Name: Kaushik Kotian Roll No.:30 Div:D15B Batch:B

EXPERIMENT: 08

<u>AIM:</u> Recommendation system using Machine Learning

PROBLEM STATEMENT:

- 1. Use any dataset
- 2. Use any type of Recommendation technique

THEORY:

What is a Recommendation System?

A recommendation system is an algorithmic tool that provides suggestions for items a user might like, based on their past behavior, preferences, and interactions. These systems are pivotal in helping users discover products or content that align with their interests, significantly enhancing user experience in various digital platforms such as online stores, streaming services, and social media. The essence of a recommendation system lies in its ability to personalize the browsing experience, steering users towards items they might not have discovered otherwise.

Types of Recommendation Systems:

Recommendation systems can be broadly classified into several types, each harnessing different methodologies to curate personalized recommendations:

- 1. <u>Content-Based Filtering:</u> This technique recommends items by comparing the description of the items to a profile of the user's preferences. The system analyzes items previously engaged with by the user and builds recommendations for similar items. For instance, in the context of book recommendations, if a user likes books about space exploration, the system might recommend other books with similar themes or written by similar authors.
- 2. <u>Collaborative Filtering:</u> Perhaps the most well-known approach, collaborative filtering, generates recommendations based on the knowledge of users' attitudes towards items. It identifies patterns and similarities among users and items to recommend new items. This approach can be subdivided into:
- 3. <u>User-Based Collaborative Filtering:</u> This method finds users whose preferences are similar to those of the target user and recommends items they have liked.
- 4. <u>Item-Based Collaborative Filtering:</u> Instead of finding similar users, this method focuses on finding similar items based on users' interactions.
- 5. <u>Hybrid Approaches:</u> Hybrid systems combine the strengths of both content-based and collaborative filtering techniques to provide more accurate and robust recommendations. These systems can offer suggestions based on a broader range of attributes and interactions, potentially overcoming the limitations inherent in using either approach in isolation.
- 6. <u>Knowledge-Based Systems:</u> These systems recommend items based on explicit knowledge about the item features that meet user needs and preferences. Unlike other systems that rely on past interactions,

knowledge-based recommendations are particularly useful for scenarios where historical data is limited or for items that are purchased infrequently.

- 7. <u>Demographic-Based Systems:</u> These systems make recommendations based on the demographic characteristics of the user. By analyzing traits such as age, gender, or location, the system predicts and recommends items that are popular or relevant within a particular demographic group.
- 8. <u>Utility-Based Systems:</u> Utility-based recommendation systems calculate the usefulness of an item to a particular user. The utility could be determined based on a variety of factors, including the item's cost, user's budget, or any other attribute that contributes to the item's overall value to the user.

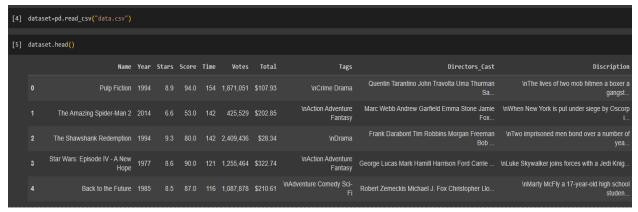
Each type of recommendation system offers distinct advantages and challenges, and the choice among them depends on specific application needs, the nature of the items being recommended, and the available data on user preferences and behaviors.

IMPLEMENTATION:

Dataset: The dataset contains various columns Import the necessary libraries

```
[30] import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
```

Import the dataset



Dropping the duplicate and null values from the dataset. Further reducing the dataset by drop few of its values for better performance and processing

```
[6] dataset.shape
      (9937, 10)
[7] dataset.info()
      <class 'pandas.core.frame.DataFrame'>
      RangeIndex: 9937 entries, 0 to 9936
     Data columns (total 10 columns):
     Non-Null Count Dtype

O Name 9937 non-null object

Year 9754 non-null object

Stars 9725 non-null float64

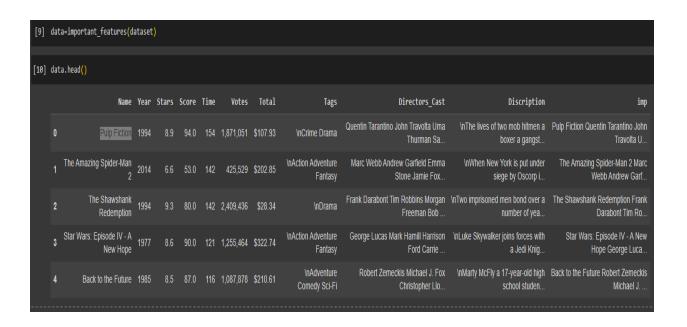
Time 9722 non-null object

Votes 9736 non-null object

Total 6892 non-null object

Tags 9929 non-null object

Directors Cast
      # Column Non-Null Count Dtype
      8 Directors_Cast 9853 non-null object
      9 Discription 6145 non-null object
     dtypes: float64(2), object(8)
     memory usage: 776.5+ KB
Only stars and score are in float rest are objects
[8] def important_features(dataset):
           data=dataset.copy()
           for i in range(0, dataset.shape[0]):
                data["imp"]=data["Name"]+' '+data["Directors_Cast"]+" "+data["Tags"]
           return data
```



Converting the tags to vectors using Vectorizer function.

```
print(vecs)
⊡
      (0, 5576)
                    0.0893374688444905
      (0, 13596)
                    0.15456324122571213
      (0, 20398)
                    0.2603320943686466
      (0, 2859)
                    0.20421580508719103
      (0, 9516)
                    0.2242257711506278
      (0, 16573)
                    0.24382936506900466
      (0, 18960)
                    0.2890712801192855
      (0, 19540)
                    0.2905118320623573
      (0, 19271)
                    0.2718421705221565
      (0, 9815)
                    0.1309293177793133
      (0, 18675)
                    0.313206178628782
      (0, 15372)
                    0.29681852648181156
      (0, 6621)
                    0.37468896774346483
      (0, 15298)
                   0.4041423251396885
      (1, 6441)
                    0.15470268436712495
      (1, 297)
                    0.1362983319226333
      (1, 13482)
                    0.10954334275032904
      (1, 7493)
                    0.2903195184358209
      (1, 14576)
                    0.1681070369020907
      (1, 6932)
                    0.2936650235131298
      (1, 9570)
                    0.232781099439005
      (1, 18198)
                    0.232781099439005
      (1, 6079)
                    0.24386892258246826
      (1, 7288)
                    0.2973017945815164
      (1, 726)
                    0.21159300846464732
      (9934, 9625) 0.2054107150011353
       (9934, 14326) 0.2673546007574271
       (9934, 3925) 0.2871181102647637
       (9934, 13195) 0.2291235994058673
       (9934, 9929) 0.2812972379940318
       (9934, 18221) 0.2499558573234468
      (9934, 13462) 0.15971716201880928
      (9934, 13598) 0.11813005123084618
      (9935, 16730) 0.34548843760396636
      (9935, 20339) 0.34548843760396636
      (9935, 18009) 0.3307598190313826
      (9935, 15914) 0.305581079275542
      (9935, 16329) 0.2870252143710226
      (9935, 3575) 0.29513095809228507
      (9935, 16230) 0.2219391162867575
      (9935, 3517) 0.2600741600466159
      (9935, 12612) 0.2748027786191996
       (9935, 9168) 0.19498806196235083
       (9935, 15724) 0.23238923747001441
       (9935, 10198) 0.2404949811912769
       (9935, 4445) 0.11634522896437435
       (9935, 18940) 0.10772702079288303
       (9935, 13482) 0.09520386844998373
       (9935, 9815) 0.11192731526076415
```

Calculating the similarity score of the vectors using cosine similarity.

Function for making the recommendations

```
[20] def recommend(title):
         movie_id=data[data.Name==title]["ids"].values[0]
         scores=list(enumerate(sim[movie_id]))
         sorted_scores=sorted(scores,key=lambda x:x[1],reverse=True)
         sorted_scores=sorted_scores[1:]
         movies=[data[movies[0]==data["ids"]]["Name"].values[0] for movies in sorted_scores]
         return movies
    def recommend_ten(movie_list):
         first_ten=[]
         count=0
         for movie in movie_list:
            if count > 9:
                 break
             count+=1
             first_ten.append(movie)
         return first_ten
```

Actual Recommendations made by the model

```
[28] 1st=recommend("Pulp Fiction")
    m=recommend_ten(1st)

[29] m

['Kill Bill: Vol. 1',
    'Kill Bill: Vol. 2',
    'Jackie Brown',
    'The Hateful Eight',
    'Sin City',
    'Be Cool',
    'Jennifer 8',
    'Basic',
    'Die Hard with a Vengeance',
    'Reservoir Dogs']
```

Conclusion:

Thus, a recommendation system using Machine Learning was implemented successfully.