- Name: Deep Salunkhe
- Roll No.:21102A0014

```
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer

def generate_random_data(num_users, num_companies, num_features):
    """Generates random preference vectors and company features with 
names."""
    user_preferences = np.random.rand(num_users, num_features)
    company_features = np.random.rand(num_companies, num_features)

    # Generate random names for users and companies
    user_names = [f"User {i+1}" for i in range(num_users)]
    company_names = [f"Company {i+1}" for i in range(num_companies)]

    return user_preferences, company_features, user_names, 
company_names

def cosine_similarity_recommendation(user_preferences, 
company_features):
    """Recommends jobs using cosine similarity."""
    similarities = user_preferences.dot(company_features.T)
    top_3_indices = np.argsort(similarities, axis=1)[:, -3:]  # Get 
top 3 indices per user
    return top_3_indices

def tfidf_recommendation(user_preferences_text, 
company_features_text):
    """Recommends jobs using TF-IDF."""
    vectorizer = TfidfVectorizer()
    user_vectors = vectorizer.fit_transform(user_preferences_text)
    company_vectors = vectorizer.transform(company_features_text)

    # Handle potential one-dimensional user_vectors:
    if user_vectors.shape[1] == 1:
        user_vectors = user_vectors.reshape(-1, 1)

    similarities = user_vectors.dot(company_vectors.T)
    top_3_indices = np.argsort(similarities, axis=1)[:, -3:]  # Get 
top 3 indices per user
    return top_3_indices

# Example usage with flexibility for numerical or textual data
num_users = 3
num_companies = 10
num_features = 10

user_preferences, company_features, user_names, company_names = 
generate_random_data(num_users, num_companies, num_features)
```

```
# **Optional** Convert to text format for TF-IDF (consider data type)
# user_preferences_text = [" ".join(map(str, user_pref)) for user_pref
in user_preferences]
# company_features_text = [" ".join(map(str, company_feat)) for
company_feat in company_features]

# **Choose appropriate recommendation based on data type**
if isinstance(user_preferences[0][0], str):  # If data is textual
    cosine_recommendations =
cosine_similarity_recommendation(user_preferences_text,
company_features_text)
    tfidf_recommendations =
tfidf_recommendation(user_preferences_text, company_features_text)
else:  # If data is numerical
    cosine_recommendations =
cosine_similarity_recommendation(user_preferences, company_features)
    tfidf_recommendations =
cosine_similarity_recommendation(user_preferences, company_features)
# Same for numerical data

print("Users:")
for i, user in enumerate(user_names):
    print(f"{i+1}. {user}: {user_preferences[i]}")

print("\nCompanies:")
for i, company in enumerate(company_names):
    print(f"{i+1}. {company}: {company_features[i]}")

print("\nCosine Similarity Recommendations:")
for i, user_id in enumerate(range(num_users)):
    print(f"User {user_id+1}:", [company_names[idx] for idx in
cosine_recommendations[i]])
    print("  Features:", [company_features[idx] for idx in
cosine_recommendations[i]])

print("\nTF-IDF Recommendations:")
for i, user_id in enumerate(range(num_users)):
    print(f"User {user_id+1}:", [company_names[idx] for idx in
tfidf_recommendations[i]])
    print("  Features:", [company_features[idx] for idx in
tfidf_recommendations[i]])

# Example of different results
user_preferences[0][0] = 0.9  # Adjust user preferences
user_preferences[0][1] = 0.1

Users:
1. User 1: [0.11036971 0.60075994 0.16833709 0.00914947 0.43507662
0.28023306
 0.02290225 0.79640019 0.98295488 0.26142673]
```

```
2. User 2: [0.425977   0.65766369 0.9701737  0.65879481 0.60786215
0.07416477
 0.95750273 0.06190925 0.34324352 0.81100159]
3. User 3: [0.88062676 0.64899705 0.49912342 0.20537359 0.18012605
0.65777237
 0.27083193 0.45187614 0.14996548 0.17305661]

Companies:
1. Company 1: [0.15329739 0.8407762  0.91807368 0.42883621 0.15573593
0.06765897
 0.22817119 0.65363924 0.30933041 0.21559711]
2. Company 2: [0.03646343 0.85845221 0.16556934 0.88705666 0.50235906
0.96790686
 0.55354814 0.35085778 0.38667873 0.16502561]
3. Company 3: [0.17942199 0.00126556 0.27302754 0.25692292 0.57520424
0.63306435
 0.01608568 0.08660631 0.86455036 0.62893808]
4. Company 4: [0.12834332 0.96050937 0.07815112 0.38392233 0.41483286
0.19376785
 0.33531162 0.91722339 0.06874432 0.34610868]
5. Company 5: [0.21907787 0.69588818 0.24097526 0.45871519 0.7790164
0.67787636
 0.92702882 0.9476927  0.21480999 0.53158108]
6. Company 6: [0.61120607 0.75131388 0.0405824  0.02794155 0.12865411
0.58715622
 0.71908204 0.05193806 0.90686114 0.87958723]
7. Company 7: [0.16163406 0.11340534 0.39113475 0.67713112 0.12810157
0.00231448
 0.96871683 0.19354144 0.09138731 0.79290645]
8. Company 8: [0.79942391 0.86844686 0.85474881 0.02766085 0.91467487
0.11048766
 0.19830592 0.93453914 0.98203221 0.215408  ]
9. Company 9: [0.00544703 0.69451812 0.42939063 0.11546643 0.10420998
0.55269172
 0.36279096 0.84073908 0.02686234 0.99649595]
10. Company 10: [0.13408179 0.21567755 0.00670276 0.08388703
0.75317958 0.87495841
 0.31811653 0.06462547 0.23948954 0.31523869]

Cosine Similarity Recommendations:
User 1: ['Company 6', 'Company 5', 'Company 8']
  Features: [array([0.61120607, 0.75131388, 0.0405824 , 0.02794155,
0.12865411,
       0.58715622, 0.71908204, 0.05193806, 0.90686114, 0.87958723]),
array([0.21907787, 0.69588818, 0.24097526, 0.45871519, 0.7790164 ,
       0.67787636, 0.92702882, 0.9476927 , 0.21480999, 0.53158108]),
array([0.79942391, 0.86844686, 0.85474881, 0.02766085, 0.91467487,
       0.11048766, 0.19830592, 0.93453914, 0.98203221, 0.215408  ])]
User 2: ['Company 7', 'Company 5', 'Company 8']
```

```
  Features: [array([0.16163406, 0.11340534, 0.39113475, 0.67713112,
0.12810157,
       0.00231448, 0.96871683, 0.19354144, 0.09138731, 0.79290645]),
array([0.21907787, 0.69588818, 0.24097526, 0.45871519, 0.7790164 ,
       0.67787636, 0.92702882, 0.9476927 , 0.21480999, 0.53158108]),
array([0.79942391, 0.86844686, 0.85474881, 0.02766085, 0.91467487,
       0.11048766, 0.19830592, 0.93453914, 0.98203221, 0.215408  ])]
User 3: ['Company 2', 'Company 5', 'Company 8']
  Features: [array([0.03646343, 0.85845221, 0.16556934, 0.88705666,
0.50235906,
       0.96790686, 0.55354814, 0.35085778, 0.38667873, 0.16502561]),
array([0.21907787, 0.69588818, 0.24097526, 0.45871519, 0.7790164 ,
       0.67787636, 0.92702882, 0.9476927 , 0.21480999, 0.53158108]),
array([0.79942391, 0.86844686, 0.85474881, 0.02766085, 0.91467487,
       0.11048766, 0.19830592, 0.93453914, 0.98203221, 0.215408  ])]

TF-IDF Recommendations:
User 1: ['Company 6', 'Company 5', 'Company 8']
  Features: [array([0.61120607, 0.75131388, 0.0405824 , 0.02794155,
0.12865411,
       0.58715622, 0.71908204, 0.05193806, 0.90686114, 0.87958723]),
array([0.21907787, 0.69588818, 0.24097526, 0.45871519, 0.7790164 ,
       0.67787636, 0.92702882, 0.9476927 , 0.21480999, 0.53158108]),
array([0.79942391, 0.86844686, 0.85474881, 0.02766085, 0.91467487,
       0.11048766, 0.19830592, 0.93453914, 0.98203221, 0.215408  ])]
User 2: ['Company 7', 'Company 5', 'Company 8']
  Features: [array([0.16163406, 0.11340534, 0.39113475, 0.67713112,
0.12810157,
       0.00231448, 0.96871683, 0.19354144, 0.09138731, 0.79290645]),
array([0.21907787, 0.69588818, 0.24097526, 0.45871519, 0.7790164 ,
       0.67787636, 0.92702882, 0.9476927 , 0.21480999, 0.53158108]),
array([0.79942391, 0.86844686, 0.85474881, 0.02766085, 0.91467487,
       0.11048766, 0.19830592, 0.93453914, 0.98203221, 0.215408  ])]
User 3: ['Company 2', 'Company 5', 'Company 8']
  Features: [array([0.03646343, 0.85845221, 0.16556934, 0.88705666,
0.50235906,
       0.96790686, 0.55354814, 0.35085778, 0.38667873, 0.16502561]),
array([0.21907787, 0.69588818, 0.24097526, 0.45871519, 0.7790164 ,
       0.67787636, 0.92702882, 0.9476927 , 0.21480999, 0.53158108]),
array([0.79942391, 0.86844686, 0.85474881, 0.02766085, 0.91467487,
       0.11048766, 0.19830592, 0.93453914, 0.98203221, 0.215408  ])]
```