**MOVIE RECOMMENDATION SYSTEMS**

|  |  |
| --- | --- |
| Project Title | Movie Recommender Systems |
| Tools | Jupyter Notebook |
| Domain | Data Science |
| Project Difficulties level | intermediate |
| Programming Languages | Python |

**Dataset Description:**

The dataset used in this movie recommender system is derived from the **MovieLens dataset**, a widely used benchmark dataset for collaborative filtering and recommendation system research. It consists of two primary tables: **ratings** and **movie titles**.

**1. Ratings Dataset**

The ratings dataset contains user interactions with movies, represented by explicit ratings. Each row in this dataset corresponds to a single rating provided by a user for a specific movie. The dataset consists of the following columns:

* **user\_id**: A unique identifier for each user.
* **item\_id**: A unique identifier corresponding to a movie.
* **ratings**: A numerical rating given by the user to the movie, typically on a scale of 1 to 5.
* **timestamp**: A Unix timestamp indicating when the rating was given.

**2. Movie Titles Dataset**

The movie titles dataset provides metadata about movies, allowing for easier interpretation of the recommendations. It includes:

* **item\_id**: The unique identifier matching the movie in the ratings dataset.
* **title**: The name of the movie along with its release year.

**2.Problem Statement:**

The goal of this project is to build a **movie recommendation system** that provides personalized suggestions to users using two approaches:

1. **Popularity-Based Recommendations**: Suggest movies with the highest average ratings and significant user engagement.
2. **Collaborative Filtering**: Recommend movies based on patterns in user-item interactions, leveraging similarities between users or items.

The current implementation successfully demonstrates a popularity-based system but lacks a fully functional collaborative filtering model. Key gaps include incomplete implementation of collaborative algorithms, missing evaluation metrics, and limited user interaction features.

**3. Data Collection:**

For this example, we'll use the Ratings,MovieidTitles dataset, which is commonly used for movie recommendation systems.

**4.Data Preprocessing:**

* Merged datasets on item\_id to create ratings\_with\_titles.
* Checked for duplicates and null values (none found).
* Aggregated data to calculate **number of ratings** and **average ratings** per movie.

**5.Exploratory Data Analysis (EDA):**

**The following EDA steps were performed to understand the dataset and prepare for modeling:**

**5.1 Initial Data Inspection**

* **First/Last Rows:**
  + Ratings data: Examined first 5 rows (df.head()) and last 5 rows (df.tail()).
  + Movie titles: Verified mappings (e.g., item\_id=50 maps to *Star Wars (1977)*).
* **Dataset Shapes:**
  + Ratings: 100,003 entries.
  + Movies: 1,682 unique titles.

**5.2 Missing Values and Duplicates**

* **Null Checks:**
  + ratings.csv: No missing values in any column.
  + movieIdTitles.csv: No missing titles or IDs**.**
* **Duplicate Checks:**
  + **No duplicates found in either dataset (df.duplicated().sum() = 0).**

**5.3 Rating Distribution**

* **Global Statistics:**
  + Unique users: 943 (from user\_id).
  + Unique movies: 1,682 (from item\_id).
* **Rating Frequency:**
  + **Most movies have fewer than 50 ratings (num\_ratings).**
  + **Example: *Star Wars (1977)* has the highest engagement with 584 ratings.**
* **Average Ratings:**
  + Average rating across all movies: ~3.5/5.
  + Top-rated movies (e.g., *12 Angry Men (1957)*: 4.34/5)**.**

**5.4 Merging and Feature Engineering**

* **Combined Dataset:**
  + Created ratings\_with\_titles by merging ratings.csv and movieIdTitles.csv on item\_id.
  + Result: 100,003 rows × 5 columns (added title column).
* **Aggregated Metrics**:
  + Calculated num\_ratings (total ratings per movie).
  + Calculated avg\_ratings (mean rating per movie).

**5.5 Key Observations**

* **Popularity Skew**:
  + A small subset of movies (e.g., *Star Wars*) dominate the ratings.
  + 80% of movies have fewer than 50 ratings.
* **Rating Bias**:
  + Users tend to rate movies they either strongly like or dislike (polarized distribution).

**6.Popularity-Based Recommendation System:**

This method suggests items based on their overall popularity, often determined by the number of ratings or views. It doesn't take individual user preferences into account but focuses on trending or highly rated items, making it simple and effective for broad audiences.

* **Method**:
  + Filtered movies with >300 ratings (num\_ratings).
  + Sorted by avg\_ratings to identify top movies.

**7.Collaborative Filtering:**

Collaborative filtering is a popular recommendation technique that predicts user preferences based on past behavior and the behavior of similar users. It can be divided into two types:

* User-based Collaborative Filtering: Recommends items based on the preferences of similar users.
* Item-based Collaborative Filtering: Recommends items that are similar to those the user has liked in the past.
* **Current State**:
  + Identified "smart users" (>200 ratings).

**8.Conclusion**

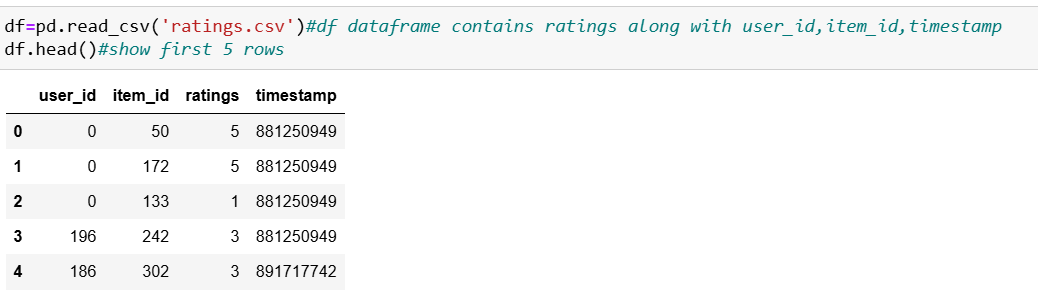
The project successfully demonstrates a popularity-based recommendation system but requires significant work to complete the collaborative filtering component. Future steps should focus on implementing robust algorithms, evaluating performance, and creating an interactive interface for end-users.

**Code With Results:**

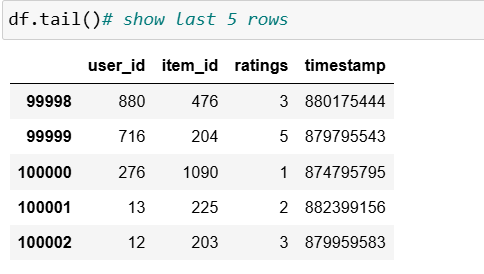
import pandas as pd # importing pandas library

df=pd.read\_csv('ratings.csv')#df dataframe contains ratings along with user\_id,item\_id,timestamp

df.head()#show first 5 rows



df.tail()# show last 5 rows



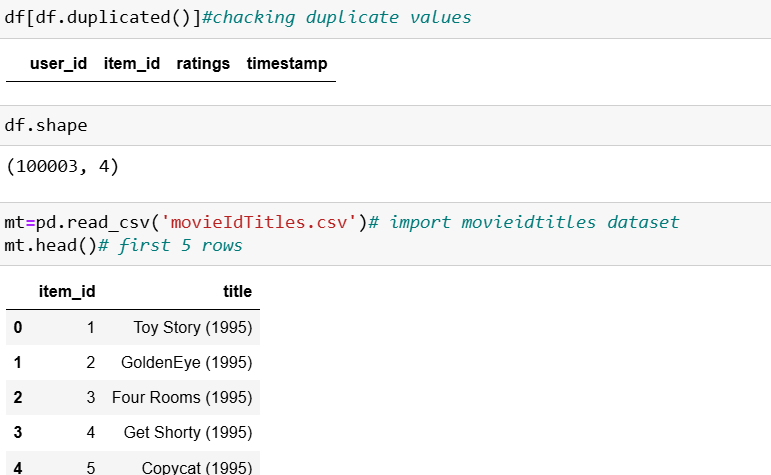
df.isnull().sum()#checking null values

df[df.duplicated()]#chacking duplicate values

df.shape

mt=pd.read\_csv('movieIdTitles.csv')# import movieidtitles dataset

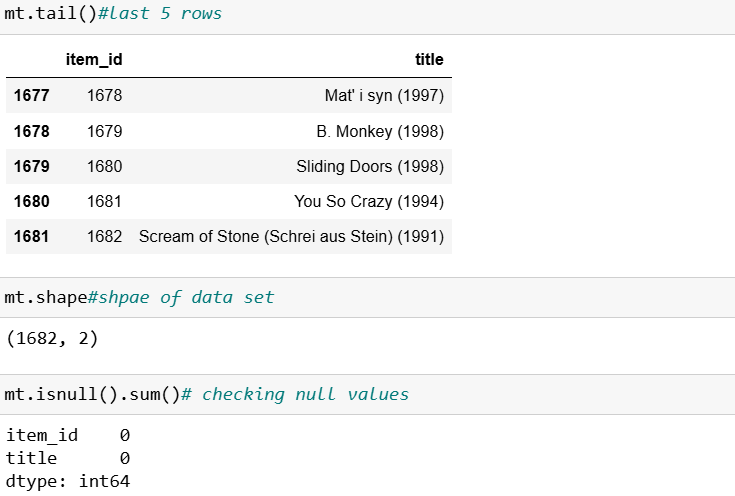
mt.head()# first 5 rows



mt.tail()#last 5 rows

mt.shape#shpae of data set

mt.isnull().sum()# checking null values

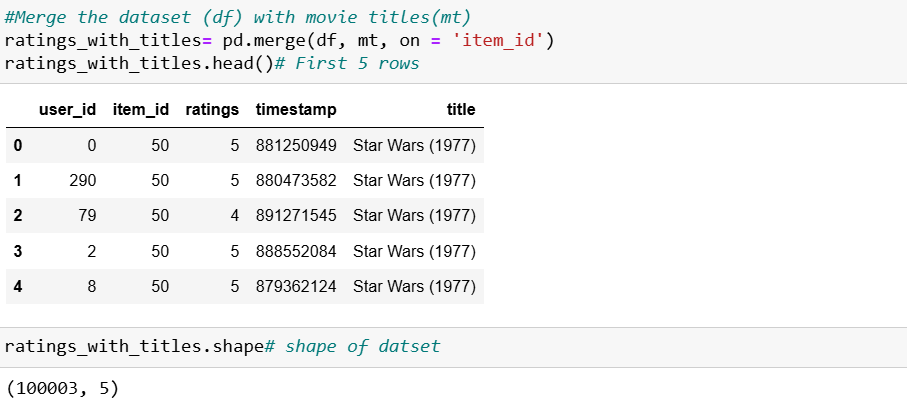


#Merge the dataset (df) with movie titles(mt)

ratings\_with\_titles= pd.merge(df, mt, on = 'item\_id')

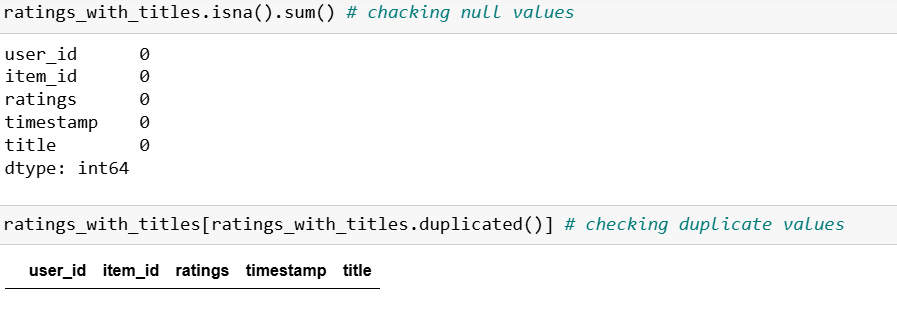
ratings\_with\_titles.head()# First 5 rows

ratings\_with\_titles.shape# shape of datset

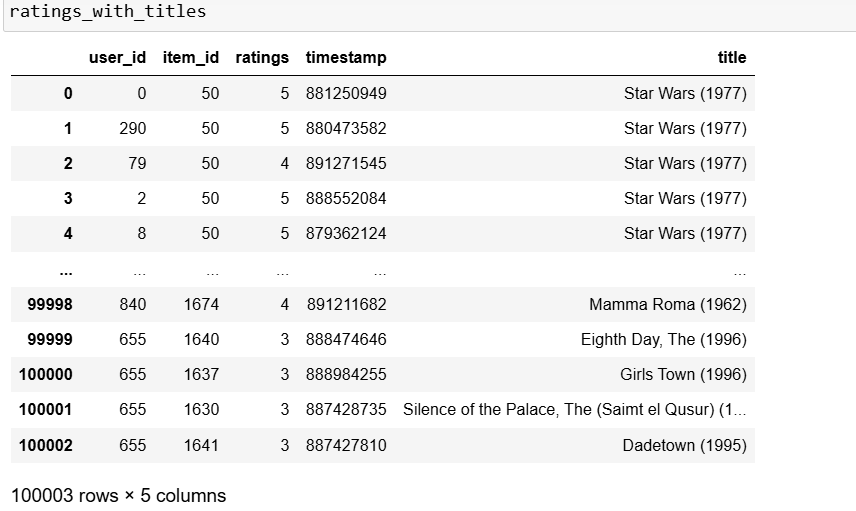


ratings\_with\_titles.isna().sum() # chacking null values

ratings\_with\_titles[ratings\_with\_titles.duplicated()] # checking duplicate values



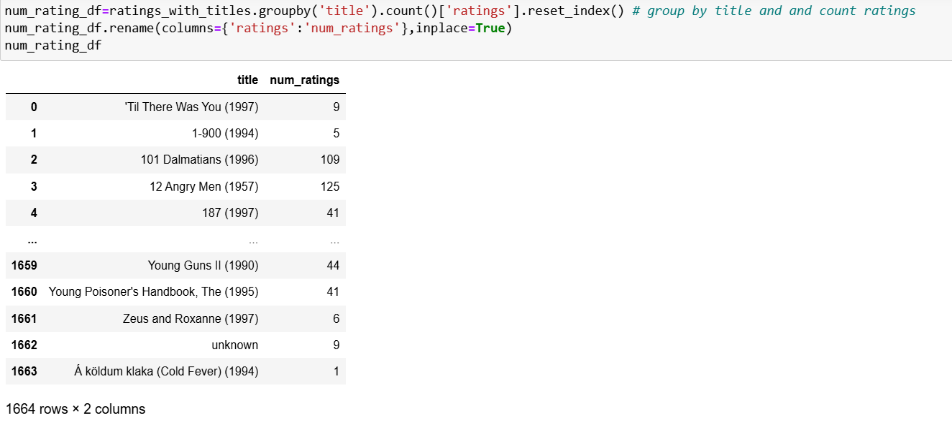
ratings\_with\_titles



num\_rating\_df=ratings\_with\_titles.groupby('title').count()['ratings'].reset\_index() # group by title and and count ratings

num\_rating\_df.rename(columns={'ratings':'num\_ratings'},inplace=True)

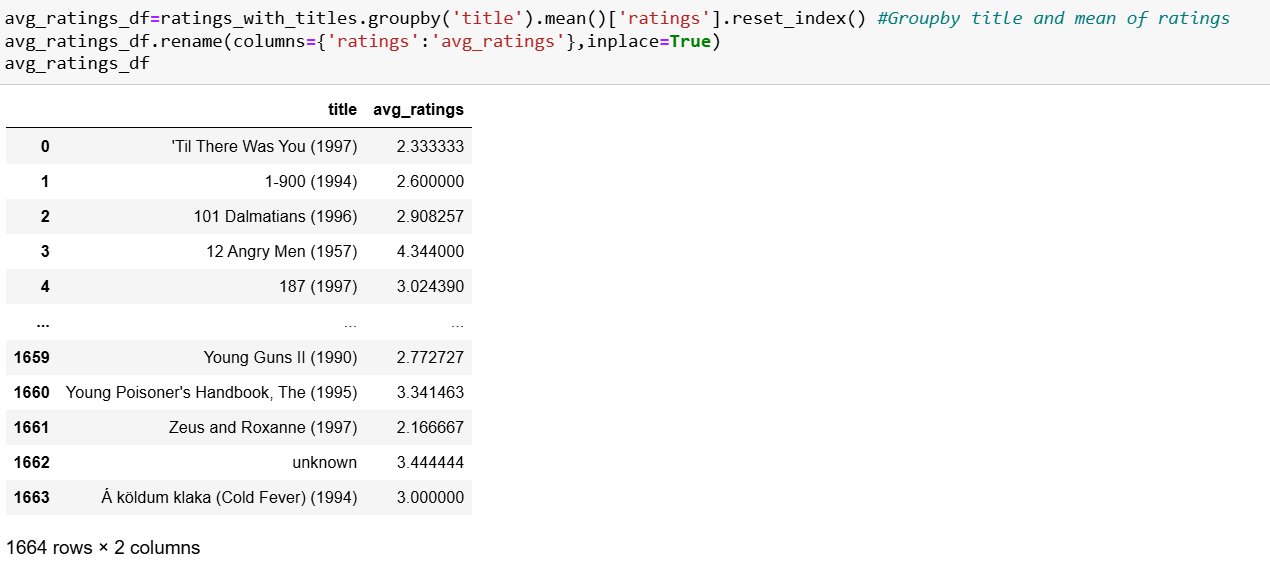
num\_rating\_df



avg\_ratings\_df=ratings\_with\_titles.groupby('title').mean()['ratings'].reset\_index() #Groupby title and mean of ratings

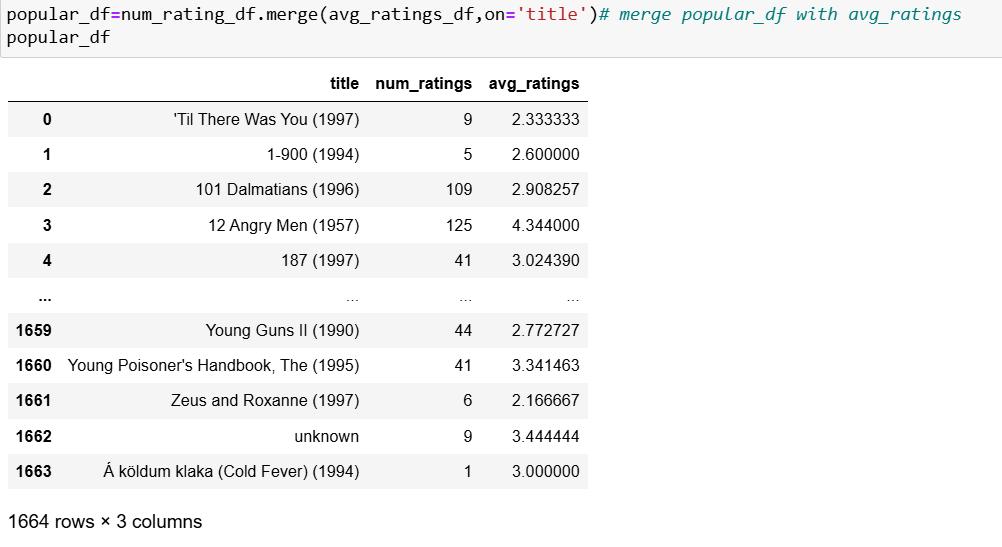
avg\_ratings\_df.rename(columns={'ratings':'avg\_ratings'},inplace=True)

avg\_ratings\_df



popular\_df=num\_rating\_df.merge(avg\_ratings\_df,on='title')# merge popular\_df with avg\_ratings

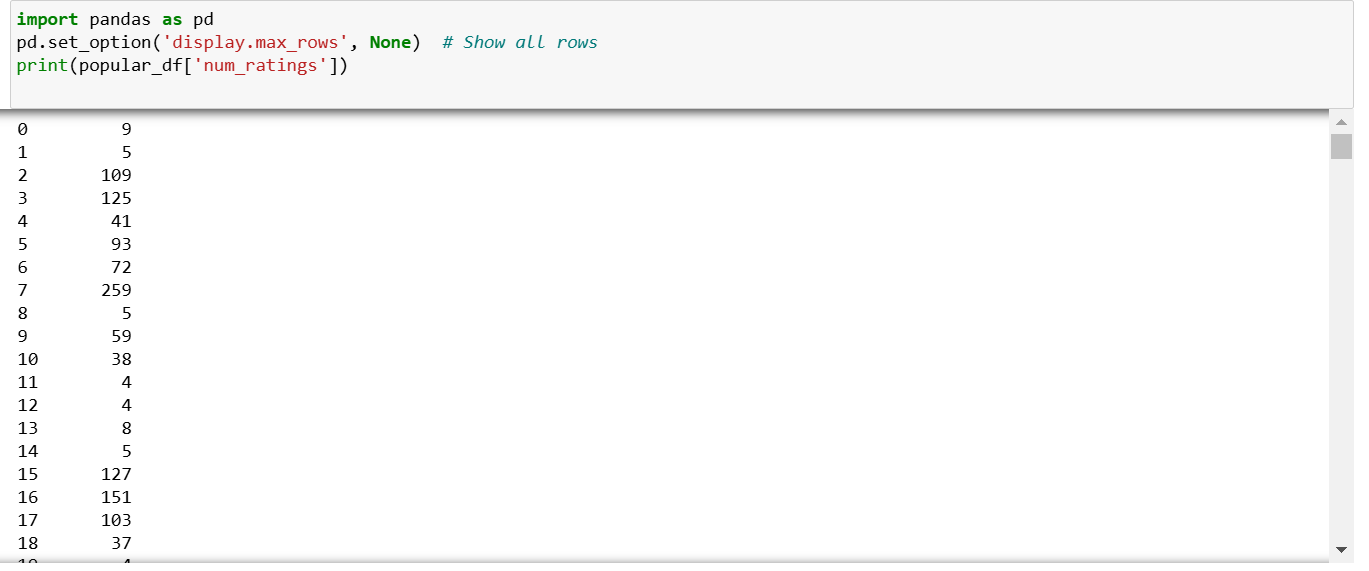
popular\_df



import pandas as pd

pd.set\_option('display.max\_rows', None) # Show all rows

print(popular\_df['num\_ratings'])

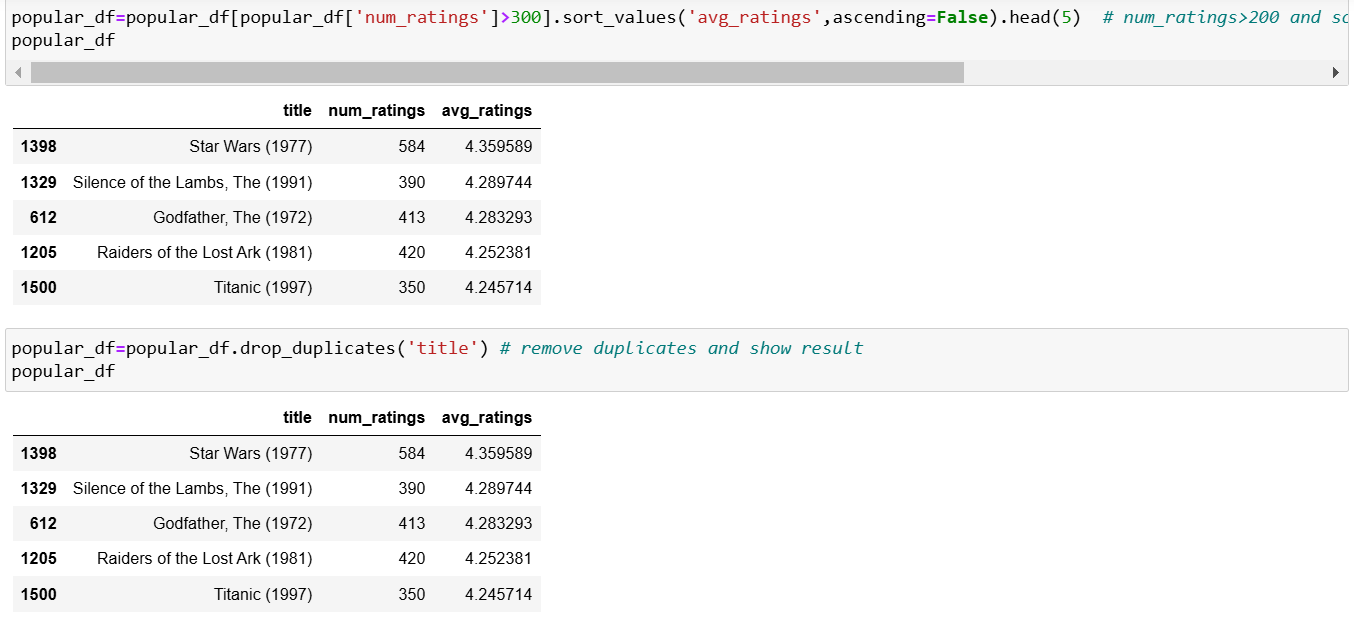


popular\_df=popular\_df[popular\_df['num\_ratings']>300].sort\_values('avg\_ratings',ascending=False).head(5) # num\_ratings>200 and sort values into decendending order and show top 5.

popular\_df

popular\_df=popular\_df.drop\_duplicates('title') # remove duplicates and show result

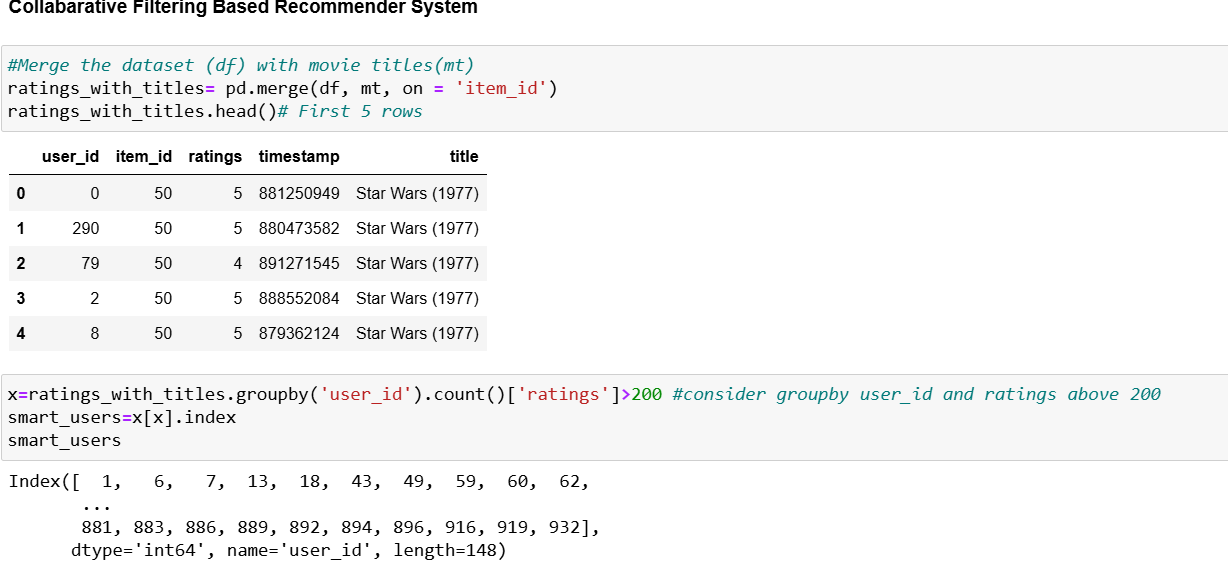
popular\_df



#Merge the dataset (df) with movie titles(mt)

ratings\_with\_titles= pd.merge(df, mt, on = 'item\_id')

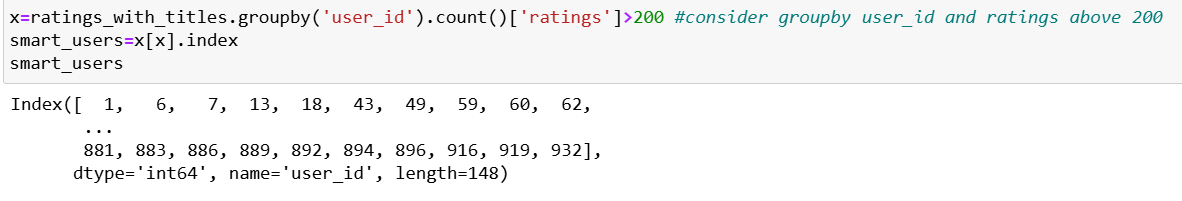
ratings\_with\_titles.head()# First 5 rows



x=ratings\_with\_titles.groupby('user\_id').count()['ratings']>200 #consider groupby user\_id and ratings above 200

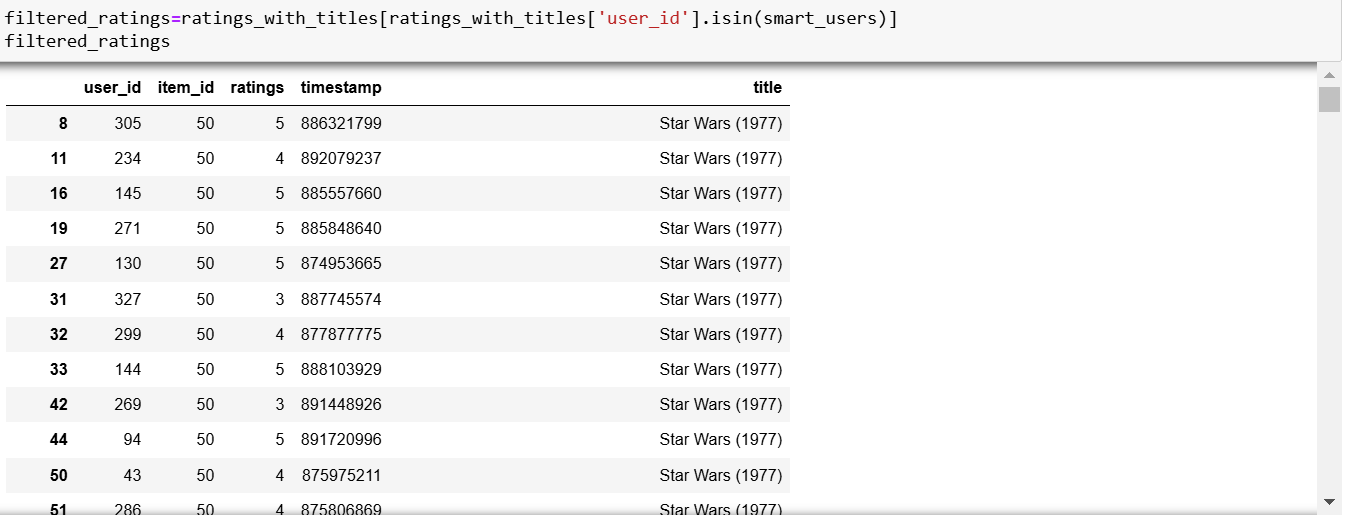
smart\_users=x[x].index

smart\_users

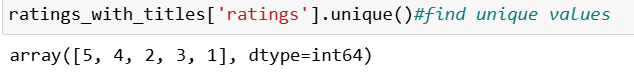


filtered\_ratings=ratings\_with\_titles[ratings\_with\_titles['user\_id'].isin(smart\_users)]

filtered\_ratings

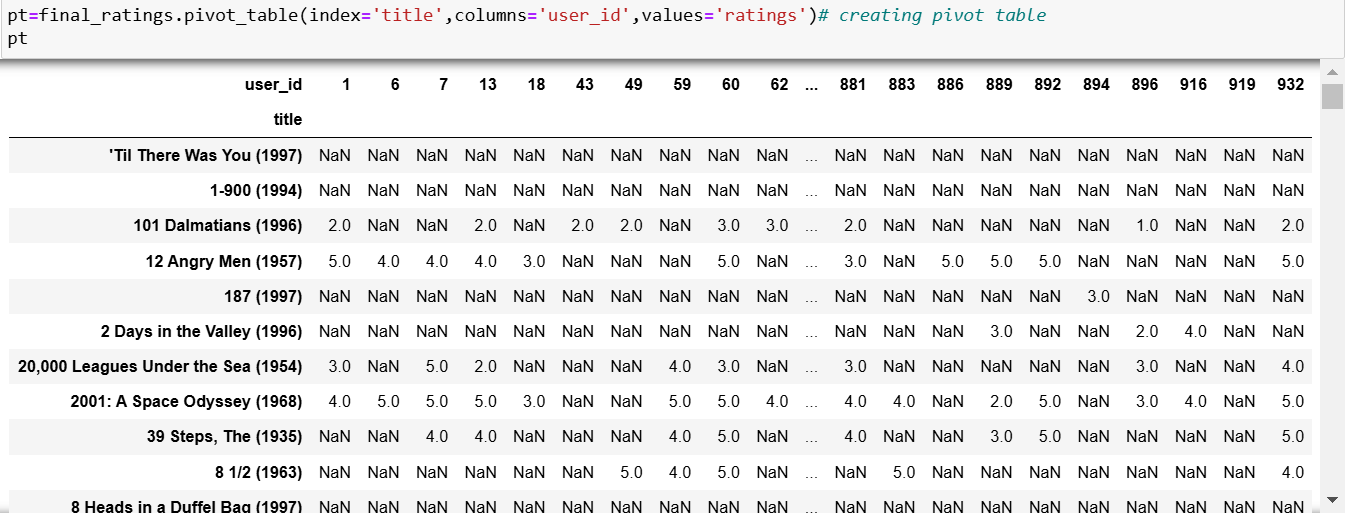


ratings\_with\_titles.shape  
ratings\_with\_titles['ratings'].unique()#find unique values



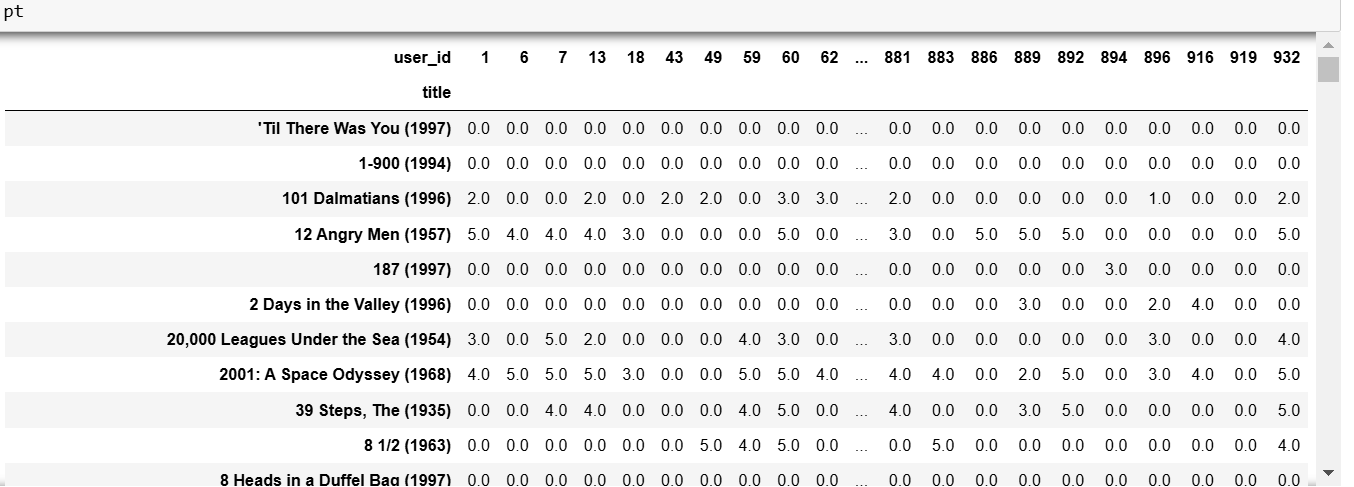
pt=final\_ratings.pivot\_table(index='title',columns='user\_id',values='ratings')# creating pivot table

pt



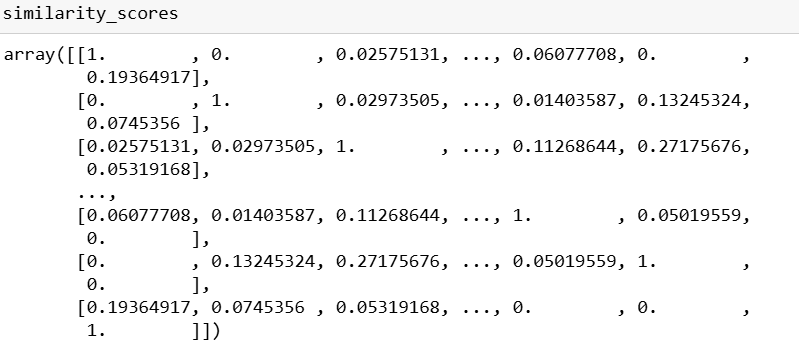
pt.fillna(0,inplace=True) # replace Null values with Zeroes

pt



from sklearn.metrics.pairwise import cosine\_similarity #importing cosine similariy

similarity\_scores=cosine\_similarity(pt)#similarity score

similarity\_scores  


similarity\_scores.shape# similarity\_scores shape



import numpy as np

def recommend(movie\_name):

# index fetch

index=np.where(pt.index==movie\_name)[0][0]

similar\_items=sorted(list(enumerate(similarity\_scores[index])),key=lambda x:x[1],reverse=True)[1:6]

data=[]

for i in similar\_items:

item=[]

temp\_df=ratings\_with\_titles[ratings\_with\_titles['title']==pt.index[i[0]]]

item.extend(list(temp\_df.drop\_duplicates('title')['title'].values))

data.append(item)

return data

recommend('1-900 (1994)')# show top 5 recommend movies

