Name: Nayaab Jindani D15C Rollno.20

Experiment 8

Aim: To implement a recommendation system on your dataset using the following machine learning technique.

Theory:

Dataset Description

- The dataset used is the **MovieLens 100K dataset**.
- It contains user ratings for different movies, in the form of:
  - userID
  - itemID (movie)
  - rating
  - timestamp (ignored in this experiment)

Steps:

1. Importing libraries

```
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
```

2. Load the dataset taken from MovieLens and we will then preprocess this data by dropping the columns we don't need

```
df = pd.read_csv("ml-100k/u.data", sep="\t", names=["user_id", "item_id", "rating", "timestamp"])
df.drop(columns="timestamp", inplace=True)
```

3. This line of code is creating a **user-item matrix**, which is a common structure used in **collaborative filtering** for recommendation systems.

```
[ ]  user_item_matrix = df.pivot(index='user_id', columns='item_id', values='rating').fillna(0)
```

4. Now that the data is in the required format, We need to apply k-mean clustering to cluster similar Clustering helps **group similar users** based on their preferences (e.g., movie ratings). Each cluster represents a **type of user behavior** (e.g., "Action Lovers", "Rom-Com Fans", etc.).

   You can then recommend **popular or high-rated items within that user's cluster**, even if the user is new.

```
inertia_vals = []
k_values = range(1, 11)

for k in k_values:
    model = KMeans(n_clusters=k, random_state=42)
    model.fit(user_item_matrix)
    inertia_vals.append(model.inertia_)

plt.figure(figsize=(8,5))
plt.plot(k_values, inertia_vals, marker='o')
plt.title('Elbow Method For Optimal k')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.grid(True)
plt.show()
```

The elbow method helps find an accurate value for k. In our case, the optimal value for k is 5, we cluster the user_item matrix

```
kmeans = KMeans(n_clusters=5, random_state=42)
# Convert all column names to strings
user_item_matrix.columns = user_item_matrix.columns.astype(str)
clusters = kmeans.fit_predict(user_item_matrix)
user_item_matrix['cluster'] = clusters
```

5. Now we are going to split the dataset into train and test

```
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
```

6. This function predict_rating is designed to **predict a user's rating** for a movie (item) using **cluster-based collaborative filtering**.

To predict the **expected rating** that a specific user (user_id) would give to a specific movie (item_id) using:

- **User clustering**
- **Cosine similarity**
- **Weighted average of ratings from similar users in the same cluster**

```
def predict_rating(user_id, item_id, matrix):
    if item_id not in matrix.columns:
        return 3.0  # neutral rating if item not found

    cluster = matrix.loc[user_id, 'cluster']
    cluster_users = matrix[matrix['cluster'] == cluster].drop(columns='cluster')

    user_ratings = cluster_users.loc[:, item_id]
    if user_ratings.sum() == 0:
        return 3.0  # neutral if no one in cluster rated it

    similarities = cosine_similarity([cluster_users.loc[user_id]], cluster_users)[0]
    weighted_sum = 0
    sim_sum = 0

    for i, other_user in enumerate(cluster_users.index):
        if other_user == user_id:
            continue
        rating = cluster_users.loc[other_user, item_id]
        sim = similarities[i]
        weighted_sum += rating * sim
        sim_sum += sim

    if sim_sum == 0:
        return 3.0  # neutral if no similarity
    return weighted_sum / sim_sum
```

This function:

- Uses clustering to narrow down the pool of similar users
- Uses cosine similarity to compute how similar users are
- Predicts the rating based on a weighted average from similar users
- Uses 3.0 as a neutral fallback in edge cases (e.g., no data)

This function recommends movies to a specific user by leveraging both clustering and collaborative filtering. It begins by identifying the cluster that the target user belongs to—this cluster groups users with similar movie preferences based on their past ratings. Within this cluster, it filters out the movies that the user hasn't rated yet, assuming these are the ones they haven't watched. For each of these unrated movies, the function predicts a rating by analyzing how similar users in the same cluster have rated them, using cosine similarity to weigh each user's input based on how closely they resemble the target user.

```
def recommend_movies_for_user(user_id, matrix, top_n=5):
    cluster_label = matrix.loc[user_id, 'cluster']
    cluster_users = matrix[matrix['cluster'] == cluster_label].drop(columns='cluster')

    unrated_items = cluster_users.columns[cluster_users.loc[user_id] == 0]

    recommendations = []
    for item_id in unrated_items:
        pred = predict_rating(user_id, item_id, matrix)
        recommendations.append((item_id, pred))

    top_recommendations = sorted(recommendations, key=lambda x: x[1], reverse=True)[:top_n]
    return [(id_to_title.get(int(item_id), f"Movie {int(item_id)}"), round(pred, 2)) for item_id, pred in top_recommendations]
```

The predicted ratings are then sorted, and the top ones are selected as recommendations. Finally, these movie IDs are converted into human-readable movie titles using a lookup dictionary, and the function returns a neatly formatted list of the top recommendations with predicted scores. This approach ensures that recommendations are not just based on general popularity but are tailored to the tastes of users who think and rate similarly.

```
recommendations = recommend_movies_for_user(100, user_item_matrix)
for title, score in recommendations:
    print(f"{title} → Predicted Rating: {score}")
```

```
Doom Generation, The (1995) → Predicted Rating: 3.0
Nadja (1994) → Predicted Rating: 3.0
Brother Minister: The Assassination of Malcolm X (1994) → Predicted Rating: 3.0
Carlito's Way (1993) → Predicted Rating: 3.0
Robert A. Heinlein's The Puppet Masters (1994) → Predicted Rating: 3.0
```

The movies are then recommended for random users from user_item matrix