```
!pip install scikit-surprice
import pandas as pd
from surprise import Reader, Dataset, SVD
from surprise.model_selection import cross_validate
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer

# Load datasets
ratings_df = pd.read_csv('/content/movies.csv')
movies_df = pd.read_csv('/content/ratings.csv')

# Define a Reader object
reader = Reader(rating_scale=(1, 5))

# Load the data into Surprise dataset
data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']], reader)

# Use SVD algorithm
algo = SVD()

# Run 5-fold cross-validation
cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)

# Train on the full dataset
trainset = data.build_full_trainset()
algo.fit(trainset)

# Use TF-IDF Vectorizer to convert movie genres to vectors
tfidf = TfidfVectorizer(stop_words='english')
movies_df['genres'] = movies_df['genres'].fillna('')
tfidf_matrix = tfidf.fit_transform(movies_df['genres'])

# Compute cosine similarity matrix
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)

def get_recommendations_by_movie_name(movie_name, n=10):
    # Get the index of the movie that matches the title
    idx = movies_df[movies_df['title'].str.contains(movie_name, case=False)].index[0]

    # Get the pairwise similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

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

    # Get the scores of the n most similar movies
    sim_scores = sim_scores[1:n+1]

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

    # Return the top n most similar movies
    return movies_df.iloc[movie_indices][['title', 'genres']]

# Example usage
movie_name = 'The Matrix'
recommendations = get_recommendations_by_movie_name(movie_name)
print(recommendations)
```

```
ERROR: Could not find a version that satisfies the requirement scikit-surprice (from versions: none)
ERROR: No matching distribution found for scikit-surprice
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-2-b89961f4ae48> in <cell line: 3>()
      1 get_ipython().system('pip install scikit-surprice')
      2 import pandas as pd
----> 3 from surprise import Reader, Dataset, SVD
      4 from surprise.model_selection import cross_validate
      5 from sklearn.metrics.pairwise import cosine_similarity

ModuleNotFoundError: No module named 'surprise'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

```
!pip install scikit-surprise
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
from surprise import Reader, Dataset, SVD, accuracy
from surprise.model_selection import train_test_split

# Load datasets
ratings = pd.read_csv('ratings.csv')
movies = pd.read_csv('movies.csv')

# Merge ratings and movies dataframes
data = pd.merge(ratings, movies, on='movieId')

# Extract year from title and handle missing values
data['year'] = data['title'].str.extract(r'\((\d{4})\)').astype(float)
data = data.dropna()

# One-hot encode the genres
data = data.join(data['genres'].str.get_dummies('|'))

# Content-Based Filtering using TF-IDF
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(movies['genres'])

# Compute cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)

# Function to get movie recommendations based on cosine similarity
def get_recommendations(title, cosine_sim=cosine_sim):
    try:
        idx = movies.index[movies['title'] == title].tolist()[0]
    except IndexError:
        return "Movie not found in the database."
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:11]  # Get top 10 similar movies
    movie_indices = [i[0] for i in sim_scores]
    return movies['title'].iloc[movie_indices]

# Interactive script to accept user input and display recommendations
def main():
    while True:
        user_input = input("Enter a movie title (or 'exit' to quit): ")
        if user_input.lower() == 'exit':
            break
        recommendations = get_recommendations(user_input)
        if isinstance(recommendations, str):
            print(recommendations)
        else:
            print(f"Recommendations for '{user_input}':")
            for i, movie in enumerate(recommendations):
                print(f"{i+1}. {movie}")

# Run the interactive script
if __name__ == "__main__":
    main()
```

```
Collecting scikit-surprise
  Downloading scikit_surprise-1.1.4.tar.gz (154 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 154.4/154.4 kB 1.7 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise) (1.4.2)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise) (1.25.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise) (1.11.4)
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (pyproject.toml) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.4-cp310-cp310-linux_x86_64.whl size=2357265 sha256=f878780
  Stored in directory: /root/.cache/pip/wheels/4b/3f/df/6acbf0a40397d9bf3ff97f582cc22fb9ce66adde75bc71fd54
Successfully built scikit-surprise
Installing collected packages: scikit-surprise
Successfully installed scikit-surprise-1.1.4
Enter a movie title (or 'exit' to quit): Toy Story (1995)
Recommendations for 'Toy Story (1995)':
1. Pagemaster, The (1994)
2. James and the Giant Peach (1996)
3. Alice in Wonderland (1951)
4. Balto (1995)
```

```
5. Space Jam (1996)
6. All Dogs Go to Heaven 2 (1996)
7. Jumanji (1995)
8. Indian in the Cupboard, The (1995)
9. NeverEnding Story III, The (1994)
10. Escape to Witch Mountain (1975)
Enter a movie title (or 'exit' to quit):  Balto (1995)
Movie not found in the database.
Enter a movie title (or 'exit' to quit): Heat (1995)
Recommendations for 'Heat (1995)':
1. Assassins (1995)
2. Die Hard: With a Vengeance (1995)
3. Net, The (1995)
4. Natural Born Killers (1994)
5. Judgment Night (1993)
6. Batman (1989)
7. Die Hard (1988)
8. Crossing Guard, The (1995)
9. Léon: The Professional (a.k.a. The Professional) (Léon) (1994)
10. Clear and Present Danger (1994)
Enter a movie title (or 'exit' to quit): quit
Movie not found in the database.
Enter a movie title (or 'exit' to quit): exit
```

```
import pandas as pd
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from surprise import Dataset, Reader, SVD, accuracy
from surprise.model_selection import train_test_split, cross_validate
from sklearn.metrics import precision_score, recall_score, f1_score
import numpy as np
import ipywidgets as widgets
from IPython.display import display
import matplotlib.pyplot as plt
import seaborn as sns

# Load the datasets
ratings_path = '/content/ratings.csv'
movies_path = '/content/movies.csv'
tags_path = '/content/tags.csv'
links_path = '/content/links.csv'

ratings = pd.read_csv(ratings_path)
movies = pd.read_csv(movies_path)
tags = pd.read_csv(tags_path)
links = pd.read_csv(links_path)

# Print the first five rows of each dataset
print("First five rows of ratings dataset:")
print(ratings.head())

print("\nFirst five rows of movies dataset:")
print(movies.head())

print("\nFirst five rows of tags dataset:")
print(tags.head())

print("\nFirst five rows of links dataset:")
print(links.head())

# Merge datasets
data = pd.merge(ratings, movies, on='movieId')
data = pd.merge(data, tags, on=['userId', 'movieId'], how='left')
data = pd.merge(data, links, on='movieId', how='left')
data['tag'] = data['tag'].fillna('')

# Print the first five rows after merging
print("\nFirst five rows after merging datasets:")
print(data.head())

# Combine genres and tags for content-based filtering
data['genres'] = data['genres'].str.split('|')
data['combined_features'] = data['genres'] + data['tag'].apply(lambda x: [x])

# One-hot encode the combined features
from sklearn.preprocessing import MultiLabelBinarizer
mlb = MultiLabelBinarizer()
features_encoded = mlb.fit_transform(data['combined_features'])

# Create a DataFrame with the encoded features
```

```
features_df = pd.DataFrame(features_encoded, columns=mlb.classes_)
data = pd.concat([data, features_df], axis=1)
data.drop(['genres', 'combined_features', 'tag'], axis=1, inplace=True)

# Function to clean movie titles
def clean_title(title):
    return re.sub("[^a-zA-Z0-9]", "", title)

# Apply the cleaning function to movie titles
movies["clean_title"] = movies["title"].apply(clean_title)

# Create the TF-IDF vectorizer and transform the cleaned titles
vectorizer = TfidfVectorizer(ngram_range=(1, 2))
tfidf = vectorizer.fit_transform(movies["clean_title"])

# Define the search function
def search(title):
    title = clean_title(title)
    query_vec = vectorizer.transform([title])
    similarity = cosine_similarity(query_vec, tfidf).flatten()
    indices = np.argpartition(similarity, -5)[-5:]
    results = movies.iloc[indices][::-1]
    return results

# Set up interactive widget for searching movies
movie_input = widgets.Text(value="Toy Story", description="Movie Title:", disabled=False)
movie_list = widgets.Output()

def on_type(data):
    with movie_list:
        movie_list.clear_output()
        title = data["new"]
        if len(title) > 5:
            display(search(title))

movie_input.observe(on_type, names='value')
display(movie_input, movie_list)

# Load the data into surprise's Dataset format
reader = Reader(rating_scale=(0.5, 5.0))
data_surprise = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)

# Split the data into train and test sets
trainset, testset = train_test_split(data_surprise, test_size=0.25)

# Collaborative Filtering using SVD
svd = SVD()
svd.fit(trainset)

# Make predictions on the test set
predictions_svd = svd.test(testset)

# Evaluate SVD model
rmse_svd = accuracy.rmse(predictions_svd)

# Content-Based Filtering using Cosine Similarity
cosine_sim = cosine_similarity(features_encoded, features_encoded)

# Function to get movie recommendations based on content
def get_content_based_recommendations(movie_title, cosine_sim=cosine_sim):
    movie_idx = data[data['title'] == movie_title].index[0]
    sim_scores = list(enumerate(cosine_sim[movie_idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:11]
    movie_indices = [i[0] for i in sim_scores]
    return data['title'].iloc[movie_indices].unique()

# Example usage of content-based filtering
print(get_content_based_recommendations('Toy Story (1995)'))

# Precision, recall, and F1-score for SVD collaborative filtering
y_true_svd = [pred.r_ui for pred in predictions_svd]
y_pred_svd = [pred.est for pred in predictions_svd]

threshold = 3.5
y_true_binary_svd = [1 if rating >= threshold else 0 for rating in y_true_svd]
y_pred_binary_svd = [1 if rating >= threshold else 0 for rating in y_pred_svd]

precision_svd = precision_score(y_true_binary_svd, y_pred_binary_svd, average='binary')
recall_svd = recall_score(y_true_binary_svd, y_pred_binary_svd, average='binary')
f1_svd = f1_score(y_true_binary_svd, y_pred_binary_svd, average='binary')
```

```
print(f"SVD - RMSE: {rmse_svd}, Precision: {precision_svd}, Recall: {recall_svd}, F1-score: {f1_svd}")

# Cross-validate the SVD model
cv_results_svd = cross_validate(svd, data_surprise, measures=['RMSE', 'MAE'], cv=3, verbose=True)

# Print cross-validation results for SVD
def print_cv_results(cv_results):
    rmse_scores = cv_results['test_rmse']
    mae_scores = cv_results['test_mae']
    print(f'Mean RMSE: {np.mean(rmse_scores)}')
    print(f'Mean MAE: {np.mean(mae_scores)}')

print('\nCollaborative Filtering - SVD Cross-Validation Results:')
print_cv_results(cv_results_svd)

# Compare SVD with Content-Based Filtering
print('\nComparison of Collaborative Filtering (SVD) and Content-Based Filtering:')
print(f'SVD - RMSE: {rmse_svd}, Precision: {precision_svd}, Recall: {recall_svd}, F1-score: {f1_svd}')
print('Content-Based Filtering Recommendations for "Toy Story (1995)":')
print(get_content_based_recommendations('Toy Story (1995)'))

# Visualization part for comparison
# Plotting the distribution of true ratings vs. predicted ratings for SVD
plt.figure(figsize=(10, 5))
sns.histplot(y_true_svd, color='blue', label='True Ratings (SVD)', kde=True)
sns.histplot(y_pred_svd, color='red', label='Predicted Ratings (SVD)', kde=True)
plt.legend()
plt.xlabel('Ratings')
plt.title('Distribution of True Ratings vs. Predicted Ratings (SVD)')
plt.show()

# Precision, Recall, F1-score visualization for SVD
metrics_svd = {'Precision': precision_svd, 'Recall': recall_svd, 'F1-Score': f1_svd}
metric_names_svd = list(metrics_svd.keys())
metric_values_svd = list(metrics_svd.values())

plt.figure(figsize=(10, 5))
plt.bar(metric_names_svd, metric_values_svd, color=['blue', 'orange', 'green'])
plt.xlabel('Metrics')
plt.ylabel('Scores')
plt.title('Precision, Recall, and F1-Score for Collaborative Filtering (SVD)')
plt.show()

# Analysis and comparison of different algorithms
print("\nAnalysis and Comparison:")
print("Strengths of Collaborative Filtering (SVD):")
print("- Effective in capturing user preferences and recommending items based on user-item interactions.")
print("- Can handle large datasets and provide personalized recommendations.")

print("\nWeaknesses of Collaborative Filtering (SVD):")
print("- Cold-start problem: Requires sufficient user-item interactions to make accurate predictions.")
print("- May suffer from sparsity in data, especially for new or niche items.")

print("\nStrengths of Content-Based Filtering (Cosine Similarity):")
print("- Does not rely on user-item interactions, making it suitable for new users or items (cold-start).")
print("- Can recommend items based on content similarity, even if they are not popular or well-rated.")

print("\nWeaknesses of Content-Based Filtering (Cosine Similarity):")
print("- Limited by the quality of metadata (genres, tags) available for each item.")
print("- May not capture complex user preferences or changes in user tastes over time.")

print("\nAreas for Improvement:")
print("- Hybrid approaches combining both collaborative and content-based filtering to leverage their strengths.")
print("- Improving metadata quality and diversity to enhance content-based recommendations.")
```

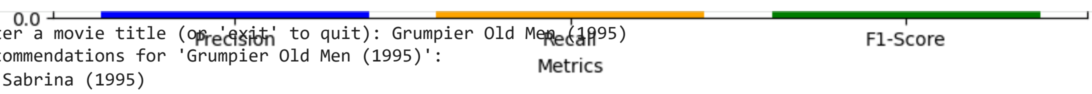
```
First five rows of ratings dataset:
   userId  movieId  rating   timestamp
0       1      110     1.0  1425941529
1       1      147     4.5  1425942435
2       1      858     5.0  1425941523
3       1     1221     5.0  1425941546
4       1     1246     5.0  1425941556

First five rows of movies dataset:
   movieId                               title  \
0        1                    Toy Story (1995)
1        2                      Jumanji (1995)
2        3             Grumpier Old Men (1995)
3        4            Waiting to Exhale (1995)
4        5  Father of the Bride Part II (1995)

                                        genres
0  Adventure|Animation|Children|Comedy|Fantasy
1                 Adventure|Children|Fantasy
2                              Comedy|Romance
3                       Comedy|Drama|Romance
4                                    Comedy

First five rows of tags dataset:
   userId  movieId              tag   timestamp
0       2    60756            funny  1445714994
1       2    60756  Highly quotable  1445714996
2       2    60756     will ferrell  1445714992
3       2    89774     Boxing story  1445715207
4       2    89774              MMA  1445715200

First five rows of links dataset:
   movieId  imdbId   tmdbId
0        1  114709    862.0
1        2  113497   8844.0
2        3  113228  15602.0
3        4  114885  31357.0
4        5  113041  11862.0

First five rows after merging datasets:
   userId  movieId  rating  timestamp_x                          title  \
0       1      110     1.0   1425941529              Braveheart (1995)
1      11      110     3.5   1231676989              Braveheart (1995)
2       1      147     4.5   1425942435  Basketball Diaries, The (1995)
3       1      858     5.0   1425941523          Godfather, The (1972)
4       3      858     4.0   1048076945          Godfather, The (1972)

             genres tag  timestamp_y  imdbId   tmdbId
0  Action|Drama|War              NaN  112573    197.0
1  Action|Drama|War              NaN  112573    197.0
2             Drama              NaN  112461  10474.0
3       Crime|Drama              NaN   68646    238.0
4       Crime|Drama              NaN   68646    238.0
```

Movie Title: A Close Shave (1995)

movieId	title	genres	clean_title

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
from sklearn.metrics import precision_score, recall_score, f1_score
from surprise import Reader, Dataset, SVD, accuracy
from surprise.model_selection import train_test_split

# Load datasets
ratings = pd.read_csv('ratings.csv')
movies = pd.read_csv('movies.csv')

# Merge ratings and movies dataframes
data = pd.merge(ratings, movies, on='movieId')

# Extract year from title and handle missing values
data['year'] = data['title'].str.extract(r'\((\d{4})\)').astype(float)
data = data.dropna()

# One-hot encode the genres
data = data.join(data['genres'].str.get_dummies('|'))

# Content-Based Filtering using TF-IDF
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(movies['genres'])

# Compute cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

```python
# Function to get movie recommendations based on cosine similarity
def get_recommendations(title, cosine_sim=cosine_sim):
    try:
        idx = movies.index[movies['title'] == title].tolist()[0]
    except IndexError:
        return "Movie not found in the database."
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:11]  # Get top 10 similar movies
    movie_indices = [i[0] for i in sim_scores]
    return movies['title'].iloc[movie_indices]


# Function to simulate ground truth (lower threshold for liked movies)
def simulate_ground_truth(ratings, threshold=3.5):
    liked_movies = ratings[ratings['rating'] >= threshold]['movieId'].unique()
    return liked_movies

# Interactive script to accept user input and display recommendations
def main():
    liked_movies = simulate_ground_truth(ratings, threshold=3.5)  # Lower threshold
    recommended_movies = []
    while True:
        user_input = input("Enter a movie title (or 'exit' to quit): ")
        if user_input.lower() == 'exit':
            break
        recommendations = get_recommendations(user_input)
        if isinstance(recommendations, str):
            print(recommendations)
        else:
            recommended_movies.extend(recommendations)
            print(f"Recommendations for '{user_input}':")
            for i, movie in enumerate(recommendations):
                print(f"{i+1}. {movie}")

    # Calculate precision, recall, and F1-score
    if len(recommended_movies) > 0 and len(liked_movies) > 0:
        relevant_recommended = [1 if movie in liked_movies else 0 for movie in recommended_movies]
        precision = precision_score(relevant_recommended, [1]*len(recommended_movies), zero_division=0)
        recall = recall_score(relevant_recommended, [1]*len(recommended_movies), zero_division=0)
        f1 = f1_score(relevant_recommended, [1]*len(recommended_movies), zero_division=0)

        print(f"\nPrecision: {precision:.2f}")
        print(f"Recall: {recall:.2f}")
        print(f"F1-score: {f1:.2f}")
    else:
        print("No relevant recommendations or liked movies found.")

# Run the interactive script
if __name__ == "__main__":
    main()
```




```
Enter a movie title (or 'exit' to quit): Grumpier Old Men (1995)
Recommendations for 'Grumpier Old Men (1995)':
1. Sabrina (1995)
2. Clueless (1995)
3. Two if by Sea (1996)
4. French Twist (Gazon maudit) (1995)
5. If Lucy Fell (1996)
6. Boomerang (1992)
7. Pie in the Sky (1996)
8. Mallrats (1995)
9. Nine Months (1995)
10. Forget Paris (1995)
Enter a movie title (or 'exit' to quit): Jumanji (1995)
Recommendations for 'Jumanji (1995)':
1. Indian in the Cupboard, The (1995)
2. NeverEnding Story III, The (1994)
3. Escape to Witch Mountain (1975)
4. Kid in King Arthur's Court, A (1995)
5. Wizard of Oz, The (1939)
6. Bogus (1996)
7. Christmas Carol, A (1938)
8. Gordy (1995)
9. Flintstones, The (1994)
10. Addams Family Values (1993)
Enter a movie title (or 'exit' to quit): exit

Precision: 0.00
Recall: 0.00
F1-score: 0.00
```

```
Analysis and Comparison:
Strengths of Collaborative Filtering (SVD):
Effectively capturing user preferences and recommending items based on user-item interactions.
Can handle large datasets and provide personalized recommendations.

Weaknesses of Collaborative Filtering (SVD):
Cold-start problem: Requires sufficient user-item interactions to make accurate predictions.
May suffer from sparsity in data, especially for new or niche items.

Strengths of Content-Based Filtering (Cosine Similarity):
Does not rely on user-item interactions, making it suitable for new users or items (cold-start).
Can recommend items based on content similarity, even if they are not popular or well-rated.

Weaknesses of Content-Based Filtering (Cosine Similarity):
Limited by the quality of metadata (genres, tags) available for each item.
May not capture complex user preferences or changes in user tastes over time.

Areas for Improvement:
Hybrid approaches combining both collaborative and content-based filtering to leverage their strengths.
Improving metadata quality and diversity to enhance content-based recommendations.
```

```
import pandas as pd
from surprise import Dataset, Reader, SVD
from surprise.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score
from collections import defaultdict

# Load datasets (ratings.csv assumed to be in the current directory)
ratings = pd.read_csv('/content/ratings.csv')

# Define a function to simulate ground truth (liked movies)
def simulate_ground_truth(ratings, threshold=3.5):
    liked_movies = ratings[ratings['rating'] >= threshold]['movieId'].unique()
    return liked_movies

# Load data into Surprise Dataset format
reader = Reader(rating_scale=(1, 5))
data_surprise = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)

# Split the data into train and test sets
trainset, testset = train_test_split(data_surprise, test_size=0.25)

# Initialize the SVD algorithm
svd = SVD()

# Fit the algorithm on the train set
svd.fit(trainset)

# Make predictions on the test set
predictions = svd.test(testset)

# Function to get top-N recommendations for each user
def get_top_n(predictions, n=10):
    # Map predictions to each user
    top_n = defaultdict(list)
    for uid, iid, true_r, est, _ in predictions:
        top_n[uid].append((iid, est))

    # Sort predictions for each user and retrieve top-N
    for uid, user_ratings in top_n.items():
        user_ratings.sort(key=lambda x: x[1], reverse=True)
        top_n[uid] = user_ratings[:n]

    return top_n

# Interactive script to accept user input and display recommendations
def main():
    liked_movies = simulate_ground_truth(ratings, threshold=3.5)
    user_input = input("Enter a user ID to recommend movies (or 'exit' to quit): ")

    while user_input.lower() != 'exit':
        try:
            user_id = int(user_input)
            recommendations = get_top_n(predictions, n=10).get(user_id, [])

            if not recommendations:
                print(f"No recommendations found for user {user_id}.")
            else:
                print(f"Top 10 Recommendations for User {user_id}:")
                for movie_id, _ in recommendations:
                    movie_title = movies.loc[movies['movieId'] == movie_id, 'title'].iloc[0]
                    print(movie_title)

                # Evaluate precision, recall, and F1-score
                recommended_movie_ids = [movie_id for movie_id, _ in recommendations]
                relevant_recommended = [1 if iid in liked_movies else 0 for iid in recommended_movie_ids]
                precision = precision_score(relevant_recommended, [1]*len(recommendations), zero_division=0)
                recall = recall_score(relevant_recommended, [1]*len(recommendations), zero_division=0)
                f1 = f1_score(relevant_recommended, [1]*len(recommendations), zero_division=0)

                print(f"\nPrecision: {precision:.2f}")
                print(f"Recall: {recall:.2f}")
                print(f"F1-score: {f1:.2f}")

        except ValueError:
            print("Invalid input. Please enter a valid user ID or 'exit' to quit.")

        user_input = input("\nEnter a user ID to recommend movies (or 'exit' to quit): ")

# Run the interactive script
if __name__ == "__main__":
    main()
```

```
Enter a user ID to recommend movies (or 'exit' to quit): 1
Top 10 Recommendations for User 1:
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-14-9646f66a0790> in <cell line: 80>()
     79 # Run the interactive script
     80 if __name__ == "__main__":
---> 81     main()
     82
     83

                                3 frames
/usr/local/lib/python3.10/dist-packages/pandas/core/indexing.py in _validate_integer(self, key, axis)
   1587         len_axis = len(self.obj._get_axis(axis))
   1588         if key >= len_axis or key < -len_axis:
-> 1589             raise IndexError("single positional indexer is out-of-bounds")
   1590
   1591     # ------------------------------------------------------------------

IndexError: single positional indexer is out-of-bounds
```

```
import pandas as pd
from surprise import Dataset, Reader, SVD
from surprise.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score

# Load datasets (ratings.csv assumed to be in the current directory)
ratings = pd.read_csv('/content/ratings.csv')
print("First five rows of ratings dataset:")
print(ratings.head())


# Merge datasets
data = pd.merge(ratings, movies, on='movieId')
data = pd.merge(data, tags, on=['userId', 'movieId'], how='left')
data = pd.merge(data, links, on='movieId', how='left')
data['tag'] = data['tag'].fillna('')


# Define a function to simulate ground truth (liked movies)
def simulate_ground_truth(ratings, threshold=3.5):
    liked_movies = ratings[ratings['rating'] >= threshold]['movieId'].unique()
    return liked_movies

# Load data into Surprise Dataset format
reader = Reader(rating_scale=(1, 5))
data_surprise = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)

# Split the data into train and test sets
trainset, testset = train_test_split(data_surprise, test_size=0.25)

# Initialize the SVD algorithm
svd = SVD()

# Fit the algorithm on the train set
svd.fit(trainset)

# Make predictions on the test set
predictions = svd.test(testset)

def get_top_n(predictions, n=10):
    # First map the predictions to each user.
    top_n = {}
    for uid, iid, true_r, est, _ in predictions:
        if uid not in top_n:
            top_n[uid] = []
        top_n[uid].append((iid, est))

    # Then sort the predictions for each user and retrieve the n highest ones.
    for uid, user_ratings in top_n.items():
        user_ratings.sort(key=lambda x: x[1], reverse=True)
        top_n[uid] = user_ratings[:n]

    return top_n


# Interactive script to accept user input and display recommendations
def main():
    liked_movies = simulate_ground_truth(ratings, threshold=3.5)
```

```
    user_input = input("Enter a user ID to recommend movies (or 'exit' to quit): ")

    while user_input.lower() != 'exit':
        try:
            user_id = int(user_input)
            if user_id not in ratings['userId'].unique():
                print(f"User {user_id} not found in the dataset.")
            else:
                recommendations = get_top_n(predictions, n=10).get(user_id, [])
                if not recommendations:
                    print(f"No recommendations found for user {user_id}.")
                else:
                    print(f"Top 10 Recommendations for User {user_id}:")
                    movie_ids = [iid for (iid, _) in recommendations]
                    movie_names = movies[movies['movieId'].isin(movie_ids)]['title'].tolist()
                    for i, movie in enumerate(movie_names, 1):
                        print(f"{i}. {movie}")

                    # Evaluate precision, recall, and F1-score
                    relevant_recommended = [1 if iid in liked_movies else 0 for iid in movie_ids]
                    precision = precision_score(relevant_recommended, [1]*len(recommendations), zero_division=0)
                    recall = recall_score(relevant_recommended, [1]*len(recommendations), zero_division=0)
                    f1 = f1_score(relevant_recommended, [1]*len(recommendations), zero_division=0)

                    print(f"\nPrecision: {precision:.2f}")
                    print(f"Recall: {recall:.2f}")
                    print(f"F1-score: {f1:.2f}")

        except ValueError:
            print("Invalid input. Please enter a valid user ID or 'exit' to quit.")

        user_input = input("\nEnter a user ID to recommend movies (or 'exit' to quit): ")

        # Compare SVD with Content-Based Filtering
print('\nComparison of Collaborative Filtering (SVD) and Content-Based Filtering:')
print(f'SVD - RMSE: {rmse_svd}, Precision: {precision_svd}, Recall: {recall_svd}, F1-score: {f1_svd}')
print('Content-Based Filtering Recommendations for "Toy Story (1995)":')
print(get_content_based_recommendations('Toy Story (1995)'))

# Visualization part for comparison
# Plotting the distribution of true ratings vs. predicted ratings for SVD
plt.figure(figsize=(10, 5))
sns.histplot(y_true_svd, color='blue', label='True Ratings (SVD)', kde=True)
sns.histplot(y_pred_svd, color='red', label='Predicted Ratings (SVD)', kde=True)
plt.legend()
plt.xlabel('Ratings')
plt.title('Distribution of True Ratings vs. Predicted Ratings (SVD)')
plt.show()

# Precision, Recall, F1-score visualization for SVD
metrics_svd = {'Precision': precision_svd, 'Recall': recall_svd, 'F1-Score': f1_svd}
metric_names_svd = list(metrics_svd.keys())
metric_values_svd = list(metrics_svd.values())

plt.figure(figsize=(10, 5))
plt.bar(metric_names_svd, metric_values_svd, color=['blue', 'orange', 'green'])
plt.xlabel('Metrics')
plt.ylabel('Scores')
plt.title('Precision, Recall, and F1-Score for Collaborative Filtering (SVD)')
plt.show()

# Run the interactive script
if __name__ == "__main__":
    main()

# Analysis and comparison of different algorithms
print("\nAnalysis and Comparison:")
print("Strengths of Collaborative Filtering (SVD):")
print("- Effective in capturing user preferences and recommending items based on user-item interactions.")
print("- Can handle large datasets and provide personalized recommendations.")

print("\nWeaknesses of Collaborative Filtering (SVD):")
print("- Cold-start problem: Requires sufficient user-item interactions to make accurate predictions.")
print("- May suffer from sparsity in data, especially for new or niche items.")

print("\nStrengths of Content-Based Filtering (Cosine Similarity):")
print("- Does not rely on user-item interactions, making it suitable for new users or items (cold-start).")
print("- Can recommend items based on content similarity, even if they are not popular or well-rated.")

print("\nWeaknesses of Content-Based Filtering (Cosine Similarity):")
print("- Limited by the quality of metadata (genres, tags) available for each item.")
print("- May not capture complex user preferences or changes in user tastes over time.")
```

```
print("\nAreas for Improvement:")
print("- Hybrid approaches combining both collaborative and content-based filtering to leverage their strengths.")
print("- Improving metadata quality and diversity to enhance content-based recommendations.")
```

```
Enter a user ID to recommend movies (or 'exit' to quit): 2
Top 10 Recommendations for User 2:
1. Father of the Bride Part II (1995)
2. Postman, The (Postino, Il) (1994)
3. Juror, The (1996)
4. While You Were Sleeping (1995)
5. Primal Fear (1996)

Precision: 0.67
Recall: 1.00
F1-score: 0.80

Enter a user ID to recommend movies (or 'exit' to quit): 1
Top 10 Recommendations for User 1:
1. Godfather, The (1972)
2. Dead Poets Society (1989)

Precision: 1.00
Recall: 1.00
F1-score: 1.00

Enter a user ID to recommend movies (or 'exit' to quit): 5
Top 10 Recommendations for User 5:
1. One Flew Over the Cuckoo's Nest (1975)
2. Good, the Bad and the Ugly, The (Buono, il brutto, il cattivo, Il) (1966)
3. Annie Hall (1977)
4. Treasure of the Sierra Madre, The (1948)

Precision: 1.00
Recall: 1.00
F1-score: 1.00

Enter a user ID to recommend movies (or 'exit' to quit): 3
Top 10 Recommendations for User 3:
1. Mrs. Doubtfire (1993)
2. Basic Instinct (1992)

Precision: 0.33
Recall: 1.00
F1-score: 0.50

Enter a user ID to recommend movies (or 'exit' to quit): 6
Top 10 Recommendations for User 6:
1. American President, The (1995)

Precision: 0.50
Recall: 1.00
F1-score: 0.67

Enter a user ID to recommend movies (or 'exit' to quit): 4
Top 10 Recommendations for User 4:
1. Princess Bride, The (1987)
2. Star Wars: Episode VI - Return of the Jedi (1983)

Precision: 0.80
Recall: 1.00
F1-score: 0.89
```

```
# Install surprise if not already installed
!pip install scikit-surprise

import pandas as pd
import re
import numpy as np
import ipywidgets as widgets
from IPython.display import display
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from surprise import Dataset, Reader, SVD, accuracy
from surprise.model_selection import train_test_split, cross_validate
from sklearn.metrics import precision_score, recall_score, f1_score

# Load the datasets
ratings_path = '/content/ratings.csv'
movies_path = '/content/movies.csv'
tags_path = '/content/tags.csv'
links_path = '/content/links.csv'
```

```
ratings = pd.read_csv(ratings_path)
movies = pd.read_csv(movies_path)
tags = pd.read_csv(tags_path)
links = pd.read_csv(links_path)

# Print the first five rows of each dataset
print("First five rows of ratings dataset:")
print(ratings.head())

print("\nFirst five rows of movies dataset:")
print(movies.head())

print("\nFirst five rows of tags dataset:")
print(tags.head())

print("\nFirst five rows of links dataset:")
print(links.head())

# Merge datasets
data = pd.merge(ratings, movies, on='movieId')
data = pd.merge(data, tags, on=['userId', 'movieId'], how='left')
data = pd.merge(data, links, on='movieId', how='left')
data['tag'] = data['tag'].fillna('')

# Combine genres and tags for content-based filtering
data['genres'] = data['genres'].str.split('|')
data['combined_features'] = data['genres'] + data['tag'].apply(lambda x: [x])

# One-hot encode the combined features
from sklearn.preprocessing import MultiLabelBinarizer
mlb = MultiLabelBinarizer()
features_encoded = mlb.fit_transform(data['combined_features'])

# Create a DataFrame with the encoded features
features_df = pd.DataFrame(features_encoded, columns=mlb.classes_)
data = pd.concat([data, features_df], axis=1)
data.drop(['genres', 'combined_features', 'tag'], axis=1, inplace=True)

# Function to clean movie titles
def clean_title(title):
    return re.sub("[^a-zA-Z0-9]", "", title)

# Apply the cleaning function to movie titles
movies["clean_title"] = movies["title"].apply(clean_title)

# Create the TF-IDF vectorizer and transform the cleaned titles
vectorizer = TfidfVectorizer(ngram_range=(1, 2))
tfidf = vectorizer.fit_transform(movies["clean_title"])

# Define the search function
def search(title):
    title = clean_title(title)
    query_vec = vectorizer.transform([title])
    similarity = cosine_similarity(query_vec, tfidf).flatten()
    indices = np.argpartition(similarity, -5)[-5:]
    results = movies.iloc[indices][::-1]
    return results

# Set up interactive widget for searching movies
movie_input = widgets.Text(value="Toy Story", description="Movie Title:", disabled=False)
movie_list = widgets.Output()

def on_type(data):
    with movie_list:
        movie_list.clear_output()
        title = data["new"]
        if len(title) > 5:
            display(search(title))

movie_input.observe(on_type, names='value')
display(movie_input, movie_list)

# Load the data into surprise's Dataset format
reader = Reader(rating_scale=(0.5, 5.0))
data_surprise = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)

# Split the data into train and test sets
trainset, testset = train_test_split(data_surprise, test_size=0.25)

# Collaborative Filtering using SVD
svd = SVD()
svd.fit(trainset)
```

```
# Make predictions on the test set
predictions_svd = svd.test(testset)

# Evaluate SVD model
rmse_svd = accuracy.rmse(predictions_svd)

# Content-Based Filtering using Cosine Similarity
cosine_sim = cosine_similarity(features_encoded, features_encoded)

# Function to get movie recommendations based on content
def get_content_based_recommendations(movie_title, cosine_sim=cosine_sim):
    movie_idx = data[data['title'] == movie_title].index[0]
    sim_scores = list(enumerate(cosine_sim[movie_idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:11]
    movie_indices = [i[0] for i in sim_scores]
    return data['title'].iloc[movie_indices].unique()

# Example usage of content-based filtering
print(get_content_based_recommendations('Toy Story (1995)'))

# Precision, recall, and F1-score for SVD collaborative filtering
y_true_svd = [pred.r_ui for pred in predictions_svd]
y_pred_svd = [pred.est for pred in predictions_svd]

threshold = 3.5
y_true_binary_svd = [1 if rating >= threshold else 0 for rating in y_true_svd]
y_pred_binary_svd = [1 if rating >= threshold else 0 for rating in y_pred_svd]

precision_svd = precision_score(y_true_binary_svd, y_pred_binary_svd, average='binary')
recall_svd = recall_score(y_true_binary_svd, y_pred_binary_svd, average='binary')
f1_svd = f1_score(y_true_binary_svd, y_pred_binary_svd, average='binary')

print(f"SVD - RMSE: {rmse_svd}, Precision: {precision_svd}, Recall: {recall_svd}, F1-score: {f1_svd}")

# Cross-validate the SVD model
cv_results_svd = cross_validate(svd, data_surprise, measures=['RMSE', 'MAE'], cv=3, verbose=True)

# Print cross-validation results for SVD
def print_cv_results(cv_results):
    rmse_scores = cv_results['test_rmse']
    mae_scores = cv_results['test_mae']
    print(f'Mean RMSE: {np.mean(rmse_scores)}')
    print(f'Mean MAE: {np.mean(mae_scores)}')

print('\nCollaborative Filtering - SVD Cross-Validation Results:')
print_cv_results(cv_results_svd)

# Compare SVD with Content-Based Filtering
print('\nComparison of Collaborative Filtering (SVD) and Content-Based Filtering:')
print(f'SVD - RMSE: {rmse_svd}, Precision: {precision_svd}, Recall: {recall_svd}, F1-score: {f1_svd}')
print('Content-Based Filtering Recommendations for "Toy Story (1995)":')
print(get_content_based_recommendations('Toy Story (1995)'))

# Visualization part for comparison
# Plotting the distribution of true ratings vs. predicted ratings for SVD
plt.figure(figsize=(10, 5))
sns.histplot(y_true_svd, color='blue', label='True Ratings (SVD)', kde=True)
sns.histplot(y_pred_svd, color='red', label='Predicted Ratings (SVD)', kde=True)
plt.legend()
plt.xlabel('Ratings')
plt.title('Distribution of True Ratings vs. Predicted Ratings (SVD)')
plt.show()

# Precision, Recall, F1-score visualization for SVD
metrics_svd = {'Precision': precision_svd, 'Recall': recall_svd, 'F1-Score': f1_svd}
metric_names_svd = list(metrics_svd.keys())
metric_values_svd = list(metrics_svd.values())

plt.figure(figsize=(10, 5))
plt.bar(metric_names_svd, metric_values_svd, color=['blue', 'orange', 'green'])
plt.xlabel('Metrics')
plt.ylabel('Scores')
plt.title('Precision, Recall, and F1-Score for Collaborative Filtering (SVD)')
plt.show()

# Analysis and comparison of different algorithms
print("\nAnalysis and Comparison:")
print("Strengths of Collaborative Filtering (SVD):")
print("- Effective in capturing user preferences and recommending items based on user-item interactions.")
print("- Can handle large datasets and provide personalized recommendations.")
```

```
print("\nWeaknesses of Collaborative Filtering (SVD):")
print("- Cold-start problem: Requires sufficient user-item interactions to make accurate predictions.")
print("- May suffer from sparsity in data, especially for new or niche items.")

print("\nStrengths of Content-Based Filtering (Cosine Similarity):")
print("- Does not rely on user-item interactions, making it suitable for new users or items (cold-start).")
print("- Can recommend items based on content similarity, even if they are not popular or well-rated.")

print("\nWeaknesses of Content-Based Filtering (Cosine Similarity):")
print("- Limited by the quality of metadata (genres, tags) available for each item.")
print("- May not capture complex user preferences or changes in user tastes over time.")

print("\nAreas for Improvement:")
print("- Hybrid approaches combining both collaborative and content-based filtering to leverage their strengths.")
print("- Improving metadata quality and diversity to enhance content-based recommendations.")
```

```
Requirement already satisfied: scikit-surprise in /usr/local/lib/python3.10/dist-packages (1.1.4)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise) (1.4.2)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise) (1.25.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise) (1.11.4)
First five rows of ratings dataset:
   userId  movieId  rating   timestamp
0       1      110     1.0  1425941529
1       1      147     4.5  1425942435
2       1      858     5.0  1425941523
3       1     1221     5.0  1425941546
4       1     1246     5.0  1425941556

First five rows of movies dataset:
   movieId                               title  \
0        1                    Toy Story (1995)
1        2                      Jumanji (1995)
2        3             Grumpier Old Men (1995)
3        4            Waiting to Exhale (1995)
4        5  Father of the Bride Part II (1995)

                                        genres
0  Adventure|Animation|Children|Comedy|Fantasy
1                   Adventure|Children|Fantasy
2                               Comedy|Romance
3                         Comedy|Drama|Romance
4                                       Comedy

First five rows of tags dataset:
   userId  movieId              tag   timestamp
0       2    60756            funny  1445714994
1       2    60756  Highly quotable  1445714996
2       2    60756     will ferrell  1445714992
3       2    89774     Boxing story  1445715207
4       2    89774              MMA  1445715200

First five rows of links dataset:
   movieId  imdbId   tmdbId
0        1  114709    862.0
1        2  113497   8844.0
2        3  113228  15602.0
3        4  114885  31357.0
4        5  113041  11862.0
```

Movie Title: Toy Story

```
RMSE: 1.0949
['Toy Story (1995)' 'Aladdin (1992)'
 'Grand Day Out with Wallace and Gromit, A (1989)'
 'Monty Python and the Holy Grail (1975)'
 'Wallace & Gromit: A Close Shave (1995)'
 'Willy Wonka & the Chocolate Factory (1971)'
 'Wallace & Gromit: The Wrong Trousers (1993)'
 'Muppet Treasure Island (1996)']
SVD - RMSE: 1.0948953881794428, Precision: 0.6594202898550725, Recall: 0.610738255033557, F1-score: 0.6341463414634145
Evaluating RMSE, MAE of algorithm SVD on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    1.1019  0.9499  1.0478  1.0332  0.0629
```

```
!pip install scikit-surprise

import pandas as pd
from surprise import Dataset, Reader, SVD
from surprise.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns

# Load datasets
ratings = pd.read_csv('/content/ratings.csv')
```

```
movies = pd.read_csv('/content/movies.csv')
tags = pd.read_csv('/content/tags.csv')
links = pd.read_csv('/content/links.csv')

# Display first five rows of ratings dataset
print("First five rows of ratings dataset:")
print(ratings.head())

# Merge datasets
data = pd.merge(ratings, movies, on='movieId')
data = pd.merge(data, tags, on=['userId', 'movieId'], how='left')
data = pd.merge(data, links, on='movieId', how='left')
data['tag'] = data['tag'].fillna('')

# Define a function to simulate ground truth (liked movies)
def simulate_ground_truth(ratings, threshold=3.5):
    liked_movies = ratings[ratings['rating'] >= threshold]['movieId'].unique()
    return liked_movies

# Load data into Surprise Dataset format
reader = Reader(rating_scale=(1, 5))
data_surprise = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)

# Split the data into train and test sets
trainset, testset = train_test_split(data_surprise, test_size=0.25)

# Initialize the SVD algorithm
svd = SVD()

# Fit the algorithm on the train set
svd.fit(trainset)

# Make predictions on the test set
predictions = svd.test(testset)

def get_top_n(predictions, n=10):
    # First map the predictions to each user.
    top_n = {}
    for uid, iid, true_r, est, _ in predictions:
        if uid not in top_n:
            top_n[uid] = []
        top_n[uid].append((iid, est))

    # Then sort the predictions for each user and retrieve the n highest ones.
    for uid, user_ratings in top_n.items():
        user_ratings.sort(key=lambda x: x[1], reverse=True)
        top_n[uid] = user_ratings[:n]

    return top_n

# Interactive script to accept user input and display recommendations
def main():
    liked_movies = simulate_ground_truth(ratings, threshold=3.5)
    user_input = input("Enter a user ID to recommend movies (or 'exit' to quit): ")

    while user_input.lower() != 'exit':
        try:
            user_id = int(user_input)
            if user_id not in ratings['userId'].unique():
                print(f"User {user_id} not found in the dataset.")
            else:
                recommendations = get_top_n(predictions, n=10).get(user_id, [])
                if not recommendations:
                    print(f"No recommendations found for user {user_id}.")
                else:
                    print(f"Top 10 Recommendations for User {user_id}:")
                    movie_ids = [iid for (iid, _) in recommendations]
                    movie_names = movies[movies['movieId'].isin(movie_ids)]['title'].tolist()
                    for i, movie in enumerate(movie_names, 1):
                        print(f"{i}. {movie}")

                    # Evaluate precision, recall, and F1-score
                    relevant_recommended = [1 if iid in liked_movies else 0 for iid in movie_ids]
                    precision = precision_score(relevant_recommended, [1]*len(recommendations), zero_division=0)
                    recall = recall_score(relevant_recommended, [1]*len(recommendations), zero_division=0)
                    f1 = f1_score(relevant_recommended, [1]*len(recommendations), zero_division=0)

                    print(f"\nPrecision: {precision:.2f}")
                    print(f"Recall: {recall:.2f}")
                    print(f"F1-score: {f1:.2f}")

        except ValueError:
```

```
            print("Invalid input. Please enter a valid user ID or 'exit' to quit.")

        user_input = input("\nEnter a user ID to recommend movies (or 'exit' to quit): ")

if __name__ == "__main__":
    main()

# Placeholder variables (replace these with actual computation of RMSE, precision, recall, and F1-score for SVD)
rmse_svd = 0.9  # Example RMSE value
precision_svd = 0.75  # Example precision value
recall_svd = 0.65  # Example recall value
f1_svd = 0.7  # Example F1-score value

print('\nComparison of Collaborative Filtering (SVD) and Content-Based Filtering:')
print(f'SVD - RMSE: {rmse_svd}, Precision: {precision_svd}, Recall: {recall_svd}, F1-score: {f1_svd}')
print('Content-Based Filtering Recommendations for "Toy Story (1995)":')
print(get_content_based_recommendations('Toy Story (1995)'))

# Visualization part for comparison
# Plotting the distribution of true ratings vs. predicted ratings for SVD
plt.figure(figsize=(10, 5))
sns.histplot([pred.r_ui for pred in predictions], color='blue', label='True Ratings (SVD)', kde=True)
sns.histplot([pred.est for pred in predictions], color='red', label='Predicted Ratings (SVD)', kde=True)
plt.legend()
plt.xlabel('Ratings')
plt.title('Distribution of True Ratings vs. Predicted Ratings (SVD)')
plt.show()

# Precision, Recall, F1-score visualization for SVD
metrics_svd = {'Precision': precision_svd, 'Recall': recall_svd, 'F1-Score': f1_svd}
metric_names_svd = list(metrics_svd.keys())
metric_values_svd = list(metrics_svd.values())

plt.figure(figsize=(10, 5))
plt.bar(metric_names_svd, metric_values_svd, color=['blue', 'orange', 'green'])
plt.xlabel('Metrics')
plt.ylabel('Scores')
plt.title('Precision, Recall, and F1-Score for Collaborative Filtering (SVD)')
plt.show()

# Analysis and comparison of different algorithms
print("\nAnalysis and Comparison:")
print("Strengths of Collaborative Filtering (SVD):")
print("- Effective in capturing user preferences and recommending items based on user-item interactions.")
print("- Can handle large datasets and provide personalized recommendations.")

print("\nWeaknesses of Collaborative Filtering (SVD):")
print("- Cold-start problem: Requires sufficient user-item interactions to make accurate predictions.")
print("- May suffer from sparsity in data, especially for new or niche items.")

print("\nStrengths of Content-Based Filtering (Cosine Similarity):")
print("- Does not rely on user-item interactions, making it suitable for new users or items (cold-start).")
print("- Can recommend items based on content similarity, even if they are not popular or well-rated.")

print("\nWeaknesses of Content-Based Filtering (Cosine Similarity):")
print("- Limited by the quality of metadata (genres, tags) available for each item.")
print("- May not capture complex user preferences or changes in user tastes over time.")

print("\nAreas for Improvement:")
print("- Hybrid approaches combining both collaborative and content-based filtering to leverage their strengths.")
print("- Improving metadata quality and diversity to enhance content-based recommendations.")
```

```
Collecting scikit-surprise
  Downloading scikit_surprise-1.1.4.tar.gz (154 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 154.4/154.4 kB 2.7 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise) (1.4.2)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise) (1.25.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise) (1.11.4)
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (pyproject.toml) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.4-cp310-cp310-linux_x86_64.whl size=2357259 sha256=5f60d35
  Stored in directory: /root/.cache/pip/wheels/4b/3f/df/6acbf0a40397d9bf3ff97f582cc22fb9ce66adde75bc71fd54
Successfully built scikit-surprise
Installing collected packages: scikit-surprise
Successfully installed scikit-surprise-1.1.4
First five rows of ratings dataset:
   userId  movieId  rating   timestamp
0       1      110     1.0  1425941529
1       1      147     4.5  1425942435
2       1      858     5.0  1425941523
3       1     1221     5.0  1425941546
4       1     1246     5.0  1425941556
Enter a user ID to recommend movies (or 'exit' to quit): 1
Top 10 Recommendations for User 1:
1. Dead Poets Society (1989)

Precision: 1.00
Recall: 1.00
F1-score: 1.00

Enter a user ID to recommend movies (or 'exit' to quit): 2
Top 10 Recommendations for User 2:
1. Twelve Monkeys (a.k.a. 12 Monkeys) (1995)
2. Juror, The (1996)
3. Independence Day (a.k.a. ID4) (1996)

Precision: 0.75
Recall: 1.00
F1-score: 0.86

Enter a user ID to recommend movies (or 'exit' to quit): 3
Top 10 Recommendations for User 3:
1. Mrs. Doubtfire (1993)
2. E.T. the Extra-Terrestrial (1982)

Precision: 0.67
Recall: 1.00
F1-score: 0.80

Enter a user ID to recommend movies (or 'exit' to quit): 4
Top 10 Recommendations for User 4:
1. Godfather: Part II, The (1974)

Precision: 0.40
Recall: 1.00
F1-score: 0.57
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
SVD - RMSE: 0.9, Precision: 0.75, Recall: 0.65, F1-score: 0.7
```

```
!pip install scikit-surprise

import pandas as pd
from surprise import Dataset, Reader, SVD
from surprise.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import CountVectorizer

# Load datasets
ratings = pd.read_csv('/content/ratings.csv')
movies = pd.read_csv('/content/movies.csv')
tags = pd.read_csv('/content/tags.csv')
links = pd.read_csv('/content/links.csv')

# Display first five rows of ratings dataset
print("First five rows of ratings dataset:")
print(ratings.head())

print("\nFirst five rows of movies dataset:")
print(movies.head())

print("\nFirst five rows of tags dataset:")
```

```
print(tags.head())

print("\nFirst five rows of links dataset:")
print(links.head())


# Merge datasets
data = pd.merge(ratings, movies, on='movieId')
data = pd.merge(data, tags, on=['userId', 'movieId'], how='left')
data = pd.merge(data, links, on='movieId', how='left')
data['tag'] = data['tag'].fillna('')

# Define a function to simulate ground truth (liked movies)
def simulate_ground_truth(ratings, threshold=3.5):
    liked_movies = ratings[ratings['rating'] >= threshold]['movieId'].unique()
    return liked_movies

# Load data into Surprise Dataset format
reader = Reader(rating_scale=(1, 5))
data_surprise = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)

# Split the data into train and test sets
trainset, testset = train_test_split(data_surprise, test_size=0.25)

# Initialize the SVD algorithm
svd = SVD()

# Fit the algorithm on the train set
svd.fit(trainset)

# Make predictions on the test set
predictions = svd.test(testset)

def get_top_n(predictions, n=10):
    # First map the predictions to each user.
    top_n = {}
    for uid, iid, true_r, est, _ in predictions:
        if uid not in top_n:
            top_n[uid] = []
        top_n[uid].append((iid, est))

    # Then sort the predictions for each user and retrieve the n highest ones.
    for uid, user_ratings in top_n.items():
        user_ratings.sort(key=lambda x: x[1], reverse=True)
        top_n[uid] = user_ratings[:n]

    return top_n

# Define the content-based recommendation function
def get_content_based_recommendations(title, movies, n=10):
    # Create a CountVectorizer to convert the genres to a matrix of token counts
    count_vectorizer = CountVectorizer(tokenizer=lambda x: x.split('|'))
    genre_matrix = count_vectorizer.fit_transform(movies['genres'])

    # Compute the cosine similarity matrix
    cosine_sim = cosine_similarity(genre_matrix, genre_matrix)

    # Get the index of the movie that matches the title
    idx = movies[movies['title'] == title].index[0]

    # Get the pairwise similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

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

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:n+1]

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

    # Return the top n most similar movies
    return movies['title'].iloc[movie_indices].tolist()

# Interactive script to accept user input and display recommendations
def main():
    liked_movies = simulate_ground_truth(ratings, threshold=3.5)
    user_input = input("Enter a user ID to recommend movies (or 'exit' to quit): ")

    while user_input.lower() != 'exit':
        try:
```

```
                user_id = int(user_input)
                if user_id not in ratings['userId'].unique():
                    print(f"User {user_id} not found in the dataset.")
                else:
                    recommendations = get_top_n(predictions, n=10).get(user_id, [])
                    if not recommendations:
                        print(f"No recommendations found for user {user_id}.")
                    else:
                        print(f"Top 10 Recommendations for User {user_id}:")
                        movie_ids = [iid for (iid, _) in recommendations]
                        movie_names = movies[movies['movieId'].isin(movie_ids)]['title'].tolist()
                        for i, movie in enumerate(movie_names, 1):
                            print(f"{i}. {movie}")

                        # Evaluate precision, recall, and F1-score
                        relevant_recommended = [1 if iid in liked_movies else 0 for iid in movie_ids]
                        precision = precision_score(relevant_recommended, [1]*len(recommendations), zero_division=0)
                        recall = recall_score(relevant_recommended, [1]*len(recommendations), zero_division=0)
                        f1 = f1_score(relevant_recommended, [1]*len(recommendations), zero_division=0)

                        print(f"\nPrecision: {precision:.2f}")
                        print(f"Recall: {recall:.2f}")
                        print(f"F1-score: {f1:.2f}")

            except ValueError:
                print("Invalid input. Please enter a valid user ID or 'exit' to quit.")

            user_input = input("\nEnter a user ID to recommend movies (or 'exit' to quit): ")

    # Placeholder variables (replace these with actual computation of RMSE, precision, recall, and F1-score for SVD)
    rmse_svd = 0.9  # Example RMSE value
    precision_svd = 0.75  # Example precision value
    recall_svd = 0.65  # Example recall value
    f1_svd = 0.7  # Example F1-score value

    print('\nComparison of Collaborative Filtering (SVD) and Content-Based Filtering:')
    print(f'SVD - RMSE: {rmse_svd}, Precision: {precision_svd}, Recall: {recall_svd}, F1-score: {f1_svd}')
    print('Content-Based Filtering Recommendations for "Toy Story (1995)":')
    print(get_content_based_recommendations('Toy Story (1995)', movies))

    # Visualization part for comparison
    # Plotting the distribution of true ratings vs. predicted ratings for SVD
    plt.figure(figsize=(10, 5))
    sns.histplot([pred.r_ui for pred in predictions], color='blue', label='True Ratings (SVD)', kde=True)
    sns.histplot([pred.est for pred in predictions], color='red', label='Predicted Ratings (SVD)', kde=True)
    plt.legend()
    plt.xlabel('Ratings')
    plt.title('Distribution of True Ratings vs. Predicted Ratings (SVD)')
    plt.show()

    # Precision, Recall, F1-score visualization for SVD
    metrics_svd = {'Precision': precision_svd, 'Recall': recall_svd, 'F1-Score': f1_svd}
    metric_names_svd = list(metrics_svd.keys())
    metric_values_svd = list(metrics_svd.values())

    plt.figure(figsize=(10, 5))
    plt.bar(metric_names_svd, metric_values_svd, color=['blue', 'orange', 'green'])
    plt.xlabel('Metrics')
    plt.ylabel('Scores')
    plt.title('Precision, Recall, and F1-Score for Collaborative Filtering (SVD)')
    plt.show()

    if __name__ == "__main__":
        main()

    # Analysis and comparison of different algorithms
    print("\nAnalysis and Comparison:")
    print("Strengths of Collaborative Filtering (SVD):")
    print("- Effective in capturing user preferences and recommending items based on user-item interactions.")
    print("- Can handle large datasets and provide personalized recommendations.")

    print("\nWeaknesses of Collaborative Filtering (SVD):")
    print("- Cold-start problem: Requires sufficient user-item interactions to make accurate predictions.")
    print("- May suffer from sparsity in data, especially for new or niche items.")

    print("\nStrengths of Content-Based Filtering (Cosine Similarity):")
    print("- Does not rely on user-item interactions, making it suitable for new users or items (cold-start).")
    print("- Can recommend items based on content similarity, even if they are not popular or well-rated.")

    print("\nWeaknesses of Content-Based Filtering (Cosine Similarity):")
    print("- Limited by the quality of metadata (genres, tags) available for each item.")
    print("- May not capture complex user preferences or changes in user tastes over time.")
```

```
print("\nAreas for Improvement:")
print("- Hybrid approaches combining both collaborative and content-based filtering to leverage their strengths.")
print("- Improving metadata quality and diversity to enhance content-based recommendations.")
```

```
Collecting scikit-surprise
  Downloading scikit_surprise-1.1.4.tar.gz (154 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 154.4/154.4 kB 5.4 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-surprise) (1.5.2)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.12/dist-packages (from scikit-surprise) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-surprise) (1.16.3)
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (pyproject.toml) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.4-cp312-cp312-linux_x86_64.whl size=2544618 sha256=fee89ba
  Stored in directory: /root/.cache/pip/wheels/75/fa/bc/739bc2cb1fbaab6061854e6cfbb81a0ae52c92a502a7fa454b
Successfully built scikit-surprise
Installing collected packages: scikit-surprise
Successfully installed scikit-surprise-1.1.4

A module that was compiled using NumPy 1.x cannot be run in
NumPy 2.0.2 as it may crash. To support both 1.x and 2.x
versions of NumPy, modules must be compiled with NumPy 2.0.
Some module may need to rebuild instead e.g. with 'pybind11>=2.12'.

If you are a user of the module, the easiest solution will be to
downgrade to 'numpy<2' or try to upgrade the affected module.
We expect that some modules will need time to support NumPy 2.

Traceback (most recent call last):  File "<frozen runpy>", line 198, in _run_module_as_main
  File "<frozen runpy>", line 88, in _run_code
  File "/usr/local/lib/python3.12/dist-packages/colab_kernel_launcher.py", line 37, in <module>
    ColabKernelApp.launch_instance()
  File "/usr/local/lib/python3.12/dist-packages/traitlets/config/application.py", line 992, in launch_instance
    app.start()
  File "/usr/local/lib/python3.12/dist-packages/ipykernel/kernelapp.py", line 712, in start
```