

```

In [1]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import NearestNeighbors

# Input data
X = np.array([[2.1, 1.3], [1.3, 3.2], [2.9, 2.5], [2.7, 5.4], [3.8, 0.9],
              [7.3, 2.1], [4.2, 6.5], [3.8, 3.7], [2.5, 4.1], [3.4, 1.9],
              [5.7, 3.5], [6.1, 4.3], [5.1, 2.2], [6.2, 1.1]])

# Number of nearest neighbors
k = 5

# Test datapoint
test_datapoint = [4.3, 2.7]

# Plot input data
plt.figure()
plt.title('Input data')
plt.scatter(X[:,0], X[:,1], marker='o', s=75, color='black')

# Build K Nearest Neighbors model
knn_model = NearestNeighbors(n_neighbors=k, algorithm='ball_tree').fit(X)
distances, indices = knn_model.kneighbors([test_datapoint])

# Print the 'k' nearest neighbors
print("\nK Nearest Neighbors:")
for rank, index in enumerate(indices[0][:k], start=1):
    print(str(rank) + " ==>", X[index])

# Visualize the nearest neighbors along with the test datapoint
plt.figure()
plt.title('Nearest neighbors')
plt.scatter(X[:, 0], X[:, 1], marker='o', s=75, color='k')
plt.scatter(X[indices[0][:k][:, 0], 0], X[indices[0][:k][:, 1], 1],
            marker='o', s=250, color='k', facecolors='none')
plt.scatter(test_datapoint[0], test_datapoint[1],
            marker='x', s=75, color='k')

plt.show()

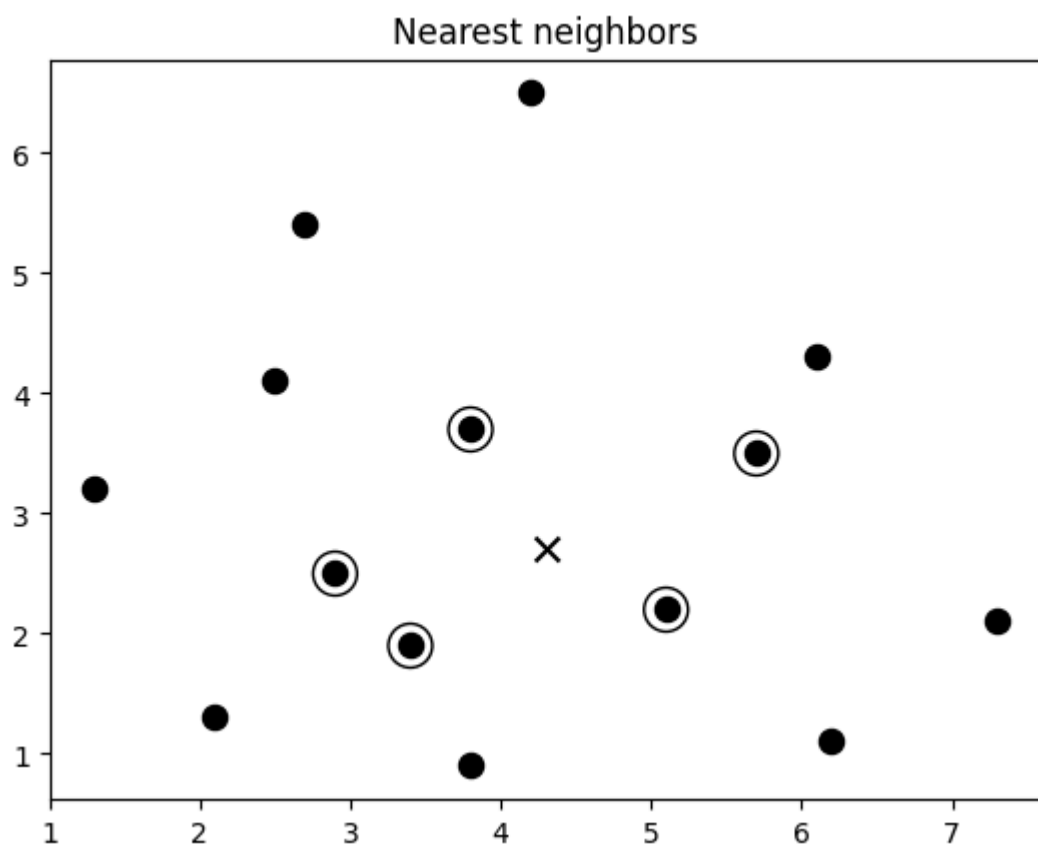
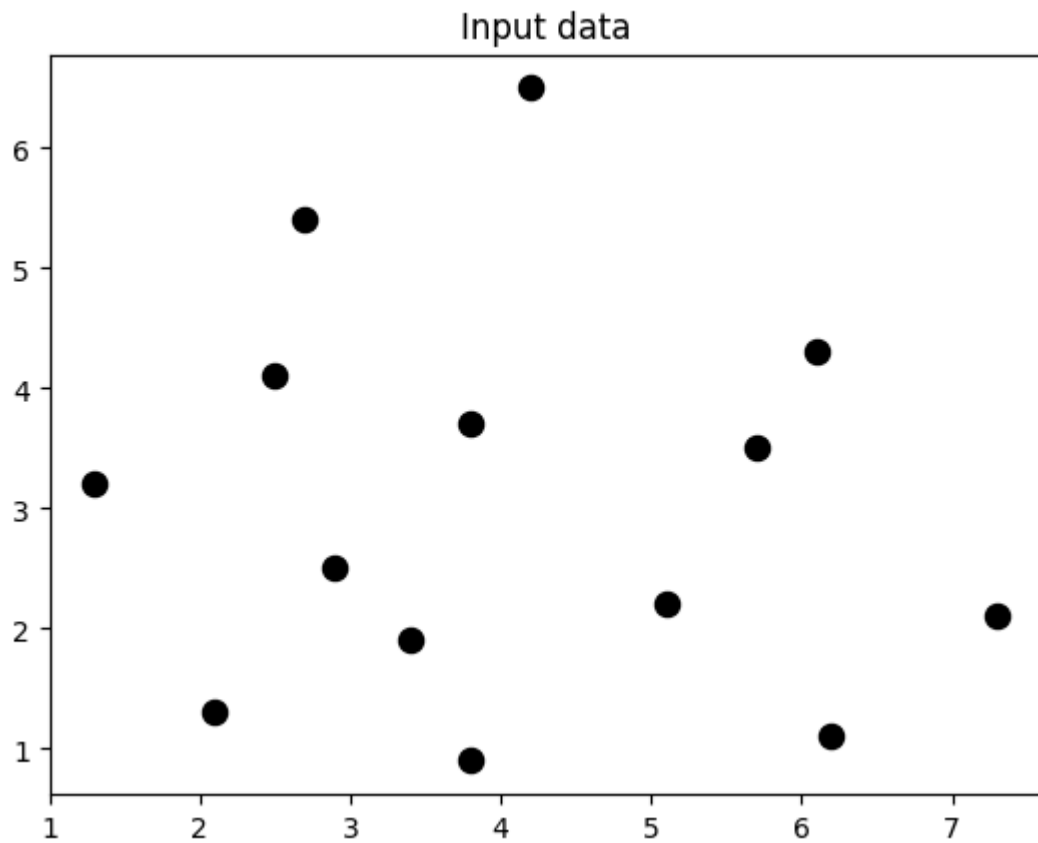
```

K Nearest Neighbors:

```

1 ==> [5.1 2.2]
2 ==> [3.8 3.7]
3 ==> [3.4 1.9]
4 ==> [2.9 2.5]
5 ==> [5.7 3.5]

```



Ce code met en œuvre l'algorithme des k plus proches voisins (KNN) pour identifier les 5 points les plus proches d'un point test dans un ensemble de données 2D.

Il utilise la bibliothèque scikit-learn pour créer un modèle KNN avec l'algorithme Ball Tree, qui est optimisé pour la recherche de voisinage dans des espaces de grande dimension.

Après l'entraînement du modèle, le programme affiche les voisins les plus proches et trace un graphique avec les points d'origine, le point test marqué par un X, et les voisins entourés de cercles.

```
In [4]: import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn import neighbors

# 🚀 Charger Les données
input_file = r"C:\Users\smain\Downloads\data.txt"

data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1].astype(int)

# 🚀 Affichage des données d'entrée
plt.figure()
plt.title('Input data')
marker_shapes = 'v^os'
mapper = [marker_shapes[i] for i in y]

for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
                s=75, edgecolors='black', facecolor='none')

# 🚀 Initialisation du classificateur KNN
num_neighbors = 12
step_size = 0.01
classifier = neighbors.KNeighborsClassifier(num_neighbors, weights='distance')
classifier.fit(X, y)

# 🚀 Création du maillage pour visualiser les frontières de décision
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
x_values, y_values = np.meshgrid(np.arange(x_min, x_max, step_size),
                                  np.arange(y_min, y_max, step_size))

# 🚀 Prédiction sur le maillage
output = classifier.predict(np.c_[x_values.ravel(), y_values.ravel()])
output = output.reshape(x_values.shape)

# 🚀 Visualisation des résultats
plt.figure()
plt.pcolormesh(x_values, y_values, output.astype(float), cmap=cm.Paired)

# 🚀 Superposition des points d'entraînement
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
                s=50, edgecolors='black', facecolor='none')

plt.xlim(x_values.min(), x_values.max())
plt.ylim(y_values.min(), y_values.max())
plt.title('K Nearest Neighbors classifier model boundaries')

# 🚀 Test d'un point de données
test_datapoint = [5.1, 3.6]
plt.figure()
plt.title('Test datapoint')
```

```

for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
                s=75, edgecolors='black', facecolor='none')

plt.scatter(test_datapoint[0], test_datapoint[1], marker='x',
            linewidth=6, s=200, facecolor='black')

# ✦ Extraction des K voisins les plus proches
_, indices = classifier.kneighbors([test_datapoint])
indices = indices.astype(int)[0]

# ✦ Visualisation des K voisins les plus proches
plt.figure()
plt.title('K Nearest Neighbors')

for i in indices:
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[y[i]],
                linewidth=3, s=100, facecolor='black')

plt.scatter(test_datapoint[0], test_datapoint[1], marker='x',
            linewidth=6, s=200, facecolor='black')

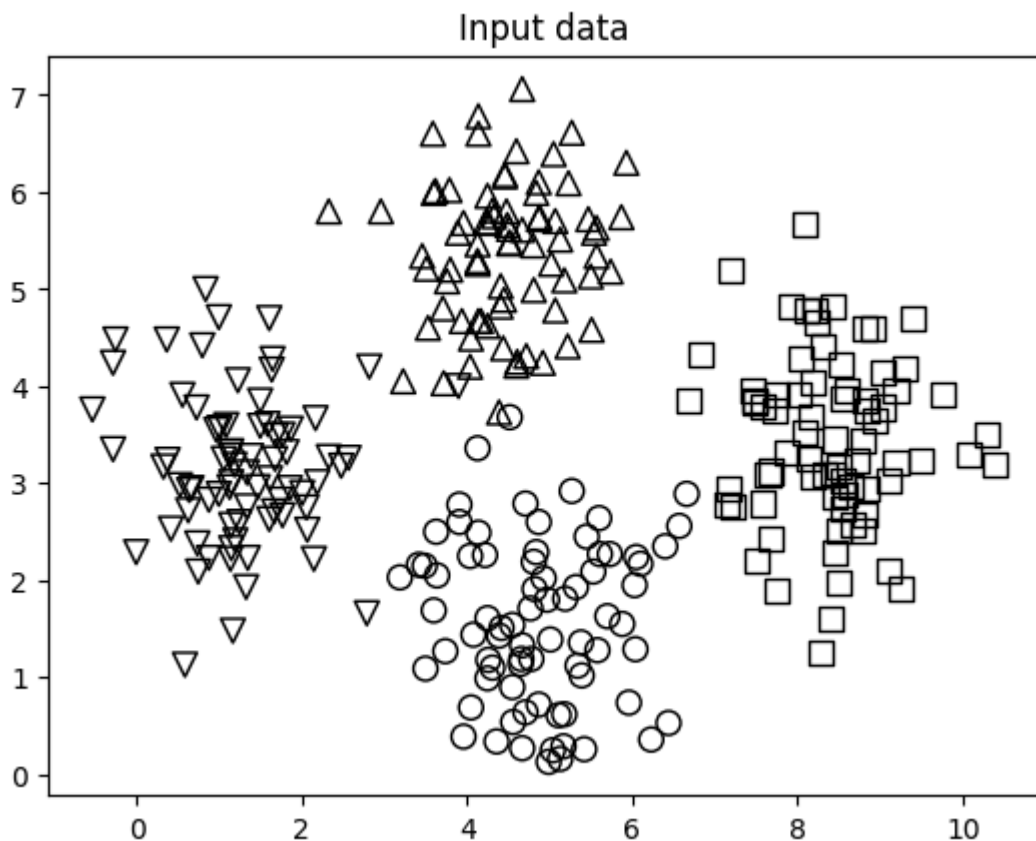
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
                s=75, edgecolors='black', facecolor='none')

# ✦ Affichage du résultat final
prediction = classifier.predict([test_datapoint])[0]
print("✅ Prédiction pour le point de test :", prediction)

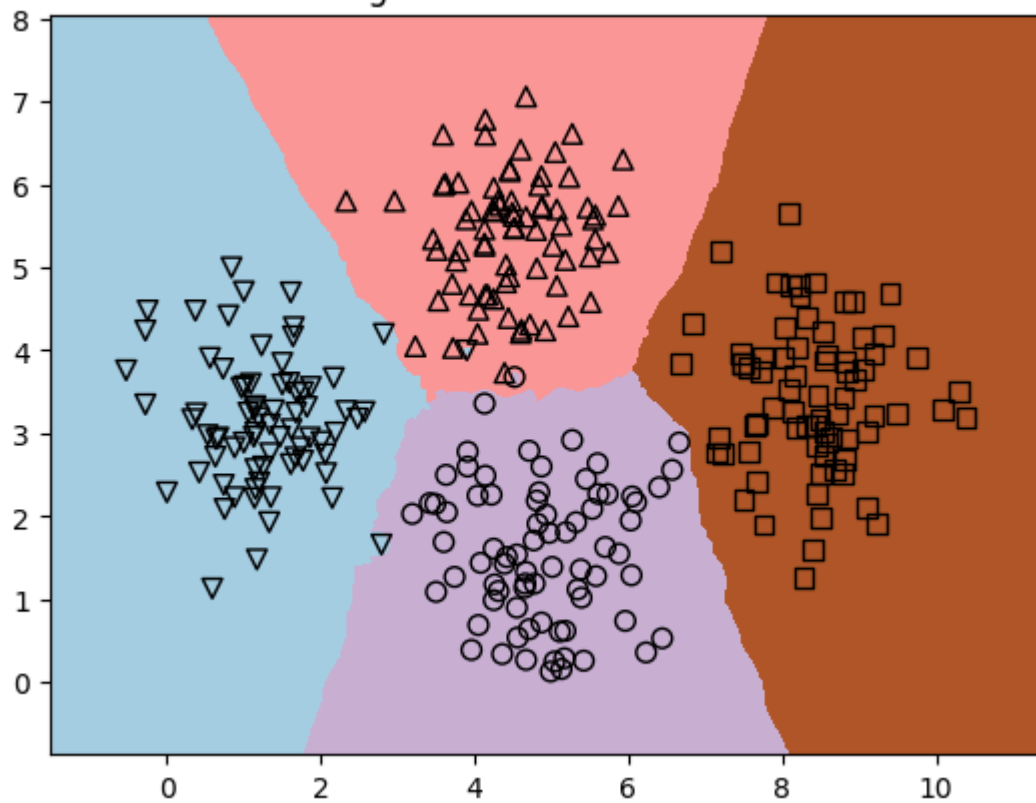
plt.show()

```

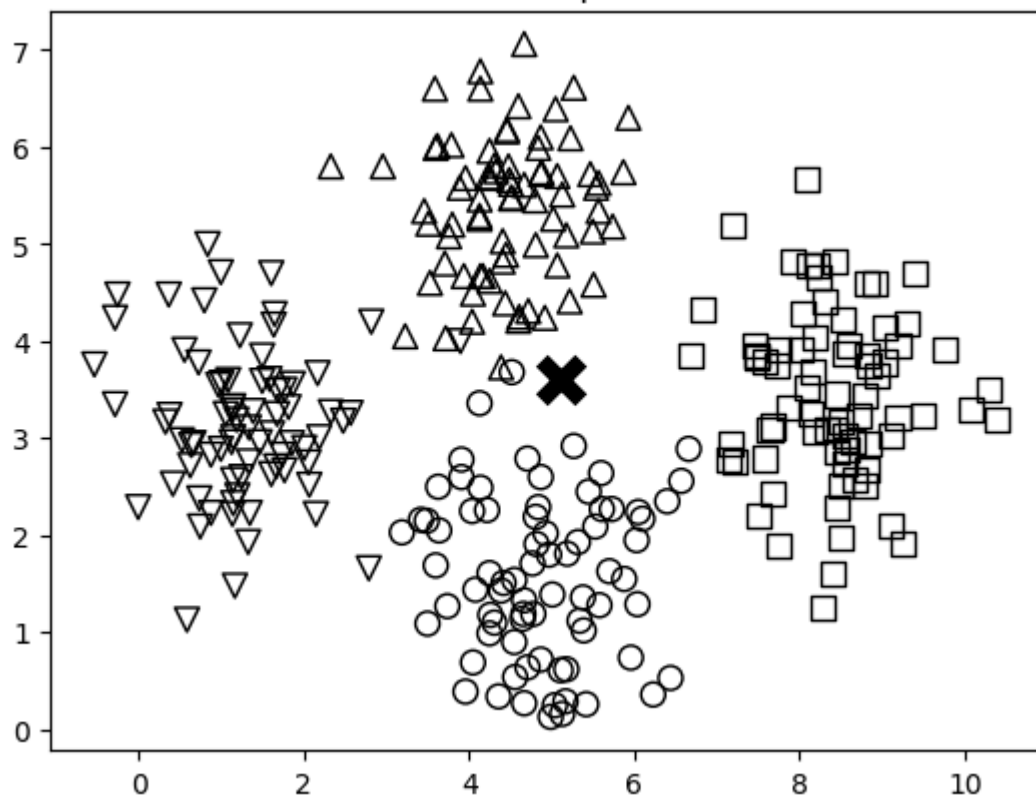
✅ Prédiction pour le point de test : 1

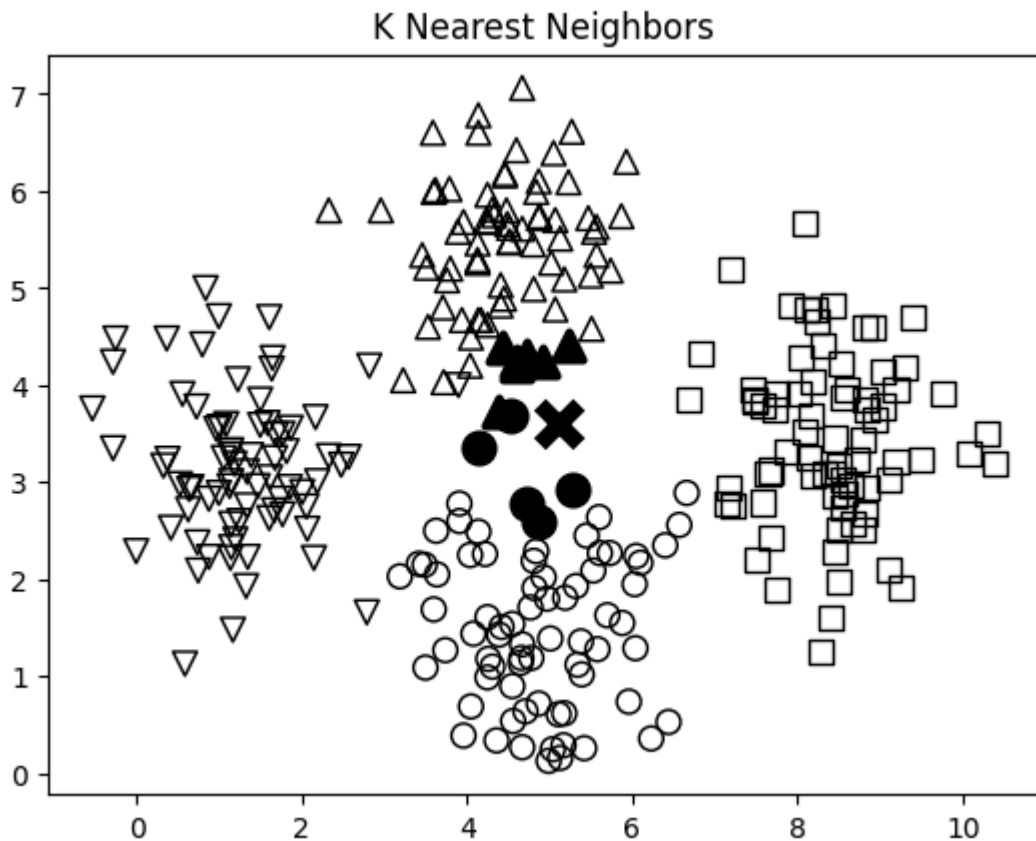


K Nearest Neighbors classifier model boundaries



Test datapoint





Ce code implémente un classificateur K-Nearest Neighbors (KNN) pour classifier des données bidimensionnelles. Il charge les données depuis un fichier, entraîne un modèle KNN avec 12 voisins, et visualise les frontières de décision.

Il teste un point de données [5.1, 3.6] et affiche les 12 plus proches voisins, avant de prédire sa classe. Plusieurs graphes sont générés pour représenter les données d'origine, la décision du modèle, et la proximité des voisins du point de test.

```
In [10]: import json
import numpy as np
import os

# 📌 Spécifier le chemin absolu du fichier JSON
file_path = r"C:\Users\smain\Downloads\ratings.json"

# 📌 Fonction pour charger les données JSON
def load_ratings(file_path):
    if not os.path.exists(file_path):
        raise FileNotFoundError(f"❌ ERREUR : Le fichier {file_path} est introuvable")

    try:
        with open(file_path, 'r', encoding='utf-8') as f:
            data = json.load(f)
        return data
    except json.JSONDecodeError:
        raise ValueError(f"❌ ERREUR : Le fichier {file_path} n'est pas un JSON")

# 📌 Afficher la liste des utilisateurs disponibles
def list_users(file_path):
    data = load_ratings(file_path)
```

```

print("\n🚀 Utilisateurs disponibles :")
for user in data.keys():
    print(f"    → {user}")

# 🚀 Calcul de la distance euclidienne
def euclidean_score(dataset, user1, user2):
    if user1 not in dataset or user2 not in dataset:
        list_users(file_path)
        raise ValueError(f"❌ ERREUR : Un des utilisateurs ({user1}, {user2}) n'est pas dans le dataset")

    common_movies = {item for item in dataset[user1] if item in dataset[user2]}
    if not common_movies:
        return 0

    squared_diff = [np.square(dataset[user1][item] - dataset[user2][item]) for item in common_movies]
    return 1 / (1 + np.sqrt(np.sum(squared_diff)))

# 🚀 Calcul du score de Pearson
def pearson_score(dataset, user1, user2):
    if user1 not in dataset or user2 not in dataset:
        list_users(file_path)
        raise ValueError(f"❌ ERREUR : Un des utilisateurs ({user1}, {user2}) n'est pas dans le dataset")

    common_movies = {item for item in dataset[user1] if item in dataset[user2]}
    num_ratings = len(common_movies)

    if num_ratings == 0:
        return 0

    user1_ratings = np.array([dataset[user1][item] for item in common_movies])
    user2_ratings = np.array([dataset[user2][item] for item in common_movies])

    mean_user1 = np.mean(user1_ratings)
    mean_user2 = np.mean(user2_ratings)

    numerator = np.sum((user1_ratings - mean_user1) * (user2_ratings - mean_user2))
    denominator = np.sqrt(np.sum((user1_ratings - mean_user1) ** 2) * np.sum((user2_ratings - mean_user2) ** 2))

    return 0 if denominator == 0 else numerator / denominator

# 🚀 Comparaison entre deux utilisateurs
def compare_users(user1, user2, score_type="Euclidean", file_path=file_path):
    try:
        data = load_ratings(file_path)

        if score_type == "Euclidean":
            score = euclidean_score(data, user1, user2)
            print(f"\n📊 Score Euclidien entre {user1} et {user2} : {score:.4f}")
        elif score_type == "Pearson":
            score = pearson_score(data, user1, user2)
            print(f"\n📊 Score de Pearson entre {user1} et {user2} : {score:.4f}")
        else:
            print("❌ ERREUR : Méthode non reconnue. Utilisez 'Euclidean' ou 'Pearson'")


    except FileNotFoundError as e:
        print(e)
    except ValueError as e:
        print(e)

# 🚀 Exemple d'exécution

```

```
compare_users("David Smith", "Brenda Peterson", "Euclidean")
compare_users("David Smith", "Brenda Peterson", "Pearson")
```

 Score Euclidien entre David Smith et Brenda Peterson : 0.1424

 Score de Pearson entre David Smith et Brenda Peterson : -0.7237

Ce code compare les préférences de films entre deux utilisateurs en utilisant deux métriques de similarité : la distance euclidienne et le coefficient de Pearson. Il charge les données d'évaluations à partir d'un fichier JSON et vérifie l'existence des utilisateurs avant de calculer les scores.

Si les utilisateurs existent et ont des films en commun, il affiche leur score de similarité. Sinon, il signale une erreur. Deux comparaisons sont effectuées entre David Smith et Brenda Peterson.

```
In [12]: import json
import numpy as np
import os

# 📌 Spécifier le chemin absolu du fichier JSON
file_path = r"C:\Users\smain\Downloads\ratings.json"

# 📌 Fonction pour charger les données JSON
def load_ratings(file_path):
    if not os.path.exists(file_path):
        raise FileNotFoundError(f"❌ ERREUR : Le fichier {file_path} est introuvable")

    try:
        with open(file_path, 'r', encoding='utf-8') as f:
            data = json.load(f)
        return data
    except json.JSONDecodeError:
        raise ValueError(f"❌ ERREUR : Le fichier {file_path} n'est pas un JSON")

# 📌 Afficher la liste des utilisateurs disponibles
def list_users(file_path):
    data = load_ratings(file_path)
    print("\n📌 Utilisateurs disponibles :")
    for user in data.keys():
        print(f"    → {user}")

# 📌 Calcul du score de Pearson (déjà défini dans `compute_scores.py`, on le redefinit ici)
def pearson_score(dataset, user1, user2):
    if user1 not in dataset or user2 not in dataset:
        list_users(file_path)
        raise ValueError(f"❌ ERREUR : Un des utilisateurs ({user1}, {user2}) n'existe pas")

    common_movies = {item for item in dataset[user1] if item in dataset[user2]}
    num_ratings = len(common_movies)

    if num_ratings == 0:
        return 0

    user1_ratings = np.array([dataset[user1][item] for item in common_movies])
    user2_ratings = np.array([dataset[user2][item] for item in common_movies])
```



```

mean_user1 = np.mean(user1_ratings)
mean_user2 = np.mean(user2_ratings)

numerator = np.sum((user1_ratings - mean_user1) * (user2_ratings - mean_user2))
denominator = np.sqrt(np.sum((user1_ratings - mean_user1) ** 2) * np.sum((user2_ratings - mean_user2) ** 2))

return 0 if denominator == 0 else numerator / denominator

# 🚀 Trouver Les utilisateurs similaires
def find_similar_users(dataset, user, num_users=3):
    if user not in dataset:
        list_users(file_path)
        raise ValueError(f"❌ ERREUR : L'utilisateur '{user}' n'existe pas dans le dataset")

    # Calculer Les scores de similarité avec Les autres utilisateurs
    scores = np.array([[x, pearson_score(dataset, user, x)] for x in dataset])

    # Trier Les scores en ordre décroissant
    scores_sorted = scores[scores[:, 1].argsort()[::-1]]

    # Extraire Les `num_users` meilleurs scores
    top_users = scores_sorted[:num_users]

    return top_users

# 🚀 Fonction pour exécuter dans Jupyter Notebook
def find_similar_users_notebook(user, num_users=3):
    try:
        data = load_ratings(file_path)
        similar_users = find_similar_users(data, user, num_users)

        print(f"\n🚀 Utilisateurs similaires à {user} :\n")
        print(f"{'Utilisateur':<25}{'Score de Similarité'}")
        print('-' * 50)
        for item in similar_users:
            print(f"{item[0]:<25}{float(item[1]):.4f}")

    except FileNotFoundError as e:
        print(e)
    except ValueError as e:
        print(e)

# 🚀 Exemple d'exécution dans Jupyter
find_similar_users_notebook("David Smith", 3)

```

🚀 Utilisateurs similaires à David Smith :

Utilisateur	Score de Similarité
Chris Duncan	1.0000
Bill Duffy	0.9910
Adam Cohen	0.9081

Ce code identifie les utilisateurs les plus similaires à un utilisateur donné en analysant leurs évaluations de films à l'aide du coefficient de Pearson.

Il commence par charger les données JSON et vérifier l'existence de l'utilisateur cible. Ensuite, il calcule la similarité entre cet utilisateur et les autres, trie les résultats, et affiche les trois utilisateurs les plus proches avec leurs scores.

Si l'utilisateur n'existe pas, il affiche la liste des utilisateurs disponibles.

```
In [15]: import json
import numpy as np
import os

# 📌 Spécifier le chemin absolu du fichier JSON
file_path = r"C:\Users\smain\Downloads\ratings.json"

# 📌 Fonction pour charger les données JSON
def load_ratings(file_path):
    if not os.path.exists(file_path):
        raise FileNotFoundError(f"❌ ERREUR : Le fichier {file_path} est introuvable")

    try:
        with open(file_path, 'r', encoding='utf-8') as f:
            data = json.load(f)
        return data
    except json.JSONDecodeError:
        raise ValueError(f"❌ ERREUR : Le fichier {file_path} n'est pas un JSON")

# 📌 Afficher la liste des utilisateurs disponibles
def list_users(file_path):
    data = load_ratings(file_path)
    print("\n📌 Utilisateurs disponibles :")
    for user in data.keys():
        print(f"    → {user}")

# 📌 Calcul du score de Pearson
def pearson_score(dataset, user1, user2):
    if user1 not in dataset or user2 not in dataset:
        list_users(file_path)
        raise ValueError(f"❌ ERREUR : Un des utilisateurs ({user1}, {user2}) n'existe pas")

    common_movies = {item for item in dataset[user1] if item in dataset[user2]}
    num_ratings = len(common_movies)

    if num_ratings == 0:
        return 0

    user1_ratings = np.array([dataset[user1][item] for item in common_movies])
    user2_ratings = np.array([dataset[user2][item] for item in common_movies])

    mean_user1 = np.mean(user1_ratings)
    mean_user2 = np.mean(user2_ratings)

    numerator = np.sum((user1_ratings - mean_user1) * (user2_ratings - mean_user2))
    denominator = np.sqrt(np.sum((user1_ratings - mean_user1) ** 2) * np.sum((user2_ratings - mean_user2) ** 2))

    return 0 if denominator == 0 else numerator / denominator

# 📌 Obtenir les recommandations de films
def get_recommendations(dataset, input_user):
    if input_user not in dataset:
        list_users(file_path)
        raise ValueError(f"❌ ERREUR : L'utilisateur '{input_user}' n'existe pas")

    overall_scores = {}
```

```

similarity_scores = {}

for user in dataset:
    if user == input_user:
        continue

    similarity_score = pearson_score(dataset, input_user, user)

    if similarity_score <= 0:
        continue

    filtered_list = [x for x in dataset[user] if x not in dataset[input_user]]

    for item in filtered_list:
        if item not in overall_scores:
            overall_scores[item] = 0
            similarity_scores[item] = 0

        overall_scores[item] += dataset[user][item] * similarity_score
        similarity_scores[item] += similarity_score

if len(overall_scores) == 0:
    return ['Aucune recommandation possible']

# Générer Les scores normalisés
movie_scores = np.array([[score / similarity_scores[item], item] for item, score in overall_scores.items()])

# Trier par score décroissant
movie_scores = movie_scores[np.argsort(movie_scores[:, 0])[::-1]]

# Extraire Les recommandations de films
movie_recommendations = [movie for _, movie in movie_scores]

return movie_recommendations


# 🚩 Fonction pour exécuter dans Jupyter Notebook
def get_recommendations_notebook(user):
    try:
        data = load_ratings(file_path)
        movies = get_recommendations(data, user)

        print(f"\n📺 Recommandations de films pour {user} :\n")
        if movies == ['Aucune recommandation possible']:
            print("⚠️ Aucune recommandation disponible pour cet utilisateur.")
        else:
            for i, movie in enumerate(movies, 1):
                print(f"{i}. {movie}")

    except FileNotFoundError as e:
        print(e)
    except ValueError as e:
        print(e)

# 🚩 Exemple d'exécution dans Jupyter
get_recommendations_notebook("Chris Duncan")

```

 Recommandations de films pour Chris Duncan :

1. Vertigo
2. Scarface
3. Goodfellas
4. Roman Holiday

Ce code génère des recommandations de films personnalisées pour un utilisateur en fonction des préférences des autres utilisateurs.

Il charge un fichier JSON contenant des notations de films, calcule la similarité de Pearson entre utilisateurs, puis agrège les évaluations des utilisateurs similaires pour recommander les films non encore notés par l'utilisateur cible.

Les films sont triés par pertinence et affichés sous forme de liste. Si aucune recommandation n'est possible, un message d'alerte est affiché.