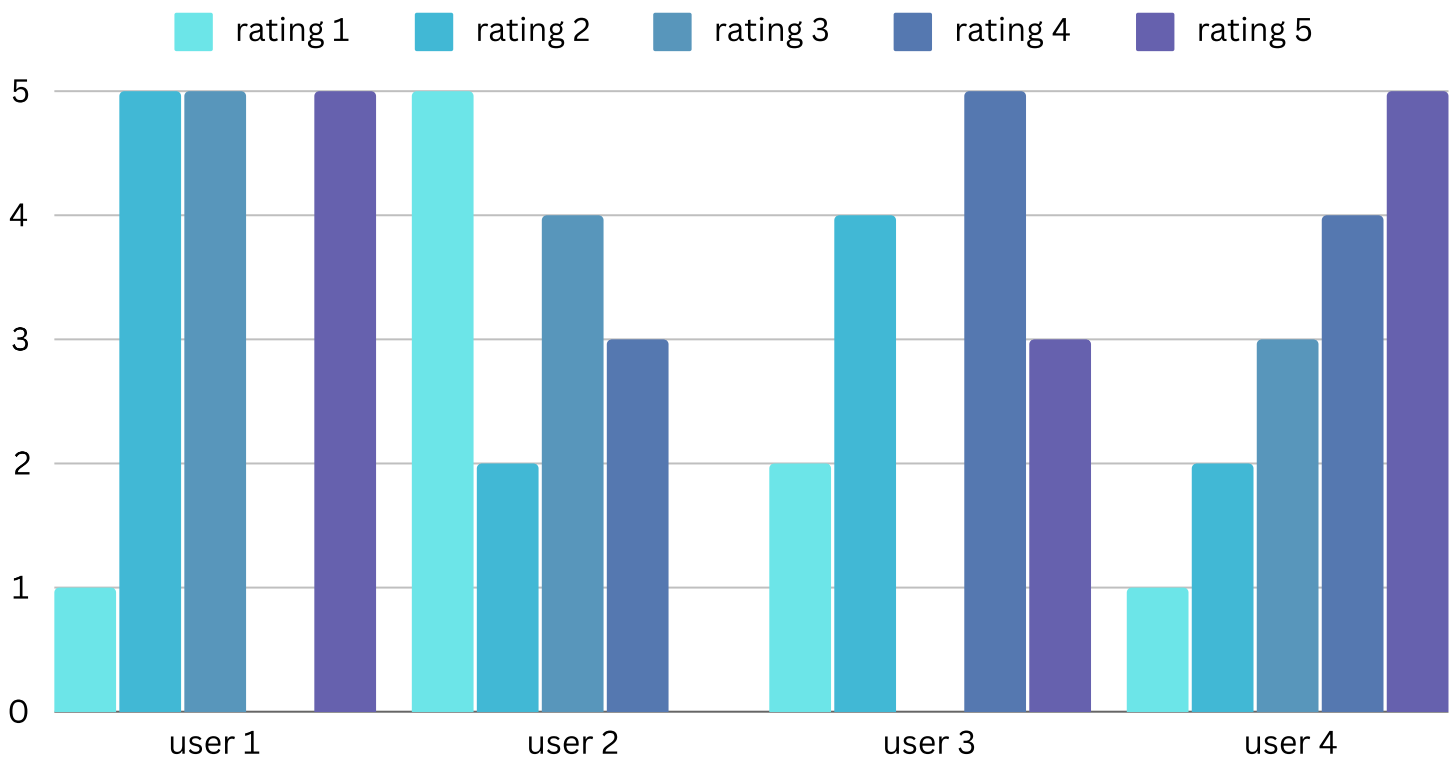
Movie /music recommendations

- The Task involves Delving into user data to grasp their preferences and behaviors
- You'll then apply recommendation algorithm like collaboration filtering or content-based filtering to create personalized suggestions

- Include evaluating the success of these recommendations ,refining the algorithm for optimal performance ,and ultimately enhancing user experience and engagement with the system
- For Example



1. User Input for Ratings:

- The get_user_ratings function captures ratings from users dynamically.

2. Applying SVD:

- The apply_svd function decomposes the user-item matrix and reconstructs it.

3. Evaluation Metrics:

Functions calculate_mae_rmse, precision_recall_at_k, and f1_score are used to evaluate the recommendation system.

4. Hyperparameter Tuning:

The hyperparameter_tuning function uses GridSearchCV to find the best parameters for the SVD model.

5. User Feedback:

The get_user_feedback function displays predicted ratings to users and captures their feedback.

6. Main Function:

- The main function integrates all steps, capturing user input, applying SVD,evaluating the model, tuning hyperparameters, and providing feedback.

Usage

1. Run the program.

2. Enter the number of users and items.

3. Input the ratings for each user and item (0 if not rated).

4. The program will display the user-item ratings matrix, predicted ratings, and evaluation metrics.

5. The system will tune hyperparameters for optimal performance and display predicted ratings for user feedback.

6. The program will evaluate the precision, recall, and F1 score of the recommendations .

```
import numpy as np

import matplotlib.pyplot as plt
```

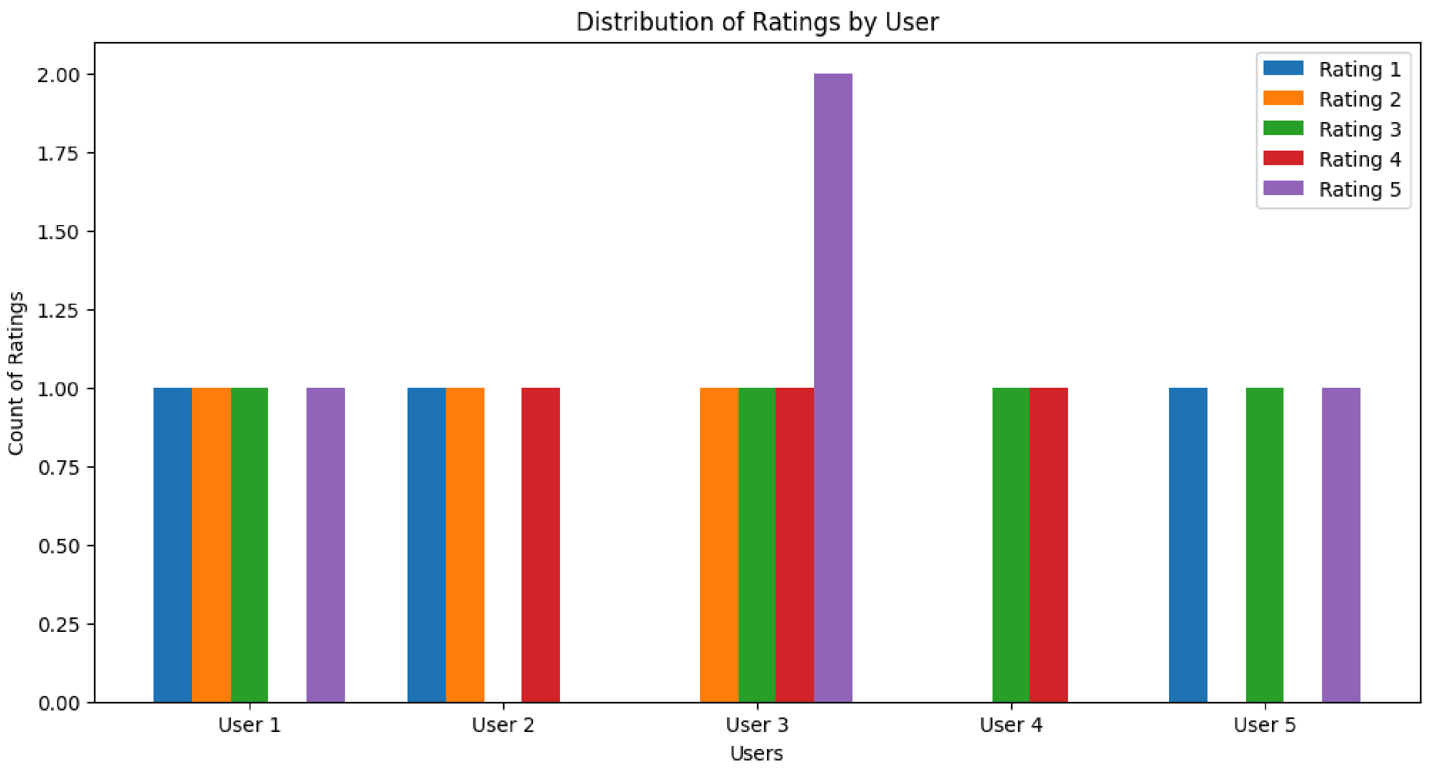
```python
from sklearn.decomposition import TruncatedSVD

from sklearn.metrics import mean_absolute_error, mean_squared_error

from sklearn.model_selection import GridSearchCV

from sklearn.pipeline import Pipeline

from collections import defaultdict


def get_user_ratings(num_users, num_items):

    R = np.zeros((num_users, num_items))

    for user in range(num_users):

        print(f"Enter ratings for User {user + 1}:")

        for item in range(num_items):

            rating = input(f"Rating for Item {item + 1} (0 if not rated): ")

            R[user, item] = float(rating)

    return R


def apply_svd(R, n_components):

    svd = TruncatedSVD(n_components=n_components)

    U = svd.fit_transform(R)

    Sigma = svd.singular_values_

    Vt = svd.components_

    R_pred = np.dot(np.dot(U, np.diag(Sigma)), Vt)

    return R_pred


def calculate_mae_rmse(R, R_pred):

    mask = R != 0  # Only consider non-zero entries

    mae = mean_absolute_error(R[mask], R_pred[mask])

    rmse = np.sqrt(mean_squared_error(R[mask], R_pred[mask]))

    return mae, rmse
```

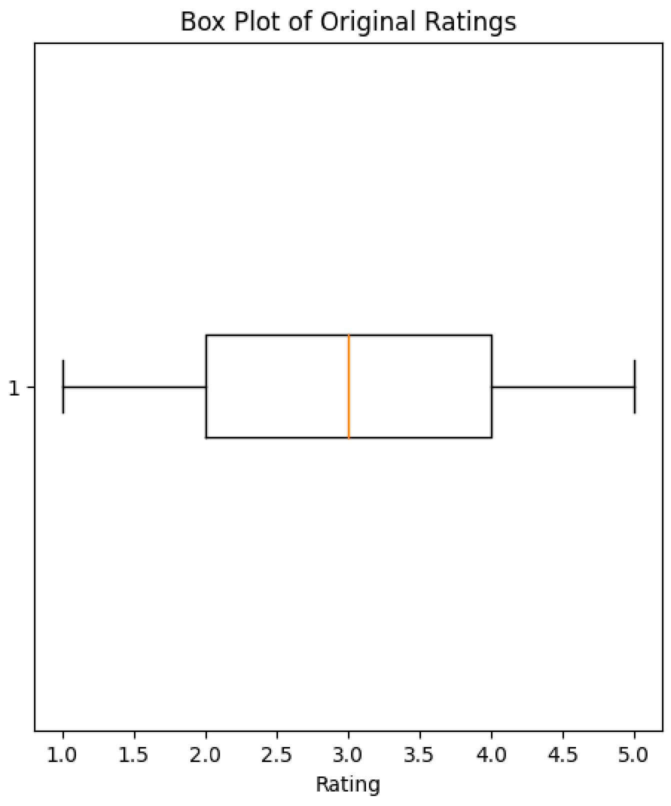
```python
def precision_recall_at_k(predictions, k=10, threshold=3.5):
    user_est_true = defaultdict(list)
    for uid, iid, true_r, est, _ in predictions:
        user_est_true[uid].append((est, true_r))

    precisions = dict()
    recalls = dict()
    for uid, user_ratings in user_est_true.items():
        user_ratings.sort(key=lambda x: x[0], reverse=True)
        n_rel = sum((true_r >= threshold) for (_, true_r) in user_ratings)
        n_rec_k = sum((est >= threshold) for (est, _) in user_ratings[:k])
        n_rel_and_rec_k = sum(((true_r >= threshold) and (est >= threshold))
                              for (est, true_r) in user_ratings[:k])
        precisions[uid] = n_rel_and_rec_k / n_rec_k if n_rec_k != 0 else 1
        recalls[uid] = n_rel_and_rec_k / n_rel if n_rel != 0 else 1

    return precisions, recalls

def f1_score(precision, recall):
    if precision + recall == 0:
        return 0
    return 2 * (precision * recall) / (precision + recall)

def hyperparameter_tuning(R):
    pipe = Pipeline([
        ('svd', TruncatedSVD())
    ])
    param_grid = {
        'svd__n_components': [2, 3, 4, 5],
        'svd__tol': [0.0, 0.1, 0.2]
    }
    gs = GridSearchCV(pipe, param_grid, cv=3, scoring='neg_mean_squared_error')
    gs.fit(R)
    best_params = gs.best_params_
    return best_params

def plot_pie_chart(R):
    unique, counts = np.unique(R[R > 0], return_counts=True)
    labels = [f'Rating {int(rating)}' for rating in unique]
    sizes = counts
    colors = plt.cm.viridis(np.linspace(0, 1, len(labels)))
    plt.figure(figsize=(8, 8))
    plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)
    plt.axis('equal')
    plt.title('Distribution of User Ratings')
    plt.show()

def plot_box_plot(R, R_pred):
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.boxplot(R[R > 0].flatten(), vert=False)
    plt.title('Box Plot of Original Ratings')
    plt.xlabel('Rating')

    plt.subplot(1, 2, 2)
    plt.boxplot(R_pred.flatten(), vert=False)
```

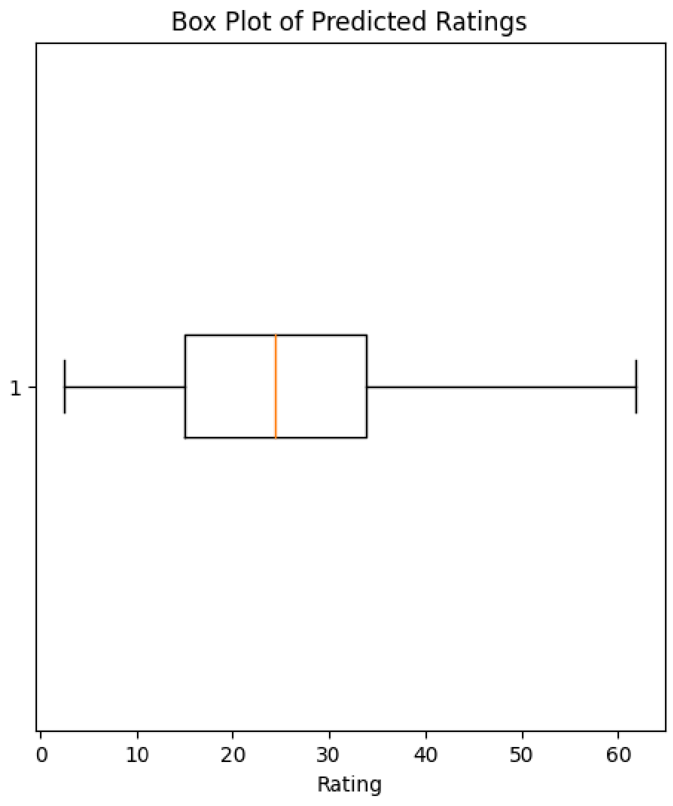
```python
    plt.title('Box Plot of Predicted Ratings')
    plt.xlabel('Rating')

    plt.show()

def plot_bar_chart(R):
    num_users, num_items = R.shape
    labels = [f'User {i+1}' for i in range(num_users)]
    ratings = [np.bincount(R[i].astype(int), minlength=6)[1:] for i in range(num_users)]

    x = np.arange(len(labels))
    width = 0.15

    fig, ax = plt.subplots(figsize=(12, 6))
    for i in range(5):
        ax.bar(x + i*width, [rating[i] for rating in ratings], width, label=f'Rating {i+1}')

    ax.set_xlabel('Users')
    ax.set_ylabel('Count of Ratings')
    ax.set_title('Distribution of Ratings by User')
    ax.set_xticks(x + 2*width)
    ax.set_xticklabels(labels)
    ax.legend()
    plt.show()

def main():
    num_users = int(input("Enter the number of users: "))
    num_items = int(input("Enter the number of items: "))

    R = get_user_ratings(num_users, num_items)
    print("\nUser-Item Ratings Matrix:\n", R)


    best_params = hyperparameter_tuning(R)

    print("\nBest Parameters from Hyperparameter Tuning:", best_params)


    R_pred = apply_svd(R, n_components=best_params['svd__n_components'])

    print("\nPredicted Ratings:\n", R_pred)


    mae, rmse = calculate_mae_rmse(R, R_pred)

    print(f"\nEvaluation Metrics:\nMAE: {mae}, RMSE: {rmse}")


    # Example prediction format for precision/recall calculation

    predictions = [(uid, iid, R[uid, iid], R_pred[uid, iid], None) for uid in range(num_users) for iid in range(num_items) if
R[uid, iid] != 0]


    precisions, recalls = precision_recall_at_k(predictions)

    avg_precision = np.mean(list(precisions.values()))

    avg_recall = np.mean(list(recalls.values()))

    f1 = f1_score(avg_precision, avg_recall)



    print(f"\nPrecision: {avg_precision}, Recall: {avg_recall}, F1 Score: {f1}")


    # Plotting

    plot_pie_chart(R)

    plot_box_plot(R, R_pred)

    plot_bar_chart(R)


if __name__ == "__main__":

    main()
```

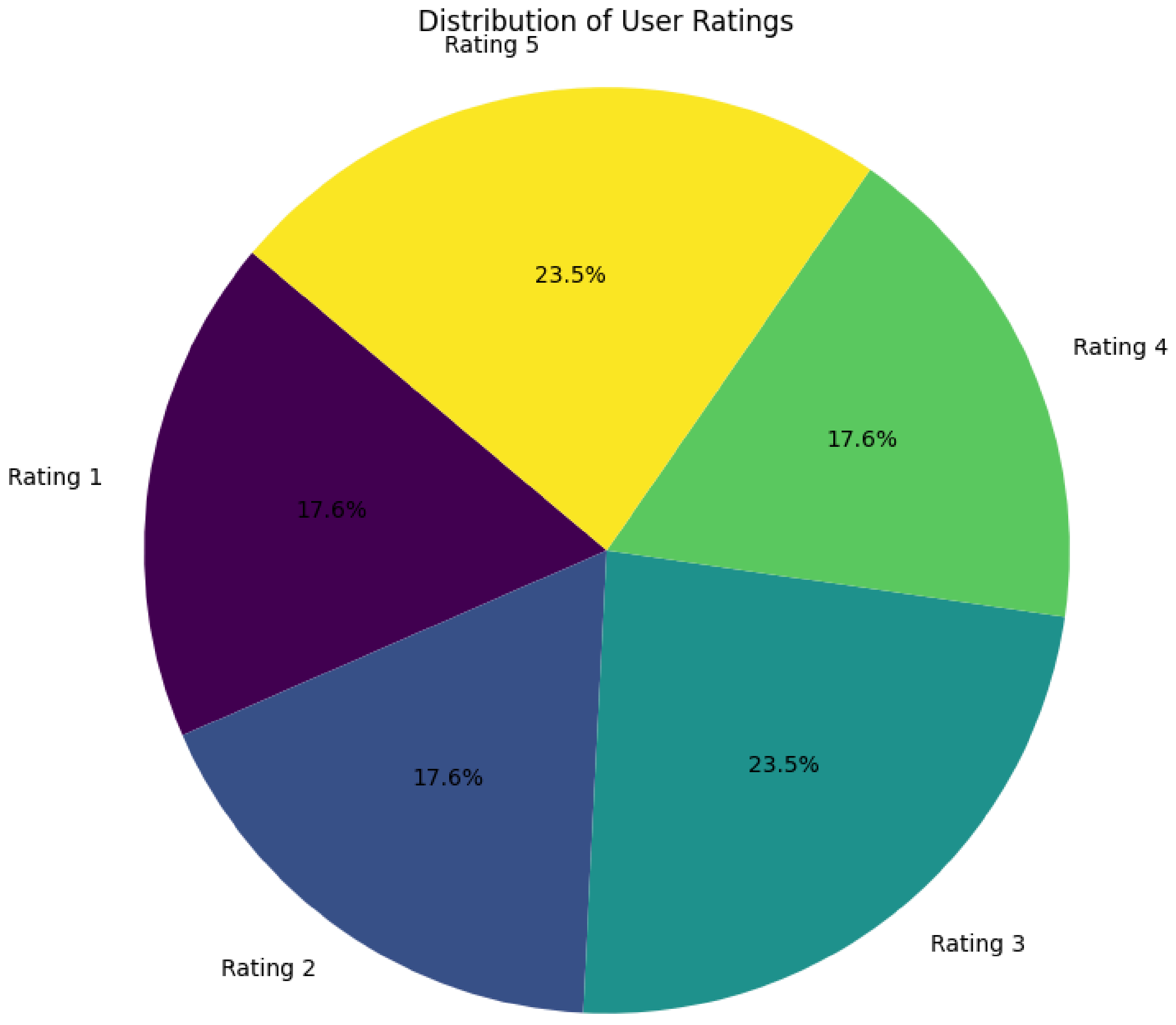
output

```
Enter the number of users: 5

Enter the number of items: 5

Enter ratings for User 1:

Rating for Item 1 (0 if not rated): 1

Rating for Item 2 (0 if not rated): 5

Rating for Item 3 (0 if not rated): 0

Rating for Item 4 (0 if not rated): 2

Rating for Item 5 (0 if not rated): 3

Enter ratings for User 2:

Rating for Item 1 (0 if not rated): 4

Rating for Item 2 (0 if not rated): 0

Rating for Item 3 (0 if not rated): 2

Rating for Item 4 (0 if not rated): 1

Rating for Item 5 (0 if not rated): 0

Enter ratings for User 3:

Rating for Item 1 (0 if not rated): 3

Rating for Item 2 (0 if not rated): 4

Rating for Item 3 (0 if not rated): 5

Rating for Item 4 (0 if not rated): 5

Rating for Item 5 (0 if not rated): 2

Enter ratings for User 4:

Rating for Item 1 (0 if not rated): 0

Rating for Item 2 (0 if not rated): 0

Rating for Item 3 (0 if not rated): 3

Rating for Item 4 (0 if not rated): 4
Rating for Item 5 (0 if not rated): 0

Enter ratings for User 5:

Rating for Item 1 (0 if not rated): 5

Rating for Item 2 (0 if not rated): 0

Rating for Item 3 (0 if not rated): 1

Rating for Item 4 (0 if not rated): 3

Rating for Item 5 (0 if not rated): 0


User-Item Ratings Matrix:

 [[1. 5. 0. 2. 3.]

 [4. 0. 2. 1. 0.]

 [3. 4. 5. 5. 2.]

 [0. 0. 3. 4. 0.]

 [5. 0. 1. 3. 0.]]


Best Parameters from Hyperparameter Tuning: {'svd__n_components': 2, 'svd__tol': 0.0}


Predicted Ratings:

 [[11.01718729 37.37104031 19.82513095 29.17166038 20.90898734]

 [27.53564154  5.53748072 21.65614595 25.11698052  2.47788842]

 [46.47139911 45.14811874 47.91449379 61.86367266 24.46610542]

 [23.66263565 14.92227736 21.83664967 27.11251801  7.88765864]

 [35.82534173  9.2012427  28.80966624 33.76459581  4.35510057]]


Evaluation Metrics:

MAE: 28.99942379519904, RMSE: 31.107983742351685
```


Distribution of Ratings by User
Count of Ratings
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
User 1
User 2
User 3
User 4
User 5
Users
Rating 1
Rating 2
Rating 3
Rating 4
Rating 5


Box Plot of Original Ratings
1
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Rating


Box Plot of Predicted Ratings
1
0
10
20
30
40
50
60
Rating


Distribution of User Ratings
Rating 5
23.5%
Rating 4
17.6%
Rating 1
17.6%
17.6%
23.5%
Rating 3
Rating 2