In [4]:
```
# Name    : Lamak Shahiwala
# Enroll  : 202202626010046
# Roll no : A22CSE046
# Task    : 1
# Sub     : RS-LAB
```

In [1]:
```
import numpy as np
import pandas as pd
from scipy.stats import pearsonr

# Utility matrix
data = {
    'Northanger Abby': [5, 1, 1, None],
    'Wuthering Heights': [4, 2, 2, 4],
    'Oroonoko': [3, 4, 3, 3],
    "Bondswoman’s Narrative": [4, 5, None, 1]
}
users = ['Alex', 'Loren', 'Taylor', 'Ainsley']
df = pd.DataFrame(data, index=users)

# Helper functions
def user_mean_centered_ratings(user):
    return df.loc[user] - df.loc[user].mean(skipna=True)

def pearson_sim(user1, user2):
    common_ratings = df.loc[[user1, user2]].dropna(axis=1)
    if len(common_ratings.columns) < 2:
        return 0
    return pearsonr(common_ratings.loc[user1], common_ratings.loc[user2])[0]

# (a) User-based collaborative filtering with Pearson and mean-centering
def predict_user_based(target_user, target_item):
    # Compute similarities with other users
    similarities = {}
    for user in users:
        if user != target_user and not pd.isna(df.loc[user, target_item]):
            sim = pearson_sim(target_user, user)
            similarities[user] = sim

    # Select users with positive similarity
    similarities = {k:v for k,v in similarities.items() if v > 0}
    if not similarities:
        return df[target_item].mean()  # Fallback to global mean

    # Calculate weighted sum of mean-centered ratings
    numerator = 0
    denominator = 0
    target_mean = df.loc[target_user].mean(skipna=True)
    for user, sim in similarities.items():
        user_mean = df.loc[user].mean(skipna=True)
        rating = df.loc[user, target_item]
        numerator += sim * (rating - user_mean)
        denominator += abs(sim)

    if denominator == 0:
        return target_mean
    return target_mean + (numerator / denominator)
```

```
# (b) Item-based collaborative filtering with adjusted cosine similarity
def adjusted_cosine(item1, item2):
    # Subtract user mean for each rating
    common_users = df[[item1, item2]].dropna().index
    if len(common_users) == 0:
        return 0
    adjusted_ratings = []
    for user in common_users:
        user_mean = df.loc[user].mean(skipna=True)
        adj1 = df.loc[user, item1] - user_mean
        adj2 = df.loc[user, item2] - user_mean
        adjusted_ratings.append((adj1, adj2))
    a = np.array([x[0] for x in adjusted_ratings])
    b = np.array([x[1] for x in adjusted_ratings])
    return np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b) + 1e-10)

def predict_item_based(target_user, target_item):
    # Find similar items
    similarities = {}
    for item in df.columns:
        if item != target_item and not pd.isna(df.loc[target_user, item]):
            sim = adjusted_cosine(target_item, item)
            similarities[item] = sim

    # Calculate weighted average
    numerator = 0
    denominator = 0
    for item, sim in similarities.items():
        rating = df.loc[target_user, item]
        numerator += sim * rating
        denominator += abs(sim)

    if denominator == 0:
        return df[target_item].mean()
    return numerator / denominator

# Predict missing values
print("Question 1a: User-based predictions")
print("Taylor's Bondswoman’s Narrative:", predict_user_based('Taylor', "Bondswoman’s Narrative"))
print("Ainsley's Northanger Abby:", predict_user_based('Ainsley', 'Northanger Abby'))

print("\nQuestion 1b: Item-based predictions")
print("Taylor's Bondswoman’s Narrative:", predict_item_based('Taylor', "Bondswoman’s Narrative"))
print("Ainsley's Northanger Abby:", predict_item_based('Ainsley', 'Northanger Abby'))
```

```
Question 1a: User-based predictions
Taylor's Bondswoman’s Narrative: 4.0
Ainsley's Northanger Abby: 2.3333333333333335

Question 1b: Item-based predictions
Taylor's Bondswoman’s Narrative: -0.753552340806314
Ainsley's Northanger Abby: -0.1721605090644537
```

In [2]:

```
#  Sample dictionary
dataset = {
    'Rahul': {'Special Ops': 5, 'Criminal Justice': 3, 'Panchayat': 3, 'Sacred Games': 3, 'Apharan': 2, 'Mirzapur': 3},
    'Rishabh': {'Special Ops': 5, 'Criminal Justice': 3, 'Sacred Games': 5, 'Panchayat': 5, 'Mirzapur': 3, 'Apharan': 3},
    'Sonali': {'Special Ops': 2, 'Panchayat': 5, 'Sacred Games': 3, 'Mirzapur': 4},
    'Ritvik': {'Panchayat': 5, 'Mirzapur': 4, 'Sacred Games': 4},
    'Harshita': {'Special Ops': 4, 'Criminal Justice': 4, 'Panchayat': 4, 'Mirzapur': 3, 'Apharan': 2},
```

```python
    'Shubhi': {'Special Ops': 3, 'Panchayat': 4, 'Mirzapur': 3, 'Sacred Games': 5, 'Apharan': 3},
    'Shaurya': {'Panchayat': 4, 'Apharan': 1, 'Sacred Games': 4}
}

#  Unique web series
def unique_series(data):
    series = set()
    for user in data.values():
        series.update(user.keys())
    return sorted(series)
print("\nQuestion 2b: Unique series:", unique_series(dataset))

#  Cosine similarity between two items
def cosine_sim(item1, item2, data):
    # Collect ratings for users who rated both items
    common_users = []
    for user, ratings in data.items():
        if item1 in ratings and item2 in ratings:
            common_users.append((ratings[item1], ratings[item2]))
    if not common_users:
        return 0
    a = np.array([x[0] for x in common_users])
    b = np.array([x[1] for x in common_users])
    return np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b) + 1e-10)

#  Similarity between target and others
def item_similarities(target_item, data):
    items = unique_series(data)
    similarities = {}
    for item in items:
        if item != target_item:
            similarities[item] = cosine_sim(target_item, item, data)
    return similarities

#  Seen and unseen series
def seen_unseen(user, data):
    seen = set(data[user].keys())
    all_series = unique_series(data)
    unseen = [s for s in all_series if s not in seen]
    return seen, unseen

# (e & g) Recommender function
def recommend(user, data, top_n=3):
    seen, unseen = seen_unseen(user, data)
    item_scores = {}
    for seen_item in seen:
        sims = item_similarities(seen_item, data)
        for unseen_item, sim in sims.items():
            if unseen_item in unseen:
                if unseen_item not in item_scores:
                    item_scores[unseen_item] = 0
                item_scores[unseen_item] += sim * data[user][seen_item]
    # Sort by score
    sorted_items = sorted(item_scores.items(), key=lambda x: x[1], reverse=True)
    return [item[0] for item in sorted_items[:top_n]]

# Example usage
print("\nQuestion 2e/g: Recommendations for Ritvik:", recommend('Ritvik', dataset))
```

```
Question 2b: Unique series: ['Apharan', 'Criminal Justice', 'Mirzapur', 'Panchayat', 'Sacred Games', 'Special Ops']

Question 2e/g: Recommendations for Ritvik: ['Criminal Justice', 'Apharan', 'Special Ops']
```

In [6]:

```
from sklearn.decomposition import TruncatedSVD, PCA
from sklearn.impute import SimpleImputer

data = np.array([[5, 3, 0, 1], [4, 0, 0, 1], [1, 1, 0, 5], [0, 3, 4, 0]])

import numpy as np
from numpy.linalg import svd

def svd_impute(data, k=2):
    data = np.array(data, dtype=np.float64)
    mask = data == 0  # Identify missing values
    mean_values = np.mean(data, axis=1, where=~mask, keepdims=True)
    # Tile mean_values to match the shape of mask
    mean_values = np.tile(mean_values, (1, data.shape[1]))
    data[mask] = mean_values[mask]  # Temporary fill with row mean

    U, S, Vt = svd(data, full_matrices=False)  # Compute SVD
    S = np.diag(S[:k])  # Reduce rank to k
    U = U[:, :k]
    Vt = Vt[:k, :]
    print('U',U)
    print('S',S)
    print('Vt',Vt)

    reconstructed = U @ S @ Vt  # Reconstruct the matrix

    data[mask] = reconstructed[mask]  # Replace only missing values
    return data

data = [
    [5, 3, 0, 1],
    [4, 0, 0, 1],
    [1, 1, 0, 5],
    [0, 3, 4, 0],
]

predicted_ratings = svd_impute(data, k=2)
print("\nmissing predicted")

print(np.round(predicted_ratings))
```

```
U [[-0.54160611 -0.46810733]
 [-0.44886273 -0.34079387]
 [-0.37893089  0.79651311]
 [-0.6013289   0.17407452]]
S [[11.595673    0.        ]
 [ 0.          4.53422491]]
Vt [[-0.60255783 -0.42514934 -0.52057891 -0.43031348]
 [-0.50679847 -0.20677625  0.06583749  0.8343047 ]]

missing predicted
[[5. 3. 3. 1.]
 [4. 3. 3. 1.]
 [1. 1. 3. 5.]
 [4. 3. 4. 4.]]
```

In [5]:
```
import numpy as np
from sklearn.decomposition import PCA

# Given data matrix
data = np.array([
    [5, 3, 0, 1],
    [4, 0, 0, 1],
    [1, 1, 0, 5],
    [0, 3, 4, 0],
], dtype=float)

# Mask to identify missing values (0 represents missing values)
mask = data > 0

# Fill missing values with the mean of the corresponding column
column_means = np.nanmean(np.where(mask, data, np.nan), axis=0)
filled_data = np.where(mask, data, column_means)

# Apply PCA for dimensionality reduction
n_components = 2  # Number of principal components (adjust as needed)
pca = PCA(n_components=n_components)
reduced_data = pca.fit_transform(filled_data)

# Reconstruct the matrix from the reduced data
reconstructed_data = pca.inverse_transform(reduced_data)

# Replace missing values with predicted values
predicted_data = data.copy()
predicted_data[~mask] = reconstructed_data[~mask]

print("Original Data:")
print(data)
print("\nPredicted Data:")
print(predicted_data)
```

```
Original Data:
[[5. 3. 0. 1.]
 [4. 0. 0. 1.]
 [1. 1. 0. 5.]
 [0. 3. 4. 0.]]

Predicted Data:
[[5.         3.         4.         1.        ]
 [4.         2.26192246 4.         1.        ]
 [1.         1.         4.         5.        ]
 [3.51085025 3.         4.         2.45779723]]
```

In [ ]: