Step 1: Import Libraries

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MultiLabelBinarizer, LabelEncoder
from sklearn.neighbors import NearestNeighbors
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns
```

Step 2: Generate Sample Data

```
# Number of aspirants and mentors to generate
num_aspirants = 20
num_mentors = 10

# Possible values for features
possible_subjects = ['Constitutional Law', 'Legal Reasoning', 'Contract Law', 'Criminal Law', 'Torts', 'Jurisprudence', 'Family Law']
possible_colleges = ['NLSIU Bangalore', 'NALSAR Hyderabad', 'NUJS Kolkata', 'NLIU Bhopal', 'GNLU Gandhinagar', 'NLU Jodhpur']
possible_preparation_levels = ['Beginner', 'Intermediate', 'Advanced']
possible_learning_styles = ['Visual', 'Analytical', 'Conceptual', 'Practical', 'Theoretical']
possible_mentoring_styles = ['Structured, Concept-focused', 'Analytical, Discussion-based', 'Practical, Problem-solving', 'Supportive, Persona

# --- Generate Aspirant Data ---
aspirant_data = {
    'aspirant_id': range(1, num_aspirants + 1),
    'preferred_subjects': [', '.join(np.random.choice(possible_subjects, size=np.random.randint(1, 3), replace=False)) for _ in range(num_asp
    'target_colleges': [', '.join(np.random.choice(possible_colleges, size=np.random.randint(1, 3), replace=False)) for _ in range(num_aspira
    'preparation_level': np.random.choice(possible_preparation_levels, size=num_aspirants),
    'learning_style': np.random.choice(possible_learning_styles, size=num_aspirants)
}
aspirant_df = pd.DataFrame(aspirant_data).set_index('aspirant_id')

# --- Generate Mentor Data ---
mentor_data = {
    'mentor_id': range(101, num_mentors + 101),
    'expertise_subjects': [', '.join(np.random.choice(possible_subjects, size=np.random.randint(1, 4), replace=False)) for _ in range(num_men
    'graduated_college': np.random.choice(possible_colleges, size=num_mentors),
    'mentoring_style': np.random.choice(possible_mentoring_styles, size=num_mentors),
    'clat_rank': np.random.randint(1, 50, size=num_mentors)
}
mentor_df = pd.DataFrame(mentor_data).set_index('mentor_id')

print("--- Sample Aspirant Data ---")
print(aspirant_df.head())
print("\n--- Sample Mentor Data ---")
print(mentor_df.head())
```

```
--- Sample Aspirant Data ---
                        preferred_subjects                target_colleges  \
aspirant_id
1                              Family Law                    NLIU Bhopal
2              Torts, Constitutional Law  NUJS Kolkata, NALSAR Hyderabad
3                         Legal Reasoning   NLSIU Bangalore, NUJS Kolkata
4                           Contract Law               NALSAR Hyderabad
5            Contract Law, Jurisprudence                    NLU Jodhpur

             preparation_level learning_style
aspirant_id
1                    Advanced      Practical
2                Intermediate    Theoretical
3                    Advanced      Practical
4                    Advanced    Theoretical
5                    Beginner     Conceptual

--- Sample Mentor Data ---
                                        expertise_subjects graduated_college  \
mentor_id
101                      Torts, Criminal Law, Jurisprudence      NUJS Kolkata
102                                   Legal Reasoning   NLSIU Bangalore
103        Constitutional Law, Family Law, Legal Reasoning       NLIU Bhopal
104                          Criminal Law, Contract Law       NLIU Bhopal
105        Constitutional Law, Contract Law, Jurisprudence   NLSIU Bangalore

                          mentoring_style  clat_rank
mentor_id
101         Practical, Problem-solving         33
102        Structured, Concept-focused         33
103        Structured, Concept-focused         19
104         Practical, Problem-solving         37
105         Practical, Problem-solving         15
```

Step 3: Feature Engineering and Preprocessing

```
# --- Feature Processing for Aspirants ---
mlb_aspirant_subjects = MultiLabelBinarizer()
aspirant_subjects_matrix = mlb_aspirant_subjects.fit_transform(aspirant_df['preferred_subjects'].apply(lambda x: x.split(', ')))
aspirant_subjects_df = pd.DataFrame(aspirant_subjects_matrix, index=aspirant_df.index, columns=mlb_aspirant_subjects.classes_)

mlb_aspirant_colleges = MultiLabelBinarizer()
aspirant_colleges_matrix = mlb_aspirant_colleges.fit_transform(aspirant_df['target_colleges'].apply(lambda x: x.split(', ')))
aspirant_colleges_df = pd.DataFrame(aspirant_colleges_matrix, index=aspirant_df.index, columns=mlb_aspirant_colleges.classes_)

aspirant_preparation_df = pd.get_dummies(aspirant_df['preparation_level'], prefix='prep_level', dummy_na=False)
aspirant_learning_df = pd.get_dummies(aspirant_df['learning_style'], prefix='learn_style', dummy_na=False)
aspirant_profile_df = pd.concat([aspirant_subjects_df, aspirant_colleges_df, aspirant_preparation_df, aspirant_learning_df], axis=1).fillna(
```

```
# --- Feature Processing for Mentors ---
mlb_mentor_subjects = MultiLabelBinarizer()
mentor_subjects_matrix = mlb_mentor_subjects.fit_transform(mentor_df['expertise_subjects'].apply(lambda x: x.split(', ')))
mentor_subjects_df = pd.DataFrame(mentor_subjects_matrix, index=mentor_df.index, columns=mlb_mentor_subjects.classes_)

mentor_college_df = pd.get_dummies(mentor_df['graduated_college'], prefix='grad_college', dummy_na=False)
mentor_mentoring_df = pd.get_dummies(mentor_df['mentoring_style'], prefix='mentor_style', dummy_na=False)
mentor_profile_df = pd.concat([mentor_subjects_df, mentor_college_df, mentor_mentoring_df], axis=1).fillna(0)

# --- Align Features ---
common_features = list(set(aspirant_profile_df.columns) & set(mentor_profile_df.columns))
aspirant_profile_aligned = aspirant_profile_df[common_features].fillna(0)
mentor_profile_aligned = mentor_profile_df[common_features].fillna(0)
```

Step 4: Train the KNN Model

```
n_neighbors = 5  # Define the number of nearest neighbors to consider
knn_model_aspirants = NearestNeighbors(n_neighbors=n_neighbors, metric='cosine')
knn_model_aspirants.fit(aspirant_profile_aligned)
```

```
NearestNeighbors
NearestNeighbors(metric='cosine')
```

Step 5: Define the Recommendation Function using KNN

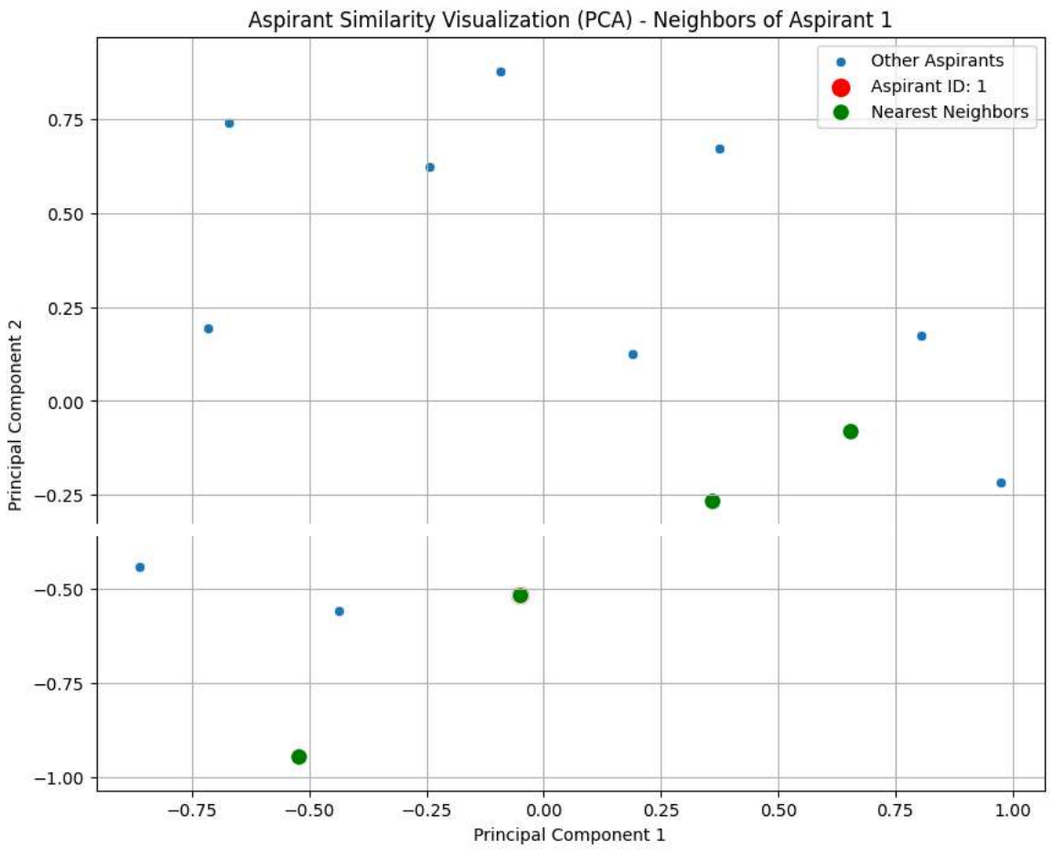
```
def recommend_mentors_knn(aspirant_id, aspirant_profiles, mentor_profiles, mentor_df, knn_model, top_n=3):
    if aspirant_id not in aspirant_profiles.index:
        return f"Aspirant ID {aspirant_id} not found."

    aspirant_vector = aspirant_profiles.loc[[aspirant_id]]
    distances, indices = knn_model.kneighbors(aspirant_vector)

    # Get the IDs of the nearest neighbors (other aspirants)
    nearest_aspirant_ids = aspirant_profiles.index[indices[0]].tolist()
    nearest_aspirant_ids.remove(aspirant_id) # Exclude the aspirant themselves

    if not nearest_aspirant_ids:
        return "No similar aspirants found."

    # Calculate the average profile of the nearest neighbors
    avg_neighbor_profile = aspirant_profiles.loc[nearest_aspirant_ids].mean(axis=0).values.reshape(1, -1)

    # Calculate cosine similarity between the average neighbor profile and all mentor profiles
    similarities_to_mentors = cosine_similarity(mentor_profiles, avg_neighbor_profile)
    similarities_series = pd.Series(similarities_to_mentors.flatten(), index=mentor_profiles.index)
    top_mentor_ids = similarities_series.sort_values(ascending=False).head(top_n).index.tolist()
    top_mentors_info = mentor_df.loc[top_mentor_ids][['expertise_subjects', 'graduated_college', 'clat_rank']]
    return top_mentors_info

# Get KNN-based recommendations for the first aspirant
aspirant_id_to_recommend = aspirant_profile_aligned.index[0]
top_recommendations_knn = recommend_mentors_knn(aspirant_id_to_recommend, aspirant_profile_aligned, mentor_profile_aligned, mentor_df, knn_m
print(f"\n--- Top 3 Mentor Recommendations (KNN) for Aspirant ID {aspirant_id_to_recommend} ---")
print(top_recommendations_knn)
```

```
--- Top 3 Mentor Recommendations (KNN) for Aspirant ID 1 ---
                                    expertise_subjects graduated_college  \
mentor_id
103        Constitutional Law, Family Law, Legal Reasoning       NLIU Bhopal
104                             Criminal Law, Contract Law       NLIU Bhopal
106                                         Contract Law      NUJS Kolkata

           clat_rank
mentor_id
103               19
104               37
106               20
```

Step 6: Visualization of Aspirant Similarities (using PCA)

```
from sklearn.decomposition import PCA

# Reduce dimensionality of aspirant profiles using PCA
pca = PCA(n_components=2)
principal_components_aspirants = pca.fit_transform(aspirant_profile_aligned)
pca_aspirant_df = pd.DataFrame(data=principal_components_aspirants, index=aspirant_profile_aligned.index, columns=['PC1', 'PC2'])

# Select an aspirant to visualize their neighbors
aspirant_to_visualize = aspirant_profile_aligned.index[0]
aspirant_vector_to_visualize = aspirant_profile_aligned.loc[[aspirant_to_visualize]]
distances, indices = knn_model_aspirants.kneighbors(aspirant_vector_to_visualize)
neighbor_ids = aspirant_profile_aligned.index[indices[0]].tolist()

# Visualization
plt.figure(figsize=(10, 8))
sns.scatterplot(x='PC1', y='PC2', data=pca_aspirant_df, label='Other Aspirants')
plt.scatter(x=pca_aspirant_df.loc[aspirant_to_visualize, 'PC1'], y=pca_aspirant_df.loc[aspirant_to_visualize, 'PC2'], color='red', s=100, la
sns.scatterplot(x=pca_aspirant_df.loc[neighbor_ids[1:], 'PC1'], y=pca_aspirant_df.loc[neighbor_ids[1:], 'PC2'], color='green', s=100, label=

plt.title(f'Aspirant Similarity Visualization (PCA) - Neighbors of Aspirant {aspirant_to_visualize}')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
```

```
plt.grid(True)
plt.show()

print(f"\nNearest Neighbors of Aspirant {aspirant_to_visualize}: {neighbor_ids}")
```




```
Nearest Neighbors of Aspirant 1: [1, 17, 7, 3, 4]
```

Step 7: (Optional) Visualization of Mentor Distribution (using PCA)

Step 7: (Optional) Visualization of Mentor Distribution (using PCA)

```
# Reduce dimensionality of mentor profiles using PCA
pca_mentors = PCA(n_components=2)
principal_components_mentors = pca_mentors.fit_transform(mentor_profile_aligned)
pca_mentor_df = pd.DataFrame(data=principal_components_mentors, index=mentor_profile_aligned.index, columns=['PC1', 'PC2'])

# Visualization
plt.figure(figsize=(8, 6))
sns.scatterplot(x='PC1', y='PC2', data=pca_mentor_df, label='Mentors')
plt.title('Mentor Distribution Visualization (PCA)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.grid(True)
plt.show()
```