# MAJOR PROJECT

## Online Recommendation System

Developing an Online recommendation system involves building a system that suggests products, services, or content to users based on their preferences, behavior, and other factors.

**Understanding Recommendation Systems**

A recommendation system filters and suggests relevant items to users based on different approaches:

**Types of Recommendation Systems**

1. **Content-Based Filtering**
    - Recommends items similar to those a user has previously interacted with.
    - Uses item features (e.g., movie genre, product category).
2. **Collaborative Filtering**
    - Recommends items based on user behavior and preferences.
    - Two types:
        - **User-based Collaborative Filtering:** Suggests items liked by similar users.
        - **Item-based Collaborative Filtering:** Suggests items similar to those the user liked.
    - Example: Amazon suggesting products based on other customers' purchases.
3. **Hybrid Recommendation System**
    - Combines content-based and collaborative filtering for better accuracy.
    - Example: Spotify recommending songs based on your listening history and what similar users enjoy.
4. **Knowledge-Based Recommendations**
    - Uses explicit user inputs (e.g., surveys, questionnaires) to recommend items.

## SOURCE CODE:

```
import pandas as pd
data = {
    'user_id': [1, 2, 3, 4, 5, 6, 8, 7, 9, 11],
    'item_id': [101, 102, 103, 104, 105, 106, 107, 108, 109, 110],
    'rating': [5, 4, 5, 4, 5, 2, 3, 5, 4, 3]
}
df = pd.DataFrame(data)
print("User -Item Interaction Data:")
print(df)
user_item_matrix = df.pivot(index='user_id', columns='item_id',
values='rating').fillna(0)
print("\nUser -Item Matrix:")
print(user_item_matrix)
from sklearn.metrics.pairwise import cosine_similarity
user_similarity = cosine_similarity(user_item_matrix)
user_similarity_df = pd.DataFrame(user_similarity,
index=user_item_matrix.index, columns=user_item_matrix.index)
print("\nUser  Similarity Matrix:")
print(user_similarity_df)
def get_recommendations(user_id, user_item_matrix, user_similarity_df,
num_recommendations=3):
    user_ratings = user_item_matrix.loc[user_id]
```

```
    similar_users = user_similarity_df[user_id].sort_values(ascending=False)
    weighted_scores = {}
    for similar_user, similarity in similar_users.items():
        if similar_user == user_id:
            continue
        for item_id, rating in user_item_matrix.loc[similar_user].items():
            if rating > 0:
                if item_id not in weighted_scores:
                    weighted_scores[item_id] = 0
                weighted_scores[item_id] += similarity * rating
    recommended_items = sorted(weighted_scores.items(), key=lambda x: x[1],
reverse=True)
    return [item[0] for item in recommended_items[:num_recommendations]]
user_id = 1
recommendations = get_recommendations(user_id, user_item_matrix,
user_similarity_df)
print(f"\nRecommendations for User {user_id}: {recommendations}")
```

## Explanation  of the code:

**1. Importing Necessary Libraries:**

- pandas is used for data manipulation and analysis.
- cosine_similarity from sklearn.metrics.pairwise is used to compute the similarity between users based on their ratings.

**2. Creating the Dataset:**

- A dictionary data is defined with three keys: user_id, item_id, and rating, representing user IDs, item IDs, and the corresponding ratings given by users.
- This dictionary is converted into a pandas DataFrame df for easier manipulation and visualization.

**3. Creating the User-Item Matrix:**

- The pivot function reshapes the DataFrame df to create a matrix where:
    - Rows represent user_id.
    - Columns represent item_id.
    - The values are the corresponding rating.
- The fillna(0) method replaces any missing values (NaN) with 0, indicating that a user has not rated that particular item.

**4. Calculating User Similarity Matrix:**

- Cosine_similarity computes the cosine similarity between users based on their ratings in the user_item_matrix.
- The result is stored in user_similarity, which is then converted into a DataFrame user_similarity_df with both rows and columns labeled by

user_id. This matrix indicates how similar each user is to every other user.

**5. Defining the Recommendation Function:**

- **Parameters:**
  - user_id: The ID of the user for whom recommendations are to be generated.
  - user_item_matrix: The matrix containing user-item interactions.
  - user_similarity_df: The DataFrame containing user-user similarity scores.
  - num_recommendations: The number of recommendations to return (default is 3).
- Process:
  - Retrieve the ratings of the target user using user_item_matrix.loc[user_id].
  - Obtain a list of users sorted by their similarity to the target user in descending order.
  - Initialize an empty dictionary weighted_scores to store the weighted sum of ratings for each item.
  - Iterate over each similar user and their similarity score:
    - Skip the iteration if the similar user is the target user.
    - For each item rated by the similar user:
- If the rating is greater than 0 (i.e., the user has rated the item):
- If the item is not already in weighted_scores, initialize its score to 0.
    - Add the product of the similarity score and the rating to the item's score in weighted_scores.
  - Sort the items in weighted_scores by their scores in descending order.
  - Return the top num_recommendations item IDs.

**6. Generating Recommendations for a Specific User:**

**Summary:**

This code implements a user-based collaborative filtering recommendation system. It calculates the similarity between users based on their item ratings using cosine similarity. For a target user, it identifies similar users and recommends items that these similar users have rated highly but the target user has not yet interacted with. The recommendations are based on a weighted sum of ratings from similar users, where the weights are the similarity scores.