EX NO:1 DATE:	IMPLEMENT DATA SIMILARITY MEASURES USING PYTHON

AIM:

To implement Data Similarity Measures using python using Cosine Similarity.

ALGORITHM:

Step 1: Start the program.

Step 2: Read the dataset containing Spotify song information.

Step 3: Preprocess the data by dropping unnecessary columns.

Step 4: Sample the data to ensure genre representation.

Step 5: Normalize the data for cosine similarity calculation.

Step 6: Compute cosine similarity matrix

Step 7: Define function to recommend songs based on userinput.

Step 8: Take user's playlist input

Step 9: Calculate average similarity scores for input songs.

Step 10: Recommend top 10 songs based on similarity scores.

Step 11: End the program.

PROGRAM:

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

# Load the data
data = pd.read_csv("/content/dataset.csv")

# Drop irrelevant columns
data.drop(columns=["explicit", "time_signature", "duration_ms"], inplace=True)
```

```
# Remove the first column
data.drop(data.columns[0], axis=1, inplace=True)

# Group by track genre and sample songs
sampled_data = data.groupby("track_genre", group_keys=False).apply(lambda x: x.sample(min(len(x), 200)))
sampled_data.reset_index(drop=True, inplace=True)

# Calculate cosine similarity
cosine_data = sampled_data.drop(["track_id", "artists", "album_name", "track_name", "track_genre"], axis=1)
cosine_data_normalized = (cosine_data - cosine_data.mean()) / cosine_data.std()
cosine_similarity_matrix = cosine_similarity(cosine_data_normalized)

# Function to recommend songs
def recommend_songs(input_songs):
    input_song_indices = []

    for name in input_songs:
        song_indic    if not input_song_indices:
        return []

es = sampled_data.index[sampled_data["track_name"] == name].tolist()
        if song_indices:
            input_song_indices.extend(song_indices)
        else:
            print(f"Song '{name}' not found in the dataset.")
```

```python
    average_similarity_scores = np.mean(cosine_similarity_matrix[input_song_indices], axis=0)
    sorted_indices = np.argsort(average_similarity_scores)[::-1]
    recommended_indices = [idx for idx in sorted_indices if idx not in input_song_indices]
    top_10_recommendations = recommended_indices[:10]
    recommended_song_names = [sampled_data.loc[idx, "track_name"] for idx in top_10_recommendations]
    return recommended_song_names

# Get user input for five songs
input_songs = []
print("Enter the names of 5 songs from your playlist (one song per line):")

for _ in range(5):
    song = input("Enter song name: ").strip()
    input_songs.append(song)

# Print input songs
print("\nInput Songs:")
print("\n".join(input_songs))

# Get recommendations
recommended_songs = recommend_songs(input_songs)

# Print recommended songs
print("\nTop-10 Recommended Songs:")
for song in recommended_songs:
    print(song)
```

OUTPUT:

Enter the names of 5 songs from your playlist (one song per line):

Enter song name: Comedy

Enter song name: Hunger

Enter song name: Ontario

Enter song name: Starboy

Enter song name: Night Changes

Input Songs:

Comedy

Hunger

Ontario

Starboy

Night Changes

Song 'Ontario' not found in the dataset.

Song 'Night Changes' not found in the dataset.

Top-10 Recommended Songs:

Mount Everest

Mount Everest

2002

Famous

Play with Fire (feat. Yacht Money)

Over 85

Walls Could Talk

Walls Could Talk

Dreamin (with blackbear)

Sorry

RESULT:

Thus, Data similarity measures have been implemented in Python and the output has been verified successfully.

EX NO:2 DATE:	**IMPLEMENT DIMENSION REDUCTION TECHNIQUES FOR RECOMMENDER SYSTEMS**

AIM:

To implement Dimension Reduction Techniques for recommender systems using Python with PCA.

ALGORITHM:

Step 1: Start the program.

Step 2: Load the image and split it into its red, green, and blue channels.

Step 3: Normalize the pixel values by dividing them by 255 to bring them into the range [0, 1].

Step 4: Apply PCA separately to each normalized channel, choosing the number of components desired for dimensionality reduction.

Step 5: Transform each channel using the PCA components to reduce its dimensionality.

Step 6: Inverse transform the transformed channels to reconstruct the pixel values.

Step 7: Merge the reconstructed channels back together to form the reduced image.

Step 8: Display both the original and reduced images for comparison.

Step 9: Analyze the explained variance ratio to understand how much information is retained after dimensionality reduction.

Step 10: Plot the variation explained by each principal component to visualize their contribution to the total variance.

Step 11: End the program.

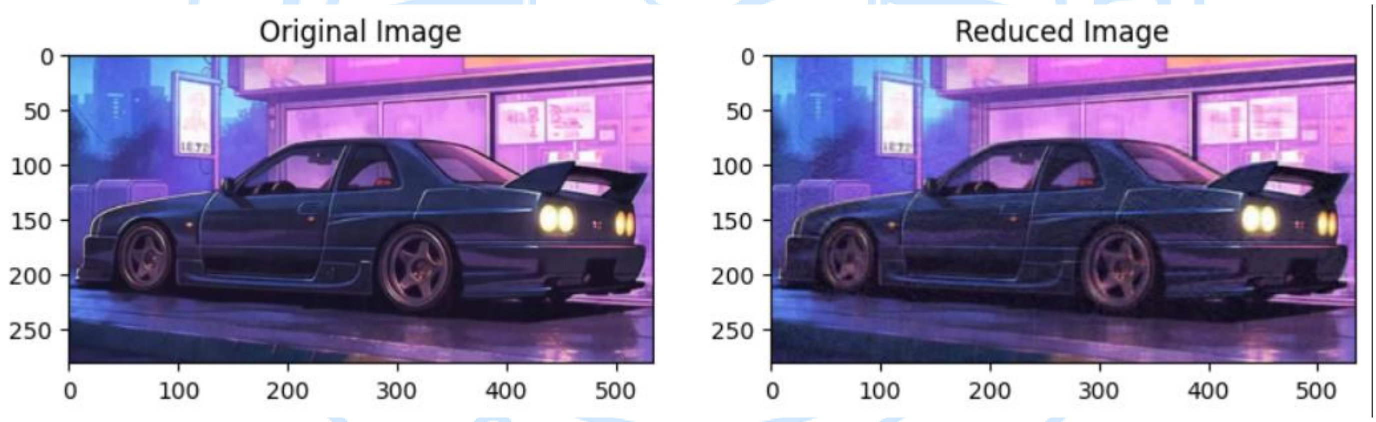
PROGRAM:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt
```

```
from sklearn.decomposition import PCA
import cv2
img = cv2.cvtColor(cv2.imread('/content/Screenshot 2024-08-14 121607.png'), cv2.COLOR_BGR2RGB)
blue, green, red = cv2.split(img)
blue_temp_df = pd.DataFrame(data=blue)
df_blue = blue / 255
green_temp_df = pd.DataFrame(data=green)
df_green = green / 255
red_temp_df = pd.DataFrame(data=red)
df_red = red / 255
pca_b = PCA(n_components=50)
pca_b.fit(df_blue)
trans_pca_b = pca_b.transform(df_blue)
pca_g = PCA(n_components=50)
pca_g.fit(df_green)
trans_pca_g = pca_g.transform(df_green)
pca_r = PCA(n_components=50)
pca_r.fit(df_red)
trans_pca_r = pca_r.transform(df_red)
b_arr = pca_b.inverse_transform(trans_pca_b)
g_arr = pca_g.inverse_transform(trans_pca_g)
r_arr = pca_r.inverse_transform(trans_pca_r)
```

```
img_reduced = cv2.merge((b_arr, g_arr, r_arr))

fig = plt.figure(figsize=(10, 7.2))

fig.add_subplot(121)

plt.title("Original Image")

plt.imshow(img)

fig.add_subplot(122)

plt.title("Reduced Image")

plt.imshow(img_reduced)

plt.show()
```

OUTPUT:



RESULT:

Thus the Dimension Reduction techniques have been implemented in Python and the output has been verified successfully.

EX NO:3 **DATE:**	**IMPLEMENT USER PROFILE LEARNING**

AIM:

To Implement User Profile Learning using Python.

ALGORITHM:

Step 1: Start the program.

Step 2: Load the MovieLens-100k dataset using Surprise and access the raw ratings data.

Step 3: Define column names for the raw ratings data.

Step 4: Display the first few rows of the raw ratings data with column names.

Step 5: Train a user-based collaborative filtering model with the KNNBasic algorithm using Surprise.

Step 6: Define a function to make predictions for a specific user with custom ratings.

Step 7: Predict ratings for a specific user with custom ratings (e.g., user 7 with custom ratings for items).

Step 8: Display the predicted ratings for the user with custom ratings.

Step 9: End the program.

PROGRAM:

```
!pip install scikit-surprise
from surprise import Dataset, Reader, KNNBasic
data = Dataset.load_builtin('ml-100k')
raw_data = data.raw_ratings
columns = ['user_id', 'item_id', 'rating', 'timestamp']
for i in range(5):
    row = raw_data[i]
    print({columns[j]: row[j] for j in range(len(columns))})
model = KNNBasic(sim_options={'user_based': True})
trainset = data.build_full_trainset()
model.fit(trainset)
def predict_custom_ratings(user_id, model, custom_ratings):
    custom_data = [(user_id, item_id, rating) for item_id, rating in custom_ratings.items()]
    testset = trainset.build_testset() + custom_data
    predictions = model.test(testset)
    return predictions
user_id = 7
custom_ratings = {0: 4, 1: 2, 2: 5}
predictions = predict_custom_ratings(user_id, model, custom_ratings)


for i, prediction in enumerate(predictions[:15]):
    print(f"Item ID: {prediction.iid}, Estimated Rating: {prediction.est:.2f}")
```

OUTPUT:

```
First 5 rows of raw data:
{'user_id': '196', 'item_id': '242', 'rating': 3.0, 'timestamp': '881250949'}
{'user_id': '186', 'item_id': '302', 'rating': 3.0, 'timestamp': '891717742'}
{'user_id': '22', 'item_id': '377', 'rating': 1.0, 'timestamp': '878887116'}
{'user_id': '244', 'item_id': '51', 'rating': 2.0, 'timestamp': '880606923'}
{'user_id': '166', 'item_id': '346', 'rating': 1.0, 'timestamp': '886397596'}
Computing the msd similarity matrix...
Done computing similarity matrix.

Predicted ratings for user 7 with custom ratings:
Item ID: 242, Estimated Rating: 3.80
Item ID: 393, Estimated Rating: 3.45
Item ID: 381, Estimated Rating: 3.71
Item ID: 251, Estimated Rating: 4.20
Item ID: 655, Estimated Rating: 4.39
Item ID: 67, Estimated Rating: 3.60
Item ID: 306, Estimated Rating: 4.03
Item ID: 238, Estimated Rating: 4.12
Item ID: 663, Estimated Rating: 4.39
Item ID: 111, Estimated Rating: 3.83
Item ID: 580, Estimated Rating: 3.25
Item ID: 25, Estimated Rating: 4.00
Item ID: 286, Estimated Rating: 4.86
Item ID: 94, Estimated Rating: 3.11
Item ID: 692, Estimated Rating: 4.04
```


RESULT:

Thus User profile learning have been implemented using Python and the output has been verified successfully

EX NO:4 **DATE:**	**IMPLEMENT CONTENT-BASED RECOMMENDATION SYSTEM**

AIM:

To Implement Content-based recommendation system using Python

ALGORITHM:

Step 1: Import necessary libraries.

Step 2: Load the dataset into a DataFrame.

Step 3: Check for Null Values in each column and print the count.

Step 4: Perform Univariate analysis.

Step 5: Analyze the distribution of columns like 'category', 'sub_category', 'brand', etc.

Step 6: Create a feature space using CountVectorizer.

Step 7: Fit CountVectorizer to the 'description' column to create a feature space.

Step 8: Calculate the cosine similarity between feature vectors of each pair of descriptions.

Step 9: Define a function for recommendation.

Step 10: Implement recommendation function (assuming it's implemented elsewhere in the code)

PROGRAM:

```
import pandas as pd

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine_similarity

df = pd.read_csv('BigBasket Products.csv')

print(df.isnull().sum())

count = CountVectorizer(stop_words='english')

count_matrix = count.fit_transform(df['description'].fillna(''))
```

```
cosine_sim = cosine_similarity(count_matrix, count_matrix)

def recommend_products(product_name, cosine_sim=cosine_sim):

    idx = df[df['product'] == product_name].index[0]

    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

    product_indices = [i[0] for i in sim_scores]

    return df['product'].iloc[product_indices]

product_name = "Tea - Stress Relief"

recommendations = recommend_products(product_name)

print(recommendations)
```

OUTPUT:

```
8995                      Ayurvedic Ashwagandha Powder
1633                         Karpooradi Shower Cream
23146                           Organic Ashwagandha
18961       Get Slim Ayurvedic Tea - 7 Active Ingredients
20705    Slim Tea - For Detox & Weight-loss, Unflavoure...
25181                Slim Tea - For Detox & Weight-loss
18453    Tulsi Drops - 100% Ayurvedic Immunity Booster,...
23716       Ayurvedic Tea Masala - 21 Active Ingredients
26256                                     Tea - Green
12461    100% Organic Probiotic Digestive Enzymes For D...
Name: product, dtype: object
```

RESULT:

Thus, the content-based recommender system have been implemented in Python and the output has been verified successfully.

EX NO:5 **DATE:**	**IMPLEMENT COLLABORATIVE-FILTER TECHNIQUES**

AIM:

To Implement Collaborative-filter techniques using Python

ALGORITHM:

Step 1: Import necessary libraries.

Step 2: Import the dataset.

Step 3: Check the head of the data.

Step 4: Get all the movies and their IDs.

Step 5: Merge datasets to get complete information.

Step 6: Calculate mean rating of all movies.

Step 7: Calculate count rating of all movies.

Step 8: Create a data frame with 'rating' count values.

Step 9: Plot 'num of ratings' histogram.

Step 10: Plot 'ratings' histogram.

Step 11: Analyse correlation with similar movies.

Step 12: Display recommended movies based on correlation.

PROGRAM:

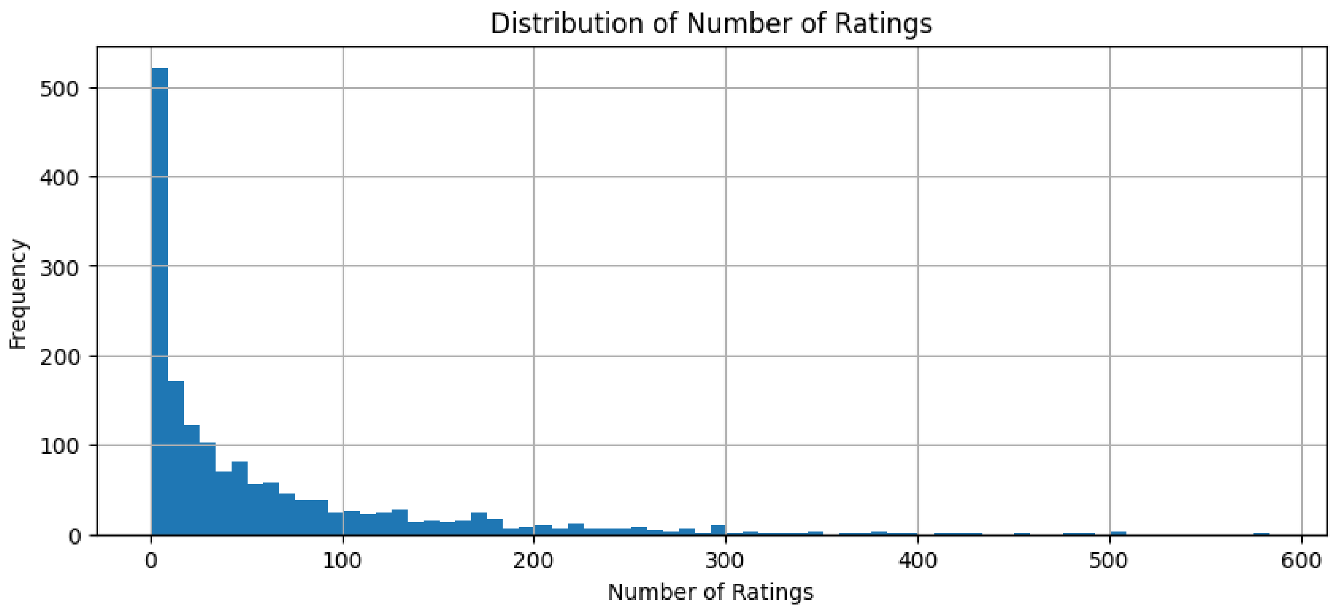
```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns



column_names = ['user_id', 'item_id', 'rating', 'timestamp']

path = 'https://media.geeksforgeeks.org/wp-content/uploads/file.tsv'
```

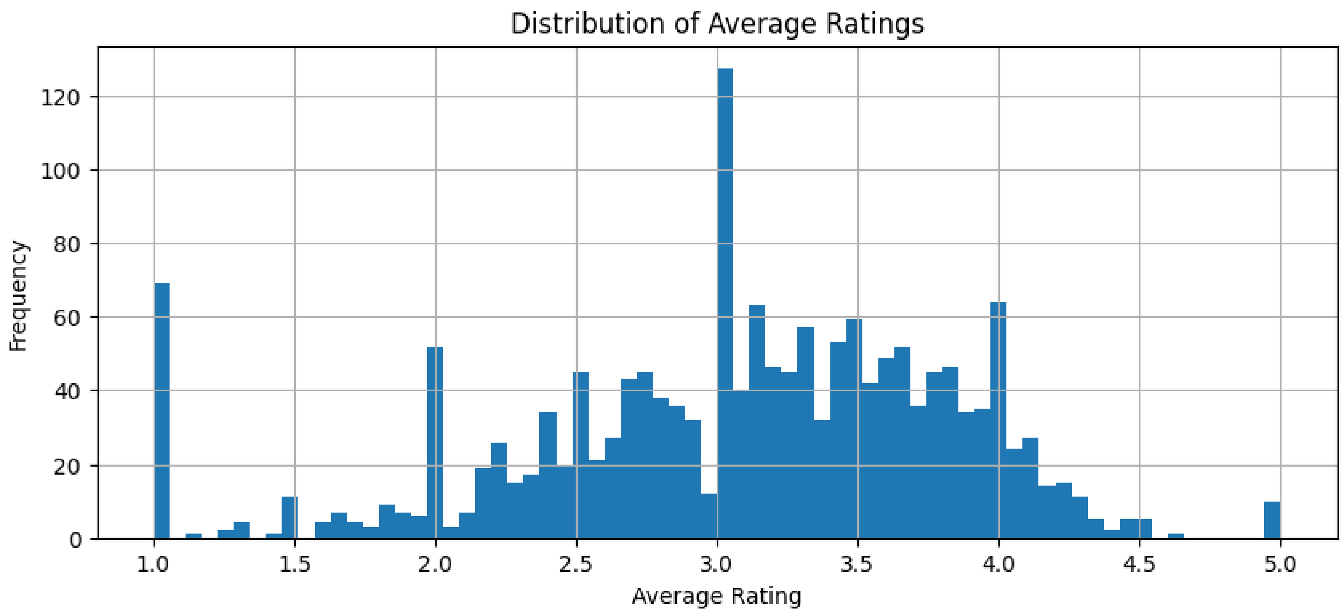
```
df = pd.read_csv(path, sep='\t', names=column_names)

print(df.head())

movie_titles_path = 'https://media.geeksforgeeks.org/wp-content/uploads/Movie_Id_Titles.csv'

movie_titles = pd.read_csv(movie_titles_path)

data = pd.merge(df, movie_titles, on='item_id')

average_ratings = data.groupby('title')['rating'].mean().sort_values(ascending=False)

rating_counts = data.groupby('title')['rating'].count().sort_values(ascending=False)

ratings = pd.DataFrame(average_ratings)

ratings['num of ratings'] = rating_counts

plt.figure(figsize=(10, 4))

ratings['num of ratings'].hist(bins=70)

plt.title('Distribution of Number of Ratings')

plt.xlabel('Number of Ratings')

plt.ylabel('Frequency')

plt.show()

plt.figure(figsize=(10, 4))

ratings['rating'].hist(bins=70)

plt.title('Distribution of Average Ratings')

plt.xlabel('Average Rating')

plt.ylabel('Frequency')

plt.show()
```

```
moviemat = data.pivot_table(index='user_id', columns='title', values='rating')

starwars_user_ratings = moviemat['Star Wars (1977)']

liarliar_user_ratings = moviemat['Liar Liar (1997)']

similar_to_starwars = moviemat.corrwith(starwars_user_ratings)

similar_to_liarliar = moviemat.corrwith(liarliar_user_ratings)

corr_starwars = pd.DataFrame(similar_to_starwars, columns=['Correlation'])

corr_starwars.dropna(inplace=True)


top_starwars_corr = corr_starwars.sort_values('Correlation', ascending=False).head(10)


print(top_starwars_corr)
```

OUTPUT:

	user_id	item_id	rating	timestamp
0	0	50	5	881250949
1	0	172	5	881250949
2	0	133	1	881250949
3	196	242	3	881250949
4	186	302	3	891717742






```
                                                    Correlation
title
Hollow Reed (1996)                                          1.0
Commandments (1997)                                         1.0
Cosi (1996)                                                 1.0
No Escape (1994)                                            1.0
Stripes (1981)                                              1.0
Star Wars (1977)                                            1.0
Man of the Year (1995)                                      1.0
Beans of Egypt, Maine, The (1994)                           1.0
Old Lady Who Walked in the Sea, The (Vieille qu...          1.0
Outlaw, The (1943)                                          1.0
```

RESULT:

Thus the Collaborative filter techniques have been implemented in Python and the output has been verified successfully

EX NO:6 DATE:	CREATE AN ATTACK FOR TAMPERING WITH RECOMMENDER SYSTEM

AIM:

To Create an Attack for tampering with Recommender System using Python.

ALGORITHM:

Step 1: Start the program.

Step 2: Import random for generating random numbers.

Step 3: Create placeholder functions create_user(profile), add_rating(user, item_id, rating), and get_random_item() to simulate user creation, rating addition, and random item selection.

Step 4: Initialize fake_users list and populate it with 10 users, each having a unique age, with similar gender and interests.

Step 5: Define target_item_id as the item ID to be promoted in the shilling attack.

Step 6: Loop through fake_users and rate target_item_id highly for each user to promote the item.

Step 7: For each user, rate three random items with moderate ratings to camouflage the biased ratings on the target item.

Step 8: End the Program.

PROGRAM:

```
import random

def create_user(profile):

        return {"profile": profile}

def add_rating(user, item_id, rating):

        print(f"User {user['profile']} rated item {item_id} with rating {rating}")
```

```
def get_random_item():
    return random.randint(1, 100000)
fake_users = []
for i in range(10):
    profile = {
        "age": 25 + i,
        "gender": "Male",
        "interests": ["Action movies", "Sci-Fi books"],
    }
    fake_users.append(create_user(profile))
target_item_id = 12345
print("Injecting biased ratings for the shilling attack:")
for user in fake_users:
    add_rating(user, target_item_id, 5)
print("\nAdding genuine ratings for camouflage:")
for user in fake_users:
    for _ in range(3):
        random_item_id = get_random_item()
        rating = random.randint(3, 5)
        add_rating(user, random_item_id, rating)
```

OUTPUT:

```
Injecting biased ratings for the shilling attack:
User {'age': 25, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 12345 with rating 5
User {'age': 26, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 12345 with rating 5
User {'age': 27, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 12345 with rating 5
User {'age': 28, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 12345 with rating 5
User {'age': 29, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 12345 with rating 5
User {'age': 30, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 12345 with rating 5
User {'age': 31, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 12345 with rating 5
User {'age': 32, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 12345 with rating 5
User {'age': 33, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 12345 with rating 5
User {'age': 34, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 12345 with rating 5
```

```
Adding genuine ratings for camouflage:
User {'age': 25, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 54569 with rating 5
User {'age': 25, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 33215 with rating 5
User {'age': 25, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 85234 with rating 5
User {'age': 26, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 2373 with rating 3
User {'age': 26, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 18874 with rating 3
User {'age': 26, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 3392 with rating 4
User {'age': 27, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 112 with rating 4
User {'age': 27, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 67635 with rating 5
User {'age': 27, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 33353 with rating 4
User {'age': 28, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 10374 with rating 3
User {'age': 28, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 17264 with rating 5
User {'age': 28, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 8824 with rating 4
User {'age': 29, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 49890 with rating 3
User {'age': 29, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 99207 with rating 3
User {'age': 29, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 62354 with rating 3
User {'age': 30, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 5569 with rating 5
User {'age': 30, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 56720 with rating 4
User {'age': 30, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 54796 with rating 5
User {'age': 31, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 79660 with rating 4
User {'age': 31, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 85503 with rating 5
User {'age': 31, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 18141 with rating 5
User {'age': 32, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 43652 with rating 5
User {'age': 32, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 94995 with rating 4
User {'age': 32, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 46287 with rating 4
User {'age': 33, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 20277 with rating 5
User {'age': 33, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 74092 with rating 3
User {'age': 33, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 33810 with rating 3
User {'age': 34, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 48050 with rating 5
User {'age': 34, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 54437 with rating 5
User {'age': 34, 'gender': 'Male', 'interests': ['Action movies', 'Sci-Fi books']} rated item 71909 with rating 3
```

RESULT:

Thus the attack for tampering with recommender system have been implemented in Python and the output has been verified successfully.

EX NO: 07 **DATE:**	**IMPLEMENT ACCURACY METRICS LIKE RECEIVER OPERATING CHARACTERISTIC CURVES**

AIM:

To implement Receiver Operating Characteristic (ROC) curves in python.

ALGORITHM:

Step 1: Start the program.

Step 2: Import necessary libraries such as numpy, matplotlib, and modules from scikit-learn.

Step 3: Load or generate a dataset suitable for classification.

Step 4: Split the dataset into training and testing sets using train_test_split.

Step 5: Choose a classifier, such as RandomForestClassifier, and train it on the training data.

Step 6: Use the predict_proba method to get predicted probabilities for the positive class on the test data.

Step 7: Calculate the True Positive Rate (TPR) and False Positive Rate (FPR) for different thresholds using roc_curve.

Step 8: Plot the ROC curve with TPR against FPR using matplotlib, adding a diagonal line for random classification reference.

Step 9: Calculate the area under the ROC curve (AUC) with roc_auc_score to measure classifier performance.

Step 10: Interpret the ROC curve and AUC to assess the model's performance.

Step 11: repeat the steps with different classifiers or parameters to compare ROC and AUC results.

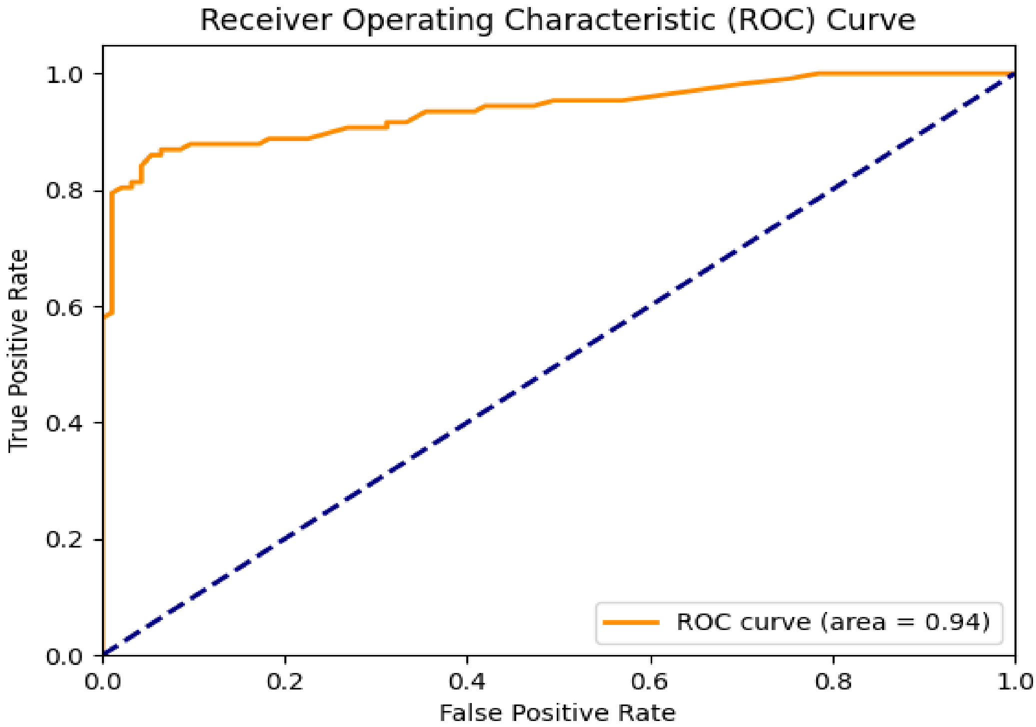
Step 12: End the program.

PROGRAM:

```
from sklearn.datasets import make_classification

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import roc_curve, auc

import matplotlib.pyplot as plt

X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

classifier = RandomForestClassifier(n_estimators=100, random_state=42)

classifier.fit(X_train, y_train)

y_probs = classifier.predict_proba(X_test)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, y_probs)

roc_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.legend(loc="lower right")
```

plt.show()

OUTPUT:



RESULT:

Thus, Python program to implement the Receiver Operated Characteristic curves have been implemented and the output has been verified successfully.