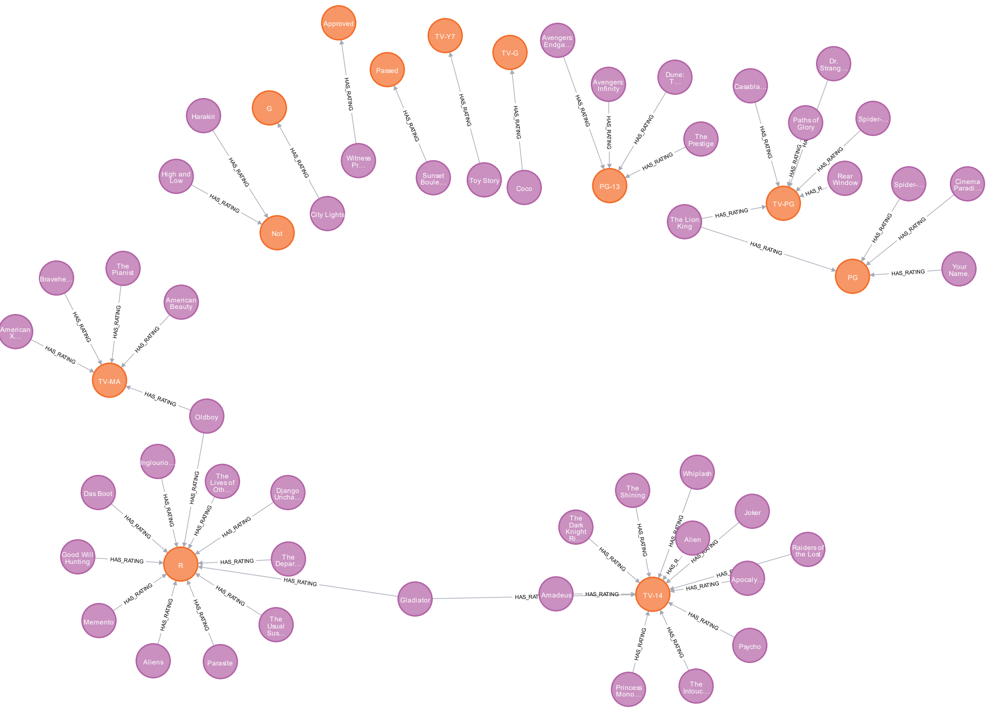


MOVIE DATA ANALYSIS

A Study on Movie Data from IMDb and Metacritic

Abstract

This report explores the process of collecting, processing, and analyzing movie data using web scraping and a graph database. Data is extracted from IMDb and Metacritic, structured, and stored in Neo4j to examine relationships between movies, ratings, and genres. Cypher queries are used to derive insights such as popular movies, rating distributions, and director collaborations. This study demonstrates the power of graph databases in analyzing interconnected data and uncovering meaningful trends in the movie industry.

GANESHMAHARAJ KAMATHAM

22BDS0168

Contents

1. Project Overview

Introduction

In today's digital era, vast amounts of data are generated across various domains, including entertainment, where movie databases hold extensive information about films, ratings, genres, and industry trends. Analyzing such data can provide valuable insights into audience preferences, critical reception, and patterns in filmmaking.

This project focuses on extracting, processing, and analyzing movie data using a graph database approach. Unlike traditional relational databases, graph databases like Neo4j efficiently store and analyze relationships between entities, such as movies, directors, genres, and ratings.

Why Movie Data?

Movie data serves as an excellent domain for analysis due to its structured nature and real-world applicability. It helps answer various industry-relevant questions, such as :

- What makes a movie highly rated?
- Do certain directors consistently produce top-rated films?
- How do critic and audience ratings compare across genres?
- Is there a correlation between a movie's runtime and its rating?

Objective

The primary goal of this project is to **collect, store, and analyze movie-related data** using a **graph database (Neo4j)**. The project involves the following key steps:

- **Web Scraping**: Extracting movie data from two different sources to gather information on attributes such as title, year, IMDb rating, Metascore, runtime, and MPAA rating.

- **Data Processing**: Cleaning, transforming, and merging datasets to ensure consistency and correctness.

- **Graph Database Storage**: Structuring the movie data in a graph format and storing it in **Neo4j**, where entities (movies, ratings, etc.) are represented as **nodes**, and relationships (e.g., HAS_RATING) are **edges**.

- **Querying & Analysis**: Writing **Cypher queries** to retrieve insights such as highly-rated movies, year-based trends, and relationships between different attributes.

- **Visualization**: Using **graph-based visualization techniques** to understand and explore relationships between movies and their attributes.

This project showcases how **graph databases** can be effectively utilized for **complex data relationships and network-based analysis** in the entertainment industry.

Tools Used

Programming Language

- **Python** → Used for web scraping, data processing, database interaction, querying, and visualization.

Libraries & Frameworks

- **BeautifulSoup** → Extracting structured data from web pages.
- **Requests** → Fetching web pages for scraping.
- **Pandas** → Data processing, transformation, and merging datasets.
- **Py2Neo** → Connecting to **Neo4j**, inserting data, and running Cypher queries.
- **Matplotlib** → Basic data visualization.
- **PyVis** → Creating **interactive graph visualizations** of relationships stored in Neo4j.
- **NetworkX** → Graph-based visualization and analysis of data relationships.

Database

- **Neo4j** → A **graph database** used to store and analyze relationships between movies and attributes, providing efficient querying via **Cypher**.

2. Data Collection

Data collection is a crucial step in any data-driven project, as the quality and comprehensiveness of the data directly impact the insights that can be derived. In this project, we collected **movie-related data** from two different but related sources: **IMDb** and **Metacritic**. The goal was to extract relevant attributes for each movie and merge the datasets for further analysis in a graph database.

2.1 Web Scraping Approach

Data Sources Chosen

1. **IMDb (Internet Movie Database)** – One of the largest online movie databases, providing information about movies, TV shows, actors, directors, ratings, and more.
2. **Metacritic** – A movie review aggregation website that provides Metascores based on critics' reviews.

These sources were chosen because they provide complementary information. While IMDb offers user ratings and movie metadata, Metacritic aggregates critic reviews, making it possible to analyze correlations between public and critic opinions.

Attributes Extracted

From these websites we extracted these attributes :

IMDB -> Title, Year, IMDb Rating, Runtime, Votes

Metacritic -> Title, Year, Metascore, MPAA rating, Short description

2.2 Writing Web Scraping Code

To extract the data, **BeautifulSoup** (for HTML parsing) and **Requests** (for sending HTTP requests) were used. Below is an overview of the web scraping process :

Step 1: Fetching the Webpage Content

We sent HTTP requests to IMDb and Metacritic movie listings and retrieved their HTML content.

Step 2: Parsing the HTML & Extracting Information

Using **BeautifulSoup**, we navigated the HTML structure to extract movie details based on CSS selectors.

Step 3: Handling Pagination & Dynamic Content

- IMDb contains multiple pages of movies, so we implemented pagination handling.

- Metacritic's pages had dynamic elements, which required careful handling to avoid missing data.

Step 4: Handling Missing Data & Delays to Avoid Bans

- Some movies had missing ratings or directors, so we used placeholders (None or N/A) for consistency.
- **time.sleep()** was used to introduce delays between requests to prevent IP bans.

You can find the **PYTHON SCRIPT** for this here.

2.3 Saving Scraped Data

Before further processing, the extracted data was **stored in CSV format** for easier manipulation and merging.

3. Data Processing & Cleaning

Once the data was collected from **IMDb** and **Metacritic**, it needed to be **cleaned, transformed, and merged** into a structured format suitable for storage in a **graph database (Neo4j)**. This step involved handling missing values, standardizing data fields, and combining the datasets.

3.1 Cleaning & Standardizing the Data

Raw data obtained from web scraping often contains inconsistencies, missing values, or formatting issues. To ensure consistency and accuracy, the following cleaning steps were performed using **Pandas**:

1. Handling Missing Values

- Some movies lacked IMDb ratings or Metascores.
- Missing values were replaced with **Nan** and later handled using imputation techniques (e.g., mean substitution).

2. Standardizing Data Fields

- **Year:** Converted from string to integer for numerical operations.
- **Ratings (IMDb & Metascore):** Converted to float type for calculations.
- **Genres & Cast:** Split into lists for better relational mapping in Neo4j.

3. Removing Duplicates

- If the same movie appeared multiple times (due to different editions or re-releases), duplicates were removed based on **title & year** combination.

3.2 Merging IMDb & Metacritic Data

Since both datasets contained **movie titles** and **years**, we used these attributes as the primary keys to merge the data. The goal was to **combine IMDb's user ratings with Metacritic's critic scores** for a more comprehensive analysis.

Merging Strategy

- **Inner Join:** Retained only movies present in both IMDb and Metacritic for better accuracy.
- **Outer Join:** Included all movies, filling missing Metascores or IMDb ratings with placeholders.

You can find the code for **Cleaning**, **Standardization** and **Merging** in Jupyter Notebook here.

3.3 Sample Processed Data

	Title	YEAR	IMDb Rating	Runtime	Votes	Metascore	MPAA Rating	Short Description
0	The Shawshank Redemption	1994	9.3	142	3000000	82.0	TV-14	Wrongly convicted, Andy Dufresne (Tim Robbins)...
1	The Godfather	1972	9.2	175	2100000	100.0	TV-14	Francis Ford Coppola's epic features Marlon Br...
2	The Dark Knight	2008	9.0	152	3000000	84.0	TV-14	Batman raises the stakes in his war on crime. ...
3	12 Angry Men	1957	9.0	96	914000	97.0	Approved	12 Angry Men, by Sidney Lumet, is a behind-clo...
4	The Lord of the Rings: The Return of the King	2003	9.0	201	2100000	94.0	PG-13	Sauron's forces have laid siege to Minas Tirit...

3.4 Info about Processed Data

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 238 entries, 0 to 237
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Title              238 non-null    object 
 1   YEAR               238 non-null    int64  
 2   IMDb Rating        238 non-null    float64
 3   Runtime            238 non-null    int64  
 4   Votes              238 non-null    int64  
 5   Metascore          238 non-null    float64
 6   MPAA Rating        238 non-null    object 
 7   Short Description  238 non-null    object 
dtypes: float64(2), int64(3), object(3)
memory usage: 15.0+ KB
```

4. Storing Data in Neo4j

After cleaning and merging the data, the next step was to store it in **Neo4j**, a graph database that efficiently handles **relationships** between entities like movies, directors, genres, and actors. Using **graph-based storage** allows us to analyze connections between different elements, such as finding movies by the same director or comparing ratings across genres.

4.1 Why Use Neo4j?

Unlike traditional relational databases, **Neo4j** is well-suited for movie data because:

- It enables **fast relationship-based queries** (e.g., "Find all movies directed by Christopher Nolan").
- It naturally represents entities like **Movies, Directors, and Genres as nodes**.
- It allows **flexible and scalable** data storage without complex joins.

4.2 Graph Database Schema

To structure the data, we defined a schema with **nodes** and **relationships**:

1. Nodes (Entities)

- **Movie** (:Movie) → Represents a movie with attributes like title, year, IMDb rating, Metascore, genre, etc.
- **MPAA Rating** (:MPAA_Rating) → Represents the content rating (e.g., PG-13, R).

2. Relationships (Edges)

- (:Movie)-[:HAS_RATING]->(:MPAA_Rating) → A movie has an MPAA content rating.

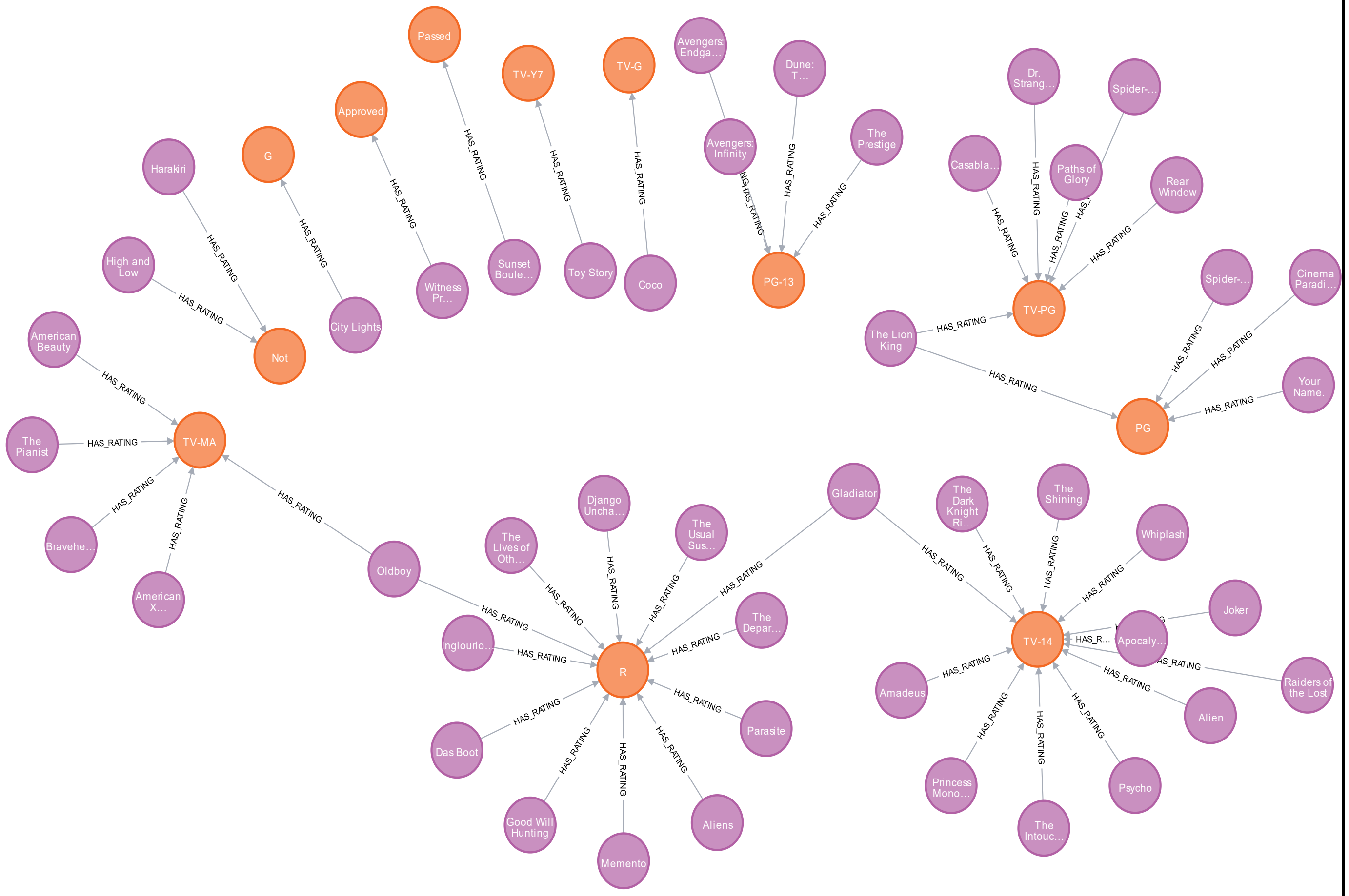
4.3 Loading Data into Neo4j using Py2Neo

To insert the processed data into Neo4j, we used **Py2Neo**, a Python library that interacts with Neo4j via Cypher queries.

1. Connecting to Neo4j Database
2. Inserting Movie Data as Nodes
3. Inserting MPAA Rating and Creating Relationships

You can find the code for **Schema design and Storing Data** in Jupyter Notebook here.

4.4 Database visualized



5. Querying the Graph Database

After storing the structured movie dataset in **Neo4j**, we can leverage **Cypher queries** to extract insights. Cypher, Neo4j's query language, is optimized for traversing relationships and finding patterns in graph data.

In this section, we perform different types of queries to:

- Retrieve all movies and their relationships.
- Find data based on specific attributes .(IMDB Rating above 9)
- Analyze relationships (Movies Having MPAA rating as PG-13)

5.1 Find all Movies and their MPAA Ratings

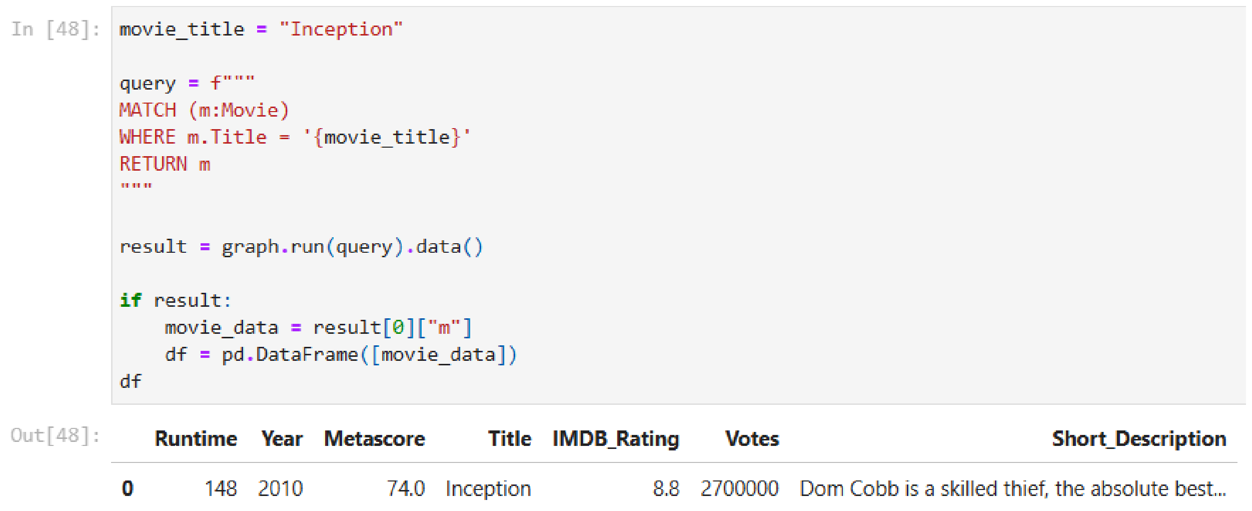
```
In [20]: query = """
MATCH (m:Movie)-[:HAS_RATING]->(r:MPAA_Rating)
RETURN m.Title AS Title, m.Year AS Year, r.Rating AS MPAA_Rating
"""

result = graph.run(query).data()
pd.DataFrame(result)
```

Out[20]:

	Title	Year	MPAA_Rating
0	The Lion King	1994	PG
1	Spider-Man: Across the Spider-Verse	2023	PG
2	Cinema Paradiso	1988	PG
3	Your Name.	2016	PG
4	Scarface	1983	PG
...	...	...	...
230	Room	2015	Not
231	The Handmaiden	2016	Not
232	The Battle of Algiers	1966	Not
233	Gangs of Wasseypur	2012	Not
234	Seven Samurai	1954	Not

235 rows × 3 columns

5.2 Retrieve Data for a Specific Movie

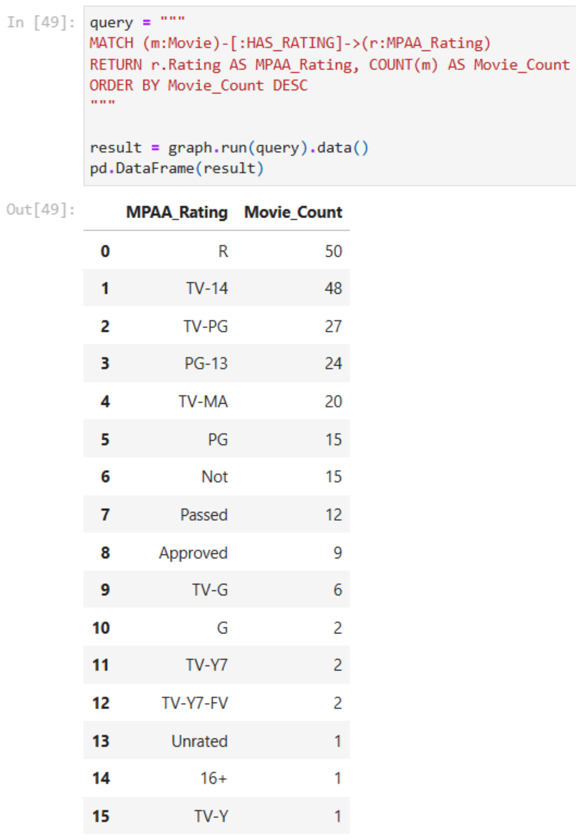


```
In [48]: movie_title = "Inception"

         query = f"""
         MATCH (m:Movie)
         WHERE m.Title = '{movie_title}'
         RETURN m
         """

         result = graph.run(query).data()

         if result:
             movie_data = result[0]["m"]
             df = pd.DataFrame([movie_data])
         df
```

Out[48]:

	Runtime	Year	Metascore	Title	IMDB_Rating	Votes	Short_Description
0	148	2010	74.0	Inception	8.8	2700000	Dom Cobb is a skilled thief, the absolute best...

5.3 Find MPAA Ratings and how many movies belong to each category

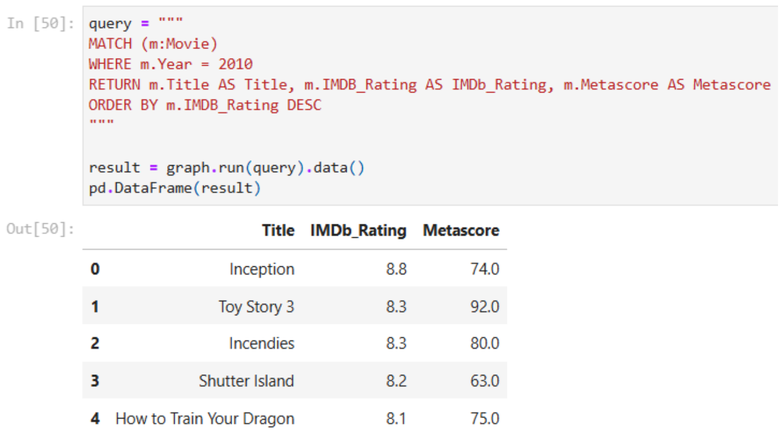


```
In [49]: query = """
         MATCH (m:Movie)-[:HAS_RATING]->(r:MPAA_Rating)
         RETURN r.Rating AS MPAA_Rating, COUNT(m) AS Movie_Count
         ORDER BY Movie_Count DESC
         """

         result = graph.run(query).data()
         pd.DataFrame(result)
```

Out[49]:

	MPAA_Rating	Movie_Count
0	R	50
1	TV-14	48
2	TV-PG	27
3	PG-13	24
4	TV-MA	20
5	PG	15
6	Not	15
7	Passed	12
8	Approved	9
9	TV-G	6
10	G	2
11	TV-Y7	2
12	TV-Y7-FV	2
13	Unrated	1
14	16+	1
15	TV-Y	1

5.4 Find all movies released in a specific year (e.g., 2010)

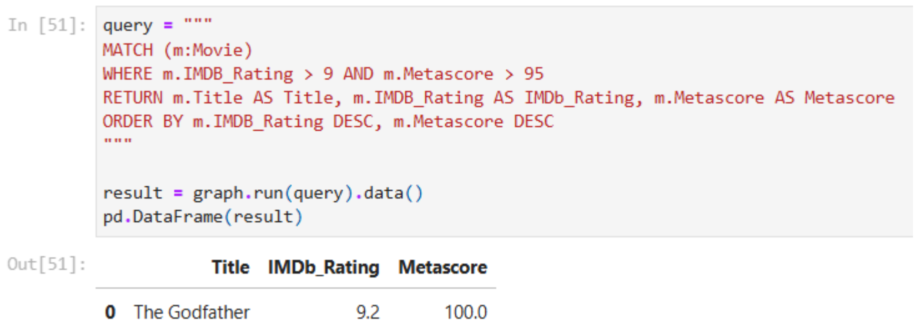


In [50]:
```
query = """
MATCH (m:Movie)
WHERE m.Year = 2010
RETURN m.Title AS Title, m.IMDB_Rating AS IMDb_Rating, m.Metascore AS Metascore
ORDER BY m.IMDB_Rating DESC
"""

result = graph.run(query).data()
pd.DataFrame(result)
```

Out[50]:

	Title	IMDb_Rating	Metascore
0	Inception	8.8	74.0
1	Toy Story 3	8.3	92.0
2	Incendies	8.3	80.0
3	Shutter Island	8.2	63.0
4	How to Train Your Dragon	8.1	75.0

5.5 Find movies with IMDb rating greater than 9 and Metascore greater than 95

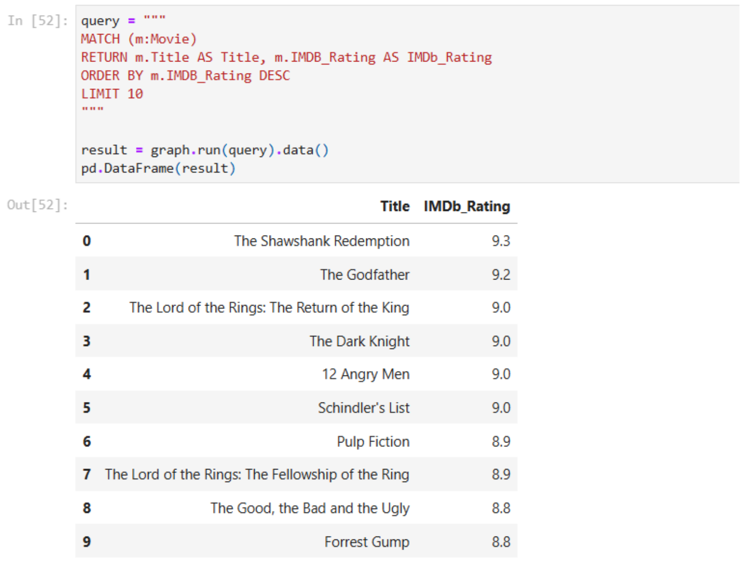


In [51]:
```
query = """
MATCH (m:Movie)
WHERE m.IMDB_Rating > 9 AND m.Metascore > 95
RETURN m.Title AS Title, m.IMDB_Rating AS IMDb_Rating, m.Metascore AS Metascore
ORDER BY m.IMDB_Rating DESC, m.Metascore DESC
"""

result = graph.run(query).data()
pd.DataFrame(result)
```

Out[51]:

	Title	IMDb_Rating	Metascore
0	The Godfather	9.2	100.0

5.6 Find top 10 highest-rated movies (IMDb rating descending)

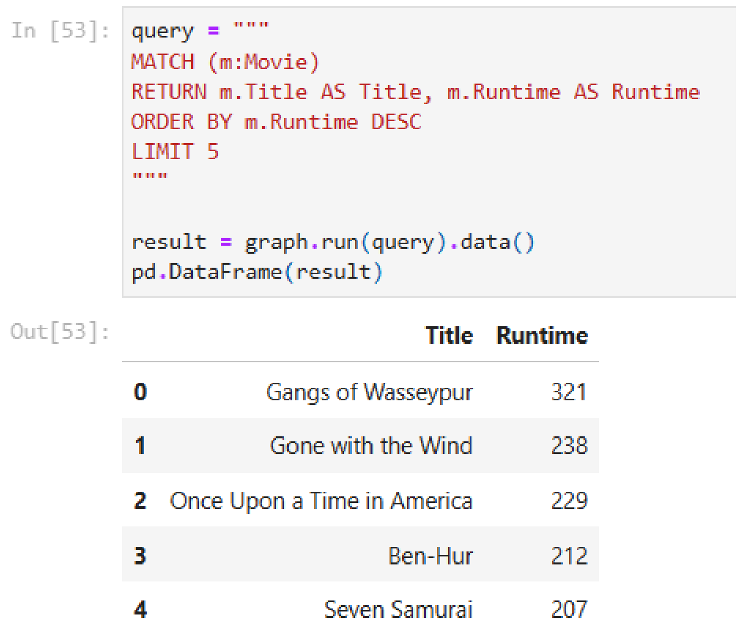


In [52]:
```
query = """
MATCH (m:Movie)
RETURN m.Title AS Title, m.IMDB_Rating AS IMDb_Rating
ORDER BY m.IMDB_Rating DESC
LIMIT 10
"""

result = graph.run(query).data()
pd.DataFrame(result)
```

Out[52]:

	Title	IMDb_Rating
0	The Shawshank Redemption	9.3
1	The Godfather	9.2
2	The Lord of the Rings: The Return of the King	9.0
3	The Dark Knight	9.0
4	12 Angry Men	9.0
5	Schindler's List	9.0
6	Pulp Fiction	8.9
7	The Lord of the Rings: The Fellowship of the Ring	8.9
8	The Good, the Bad and the Ugly	8.8
9	Forrest Gump	8.8

5.7 Find movies with the longest runtime (Top 5)



In [53]:

```
query = """
MATCH (m:Movie)
RETURN m.Title AS Title, m.Runtime AS Runtime
ORDER BY m.Runtime DESC
LIMIT 5
"""

result = graph.run(query).data()
pd.DataFrame(result)
```

Out[53]:

	Title	Runtime
0	Gangs of Wasseypur	321
1	Gone with the Wind	238
2	Once Upon a Time in America	229
3	Ben-Hur	212
4	Seven Samurai	207

You can find the code for **Queries** in Jupyter Notebook [here](#).

6. Data Visualization

Graph visualization plays a crucial role in understanding the relationships between entities stored in **Neo4j**. Since graph databases are structured as **nodes and edges**, visualizing the data helps in **identifying patterns, clusters, and insights**.

For this project, we use **PyVis** and **NetworkX** for interactive graph visualizations. Additionally, Neo4j's built-in **Bloom visualization tool** helps in exploring relationships dynamically.

6.1 Why Data Visualization?

- Helps in understanding complex relationships between **movies, actors, directors, and genres**.
- Identifies **clusters** of movies within the same genre.
- Highlights **highly connected nodes** (e.g., actors who worked in multiple genres).
- Allows dynamic filtering and exploration.

6.2 Using Neo4j Browser for Quick Visualization

Neo4j provides a built-in **graph view** that helps in interactive exploration. Running the following Cypher query in the **Neo4j browser** displays all relationships

Cypher Query for Visualization :

MATCH (n)-[r]->(m) RETURN n, r, m LIMIT 50;

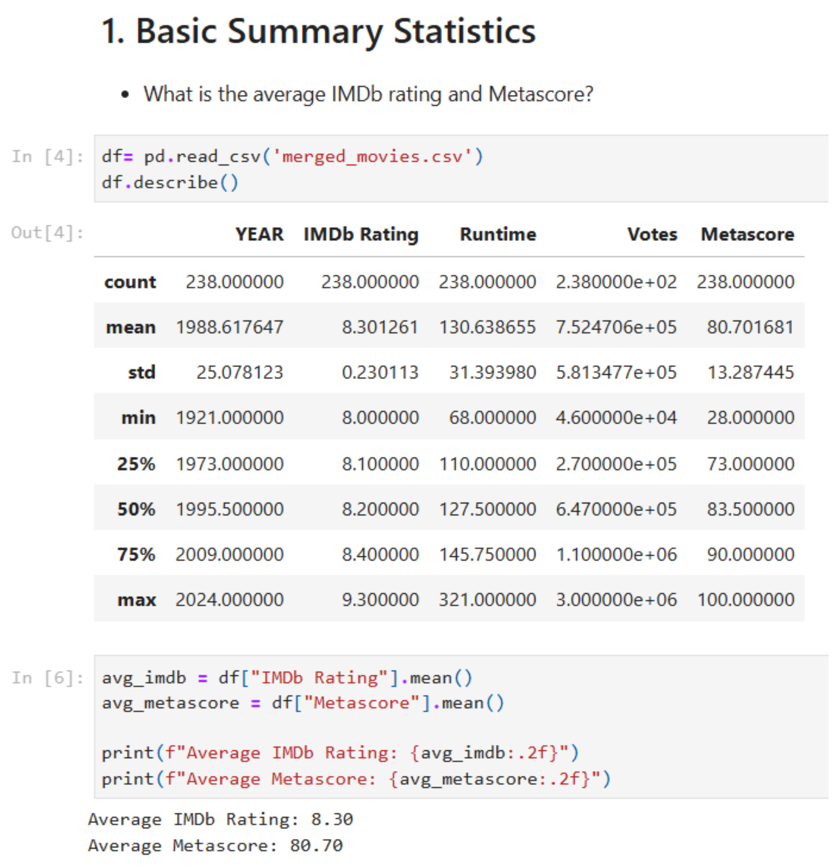
6.3 Visualizing the Graph Using PyVis

For an interactive visualization in Python, we use **PyVis**, which allows us to render the graph in a browser

Output: A **dynamic, interactive graph** that can be zoomed, dragged, and explored in a web browser.

You can find the code for **Visualizing Graph** in Jupyter Notebook and HTML file here.

7. Exploratory Data Analysis (EDA)



1. Basic Summary Statistics

- What is the average IMDb rating and Metascore?

In [4]:
```python
df= pd.read_csv('merged_movies.csv')
df.describe()
```

Out[4]:

	YEAR	IMDb Rating	Runtime	Votes	Metascore
count	238.000000	238.000000	238.000000	2.380000e+02	238.000000
mean	1988.617647	8.301261	130.638655	7.524706e+05	80.701681
std	25.078123	0.230113	31.393980	5.813477e+05	13.287445
min	1921.000000	8.000000	68.000000	4.600000e+04	28.000000
25%	1973.000000	8.100000	110.000000	2.700000e+05	73.000000
50%	1995.500000	8.200000	127.500000	6.470000e+05	83.500000
75%	2009.000000	8.400000	145.750000	1.100000e+06	90.000000
max	2024.000000	9.300000	321.000000	3.000000e+06	100.000000

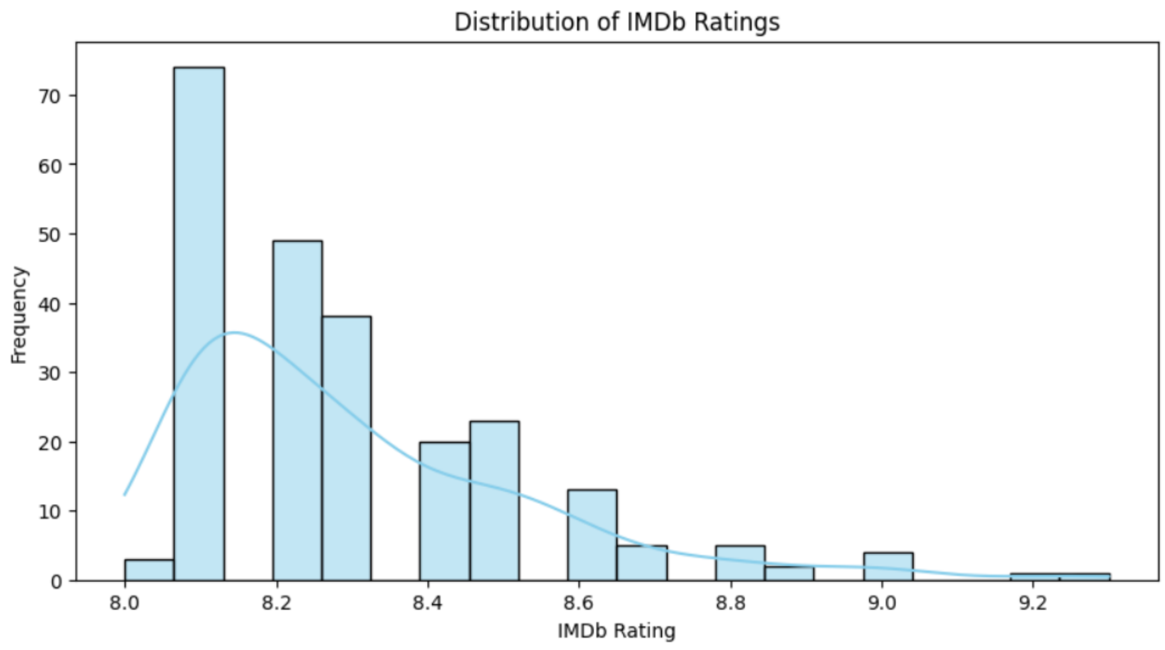
In [6]:
```python
avg_imdb = df["IMDb Rating"].mean()
avg_metascore = df["Metascore"].mean()

print(f"Average IMDb Rating: {avg_imdb:.2f}")
print(f"Average Metascore: {avg_metascore:.2f}")
```

```
Average IMDb Rating: 8.30
Average Metascore: 80.70
```

2. Distribution of IMDb Ratings

- Most movies cluster around what range of ratings?
- Is the distribution skewed?
- The distribution skewed toward high or low ratings?

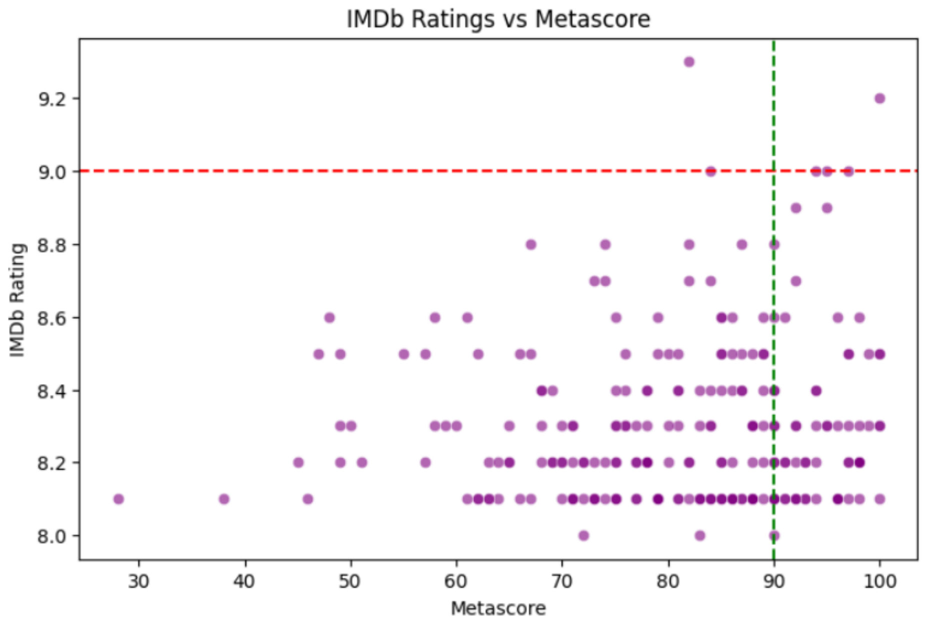
In [8]:
```
plt.figure(figsize=(10, 5))
sns.histplot(df["IMDb Rating"], bins=20, kde=True, color="skyblue")
plt.title("Distribution of IMDb Ratings")
plt.xlabel("IMDb Rating")
plt.ylabel("Frequency")
plt.show()
```



3. IMDb Rating vs. Metascore

- Do movies with high IMDb ratings (above 9) also have high Metascores (above 90)?
- Are there outliers (movies with low Metascores but high IMDb ratings)?

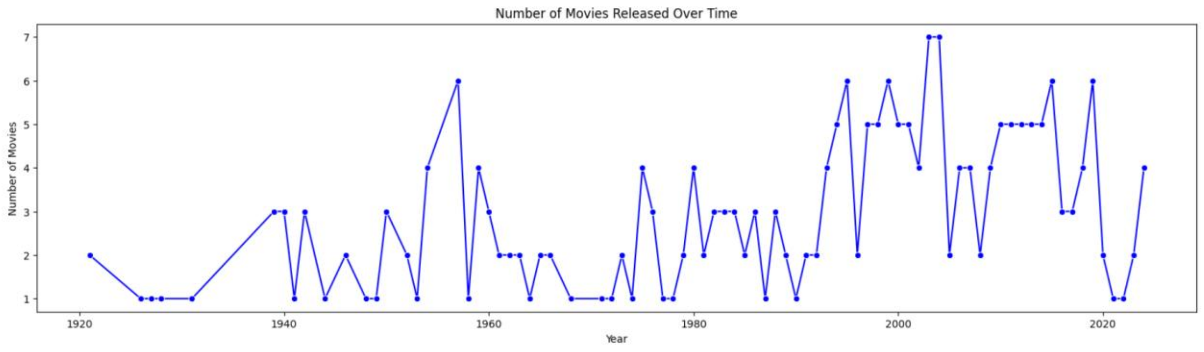
```
In [10]: plt.figure(figsize=(8,5))
         sns.scatterplot(x=df["Metascore"], y=df["IMDb Rating"], alpha=0.6, color="purple")
         plt.axhline(9, color="red", linestyle="dashed")  # Highlight high IMDb rating
         plt.axvline(90, color="green", linestyle="dashed")  # Highlight high Metascore
         plt.title("IMDb Ratings vs Metascore")
         plt.xlabel("Metascore")
         plt.ylabel("IMDb Rating")
         plt.show()
```



4. IMDb Ratings & Metascore Trends Over Time

- Has the number of movies increased significantly over time?
- Did any specific years see a spike in movie releases?

```
In [22]: movies_per_year = df.groupby("YEAR").size()

         plt.figure(figsize=(20, 5))
         sns.lineplot(x=movies_per_year.index, y=movies_per_year.values, marker="o", color="blue")
         plt.title("Number of Movies Released Over Time")
         plt.xlabel("Year")
         plt.ylabel("Number of Movies")
         plt.show()
```

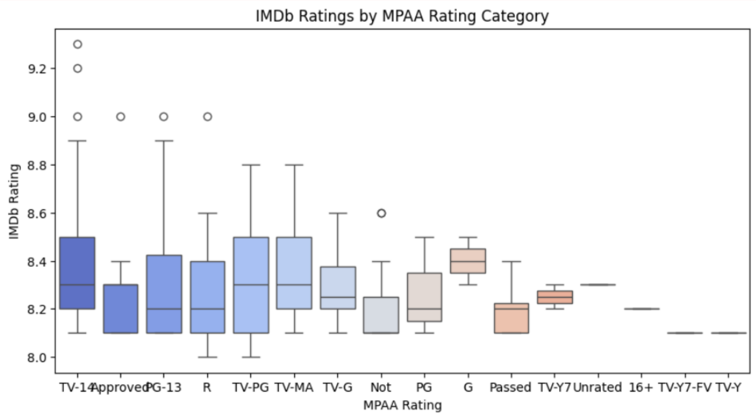


5. Rating Comparison for MPAA Ratings

- Do different MPAA Ratings (PG, PG-13, R) have significantly different IMDb Ratings?

```
In [12]: plt.figure(figsize=(10, 5))
         sns.boxplot(x=df["MPAA Rating"], y=df["IMDb Rating"], palette="coolwarm")
         plt.title("IMDb Ratings by MPAA Rating Category")
         plt.xlabel("MPAA Rating")
         plt.ylabel("IMDb Rating")
         plt.show()
```

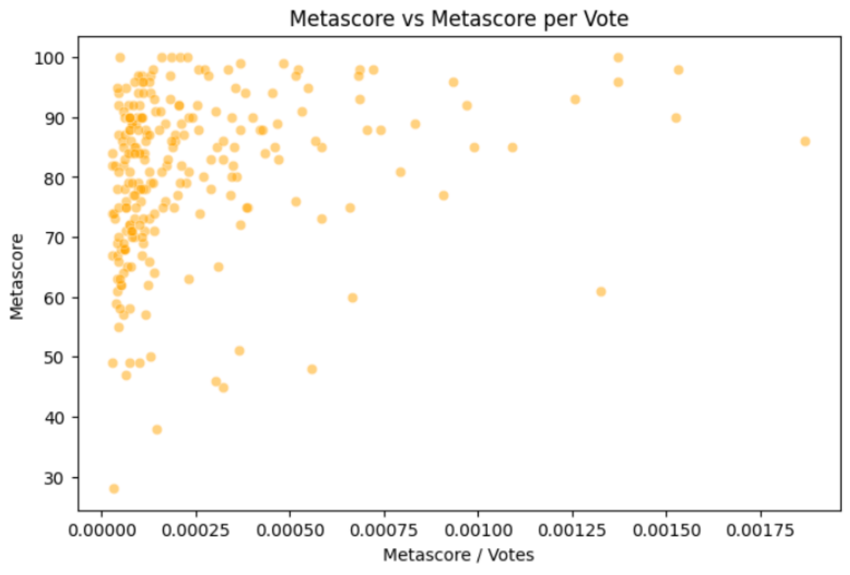
```
C:\Users\GANESH MAHARAJ K\AppData\Local\Temp\ipykernel_8784\4260104663.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x=df["MPAA Rating"], y=df["IMDb Rating"], palette="coolwarm")
```



6. Best Factor for Ratings: Metascore vs. Metascore/Votes

- Which is a better factor for analyzing good movies?

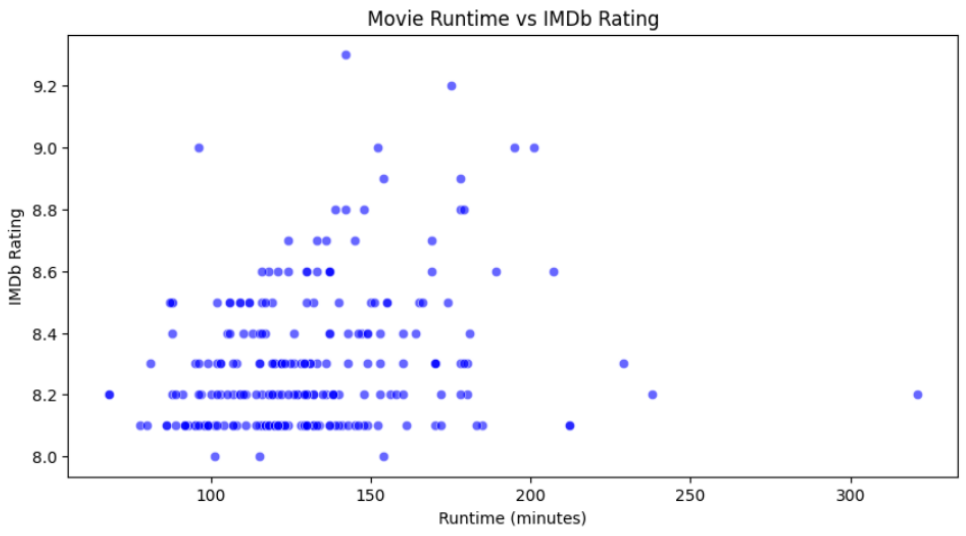
```
In [25]: df["Metascore_per_vote"] = df["Metascore"] / df["Votes"]

         plt.figure(figsize=(8,5))
         sns.scatterplot(x=df["Metascore_per_vote"], y=df["Metascore"], alpha=0.5, color="orange")
         plt.title("Metascore vs Metascore per Vote")
         plt.xlabel("Metascore / Votes")
         plt.ylabel("Metascore")
         plt.show()

         print('Corr :',df["Metascore"].corr(df["IMDb Rating"]))
```



```
Corr : 0.09616927532693909
```

7. Do longer movies (higher Runtime) have better ratings?

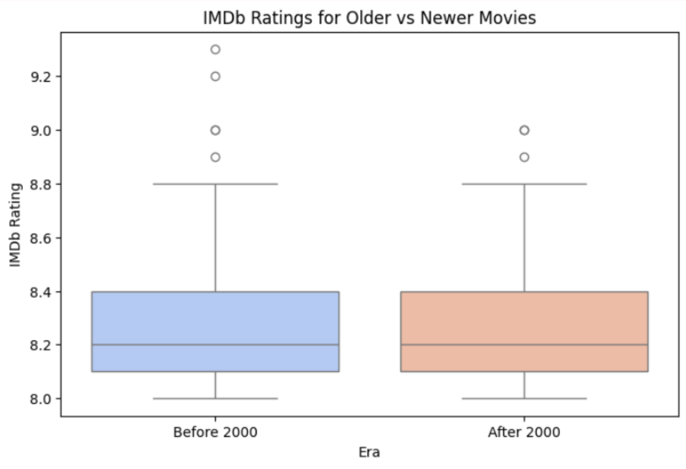
```
In [15]: plt.figure(figsize=(10, 5))
         sns.scatterplot(x=df["Runtime"], y=df["IMDb Rating"], alpha=0.6, color="blue")
         plt.title("Movie Runtime vs IMDb Rating")
         plt.xlabel("Runtime (minutes)")
         plt.ylabel("IMDb Rating")
         plt.show()
```



8. Are newer movies (after 2000) rated higher than older classics?

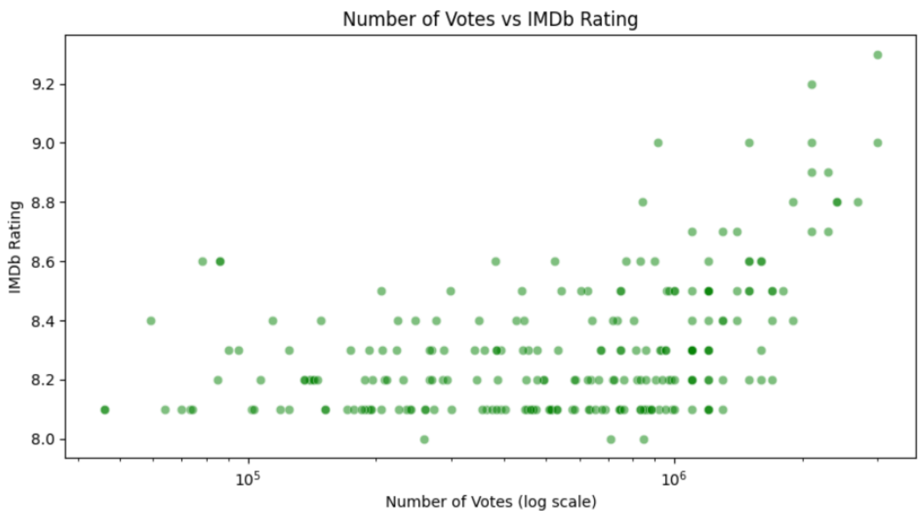
```
In [17]: df["Era"] = df["YEAR"].apply(lambda x: "Before 2000" if x < 2000 else "After 2000")

         plt.figure(figsize=(8,5))
         sns.boxplot(x=df["Era"], y=df["IMDb Rating"], palette="coolwarm")
         plt.title("IMDb Ratings for Older vs Newer Movies")
         plt.xlabel("Era")
         plt.ylabel("IMDb Rating")
         plt.show()
```

```
C:\Users\GANESH MAHARAJ K\AppData\Local\Temp\ipykernel_8784\3770745931.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x=df["Era"], y=df["IMDb Rating"], palette="coolwarm")
```



9. Is there a relationship between Votes and IMDb rating?

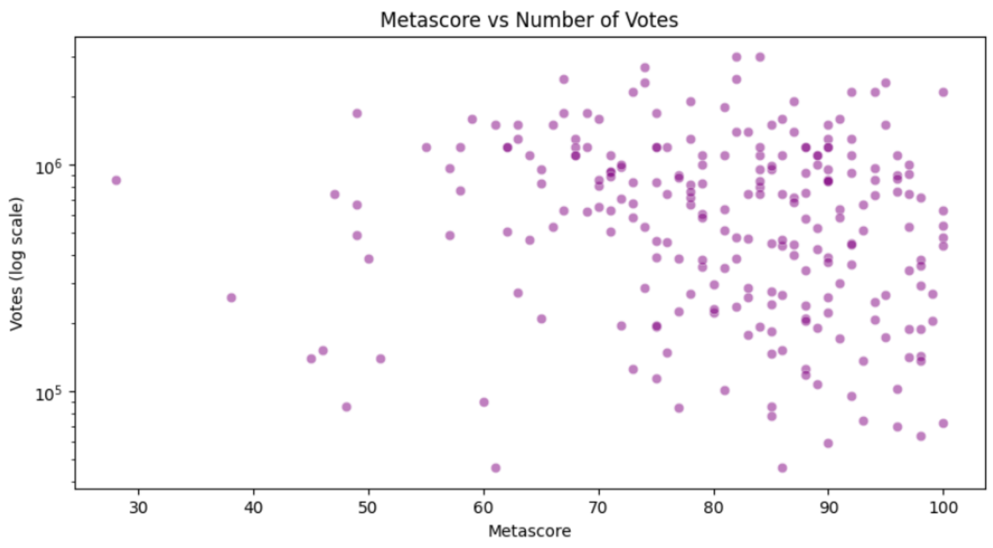
In [26]:
```
plt.figure(figsize=(10, 5))
sns.scatterplot(x=df["Votes"], y=df["IMDb Rating"], alpha=0.5, color="green")
plt.xscale("log")  # Log scale because votes can be very large
plt.title("Number of Votes vs IMDb Rating")
plt.xlabel("Number of Votes (log scale)")
plt.ylabel("IMDb Rating")
plt.show()

print('corr : ',df['IMDb Rating'].corr(df['Votes']))
```



```
corr :  0.6163369171317787
```

10. Do higher Metascore movies also get more votes?

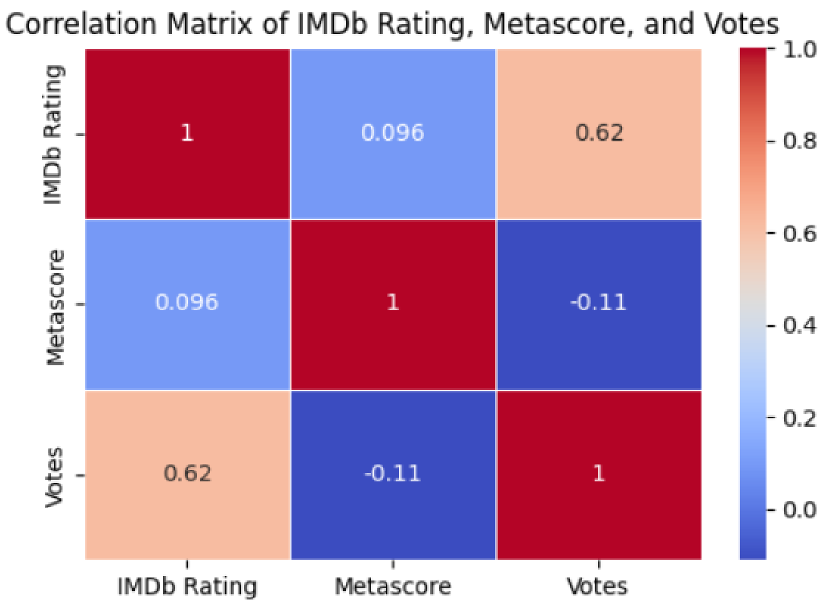
In [28]:
```
plt.figure(figsize=(10, 5))
sns.scatterplot(x=df["Metascore"], y=df["Votes"], alpha=0.5, color="purple")
plt.yscale("log")  # Log scale for better visualization
plt.title("Metascore vs Number of Votes")
plt.xlabel("Metascore")
plt.ylabel("Votes (log scale)")
plt.show()

print('corr : ',df['Metascore'].corr(df['Votes']),)
```



```
corr :  -0.1095802725784205
```

11. Do IMDb ratings correlate with both Metascore and Votes?

```
In [21]: corr_matrix = df[["IMDb Rating", "Metascore", "Votes"]].corr()

         plt.figure(figsize=(6,4))
         sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", linewidths=0.5)
         plt.title("Correlation Matrix of IMDb Rating, Metascore, and Votes")
         plt.show()
```



You can find the code for **EDA** in Jupyter Notebook here.

8. Insights and Conclusion

After Performing EDA on the merged dataset I have following insights :

8.1. Average IMDb & Metascore Ratings

- The **average IMDb rating** is **8.3**, and the **average Metascore** is **80.7**.
- Bringing both to the same scale, the **adjusted Metascore** is **8.07**, showing that IMDb and Metascore ratings are quite similar on average.

8.2. IMDb Rating Distribution

- Most movies **cluster around IMDb ratings of 8.0 - 8.3**.
- The distribution is **right-skewed**, meaning **more movies have high ratings** than low ratings.

8.3. IMDb vs. Metascore Analysis

- **Not all** movies with high IMDb ratings **(above 9)** have high Metascores **(above 90)**.
- **No major outliers** where movies have **low Metascores but high IMDb ratings**.

8.4. Trends in Movie Releases Over Time

- The **number of movies has increased significantly** over time.
- This suggests **strong growth in the film industry globally**.
- **Spikes in movie releases** were observed in **1955 & 2002**.

8.5. MPAA Ratings vs. IMDb Ratings

- **Different MPAA Ratings do not show a significant difference in IMDb ratings.**

8.6. Metascore as a Factor

- **Metascore_per_vote** and **Metascore** show a **positive but weak correlation**.
- **Metascore is NOT a strong factor** for deeper insights.

8.7. Movie Runtime vs. Ratings

- **Longer movies do NOT necessarily have better ratings.**

8.8. Newer vs. Older Movies

- **Newer movies (after 2000) are NOT rated higher** than older classics.
- **Both eras (before & after 2000) have similar rating statistics**, meaning IMDb ratings are **consistent over time**.

8.9. Votes vs. IMDb Ratings

- A **strong positive correlation** exists between **IMDb ratings & number of votes**.
- This means **movies with more votes tend to have higher IMDb ratings**.

8.10. Metascore vs. Votes

- **Higher Metascore movies do NOT get more votes**.

8.11. Correlation Matrix Insights

- **IMDb Rating & Votes show the highest correlation**.
- Other factors (Metascore, Runtime, etc.) are **not strongly correlated**.

These insights help us understand key trends in the movie dataset and how different factors affect ratings and popularity.

You can find the code for **EDA and Insights** in Jupyter Notebook and HTML file here.