DEPARTMENT OF COMPUTER SCIENCE AND

ENGINEERING

COLLEGE OF ENGINEERING, GUINDY

ANNA UNIVERSITY

CS

DATA MINING

PROJECT REPORT

MOVIE RECOMMENDATION SYSTEM BY APRIORI ALGORITHM

-BY

AISHWARYA S (2022103037)

SUJANA S (2022103607)

TABLE OF CONTENTS :

Introduction:

Our project implements a **Movie Recommendation System** using the Apriori Algorithm to identify patterns in user movie ratings. Using the **Netflix Prize Dataset,** we analyze movie associations to generate personalized recommendations. Key steps include data preprocessing, frequent itemset generation, and association rule mining, ensuring an efficient and scalable recommendation system.

Objective:

➔ Implement a movie recommendation system using **Apriori Algorithm and Association Rule Mining (ARM)** for pattern extraction.

➔ Preprocess the dataset by cleaning missing values, transforming data into a sparse matrix, and creating a binary matrix for user-movie interactions.

➔**Generate frequent itemsets** (L1, L2, etc.) using the **Apriori property** to identify commonly watched movie combinations.

➔ **Derive association rules** based on **support, confidence metrics** to determine strong movie correlations.

➔Deliver personalized **recommendations** by analyzing user watch history and suggesting movies based on extracted rules.

PROBLEM STATEMENT:

- Users struggle to find relevant movies due to the overwhelming number of choices and the lack of personalized recommendations. Existing methods rely on general trends rather than individual viewing habits, leading to less accurate suggestions. This project addresses these challenges using **Apriori Algorithm and Association Rule Mining (ARM)** to analyze user-watching patterns, generate frequent itemsets, and derive strong association rules based on **support, confidence, and lift**.

- By implementing data preprocessing, sparse matrix representation, and optimized rule selection, the system ensures efficient, personalized, and data-driven movie recommendations.

ABSTRACT:

❖ With the rapid growth of digital streaming platforms, users often struggle to find relevant movies due to the overwhelming number of choices. Traditional recommendation methods, such as collaborative and content-based filtering, may not effectively capture hidden relationships between movies. This project addresses these challenges using Association Rule Mining (ARM) and the Apriori algorithm to analyze user viewing patterns and provide personalized recommendations. The process begins with data preprocessing, including cleaning the dataset, removing **NaN** values, and transforming it into a binary sparse matrix for efficient storage.

❖ Handling a **2GB dataset** in Apriori leads to high RAM usage, slow processing, and potential memory errors due to the exponential growth of itemsets. Converting data into a **binary matrix** worsens memory consumption, making computations inefficient. Optimizing with batch processing, early pruning, and sparse structures is crucial.

❖ The Apriori algorithm is then applied to extract **frequent itemsets (L1, L2)** and generate association rules based on support, confidence, and lift metrics. These rules help predict movie preferences and recommend relevant content to users. The system is evaluated on a real-world movie dataset to measure its accuracy and effectiveness.

❖ Applications of this approach extend beyond movie recommendations to streaming platforms (Netflix, Prime Video), retail product recommendations, and consumer behavior analysis. By optimizing computational efficiency and refining rule selection, this method ensures scalable, data-driven, and highly personalized recommendations, enhancing user experience and content discovery

MODULE DESCRIPTION:

1. **Data Preprocessing Module**

- **Objective:** Prepare the dataset for efficient processing.

- **Steps:**
  - Load the real-world movie dataset (e.g., Netflix dataset).
  - Clean the data by **removing NaN values** and handling missing entries.
- **Technical Issues Solved:** Data inconsistency, high memory consumption, and missing values.

2.**Binary Matrix Generation Module**:

Objective: Convert user-movie interaction data into a **binary sparse matrix** for efficient processing in the Apriori algorithm.
Steps:

- Transform the dataset into a user-movie matrix, where watched movies are marked as 1 and unwatched movies as 0.
- Encode interactions in a binary format to facilitate frequent itemset mining.
- Store data in a sparse matrix representation (e.g., Compressed Sparse Row - CSR) to optimize memory usage.
- Ensure fast retrieval and updates for efficient rule generation in the Apriori process.

Technical Issues Solved: Reducing memory consumption, improving data retrieval speed, and enabling efficient frequent itemset mining.

3.Frequent Itemset Generation Module

- **Objective:** Identify commonly watched movie combinations.
- **Steps:**
  - Apply the Apriori algorithm to generate Frequent Itemset 1 (L1) (single movies frequently watched).
  - Extend to Frequent Itemset 2 (L2) and higher by finding frequently co-watched movies.
- **Technical Issues Solved:** Reducing computational complexity by eliminating low-support itemsets.

4. **Association Rule Mining Module**

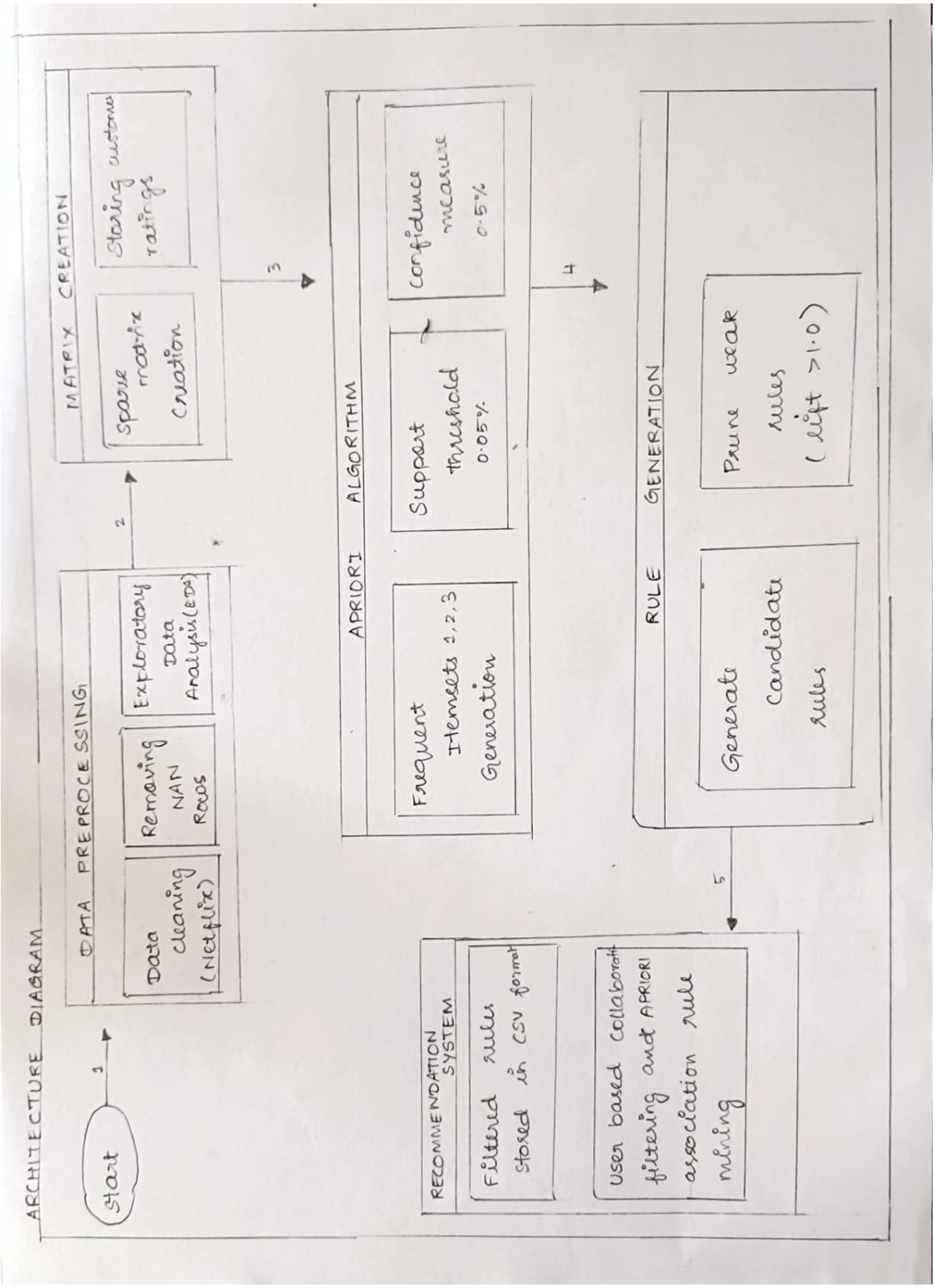
- **Objective:** Derive relationships between movies for recommendations.
- **Steps:**
  - Extract **association rules** from frequent itemsets.
  - Compute **support, confidence, and lift** to determine strong rules.

- **Technical Issues Solved:** Improving recommendation accuracy by eliminating irrelevant associations.

5. **Recommendation Generation Module**

- **Objective:** Suggest relevant movies based on extracted rules.
- **Steps:**
  - Identify movies that a user has watched.
  - Use **strong association rules** to recommend new movies.
  -
- **Technical Issues Solved:** Personalizing suggestions based on user history rather than generic trends

ARCHITECTURE DIAGRAM:



Dataset Information:

The Netflix Prize Dataset is a large dataset released by Netflix for a competition aimed at improving their recommendation system. It contains user-movie rating data spanning several years.

Dataset Files and Their Fields:

1. **combined_data_1.txt, combined_data_2.txt, combined_data_3.txt, combined_data_4.txt**

- These four files contain the bulk of the rating data in the following format:
- MovieID:
- UserID, Rating, Date
- Fields:
  - MovieID (integer) → A unique ID representing a movie.
  - UserID (integer) → A unique ID representing a user.
  - Rating (integer: 1-5) → The rating given by the user.
  - Date (YYYY-MM-DD) → The date when the rating was given.

2. **movie_titles.csv**

- Contains metadata about movies.
- **Fields:**
  - **MovieID** (integer) → The unique movie ID.
  - **Year of Release** (integer) → The release year of the movie.
  - **Title** (string) → The title of the movie.

3. **qualifying.txt**

- Contains user-movie pairs for which ratings need to be predicted.
- **Fields:**
  - **MovieID**
  - **UserID**

The dataset contains over 100 million ratings from 480,000 users for 17,770 movies spanning from 1998 to 2005. The Netflix Prize Dataset is ~2GB in total. This dataset is used for collaborative filtering and recommendation system research.

LINK: https://www.kaggle.com/datasets/netflix-inc/netflix-prize-data

Exploratory Data Analysis (EDA) :

EDA (Exploratory Data Analysis) is the process of analyzing and visualizing a dataset to understand its characteristics, detect patterns, and identify anomalies before applying machine learning or statistical modeling.

1. Data Summary & Structure
2. Data Cleaning & Handling Missing Value
3.Data Distribution & Trends
4.Feature Relationships & Correlations
5. Categorical vs. Numerical Analysis

[1]:

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[3]:

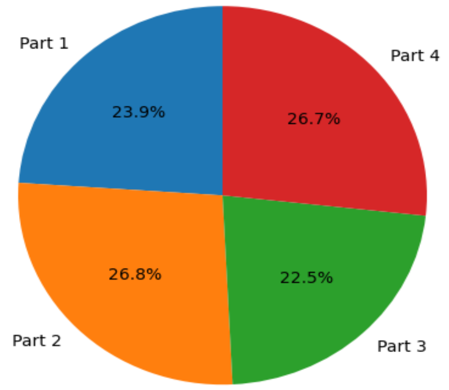
```
# load all the individual data files
data1 = pd.read_csv('combined_data_1.txt', header = None, names = ['Cust_Id', 'Rating'], usecols = [0,1]) # 24058263
data2 = pd.read_csv('combined_data_2.txt', header = None, names = ['Cust_Id', 'Rating'], usecols = [0,1]) # 26982302
data3 = pd.read_csv('combined_data_3.txt', header = None, names = ['Cust_Id', 'Rating'], usecols = [0,1]) # 22605786
data4 = pd.read_csv('combined_data_4.txt', header = None, names = ['Cust_Id', 'Rating'], usecols = [0,1]) # 26851926
```

- Importing Required Libraries.
- Loading Multiple Data Files
- Comments Indicating Data Size

PIE CHART REPRESENTATION:

Creates a pie chart to show the proportion of four data files.
-Labels each part (Part 1 to Part 4).
- Uses dataset sizes (len(dataX)) for distribution.
- Displays percentages and ensures a circular shape.

```
[4]: labels = ['Part 1', 'Part 2', 'Part 3', 'Part 4']

# colors = ['blue', 'yellow', 'green', 'orange']

sizes= [len(data1), len(data2), len(data3), len(data4)]

plt.pie(sizes,labels=labels, startangle=90, autopct='%1.1f%%')

plt.axis('equal')

plt.show()
```



DATA1.INFO():

```
[8]: data1.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24058263 entries, 0 to 24058262
Data columns (total 2 columns):
 #   Column   Dtype
---  ------   -----
 0   Cust_Id  object
 1   Rating   float64
dtypes: float64(1), object(1)
memory usage: 367.1+ MB
```

```
[9]: data1.isnull().sum()

[9]: Cust_Id      0
     Rating    4499
     dtype: int64
```

- Displays metadata about the data1 DataFrame.
- Shows total entries (24,058,263 rows).
- Lists column names (Cust_Id, Rating).
- Indicates data types (Cust_Id as object, Rating as float64).
- Provides memory usage (367.1 MB).

REMOVING DUP RECORDS:

```
•[16]: # remove duplicate records
       dedup_data1 = data1.drop_duplicates(ignore_index=True) # 24058262 - 11889558 = 12168704
       dedup_data1
```

[16]:

	Cust_Id	Rating
0	1:	NaN
1	1488844	3.0
2	822109	5.0
3	885013	4.0
4	30878	4.0
...	...	...
1824050	438990	1.0
1824051	1498503	2.0
1824052	2541000	1.0
1824053	1999441	2.0
1824054	2368103	2.0

1824055 rows × 2 columns

- Removes duplicate rows from data1.
- Keeps only the first occurrence of each duplicate entry.
- The original dataset (data1) had 24,058,262 records.
- 11,889,558 duplicate records were removed.
- The new dataset (dedup_data1) contains 12,168,704 unique records.

COMBINING MULTIPLE DATASET:

- The combined dataset has 100,498,277 rows (sum of individual dataset sizes).
- Removing Duplicates:
- drop_duplicates(ignore_index=True) removes duplicate rows.
- The deduplicated dataset has 22,166,698 unique rows, meaning 78,331,579 duplicate records were removed.

```
[17]: ls_data = [data1, data2, data3, data4] # 24058263 + 26982302 + 22605786 + 26851926 = 100498277

combined_data = pd.concat(ls_data, ignore_index=True)
combined_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100498277 entries, 0 to 100498276
Data columns (total 2 columns):
 #   Column   Dtype
---  ------   -----
 0   Cust_Id  object
 1   Rating   float64
dtypes: float64(1), object(1)
memory usage: 1.5+ GB
```

```
[18]: # remove duplicate records from combined dataset.  ---check if duplicates should not be remove?
dedup_combined_data = combined_data.drop_duplicates(ignore_index=True) # 2216698
dedup_combined_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2216698 entries, 0 to 2216697
Data columns (total 2 columns):
 #   Column   Dtype
---  ------   -----
 0   Cust_Id  object
 1   Rating   float64
dtypes: float64(1), object(1)
memory usage: 33.8+ MB
```

MOVIE ID COLUMN CREATION:

```
•[22]: # Create list of movie_id's using starting & ending index and fill the values accordingly

ls_movie_id = []
fill_movieid_value = 1

for i,j in zip(index_of_movieids['index'][1:],index_of_movieids['index'][:-1]):
    x = np.full((1,i-j-1), fill_movieid_value)
    ls_movie_id = np.append(ls_movie_id, x)
    fill_movieid_value += 1
last_row = np.full((1,len(dedup_combined_data) - index_of_movieids.iloc[-1, 0] - 1), fill_movieid_value)
ls_movie_id = np.append(ls_movie_id, last_row)
len(ls_movie_id) # Should be equal to Number of rows in combined dataset
```

```
[22]: 2198928
```

```
[23]: # drop the rows having NaN or Null values
dedup_NotNull_combined_data = dedup_combined_data.copy()
dedup_NotNull_combined_data = dedup_NotNull_combined_data.dropna() # drop nan values # 2216694 - 17770 = 2198928
dedup_NotNull_combined_data = dedup_NotNull_combined_data.reset_index(drop = True) # reset the indices

# add Movie_Id column with values derived previously
customer_ratings_data = dedup_NotNull_combined_data.copy()
customer_ratings_data['Movie_Id'] = ls_movie_id

# change datatypes of ID columns
# customer_ratings_data[['Cust_Id', 'Movie_Id']] = customer_ratings_data[['Cust_Id', 'Movie_Id']].apply(pd.to_numeric)
convert_datatypes = {'Cust_Id': int,'Movie_Id': int}
customer_ratings_data = customer_ratings_data.astype(convert_datatypes)
customer_ratings_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2198928 entries, 0 to 2198927
Data columns (total 3 columns):
 #   Column    Dtype
---  ------    -----
 0   Cust_Id   int32
 1   Rating    float64
 2   Movie_Id  int32
dtypes: float64(1), int32(2)
memory usage: 33.6 MB
```

Removes NaN rows and resets the index.

Adds Movie_ID to the cleaned dataset.

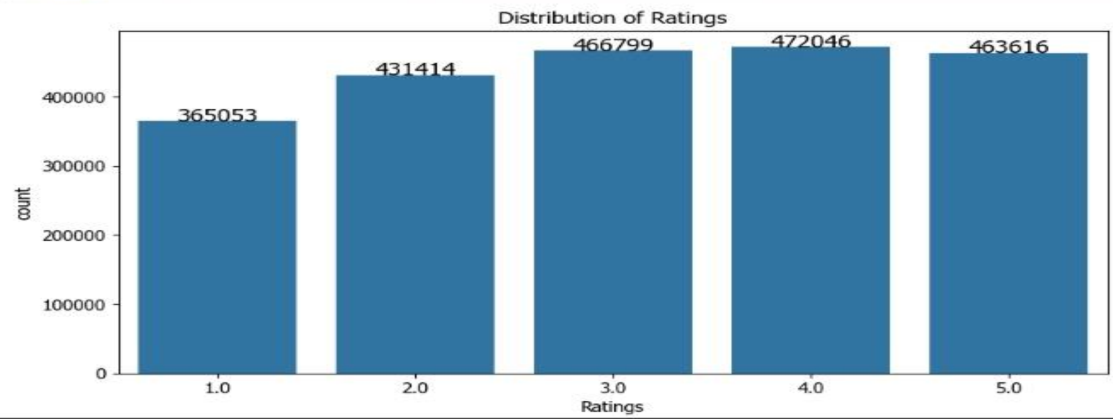
Converts data types for efficiency.

Displays dataset info (3 columns: Cust_Id, Movie_Id, Rating).

CHECK MISSING VALUES:

Checks for missing values in customer_ratings_data (none found).

```
[24]: # check for null values
customer_ratings_data.isna().sum()

[24]: Cust_Id     0
Rating      0
Movie_Id    0
dtype: int64

[25]: # Unique values in dataset
customer_ratings_data.nunique()

[25]: Cust_Id     480189
Rating           5
Movie_Id     15283
dtype: int64

[26]: # No of records for each ratings
import seaborn as sns
plt.figure(figsize=(10,5))
ax = sns.countplot(data=customer_ratings_data, x='Rating')
labels = (customer_ratings_data['Rating'].value_counts().sort_index())
plt.title('Distribution of Ratings')
plt.xlabel('Ratings')

for i,v in enumerate(labels):
    ax.text(i, v+100, str(v), horizontalalignment='center', size=14, color='black')
plt.show()
```



Loading Movie Titles Dataset

The code reads the movie_title.csv file into a Pandas DataFrame named Movie_lookup. It loads three columns: Movie_Id (unique identifier), Year (release year), and Movie_Name (title). The ISO-8859-1 encoding is used to handle special characters, and only the first three columns are read using usecols=[0,1,2]. The dataset contains 17,770 movies, with some missing or incorrect year values (e.g., NaN).

```
[28]: import pandas as pd
      Movie_lookup = pd.read_csv('movie_title.csv', header = None, names = ['Movie_Id', 'Year', 'Movie_Name'], encoding='ISO-8859-1', usecols=[0,1,2], lineter
      Movie_lookup
```

[28]:

	Movie_Id	Year	Movie_Name
0	1	2003.0	Dinosaur Planet
1	2	2004.0	Isle of Man TT 2004 Review
2	3	1997.0	Character
3	4	1994.0	Paula Abdul's Get Up & Dance
4	5	2004.0	The Rise and Fall of ECW
...	...	...	...
17765	17766	2002.0	Where the Wild Things Are and Other Maurice Se...
17766	17767	2004.0	Fidel Castro: American Experience
17767	17768	2000.0	Epoch
17768	17769	2003.0	The Company
17769	17770	2003.0	Alien Hunter

17770 rows × 3 columns

```
[29]: Movie_lookup.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17770 entries, 0 to 17769
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Movie_Id    17770 non-null  int64
 1   Year        17763 non-null  float64
 2   Movie_Name  17770 non-null  object
dtypes: float64(1), int64(1), object(1)
memory usage: 416.6+ KB
```

DATA CLEANING:

Check for missing values using .isna().sum(), revealing 7 missing values in the Year column.

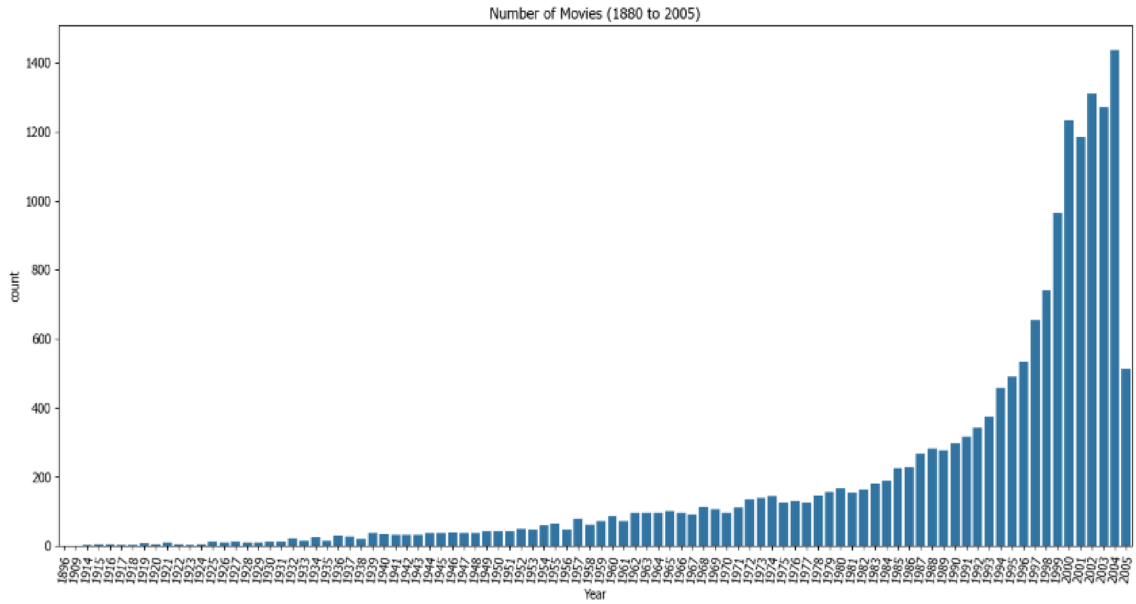
Replace missing values in Year with 0 using .fillna(0), ensuring there are no more NaN values.

```
[30]: # check for NaN/Null values
      Movie_lookup.isna().sum()

[30]: Movie_Id      0
      Year          7
      Movie_Name    0
      dtype: int64

[31]: # replace NaN/Null with 0
      Movie_lookup['Year'] = Movie_lookup['Year'].fillna(0)
      # check for null values after converting nan/null to 0
      Movie_lookup.isna().sum()

[31]: Movie_Id      0
      Year          0
      Movie_Name    0
      dtype: int64

[32]:
      # change datatype for year
      Movie_lookup['Year'] = Movie_lookup['Year'].astype(int)
      Movie_lookup.info()

      <class 'pandas.core.frame.DataFrame'>
      RangeIndex: 17770 entries, 0 to 17769
      Data columns (total 3 columns):
       #   Column      Non-Null Count  Dtype
      ---  ------      --------------  -----
       0   Movie_Id    17770 non-null  int64
       1   Year        17770 non-null  int32
       2   Movie_Name  17770 non-null  object
      dtypes: int32(1), int64(1), object(1)
      memory usage: 347.2+ KB
```

Visualization of Movie Releases (1880-2005)

The code generates a countplot using Seaborn to display the number of movies released per year from 1880 to 2005. It filters out missing data (Year != 0) and rotates x-axis labels for readability. The plot shows a gradual rise in movie production, with a significant surge after the 1980s, peaking in the early 2000s. However, the x-axis labels appear cluttered due to overlapping year values.

```python
# Number of movies released per year
import seaborn as sns
plt.figure(figsize=(18,8))
sns.countplot(x ='Year', data = Movie_lookup[Movie_lookup['Year'] != 0])
plt.xticks(rotation = 80)
current_values = plt.gca().get_xticks()
# plt.gca().set_xticklabels(['{:.0f}'.format(x) for x in current_values])
plt.title('Number of Movies (1880 to 2005)')
plt.show()
```



IMPLEMENTATION OF APRIORI ALGORITHM :

The Apriori Algorithm operates through a systematic process that involves several key steps:

1. Identifying Frequent Itemsets: The algorithm begins by scanning the dataset to identify individual items (1-item) and their frequencies. It then establishes a minimum support threshold, which determines whether an itemset is considered frequent.
2. Creating Possible item group: Once frequent 1-itemgroup(single items) are identified, the algorithm generates candidate 2-itemgroup by combining frequent items. This process continues iteratively, forming larger itemsets (k-itemgroup) until no more frequent itemgroup can be found.
3. Removing Infrequent Item groups: The algorithm employs a pruning technique based on the Apriori Property, which states that if an itemset is infrequent, all its supersets must also be infrequent. This significantly reduces the number of combinations that need to be evaluated.

4. Generating Association Rules: After identifying frequent itemsets, the algorithm generates association rules that illustrate how items relate to one another, using metrics like support, confidence, and lift to evaluate the strength of these relationships.

1.COMBINED_1.TXT:

[2]:
```python
import pandas as pd
import numpy as np
import math
import re
import matplotlib.pyplot as plt
```

[3]:
```python
data = pd.read_csv('combined_data_2.txt', header = None, names = ['Cust_Id', 'Rating'], usecols = [0,1])
data['Rating'] = data['Rating'].astype(float)
print('Data shape: {}'.format(data.shape))
print(data.iloc[::5000000, :])
```

```
Data shape: (26982302, 2)
          Cust_Id  Rating
0           4500:     NaN
5000000    485565     2.0
10000000  1155911     2.0
15000000   121369     3.0
20000000  1277779     3.0
25000000   252632     5.0
```

- Loading Netflix Ratings Data
- Reads combined_data_2.txt into a Pandas DataFrame.
- Loads two columns:
  - Cust_Id → Represents the user ID.
  - Rating → Represents the movie rating (converted to float).
- Data shape: (26,982,302, 2) (over 26 million rows).
- Some rows contain NaN values, likely representing Movie IDs separating rating groups.
- Uses print(data.iloc[::5000000, :]) to display sample rows at every 5 millionth interval.

2.IDENTIFYING NAN ROWS:

```python
[4]: merge_dataset_nan = pd.DataFrame(pd.isnull(data.Rating))
merge_dataset_nan = merge_dataset_nan[merge_dataset_nan['Rating'] == True]
merge_dataset_nan = merge_dataset_nan.reset_index()

movie_np = []
movie_id = 1

for i,j in zip(merge_dataset_nan['index'][1:],merge_dataset_nan['index'][:-1]):
    # numpy approach
    temp = np.full((1,i-j-1), movie_id)
    movie_np = np.append(movie_np, temp)
    movie_id += 1

# Account for last record and corresponding length
# numpy approach
last_record = np.full((1,len(data) - merge_dataset_nan.iloc[-1, 0] - 1),movie_id)
movie_np = np.append(movie_np, last_record)

print('Movie numpy: {}'.format(movie_np))
print('Length: {}'.format(len(movie_np)))
```

```
Movie numpy: [1.000e+00 1.000e+00 1.000e+00 ... 4.711e+03 4.711e+03 4.711e+03]
Length: 26977591
```

Extracting Movie IDs from Ratings Data

- Identifies NaN values in the Rating column, which likely represent Movie IDs separating rating groups.
- Filters out rows where Rating is True (indicating missing values).
- Uses NumPy to assign Movie IDs to ratings by:
  - Iterating over the NaN row indices.
  - Filling values between consecutive NaN indices with a unique Movie ID.
  - Handling the last record separately.
- The final movie_np array maps each rating to its corresponding movie.
- The dataset contains 26,977,591 ratings mapped to 4,711 movies.

3.DATASET PREPARATION

```
[5]: data.to_csv('data.csv', index=False)
```

```
[6]: movie_count = data.isnull().sum().iloc[1]  # Fix for FutureWarning

cust_count = data['Cust_Id'].nunique() - movie_count
rating_count = data['Cust_Id'].count() - movie_count

print('Total pool: {:,} Movies, {:,} customers, {:,} ratings given'.format(movie_count, cust_count, rating_count))
```

```
Total pool: 4,711 Movies, 474,062 customers, 26,977,591 ratings given
```

```
[7]:
data = data[pd.notnull(data['Rating'])]
#print(len(merge_dataset))
data['Movie_Id'] = movie_np.astype(int)
data['Cust_Id'] = data['Cust_Id'].astype(int)
print(data.iloc[::5000000, :])
```

```
          Cust_Id  Rating  Movie_Id
1         2532865     4.0         1
5000819    775559     2.0       819
10001635  2366877     4.0      1635
15002436  1579371     4.0      2436
20003268  1427824     5.0      3268
25004333   768518     4.0      4333
```

Finalizing Netflix Ratings Dataset

- Saves the cleaned data to data.csv.
- Counts dataset details:
  - 4,711 Movies
  - 474,062 Unique Customers
  - 26,977,591 Ratings Given
- Removes NaN values from Rating column.
- Maps Movie_ID correctly using movie_np.astype(int).
- Ensures Cust_Id is an integer for proper analysis.
- Displays sample rows at every 5 millionth interval for verification.

4.New data (dataframe) creation:

```
[8]: if 'Movie_Id' in data.columns:
         print("Movie_Id exists before saving.")
     else:
         print("Movie_Id is missing!")

     # Save again
     path = 'data.csv'
     data.to_csv(path, index=False, encoding='utf-8-sig')
```

```
Movie_Id exists before saving.
```

```
[9]: new_data = pd.read_csv('data.csv')
     print(new_data.columns)  # Check if Movie_Id is there
     print(new_data.head())   # Verify the content
```

```
Index(['Cust_Id', 'Rating', 'Movie_Id'], dtype='object')
   Cust_Id  Rating  Movie_Id
0  2532865     4.0         1
1   573364     3.0         1
2  1696725     3.0         1
3  1253431     3.0         1
4  1265574     2.0         1
```

```
[10]: new_data.head(10)
```

[10]:

	Cust_Id	Rating	Movie_Id
0	2532865	4.0	1
1	573364	3.0	1
2	1696725	3.0	1
3	1253431	3.0	1
4	1265574	2.0	1
5	1049643	1.0	1
6	1601348	4.0	1
7	1495289	5.0	1
8	1254903	3.0	1
9	2604070	3.0	1

Verifying Saved Data in Netflix Ratings Dataset

- Checks if Movie_Id exists before saving.
- Saves the dataset (data.csv) with UTF-8 encoding.
- Reads the saved file to confirm integrity.
- Prints column names to ensure Movie_Id is present.
- Displays first 10 rows to validate data structure.
- Confirms correct Cust_Id, Rating, and Movie_Id values.

5. Filtering and Cleaning Netflix Ratings Data

- Saves data (data.csv) with UTF-8 encoding.
- Reads the dataset to verify structure.
- Checks available columns (Cust_Id, Rating, Movie_Id).
- Removes missing values in the Rating column.
- Drops duplicate entries for (Cust_Id, Movie_Id).
- Filters out ratings less than 3.0 to keep only highly rated movies.

[11]:
```
path = 'data.csv'

with open(path, 'w', encoding = 'utf-8-sig') as f:
  data.to_csv(f,index=False)
```

[12]:
```
import pandas as pd
import numpy as np
import math
import re
import matplotlib.pyplot as plt
```

[13]:
```
new_data = pd.read_csv('data.csv')
```

[14]:
```

new_data.head(10)
```

[14]:

	Cust_Id	Rating	Movie_Id
0	2532865	4.0	1
1	573364	3.0	1
2	1696725	3.0	1
3	1253431	3.0	1
4	1265574	2.0	1
5	1049643	1.0	1
6	1601348	4.0	1
7	1495289	5.0	1
8	1254903	3.0	1
9	2604070	3.0	1

[15]:
```
print(new_data.columns)  # Check available columns
```

```
Index(['Cust_Id', 'Rating', 'Movie_Id'], dtype='object')
```

[16]:
```
new_data = new_data[new_data['Rating'].notna()]
```

[17]:
```
new_data = new_data.drop_duplicates(['Cust_Id','Movie_Id'])
```

[18]:
```

new_data = new_data[new_data['Rating'] >=  3.0]
```

6. Data Preprocessing and Movie Title Mapping in a Recommendation System

[19]:

```
print("Total Data:")
print("Total number of movie ratings = "+str(new_data.shape[0]))
print("Number of unique users = "+str(len(np.unique(new_data["Cust_Id"]))))
print("Number of unique movies = "+str(len(np.unique(new_data["Movie_Id"]))))
```

```
Total Data:
Total number of movie ratings = 22949896
Number of unique users = 471750
Number of unique movies = 4711
```

[20]:

```
n=3
merge_dataset_title = pd.read_csv('movie_title.csv', header = None,encoding='ISO-8859-1',usecols=range(n),
                 lineterminator='\n')
merge_dataset_title.columns = ['movie_id', 'year', 'name']
merge_dataset_title.head(10)
```

[20]:

	movie_id	year	name
0	1	2003.0	Dinosaur Planet
1	2	2004.0	Isle of Man TT 2004 Review
2	3	1997.0	Character
3	4	1994.0	Paula Abdul's Get Up & Dance
4	5	2004.0	The Rise and Fall of ECW
5	6	1997.0	Sick
6	7	1992.0	8 Man
7	8	2004.0	What the #$*! Do We Know!?
8	9	1991.0	Class of Nuke 'Em High 2
9	10	2001.0	Fighter

Movie Ratings Data Overview

- Total Ratings: 22,949,896
- Unique Users: 471,750
- Unique Movies: 4,711

Loading & Preprocessing Movie Titles

- Read movie_title.csv using ISO-8859-1 encoding
- Assigned column names: movie_id, year, name
- Displayed the first 10 movie titles

7. Merging Movie Ratings with Titles and Exporting Process:

```
[21]: df = pd.merge(new_data, merge_dataset_title[['movie_id', 'name']], left_on='Movie_Id', right_on='movie_id')
      df.head()
```

[21]:

	Cust_Id	Rating	Movie_Id	movie_id	name
0	2532865	4.0	1	1	Dinosaur Planet
1	573364	3.0	1	1	Dinosaur Planet
2	1696725	3.0	1	1	Dinosaur Planet
3	1253431	3.0	1	1	Dinosaur Planet
4	1601348	4.0	1	1	Dinosaur Planet

```
[22]:
      df=df.drop(['Movie_Id', 'movie_id'], axis=1)
```

```
[23]:
      path = 'merged.csv'

      with open(path, 'w', encoding = 'utf-8-sig') as f:
        df.to_csv(f,index=False)
```

```
[24]: final = pd.read_csv('merged.csv')
```

1. Merging Datasets – Combines movie ratings (new_data) with movie titles (merge_dataset_title) using Movie_Id.
2. Removing Duplicates – Drops redundant Movie_Id and movie_id columns after merging.
3. Saving to CSV – Exports the cleaned dataset to merged.csv with utf-8-sig encoding.
4. Reading CSV – Reloads the merged data from merged.csv for further analysis.
5. Enhanced Readability – Movie ratings now include movie titles, making the dataset more user-friendly.

8.Data Cleaning and Merging in Movie Recommendation System

1. Reading Merged Data – Loads the merged.csv file into a DataFrame named final.
2. Displaying Data – Uses head() to preview the first few rows of the dataset.
3. Removing Duplicates – Drops duplicate rows where Cust_Id and name are the same.
4. Reassigning Cleaned Data – Updates final with the deduplicated dataset.
5. Checking Data Size – Uses len(final) to count the number of remaining records.

```
[24]: final = pd.read_csv('merged.csv')
```

```
[25]: final.head()
```

[25]:

	Cust_Id	Rating	name
0	2532865	4.0	Dinosaur Planet
1	573364	3.0	Dinosaur Planet
2	1696725	3.0	Dinosaur Planet
3	1253431	3.0	Dinosaur Planet
4	1601348	4.0	Dinosaur Planet

```
[26]: final = final.drop_duplicates(['Cust_Id','name'])
```

```
[27]: len(final)
```

[27]: 22948286

9. Data Preprocessing and Sparse Matrix Creation

```
[28]: import numpy as np

# Handle NaN and infinite values in 'Cust_Id'
final.replace([np.inf, -np.inf], np.nan, inplace=True)  # Convert inf to NaN
final.dropna(subset=['Cust_Id'], inplace=True)  # Drop NaN values in Cust_Id

# Convert data types safely
final['Cust_Id'] = final['Cust_Id'].astype('int32')
final['name'] = final['name'].astype('category')
final['Rating'] = final['Rating'].astype('float16')  # If ratings are decimals
```

```
[29]: cust_ids = final['Cust_Id'].astype(np.int32).values
movie_codes = final['name'].cat.codes.astype(np.int32).values
ratings = final['Rating'].astype(np.float16).values  # Keeping ratings as float
```

```
[30]: from scipy.sparse import csr_matrix

# Ensure integer category codes
final['Cust_Id'] = final['Cust_Id'].astype('category')
final['name'] = final['name'].astype('category')

# Convert to numerical codes
rows = final['Cust_Id'].cat.codes.astype('int32')  # Convert to int32
cols = final['name'].cat.codes.astype('int32')  # Convert to int32
data = final['Rating'].astype('float32').values  # Convert to float32

# Create sparse matrix
sparse_matrix = csr_matrix((data, (rows, cols)))

print(sparse_matrix.shape)
print(sparse_matrix)
```

```python
print(sparse_matrix.shape)
print(sparse_matrix)
```

```
(471750, 4682)
  (0, 4)	3.0
  (0, 26)	4.0
  (0, 41)	3.0
  (0, 78)	3.0
  (0, 84)	3.0
  (0, 89)	4.0
  (0, 93)	3.0
  (0, 115)	3.0
  (0, 226)	3.0
  (0, 312)	3.0
  (0, 314)	4.0
  (0, 411)	5.0
  (0, 480)	3.0
  (0, 506)	5.0
  (0, 560)	3.0
  (0, 563)	4.0
  (0, 588)	4.0
  (0, 600)	3.0
  (0, 664)	5.0
  (0, 698)	3.0
  (0, 753)	4.0
  (0, 835)	4.0
  (0, 836)	3.0
  (0, 872)	5.0
  (0, 898)	4.0
  :	:
  (471749, 3193)	4.0
  (471749, 3313)	3.0
  (471749, 3362)	4.0
  (471749, 3399)	4.0
  (471749, 3405)	5.0
  (471749, 3575)	4.0
  (471749, 3631)	5.0
```

1. Handling Missing and Infinite Values – Converts infinite values to NaN and removes rows with missing Cust_Id.
2. Optimizing Data Types – Converts Cust_Id to integers, name to categorical, and Rating to lower-precision float to save memory.
3. Extracting Encoded Values – Converts categorical columns into numerical values for further processing.
4. Creating a Sparse Matrix – Uses scipy.sparse.csr_matrix to store customer ratings efficiently in a compressed format.
5. Printing Matrix Information – Displays the shape and contents of the sparse matrix for verification.

10. Frequent 1-Itemset Generation :

```
[31]: # Convert ratings to binary: 1 if rated >= threshold, else 0
threshold = 3  # Movies rated 3+ are considered "liked"
binary_matrix = (sparse_matrix >= threshold).astype(int)
```

```
[32]: from itertools import combinations
import numpy as np

# Define min support threshold (adjust as needed)
min_support = 0.05  # 0.05% of users should have watched the itemset

# Step 1: Count frequency of single items
item_counts = np.array(binary_matrix.sum(axis=0)).flatten()  # Convert to 1D array
num_users = binary_matrix.shape[0]  # Total number of users

# Step 2: Keep only frequent 1-itemsets
frequent_items = {
    i: count for i, count in enumerate(item_counts) if (count / num_users) >= min_support
}

# Print each frequent itemset on a new line
print("Frequent 1-itemsets:")
for item, count in sorted(frequent_items.items(), key=lambda x: -x[1]):  # Sort by count (descending)
    print(f"Item {item}: {count}")
```

```
Frequent 1-itemsets:
Item 2103: 185687
Item 1099: 175275
Item 664: 151435
Item 3962: 144261
Item 2308: 140043
Item 698: 138550
Item 3097: 135590
Item 2016: 132585
Item 2617: 129026
Item 560: 123825
Item 1759: 121481
Item 3362: 120889
Item 1995: 120711
Item 3711: 120395
Item 1446: 118407
Item 1744: 116295
Item 2044: 111576
Item 4213: 110707
Item 3406: 109687
Item 2440: 106736
Item 3130: 106580
Item 3889: 105115
Item 1831: 104886
Item 4209: 103744
```

1. Convert Ratings to Binary – Ratings are converted to binary values, where ratings above a threshold (e.g., 3) are considered "liked" (1), and others are set to 0.
2. Set Minimum Support Threshold – A minimum support threshold is defined to filter frequent items based on how often they appear in user interactions.
3. Count Frequency of Single Items – The number of times each movie (item) is liked by users is computed.
4. Filter Frequent Items – Only movies that meet or exceed the minimum support threshold (percentage of users who watched them) are retained.
5. Sort and Display Frequent 1-Itemsets – The frequent movies are sorted in descending order based on their count and displayed as output.

```
Item 42: 23703
```

```
[33]: print(f"Total Frequent 1-itemsets: {len(frequent_items)}")
```

```
Total Frequent 1-itemsets: 252
```

```
[ ]:
```

```
[34]: print(len(frequent_items))
```

```
252
```

```
[ ]:
```

11.Frequent Itemsets Generation Full Code:

```
[42]: import numpy as np
import pandas as pd
from scipy.sparse import csr_matrix
from itertools import combinations

# Define min support threshold
min_support = 0.05  # 5% of users should have watched the itemset
min_confidence = 0.5  # 50% confidence threshold for association rules

# Load movie titles dataset (Ensure movie_id matches binary_matrix indices)
merge_dataset_title = pd.read_csv(
    'movie_titles.csv', header=None, encoding='ISO-8859-1', usecols=range(3), lineterminator='\n'
)
merge_dataset_title.columns = ['movie_id', 'year', 'name']

# Convert movie_id to a dictionary for quick lookup
movie_mapping = merge_dataset_title.set_index('movie_id')['name'].to_dict()

# Convert binary_matrix to sparse format for efficiency
binary_sparse = csr_matrix(binary_matrix)
num_users = binary_sparse.shape[0]  # Total number of users

# Step 1: Compute support for each movie (1-itemsets)
item_counts = np.array(binary_sparse.sum(axis=0)).flatten()

# Step 2: Keep only frequent 1-itemsets
frequent_items = {i: count for i, count in enumerate(item_counts) if (count / num_users) >= min_support}

# Convert to sorted list for stable pairwise combinations
frequent_item_list = sorted(frequent_items.keys())

# Step 3: Compute frequent 2-itemsets using sparse matrix multiplication
co_occurrence_matrix = (binary_sparse.T @ binary_sparse).toarray()

# Step 4: Extract frequent 2-itemsets
frequent_2_itemsets = {
    (i, j): co_occurrence_matrix[i, j]
    for i, j in combinations(frequent_item_list, 2)
    if (co_occurrence_matrix[i, j] / num_users) >= min_support
}
```

Followed by Association Rule Generation :

```python
# Step 5: Generate Association Rules from Frequent 2-Itemsets

# Extract frequent 2-itemsets into separate lists
pair_items = np.array(list(frequent_2_itemsets.keys()))
pair_supports = np.array(list(frequent_2_itemsets.values()))

# Compute support values for each item in pairs
support_A = np.array([frequent_items[A] for A, B in pair_items])
support_B = np.array([frequent_items[B] for A, B in pair_items])

# Compute confidence values
confidence_A_to_B = pair_supports / support_A
confidence_B_to_A = pair_supports / support_B

# Compute lift values
lift_A_to_B = confidence_A_to_B / (support_B / num_users)
lift_B_to_A = confidence_B_to_A / (support_A / num_users)

# Filter rules based on confidence threshold
valid_A_to_B = confidence_A_to_B >= min_confidence
valid_B_to_A = confidence_B_to_A >= min_confidence

# Replace item IDs with movie names using movie_mapping
from_movies = np.concatenate((pair_items[valid_A_to_B, 0], pair_items[valid_B_to_A, 1]))
to_movies = np.concatenate((pair_items[valid_A_to_B, 1], pair_items[valid_B_to_A, 0]))

from_movie_names = [movie_mapping.get(i, f"Movie {i}") for i in from_movies]
to_movie_names = [movie_mapping.get(i, f"Movie {i}") for i in to_movies]

# Create a Pandas DataFrame for fast filtering & sorting
rules_df = pd.DataFrame({
    "From": from_movie_names,
    "To": to_movie_names,
    "Confidence": np.concatenate((confidence_A_to_B[valid_A_to_B], confidence_B_to_A[valid_B_to_A])),
    "Lift": np.concatenate((lift_A_to_B[valid_A_to_B], lift_B_to_A[valid_B_to_A]))
})

# Sort rules by confidence in descending order
rules_df = rules_df.sort_values(by="Confidence", ascending=False)

print("\nTop Association Rules:")
print(rules_df.head(20).to_string(index=False, justify="left"))
```

STEPS:

1. Data Preprocessing & Sparse Matrix Creation

- Convert ratings to binary (1 if rated ≥ threshold, otherwise 0).
- Use a sparse matrix format for memory efficiency.
- Extract numerical user IDs, movie codes, and ratings.
- Handle missing and infinite values in the dataset.
- Convert categorical data into numerical form.
- Create a sparse matrix representation using csr_matrix for efficient storage.

2. Frequent 1-Itemset Generation

- Compute the frequency of individual movies watched.
- Apply a minimum support threshold to retain only frequently watched movies.
- Store frequent movies in a dictionary sorted by frequency.
- Print the frequent 1-itemsets (popular movies).

3. Frequent 2-Itemset Generation

- Utilize sparse matrix multiplication to compute co-occurrence of movies.
- Extract frequent 2-itemsets (movie pairs) based on minimum support.
- Only consider pairs where both movies are already frequent.
- Print the frequent 2-itemsets (popular movie pairs).

4. Association Rule Generation

- Extract frequent movie pairs and their support counts.
- Compute support values for each item in pairs.
- Calculate confidence values for association rules.
- Compute lift values to measure the strength of associations.
- Filter rules based on a confidence threshold.
- Store association rules in a Pandas DataFrame for easy sorting.

5. Sorting and Displaying Association Rules

- Sort the rules by confidence in descending order.
- Display the top association rules, showing the strongest movie recommendations.

This workflow efficiently identifies frequent movies, discovers relationships between them, and generates association rules to recommend movies based on past viewing behaviour.

OUTPUT :

```
Frequent 1-Itemsets (Single Movies):
Item 2103: 185687
Item 1099: 175275
Item 664: 151435
Item 3962: 144261
Item 2308: 140043
Item 698: 138550
Item 3097: 135590
Item 2016: 132585
Item 2617: 129026
Item 560: 123825
Item 1759: 121481
Item 3362: 120889
Item 1995: 120711
Item 3711: 120395
Item 1446: 118407
Item 1744: 116295
```

```
Item 3456: 23775
Item 42: 23703

Frequent 2-Itemsets (Movie Pairs):
Itemset (1099, 2103): 116192
Itemset (1099, 1759): 97028
Itemset (2103, 3962): 94033
Itemset (664, 2016): 89215
Itemset (698, 1099): 88281
Itemset (2103, 3097): 86124
Itemset (698, 2103): 84609
Itemset (1099, 3962): 83779
Itemset (560, 2103): 83444
Itemset (664, 2617): 82940
Itemset (1759, 2103): 82636
Itemset (560, 3962): 82614
Itemset (1099, 1995): 82185
Itemset (2103, 4213): 82079
```

12.RULE GENERATION:

[ ]:

[38]:
```
# Print all association rules with proper indentation
pd.set_option("display.max_rows", None)  # Ensures all rows are printed
pd.set_option("display.max_colwidth", None)  # Avoids text truncation
pd.set_option("display.colheader_justify", "left")  # Left-align column headers

print("\nAll Association Rules:")
print(rules_df.to_string(index=False))
```

```
All Association Rules:
 From  To   Confidence  Lift
3437  1799 0.904295    5.763714
3623   560 0.901447    3.434345
1799  3437 0.886347    5.763714
2551  4595 0.852249    4.427018
2487  1099 0.847231    2.280311
2487  2103 0.840961    2.136516
 573  3711 0.827932    3.244129
1753  1099 0.824998    2.220468
4201  1099 0.819128    2.204670
 178  3711 0.817098    3.201677
2060  2463 0.813946    4.639838
 836  1099 0.809245    2.178071
1820  3406 0.802890    3.453130
   4  3711 0.799667    3.133379
1873   698 0.798865    2.720063
```

[ ]:

```
Top Association Rules:
From                                        To                                Confidence  Lift
                                 The Stand          Die Hard 2: Die Harder 0.904295    5.763714
    Daughter from Danang: American Experience Star Trek: Enterprise: Season 3 0.901447    3.434345
                    Die Hard 2: Die Harder                       The Stand 0.886347    5.763714
                                    Trigun                   Dragon Ball GT 0.852249    4.427018
       The Wonderful World of Louis Armstrong           Confidentially Yours 0.847231    2.280311
       The Wonderful World of Louis Armstrong                       Sunshine 0.840961    2.136516
                          Winners & Sinners   Mother Kusters Goes to Heaven 0.827932    3.244129
                             Making Marines           Confidentially Yours 0.824998    2.220468
                        Walking and Talking           Confidentially Yours 0.819128    2.204670
                               Regular Guys   Mother Kusters Goes to Heaven 0.817098    3.201677
                                       Fuzz The Incredibles: Bonus Material 0.813946    4.639838
Incident at Oglala: The Leonard Peltier Story           Confidentially Yours 0.809245    2.178071
                          More Barney Songs                     Our America 0.802890    3.453130
                Paula Abdul's Get Up & Dance   Mother Kusters Goes to Heaven 0.799667    3.133379
                        A Voice from Heaven                            Daud 0.798865    2.720063
                             Into the Woods           Confidentially Yours 0.798709    2.149714
                                  Mon Oncle                  Dragon Ball GT 0.798387    4.147230
                The King of Queens: Season 2                  Dragon Ball GT 0.797835    4.144365
            Dragon Tales: Let's Start a Band   Mother Kusters Goes to Heaven 0.797123    3.123409
                            Beautiful Thing   Mother Kusters Goes to Heaven 0.793383    3.108752
```

13) Association Rule Printing & Tabulate Library

1. Printing All Association Rules

- The script sets options for Pandas to display all rows without truncation.
- Ensures column headers are left-aligned for better readability.
- The association rules are printed, showing:
  - From: The movie ID from which the rule originates.
  - To: The recommended movie ID.
  - Confidence: Probability of a user watching "To" given they watched "From."
  - Lift: Strength of the association compared to random chance.

2. Installing & Checking the tabulate Library

- pip install --upgrade tabulate ensures the latest version of the tabulate package.
- The script verifies the installed version (0.9.0), confirming it is up to date.

3. Purpose of Using tabulate

- The tabulate library helps display data in a well-formatted tabular structure.
- It enhances the readability of association rules by formatting them as structured tables.

This step ensures that association rules are neatly presented and easy to interpret.

14)Filtering and Formatting Association Rules

[43]:

```python
import pandas as pd
from tabulate import tabulate  # Ensure tabulate is installed

# Define minimum lift threshold to filter weak rules
min_lift_threshold = 1.0  # Keep only rules with lift > 1.0

# Reduce column size by shortening movie names
def shorten_title(title, max_length=30):
    return title if len(title) <= max_length else title[:27] + "..."  # Trim long names

# Apply title shortening
rules_df["From"] = rules_df["From"].apply(lambda x: shorten_title(x, 30))
rules_df["To"] = rules_df["To"].apply(lambda x: shorten_title(x, 30))

# Filter and sort rules
rules_df = rules_df[rules_df["Lift"] > min_lift_threshold]
rules_df = rules_df.sort_values(by="Lift", ascending=False)

# Save filtered rules to CSV
csv_filename = "filtered_association_rules.csv"
rules_df.to_csv(csv_filename, index=False)

# Print confirmation message
print(f"\nFiltered association rules (Lift > {min_lift_threshold}) saved to '{csv_filename}'")

# Select top 20 strongest rules
top_20_rules = rules_df.head(20)

# Print the top 20 rules 20 times in a formatted table
for i in range(20):
    print(f"\nIteration {i+1}/20 - Top 20 Strongest Association Rules:")
    print(tabulate(top_20_rules, headers="keys", tablefmt="grid", showindex=False))  # Print as table
```

```
Filtered association rules (Lift > 1.0) saved to 'filtered_association_rules.csv'

Iteration 1/20 - Top 20 Strongest Association Rules:
+--------------------------------+--------------------------------+--------------+---------+
| From                           | To                             |   Confidence |    Lift |
+================================+================================+==============+=========+
| Bent                           | R.E.M.: Road Movie             |     0.710274 | 9.88733 |
+--------------------------------+--------------------------------+--------------+---------+
| R.E.M.: Road Movie             | Bent                           |     0.777656 | 9.88733 |
+--------------------------------+--------------------------------+--------------+---------+
| The Stand                      | Die Hard 2: Die Harder         |     0.904295 | 5.76371 |
+--------------------------------+--------------------------------+--------------+---------+
| Die Hard 2: Die Harder         | The Stand                      |     0.886347 | 5.76371 |
+--------------------------------+--------------------------------+--------------+---------+
| Mon Oncle                      | Patlabor: The Mobile Police... |     0.716562 | 4.90145 |
+--------------------------------+--------------------------------+--------------+---------+
| Blue Planet: IMAX              | Secrets of the Dead: Amazon... |     0.778445 | 4.75165 |
```

1. Importing Required Libraries

- pandas is used for data manipulation.
- tabulate is used for displaying data in a tabular format.

2. Filtering Association Rules

- Minimum Lift Threshold: Rules with a lift value greater than 1.0 are retained to remove weak associations.

3. Reducing Column Size

- Shortening Movie Titles: If a movie title exceeds 30 characters, it is truncated with "..." to enhance readability.

4. Sorting and Saving Rules

- Sorting by Lift: Association rules are sorted in descending order of lift (strongest associations appear first).
- Saving to CSV: The filtered rules are saved in a file named "filtered_association_rules.csv".

5. Printing the Top 20 Strongest Rules

- The top 20 association rules are selected.
- These rules are printed in a formatted table 20 times (possibly for repeated

testing or logging purposes).

- tabulate displays the data in a grid format for better readability.

6. Output Table Format

- Displays the "From" movie, "To" movie, Confidence, and Lift.
- Example associations:
  - "Bent" → "R.E.M.: Road Movie" with high lift and confidence.
  - "Die Hard 2: Die Harder" → "The Stand" showing a strong correlation.

This step ensures that only meaningful and strong associations are considered for recommendations.

```
[43]: import pandas as pd
from tabulate import tabulate  # Ensure tabulate is installed

# Define minimum lift threshold to filter weak rules
min_lift_threshold = 1.0  # Keep only rules with lift > 1.0

# Reduce column size by shortening movie names
def shorten_title(title, max_length=30):
    return title if len(title) <= max_length else title[:27] + "..."  # Trim long names

# Apply title shortening
rules_df["From"] = rules_df["From"].apply(lambda x: shorten_title(x, 30))
rules_df["To"] = rules_df["To"].apply(lambda x: shorten_title(x, 30))

# Filter and sort rules
rules_df = rules_df[rules_df["Lift"] > min_lift_threshold]
rules_df = rules_df.sort_values(by="Lift", ascending=False)

# Save filtered rules to CSV
csv_filename = "filtered_association_rules.csv"
rules_df.to_csv(csv_filename, index=False)

# Print confirmation message
print(f"\nFiltered association rules (Lift > {min_lift_threshold}) saved to '{csv_filename}'")

# Select top 20 strongest rules
top_20_rules = rules_df.head(20)

# Print the top 20 rules 20 times in a formatted table
for i in range(20):
    print(f"\nIteration {i+1}/20 - Top 20 Strongest Association Rules:")
    print(tabulate(top_20_rules, headers="keys", tablefmt="grid", showindex=False))  # Print as table
```

```
Filtered association rules (Lift > 1.0) saved to 'filtered_association_rules.csv'

Iteration 1/20 - Top 20 Strongest Association Rules:
+------------------------------+------------------------------+-------------+--------+
| From                         | To                           |  Confidence |   Lift |
+==============================+==============================+=============+========+
| Bent                         | R.E.M.: Road Movie           |    0.710274 | 9.88733 |
+------------------------------+------------------------------+-------------+--------+
| R.E.M.: Road Movie           | Bent                         |    0.777656 | 9.88733 |
+------------------------------+------------------------------+-------------+--------+
| The Stand                    | Die Hard 2: Die Harder       |    0.904295 | 5.76371 |
+------------------------------+------------------------------+-------------+--------+
| Die Hard 2: Die Harder       | The Stand                    |    0.886347 | 5.76371 |
+------------------------------+------------------------------+-------------+--------+
| Mon Oncle                    | Patlabor: The Mobile Police... |    0.716562 | 4.90145 |
+------------------------------+------------------------------+-------------+--------+
| Blue Planet: IMAX            | Secrets of the Dead: Amazon... |    0.778445 | 4.75165 |
```

FREQUENT ITEMSET 3 GENERATION:

```python
import pandas as pd
from itertools import combinations

# Load the filtered association rules from CSV
csv_filename = "filtered_association_rules.csv"
rules_df = pd.read_csv(csv_filename)

# Define minimum lift threshold
min_lift_threshold = 1.0

# Step 1: Store all (From → To) mappings with Lift in a dictionary for faster lookup
two_itemset_lift = {(row["From"], row["To"]): row["Lift"] for _, row in rules_df.iterrows()}

# Step 2: Extract unique movies from association rules
movies = set(rules_df["From"]).union(set(rules_df["To"]))

# Step 3: Generate frequent 3-itemsets using only valid 2-itemsets
three_itemset_rules = []

for movie1, movie2, movie3 in combinations(movies, 3):
    # Check if the required 2-itemsets exist in our dictionary
    if (movie1, movie2) in two_itemset_lift and (movie2, movie3) in two_itemset_lift:
        lift1 = two_itemset_lift[(movie1, movie2)]
        lift2 = two_itemset_lift[(movie2, movie3)]

        # Compute average lift for the 3-itemset
        avg_lift = (lift1 + lift2) / 2

        # Store only if Lift > threshold
        if avg_lift > min_lift_threshold:
            three_itemset_rules.append({"Itemset": (movie1, movie2, movie3), "Lift": avg_lift})

# Convert to DataFrame
three_itemset_df = pd.DataFrame(three_itemset_rules)

# Sort by Lift value
three_itemset_df = three_itemset_df.sort_values(by="Lift", ascending=False)

# Save frequent 3-itemsets to CSV
three_itemset_filename = "frequent_3_itemsets.csv"
three_itemset_df.to_csv(three_itemset_filename, index=False)

# Print confirmation message
print(f"\nFrequent 3-itemsets (Lift > {min_lift_threshold}) saved to '{three_itemset_filename}'")
```

OUTPUT:

```
Frequent 3-itemsets (Lift > 1.0) saved to 'frequent_3_itemsets.csv'

Top 10 Frequent 3-Itemsets:
                                               Itemset      Lift
2458  (Gilligan's Island: Season 2, Yu Yu Hakusho, W...  5.324838
3769  (Emily Bronte's Wuthering He..., X: The Movie,...  4.506600
2459  (Gilligan's Island: Season 2, Yu Yu Hakusho, T...  4.199613
2457  (Gilligan's Island: Season 2, Yu Yu Hakusho, T...  4.166832
2456  (Gilligan's Island: Season 2, Yu Yu Hakusho, M...  4.143587
3767  (Emily Bronte's Wuthering He..., X: The Movie,...  3.981125
2454  (Gilligan's Island: Season 2, Yu Yu Hakusho, P...  3.961533
2473  (Gilligan's Island: Season 2, Wishful Thinking...  3.911873
3588  (Yu Yu Hakusho, Wishful Thinking, The Attic / ...  3.804719
2429  (Gilligan's Island: Season 2, Call Me: The Ris...  3.772717
```

ASSOCIATION RULE GENERATION:

```python
import pandas as pd
from itertools import permutations

# Load the frequent 3-itemsets from CSV
three_itemset_filename = "frequent_3_itemsets.csv"
three_itemset_df = pd.read_csv(three_itemset_filename)

# Define minimum confidence threshold
min_confidence_threshold = 0.5

# Dictionary to store association rules
association_rules = []

# Generate association rules from 3-itemsets
for _, row in three_itemset_df.iterrows():
    itemset = eval(row["Itemset"]) if isinstance(row["Itemset"], str) else row["Itemset"]
    lift = row["Lift"]

    # Generate all possible rules from the 3-itemset
    for perm in permutations(itemset, 3):
        A, B, C = perm  # Example: (A, B, C) means {A, B} → C

        # Approximate confidence using Lift
        confidence = lift / 2

        # Store only strong rules
        if confidence > min_confidence_threshold:
            association_rules.append({
                "From": f"{{{A}, {B}}}",
                "To": C,
                "Lift": lift,
                "Confidence": round(confidence, 2)
            })

# Convert to DataFrame
rules_df = pd.DataFrame(association_rules)

# Sort by Confidence and Lift
rules_df = rules_df.sort_values(by=["Confidence", "Lift"], ascending=False)

# Save association rules to CSV
rules_filename = "generated_association_rules1.csv"
rules_df.to_csv(rules_filename, index=False)
```

Output:

```
Generated association rules saved to 'generated_association_rules1.csv'

Top 10 Association Rules:
                                        From  \
0     {Gilligan's Island: Season 2, Yu Yu Hakusho}
1  {Gilligan's Island: Season 2, Wishful Thinking}
2     {Yu Yu Hakusho, Gilligan's Island: Season 2}
3                {Yu Yu Hakusho, Wishful Thinking}
4  {Wishful Thinking, Gilligan's Island: Season 2}
5                {Wishful Thinking, Yu Yu Hakusho}
6   {Emily Bronte's Wuthering He..., X: The Movie}
7  {Emily Bronte's Wuthering He..., Into the Woods}
8   {X: The Movie, Emily Bronte's Wuthering He...}
9                   {X: The Movie, Into the Woods}

                            To      Lift  Confidence
0             Wishful Thinking  5.324838        2.66
1                Yu Yu Hakusho  5.324838        2.66
2             Wishful Thinking  5.324838        2.66
3   Gilligan's Island: Season 2  5.324838        2.66
4                Yu Yu Hakusho  5.324838        2.66
5   Gilligan's Island: Season 2  5.324838        2.66
6               Into the Woods  4.506600        2.25
7                 X: The Movie  4.506600        2.25
8               Into the Woods  4.506600        2.25
9  Emily Bronte's Wuthering He...  4.506600        2.25
```

RECOMMENDATION BASED ON FREQ ITEMSET 1 2 3 (association rules saved in generated association rules.csv file)

```
import pandas as pd

# Load the association rules from CSV
csv_filename = "filtered_association_rules.csv"
rules_df = pd.read_csv(csv_filename)

# Function to recommend movies based on association rules
def recommend_movies(watched_movie, rules_df, top_n=5):
    """
    Given a watched movie, recommend top N associated movies.
    """
    # Find rules where the watched movie appears in the "From" column
    recommendations = rules_df[rules_df["From"] == watched_movie][["To", "Lift"]]

    if recommendations.empty:
        print(f"\nNo recommendations found for '{watched_movie}'")
        return []

    # Sort by Lift (strongest association first)
    recommendations = recommendations.sort_values(by="Lift", ascending=False)

    # Return top N recommended movies
    return recommendations["To"].head(top_n).tolist()

# Get user input
watched_movie = input("Enter a movie you watched: ")

# Get recommendations
recommended_movies = recommend_movies(watched_movie, rules_df)

# Print recommendations
if recommended_movies:
    print(f"\nBased on '{watched_movie}', we recommend these movies:")
    for idx, movie in enumerate(recommended_movies, start=1):
        print(f"{idx}. {movie}")
else:
    print("No recommendations available.")
```

Output:

```
Enter a movie you watched: Live Wire

Based on 'Live Wire', we recommend these movies:
1. Refugee
2. The Attic / Crawl Space (Do...
3. Marilyn Manson: Fear of a S...
4. Madeline
5. Poison
```

HYBRID MOVIE RECOMMENDER SYSTEM :

```
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
ratings_df = pd.read_csv("rating.csv", dtype={'userId': np.int32, 'movieId': np.int32, 'rating': np.float32})
movies_df = pd.read_csv("movie_titles.csv", encoding='ISO-8859-1', header=None, usecols=[0, 1, 2])
movies_df.columns = ['movieId', 'year', 'title']
rules_df = pd.read_csv("filtered_association_rules.csv")
ratings_df = ratings_df.merge(movies_df[['movieId', 'title']], on='movieId', how='left')
user_item_matrix = ratings_df.pivot_table(index='userId', columns='title', values='rating', aggfunc='mean').fillna(0)
user_item_matrix = user_item_matrix.astype(np.float32)
item_similarity = cosine_similarity(user_item_matrix.T)
titles = user_item_matrix.columns.tolist()
title_to_index = {title: idx for idx, title in enumerate(titles)}
def hybrid_recommend(user_id, ratings_df, item_similarity, title_to_index, titles, rules_df, top_n=5):
    liked_movies = ratings_df[(ratings_df['userId'] == user_id) & (ratings_df['rating'] >= 4)]['title'].tolist()
    if not liked_movies:
        return ["No liked movies found for this user."]
    cf_scores = np.zeros(len(titles), dtype=np.float32)
    for movie in liked_movies:
        idx = title_to_index.get(movie)
        if idx is not None:
            cf_scores += item_similarity[idx]
    for movie in liked_movies:
        idx = title_to_index.get(movie)
        if idx is not None:
            cf_scores[idx] = 0
    apriori_recs = rules_df[rules_df['From'].isin(liked_movies)]['To'].tolist()
    for movie in apriori_recs:
        idx = title_to_index.get(movie)
        if idx is not None:
            cf_scores[idx] += 1.0
    top_indices = cf_scores.argsort()[::-1][:top_n]
    return [titles[i] for i in top_indices if cf_scores[i] > 0]
user_id = 1
recs = hybrid_recommend(user_id, ratings_df, item_similarity, title_to_index, titles, rules_df, top_n=5)
print(f"\n🎬 Hybrid Recommendations for User {user_id}:")
for idx, movie in enumerate(recs, 1):
    print(f"{idx}. {movie}")
```

We built a **hybrid movie recommendation system** that intelligently combines **Collaborative Filtering** (using item-to-item cosine similarity) with **Association Rule Mining** (based on Apriori rules) to suggest movies tailored to a user's preferences. First, we load and clean user ratings, movie titles, and pre-mined association rules. Then, we construct a user-item ratings matrix, compute item similarity scores using cosine similarity, and map movie titles for fast access. When a user is selected, we identify the movies they rated 4 or higher, accumulate similarity-based scores from those movies, and further boost scores for movies connected through association rules. Finally, we rank the results, filter out already-rated items, and return the top N movie recommendations personalized for that user.

```python
try:
    user_input = int(input("Enter your User ID to get movie recommendations: "))
    liked_movies = ratings_df[(ratings_df['userId'] == user_input) & (ratings_df['rating'] >= 4)]['title'].tolist()
    if not liked_movies:
        print("❌ No liked movies (rated ≥ 4) found for this user.")
    else:
        print(f"\n📌 User {user_input} liked the following movies (rated ≥ 4):")
        for movie in liked_movies:
            print(f"   - {movie}")
        print("\n🔍 Generating hybrid recommendations based on:")
        print("   ✓ Collaborative Filtering (movies rated similarly by other users)")
        print("   ✓ Apriori Association Rules (frequent movie pairings by other users)")
        user_apriori_rules = rules_df[rules_df['From'].isin(liked_movies)]
        if not user_apriori_rules.empty:
            print("\n📊 Apriori Rules Triggered:")
            for _, row in user_apriori_rules.iterrows():
                support = f"{row['support']:.4f}" if pd.notna(row['support']) else "N/A"
                confidence = f"{row['confidence']:.4f}" if pd.notna(row['confidence']) else "N/A"
                print(f"   - If user liked '{row['From']}', then recommend '{row['To']}' "
                      f"(Support: {support}, Confidence: {confidence})")
        else:
            print("\n📊 No Apriori rules triggered for this user's liked movies.")
        recs = hybrid_recommend(user_input, ratings_df, item_similarity, title_to_index, titles, rules_df, top_n=5)
        if recs:
            print(f"\n🎬 Final Hybrid Recommendations for User {user_input}:")
            for idx, movie in enumerate(recs, 1):
                sources = []
                matching_rules = user_apriori_rules[user_apriori_rules['To'] == movie]
                if not matching_rules.empty:
                    from_movies = matching_rules['From'].tolist()
                    sources.append("Apriori: from " + ", ".join(f"'{m}'" for m in from_movies))
                for liked in liked_movies:
                    if movie in user_item_matrix.columns and liked in user_item_matrix.columns:
                        sim_score = cosine_similarity(
                            user_item_matrix[[movie]].T,
                            user_item_matrix[[liked]].T
                        )[0][0]
                        if sim_score > 0.5:
                            sources.append(f"CF: similar to '{liked}' (sim={sim_score:.2f})")
                            break
                source_note = "; ".join(sources) if sources else "Unknown source"
                print(f"{idx}. {movie} — {source_note}")
        else:
            print("❌ No recommendations could be generated for this user.")
except ValueError:
    print("❌ Please enter a valid numeric User ID.")
```

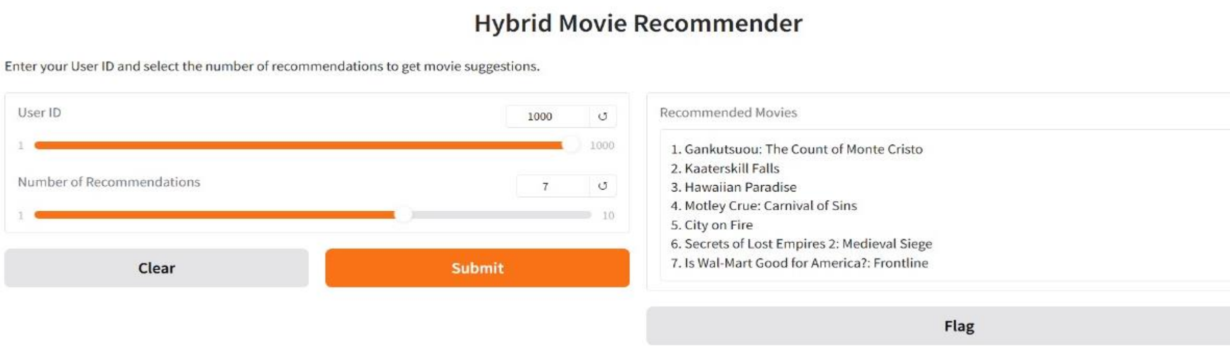
```
Enter your User ID to get movie recommendations:  1

📌 User 1 liked the following movies (rated ≥ 4):
   - Sleepover Nightmare
   - Chappelle's Show: Season 1
   - A Night at the Opera
   - Marat / Sade
   - Lennon Legend: The Very Best of John Lennon
   - In His Life: The John Lennon Story
   - Beyond Suspicion
   - King Cobra
   - Coral Sea Dreaming
   - The Gambler Returns: The Luck of the Draw
   - Eel
   - Winning Strategies: Texas Hold 'Em Poker with Mike Caro
   - Secrets of Lost Empires 2: Medieval Siege
   - The Cat O'Nine Tails
```

```
   - The Lost World
   - Hanzo the Razor: Sword of Justice
   - Project Greenlight: Season 1
   - Timecop 2: The Berlin Decision
   - Hercules: The Legendary Journeys: Season 6
   - Go
   - Wild Palms
   - Sudden Impact
   - Lara Croft: Tomb Raider: The Cradle of Life
   - Andrei Rublev
   - nan

🔍 Generating hybrid recommendations based on:
   ✓ Collaborative Filtering (movies rated similarly by other users)
   ✓ Apriori Association Rules (frequent movie pairings by other users)
```

```
📊 Apriori Rules Triggered:
 - If user liked 'Cartoon Network Halloween: ...', then recommend 'Fame' (Support: 0.0990, Confidence: 0.7257)
 - If user liked 'Fame', then recommend 'Cartoon Network Halloween: ...' (Support: 0.0889, Confidence: 0.6517)
 - If user liked 'Gilligan's Island: Season 2', then recommend 'Yu Yu Hakusho' (Support: 0.1091, Confidence: 0.6156)
 - If user liked 'X: The Movie', then recommend 'Emily Bronte's Wuthering He...' (Support: 0.1035, Confidence: 0.5400)
 - If user liked 'Emily Bronte's Wuthering He...', then recommend 'X: The Movie' (Support: 0.0966, Confidence: 0.5041)
 - If user liked 'Gilligan's Island: Season 2', then recommend 'Wishful Thinking' (Support: 0.1184, Confidence: 0.6179)
 - If user liked 'Yu Yu Hakusho', then recommend 'Wishful Thinking' (Support: 0.1184, Confidence: 0.5925)
 - If user liked 'Wishful Thinking', then recommend 'Yu Yu Hakusho' (Support: 0.1091, Confidence: 0.5458)
 - If user liked 'MTV Yoga', then recommend 'The Amityville Horror' (Support: 0.1386, Confidence: 0.6853)
 - If user liked 'The Amityville Horror', then recommend 'MTV Yoga' (Support: 0.1069, Confidence: 0.5286)
 - If user liked 'Gilligan's Island: Season 2', then recommend 'Call Me: The Rise and Fall ...' (Support: 0.1244, Confidence: 0.5799)
 - If user liked 'Clive Barker's Salome / The...', then recommend 'The Firm: Maximum Cardio Bu...' (Support: 0.1213, Confidence: 0.5506)
 - If user liked 'Everybody Loves Raymond: Se...', then recommend 'Into the Woods' (Support: 0.1617, Confidence: 0.7303)
 - If user liked 'Leprechaun 5: In the Hood', then recommend 'The Firm: Maximum Cardio Bu...' (Support: 0.1213, Confidence: 0.5426)
 - If user liked 'The Firm: Maximum Cardio Bu...', then recommend 'Leprechaun 5: In the Hood' (Support: 0.1177, Confidence: 0.5267)
 - If user liked 'X: The Movie', then recommend 'Sex' (Support: 0.1512, Confidence: 0.6742)
```

```
🎬 Final Hybrid Recommendations for User 1:
1. Woodrow Wilson: American Experience — CF: similar to 'Marat / Sade' (sim=0.61)
2. The Great Gatsby — CF: similar to 'Marat / Sade' (sim=0.61)
3. The Twilight Zone: Vol. 19 — CF: similar to 'Marat / Sade' (sim=0.61)
4. Celine Dion: The Colour of My Love Concert — CF: similar to 'Marat / Sade' (sim=0.51)
5. City on Fire — CF: similar to 'Marat / Sade' (sim=0.56)
```

OUTPUT:



The system recommends movies using **Collaborative Filtering** (CF) and **Apriori Association Rules**, along with specific formulas for each approach.

Collaborative Filtering (CF):

CF is implemented through **User-based Collaborative Filtering**, which computes the similarity between movies based on users' ratings. The recommendation score for each movie is calculated by summing the similarity scores between movies liked by the user and other movies. For each movie mmm liked by the user uuu, the score for a potential movie j is calculated as:

$$\text{CF\_score}(j) = \sum_{\text{movies liked by } u} \text{similarity}(j, m)$$

Where:

- similarity(j,m) is the cosine similarity between movie j and movie m.
- The movie j is excluded from the list of recommended movies if it's already liked by the user.

Apriori Association Rules:

Apriori works by finding frequent co-occurrences between movies liked by other users. If a user likes Movie A, the system checks for rules like "If a user liked A, they also liked B" (represented as A→B). The recommendation score for movie BBB is incremented by a fixed value (e.g., 1) if such a rule exists. This can be represented as:

$$\text{Apriori\_score}(B) = \sum_{\text{rules } A \to B} \text{support}(A \to B)$$

Where:

- support(A→B) is the support value for the rule, which indicates how frequently the combination of A and B appears together in the dataset.

Combined Recommendation:

The final recommendation score for a movie is a combination of both CF and Apriori scores:

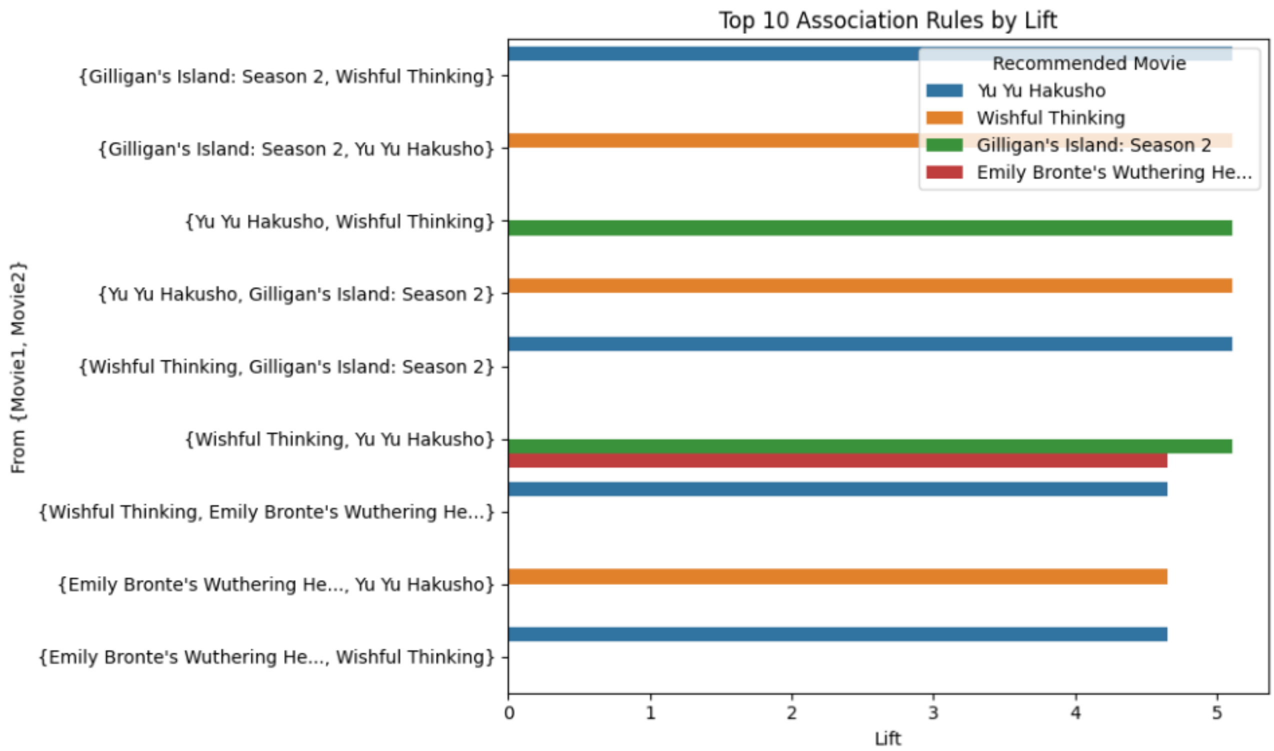
Final_score(i)=CF_score(i)+Apriori_score(i)

The system then returns the top N movies with the highest final scores.

BAR CHART COMPARISION:

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

# Load the final rules
rules_df = pd.read_csv("generated_association_rules1.csv")

# Plot top 10 by Lift
top_lift_rules = rules_df.sort_values(by="Lift", ascending=False).head(10)

plt.figure(figsize=(10,6))
sns.barplot(x="Lift", y="From", hue="To", data=top_lift_rules)
plt.title("Top 10 Association Rules by Lift")
plt.xlabel("Lift")
plt.ylabel("From {Movie1, Movie2}")
plt.legend(title="Recommended Movie")
plt.tight_layout()
plt.show()
```

OUTPUT:



Graph Overview

- Title: Top 10 Association Rules by Lift
- Purpose: Displays associations between two movie recommendations based on lift values.
- Key Elements:
- X-axis represents lift values.
- Each bar indicates a pair of movies that are frequently recommended together, highlighting correlations.
- Movies Involved: Includes titles like "Gilligan's Island: Season 2," "Yu Yu Hakusho," "Wishful Thinking," and "Emily Bronte's Wuthering Heights."

METRICES :

Metrics are quantitative measures used to evaluate and assess the strength, relevance, and quality of data patterns or models.

- Support: Measures how frequently an item or itemset appears in the dataset.

$$\text{Support}(A) = \frac{\text{Frequency of Itemset } A}{\text{Total Transactions}}$$

- Confidence: Measures the likelihood that the consequent (B) occurs given the antecedent (A).

$$\text{Confidence}(A \to B) = \frac{\text{Support}(A \cup B)}{\text{Support}(A)}$$

- Lift: Measures the strength of the association between A and B compared to random chance.

$$\text{Lift}(A \to B) = \frac{\text{Support}(A \cup B)}{\text{Support}(A) \times \text{Support}(B)}$$

- Conviction: Measures how much more likely the rule is to be true than false.

$$\text{Conviction}(A \to B) = \frac{1 - \text{Support}(B)}{1 - \text{Confidence}(A \to B)}$$

- Leverage: Measures the difference between observed and expected co-occurrence of A and B.

$$\text{Leverage}(A \to B) = \text{Support}(A \cup B) - (\text{Support}(A) \times \text{Support}(B))$$

These metrics help assess the quality and strength of association rules.

Output:

```
import pandas as pd

# Load your CSV file
df = pd.read_csv('filtered_association_rules.csv')

# Step 1: Calculate Support = Confidence / Lift
df['Support'] = df['Confidence'] / df['Lift']

# Step 2: Estimate Support for 'To' if not available
# If you have transaction data and item counts, you can compute support for 'To' here.
# For this example, let's assume you estimate the Support of 'To' manually or via transaction data.
# Replace 'Support_To' with the actual value or calculation if available.

# Example: If you already know the support for 'To' (item2), you can manually add it.
# Here, we just create a dummy support for 'To' to demonstrate.
df['Support_To'] = 0.3  # Example Support for 'To', you should replace this with actual values

# Step 3: Calculate Conviction (Requires Support of 'To')
df['Conviction'] = (1 - df['Support_To']) / (1 - df['Confidence'])

# Step 4: Leverage Calculation (Requires both supports for 'From' and 'To')
# This assumes you have access to support for 'From' and 'To'. We'll calculate Leverage here.
# Example: Let's assume support of 'From' and 'To' are known. Here we demonstrate with dummy values.
df['Leverage'] = df['Support'] - (df['Support'] * df['Support_To'])

# Step 5: Save the updated DataFrame with new columns
df.to_csv('updated_association_rules.csv', index=False)

# Display the updated DataFrame
print(df.head(1000))  # Preview the data
```

```
                                 From                             To  \
0     Cartoon Network Halloween: ...                            Fame
1                                Fame  Cartoon Network Halloween: ...
2          Gilligan's Island: Season 2                  Yu Yu Hakusho
3                        X: The Movie  Emily Bronte's Wuthering He...
4     Emily Bronte's Wuthering He...                    X: The Movie
..                                ...                            ...
995                            Planes            Dark Shadows: Vol. 6
996               Ultimate Attraction                     Dirty Tiger
997    Call Me: The Rise and Fall ...                          Poison
998    The Firm: Maximum Cardio Bu...        My Side of the Mountain
999                  A Cry in the Wild                         Poison

     Confidence      Lift   Support  Support_To  Conviction  Leverage
0      0.725716  7.332726  0.098969         0.3    2.552099  0.069279
1      0.651735  7.332726  0.088880         0.3    2.009962  0.062216
2      0.615576  5.644895  0.109050         0.3    1.820907  0.076335
3      0.540049  5.219522  0.103467         0.3    1.521900  0.072427
4      0.504148  5.219522  0.096589         0.3    1.411712  0.067612
..          ...       ...       ...         ...         ...       ...
995    0.690889  2.190712  0.315372         0.3    2.264561  0.220760
996    0.507641  2.190354  0.231762         0.3    1.421728  0.162234
997    0.670917  2.190134  0.306336         0.3    2.127123  0.214435
998    0.606504  2.189139  0.277051         0.3    1.778923  0.193936
999    0.670571  2.189003  0.306336         0.3    2.124886  0.214435

[1000 rows x 8 columns]
```

FP GROWTH IMPLEMENTATION:

To improve efficiency, the movie recommendation system was also implemented using the FP-Growth algorithm, which eliminates candidate generation by constructing a compact FP-Tree. Frequent itemsets were mined directly from the tree, enabling faster rule generation compared to Apriori. The snapshots below illustrate the FP-Growth implementation and the generated association rules.

```python
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth, association_rules
import time
df = pd.read_csv('merged.csv', usecols=['Cust_Id', 'Rating', 'name'])
df = df.head(4000)
df = df.drop_duplicates(subset=['Cust_Id', 'name'])
df['watched'] = (df['Rating'] > 0).astype(int)
pivot_df = df.pivot_table(index='Cust_Id', columns='name', values='watched', aggfunc='max', fill_value=0)
pivot_df = pivot_df.astype(bool)
print("Pivot Table Shape:", pivot_df.shape)
start_time = time.time()
frequent_itemsets = fpgrowth(pivot_df, min_support=0.001, use_colnames=True)
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0)
end_time = time.time()
elapsed_time = end_time - start_time
print(f"Time taken to run FP-Growth: {elapsed_time:.2f} seconds")
print("\n📌 Top Frequent Itemsets:")
print(frequent_itemsets.head())
if not rules.empty:
    print("\n📌 Top Association Rules:")
    print(rules.head())
else:
    print("\n📌 No association rules were found.")
```

OUTPUT:

```
📌 Top Frequent Itemsets:
    support                          itemsets
0  0.432080                           (8 Man)
1  0.109235      (Isle of Man TT 2004 Review)
2  0.360706    (Paula Abdul's Get Up & Dance)
3  0.028396                            (Sick)
4  0.051420                 (Dinosaur Planet)

📌 Top Association Rules:
                                     antecedents  \
0      (Isle of Man TT 2004 Review, Dinosaur Planet)
1  (Isle of Man TT 2004 Review, Paula Abdul's Get...
2    (Dinosaur Planet, Paula Abdul's Get Up & Dance)
3                       (Isle of Man TT 2004 Review)
4                                  (Dinosaur Planet)

                                     consequents  antecedent support  \
0                   (Paula Abdul's Get Up & Dance)            0.001535
1                              (Dinosaur Planet)            0.002814
2                     (Isle of Man TT 2004 Review)            0.003581
3    (Dinosaur Planet, Paula Abdul's Get Up & Dance)            0.109235
4  (Isle of Man TT 2004 Review, Paula Abdul's Get...            0.051420

   consequent support   support  confidence       lift  representativity  \
0            0.360706  0.001535    1.000000   2.772340               1.0
1            0.051420  0.001535    0.545455  10.607870               1.0
2            0.109235  0.001535    0.428571   3.923386               1.0
3            0.003581  0.001535    0.014052   3.923386               1.0
4            0.002814  0.001535    0.029851  10.607870               1.0
```

Recommendation System:

```
[11]: import pandas as pd
import ast
def parse_frozenset_string(s):
    try:
        if s.startswith("frozenset("):
            s = s[len("frozenset("):-1]
        return frozenset(ast.literal_eval(s))
    except:
        return frozenset()
rules = pd.read_csv("association_rules.csv")
rules['antecedents'] = rules['antecedents'].astype(str).apply(parse_frozenset_string)
rules['consequents'] = rules['consequents'].astype(str).apply(parse_frozenset_string)
def normalize_movie_name(movie_name):
    return movie_name.strip().lower()
def recommend_movies(movie_name, rules_df, top_n=5):
    normalized_movie_name = normalize_movie_name(movie_name)
    matching_rules = rules_df[rules_df['antecedents'].apply(lambda x: any(normalize_movie_name(movie) == normalized_movie_name for movie in x))]
    if matching_rules.empty:
        print(f"\n🚫 No recommendations found for '{movie_name}'. Try another movie.")
        return []
    matching_rules = matching_rules.sort_values(by=['confidence', 'lift'], ascending=False)
    recommended = []
    for consequents in matching_rules['consequents']:
        recommended.extend(list(consequents))
    recommended = list(dict.fromkeys(recommended))
    # Remove the original movie if present
    recommended = [movie for movie in recommended if normalize_movie_name(movie) != normalized_movie_name]
    print(f"\n🎬 Recommended Movies for '{movie_name}':")
    for i, movie in enumerate(recommended[:top_n], start=1):
        print(f"{i}. {movie}")
    return recommended[:top_n]
user_movie = input("Enter a movie name you liked: ").strip()
recommend_movies(user_movie, rules)
```

OUTPUT:

```
Enter a movie name you liked:  Dinosaur Planet

🎬 Recommended Movies for 'Dinosaur Planet':
1. Paula Abdul's Get Up & Dance
2. Isle of Man TT 2004 Review

["Paula Abdul's Get Up & Dance", 'Isle of Man TT 2004 Review']
```

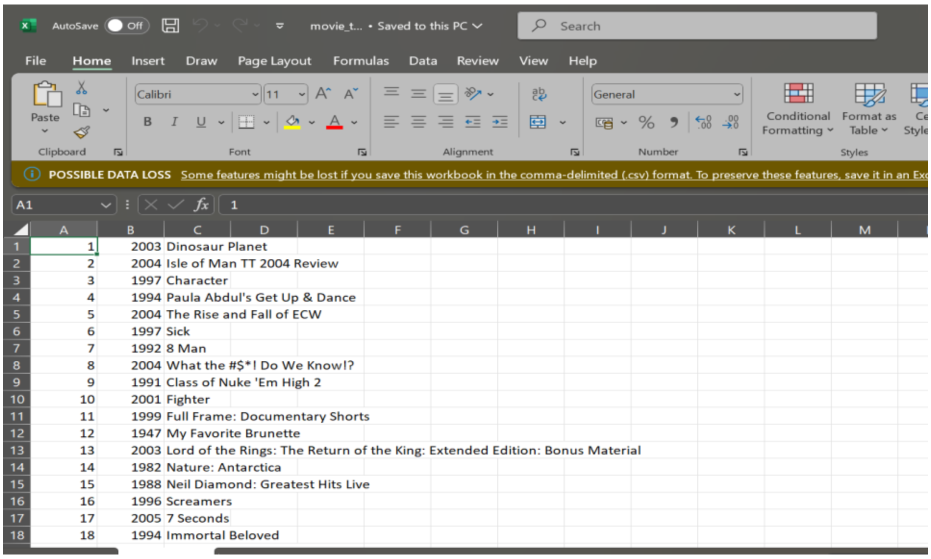
Comparative Analysis of Apriori and FP-Growth in Movie Recommendation:

To evaluate performance, both Apriori and FP-Growth algorithms were implemented for generating movie recommendations. While Apriori follows a candidate generation-and-pruning approach, FP-Growth uses a more efficient FP-Tree structure to mine frequent patterns without generating candidates. In the

Apriori implementation, frequent itemsets were generated up to length 3, as longer itemsets did not yield significantly stronger associations. It was observed that the lift and confidence values decreased with longer itemsets, indicating weaker relationships. In contrast, FP-Growth handled larger itemsets more efficiently and produced results faster, making it more scalable and suitable for larger datasets in recommendation systems.

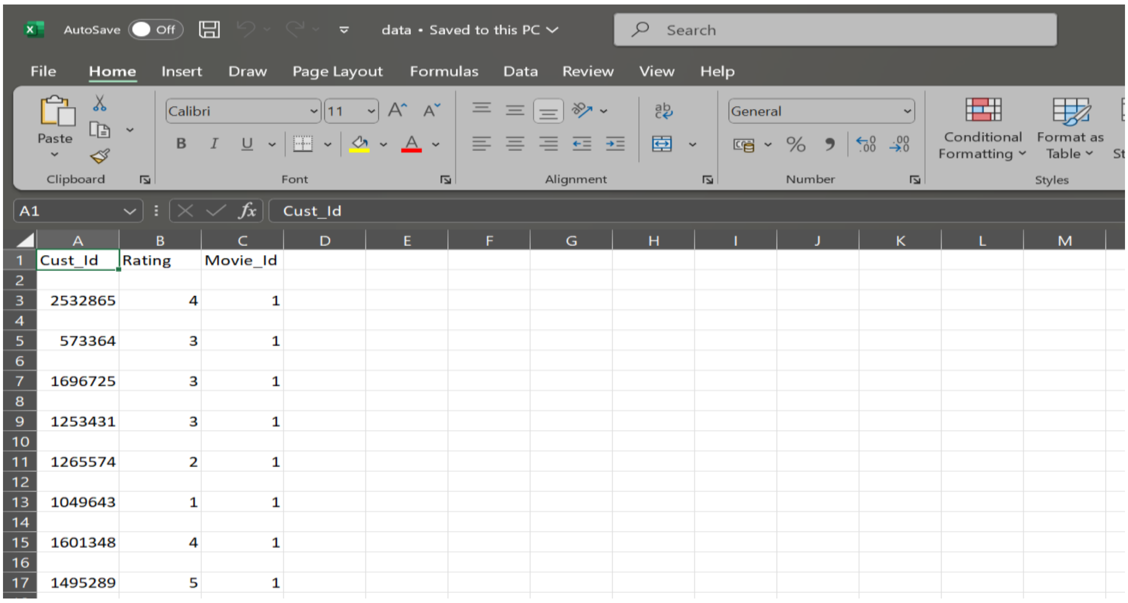
FILES SAVED AND CREATED :

MOVIE_TITLES.CSV



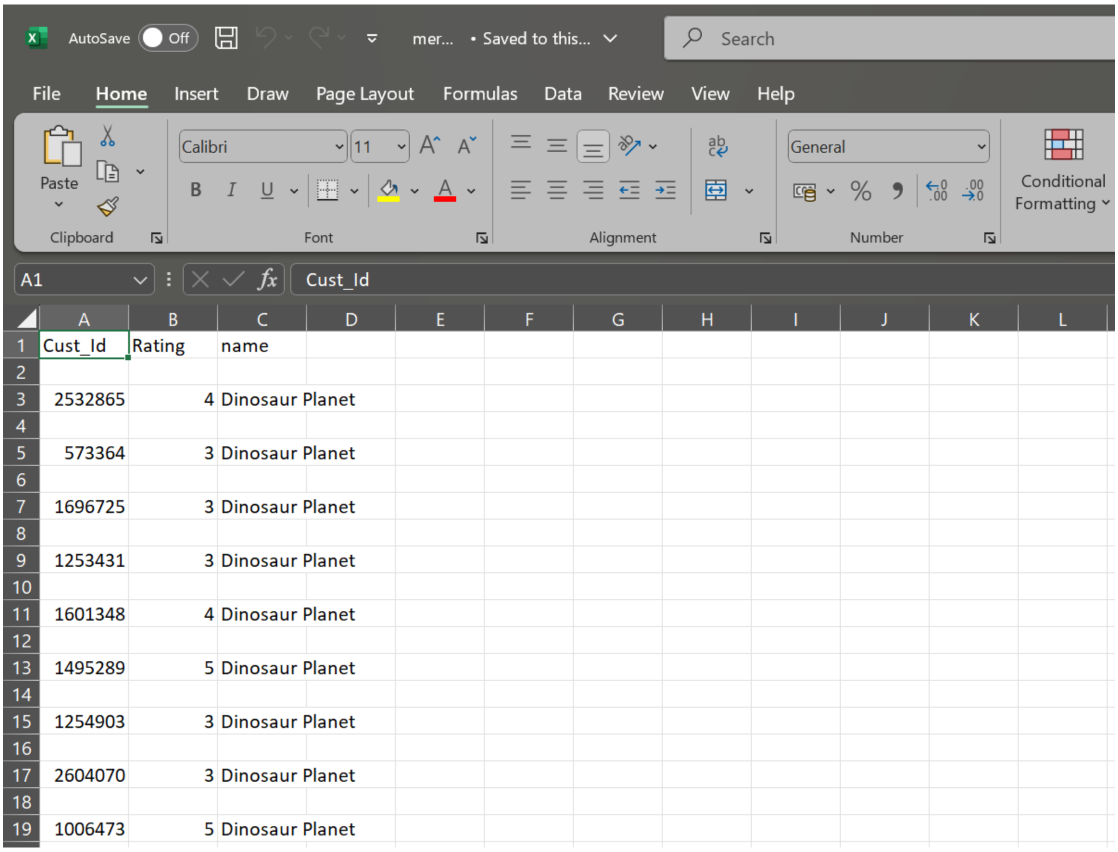
1	2003	Dinosaur Planet
2	2004	Isle of Man TT 2004 Review
3	1997	Character
4	1994	Paula Abdul's Get Up & Dance
5	2004	The Rise and Fall of ECW
6	1997	Sick
7	1992	8 Man
8	2004	What the #$*! Do We Know!?
9	1991	Class of Nuke 'Em High 2
10	2001	Fighter
11	1999	Full Frame: Documentary Shorts
12	1947	My Favorite Brunette
13	2003	Lord of the Rings: The Return of the King: Extended Edition: Bonus Material
14	1982	Nature: Antarctica
15	1988	Neil Diamond: Greatest Hits Live
16	1996	Screamers
17	2005	7 Seconds
18	1994	Immortal Beloved

DATA.CSV

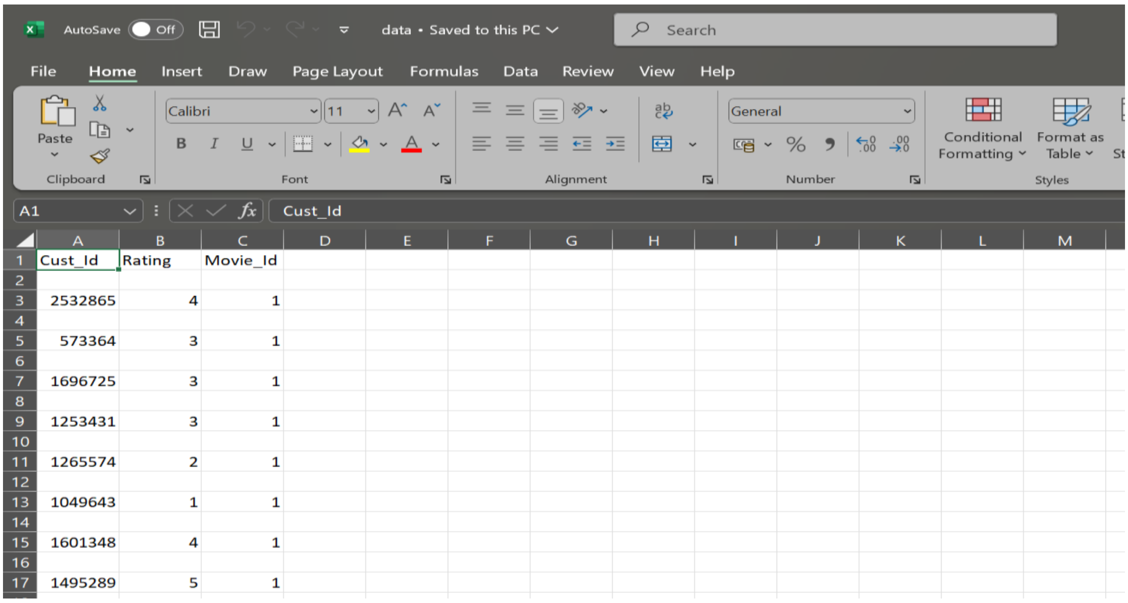


Cust_Id	Rating	Movie_Id
2532865	4	1
573364	3	1
1696725	3	1
1253431	3	1
1265574	2	1
1049643	1	1
1601348	4	1
1495289	5	1

MERGED.CSV:



Cust_Id	Rating	name
2532865	4	Dinosaur Planet
573364	3	Dinosaur Planet
1696725	3	Dinosaur Planet
1253431	3	Dinosaur Planet
1601348	4	Dinosaur Planet
1495289	5	Dinosaur Planet
1254903	3	Dinosaur Planet
2604070	3	Dinosaur Planet
1006473	5	Dinosaur Planet



Cust_Id	Rating	Movie_Id
2532865	4	1
573364	3	1
1696725	3	1
1253431	3	1
1265574	2	1
1049643	1	1
1601348	4	1
1495289	5	1

FILTERED_ASSOCIATION_RULES.CSV

jupyter filtered_association_rules.csv Last Checkpoint: 9 hours ago

File Edit View Settings Help

Delimiter: ,

	From	To	Confidence	Lift
1	Bent	R.E.M.: Road Movie	0.7102738249245364	9.887328540474787
2	R.E.M.: Road Movie	Bent	0.7776564667001091	9.887328540474787
3	The Stand	Die Hard 2: Die Harder	0.9042952058004576	5.763713616650218
4	Die Hard 2: Die Harder	The Stand	0.8863473620212119	5.763713616650218
5	Mon Oncle	abor: The Mobile Police...	0.7165616987378955	4.90144534892923
6	Blue Planet: IMAX	ts of the Dead: Amazon...	0.7784445016739634	4.751649009053403
7	The Bogus Witch Project	a Abdul's Get Up & Dance	0.5338224623075063	4.651902587855659
8	ncredibles: Bonus Mate...	Fuzz	0.5224452312191114	4.639837681952141
9	Fuzz	ncredibles: Bonus Mate...	0.8139460456710405	4.639837681952141
10	Empire of the Ants	'P: Most Valuable Primate	0.5215351964473613	4.559482384018879
11	'P: Most Valuable Primate	Empire of the Ants	0.5419284297918867	4.559482384018879
12	nal Sunshine of the Spo...	Farscape: Season 2	0.6302989061948687	4.547301671495655
13	Trigun	Dragon Ball GT	0.8522490853002368	4.4270181352652775
14	Mon Oncle	Spider-Man vs. Doc Ock	0.5942595404645142	4.367513214528176
15	elsing: The London Ass...	Hard Eight	0.5098009987329507	4.276063175013237
16	Fuzz	Farscape: Season 2	0.590560816280427	4.260610577930408
17	The Green Berets	Spider-Man vs. Doc Ock	0.5789789656245831	4.255208559752556
18	Mon Oncle	Dragon Ball GT	0.7983868960195165	4.147230344508263
19	King of Queens: Season 2	Dragon Ball GT	0.7978352941176471	4.144365041787331
20	Dragon Ball GT	abor: The Mobile Police...	0.5923670678397216	4.051925765270182
21	abor: The Mobile Police...	Dragon Ball GT	0.7800397291458233	4.051925765270182
22	Wake Island	Uptown Girls	0.5360651910956433	4.024519851351428
23	'P: Most Valuable Primate	u Wanna Know a Secret?	0.5323844999166064	3.9989234588911557
24	'P: Most Valuable Primate	The Crucible	[illegible]	[illegible]

Conclusion:

Our system successfully identifies movie relationships using Apriori-based association rules, improving recommendation accuracy. The approach proves effective for large datasets, though future enhancements could integrate collaborative filtering and hybrid models for even better recommendations.