**1. Approche 1 : Filtrage Collaboratif**

**Contexte**

Le filtrage collaboratif repose sur les préférences des utilisateurs similaires. L'algorithme k-plus proches voisins (k-NN) sera utilisé pour identifier ces similarités.

**Étape 1 : Préparer les données**

1. **Crée une matrice utilisateur-item :**
   * Les lignes représentent les utilisateurs, les colonnes les films, et les valeurs sont les notes données.
   * Utilise pivot\_table de Pandas :

1. user\_item\_matrix = ratings.pivot\_table(index="user\_id", columns="item\_id", values="rating").fillna(0)

2.

1. **Normalise les données :**
   * Standardise les notes pour réduire les biais :

1. from sklearn.preprocessing import StandardScaler

2.

3. scaler = StandardScaler()

4. normalized\_matrix = scaler.fit\_transform(user\_item\_matrix)

5.

**Étape 2 : Implémenter k-NN**

1. **Calculer les similarités entre utilisateurs :** Utilise la distance cosine pour mesurer la similarité.

from sklearn.metrics.pairwise import cosine\_similarity

user\_similarity = cosine\_similarity(normalized\_matrix)

1. **Recommander des films :**
   * Pour un utilisateur donné, identifie les utilisateurs les plus similaires (k voisins).
   * Recommande les films que les voisins ont le mieux notés mais que l'utilisateur n'a pas encore vus.

Exemple :

1. def recommend\_movies(user\_id, k=5):

2. user\_index = user\_item\_matrix.index.get\_loc(user\_id)

3. similar\_users = user\_similarity[user\_index].argsort()[-(k+1):-1]

4. recommended\_movies = []

5.

6. for neighbor in similar\_users:

7. neighbor\_ratings = user\_item\_matrix.iloc[neighbor]

8. unseen\_movies = neighbor\_ratings[user\_item\_matrix.iloc[user\_index] == 0]

9. recommended\_movies.extend(unseen\_movies[unseen\_movies > 3.5].index)

10.

11. return set(recommended\_movies)

12.

**Étape 3 : Évaluer les performances**

1. Divise les données en ensemble d'entraînement et de test.

from sklearn.model\_selection import train\_test\_split

train, test = train\_test\_split(ratings, test\_size=0.2, random\_state=42)

1. Implémente des métriques pour évaluer la précision, le rappel, et le F1-score.

**2. Approche 2 : Basé sur le contenu**

**Contexte**

Ici, les recommandations reposent sur les caractéristiques des films. Des techniques NLP comme TF-IDF ou CountVectorizer seront utilisées pour analyser les descriptions.

**Étape 1 : Préparer les données**

1. **Créer un corpus textuel :**
   * Utilise les titres des films ou les descriptions (si disponibles) comme caractéristiques textuelles.
   * Exemple :

movie\_corpus = movies["movie\_title"] + " " + movies.iloc[:, 5:].astype(str).apply(lambda x: " ".join(x), axis=1)

1. **Appliquer TF-IDF :**
   * Transforme le texte en vecteurs numériques.

from sklearn.feature\_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(stop\_words="english")

movie\_features = vectorizer.fit\_transform(movie\_corpus)

**Étape 2 : Calculer les similarités**

1. **Utilise la distance cosine :**
   * Mesure la similarité entre les films en fonction de leurs vecteurs.

movie\_similarity = cosine\_similarity(movie\_features)

1. **Recommander des films similaires :**
   * Pour un film donné, identifie les films les plus similaires.

def recommend\_similar\_movies(movie\_id, k=5):

movie\_index = movies.index[movies["movie\_id"] == movie\_id].tolist()[0]

similar\_movies = movie\_similarity[movie\_index].argsort()[-(k+1):-1]

return movies.iloc[similar\_movies]["movie\_title"].tolist()

**3. Comparaison des deux approches**

**Étape 1 : Calculer les performances**

* Utilise les ensembles d'entraînement et de test pour évaluer les deux méthodes.
* Implémente les métriques de précision, rappel et F1-score :

from sklearn.metrics import precision\_score, recall\_score, f1\_score

precision = precision\_score(y\_true, y\_pred, average="weighted")

recall = recall\_score(y\_true, y\_pred, average="weighted")

f1 = f1\_score(y\_true, y\_pred, average="weighted")

**Étape 2 : Résumer les résultats**

* Compile les performances des deux approches dans un tableau :

import pandas as pd

results = pd.DataFrame({

"Approche": ["Collaboratif", "Basé sur le contenu"],

"Précision": [precision\_collaborative, precision\_content],

"Rappel": [recall\_collaborative, recall\_content],

"F1-score": [f1\_collaborative, f1\_content]

})

print(results)

* Visualise les résultats sous forme de graphique :

results.plot(x="Approche", kind="bar", figsize=(8, 6))

**Livrables**

1. **Scripts Python :**
   * Un fichier pour chaque approche : collaborative\_filtering.py et content\_based\_filtering.py.
2. **Résultats :**
   * Tableau ou graphique comparant les performances des deux approches.