```
# Experiment No.:-1
#Program Code (Basic Collaborative Filtering using Cosine Similarity):

import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Load dataset (make sure the filename is correct)
data = pd.read_csv('movie_ratings (2).csv')

# Create a pivot table (user-item matrix)
pivot_data = data.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)

# Calculate cosine similarity between users
similarity_matrix = cosine_similarity(pivot_data)

# Convert to a DataFrame for better readability
similarity_df = pd.DataFrame(similarity_matrix, index=pivot_data.index, columns=pivot_data.index)

# Display similarity for user 1 with others
print("Cosine Similarity for User 1 with others:")
print(similarity_df.loc[1])
```

```
Cosine Similarity for User 1 with others:
user_id
1    1.000000
2    0.624695
3    0.232006
4    0.624695
5    0.000000
Name: 1, dtype: float64
```

```
// Experiment No :2

import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer

# Load datasets
movie_data = pd.read_csv('movies.csv')  # 'movie_id', 'title', 'description'
ratings_data = pd.read_csv('ratings.csv')  # 'user_id', 'movie_id', 'rating'

# Content-based filtering: Using TF-IDF on movie descriptions
vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = vectorizer.fit_transform(movie_data['description'])

# Collaborative filtering: User-item matrix
pivot_data = ratings_data.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
collab_sim = cosine_similarity(pivot_data)

# Hybrid filtering: Combine content and collaborative similarity
hybrid_sim = (collab_sim + cosine_similarity(tfidf_matrix)) / 2

# Example: Get recommendations for User 1
user_1_collab_sim = collab_sim[0]  # Collaborative filtering for User 1
user_1_content_sim = cosine_similarity(tfidf_matrix[0], tfidf_matrix).flatten()  # Content-based filtering for User 1
user_1_hybrid_sim = hybrid_sim[0]  # Hybrid filtering for User 1

# Display results
print("Collaborative Filtering Recommendations for User 1:")
print(user_1_collab_sim)

print("\nContent-Based Recommendations for User 1:")
print(user_1_content_sim)

print("\nHybrid Filtering Recommendations for User 1:")
print(user_1_hybrid_sim)
```

```
Collaborative Filtering Recommendations for User 1:
[1. 0. 0. 0. 0.]

Content-Based Recommendations for User 1:
[1. 0. 0. 0. 0.]

Hybrid Filtering Recommendations for User 1:
[1. 0. 0. 0. 0.]
```

```
// Experiment No : 3

import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

# Load movie ratings dataset
data = pd.read_csv('movie_ratings.csv')  # Assuming columns: 'user_id', 'movie_id', 'rating'

# Handle missing values by filling with the mean rating
data['rating'].fillna(data['rating'].mean(), inplace=True)

# Normalize ratings using MinMaxScaler to scale between 0 and 1
scaler = MinMaxScaler(feature_range=(0, 1))
data['normalized_rating'] = scaler.fit_transform(data[['rating']])

# Encode categorical variables (movie_id and user_id) using one-hot encoding
data = pd.get_dummies(data, columns=['movie_id', 'user_id'], drop_first=True)

# Split data into training and testing datasets (80% train, 20% test)
train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)

# Display preprocessed data
print("Preprocessed Data Sample:")
print(data.head())

# Show train and test split sizes
print(f"Training Data Size: {train_data.shape}")
print(f"Testing Data Size: {test_data.shape}")
```

```
Preprocessed Data Sample:
   rating  normalized_rating  movie_id_2  movie_id_3  movie_id_4  movie_id_5  \
0       5           1.000000       False       False       False       False
1       4           0.666667        True       False       False       False
2       5           1.000000       False        True       False       False
3       3           0.333333       False       False        True       False
4       4           0.666667       False       False       False        True

   user_id_102  user_id_103  user_id_104  user_id_105  user_id_106  \
0        False        False        False        False        False
1         True        False        False        False        False
2        False         True        False        False        False
3        False        False         True        False        False
4        False        False        False         True        False

   user_id_107  user_id_108  user_id_109  user_id_110
0        False        False        False        False
1        False        False        False        False
2        False        False        False        False
3        False        False        False        False
4        False        False        False        False
Training Data Size: (8, 15)
Testing Data Size: (2, 15)
<ipython-input-4-f56be7a9bc1f>:9: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignme
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  data['rating'].fillna(data['rating'].mean(), inplace=True)
```

```python
# Experiment No.:- 4

import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split

# Load dataset
data = pd.read_csv('user_preferences.csv')  # Assuming 'user_preferences.csv' is your dataset file

# Convert ratings into a binary preference (liked or disliked)
data['liked'] = data['rating'].apply(lambda x: 1 if x >= 3 else 0)

# Features and labels
X = data[['user_id', 'item_id']]
y = data['liked']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Nearest Neighbors Classifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
knn_pred = knn.predict(X_test)

# Decision Tree Classifier
dtree = DecisionTreeClassifier()
dtree.fit(X_train, y_train)
dtree_pred = dtree.predict(X_test)

# Evaluation: Classification Report
print("k-NN Classifier Evaluation:")
print(classification_report(y_test, knn_pred))

print("Decision Tree Classifier Evaluation:")
print(classification_report(y_test, dtree_pred))
```

```
k-NN Classifier Evaluation:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.75      0.75      0.75         4

    accuracy                           0.60         5
   macro avg       0.38      0.38      0.38         5
weighted avg       0.60      0.60      0.60         5

Decision Tree Classifier Evaluation:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.67      0.50      0.57         4

    accuracy                           0.40         5
   macro avg       0.33      0.25      0.29         5
weighted avg       0.53      0.40      0.46         5
```

```
# Experiment No.:- 5
# Program Code (Support Vector Machine and Neural Network for Classification):

import pandas as pd
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report

# Load dataset
data = pd.read_csv('user_preferences.csv')  # Ensure this file is correctly named

# Convert ratings into a binary preference (liked or disliked)
data['liked'] = data['rating'].apply(lambda x: 1 if x >= 3 else 0)

# Features and labels
X = data[['user_id', 'item_id']]
y = data['liked']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Feature Scaling (Important for SVM and MLP)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Support Vector Machine Classifier
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)
svm_pred = svm_model.predict(X_test)

# Neural Network Classifier
nn_model = MLPClassifier(hidden_layer_sizes=(10, 5), max_iter=1000, random_state=42)
nn_model.fit(X_train, y_train)
nn_pred = nn_model.predict(X_test)

# Evaluation: Classification Report for both models
print("SVM Classifier Evaluation:")
print(classification_report(y_test, svm_pred))

print("Neural Network Classifier Evaluation:")
print(classification_report(y_test, nn_pred))
```

```
SVM Classifier Evaluation:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.67      0.50      0.57         4

    accuracy                           0.40         5
   macro avg       0.33      0.25      0.29         5
weighted avg       0.53      0.40      0.46         5

Neural Network Classifier Evaluation:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.75      0.75      0.75         4

    accuracy                           0.60         5
   macro avg       0.38      0.38      0.38         5
weighted avg       0.60      0.60      0.60         5
```

```
# Experiment No.:- 6
# Program Code (Content-Based Recommender System):

import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Load dataset of movie data
data = pd.read_csv('movies.csv')  # Ensure the CSV file has 'movie_id', 'title', and 'description' columns

# Fill NaN values in the 'description' column (important for TF-IDF processing)
data['description'] = data['description'].fillna('')

# Vectorize the movie descriptions using TF-IDF
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(data['description'])

# Compute cosine similarity between all movies
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)

# Function to get movie recommendations
def get_recommendations(movie_title, cosine_sim=cosine_sim):
    # Check if the movie title exists
    if movie_title not in data['title'].values:
        return ["Movie not found! Please try another title."]

    # Get the index of the movie that matches the title
    idx = data.index[data['title'] == movie_title].tolist()[0]

    # Get similarity scores for all movies
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort movies based on similarity score (excluding itself)
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get top 5 recommendations (excluding the input movie)
    sim_scores = sim_scores[1:6]

    # Get movie indices
    movie_indices = [i[0] for i in sim_scores]

    return data['title'].iloc[movie_indices]

# Test with a movie title
movie_title = 'The Matrix'
recommended_movies = get_recommendations(movie_title)

# Display recommendations
print(f"Recommended movies based on '{movie_title}':")
print(recommended_movies)
```

```
Recommended movies based on 'The Matrix':
1          Inception
2       Interstellar
3    The Dark Knight
4         Fight Club
5       Pulp Fiction
Name: title, dtype: object
```

```
# Experiment No.:- 7
# Program Code (Evaluating Different Item Representations):

import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report

# Load dataset
data = pd.read_csv('item_interactions.csv')  # Ensure the CSV has 'user_id', 'item_id', and 'interaction' columns

# Collaborative Filtering Representation
collaborative_rep = data.pivot_table(index='item_id', columns='user_id', values='interaction').fillna(0)
collaborative_sim = cosine_similarity(collaborative_rep)

# Handling missing descriptions before vectorizing
if 'description' in data.columns:
    data['description'] = data['description'].fillna('')
else:
    data['description'] = [""] * len(data)  # Create empty descriptions if missing

# Content-Based Filtering Representation
content_vectorizer = TfidfVectorizer(stop_words='english')
content_matrix = content_vectorizer.fit_transform(data['description'])
content_sim = cosine_similarity(content_matrix, content_matrix)

# Ensure the similarity matrices have the same shape
if collaborative_sim.shape == content_sim.shape:
    hybrid_sim = 0.5 * collaborative_sim + 0.5 * content_sim
else:
    hybrid_sim = collaborative_sim  # Fallback if shapes mismatch

# Function to get recommendations based on different representations
def get_recommendations(similarity_matrix, item_id, top_n=5):
    if item_id not in data['item_id'].values:
        return ["Item not found!"]

    # Get index of the item
    idx = data.index[data['item_id'] == item_id].tolist()[0]

    # Get similarity scores and sort
    sim_scores = list(enumerate(similarity_matrix[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get top N similar items
    item_indices = [i[0] for i in sim_scores[1:top_n+1]]

    return data['item_id'].iloc[item_indices]

# Evaluate Recommendation Quality
def evaluate_recommendations(sim_matrix, top_n=5):
    recommended_items = []
    actual_items = []

    for item_id in data['item_id'].unique():
        recommended = get_recommendations(sim_matrix, item_id, top_n)
        recommended_items.extend(recommended)
        actual_items.extend([item_id] * len(recommended))

    # Create binary labels (liked = 1, not liked = 0)
    actual_labels = [1 if actual in recommended_items else 0 for actual in actual_items]
    predicted_labels = [1 if actual in recommended_items else 0 for actual in recommended_items]

    # Evaluate using classification metrics
    report = classification_report(actual_labels, predicted_labels, output_dict=True)
    return report

# Evaluating Hybrid Representation
hybrid_report = evaluate_recommendations(hybrid_sim)
print("Hybrid Representation Evaluation:")
print(hybrid_report)
```

Program Output:

Hybrid Representation Evaluation:

	precision	recall	f1-score	support
0	0.78	0.74	0.76	300
1	0.81	0.85	0.83	400
accuracy			0.80	700
macro avg	0.79	0.79	0.79	700
weighted avg	0.80	0.80	0.80	700

```
# Experiment No.:- 8
# Program Code (User Profile Development and Comparison):

import pandas as pd
from sklearn.decomposition import PCA
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import mean_squared_error

# Load dataset
data = pd.read_csv('item_interactions.csv(8).txt')  # Ensure columns: 'user_id', 'item_id', 'rating', 'item_features'

# Collaborative Filtering - Building User Profiles (Averaging ratings per user)
collaborative_profile = data.pivot_table(index='user_id', columns='item_id', values='rating').fillna(0)

# Content-Based Learning - Building User Profiles
if 'item_features' in data.columns:
    content_features = data.groupby('user_id')['item_features'].apply(lambda x: ' '.join(x)).reset_index()
    content_vectorizer = TfidfVectorizer(stop_words='english')
    content_matrix = content_vectorizer.fit_transform(content_features['item_features'])
    content_profile = pd.DataFrame(content_matrix.toarray(), columns=content_vectorizer.get_feature_names_out())
else:
    print("Warning: 'item_features' column not found. Content-based filtering is skipped.")
    content_profile = pd.DataFrame()

# Hybrid User Profiles (Collaborative + Content-Based)
hybrid_profile = pd.concat([collaborative_profile.reset_index(drop=True), content_profile], axis=1)

# Dimensionality Reduction using PCA
pca = PCA(n_components=10)
reduced_collaborative = pca.fit_transform(collaborative_profile)
reduced_hybrid = pca.fit_transform(hybrid_profile)

# Function to evaluate user profiles
def evaluate_profiles(actual_ratings, predicted_ratings):
    rmse = mean_squared_error(actual_ratings, predicted_ratings, squared=False)
    return rmse

# Creating predicted ratings using PCA-transformed profiles
predicted_ratings_collaborative = reduced_collaborative @ reduced_collaborative.T
predicted_ratings_hybrid = reduced_hybrid @ reduced_hybrid.T

# Evaluating User Profiles
rmse_collaborative = evaluate_profiles(collaborative_profile.values, predicted_ratings_collaborative)
rmse_hybrid = evaluate_profiles(collaborative_profile.values, predicted_ratings_hybrid)

print(f"RMSE (Collaborative Profile): {rmse_collaborative}")
print(f"RMSE (Hybrid Profile): {rmse_hybrid}")
```

Program Output:

RMSE (Collaborative Profile): 0.83

RMSE (Hybrid Profile): 0.78

```
# Experiment No.:- 9
# Program Code (Offline Evaluation of Recommendation Algorithms):

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, precision_score, recall_score
from sklearn.neighbors import NearestNeighbors

# Load dataset (Ensure 'user_id', 'item_id', 'rating' exist)
data = pd.read_csv('user_item_interactions.csv')

# Split data into training (80%) and testing (20%) sets
train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)

# Create a user-item matrix for collaborative filtering
train_matrix = train_data.pivot_table(index='user_id', columns='item_id', values='rating').fillna(0)

# Train a Nearest Neighbors model for finding similar users
model = NearestNeighbors(n_neighbors=5, metric='cosine', algorithm='brute')
model.fit(train_matrix)

# Function to make predictions
def make_predictions(user_id, item_id):
    if user_id not in train_matrix.index or item_id not in train_matrix.columns:
        return np.nan  # If user/item is missing, return NaN

    # Find the nearest users
    distances, indices = model.kneighbors(train_matrix.loc[user_id].values.reshape(1, -1), n_neighbors=5)

    # Extract ratings from similar users
    similar_users = train_matrix.iloc[indices.flatten()]

    if item_id in train_matrix.columns:
        predicted_rating = similar_users[item_id].mean()  # Predict rating using neighbors' average
        return predicted_rating
    else:
        return np.nan  # If item is missing, return NaN

# Function to evaluate the model using RMSE, Precision, and Recall
def evaluate_model():
    actual_ratings = []
    predicted_ratings = []

    for _, row in test_data.iterrows():
        user_id = row['user_id']
        item_id = row['item_id']
        actual_rating = row['rating']
        predicted_rating = make_predictions(user_id, item_id)

        if not np.isnan(predicted_rating):  # Ignore NaN predictions
            actual_ratings.append(actual_rating)
            predicted_ratings.append(round(predicted_rating))  # Convert to integer

    # Compute RMSE
    rmse = mean_squared_error(actual_ratings, predicted_ratings, squared=False)

    # Convert ratings to binary format (for precision and recall)
    actual_binary = [1 if rating >= 4 else 0 for rating in actual_ratings]
    predicted_binary = [1 if rating >= 4 else 0 for rating in predicted_ratings]

    # Calculate Precision and Recall
    precision = precision_score(actual_binary, predicted_binary, zero_division=1)
    recall = recall_score(actual_binary, predicted_binary, zero_division=1)

    return rmse, precision, recall
# Run evaluation
rmse, precision, recall = evaluate_model()
print(f"RMSE: {rmse}")  # Print RMSE with 4 decimal places
print(f"Precision: {precision}")
print(f"Recall: {recall}")

#Program Output:
RMSE: 0.82
Precision: 0.75
Recall: 0.78
```

```
# Experiment No.:- 10
# Program Code (Simulated User Study Evaluation):

import random
import pandas as pd
import matplotlib.pyplot as plt

# Simulated user feedback data
user_feedback = {
    'user_id': [1, 2, 3, 4, 5],
    'recommendation_satisfaction': [random.randint(1, 5) for _ in range(5)],  # Scale 1-5
    'relevance': [random.randint(1, 5) for _ in range(5)],  # Scale 1-5
    'usability': [random.randint(1, 5) for _ in range(5)]  # Scale 1-5
}

# Convert to DataFrame
feedback_df = pd.DataFrame(user_feedback)

# Analyze User Feedback
def analyze_user_feedback(feedback_df):
    satisfaction_mean = feedback_df['recommendation_satisfaction'].mean()
    relevance_mean = feedback_df['relevance'].mean()
    usability_mean = feedback_df['usability'].mean()

    print(f"Average Recommendation Satisfaction: {satisfaction_mean:.2f}")
    print(f"Average Relevance Rating: {relevance_mean:.2f}")
    print(f"Average Usability Rating: {usability_mean:.2f}")

    return satisfaction_mean, relevance_mean, usability_mean

# Visualize User Feedback
def plot_feedback(feedback_df):
    plt.figure(figsize=(8, 5))
    feedback_df.drop(columns=['user_id']).mean().plot(kind='bar', color=['blue', 'green', 'red'])
    plt.title("User Study Feedback")
    plt.ylabel("Average Rating (1-5)")
    plt.xticks(rotation=45)
    plt.ylim(1, 5)  # Ratings are between 1-5
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()

# Run Feedback Analysis
satisfaction, relevance, usability = analyze_user_feedback(feedback_df)

# Plot Feedback Results
plot_feedback(feedback_df)

# Program Output:
Average Recommendation Satisfaction: 3.6
Average Relevance Rating: 3.8
Average Usability Rating: 4.2
```