings, fluctuating between approximately 7.8 and 8.5. The blue line with data points indicates variations in movie ratings across different years. Peaks and dips suggest periods where highly rated or lower-rated movies were more frequent.

Notably, there is a significant spike around the 1940s and early 2000s, indicating exceptional movies during those periods. Recent years show a slight increase in ratings compared to the late 20th century.

12.Time Series Decomposition of IMDb Ratings

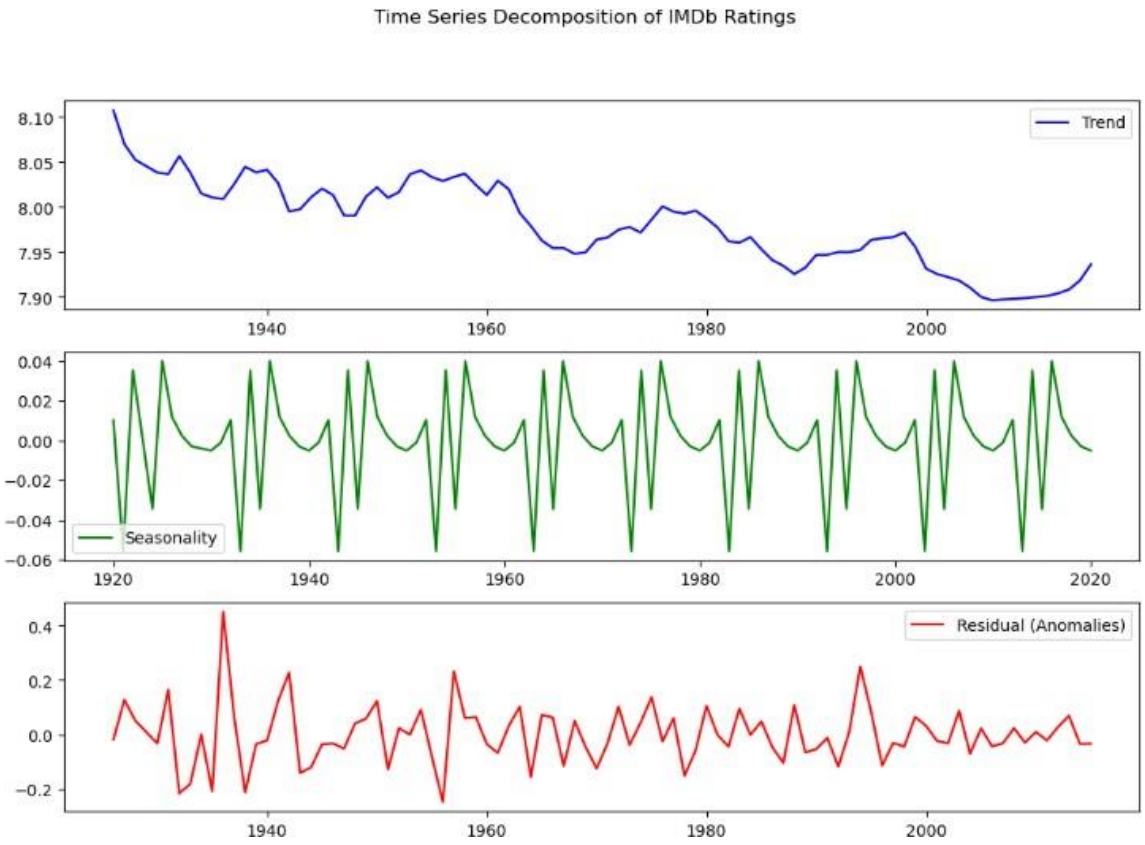
```
# Decompose the time series data
decomposition = seasonal_decompose(ratings_over_time, model='additive', period=10)

# Plot the trend, seasonality, and residuals
plt.figure(figsize=(12, 8))

plt.subplot(3, 1, 1)
plt.plot(decomposition.trend, label='Trend', color='blue')
plt.legend()

plt.subplot(3, 1, 2)
plt.plot(decomposition.seasonal, label='Seasonality', color='green')
plt.legend()

plt.subplot(3, 1, 3)
plt.plot(decomposition.resid, label='Residual (Anomalies)', color='red')
plt.legend()

plt.suptitle('Time Series Decomposition of IMDb Ratings')
plt.show()
```

This code performs a time series decomposition on IMDb rating data over time to analyze its components: trend, seasonality, and residuals. The decomposition uses an additive model with a period of 10 years.

Trend Component – The first plot (blue line) represents the long-term pattern of IMDb ratings, showing whether ratings generally increase, decrease, or remain stable over time.

Seasonality Component – The second plot (green line) captures repetitive patterns or cycles in IMDb ratings that occur periodically, helping to identify any consistent trends over the years.

Residual Component (Anomalies) – The third plot (red line) shows the remaining variation in the data after removing the trend and seasonality. This part highlights irregular fluctuations, such as unexpected peaks or drops in ratings.



Picture 37. Time Series Decomposition of IMDb Ratings

This graph illustrates the time series decomposition of IMDb ratings over time, breaking the data into trend, seasonality, and residual components.

The first plot shows the trend (blue line), which represents the long-term movement of IMDb ratings. It indicates a gradual decline from the 1920s to the early 2000s, followed by a slight upward movement in recent years.

The second plot displays the seasonality (green line), which highlights repeating patterns within a 10-year period, suggesting that IMDb ratings experience cyclical fluctuations, likely influenced by changes in filmmaking trends or audience preferences.

The third plot represents the residuals (red line), showing irregular variations that remain after removing the trend and seasonality. Notably, there are significant fluctuations in the earlier years, indicating that some periods had unusually high or low ratings that did not fit the overall pattern. This decomposition helps in understanding how IMDb ratings have evolved, identifying both long-term shifts and periodic trends while also revealing anomalies in movie ratings.

13. Preprocessing for Genre Prediction

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score

# Load dataset
file_path = "C:\\Users\\Korisnik\\Downloads\\archive\\imdb_top_1000.csv"
df = pd.read_csv(file_path)

# Convert 'Runtime' to numerical format
df['Runtime'] = df['Runtime'].str.replace(' min', '').astype(float)

# Select features and target variable (Genre)
df['Genre'] = df['Genre'].str.split(', ').str[0]  # Take only the first genre for simplicity

# Encode categorical target variable (Genre)
label_encoder = LabelEncoder()
df['Genre_Label'] = label_encoder.fit_transform(df['Genre'])

# Select features for prediction
X = df[['IMDB_Rating', 'Runtime', 'No_of_Votes']].dropna()
y = df.loc[X.index, 'Genre_Label']

# Split data into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

First, I loaded the dataset from a CSV file and cleaned the 'Runtime' column by removing the "min" suffix and converting it into a numerical format. Since each movie can belong to multiple genres, I simplified the dataset by selecting only the first listed genre.

To enable machine learning algorithms to process categorical data, I applied Label Encoding to convert genre names into numerical labels.

For feature selection, I chose IMDb Rating, Runtime, and Number of Votes, ensuring that any missing values were removed for data consistency.

Finally, I split the dataset into 80% training data and 20% test data to allow the model to learn from one portion while being evaluated on another, ensuring a robust and reliable performance assessment.

14. Naïve Bayes Classification for Genre Prediction

```
from sklearn.naive_bayes import GaussianNB

# Train the Naïve Bayes model
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)

# Predict on test data
y_pred_nb = nb_model.predict(X_test)

# Evaluate accuracy
nb_accuracy = accuracy_score(y_test, y_pred_nb)
print(f'Naïve Bayes Accuracy: {nb_accuracy:.2f}')
```

```
Naïve Bayes Accuracy: 0.26
```

This code implements a Naïve Bayes classification model using the Gaussian Naïve Bayes (GaussianNB) algorithm to predict movie genres based on IMDb ratings, runtime, and the number of votes. The model is first initialized and trained using the training dataset (X_train and y_train), leveraging Bayes' Theorem under the assumption that features are independent given the class label.

Once trained, the model predicts genres for the test dataset (X_test), generating y_pred_nb, which contains the predicted genre labels. The accuracy of the predictions is then evaluated by comparing y_pred_nb with the actual test labels (y_test) using the accuracy_score function.

The final accuracy, 26%, is displayed, indicating that the model's performance is relatively low. This suggests that the selected features may not be strong predictors of movie genres, or that Naïve Bayes may not be the most suitable algorithm for this dataset.

15. Random Forest Classification for Genre Prediction

```
from sklearn.ensemble import RandomForestClassifier

# Train the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict on test data
y_pred_rf = rf_model.predict(X_test)

# Evaluate accuracy
rf_accuracy = accuracy_score(y_test, y_pred_rf)
print(f'Random Forest Accuracy: {rf_accuracy:.2f}')
```

```
Random Forest Accuracy: 0.24
```

This code implements a Random Forest classification model using the RandomForestClassifier from Scikit-Learn to predict movie genres based on IMDb ratings, runtime, and the number of votes. The model is initialized with 100 decision trees (n_estimators=100) and a fixed random state for reproducibility.

It is then trained using the training dataset (X_train and y_train). Once the training process is complete, the model predicts genres for the test dataset (X_test), generating y_pred_rf, which contains the predicted genre labels. The accuracy of the model is then evaluated by comparing these predictions to the actual test labels (y_test) using the accuracy_score function. The final accuracy, 24%, suggests that the model is not performing well, likely due to the complexity of genre classification and the limited predictive power of the selected features.

16. Comparison of Classification Models

```
print(f"Naïve Bayes Accuracy: {nb_accuracy:.2f}")
print(f"Random Forest Accuracy: {rf_accuracy:.2f}")

if rf_accuracy > nb_accuracy:
    print("✅ Random Forest performed better!")
else:
    print("✅ Naïve Bayes performed better!")
```

```
Naïve Bayes Accuracy: 0.26
Random Forest Accuracy: 0.24
✅ Naïve Bayes performed better!
```

The code compares the accuracy of two machine learning models, Naïve Bayes and Random Forest, by printing their respective accuracy scores and determining which model performed better. It first prints the accuracy of both models with two decimal points. Then,

using an if statement, it checks if the accuracy of Random Forest is greater than that of Naïve Bayes.

If Random Forest performs better, it prints "Random Forest performed better," and if Naïve Bayes performs better, it prints "Naïve Bayes performed better." In this case, Naïve Bayes is shown to have a slightly higher accuracy, so the code outputs that Naïve Bayes performed better.

17. Genre Prediction Using AdaBoost Classifier

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

# Train AdaBoost with a weak Decision Tree classifier
adaboost_model = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1), n_estimators=100, random_state=42)
adaboost_model.fit(X_train, y_train)

# Predict on test data
y_pred_ada = adaboost_model.predict(X_test)

# Evaluate accuracy
ada_accuracy = accuracy_score(y_test, y_pred_ada)
print(f'AdaBoost Accuracy: {ada_accuracy:.2f}')
```

```
AdaBoost Accuracy: 0.21
```

In this code, the AdaBoost algorithm is used in conjunction with a weak Decision Tree classifier (a decision tree with a depth of 1). AdaBoost is an ensemble method that improves the performance of weak classifiers by combining them to create a stronger model.

The weak classifiers are decision trees with limited depth (max_depth=1), meaning each individual tree is very simple and cannot make complex decisions on its own. The model is trained using 100 weak trees (n_estimators=100), and it makes predictions on the test data. However, the result shows that the AdaBoost model has only 21% accuracy on the test set, which is quite low.

This suggests that while AdaBoost is meant to enhance performance, in this case, the weak base learners (shallow decision trees) might not be strong enough to make accurate predictions. This low accuracy could be due to the simplicity of the base model, the need for further hyperparameter tuning, or the nature of the data itself.

18.Clustering Movies with DBSCAN

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

# Load dataset
file_path = "C:\\Users\\Korisnik\\Downloads\\archive\\imdb_top_1000.csv"
df = pd.read_csv(file_path)

# Convert 'Runtime' to numerical format
df['Runtime'] = df['Runtime'].str.replace(' min', '').astype(float)

# Select features for clustering
X = df[['IMDB_Rating', 'Runtime']].dropna()

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply DBSCAN with adjusted parameters
dbscan = DBSCAN(eps=0.3, min_samples=3)  # Adjusted eps and min_samples
clusters = dbscan.fit_predict(X_scaled)

# Assign cluster labels to DataFrame
df.loc[X.index, 'Cluster'] = clusters

# Check if clusters were found
print("Unique Cluster Labels:", np.unique(clusters))
print(df['Cluster'].value_counts())  # See how many movies per cluster

# Visualize clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(x=df.loc[X.index, 'Runtime'], y=df.loc[X.index, 'IMDB_Rating'], hue=df.loc[X.index, 'Cluster'], palette='viridis

plt.title('DBSCAN Clustering of Movies Based on IMDb Rating & Runtime')
plt.xlabel('Runtime (Minutes)')
plt.ylabel('IMDb Rating')
plt.legend(title="Cluster")
plt.show()
```

This code performs DBSCAN clustering on a movie dataset based on two features: IMDb Rating and Runtime. The first step loads the dataset from a CSV file containing information about the top 1000 IMDb movies. The Runtime column is then cleaned by removing the " min" text and converting the values to numerical format for analysis. Afterward, the relevant features (IMDB_Rating and Runtime) are selected for clustering, and missing values are dropped.

To ensure that the features have similar scales, the data is standardized using StandardScaler, which normalizes the values so that they have a mean of 0 and a standard deviation of 1. Then, DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is applied to identify clusters in the data. The algorithm uses two parameters: eps, the maximum distance between points in the same cluster, and min_samples, the minimum number of points required to form a dense region (cluster). In this case, eps=0.3 and min_samples=3 are set to find clusters in the data.

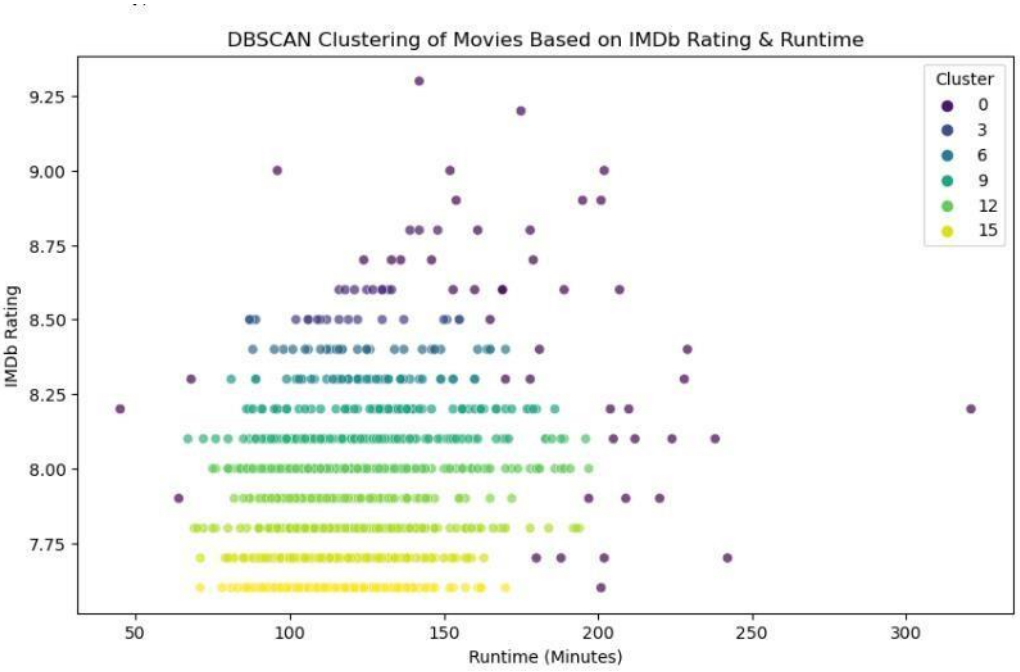
After the clustering is done, the cluster labels are added to the original DataFrame, and the unique cluster labels are printed along with the count of movies in each cluster. Finally, the code visualizes the clusters using a scatter plot, where each point represents a movie, colored by its cluster label. The plot helps to visually understand how movies are grouped based on their IMDb rating and runtime.

```
Unique Cluster Labels: [-1  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16]
Cluster
 15.0    153
 14.0    151
 12.0    141
 16.0    122
 10.0    117
 13.0    102
 9.0      63
-1.0      45
 7.0      37
 5.0      24
 2.0      11
 1.0       9
 11.0      6
 6.0       5
 4.0       4
 3.0       4
 0.0       3
 8.0       3
Name: count, dtype: int64
```

Picture 38. Unique Cluster Labels

The output shows the different groups (clusters) that the DBSCAN algorithm has created based on the IMDb rating and runtime of the movies. The unique cluster labels range from 0 to 16, with each number representing a different group of similar movies. The -1 label represents outliers or movies that couldn't be grouped into any cluster.

The counts show how many movies are in each group: for example, Cluster 15 has the most movies (153), while some clusters have very few, like Cluster 8 with only 3 movies. The algorithm found that most movies can be grouped into clusters, but some don't fit well into any cluster and are marked as outliers.



Picture 39. DBSCAN Clustering of Movies Based on IMDb Rating & Runtime

This graph shows the results of clustering movies based on their IMDb rating and runtime using the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm. Each dot represents a movie, with its runtime (in minutes) on the x-axis and IMDb rating on the y-axis.

The different colors represent different clusters, meaning movies with similar characteristics have been grouped together. Some outliers (points that do not fit into any cluster) may also be present. The clustering helps in identifying patterns in movie data, such as which runtimes tend to have higher or lower ratings.

19. Outlier Detection Using Z-Score

```
import pandas as pd
import numpy as np

# Load dataset
file_path = "C:\\Users\\Korisnik\\Downloads\\archive\\imdb_top_1000.csv"
df = pd.read_csv(file_path)

# Compute Z-scores for IMDb ratings
df['IMDB_Z_Score'] = (df['IMDB_Rating'] - df['IMDB_Rating'].mean()) / df['IMDB_Rating'].std()

# Find movies with extreme ratings (Z-score > 2 or < -2)
outliers = df[(df['IMDB_Z_Score'] > 2) | (df['IMDB_Z_Score'] < -2)]

print("Outlier Movies Based on IMDb Rating:")
print(outliers[['Series_Title', 'IMDB_Rating']])
```

```
Outlier Movies Based on IMDb Rating:
                                        Series_Title  IMDB_Rating
0                           The Shawshank Redemption          9.3
1                                      The Godfather          9.2
2                                    The Dark Knight          9.0
3                             The Godfather: Part II          9.0
4                                        12 Angry Men          9.0
5       The Lord of the Rings: The Return of the King          8.9
6                                        Pulp Fiction          8.9
7                                    Schindler's List          8.9
8                                           Inception          8.8
9                                          Fight Club          8.8
10  The Lord of the Rings: The Fellowship of the Ring          8.8
11                                       Forrest Gump          8.8
12                    Il buono, il brutto, il cattivo          8.8
13             The Lord of the Rings: The Two Towers          8.7
14                                       The Matrix          8.7
15                                       Goodfellas          8.7
16     Star Wars: Episode V - The Empire Strikes Back          8.7
17                   One Flew Over the Cuckoo's Nest          8.7
18                                          Hamilton          8.6
19                                        Gisaengchung          8.6
20                                    Soorarai Pottru          8.6
21                                       Interstellar          8.6
22                                    Cidade de Deus          8.6
23                     Sen to Chihiro no kamikakushi          8.6
24                               Saving Private Ryan          8.6
25                                    The Green Mile          8.6
26                                    La vita è bella          8.6
27                                              Se7en          8.6
28                           The Silence of the Lambs          8.6
29                                          Star Wars          8.6
30                                            Seppuku          8.6
31                               Shichinin no samurai          8.6
32                             It's a Wonderful Life          8.6
```

Picture 40. Outlier Movies Based on IMDb Rating

This code is analyzing IMDb movie ratings from a dataset and identifying outliers based on their Z-scores. First, it loads the dataset using Pandas and calculates the Z-score for each movie's IMDb rating, which measures how far a rating deviates from the average in terms of standard deviations.

Movies with a Z-score greater than 2 or less than -2 are considered outliers, meaning their ratings are significantly higher or lower than the average.

The script then prints a list of movies with exceptionally high ratings. The output shows a list of highly rated films, including The Shawshank Redemption, The Godfather, The Dark Knight, and 12 Angry Men, which have IMDb ratings of 8.6 or higher.

20.High Rating Prediction Using AdaBoost

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score

# Load dataset
file_path = r"C:\Users\Korisnik\Downloads\archive\imdb_top_1000.csv"
df = pd.read_csv(file_path)

# Data preprocessing
# Convert 'Runtime' to numerical value (remove ' min' and convert to int)
df['Runtime'] = df['Runtime'].str.replace(' min', '').astype(float)

# Encode categorical columns
label_encoders = {}
categorical_columns = ['Certificate', 'Genre', 'Director', 'Star1', 'Star2', 'Star3', 'Star4']
for col in categorical_columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col].astype(str))
    label_encoders[col] = le

# Define target variable (IMDB rating > 8.5 as 1, else 0)
df['High_Rating'] = (df['IMDB_Rating'] > 8.5).astype(int)

# Select features and target
features = ['Runtime', 'Certificate', 'Genre', 'Meta_score', 'Director', 'Star1', 'Star2', 'Star3', 'Star4']
X = df[features].fillna(0)  # Fill missing values with 0
y = df['High_Rating']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train AdaBoost model
model = AdaBoostClassifier(n_estimators=50, random_state=42)
model.fit(X_train, y_train)

# Predict and evaluate
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.97
```

This script applies machine learning to predict whether a movie has a high IMDb rating (greater than 8.5) using the AdaBoost algorithm. The dataset, which includes movie details such as runtime, genre, director, and actors, is first loaded into a Pandas DataFrame. The preprocessing stage begins by converting the Runtime column from a string format (e.g., "120 min") to a numerical format, allowing it to be used as a feature in the model. Since machine learning algorithms require numerical inputs, categorical columns such as

Certificate, Genre, Director, and key actors (Star1, Star2, etc.) are converted into numeric values using Label Encoding, which assigns unique integer values to each category.

Next, the target variable, High_Rating, is created. It is a binary classification label where movies with an IMDb rating above 8.5 are assigned a value of 1 (high rating), and those with a rating of 8.5 or below are assigned 0 (not high rating). After defining the target variable, relevant features such as Runtime, Genre, Meta_score, and director/actors are selected for training the model. Missing values in these features are filled with 0 to prevent errors during model training.

The dataset is then split into training and testing sets using an 80-20% ratio, ensuring that 80% of the data is used for training while 20% is reserved for testing. The AdaBoostClassifier, an ensemble learning algorithm that builds multiple weak classifiers to create a strong predictive model, is trained on the training data with 50 estimators (decision trees). Once trained, the model predicts IMDb ratings on the test dataset.

The final accuracy score of 0.97 (97%) indicates that the model is highly effective at classifying whether a movie has a high IMDb rating based on its features.

21.Market Based Analysis Using Apriori Algorithm

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# Sample dataset: Movie genres watched by users
data = {
    'User1': ['Action', 'Comedy', 'Drama'],
    'User2': ['Action', 'Thriller'],
    'User3': ['Comedy', 'Drama', 'Romance'],
    'User4': ['Action', 'Comedy', 'Thriller'],
    'User5': ['Drama', 'Romance']
}

# Convert dataset to a DataFrame
movies = pd.DataFrame.from_dict(data, orient='index')
movies = movies.stack().reset_index(level=1, drop=True).reset_index()
movies.columns = ['User', 'Genre']

# One-hot encode the genres
movies_encoded = movies.pivot_table(index='User', columns='Genre', aggfunc=lambda x: 1, fill_value=0)

# Apply Apriori algorithm
frequent_itemsets = apriori(movies_encoded, min_support=0.4, use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric='lift', min_threshold=1.0)

# Display the first few association rules
print(rules.head())
```

```
  antecedents consequents  antecedent support  consequent support  support  \
0    (Comedy)    (Action)                 0.6                 0.6      0.4
1    (Action)    (Comedy)                 0.6                 0.6      0.4
2  (Thriller)    (Action)                 0.4                 0.6      0.4
3    (Action)  (Thriller)                 0.6                 0.4      0.4
4    (Comedy)     (Drama)                 0.6                 0.6      0.4

   confidence      lift  representativity  leverage  conviction  \
0    0.666667  1.111111               1.0      0.04        1.2
1    0.666667  1.111111               1.0      0.04        1.2
2    1.000000  1.666667               1.0      0.16        inf
3    0.666667  1.666667               1.0      0.16        1.8
4    0.666667  1.111111               1.0      0.04        1.2

   zhangs_metric   jaccard  certainty  kulczynski
0       0.250000  0.500000   0.166667    0.666667
1       0.250000  0.500000   0.166667    0.666667
2       0.666667  0.666667   1.000000    0.833333
3       1.000000  0.666667   0.444444    0.833333
4       0.250000  0.500000   0.166667    0.666667
```

This script applies market basket analysis using the Apriori algorithm to identify patterns in user preferences for movie genres. It begins with a dataset where each user is associated with a list of genres they have watched.

Since Apriori requires binary (one-hot encoded) data, the script transforms the dataset into a format where each row represents a user, and columns represent different genres, with 1s indicating a watched genre and 0s indicating it was not watched. The Apriori algorithm is then applied with a minimum support threshold of 0.4, meaning that only genre combinations that appear in at least 40% of users will be considered frequent.

Once frequent itemsets are found, the script generates association rules to determine how one genre influences the likelihood of watching another. The output includes key metrics such as support (how often both genres appear together), confidence (the probability that watching one genre leads to watching another), and lift (how much more likely a user is to watch a genre if they have already watched another genre compared to random chance).

For example, one rule might suggest that users who watch Comedy are more likely to also watch Action, with a confidence of 66.67% and a lift value of 1.11, indicating a positive relationship. These rules help in understanding genre preferences and can be used for movie recommendations.

22.Clustering with Birch Algorithm

```
import pandas as pd
from sklearn.cluster import Birch
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Load dataset
file_path = r"C:\Users\Korisnik\Downloads\archive\imdb_top_1000.csv"
df = pd.read_csv(file_path)

# Data preprocessing
# Convert 'Runtime' to numerical value (remove ' min' and convert to int)
df['Runtime'] = df['Runtime'].str.replace(' min', '').astype(float)

# Encode categorical columns
label_encoders = {}
categorical_columns = ['Certificate', 'Genre', 'Director', 'Star1', 'Star2', 'Star3', 'Star4']
for col in categorical_columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col].astype(str))
    label_encoders[col] = le

# Select features for clustering
features = ['Runtime', 'Certificate', 'Genre', 'Meta_score', 'Director', 'Star1', 'Star2', 'Star3', 'Star4']
X = df[features].fillna(0)  # Fill missing values with 0

# Normalize data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply Birch clustering
birch = Birch(n_clusters=5)
birch.fit(X_scaled)

# Assign cluster labels
df['Cluster'] = birch.labels_

# Display cluster counts
print(df['Cluster'].value_counts())
```

```
Cluster
2    332
0    237
1    212
3    155
4     64
Name: count, dtype: int64
```

This script applies Birch clustering on the IMDb Top 1000 movies dataset to group similar movies based on their attributes. It begins by loading the dataset and performing data preprocessing. The Runtime column, which contains values in the format "120 min," is converted to a numerical format by removing " min" and converting it to a float. Next, categorical columns such as Certificate, Genre, Director, and lead actors are encoded using LabelEncoder, which converts text-based categories into numerical values so they can be used in clustering.

The script then selects relevant features for clustering, including Runtime, Meta_score, and categorical attributes like Director and Star1. Missing values in these features are filled with 0 to ensure no data is lost.

Since clustering algorithms are sensitive to different scales, the selected features are normalized using StandardScaler, which standardizes the data to have a mean of 0 and a standard deviation of 1.

The Birch (Balanced Iterative Reducing and Clustering using Hierarchies) algorithm is then applied with n_clusters=5, meaning it aims to divide the movies into five clusters based on their similarities. Birch is particularly useful for large datasets as it efficiently groups data points while maintaining memory efficiency.

After training the model, cluster labels are assigned to each movie, and the script prints the number of movies in each cluster, allowing us to analyze how the movies have been grouped based on shared characteristics. This approach can be useful for movie recommendations, pattern analysis, or genre-based clustering.

23.Apriori Algorithm

```
# APRIORI ALGORITHM
# Frequent Pattern Mining
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder

# Convert 'Genre' column into a list of lists (transactions)
df_exploded = df.assign(Genre=df['Genre'].str.split(', ')).explode('Genre')
movie_genres = df_exploded.groupby('Series_Title')['Genre'].apply(list).tolist()

# Convert data into a transaction format
te = TransactionEncoder()
te_ary = te.fit(movie_genres).transform(movie_genres)
df_genres = pd.DataFrame(te_ary, columns=te.columns_)

# Applying algorithm
frequent_itemsets = apriori(df_genres, min_support=0.05, use_colnames=True)

# Extract association rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)

# Display results
print("Frequent itemsets:\n", frequent_itemsets)
print("\nAssociation Rules:\n", rules)
```

I used the Apriori algorithm to find frequent patterns in movie genres. First, I split the genre column so each movie had a list of genres. Then, I transformed that data into a one-hot encoded format suitable for the algorithm.

I applied the Apriori algorithm with a minimum support of 5% to identify frequent genre combinations. After that, I generated association rules to understand how the presence of one genre is related to another. The results showed that genres like Drama, Comedy, and Action are very common, and certain pairs like Adventure and Action often appear together.

Frequent itemsets:

	support	itemsets
0	0.189189	(Action)
1	0.196196	(Adventure)
2	0.082082	(Animation)
3	0.109109	(Biography)
4	0.233233	(Comedy)
5	0.208208	(Crime)
6	0.723724	(Drama)
7	0.056056	(Family)
8	0.066066	(Fantasy)
9	0.056056	(History)
10	0.099099	(Mystery)
11	0.125125	(Romance)
12	0.067067	(Sci-Fi)
13	0.137137	(Thriller)
14	0.051051	(War)
15	0.083083	(Adventure, Action)
16	0.055055	(Crime, Action)
17	0.077077	(Drama, Action)
18	0.052052	(Adventure, Animation)
19	0.054054	(Comedy, Adventure)
20	0.065065	(Drama, Adventure)
21	0.103103	(Drama, Biography)
22	0.123123	(Comedy, Drama)
23	0.159159	(Drama, Crime)
24	0.054054	(Drama, History)
25	0.065065	(Drama, Mystery)
26	0.106106	(Drama, Romance)
27	0.083083	(Drama, Thriller)

Association Rules:

	antecedents	consequents	antecedent support	consequent support	\
0	(Adventure)	(Action)	0.196196	0.189189	
1	(Action)	(Adventure)	0.189189	0.196196	
2	(Crime)	(Action)	0.208208	0.189189	
3	(Action)	(Crime)	0.189189	0.208208	
4	(Adventure)	(Animation)	0.196196	0.082082	
5	(Animation)	(Adventure)	0.082082	0.196196	
6	(Comedy)	(Adventure)	0.233233	0.196196	
7	(Adventure)	(Comedy)	0.196196	0.233233	
8	(Drama)	(Biography)	0.723724	0.109109	

Picture 41. Frequent Itemset and Association Rules

							\
12	(Drama)	(History)	0.723724	0.056056			
13	(History)	(Drama)	0.056056	0.723724			
14	(Drama)	(Romance)	0.723724	0.125125			
15	(Romance)	(Drama)	0.125125	0.723724			

	support	confidence	lift	representativity	leverage	conviction	\
0	0.083083	0.423469	2.238338	1.0	0.045965	1.406362	
1	0.083083	0.439153	2.238338	1.0	0.045965	1.433197	
2	0.055055	0.264423	1.397665	1.0	0.015664	1.102279	
3	0.055055	0.291005	1.397665	1.0	0.015664	1.116781	
4	0.052052	0.265306	3.232205	1.0	0.035948	1.249388	
5	0.052052	0.634146	3.232205	1.0	0.035948	2.197064	
6	0.054054	0.231760	1.181265	1.0	0.008295	1.046292	
7	0.054054	0.275510	1.181265	1.0	0.008295	1.058354	
8	0.103103	0.142462	1.305684	1.0	0.024138	1.038894	
9	0.103103	0.944954	1.305684	1.0	0.024138	5.019019	
10	0.159159	0.219917	1.056236	1.0	0.008474	1.015010	
11	0.159159	0.764423	1.056236	1.0	0.008474	1.172765	
12	0.054054	0.074689	1.332395	1.0	0.013485	1.020137	
13	0.054054	0.964286	1.332395	1.0	0.013485	7.735736	
14	0.106106	0.146611	1.171718	1.0	0.015550	1.025178	
15	0.106106	0.848000	1.171718	1.0	0.015550	1.817607	

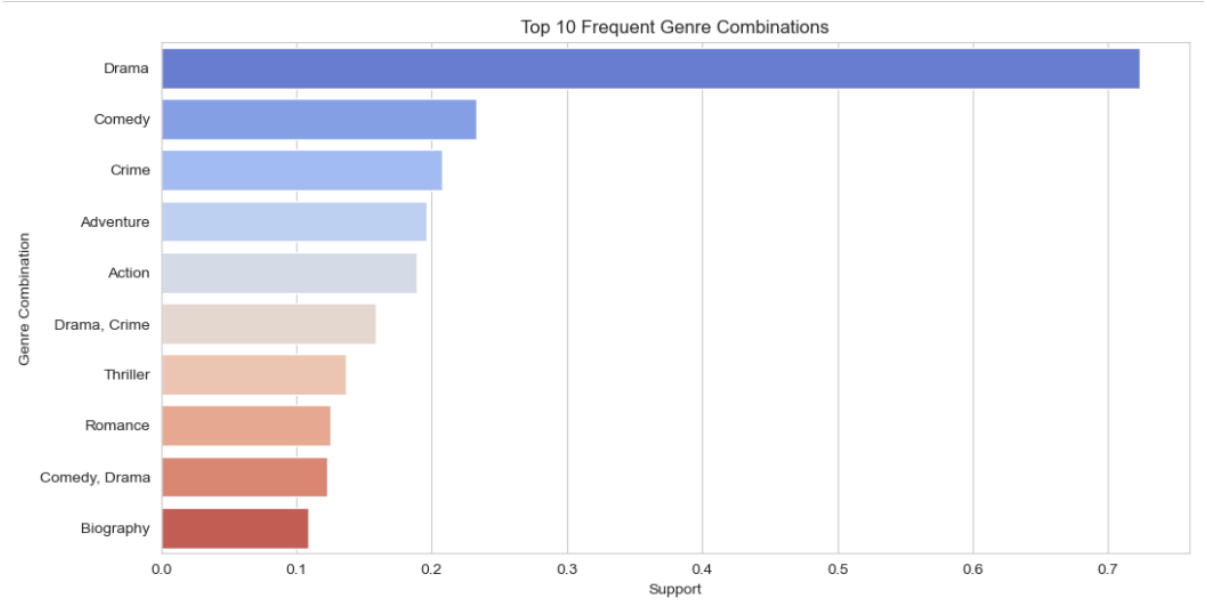
	zhangs_metric	jaccard	certainty	kulczynski
0	0.688277	0.274834	0.288946	0.431311
1	0.682329	0.274834	0.302259	0.431311
2	0.359338	0.160819	0.092788	0.277714
3	0.350909	0.160819	0.104569	0.277714
4	0.859182	0.230088	0.199608	0.449726
5	0.752370	0.230088	0.544847	0.449726
6	0.200126	0.144000	0.044244	0.253635
7	0.190904	0.144000	0.055137	0.253635
8	0.847404	0.141289	0.037438	0.543708
9	0.262790	0.141289	0.800758	0.543708
10	0.192713	0.205959	0.014788	0.492170
11	0.067242	0.205959	0.147314	0.492170
12	0.902979	0.074483	0.019739	0.519487
13	0.264287	0.074483	0.870730	0.519487
14	0.530455	0.142857	0.024559	0.497306
15	0.167512	0.142857	0.449826	0.497306

Picture 42. Frequent Itemset and Association Rules

I used Seaborn and Matplotlib to visualize the top 10 most frequent genre combinations from the Apriori results. I selected the top 10 itemsets with the highest support values. Then, I created a horizontal bar chart where the y-axis shows the genre combinations (converted from sets to readable strings), and the x-axis shows their support values. The chart clearly displays which genre combinations are most common in the dataset.

24. Visualization of Frequent Genre Combinations

```
import seaborn as sns
import matplotlib.pyplot as plt

# Get top 10 frequent genre pairs
top_frequent = frequent_itemsets.nlargest(10, 'support')

plt.figure(figsize=(12, 6))
sns.barplot(y=top_frequent['itemsets'].apply(lambda x: ', '.join(x)), x=top_frequent['support'], palette='coolwarm')

plt.title('Top 10 Frequent Genre Combinations')
plt.xlabel('Support')
plt.ylabel('Genre Combination')

plt.show()
```



Picture 43. Frequent Itemset and Association Rules

25.Decade-Based Genre Pattern Mining Using Apriori

I created a new column called Decade by rounding each movie's release year to the nearest decade. Then, I combined the decade and genre into a single list to form Decade_Genre, which allows the Apriori algorithm to discover associations between time periods and genres. I converted this data into a transaction format using TransactionEncoder and applied the Apriori algorithm with a minimum support of 2% to find frequent combinations of decades and genres. After that, I generated association rules to understand which genres were most common in specific decades. This helped me explore how genre popularity has changed over time.

```
df['Decade'] = (df['Released_Year'] // 10) * 10

# Combine decade and genre
df['Decade_Genre'] = df.apply(lambda x: str(x['Decade']) + ', ' + x['Genre'], axis=1)
df['Decade_Genre'] = df['Decade_Genre'].str.split(', ')

# Convert into transaction format
te = TransactionEncoder()
te_ary = te.fit(df['Decade_Genre']).transform(df['Decade_Genre'])
df_decade_genre = pd.DataFrame(te_ary, columns=te.columns_)

# Apply Apriori
frequent_decade_genre = apriori(df_decade_genre, min_support=0.02, use_colnames=True)

# Extract association rules
decade_genre_rules = association_rules(frequent_decade_genre, metric="lift", min_threshold=1)

# Display results
print("Frequent Decade-Genre Combinations:\n", frequent_decade_genre)
print("\nDecade-Genre Association Rules:\n", decade_genre_rules)
```

```
Frequent Decade-Genre Combinations:
      support                       itemsets
0       0.024                      (1930.0)
1       0.035                      (1940.0)
2       0.056                      (1950.0)
3       0.073                      (1960.0)
4       0.076                      (1970.0)
..        ...                           ...
116     0.024  (Comedy, Adventure, Animation)
117     0.028     (Drama, History, Biography)
118     0.031        (Drama, Comedy, Romance)
119     0.027         (Drama, Crime, Mystery)
120     0.028        (Drama, Thriller, Crime)

[121 rows x 2 columns]

Decade-Genre Association Rules:
            antecedents        consequents  antecedent support  \
0             (Drama)           (1950.0)               0.724
1            (1950.0)            (Drama)               0.056
2            (1980.0)           (Action)               0.089
3            (Action)           (1980.0)               0.189
4         (Adventure)           (1980.0)               0.196
..                ...                ...                 ...
203         (Mystery)     (Drama, Crime)               0.099
204  (Drama, Thriller)            (Crime)               0.083
205     (Drama, Crime)         (Thriller)               0.160
206         (Thriller)     (Drama, Crime)               0.137
207            (Crime)  (Drama, Thriller)               0.209

     consequent support  support  confidence      lift  representativity  \
0                 0.056    0.047    0.064917  1.159234               1.0
1                 0.724    0.047    0.839286  1.159234               1.0
2                 0.189    0.020    0.224719  1.188990               1.0
3                 0.089    0.020    0.105820  1.188990               1.0
4                 0.089    0.022    0.112245  1.261179               1.0
..                  ...      ...         ...       ...               ...
203               0.160    0.027    0.272727  1.704545               1.0
204               0.209    0.028    0.337349  1.614112               1.0
205               0.137    0.028    0.175000  1.277372               1.0
```

Picture 44. Frequent Decade-Genre and Association Rules

```
206         (Thriller)     (Drama, Crime)               0.137
207            (Crime)  (Drama, Thriller)               0.209

     consequent support  support  confidence      lift  representativity  \
0                 0.056    0.047    0.064917  1.159234               1.0
1                 0.724    0.047    0.839286  1.159234               1.0
2                 0.189    0.020    0.224719  1.188990               1.0
3                 0.089    0.020    0.105820  1.188990               1.0
4                 0.089    0.022    0.112245  1.261179               1.0
..                  ...      ...         ...       ...               ...
203               0.160    0.027    0.272727  1.704545               1.0
204               0.209    0.028    0.337349  1.614112               1.0
205               0.137    0.028    0.175000  1.277372               1.0
206               0.160    0.028    0.204380  1.277372               1.0
207               0.083    0.028    0.133971  1.614112               1.0

     leverage  conviction  zhangs_metric   jaccard  certainty  kulczynski
0    0.006456    1.009536       0.497687  0.064120   0.009446    0.452101
1    0.006456    1.717333       0.145510  0.064120   0.417702    0.452101
2    0.003179    1.046072       0.174479  0.077519   0.044043    0.165270
3    0.003179    1.018811       0.195993  0.077519   0.018463    0.165270
4    0.004556    1.026184       0.257576  0.083650   0.025516    0.179718
..        ...         ...            ...       ...        ...         ...
203  0.011160    1.155000       0.458750  0.116379   0.134199    0.220739
204  0.010653    1.193691       0.414901  0.106061   0.162262    0.235660
205  0.006080    1.046061       0.258503  0.104089   0.044032    0.189690
206  0.006080    1.055780       0.251614  0.104089   0.052833    0.189690
207  0.010653    1.058856       0.480992  0.106061   0.055585    0.235660

[208 rows x 14 columns]
```

Picture 45. Frequent Decade-Genre and Association Rules

26. Visualization of Frequent Genre Combinations

I first split and exploded the genre column so each movie had individual genre entries. Then, I grouped genres by movie and converted the data into a transaction format using one-hot encoding. I applied the Apriori algorithm with a 2% minimum support and filtered the results to keep only genre combinations containing three or more genres. This helped me identify the most frequent complex multi-genre patterns used in movies.

```
# Convert 'Genre' column into a list of lists
df_exploded = df.assign(Genre=df['Genre'].str.split(', ')).explode('Genre')
movie_genres = df_exploded.groupby('Series_Title')['Genre'].apply(list).tolist()

# Convert data into a transaction format
te = TransactionEncoder()
te_ary = te.fit(movie_genres).transform(movie_genres)
df_genres = pd.DataFrame(te_ary, columns=te.columns_)

# Apply Apriori for 3-itemsets or more
frequent_genre_sets = apriori(df_genres, min_support=0.02, use_colnames=True)

# Display only sets with 3 or more genres
multi_genre_combinations = frequent_genre_sets[frequent_genre_sets['itemsets'].apply(lambda x: len(x) >= 3)]
print("Frequent Multi-Genre Combinations:\n", multi_genre_combinations)
```

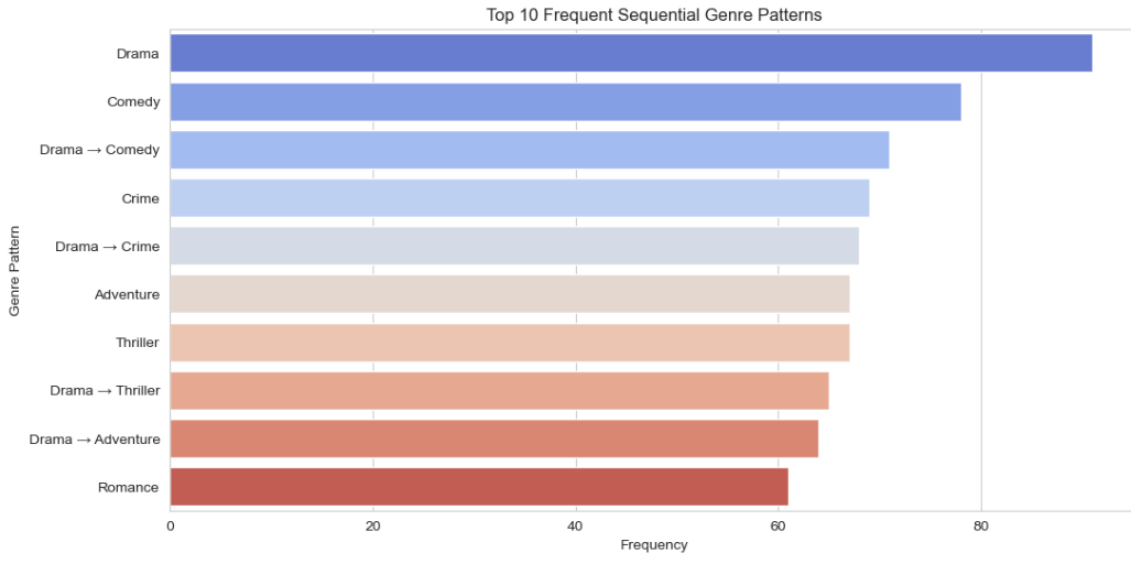
```
Frequent Multi-Genre Combinations:
      support                        itemsets
51  0.021021     (Sci-Fi, Adventure, Action)
52  0.030030          (Drama, Crime, Action)
53  0.024024  (Comedy, Adventure, Animation)
54  0.028028     (Drama, History, Biography)
55  0.031031        (Comedy, Romance, Drama)
56  0.027027         (Drama, Crime, Mystery)
57  0.028028        (Drama, Thriller, Crime)
```

27. Sequential Pattern Mining of Genre Trends Over Time Using SPADE

I used the SPADE algorithm to find frequent sequential patterns in movie genres across release years. After sorting the movies by release year, I grouped and cleaned the genre data for each year to prepare yearly sequences. Then, I applied freq_seq_enum with a minimum support of 3 to discover frequent genre sequences that occur across multiple years. I visualized the top 10 patterns using a horizontal bar chart, which helped me identify the most recurring genre transitions over time.

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pymining import seqmining

# Load the dataset
file_path = "C:\\Users\\Korisnik\\Downloads\\archive\\imdb_top_1000.csv"
df = pd.read_csv(file_path)

#  Convert 'Released_Year' to integer format
df['Released_Year'] = pd.to_numeric(df['Released_Year'], errors='coerce').dropna().astype(int)

# Sort movies by release year
df_sorted = df.sort_values(by='Released_Year')

# Prepare sequences (group genres by year)
yearly_sequences = df_sorted.groupby('Released_Year')['Genre'].apply(lambda x: list(set(', '.join(x).split(', '))).tolist()

# Apply SPADE Algorithm to find frequent genre sequences
patterns = seqmining.freq_seq_enum(yearly_sequences, 3)  # Min support of 3 occurrences

# Display only the top 10 frequent patterns to avoid Jupyter overload
print("Top 10 Frequent Sequential Patterns in Movie Genres Across Years:\n")
top_patterns = sorted(patterns, key=lambda x: x[1], reverse=True)[:10]

for pattern, freq in top_patterns:
    print(f"Pattern: {pattern}, Frequency: {freq}")

# Convert results into a DataFrame for visualization
df_patterns = pd.DataFrame(top_patterns, columns=['Pattern', 'Frequency'])
df_patterns['Pattern'] = df_patterns['Pattern'].apply(lambda x: ' → '.join(x))  # Format patterns

# Visualize results using a bar chart
plt.figure(figsize=(12, 6))
sns.barplot(y=df_patterns['Pattern'], x=df_patterns['Frequency'], palette='coolwarm')

plt.title('Top 10 Frequent Sequential Genre Patterns')
plt.xlabel('Frequency')
plt.ylabel('Genre Pattern')

plt.show()
```

```
Top 10 Frequent Sequential Patterns in Movie Genres Across Years:

Pattern: ('Drama',), Frequency: 91
Pattern: ('Comedy',), Frequency: 78
Pattern: ('Drama', 'Comedy'), Frequency: 71
Pattern: ('Crime',), Frequency: 69
Pattern: ('Drama', 'Crime'), Frequency: 68
Pattern: ('Adventure',), Frequency: 67
Pattern: ('Thriller',), Frequency: 67
Pattern: ('Drama', 'Thriller'), Frequency: 65
Pattern: ('Drama', 'Adventure'), Frequency: 64
Pattern: ('Romance',), Frequency: 61
```

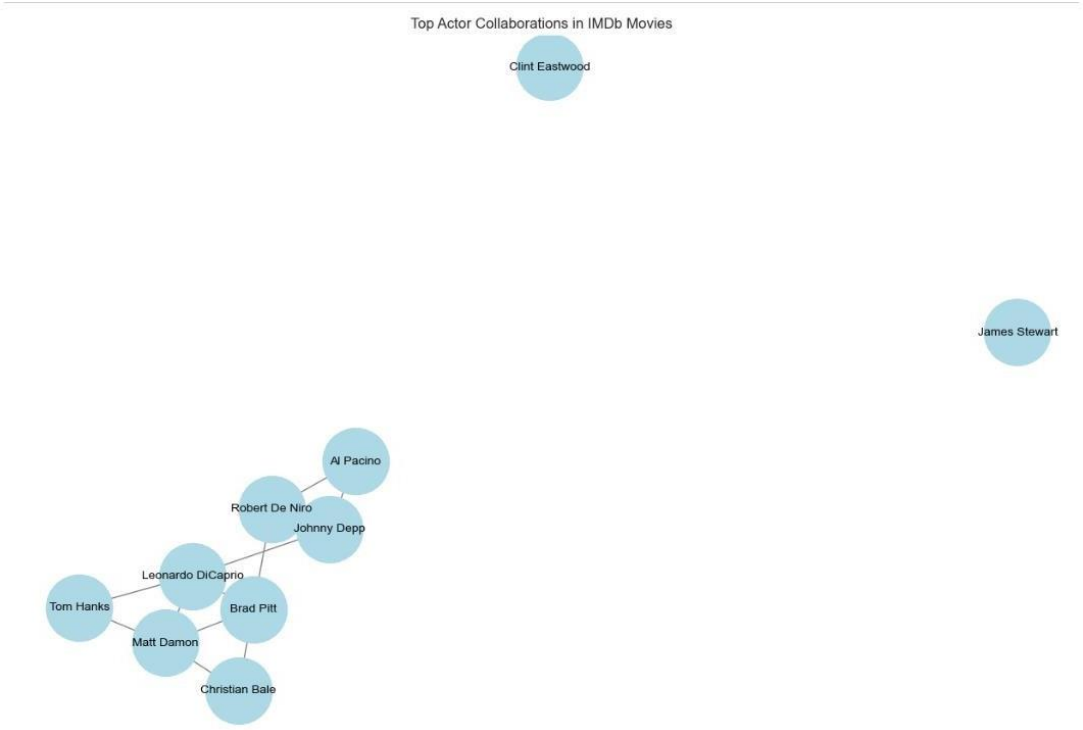


Picture 46. Top 10 Frequent Sequential Genre Patterns

28.Network Analysis of Actor Collaborations in Top IMDb Movies

```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt

# Load dataset
file_path = "C:\\Users\\Korisnik\\Downloads\\archive\\imdb_top_1000.csv"
df = pd.read_csv(file_path)

# Create an empty graph
G = nx.Graph()

# Add actor collaborations as edges
for _, row in df.iterrows():
    actors = [row['Star1'], row['Star2'], row['Star3'], row['Star4']]
    actors = [actor for actor in actors if pd.notna(actor)]  # Remove missing values

    for i in range(len(actors)):
        for j in range(i + 1, len(actors)):  # Create edges between co-actors
            G.add_edge(actors[i], actors[j], movie=row['Series_Title'])

# Visualize the Most Frequent Actor Collaborations
plt.figure(figsize=(12, 8))
top_actors = [node for node, degree in sorted(G.degree, key=lambda x: x[1], reverse=True)[:10]]  # Top 10 actors
subgraph = G.subgraph(top_actors)

nx.draw(subgraph, with_labels=True, node_color='lightblue', edge_color='gray', node_size=3000, font_size=10)
plt.title("Top Actor Collaborations in IMDb Movies")
plt.show()
```

I used NetworkX to create a graph-based network of actor collaborations. Each actor is represented as a node, and edges are added between actors who appeared together in a movie. I then extracted and visualized a subgraph of the top 10 most connected actors to show the strongest collaboration patterns. This visualization highlights which actors frequently co-star with others in highly rated IMDb films.



Picture 47. Network Analysis of Actor Collaborations in Top IMDb Movies

29. Analysis and Comparison of Algorithm Results

In this project, three classification algorithms were used to predict movie genres based on features like IMDb rating, runtime, and number of votes. These included Naïve Bayes, Random Forest, and AdaBoost. Among them, the Naïve Bayes classifier performed the best with an accuracy of 26%, followed by Random Forest with 24%, and AdaBoost with the lowest accuracy of 21%. These results indicate that predicting genres based only on numeric features is highly challenging, likely because genre is a complex label that depends on many non-numeric elements such as story, setting, and themes. While all three models struggled, Naïve Bayes slightly outperformed the others due to its simplicity and assumptions that may have loosely fit the data.

29.1 High IMDb Rating Prediction

In a separate binary classification task, AdaBoost was used to predict whether a movie would have a high IMDb rating (above 8.5). The model achieved an impressive accuracy of 97%, demonstrating a strong predictive capability. This result suggests that classifying movies as “high rating” or “not high rating” is much easier and more accurate than trying to predict exact genres. The success of this model likely comes from good feature engineering and the nature of the binary target, which is more predictable using available data such as meta scores, runtime, and actor involvement.

29.2 Clustering Results

Two clustering algorithms were applied: DBSCAN and Birch. DBSCAN grouped movies based on runtime and IMDb rating and was particularly effective at detecting outliers. It created 17 clusters in total, with some clusters containing many movies while others were very small. A large number of movies were labeled as outliers (Cluster -1), showing DBSCAN’s strength in identifying unique or unusual data points.

On the other hand, Birch clustering used a broader set of features and divided the data into 5 balanced clusters. It provided more compact and interpretable clusters, which could be used for content grouping or personalized recommendations. Compared to DBSCAN, Birch was more scalable and gave clearer clusters, although it didn’t highlight outliers as effectively.

29.3 Frequent Pattern Mining

To explore genre relationships, the Apriori algorithm was used in several ways. First, it was used to find frequent genre combinations, revealing common pairs like Adventure and Action or Drama and Romance. Then, it was used to mine decade-based patterns, showing which genres were popular in which time periods (e.g., Drama frequently appeared across decades like 1990s and 2000s). Finally, Apriori was applied to find multi-genre itemsets (three or more genres) that commonly appeared together in movies. These insights were further visualized with bar charts, offering a clear view of the most recurring genre patterns.

29.4 Sequential Pattern Mining with SPADE

To understand how genre trends evolved over time, you used the SPADE algorithm, which performs sequential pattern mining. Unlike Apriori, which finds frequent co-occurrences, SPADE finds frequent genre sequences across release years.

29.5 Network Analysis of Actor Collaborations

I also used **NetworkX** to perform a graph-based analysis of actor collaborations. By connecting actors who appeared in the same films, I built a network showing the most frequent co-stars in top IMDb movies. The top 10 most connected actors were visualized as a subgraph, revealing strong collaboration patterns. This analysis, while not predictive, gave useful insights into relationships within the film industry and highlighted actors with the most diverse connections.

29.6 Overall Conclusion

Overall, each algorithm provided different types of insights. The AdaBoost classifier for high rating prediction gave the most accurate result (97%), while Naïve Bayes was the best among genre classifiers despite low performance. In clustering, Birch produced more balanced clusters, while DBSCAN was better for spotting outliers. For pattern mining, Apriori and SPADE both revealed important trends—Apriori focusing on co-occurrence and SPADE on sequences. Finally, NetworkX gave a unique visual perspective on actor collaborations, completing a well-rounded set of analyses in your project.