VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Foundations of Data Science

Digital Assignment-1

COURSE : Foundations of Data Science

Semester : Winter Semester 2024-25

COURSE ID : BCSE 206L

CLASS NO. : VL2024250501999

SLOT : C2 + TC2

SCHOOL : SCOPE

FACULTY : Prof. SARAVANAGURU RA.K

GITHUB Repo Link : Click Here

SUBMITTED BY :

Ganesh Maharaj Kamatham (*22BDS0168*)

ALL the deliverables are in the GITHUB REPOSITORY.

PLEASE VISIT GITHUB REPO

LINK : https://github.com/GANESH-MAHARAJ/Foundations-of-DS/tree/main

It has Deliverables:

- Python scripts for scraping, processing, storing, and querying.
- Database schema design (graph structure with nodes and edges).
- Query results (screenshots).
- A short report (7-9 pages) explaining the process and insights from the data.

Question-1

1. Data Collection (Web Scraping)

Data is collected from Two popular movie rating sites:

- IMDB :
  - " https://www.imdb.com/chart/top/ "
- Metacritic :
  - "https://www.metacritic.com/browse/movie/?releaseYearMin=1910&releaseYearMax=2025&page=1"

PYTHON SCRIPT to web scrap from IMDB :

```
import time
import pandas as pd
from bs4 import BeautifulSoup
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.options import Options
from webdriver_manager.chrome import ChromeDriverManager

URL = "https://www.imdb.com/chart/top/"

# Set up Selenium WebDriver
chrome_options = Options()
# chrome_options.add_argument("--headless")  # Run in the background
chrome_options.add_argument("--disable-gpu")
chrome_options.add_argument("--no-sandbox")

driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()),
options=chrome_options)

# Open IMDb page
driver.get(URL)
time.sleep(5)  # Wait for JavaScript to load

# Scroll down to load all movies
last_height = driver.execute_script("return document.body.scrollHeight")

while True:
```

```
    driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
    time.sleep(2)
    new_height = driver.execute_script("return document.body.scrollHeight")
    if new_height == last_height:
        break
    last_height = new_height

# Parse page source with BeautifulSoup
soup = BeautifulSoup(driver.page_source, "html.parser")

# Extract movie elements
movies = soup.select(".ipc-metadata-list-summary-item")
movie_data = []

for movie in movies:
    title = movie.select_one("h3").text.strip() if movie.select_one("h3")
else "N/A"
    details = movie.select("span.sc-b189961a-8")

    metadata_spans = movie.select("span.sc-d5ea4b9d-7")  # Select all spans
inside the metadata div

    year = metadata_spans[0].text.strip() if len(metadata_spans) > 0 else
"N/A"
    runtime = metadata_spans[1].text.strip() if len(metadata_spans) > 1 else
"N/A"

    rating = movie.select_one("span.ipc-rating-star").text.strip() if
movie.select_one("span.ipc-rating-star") else "N/A"
    votes = movie.select_one("span.ipc-rating-star--
voteCount").text.strip("()") if movie.select_one("span.ipc-rating-star--
voteCount") else "N/A"



    movie_data.append([title, year, rating, runtime, votes])

# Close WebDriver
driver.quit()

# Convert to DataFrame
imdb_df = pd.DataFrame(movie_data, columns=["Title", "Year", "IMDb Rating",
"Runtime", "Votes"])
imdb_df.to_csv("imdb.csv", index=False)
```

```
# Output
print(imdb_df.head())
print(f"Total movies scraped: {imdb_df.shape[0]}")
```

OUTPUT :

```
(base) PS C:\Users\GANESH MAHARAJ K\Desktop\Foundations_ofDS\DA-1> pytho

DevTools listening on ws://127.0.0.1:52368/devtools/browser/a92046b5-885
                        Title Year IMDb Rating Runtime   Votes
0  1. The Shawshank Redemption  N/A    9.3 (3M)     N/A     (3M
1             2. The Godfather  N/A  9.2 (2.1M)     N/A   (2.1M
2           3. The Dark Knight  N/A    9.0 (3M)     N/A     (3M
3     4. The Godfather Part II  N/A  9.0 (1.4M)     N/A   (1.4M
4              5. 12 Angry Men  N/A  9.0 (915K)     N/A   (915K
Total movies scraped: 250
```

PYTHON SCRIPT to web scrap from METACRITIC :

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
import time

# Base URL with page number placeholder
BASE_URL =
"https://www.metacritic.com/browse/movie/?releaseYearMin=1910&releaseYearMax=2025
&page={}"

# Function to extract movie details from a single page
def extract_movies_from_page(url):
    headers = {"User-Agent": "Mozilla/5.0"}  # To prevent getting blocked
    response = requests.get(url, headers=headers)
    if response.status_code != 200:
        print(f"Failed to fetch page: {url}")
        return []

    soup = BeautifulSoup(response.text, 'html.parser')
    movies_data = []

    movies = soup.find_all("div", class_="c-finderProductCard_info")
```

```
    for movie in movies:
        try:
            # Extract Title
            title_tag = movie.find("h3", class_="c-
finderProductCard_titleHeading")
            title = title_tag.find_all("span")[1].text.strip() if title_tag else
"N/A"

            # Extract Year
            year_tag = movie.find("span", class_="u-text-uppercase")
            year = year_tag.text.strip().split()[-1] if year_tag else "N/A"

            # Extract Metascore
            metascore_tag = movie.find("div", class_="c-siteReviewScore")
            metascore = metascore_tag.find("span").text.strip() if metascore_tag
else "N/A"



            mpaa_rating = "N/A"
            for span in movie.find_all("span"):
                if "Rated" in span.text:
                    mpaa_rating = span.text.replace("Rated", "").strip()
                    break  # Stop once found

            # Extract Short Description
            desc_tag = movie.find("div", class_="c-
finderProductCard_description")
            description = desc_tag.text.strip() if desc_tag else "N/A"

            movies_data.append({
                "Title": title,
                "Year": year,
                "Metascore": metascore,
                "MPAA Rating": mpaa_rating,
                "Short Description": description
            })
        except Exception as e:
            print(f"Error extracting data: {e}")
            continue

    return movies_data

# Scrape multiple pages
def scrape_metacritic(pages=5):
```

```
    all_movies = []
    for page in range(1, pages + 1):
        print(f"Scraping page {page}...")
        url = BASE_URL.format(page)
        movies = extract_movies_from_page(url)
        all_movies.extend(movies)
        time.sleep(1)

    # Save data to CSV
    df = pd.DataFrame(all_movies)
    df.to_csv("metacritic_movies.csv", index=False)
    print("Scraping completed. Data saved to 'metacritic_movies.csv'")
    print(df.shape)
    return df

# Run the scraper
scrape_metacritic(pages=691)
```

OUTPUT :

```
(base) PS C:\Users\GANESH MAHARAJ K\Desktop\Foundations_ofDS\DA-
Scraping page 1...
Scraping page 2...
Scraping page 3...
Scraping page 4...
Scraping page 5...
Scraping page 6...
Scraping page 7...
Scraping page 8...
Scraping page 9
```

Converting to DataFrames and Exporting as CSV :

imdb.csv :

imdb1.csv

```
Title,Year,IMDb Rating,Runtime,Votes
1. The Shawshank Redemption,1994,9.3 (3M),2h 22m, (3M
2. The Godfather,1972,9.2 (2.1M),2h 55m, (2.1M
3. The Dark Knight,2008,9.0 (3M),2h 32m, (3M
4. The Godfather Part II,1974,9.0 (1.4M),3h 22m, (1.4M
5. 12 Angry Men,1957,9.0 (914K),1h 36m, (914K
6. The Lord of the Rings: The Return of the King,2003,9.0 (2.1M),3h 21m, (2.1M
7. Schindler's List,1993,9.0 (1.5M),3h 15m, (1.5M
8. Pulp Fiction,1994,8.9 (2.3M),2h 34m, (2.3M
9. The Lord of the Rings: The Fellowship of the Ring,2001,8.9 (2.1M),2h 58m, (2.1M
"10. The Good, the Bad and the Ugly",1966,8.8 (845K),2h 58m, (845K
11. Forrest Gump,1994,8.8 (2.4M),2h 22m, (2.4M
12. The Lord of the Rings: The Two Towers,2002,8.8 (1.9M),2h 59m, (1.9M
13. Fight Club,1999,8.8 (2.4M),2h 19m, (2.4M
14. Inception,2010,8.8 (2.7M),2h 28m, (2.7M
15. Star Wars: Episode V - The Empire Strikes Back,1980,8.7 (1.4M),2h 4m, (1.4M
16. The Matrix,1999,8.7 (2.1M),2h 16m, (2.1M
17. Goodfellas,1990,8.7 (1.3M),2h 25m, (1.3M
18. One Flew Over the Cuckoo's Nest,1975,8.7 (1.1M),2h 13m, (1.1M
19. Interstellar,2014,8.7 (2.3M),2h 49m, (2.3M
20. Se7en,1995,8.6 (1.9M),2h 7m, (1.9M
21. It's a Wonderful Life,1946,8.6 (524K),2h 10m, (524K
```

metacritic_movies.csv :

metacritic_movies1.csv

```
Title,Year,Metascore,MPAA Rating,Short Description
Dekalog (1988),1996,100,"Mar 22, 1996","This masterwork by Krzysztof Kieślowski is one of the twentieth century’s greatest achievements in v
Three Colors: Red,1994,100,"Nov 23, 1994","Krzysztof Kieslowski closes his Three Colors trilogy in grand fashion, with an incandescent medit
The Conformist,1970,100,"Oct 22, 1970","Set in Rome in the 1930s, this re-release of Bernardo Bertolucci's 1970 breakthrough feature stars J
The Godfather,1972,100,"Mar 24, 1972","Francis Ford Coppola's epic features Marlon Brando in his Oscar-winning role as the patriarch of the
Tokyo Story,1972,100,"Mar 13, 1972","Yasujiro Ozu’s Tokyo Story follows an aging couple, Tomi and Sukichi, on their journey from their rural
Citizen Kane,1941,100,"Sep 4, 1941","Following the death of a publishing tycoon, news reporters scramble to discover the meaning of his fina
Rear Window,1954,100,"Sep 1, 1954",A wheelchair-bound photographer spies on his neighbours from his apartment window and becomes convinced o
Casablanca,1943,100,"Jan 23, 1943","A Casablanca, Morocco casino owner in 1941 shelters his former lover and her husband, a Czechoslovakian
The Leopard (re-release),2004,100,"Aug 13, 2004","Set in Sicily in 1860, Luchino Visconti's spectacular 1963 adaptation of Giuseppe di Lampe
Lawrence of Arabia (re-release),2002,100,"Sep 20, 2002","The 40th anniversary re-release of David Lean's 1962 masterpiece, starring Peter O'
Boyhood,2014,100,"Jul 11, 2014","Filmed over 12 years with the same cast, Richard Linklater's Boyhood is a groundbreaking story of growing u
Notorious,1946,100,"Sep 6, 1946",A woman is asked to spy on a group of Nazi friends in South America. How far will she have to go to ingrati
Vertigo,1958,100,"May 28, 1958","Vertigo creates a dizzying web of mistaken identity, passion and murder after an acrophobic detective rescu
Fanny and Alexander (re-release),2004,100,"May 21, 2004","Set in Sweden in the early 20th century, this film focuses on the young children o
Singin' in the Rain,1952,99,"Apr 11, 1952",A silent film production company and cast make a difficult transition to sound.
```

Question-2

2. Data Processing (unstructured to structured)

After collecting raw movie data from IMDb and Metacritic, we needed to clean, transform, and merge the datasets to ensure consistency and usability. Since web-scraped data is often messy and unstructured, we performed several preprocessing steps before storing it in the graph database.

1. Clean and transform raw data into a structured format (handling missing values, standardizing fields).
2. Merge data from both sources into a single structured dataset.

As I did it in jupyterNotebook lets see each cells code and output as screenshot However the code is in GITHUB repo :

Data Processing

i. Clean and transform raw data into a structured format (handling missing values, standardizing fields).

ii. Merge data from both sources into a single structured dataset.

In [2]:
```
import pandas as pd
```

Let's see the data sets

In [3]:
```
imdb_df = pd.read_csv('imdb.csv')
print('shape : ',imdb_df.shape)
imdb_df.head()
```

```
shape :  (250, 5)
```

Out[3]:

	Title	Year	IMDb Rating	Runtime	Votes
0	1. The Shawshank Redemption	1994	9.3 (3M)	2h 22m	(3M
1	2. The Godfather	1972	9.2 (2.1M)	2h 55m	(2.1M
2	3. The Dark Knight	2008	9.0 (3M)	2h 32m	(3M
3	4. The Godfather Part II	1974	9.0 (1.4M)	3h 22m	(1.4M
4	5. 12 Angry Men	1957	9.0 (914K)	1h 36m	(914K

In [4]:
```
mc_df = pd.read_csv('metacritic_movies.csv')
print('shape : ',mc_df.shape)
mc_df.head()
```

```
shape :  (16584, 5)
```

Out[4]:

	Title	Year	Metascore	MPAA Rating	Short Description
0	Dekalog (1988)	1996.0	100.0	TV-MA	This masterwork by Krzysztof Kieślowski is one...
1	The Conformist	1970.0	100.0	R	Set in Rome in the 1930s, this re-release of B...
2	The Leopard (re-release)	2004.0	100.0	PG	Set in Sicily in 1860, Luchino Visconti's spec...

In [5]:
```
# checking if there are any null values
print('imdb null values : ',imdb_df.isnull().sum())
print('\n\nmetacritic null values : ',mc_df.isnull().sum())
```

```
imdb null values :  Title          0
Year           0
IMDb Rating    0
Runtime        0
Votes          0
dtype: int64


metacritic null values :  Title                  0
Year                   123
Metascore                9
MPAA Rating           1556
Short Description        0
dtype: int64
```

In [6]:
```
percent_of_na = mc_df.isnull().sum().sum() * 100 / mc_df.shape[0]
percent_of_na
```

Out[6]:
```
np.float64(10.178485287023637)
```

We have no null values in IMDB dataset, and some nulls in metacritic

We drop the records of null values as generally in large datasets we can drop null records <=10

In [7]:
```
mc_df = mc_df.dropna()
mc_df = mc_df.reset_index(drop=True)

print("Remaining records:", mc_df.shape)
```

```
Remaining records: (15005, 5)
```

Data Cleaning and Standardizing of **IMDB** DATASET

Now let's clean IMDB dataset columns

In [8]:
```
imdb_df.head()
```

Out[8]:

	Title	Year	IMDb Rating	Runtime	Votes
0	1. The Shawshank Redemption	1994	9.3 (3M)	2h 22m	(3M
1	2. The Godfather	1972	9.2 (2.1M)	2h 55m	(2.1M
2	3. The Dark Knight	2008	9.0 (3M)	2h 32m	(3M
3	4. The Godfather Part II	1974	9.0 (1.4M)	3h 22m	(1.4M
4	5. 12 Angry Men	1957	9.0 (914K)	1h 36m	(914K

1. in **Title** column we see it has numbers, So we strip by '.' then substr name and strip leading spaces.
2. in **IMDb Rating** column we see it has vote included in the rating so we remove it by stripping it or by just substr of first 3 charecters.
3. in **Votes** column in has '(' we need to remove it.

In [9]:
```
# Title column cleaning
imdb_df["Title"]=imdb_df["Title"].str.split('.', n=1).str[1].str.strip()
imdb_df.head()
```

Out[9]:

	Title	Year	IMDb Rating	Runtime	Votes
0	The Shawshank Redemption	1994	9.3 (3M)	2h 22m	(3M
1	The Godfather	1972	9.2 (2.1M)	2h 55m	(2.1M
2	The Dark Knight	2008	9.0 (3M)	2h 32m	(3M
3	The Godfather Part II	1974	9.0 (1.4M)	3h 22m	(1.4M
4	12 Angry Men	1957	9.0 (914K)	1h 36m	(914K

In [10]:
```
# IMDb Rating cleaning
imdb_df["IMDb Rating"] = imdb_df["IMDb Rating"].str.split().str[0]
imdb_df.head()
```

Out[10]:

	Title	Year	IMDb Rating	Runtime	Votes
0	The Shawshank Redemption	1994	9.3	2h 22m	(3M
1	The Godfather	1972	9.2	2h 55m	(2.1M
2	The Dark Knight	2008	9.0	2h 32m	(3M
3	The Godfather Part II	1974	9.0	3h 22m	(1.4M
4	12 Angry Men	1957	9.0	1h 36m	(914K

In [11]:
```
# Votes cleaning
imdb_df["Votes"] = imdb_df["Votes"].str.split('(').str[1]
imdb_df.head()
```

Out[11]:

	Title	Year	IMDb Rating	Runtime	Votes
0	The Shawshank Redemption	1994	9.3	2h 22m	3M
1	The Godfather	1972	9.2	2h 55m	2.1M
2	The Dark Knight	2008	9.0	2h 32m	3M

Now lets Standardize the imbd dataset columns

In [12]:
```
imdb_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250 entries, 0 to 249
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Title        250 non-null    object
 1   Year         250 non-null    int64 
 2   IMDb Rating  250 non-null    object
 3   Runtime      250 non-null    object
 4   Votes        250 non-null    object
dtypes: int64(1), object(4)
memory usage: 9.9+ KB
```

In [13]:
```
imdb_df['Votes'].str[-1].value_counts()
```

Out[13]:
```
Votes
K    182
M     68
Name: count, dtype: int64
```

1. the **IMDb Rating** column is of type object we convert it to float.
2. in **Runtime** column, we standardize the value to minutes and its type to int.
3. in **Votes** column, we have 'M','K' we make it as number and convert it's type to int

In [14]:
```
# IMDb Rating column
imdb_df["IMDb Rating"] = imdb_df["IMDb Rating"].astype(float)
imdb_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250 entries, 0 to 249
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Title        250 non-null    object 
 1   Year         250 non-null    int64  
 2   IMDb Rating  250 non-null    float64
 3   Runtime      250 non-null    object 
 4   Votes        250 non-null    object 
```

In [15]:
```
# Runtime column
def convert_to_minutes(runtime):
    if pd.isna(runtime):
        return None  # Handle missing values
    time_parts = runtime.split()
    minutes = 0
    for part in time_parts:
        if 'h' in part:
            minutes += int(part.replace('h', '')) * 60  # Convert hours to minutes
        elif 'm' in part:
            minutes += int(part.replace('m', ''))  # Add minutes
    return minutes

imdb_df["Runtime"] = imdb_df["Runtime"].apply(convert_to_minutes)
print(imdb_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250 entries, 0 to 249
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Title        250 non-null    object
 1   Year         250 non-null    int64 
 2   IMDb Rating  250 non-null    float64
 3   Runtime      250 non-null    int64 
 4   Votes        250 non-null    object
dtypes: float64(1), int64(2), object(2)
memory usage: 9.9+ KB
None
```

In [16]:
```
# Votes column
def convert_votes(vote):
    if pd.isna(vote):
        return None  # Handle missing values
    vote = vote.upper().replace(",", "")  # Remove commas if present
    if "M" in vote:
        return int(float(vote.replace("M", "")) * 1000000)
    elif "K" in vote:
        return int(float(vote.replace("K", "")) * 1000)
    return int(vote)  # If already a number

imdb_df["Votes"] = imdb_df["Votes"].apply(convert_votes)
```

In [17]:
```
imdb_df.head()
```

Out[17]:

	Title	Year	IMDb Rating	Runtime	Votes
0	The Shawshank Redemption	1994	9.3	142	3000000
1	The Godfather	1972	9.2	175	2100000
2	The Dark Knight	2008	9.0	152	3000000
3	The Godfather Part II	1974	9.0	202	1400000
4	12 Angry Men	1957	9.0	96	914000

In [18]:
```
imdb_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250 entries, 0 to 249
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Title        250 non-null    object
 1   Year         250 non-null    int64 
 2   IMDb Rating  250 non-null    float64
 3   Runtime      250 non-null    int64 
 4   Votes        250 non-null    int64 
dtypes: float64(1), int64(3), object(1)
memory usage: 9.9+ KB
```

Data Cleaning and Standardizing of METACRITIC DATASET

METACRITIC dataset columns looks clean, need to check data types

In [19]:
```
mc_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15005 entries, 0 to 15004
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Title              15005 non-null  object 
 1   Year               15005 non-null  float64
 2   Metascore          15005 non-null  float64
 3   MPAA Rating        15005 non-null  object 
 4   Short Description  15005 non-null  object 
dtypes: float64(2), object(3)
memory usage: 586.3+ KB
```

In [20]:
```
# convert year to int
mc_df['Year']= mc_df['Year'].astype(int)
```

In [21]:
```
mc_df.head()
```

Out[21]:

	Title	Year	Metascore	MPAA Rating	Short Description
0	Dekalog (1988)	1996	100.0	TV-MA	This masterwork by Krzysztof Kieślowski is one...
1	The Conformist	1970	100.0	R	Set in Rome in the 1930s, this re-release of B...
2	The Leopard (re-release)	2004	100.0	PG	Set in Sicily in 1860, Luchino Visconti's spec...
3	The Godfather	1972	100.0	TV-14	Francis Ford Coppola's epic features Marlon Br...
4	Lawrence of Arabia (re-release)	2002	100.0	Approved	The 40th anniversary re-release of David Lean'...

Merge data from both datasets

we will use 'INNER' join as it would be helpful to have common datapoints and we can worked further

In [22]:
```
merged_df = imdb_df.merge(mc_df, on="Title", how="inner",suffixes=('_imdb', '_mc'))
print("Merged dataset shape:", merged_df.shape)

merged_df.head()
```

```
Merged dataset shape: (238, 9)
```

Out[22]:

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

In [23]:
```
# clean merged_df
# we can remove any of duplicate columns YEAR_imdb, YEAR_mc and rename it to YEAR
merged_df.drop(columns=['Year_mc'], inplace=True)
merged_df.rename(columns={'Year_imdb': 'YEAR'}, inplace=True)
```

In [24]:
```
merged_df.head()
```

Out[24]:

	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...

In [25]:
```
merged_df.info()
```

```
<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
```

In [26]:
```
merged_df.to_csv("merged_movies.csv", index=False)
```

With exporting Dataframe of merged dataset to CSV the Data Processing step is completed.

Question-3

3. Storing Data in a Graph Database

After processing and merging the movie data, the next step was to **store it in a graph database** for efficient querying and analysis. We used **Neo4j**, a powerful graph database, to model movies, their attributes, and relationships.

i. Choosing a Graph Database: Neo4j

Neo4j was chosen because :

- It efficiently **models relationships** between entities (e.g., movies, directors, ratings).
- It allows **flexible querying** using **Cypher Query Language (CQL)**.
- It provides **fast traversal of connected data**, making it ideal for analyzing movie relationships.

ii. Defining the Graph Schema

A well-structured schema was designed to store movies and their relationships effectively. The schema consists of **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.

Then we insert the Data into the Database.

3.Storing Data in a Graph Database

i. Choose a graph database such as Neo4j.

ii. Define an appropriate graph schema (nodes, edges, relationships).

iii. Insert structured data into the database using Py2neo (Python)

In [1]:
```
from py2neo import Graph, Node, Relationship
import pandas as pd
```

In [2]:
```
# Connecting to Neo4j database
graph = Graph("bolt://localhost:7687", auth=("neo4j", "moviesgraphdb"))
```

In [3]:
```
# Clear existing data (optional, for fresh start)
graph.run("MATCH (n) DETACH DELETE n")
```

Out[3]: (No data)

In [4]:
```
merged_df=pd.read_csv('merged_movies.csv')
merged_df.head()
```

Out[4]:

	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...

In [13]:
```
print(merged_df.columns)
print(merged_df[['Title', 'YEAR', 'MPAA Rating']].head(10))
print(merged_df.isnull().sum())
```

```
Index(['Title', 'YEAR', 'IMDb Rating', 'Runtime', 'Votes', 'Metascore',
       'MPAA Rating', 'Short Description'],
      dtype='object')
                                          Title  YEAR MPAA Rating
0                         The Shawshank Redemption  1994       TV-14
1                                  The Godfather  1972       TV-14
2                                The Dark Knight  2008       TV-14
3                                    12 Angry Men  1957    Approved
4      The Lord of the Rings: The Return of the King  2003       PG-13
5                                Schindler's List  1993           R
6                                    Pulp Fiction  1994       TV-14
7  The Lord of the Rings: The Fellowship of the Ring  2001       PG-13
8                    The Good, the Bad and the Ugly  1966       TV-14
9                                    Forrest Gump  1994       TV-PG
Title               0
YEAR                0
IMDb Rating         0
Runtime             0
Votes               0
Metascore           0
MPAA Rating         0
Short Description   0
dtype: int64
```

In [5]:
```
# Insert data into Neo4j
for _, row in merged_df.iterrows():
    # Create or match Movie node
    movie = Node(
        "Movie",
        Title=row["Title"],
        Year=row["YEAR"],
        IMDB_Rating=row["IMDb Rating"],
        Runtime=row["Runtime"],
        Votes=row["Votes"],
        Metascore=row["Metascore"],
        Short_Description=row["Short Description"]
    )
    graph.merge(movie, "Movie", "Title")

    # Create or match MPAA Rating node
    mpaa_rating = Node("MPAA_Rating", Rating=row["MPAA Rating"])
    graph.merge(mpaa_rating, "MPAA_Rating", "Rating")

    # Create relationship (Movie)-[:HAS_RATING]->(MPAA_Rating)
    has_rating = Relationship(movie, "HAS_RATING", mpaa_rating)
    graph.merge(has_rating)
```

Movie Nodes → Contain movie details like Title, Year, IMDb Rating, Runtime, Votes, Metascore, Short Description.

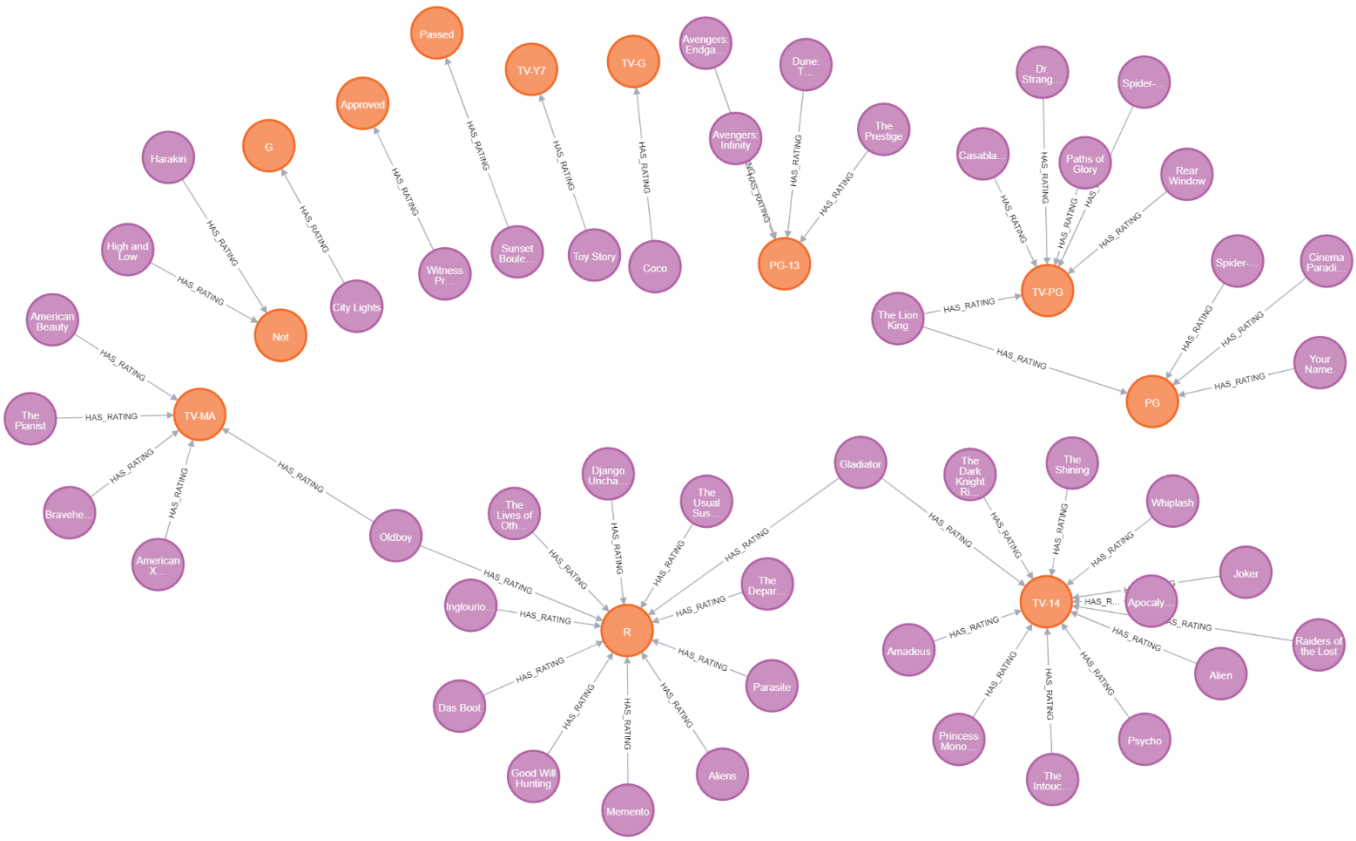
MPAA_Rating Nodes → Unique rating categories (G, PG, PG-13, R, etc.).

HAS_RATING Relationship → Links each Movie node to its respective MPAA_Rating node.

In [35]:

```
import webbrowser
from pyvis.network import Network
from py2neo import Graph

def visualize_graph():
    # Create Pyvis network (notebook=True for Jupyter)
    net = Network(notebook=True, directed=True, cdn_resources="in_line")

    # Query to fetch nodes & relationships
    query = """
    MATCH (m:Movie)-[r]->(n)
    RETURN m.Title AS Movie, type(r) AS Relationship, labels(n)[0] AS NodeType,
           COALESCE(n.Title, n.name, 'Unknown') AS ConnectedNode
    """
    result = graph.run(query).data()

    # Add nodes and edges to Pyvis network
    for record in result:
        movie = record["Movie"]
        connected_node = record["ConnectedNode"]
        relationship = record["Relationship"]

        net.add_node(movie, label=movie, color="blue")  # Movie nodes in blue
        net.add_node(connected_node, label=connected_node, color="green")  # Other nodes in green
        net.add_edge(movie, connected_node, label=relationship)

    # Save and open the graph
    output_file = "graph.html"
    net.show(output_file)
    webbrowser.open(output_file)  # Open in browser automatically

# Call the function to visualize the graph
visualize_graph()
```

graph.html



Question-4

4. Storing Data in a Graph Database

After successfully storing the structured movie data in **Neo4j**, the next step is to **query the database using Cypher** to gain insights. Unlike traditional relational databases, **Neo4j allows us to explore complex relationships between entities efficiently**, making it ideal for analyzing interconnected data such as movies, directors, genres, and ratings.

The goal of this step is to:

- Retrieve meaningful insights from the data.
- Analyze relationships between movies, genres, and directors.
- Identify patterns, such as **top-rated movies**, **most prolific directors**, and **movies from a specific time period**.

Cypher, the query language for Neo4j, enables us to express queries intuitively using **graph traversal techniques**, making it powerful for retrieving structured and unstructured relationships.

In this section, we formulate queries to answer key questions such as:

- What relationships exist between movies, directors and ratings?
- Which movies were released in a specific year?
- What are the highest-rated movies based on IMDb and Metascore?
- Which directors have worked on multiple films?

4. Querying the Graph Database

Write at least three queries using Cypher (Neo4j) to retrieve insights.

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

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...

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

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. 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

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

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