**Nom : Joye Badou**

Produit final fonctionnel

Sommaire

**Introduction :**

1. **Phase de Collecte des Données**
   1. Chargement des fichiers (ratings.csv, movies.csv, etc.).
   2. Vérification de la structure des fichiers.

* Description des données movie.csv
* Description des données rating.csv
  1. Centralisation des données dans un système structuré ( HDFS).

**II/ Phase de Fiabilisation et Normalisation**

* Nettoyage rating.csv
* Nettoyage movie.csv
* Fusion des donnees

III/ Phase d’Analyse Exploratoire

1. Analyse des Films
2. Analyse des Notes
3. Analyse des Genres
4. Analyse des Utilisateurs
5. Conclusion EDA

A faire …….

**IV et V Phase de Modélisation et d’évaluation**

* 1. Préparation des données pour la matrice utilisateur-item.
  2. Implémentation de KNN pour le filtrage collaboratif.

Modèle User-User : Création

Modèle User-User : Evaluation

Modele item\_item : Creation

Modele item\_item : Evaluation

* 1. Développement d’un modèle basé sur le contenu (genres).

Modèle basé sur le contenu : Création

Modèle basé sur le contenu : Evaluation

* 1. Création d’un modèle hybride combinant les deux approches.

Modele hybrid-Content-User: Creation

Modele hybrid-Content-User: Evaluation

Modele hybrid-content-item: Création

Modele hybrid-content-item: Evaluation

* 1. Optimisation des hyperparamètres pour améliorer les performances.

Modele Item-Item KNN : Optimisation

Modele hybrid-Content-Item KNN : Optimisation

**VI. Produit Final Fonctionnel et deploiement**

1. Script Python interactif permettant à un utilisateur d’entrer son userId et de recevoir des recommandations.
2. API RESTful
3. Guide d'Utilisation du Système de Recommandation de Films
4. Lien Github

**Conclusion**

**Introduction :**

Mon projet portera sur l’analyse et la modélisation des données du jeu de données **MovieLens**, une collection comprenant plus de 100000 notations et 943 utilisateurs et environ 1682 films. Malgré l'ambition initiale de travailler sur le dataset MovieLens contenant 33 millions de notations, nous avons opté pour le dataset de 100 000 notations (disponible sur Kaggle à l'adresse [ce lien](https://www.kaggle.com/datasets/prajitdatta/movielens-100k-dataset/data)) pour plusieurs raisons. Tout d'abord, le dataset complet de 33 millions de notations nécessite des ressources matérielles (mémoire vive, temps de calcul) considérablement supérieures, ce qui complique la manipulation et le traitement des données, surtout dans un environnement de développement standard. Ensuite, le dataset 100 k offre un compromis optimal, il est suffisamment riche pour capturer des tendances et générer des recommandations fiables, tout en restant maniable et permettant une itération rapide des modèles et analyses.

Par ailleurs, bien que l'étude des tags ait été envisagée, elle a été finalement écartée pour cette version du projet en raison de la complexité supplémentaire qu'elle imposerait au processus de modélisation. En se concentrant sur les fichiers movies et ratings, nous avons pu bâtir un système de recommandation solide et efficace, qui répond aux objectifs du projet sans surcharge de complexité ni de contraintes matérielles. Dans cette partie du projet nous aborderons : la Phase de Collecte des Données, la Phase de Fiabilisation et Normalisation, la Phase d’Analyse Exploratoire et nous feront un rapport à mi-parcours .

1. **Phase de Collecte des Données**
   1. **Chargement des fichiers (ratings.csv, movies.csv, etc.).**

Les fichiers sources ont été téléchargés à partir de Kaggle à l'adresse [ce lien](https://www.kaggle.com/datasets/prajitdatta/movielens-100k-dataset/data), c’est un fichier Zip qui contient plusieurs données. Nous nous intéresserons que de u.data que j’ai transformé ( le format) et renommé en **rating.csv** qui contient les notes des utilisateurs sur les films, et de u.item que j’ai transformé ( le format) et renommé en **moving.csv**. Voici les différents chemins des fichier :

"C:\Users\joyeb\Downloads\archive (4).zip"

"C:\projet\_work\Data-source\movie.csv"

"C:\projet\_work\Data-source\rating.csv"

Voici le code de conversion des fichiers u.data et u.item en respectivement en rating.csv et movie.csv".

**Code** : python convert\_to\_csv.py

import pandas as pd

**# Chemins des fichiers d'entrée et de sortie**

u\_data\_path = r"C:\projet\_work\Data-source\u.data"

u\_item\_path = r"C:\projet\_work\Data-source\u.item"

ratings\_output\_path = r"C:\projet\_work\Data-source\rating.csv"

movies\_output\_path = r"C:\projet\_work\Data-source\movie.csv"

**# Conversion de u.data en rating.csv**

def convert\_u\_data\_to\_csv(input\_path, output\_path):

try:

**# Chargement des données avec les colonnes appropriées**

column\_names = ['user\_id', 'movie\_id', 'rating', 'timestamp']

u\_data = pd.read\_csv(input\_path, sep='\t', names=column\_names, header=None)

**# Enregistrement dans un fichier CSV**

u\_data.to\_csv(output\_path, index=False)

print(f" Conversion de {input\_path} en {output\_path} réussie.")

except Exception as e:

print(f"❌ Erreur lors de la conversion de {input\_path} : {e}")

**# Conversion de u.item en movie.csv**

def convert\_u\_item\_to\_csv(input\_path, output\_path):

try:

**# Chargement des données avec les colonnes appropriées**

column\_names = [

'movie\_id', 'movie\_title', 'release\_date', 'video\_release\_date', 'IMDb\_URL',

'unknown', 'Action', 'Adventure', 'Animation', "Children's", 'Comedy', 'Crime',

'Documentary', 'Drama', 'Fantasy', 'Film-Noir', 'Horror', 'Musical', 'Mystery',

'Romance', 'Sci-Fi', 'Thriller', 'War', 'Western'

]

u\_item = pd.read\_csv(input\_path, sep='|', names=column\_names, encoding='latin-1', header=None)

**# Enregistrement dans un fichier CSV**

u\_item.to\_csv(output\_path, index=False)

print(f" Conversion de {input\_path} en {output\_path} réussie.")

except Exception as e:

print(f"❌ Erreur lors de la conversion de {input\_path} : {e}")

**# Exécution des conversions**

convert\_u\_data\_to\_csv(u\_data\_path, ratings\_output\_path)

convert\_u\_item\_to\_csv(u\_item\_path, movies\_output\_path)

**Execution du code**

**PS C:\Users\joyeb> cd C:\projet\_work\Script**

**PS C:\projet\_work\Script> python convert\_to\_csv.py**

Conversion de C:\projet\_work\Data-source\u.data en C:\projet\_work\Data-source\rating.csv réussie.

Conversion de C:\projet\_work\Data-source\u.item en C:\projet\_work\Data-source\movie.csv réussie.

* 1. **Vérification de la structure des fichiers.**
* **Description des données movie.csv**

Le code : **analyse\_movie.py**

import pandas as pd

**# Définition du chemin du fichier**

movies\_path = r"C:\pyprojet\processed\_data\movie.csv"

**# Chargement des données**

try:

movies\_df = pd.read\_csv(movies\_path)

print(" Fichier chargé avec succès.")

except FileNotFoundError:

print(f"❌ Erreur : Fichier introuvable à l'emplacement {movies\_path}.")

exit()

except Exception as e:

print(f"❌ Une erreur est survenue lors du chargement : {e}")

exit()

**# Analyse des données**

num\_rows, num\_cols = movies\_df.shape # Nombre de lignes et de colonnes

column\_info = movies\_df.dtypes # Types des colonnes

**# Analyse des titres**

num\_unique\_titles = movies\_df['movie\_title'].nunique() # Nombre de titres uniques

**# Analyse des genres (comptage global des genres, y compris 'unknown')**

genre\_columns = [

'unknown', 'Action', 'Adventure', 'Animation', "Children's", 'Comedy', 'Crime',

'Documentary', 'Drama', 'Fantasy', 'Film-Noir', 'Horror', 'Musical', 'Mystery',

'Romance', 'Sci-Fi', 'Thriller', 'War', 'Western'

]

num\_movies\_per\_genre = movies\_df[genre\_columns].sum() # Nombre de films par genre

**# Analyse des dates**

num\_missing\_release\_dates = movies\_df['release\_date'].isnull().sum() # Dates manquantes

num\_missing\_video\_dates = movies\_df['video\_release\_date'].isnull().sum() # Dates de sortie vidéo manquantes

**# Aperçu des trois premières lignes**

first\_three\_rows = movies\_df.head(3)

**# Résultats**

print("\n Analyse du fichier movie.csv ")

print(f"Nombre de lignes : {num\_rows}")

print(f"Nombre de colonnes : {num\_cols}\n")

print("Nom et type des colonnes :")

print(column\_info, "\n")

print(f"Nombre de titres uniques : {num\_unique\_titles}\n")

print(" Nombre de films par genre (y compris 'unknown') :")

print(num\_movies\_per\_genre, "\n")

print(f"Nombre de dates de sortie manquantes : {num\_missing\_release\_dates}")

print(f"Nombre de dates de sortie vidéo manquantes : {num\_missing\_video\_dates}\n")

print("Aperçu des trois premières lignes :")

print(first\_three\_rows)

**Exécution du code :**

**PS C:\projet\_work\Script> python analyse\_movie.py**

Nombre de lignes : 1682

Nombre de colonnes : 24

Types des colonnes :

movie\_id int64

movie\_title object

release\_date object

video\_release\_date float64

IMDb\_URL object

unknown int64

Action int64

Adventure int64

Animation int64

Children's int64

Comedy int64

Crime int64

Documentary int64

Drama int64

Fantasy int64

Film-Noir int64

Horror int64

Musical int64

Mystery int64

Romance int64

Sci-Fi int64

Thriller int64

War int64

Western int64

dtype: object

Nombre de titres uniques : 1664

Nombre de films par genre :

unknown : 2

Action : 251

Adventure : 135

Animation : 42

Children's : 122

Comedy : 505

Crime : 109

Documentary : 50

Drama : 725

Fantasy : 22

Film-Noir : 24

Horror : 92

Musical : 56

Mystery : 61

Romance : 247

Sci-Fi : 101

Thriller : 251

War : 71

Western : 27

Nombre de dates de sortie manquantes : 1

Nombre de dates de sortie vidéo manquantes : 1682

Aperçu des trois premières lignes :

movie\_id movie\_title release\_date video\_release\_date ... Sci-Fi Thriller War Western

0 1 Toy Story (1995) 01-Jan-1995 NaN ... 0 0 0 0

1 2 GoldenEye (1995) 01-Jan-1995 NaN ... 0 1 0 0

2 3 Four Rooms (1995) 01-Jan-1995 NaN ... 0 1 0 0

[3 rows x 24 columns]

**Analyses des résultats**

* Nombre de lignes : 1682
* Nombre de colonnes : 24
* Types de colonnes :
  + movie\_id est un entier (int64)
  + movie\_title et release\_date sont des objets (object), ce qui signifie qu'ils sont probablement des chaînes de caractères.
  + video\_release\_date est un flottant (float64)
  + Les colonnes de genre (unknown, Action, etc.)
* Nombre de titres uniques : 1664
* Distribution des genres :
  + Le genre le plus courant est le Drama (725 films), suivi de Comedy (505 films).
  + Action et Thriller ont également une fréquence élevée (251 films chacun).
  + Les genres les moins courants sont Fantasy (22 films) et Western (27 films).
  + Il y a 2 films classés comme unknown.
* Dates de sortie manquantes : 1
* Dates de sortie vidéo manquantes : 1682
* Exemple de données :
  + Les trois premières lignes montrent des exemples de films avec leurs informations de base et leurs genres.
  + La colonne video\_release\_date contient des valeurs NaN
  + Les colonnes de genre contiennent des valeurs binaires (0 ou 1)
* **Description des données rating.csv**

**Code : python analyse\_rating.py**

import pandas as pd

**# Chargement du fichier rating.csv**

ratings = pd.read\_csv(r'C:\projet\_work\Data-source\rating.csv')

**# Aperçu des premières lignes**

print("Aperçu des premières lignes :")

print(ratings.head(), "\n")

**# Nombre total de lignes**

nb\_lignes = ratings.shape[0]

print(f"Nombre total de lignes : {nb\_lignes}")

**# Nombre de colonnes**

nb\_colonnes = ratings.shape[1]

print(f"Nombre de colonnes : {nb\_colonnes}\n")

**# Détection des notes aberrantes (outliers) dans 'rating'**

Q1 = ratings['rating'].quantile(0.25)

Q3 = ratings['rating'].quantile(0.75)

IQR = Q3 - Q1

borne\_inf = Q1 - 1.5 \* IQR

borne\_sup = Q3 + 1.5 \* IQR

outliers = ratings[(ratings['rating'] < borne\_inf) | (ratings['rating'] > borne\_sup)]

nb\_outliers = outliers.shape[0]

print(f"Nombre de notes aberrantes détectées (outliers) : {nb\_outliers}\n")

**# Valeurs manquantes par colonne**

valeurs\_manquantes = ratings.isna().sum()

print("Valeurs manquantes par colonne :")

print(valeurs\_manquantes, "\n")

**# Nombre total de valeurs manquantes**

nb\_total\_manquantes = valeurs\_manquantes.sum()

print(f"Nombre total de valeurs manquantes : {nb\_total\_manquantes}\n")

**# Nombre d'entrées utilisateur (user\_id) en double**

nb\_user\_duplicates = ratings.duplicated(subset='user\_id').sum()

print(f"Nombre d'entrées utilisateur (user\_id) en double : {nb\_user\_duplicates}")

**# Nombre d'entrées film (movie\_id) en double**

nb\_movie\_duplicates = ratings.duplicated(subset='movie\_id').sum()

print(f"Nombre d'entrées film (movie\_id) en double : {nb\_movie\_duplicates}")

**# Nombre de lignes entièrement dupliquées**

nb\_total\_duplicates = ratings.duplicated().sum()

print(f"Nombre de lignes entièrement dupliquées : {nb\_total\_duplicates}\n")

**# Aperçu des entrées utilisateur en double**

print("Aperçu des entrées utilisateur en double :")

print(ratings[ratings.duplicated(subset='user\_id', keep=False)].sort\_values('user\_id').head(), "\n")

**# Aperçu des entrées film en double**

print("Aperçu des entrées film en double :")

print(ratings[ratings.duplicated(subset='movie\_id', keep=False)].sort\_values('movie\_id').head())

**Exécution du code**

**PS C:\projet\_work\Script> python analyse\_rating.py**

Aperçu des premières lignes :

user\_id movie\_id rating timestamp

0 196 242 3 881250949

1 186 302 3 891717742

2 22 377 1 878887116

3 244 51 2 880606923

4 166 346 1 886397596

Nombre total de lignes : 100000

Nombre de colonnes : 4

Nombre de notes aberrantes détectées (outliers) : 6110

Valeurs manquantes par colonne :

user\_id 0

movie\_id 0

rating 0

timestamp 0

dtype: int64

Nombre total de valeurs manquantes : 0

Nombre d'entrées utilisateur (user\_id) en double : 99057

Nombre d'entrées film (movie\_id) en double : 98318

Nombre de lignes entièrement dupliquées : 0

Aperçu des entrées utilisateur en double :

user\_id movie\_id rating timestamp

92049 1 28 4 875072173

62820 1 203 4 878542231

34165 1 75 4 878543238

972 1 117 3 874965739

15764 1 196 5 874965677

Aperçu des entrées film en double :

user\_id movie\_id rating timestamp

48383 749 1 4 881602577

74132 764 1 4 876244181

21683 256 1 5 882150980

89335 922 1 5 891448551

86548 881 1 4 876535796

PS C:\projet\_work\Script> python clean\_movies.py

Nombre de lignes après nettoyage : 1681

Aperçu des 3 premières lignes après nettoyage :

movie\_id movie\_title release\_date Action Adventure Animation Children's ... Musical Mystery Romance Sci-Fi Thriller War Western

0 1 Toy Story (1995) 01-Jan-1995 0 0 1 1 ... 0 0 0 0 0 0 0

1 2 GoldenEye (1995) 01-Jan-1995 1 1 0 0 ... 0 0 0 0 1 0 0

2 3 Four Rooms (1995) 01-Jan-1995 0 0 0 0 ... 0 0 0 0 1 0 0

[3 rows x 21 columns]

**Analyses des résultats**

* **Aperçu des premières lignes :**
  + Les premières lignes montrent la structure du fichier rating.csv, avec les colonnes user\_id, movie\_id, rating, et timestamp.
  + Les notes (rating) varient de 1 à 5, ce qui est attendu.
* **Nombre total de lignes : 100000**
* **Nombre de colonnes : 4**
* **Nombre de notes aberrantes détectées (outliers) : 6110**
  + il y a des notes qui s'écartent significativement de la distribution générale.
* **Valeurs manquantes par colonne :**
  + Aucune valeur manquante n'est détectée dans aucune colonne.
* **Nombre total de valeurs manquantes : 0**
* **Nombre d'entrées utilisateur (user\_id) en double : 99057**
  + Cela indique que la plupart des utilisateurs ont donné plusieurs notes.
* **Nombre d'entrées film (movie\_id) en double : 98318**
  + De même, la plupart des films ont reçu plusieurs notes.
* **Nombre de lignes entièrement dupliquées : 0**
  + Il n'y a pas de lignes entièrement dupliquées, ce qui est positif.
* **Aperçu des entrées utilisateur en double :**
  + Les exemples montrent que l'utilisateur 1 a donné plusieurs notes à différents films.
* **Aperçu des entrées film en double :**
  + Les exemples montrent que le film 1 a reçu plusieurs notes de différents utilisateurs.
  1. **Centralisation des données dans un système structuré ( HDFS).**

Je vais charger les données fusionnées dans hdfs , comme je l’avais annoncé, mais je vais faire tout le reste du travail , c’est-à-dire l’EDA, la modelisation etc.. en local.

* **Ouverture du système hdfs**

**PS C:\Users\joyeb> cd C:\hadoop-3.3.5\sbin**

**PS C:\hadoop-3.3.5\sbin> ./start-dfs.cmd**

**PS C:\hadoop-3.3.5\sbin> ./start-yarn.cmd**

starting yarn daemons

* **Verifications des different noeuds**

**PS C:\hadoop-3.3.5\sbin> jps**

45888 NodeManager

44452 NameNode

45240 ResourceManager

32812 Jps

42028 DataNode

* **Chargement des donnees dans hdfs**

**PS C:\hadoop-3.3.5\sbin> hdfs dfs -put C:\projet\_work\Data-source\merged\_final\_data.csv /user/joyeb/merged\_final\_data.csv**

* **Taille et nombre de blocs**

**PS C:\hadoop-3.3.5\sbin> hdfs dfs -du -h /user/joyeb/merged\_final\_data.csv**

7.9 M 7.9 M /user/joyeb/merged\_final\_data.csv

**PS C:\hadoop-3.3.5\sbin> hdfs fsck /user/joyeb/merged\_final\_data.csv -files -blocks -locations**

Connecting to namenode via http://localhost:9870/fsck?ugi=joyeb&files=1&blocks=1&locations=1&path=%2Fuser%2Fjoyeb%2Fmerged\_final\_data.csv

FSCK started by joyeb (auth:SIMPLE) from /127.0.0.1 for path /user/joyeb/merged\_final\_data.csv at Thu Mar 20 02:12:02 EDT 2025

/user/joyeb/merged\_final\_data.csv 8247603 bytes, replicated: replication=1, 1 block(s): OK

0. BP-991879290-10.0.0.217-1737293331093:blk\_1073742137\_1313 len=8247603 Live\_repl=1 [DatanodeInfoWithStorage[127.0.0.1:9866,DS-503ca48d-de43-4221-9ede-e693facea981,DISK]]

Status: HEALTHY

Number of data-nodes: 1

Number of racks: 1

Total dirs: 0

Total symlinks: 0

Replicated Blocks:

Total size: 8247603 B

Total files: 1

Total blocks (validated): 1 (avg. block size 8247603 B)

Minimally replicated blocks: 1 (100.0 %)

Over-replicated blocks: 0 (0.0 %)

Under-replicated blocks: 0 (0.0 %)

Mis-replicated blocks: 0 (0.0 %)

Default replication factor: 1

Average block replication: 1.0

Missing blocks: 0

Corrupt blocks: 0

Missing replicas: 0 (0.0 %)

Blocks queued for replication: 0

Erasure Coded Block Groups:

Total size: 0 B

Total files: 0

Total block groups (validated): 0

Minimally erasure-coded block groups: 0

Over-erasure-coded block groups: 0

Under-erasure-coded block groups: 0

Unsatisfactory placement block groups: 0

Average block group size: 0.0

Missing block groups: 0

Corrupt block groups: 0

Missing internal blocks: 0

Blocks queued for replication: 0

FSCK ended at Thu Mar 20 02:12:02 EDT 2025 in 10 milliseconds

The filesystem under path '/user/joyeb/merged\_final\_data.csv' is HEALTHY

Bref commentaire : Le rapport montre que le fichier /user/joyeb/merged\_final\_data.csv occupe environ 7,9 Mo dans HDFS. Le système a stocké le fichier dans un unique bloc, avec une réplication minimale (facteur de réplication de 1), et aucun problème n'a été détecté

***Passons à la phase de fiabilisation et normalisation***

**II/ Phase de Fiabilisation et Normalisation**

Dans cette phase nous allons en gros nettoyer les donner.

* ***Nettoyage de rating***

Pour ratings nous allons opérer les actions suivantes :

**Supprimer les lignes entièrement doublées :**

* **Action :** Identifier et éliminer les enregistrements identiques présents dans le jeu de données.
* **Pourquoi :** Les doublons peuvent fausser les analyses et les résultats du modèle en surreprésentant certaines interactions utilisateur-film.

**Supprimer la colonne timestamp :**

* **Action :** Retirer la colonne contenant les horodatages des notes.
* **Pourquoi :** nous ne l’utiliserons pas pour l’analyse.

**Supprimer les films ayant moins de 3 notes :**

* **Action :** Filtrer et retirer les films qui ont reçu un nombre insuffisant de notes.
* **Pourquoi :** Les films avec très peu de notes peuvent ne pas fournir suffisamment d'informations pour des recommandations fiables.

**Supprimer les utilisateurs ayant noté moins de 3 films :**

* **Action :** Filtrer et retirer les utilisateurs qui ont évalué un nombre insuffisant de films.
* **Pourquoi :** Les utilisateurs avec très peu de notes peuvent ne pas fournir un profil suffisant pour des recommandations personnalisées.

**Supprimer les utilisateurs ayant un nombre excessif de films notés (seuil 99e percentile) :**

* **Action :** Identifier et retirer les utilisateurs qui ont noté un nombre anormalement élevé de films.
* **Pourquoi :** Ces utilisateurs peuvent être des anomalies (par exemple, des robots) ou avoir un comportement de notation atypique qui pourrait biaiser le modèle. Le 99e percentile représente la valeur au-dessus de laquelle se trouvent 1 % des données. En l'utilisant comme seuil, on élimine les valeurs extrêmement élevées.

**Supprimer les utilisateurs ayant une variance des notes < 0.1 :**

* **Action :** Identifier et retirer les utilisateurs dont les notes varient très peu.
* **Pourquoi :** Ces utilisateurs peuvent noter tous les films de manière similaire, ce qui ne fournit pas d'informations utiles pour la personnalisation.

**Supprimer les films ayant uniquement des notes à 5.0 ou 1.0 :**

* **Action :** Filtrer et retirer les films qui n'ont reçu que des notes extrêmes.
* **Pourquoi :** Ces films peuvent être des anomalies ou avoir un biais de notation qui pourrait fausser les recommandations.

**Créer une colonne rating\_normalized (normalisation Min-Max) :**

* **Action :** Ajouter une nouvelle colonne contenant les notes normalisées à l'échelle de 0 à 1.
* **Pourquoi :** La normalisation permet de mettre toutes les notes sur une échelle commune, ce qui peut améliorer les performances de certains algorithmes.

Le code : **python clean\_ratings.py**

import pandas as pd

import numpy as np

**# Chargement du fichier rating.csv**

ratings = pd.read\_csv(r'C:\projet\_work\Data-source\rating.csv')

**# Suppression des lignes entièrement doublées**

nb\_doublons = ratings.duplicated().sum()

ratings\_clean = ratings.drop\_duplicates()

print(f"Nombre de lignes entièrement doublées supprimées : {nb\_doublons}")

**# Suppression de la colonne timestamp**

ratings\_clean = ratings\_clean.drop(columns=['timestamp'])

**# Suppression des films ayant moins de 3 notes**

film\_counts = ratings\_clean['movie\_id'].value\_counts()

films\_a\_supprimer = film\_counts[film\_counts < 3].index

nb\_films\_supprimes = len(films\_a\_supprimer)

ratings\_clean = ratings\_clean[~ratings\_clean['movie\_id'].isin(films\_a\_supprimer)]

print(f"Nombre de films supprimés (moins de 3 notations) : {nb\_films\_supprimes}")

**# Suppression des utilisateurs ayant noté moins de 3 films**

user\_counts = ratings\_clean['user\_id'].value\_counts()

users\_a\_supprimer = user\_counts[user\_counts < 3].index

nb\_users\_supprimes = len(users\_a\_supprimer)

ratings\_clean = ratings\_clean[~ratings\_clean['user\_id'].isin(users\_a\_supprimer)]

print(f"Nombre d'utilisateurs supprimés (moins de 3 notes) : {nb\_users\_supprimes}")

**# Suppression des utilisateurs ayant un nombre excessif de films notés (seuil 99e percentile)**

user\_counts\_updated = ratings\_clean['user\_id'].value\_counts()

seuil\_99 = np.percentile(user\_counts\_updated, 99)

users\_excessifs = user\_counts\_updated[user\_counts\_updated > seuil\_99].index

nb\_users\_excessifs\_supprimes = len(users\_excessifs)

ratings\_clean = ratings\_clean[~ratings\_clean['user\_id'].isin(users\_excessifs)]

print(f"Nombre d'utilisateurs supprimés (nombre excessif de notations, seuil 99e percentile) : {nb\_users\_excessifs\_supprimes}")

**# Suppression des utilisateurs ayant une variance des notes < 0.1**

user\_variances = ratings\_clean.groupby('user\_id')['rating'].var()

users\_variance\_faible = user\_variances[user\_variances < 0.1].index

nb\_users\_variance\_supprimes = len(users\_variance\_faible)

ratings\_clean = ratings\_clean[~ratings\_clean['user\_id'].isin(users\_variance\_faible)]

print(f"Nombre d'utilisateurs supprimés (variance des notes < 0.1) : {nb\_users\_variance\_supprimes}")

**# Suppression des films ayant uniquement des notes à 5.0 ou 1.0**

film\_notes\_unique = ratings\_clean.groupby('movie\_id')['rating'].nunique()

films\_notes\_uniques = film\_notes\_unique[film\_notes\_unique == 1].index

films\_notes\_uniques\_extremes = []

for film\_id in films\_notes\_uniques:

note\_unique = ratings\_clean[ratings\_clean['movie\_id'] == film\_id]['rating'].iloc[0]

if note\_unique in [1.0, 5.0]:

films\_notes\_uniques\_extremes.append(film\_id)

nb\_films\_notes\_extremes\_supprimes = len(films\_notes\_uniques\_extremes)

ratings\_clean = ratings\_clean[~ratings\_clean['movie\_id'].isin(films\_notes\_uniques\_extremes)]

print(f"Nombre de films supprimés (uniquement notes à 1.0 ou 5.0) : {nb\_films\_notes\_extremes\_supprimes}")

**# Création d'une colonne rating\_normalized (normalisation Min-Max)**

min\_rating = ratings\_clean['rating'].min()

max\_rating = ratings\_clean['rating'].max()

ratings\_clean['rating\_normalized'] = (ratings\_clean['rating'] - min\_rating) / (max\_rating - min\_rating)

**# Résumé final après nettoyage**

nb\_lignes\_finales = ratings\_clean.shape[0]

print(f"\nNombre total de lignes restantes après nettoyage : {nb\_lignes\_finales}")

**# Aperçu des 3 premières lignes finales après nettoyage**

print("\nAperçu des 3 premières lignes après nettoyage :")

print(ratings\_clean.head(3))

**# Sauvegarde du fichier nettoyé**

ratings\_clean.to\_csv(r'C:\projet\_work\Data-source\clean\_rating.csv', index=False)

**Execution du code**

**PS C:\projet\_work\Script> python clean\_ratings.py**

Nombre de lignes entièrement doublées supprimées : 0

Nombre de films supprimés (moins de 3 notations) : 209

Nombre d'utilisateurs supprimés (moins de 3 notes) : 0

Nombre d'utilisateurs supprimés (nombre excessif de notations, seuil 99e percentile) : 10

Nombre d'utilisateurs supprimés (variance des notes < 0.1) : 0

Nombre de films supprimés (uniquement notes à 1.0 ou 5.0) : 11

Nombre total de lignes restantes après nettoyage : 94285

Aperçu des 3 premières lignes après nettoyage :

user\_id movie\_id rating rating\_normalized

0 196 242 3 0.5

1 186 302 3 0.5

2 22 377 1 0.0

* ***Nettoyage de movie.csv***

*Nous allons procéder aux traitements suivants* :

**Supprimer les colonnes : ('video\_release\_date', 'unknown', 'IMDb\_URL')**

* **Action :** Retirer les colonnes qui contiennent des informations superflues ou inutiles pour l'analyse.
* **Pourquoi :**
  + video\_release\_date : Comme vu précédemment, cette colonne est entièrement remplie de valeurs manquantes (NaN), ce qui la rend inutile.
  + unknown : La colonne « unknown » représente un genre inconnu. La proportion de films dans cette catégorie etant faible, nous décidons de la supprimer pour simplifier l'analyse.
  + IMDb\_URL : n’est pas nécessaire pour notre projet

**Supprimer les lignes entièrement dupliquées en gardant une seule occurrence :**

* **Action :** Identifier et éliminer les lignes qui sont exactement identiques, en conservant une seule occurrence de chaque ligne unique.
* **Pourquoi :**
  + Les doublons peuvent fausser les analyses en surreprésentant certains films.
  + La suppression des doublons garantit que chaque film est représenté une seule fois dans le jeu de données, ce qui améliore la précision des analyses.

Le code

import pandas as pd

**# Chargement du fichier movie.csv**

movies = pd.read\_csv(r'C:\projet\_work\Data-source\movie.csv')

**# Suppression des lignes avec dates de sortie manquantes**

movies\_clean = movies.dropna(subset=['release\_date'])

**# Suppression des colonnes 'video\_release\_date', 'unknown', 'IMDb\_URL'**

movies\_clean = movies\_clean.drop(columns=['video\_release\_date', 'unknown', 'IMDb\_URL'])

**# Suppression des lignes entièrement dupliquées en gardant une seule occurrence**

movies\_clean = movies\_clean.drop\_duplicates(keep='first')

**# Afficher le bilan après nettoyage**

nb\_lignes\_finales = movies\_clean.shape[0]

print(f"Nombre de lignes après nettoyage : {nb\_lignes\_finales}\n")

**# Aperçu des trois premières lignes après nettoyage**

print("Aperçu des 3 premières lignes après nettoyage :")

print(movies\_clean.head(3))

**# Sauvegarder le fichier nettoyé**

movies\_clean.to\_csv(r'C:\projet\_work\Data-source\movie\_clean.csv', index=False)

**PS C:\projet\_work\Script> python clean\_movies.py**

Nombre de lignes après nettoyage : 1681

Aperçu des 3 premières lignes après nettoyage :

movie\_id movie\_title release\_date Action Adventure Animation Children's ... Musical Mystery Romance Sci-Fi Thriller War Western

0 1 Toy Story (1995) 01-Jan-1995 0 0 1 1 ... 0 0 0 0 0 0 0

1 2 GoldenEye (1995) 01-Jan-1995 1 1 0 0 ... 0 0 0 0 1 0 0

2 3 Four Rooms (1995) 01-Jan-1995 0 0 0 0 ... 0 0 0 0 1 0 0

[3 rows x 21 columns]

* ***Fusion des données nettoyées***

Nous allons maintenant procéder à la fusion des données nettoyées.

Dans les 3 scripts ci-dessus nous allons :

* Fusionner les données nettoyées
* Vérifier les movie\_id dans ratings non présents dans movies afin de s’assurer que toutes les notes se rapportent à des films existants.
* Vérifier les movie\_id dans movies non présents dans ratings afin d’Identifier les identifiants de films présents dans le jeu de données movies qui n'ont reçu aucune note dans le jeu de données ratings.
* Retirer les lignes du jeu de données ratings qui contiennent des identifiants de films non présents dans movie pour éliminer les notes qui ne peuvent pas être liées à des films existants.
* Nettoyage movies : supprimer les films sans notation afin de se concentrer sur les films qui ont été évalués par les utilisateurs.
* Et enfin fusionner immédiatement les deux fichiers nettoyés :

Code : **fusion\_donnees.py**

import pandas as pd

**# Chargement des fichiers nettoyés**

movies = pd.read\_csv(r'C:\projet\_work\Data-source\movie\_clean.csv')

ratings = pd.read\_csv(r'C:\projet\_work\Data-source\clean\_rating.csv')

**# Fusion des deux fichiers sur la colonne 'movie\_id'**

merged\_data = pd.merge(ratings, movies, on='movie\_id', how='inner')

**# Vérification rapide après fusion**

nb\_lignes\_fusionnees = merged\_data.shape[0]

print(f"Nombre total de lignes après fusion : {nb\_lignes\_fusionnees}\n")

**# Aperçu des 3 premières lignes fusionnées**

print("Aperçu des 3 premières lignes après fusion :")

print(merged\_data.head(3))

**# Sauvegarde du fichier fusionné clairement nommé 'merged\_data.csv'**

merged\_data.to\_csv(r'C:\projet\_work\Data-source\merged\_data.csv', index=False)

**Exécution du code**

**PS C:\projet\_work\Script> python fusion\_donnees.py**

Nombre total de lignes après fusion : 94276

Aperçu des 3 premières lignes après fusion :

user\_id movie\_id rating rating\_normalized movie\_title release\_date ... Mystery Romance Sci-Fi Thriller War Western

0 196 242 3 0.5 Kolya (1996) 24-Jan-1997 ... 0 0 0 0 0 0

1 186 302 3 0.5 L.A. Confidential (1997) 01-Jan-1997 ... 1 0 0 1 0 0

2 22 377 1 0.0 Heavyweights (1994) 01-Jan-1994 ... 0 0 0 0 0 0

[3 rows x 24 columns]

Code : **python verification.py**

import pandas as pd

**# Chargement des fichiers nettoyés**

movies = pd.read\_csv(r'C:\projet\_work\Data-source\movie\_clean.csv')

ratings = pd.read\_csv(r'C:\projet\_work\Data-source\clean\_rating.csv')

**# Vérifier les movie\_id dans ratings non présents dans movies**

movies\_in\_ratings\_not\_in\_movies = ratings[~ratings['movie\_id'].isin(movies['movie\_id'])]

nb\_movies\_absents\_dans\_movies = movies\_in\_ratings\_not\_in\_movies['movie\_id'].nunique()

print(f"Nombre de movie\_id présents dans ratings mais absents dans movies : {nb\_movies\_absents\_dans\_movies}")

**# Afficher ces movie\_id sans correspondance dans movies**

print("\nListe des movie\_id présents dans ratings mais absents dans movies :")

print(movies\_in\_ratings\_not\_in\_movies['movie\_id'].unique())

**# Vérifier les movie\_id dans movies non présents dans ratings**

movies\_not\_in\_ratings = movies[~movies['movie\_id'].isin(ratings['movie\_id'])]

nb\_movies\_absents\_dans\_ratings = movies\_not\_in\_ratings['movie\_id'].nunique()

print(f"\nNombre de movie\_id présents dans movies mais absents dans ratings : {nb\_movies\_absents\_dans\_ratings}")

**# Afficher ces movie\_id sans correspondance dans ratings**

print("\nListe des movie\_id présents dans movies mais absents dans ratings :")

print(movies\_not\_in\_ratings['movie\_id'].unique())

**Exécution du code**

**PS C:\projet\_work\Script> python verification.py**

Nombre de movie\_id présents dans ratings mais absents dans movies : 10

Liste des movie\_id présents dans ratings mais absents dans movies : [267]

Nombre de movie\_id présents dans movies mais absents dans ratings : 221

Liste des movie\_id présents dans movies mais absents dans ratings :

[ 119 314 437 439 599 600 677 711 784 814 830 850 852 857

858 897 907 913 …………………………………………………………

**Code : nettoyage\_et\_fusion\_finale.py**

# Chargement initial des données nettoyées

movies = pd.read\_csv(r'C:\projet\_work\Data-source\movie\_clean.csv')

ratings = pd.read\_csv(r'C:\projet\_work\Data-source\clean\_rating.csv')

**# --- Nettoyage ratings : suppression des movie\_id sans correspondance ---**

ratings\_final = ratings[ratings['movie\_id'].isin(movies['movie\_id'])]

nb\_lignes\_supprimees\_rating = ratings.shape[0] - ratings\_final.shape[0]

print(f"Lignes supprimées de ratings (movie\_id sans correspondance) : {nb\_lignes\_supprimees\_rating}")

**# --- Nettoyage movies : suppression des films sans notation ---**

movies\_final = movies[movies['movie\_id'].isin(ratings\_final['movie\_id'])]

nb\_films\_supprimes\_movie = movies.shape[0] - movies\_final.shape[0]

print(f"Films supprimés dans movies (aucune notation associée) : {nb\_films\_supprimes\_movie}\n")

**# Vérification des résultats finaux**

print(f"Nombre final de lignes dans ratings : {ratings\_final.shape[0]}")

print(f"Nombre final de films dans movies : {movies\_final.shape[0]}\n")

**# --- Fusion des deux fichiers nettoyés ---**

merged\_final\_data = pd.merge(ratings\_final, movies\_final, on='movie\_id', how='inner')

**# Vérification après fusion finale**

print(f"Nombre total de lignes après fusion : {merged\_final\_data.shape[0]}\n")

print("Aperçu des 3 premières lignes fusionnées :")

print(merged\_final\_data.head(3))

**# Sauvegarde définitive des fichiers nettoyés finaux et fusionnés**

movies\_final.to\_csv(r'C:\projet\_work\Data-source\clean\_movie.csv', index=False)

ratings\_final.to\_csv(r'C:\projet\_work\Data-source\clean\_rating.csv', index=False)

merged\_final\_data.to\_csv(r'C:\projet\_work\Data-source\merged\_final\_data.csv', index=False)

**Execution du code :**

**PS C:\projet\_work\Script> python nettoyage\_et\_fusion\_finale.py**

Lignes supprimées de ratings (movie\_id sans correspondance) : 9

Films supprimés dans movies (aucune notation associée) : 221

Nombre final de lignes dans ratings : 94276

Nombre final de films dans movies : 1460

Nombre total de lignes après fusion : 94276

Aperçu des 3 premières lignes fusionnées :

user\_id movie\_id rating rating\_normalized movie\_title release\_date ... Mystery Romance Sci-Fi Thriller War Western

0 196 242 3 0.5 Kolya (1996) 24-Jan-1997 ... 0 0 0 0 0 0

1 186 302 3 0.5 L.A. Confidential (1997) 01-Jan-1997 ... 1 0 0 1 0 0

2 22 377 1 0.0 Heavyweights (1994) 01-Jan-1994 ... 0 0 0 0 0 0

[3 rows x 24 columns]

**III/ Phase d’Analyse Exploratoire**

L'objectif de cette Exploration de Données est de plonger au cœur de nos données de films et de notes pour en extraire des informations précieuses. En comprenant les tendances, les distributions et les relations au sein de nos données, nous pourrons construire un système de recommandation de films plus performant et mieux adapté aux préférences des utilisateurs.

Dans ce projet, nous formulons plusieurs hypothèses que nous allons chercher à valider à travers cette étude :

* **Hypothèse 1 :** Les utilisateurs ayant des notations similaires pour les mêmes films partagent des goûts proches.
* **Hypothèse 2 :** Les genres sont des indicateurs fiables pour déterminer la similarité entre films.
* **Hypothèse 3 :** Les tendances globales des notations peuvent être utilisées pour guider les recommandations, notamment pour les nouveaux utilisateurs.

Notre analyse se déroulera en plusieurs étapes clés, conçues pour nous aider à tester ces hypothèses et à mieux comprendre nos données :

1. **Analyse des Films**
2. **Analyse des Notes**
3. **Analyse des Genres**
4. **Analyse des Utilisateurs**

Chaque étape nous permettra de mieux comprendre nos données, de tester nos hypothèses et de mettre en lumière les informations pertinentes pour la construction de notre système de recommandation.

1. **Analyse des Films**

Nous allons lister les points à faire ressortir et les graphiques d’illustration :

* + Nombre total de films uniques
  + Répartition des films par année de sortie
  + Top 10 films les plus notés
  + Calcul des films les mieux notés avec plus de 50 notes
  + Calcul du nombre de genres par film
  + Top 10 films avec le plus de genres
  + Distribution des films en fonction du nombre de genres auxquels ils appartiennent (mono-genre vs multi-genres).

Graphiques produits :

* Répartition des films par année de sortie (graphique à barres)
* Top 10 films les plus notés (graphique à barres horizontales)
* Top 10 des films les mieux notés (> 50 notes) (graphique à barres horizontales)
* Top 10 des films avec le plus de genres (graphique à barres horizontales)
* Distribution des films mono-genre vs multi-genres (diagramme circulaire)

Le code : **PS C:\projet\_work\Script> python analyse\_films.py**

import pandas as pd

import matplotlib.pyplot as plt

**# Chargement des données**

data = pd.read\_csv(r'C:\projet\_work\Data-source\merged\_final\_data.csv')

**# Conversion de la colonne date au format datetime**

data['release\_date'] = pd.to\_datetime(data['release\_date'])

**# ----------------------------- Analyse Générale -----------------------------**

**# Nombre total de films uniques**

total\_films = data['movie\_id'].nunique()

print(f"Nombre total de films : {total\_films}")

**# Répartition des films par année de sortie**

films\_par\_annee = data[['movie\_id', 'release\_date']].drop\_duplicates().groupby(data['release\_date'].dt.year).count()['movie\_id']

print("\nRépartition des films par année (10 premières lignes):")

print(films\_par\_annee.head(10))

**# Graphique: Répartition des films par année**

plt.figure(figsize=(10,5))

films\_par\_annee.plot(kind='bar', color='skyblue')

plt.title('Nombre de films par année de sortie')

plt.xlabel('Année de sortie')

plt.ylabel('Nombre de films')

plt.tight\_layout()

plt.show()

**# ----------------------------- Films populaires -----------------------------**

**# Top 10 films les plus notés**

top\_films\_notes = data['movie\_title'].value\_counts().head(10)

print("\nTop 10 des films les plus notés:")

print(top\_films\_notes)

**# Graphique: Top 10 films les plus notés**

plt.figure(figsize=(10,5))

top\_films\_notes.plot(kind='barh', color='salmon')

plt.xlabel('Nombre de notes')

plt.ylabel('Titre du film')

plt.title('Top 10 des films les plus notés')

plt.gca().invert\_yaxis()

plt.tight\_layout()

plt.show()

**# ----------------------------- Films les mieux notés -----------------------------**

**# Calcul des films les mieux notés avec plus de 50 notes**

ratings\_stats = data.groupby('movie\_title')['rating'].agg(['mean', 'count'])

films\_mieux\_notes = ratings\_stats[ratings\_stats['count'] > 50].sort\_values('mean', ascending=False).head(10)

print("\nTop 10 des films les mieux notés (ayant plus de 50 notes):")

print(films\_mieux\_notes)

**# Graphique: Top 10 des films les mieux notés**

plt.figure(figsize=(10,5))

films\_mieux\_notes['mean'].plot(kind='barh', color='limegreen')

plt.xlabel('Note moyenne')

plt.ylabel('Titre du film')

plt.title('Top 10 des films les mieux notés (> 50 notes)')

plt.gca().invert\_yaxis()

plt.xlim(3, 5)

plt.tight\_layout()

plt.show()

**# ----------------------------- Films multi-genres -----------------------------**

genres = ['Action','Adventure','Animation',"Children's",'Comedy','Crime',

'Documentary','Drama','Fantasy','Film-Noir','Horror','Musical',

'Mystery','Romance','Sci-Fi','Thriller','War','Western']

**# Calcul du nombre de genres par film**

films\_genres = data[['movie\_id', 'movie\_title'] + genres].drop\_duplicates()

films\_genres['nombre\_genres'] = films\_genres[genres].sum(axis=1)

**# Top 10 films avec le plus de genres**

films\_multi\_genres = films\_genres.sort\_values('nombre\_genres', ascending=False).head(10)

print("\nTop 10 films multi-genres:")

print(films\_multi\_genres[['movie\_title', 'nombre\_genres']])

**# Graphique: Films multi-genres**

plt.figure(figsize=(10,5))

films\_multi\_genres.set\_index('movie\_title')['nombre\_genres'].plot(kind='barh', color='orchid')

plt.xlabel('Nombre de genres')

plt.ylabel('Titre du film')

plt.title('Top 10 des films avec le plus de genres')

plt.gca().invert\_yaxis()

plt.tight\_layout()

plt.show()

**# ----------------------------- Distribution mono-genre vs multi-genres -----------------------------**

films\_genres\_distribution = films\_genres['nombre\_genres'].apply(lambda x: 'Mono-genre' if x==1 else 'Multi-genres').value\_counts()

print("\nDistribution films mono-genre vs multi-genres :")

print(films\_genres\_distribution)

**# Graphique: Mono-genre vs multi-genres**

plt.figure(figsize=(7,5))

films\_genres\_distribution.plot(kind='pie', autopct='%1.1f%%', colors=['gold','deepskyblue'], explode=[0, 0.05])

plt.title('Distribution des films mono-genre vs multi-genres')

plt.ylabel('')

plt.tight\_layout()

plt.show()

**Exécution du code :**

**PS C:\projet\_work\Script> python analyse\_films.py**

Nombre total de films : 1460

Répartition des films par année (10 premières lignes):

release\_date

1922 1

1930 1

1931 1

1932 1

1933 1

1934 4

1935 3

1936 2

1937 3

1938 3

Name: movie\_id, dtype: int64

Top 10 des films les plus notés:

movie\_title

Star Wars (1977) 573

Contact (1997) 500

Fargo (1996) 499

Return of the Jedi (1983) 497

Liar Liar (1997) 476

English Patient, The (1996) 474

Scream (1996) 468

Toy Story (1995) 443

Air Force One (1997) 423

Independence Day (ID4) (1996) 421

Name: count, dtype: int64

Top 10 des films les mieux notés (ayant plus de 50 notes):

mean count

movie\_title

Wrong Trousers, The (1993) 4.500000 114

Close Shave, A (1995) 4.490909 110

Schindler's List (1993) 4.472222 288

Casablanca (1942) 4.461864 236

Shawshank Redemption, The (1994) 4.446886 273

Wallace & Gromit: The Best of Aardman Animation... 4.439394 66

Usual Suspects, The (1995) 4.400778 257

Third Man, The (1949) 4.397059 68

Rear Window (1954) 4.390000 200

12 Angry Men (1957) 4.355932 118

Top 10 films multi-genres:

movie\_title nombre\_genres

2991 Transformers: The Movie, The (1986) 6

2281 Kid in King Arthur's Court, A (1995) 6

181 Empire Strikes Back, The (1980) 6

49 Return of the Jedi (1983) 5

1539 Diva (1981) 5

468 Star Wars (1977) 5

492 Army of Darkness (1993) 5

1979 Heavy Metal (1981) 5

11930 Pagemaster, The (1994) 5

411 From Dusk Till Dawn (1996) 5

Distribution films mono-genre vs multi-genres :

nombre\_genres

Multi-genres 781

Mono-genre 679

Name: count, dtype: int64

PS C:\projet\_work\Script> python analyse\_notes.py

PS C:\projet\_work\Script> python analyse\_notes.py

Nombre total de notes dans la base : 94276

Statistiques descriptives des notes :

count 94276.000000

mean 3.556547

std 1.112641

min 1.000000

25% 3.000000

50% 4.000000

75% 4.000000

max 5.000000

Name: rating, dtype: float64

Nombre moyen de notes par film : 64.57

Note moyenne des films par année (10 premières lignes):

release\_year

1922 3.580000

1930 3.875000

1931 4.075000

1932 3.833333

1933 4.034483

1934 4.033557

1935 3.931034

1936 3.727273

1937 3.739130

1938 3.894180

Name: rating, dtype: float64

Note moyenne des films selon ancienneté :

age\_group

Ancien (<1980) 3.923824

Récent (>=1980) 3.478773

Name: rating, dtype: float64

**Analyse brève des résultats :**

Le dataset comporte un total de **1460 films** uniques après nettoyage.

**Répartition des films par année :**

* Les années **1922 à 1938** présentent un nombre très faible de films (entre 1 et 4 par année), ce qui indique que la majorité des films du dataset sont probablement plus récents.

**Top 10 des films les plus notés :**

* **Star Wars (1977)** arrive clairement en tête avec **573 notes**, suivi par **Contact (1997)** et **Fargo (1996)**.
* Les films populaires proviennent principalement des années 1990 et incluent souvent des grands succès commerciaux (blockbusters), ce qui est logique vu leur popularité et leur large audience.

**Films les mieux notés (ayant plus de 50 notes) :**

* **« The Wrong Trousers (1993) »** (4.50) et **« A Close Shave (1995) »** (4.49) occupent les premières positions. Ce sont des films d'animation très appréciés.
* Les grands classiques comme **« Schindler's List (1993) »**, **« Casablanca (1942) »**, et **« Shawshank Redemption (1994) »** apparaissent également, confirmant leur statut de films très appréciés par les utilisateurs.
* Cette liste montre que les films anciens (classiques) sont souvent très bien notés.

**Films multi-genres :**

* Certains films sont classés dans jusqu’à **6 genres différents** (par ex. **Transformers (1986)**, **The Empire Strikes Back (1980)**), indiquant une complexité ou diversité dans leur classification.

**Distribution Mono-genre vs Multi-genres :**

* **781 films (53%)** sont multi-genres contre **679 (47%) mono-genre**.
* Cette distribution équilibrée montre que la plupart des films combinent plusieurs genres, ce qui pourrait être pertinent pour nos futures analyses de recommandations.

Une image contenant texte, capture d’écran, affichage, Rectangle

Le contenu généré par l’IA peut être incorrect.

On remarque clairement une augmentation significative du nombre de films à partir des années 1980, avec un pic majeur dans les années 1990 (en particulier autour de 1995-1997). Cela confirme que le jeu de données est principalement constitué de films récents, ce qui est cohérent avec le classement des films les plus notés (majoritairement des années 1990).

**Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.**

On retrouve essentiellement des blockbusters (film a très grand succès commercial et populaire) des années 1990 tels que **« Star Wars (1977) »**, **« Contact (1997) »**, et **« Fargo (1996) »**. Ce graphique est cohérent avec le graphique par année de sortie, qui montre que les films récents attirent le plus d’attention et de notations de la part des utilisateurs.

**Une image contenant texte, capture d’écran, Police, ligne

Le contenu généré par l’IA peut être incorrect.**

Les films les mieux notés sont des grands classiques ou films unanimement appréciés, tels que **« Wrong Trousers (1993) »**, **« Schindler's List (1993) »** et **« Casablanca (1942) »**. Ce graphique révèle clairement que les utilisateurs notent généralement très hautement les classiques reconnus, indépendamment de leur ancienneté.

**Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.**

Des films tels que **« Transformers (1986) »** et **« Empire Strikes Back (1980) »** apparaissent en tête avec jusqu'à **6 genres différents**. Cela montre une difficulté à classifier précisément ces films dans une seule catégorie et explique aussi leur large audience et leur popularité.

**Une image contenant texte, capture d’écran, diagramme, Police

Le contenu généré par l’IA peut être incorrect.**

Le graphique révèle une répartition relativement équilibrée : **53,5% multi-genres** et **46,5% mono-genre**. Cette répartition indique une grande diversité dans la catégorisation des films, qui se reflète dans la popularité variée des films et explique pourquoi certains films apparaissent dans les tops de plusieurs genres.

1. **Analyse des Notes**

Les points à faire ressortir et les graphiques d’illustration :

* + **Le nombre total de notes dans le jeu de données.**
  + **Les statistiques descriptives de la colonne 'rating' (moyenne, médiane, écart-type, etc.).**
  + **Le nombre moyen de notes par film.**
  + **La note moyenne des films pour chaque année.**
  + **Comparaison des notes moyennes des films anciens et récents.**

**Graphiques produits :**

* Histogramme de la distribution des notes
* Distribution du nombre de notes par film
* Évolution des notes moyennes par année
* Moyenne des notes films anciens vs récents

Le code : **analyse\_notes.py**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

**# Chargement des données**

data = pd.read\_csv(r'C:\projet\_work\Data-source\merged\_final\_data.csv')

# ----------------------- Analyse Générale des Notes ------------------------

**# Nombre total de notes**

total\_notes = data.shape[0]

print(f"Nombre total de notes dans la base : {total\_notes}")

**# Statistiques descriptives des notes**

stats\_notes = data['rating'].describe()

print("\nStatistiques descriptives des notes :")

print(stats\_notes)

**# Histogramme de la distribution des notes**

plt.figure(figsize=(8,5))

sns.histplot(data['rating'], bins=5, kde=True)

plt.title('Distribution des notes utilisateurs')

plt.xlabel('Notes')

plt.ylabel('Fréquence')

plt.tight\_layout()

plt.show()

**# ----------------------- Nombre moyen de notes par film ------------------------**

notes\_par\_film = data.groupby('movie\_id').size()

moyenne\_notes\_film = notes\_par\_film.mean().round(2)

print(f"\nNombre moyen de notes par film : {moyenne\_notes\_film}")

**# Graphique: Distribution du nombre de notes par film**

plt.figure(figsize=(8,5))

sns.histplot(notes\_par\_film, bins=30, kde=True, color='lightcoral')

plt.title('Distribution du nombre de notes par film')

plt.xlabel('Nombre de notes')

plt.ylabel('Nombre de films')

plt.tight\_layout()

plt.show()

**# ----------------------- Corrélation entre Année et Note ------------------------**

# Convertir release\_date en datetime

data['release\_date'] = pd.to\_datetime(data['release\_date'])

data['release\_year'] = data['release\_date'].dt.year

**# Note moyenne par année**

note\_par\_annee = data.groupby('release\_year')['rating'].mean()

print("\nNote moyenne des films par année (10 premières lignes):")

print(note\_par\_annee.head(10))

**# Graphique: Évolution des notes moyennes par année**

plt.figure(figsize=(12,5))

note\_par\_annee.plot(marker='o', linestyle='-', color='teal')

plt.title('Évolution de la note moyenne des films par année')

plt.xlabel('Année de sortie')

plt.ylabel('Note moyenne')

plt.grid(True)

plt.tight\_layout()

plt.show()

**# ----------------------- Films anciens vs récents ------------------------**

# Créer deux catégories de films : anciens (avant 1980), récents (1980 et après)

data['age\_group'] = data['release\_year'].apply(lambda x: 'Ancien (<1980)' if x < 1980 else 'Récent (>=1980)')

notes\_age\_group = data.groupby('age\_group')['rating'].mean()

print("\nNote moyenne des films selon ancienneté :")

print(notes\_age\_group)

**# Graphique: Moyenne des notes films anciens vs récents**

plt.figure(figsize=(7,5))

notes\_age\_group.plot(kind='bar', color=['orange','dodgerblue'])

plt.title('Note moyenne : films anciens vs récents')

plt.ylabel('Note moyenne')

plt.xlabel('Catégorie de film')

plt.xticks(rotation=0)

plt.tight\_layout()

plt.show()

**Exécution de codes :**

**PS C:\projet\_work\Script> python analyse\_notes.py**

Nombre total de notes dans la base : 94276

Statistiques descriptives des notes :

count 94276.000000

mean 3.556547

std 1.112641

min 1.000000

25% 3.000000

50% 4.000000

75% 4.000000

max 5.000000

Name: rating, dtype: float64

Nombre moyen de notes par film : 64.57

Note moyenne des films par année (10 premières lignes):

release\_year

1922 3.580000

1930 3.875000

1931 4.075000

1932 3.833333

1933 4.034483

1934 4.033557

1935 3.931034

1936 3.727273

1937 3.739130

1938 3.894180

Name: rating, dtype: float64

Note moyenne des films selon ancienneté :

age\_group

Ancien (<1980) 3.923824

Récent (>=1980) 3.478773

Name: rating, dtype: float64

Une image contenant diagramme, Tracé, ligne, conception

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, diagramme, capture d’écran, ligne

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, ligne, Tracé, Police

Le contenu généré par l’IA peut être incorrect.

**Une image contenant texte, capture d’écran, Rectangle, diagramme

Le contenu généré par l’IA peut être incorrect.**

**Synthèse des résultats et graphiques (analyse\_notes.py)**

**Nombre total et distribution des notes :**

* Le dataset contient 94 276 notations, avec une moyenne générale de 3,56/5.
* La majorité des notes attribuées par les utilisateurs se situe entre 3 et 4, indiquant une tendance générale plutôt positive dans les évaluations.

**Distribution du nombre de notes par film :**

* La majorité des films ont très peu de notes (moins de 50 notes), ce qui est typique d'un tel jeu de données où seuls quelques films très populaires concentrent une grande quantité de notations.
* Quelques rares films atteignent un nombre très élevé de notes (jusqu'à environ 600 notes, comme vu précédemment avec « Star Wars (1977) »).

**Évolution des notes moyennes par année :**

* Les films plus anciens (entre 1930-1960) obtiennent généralement des notes moyennes plus élevées.
* À partir des années 1980-1990, on remarque une nette baisse progressive des notes moyennes attribuées aux films, potentiellement due à une diversité croissante de genres et à une audience plus large et critique.

**Note moyenne : films anciens vs récents :**

* Les films anciens (avant 1980) obtiennent une note moyenne plus élevée (3,92) comparativement aux films récents (après 1980), notés en moyenne à 3,48.

1. **Analyse des Genres**

Les points à faire ressortir et les graphiques d’illustration :

* + Le nombre de films pour chaque genre.
  + La note moyenne des films pour chaque genre.
  + L'évolution de genres spécifiques (par exemple, "Sci-Fi" et "Drama") au fil des décennies.
  + La corrélation entre les différents genres.

**Graphiques produits :**

* + Un graphique à barres pour visualiser le nombre de films dans chaque genre.
  + Un graphique à barres pour visualiser la note moyenne des films pour chaque genre.
  + Un graphique linéaire pour visualiser l'évolution de genres spécifiques au fil du temps (par décennie)
  + Une heatmap pour visualiser la matrice de corrélation entre les différents genres.

**Le code : analyse\_genres.py**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

**# Chargement des données**

data = pd.read\_csv(r'C:\projet\_work\Data-source\merged\_final\_data.csv')

**# Conversion date de sortie en datetime**

data['release\_date'] = pd.to\_datetime(data['release\_date'])

data['release\_year'] = data['release\_date'].dt.year

genres = ['Action','Adventure','Animation',"Children's",'Comedy','Crime',

'Documentary','Drama','Fantasy','Film-Noir','Horror','Musical',

'Mystery','Romance','Sci-Fi','Thriller','War','Western']

# **----------------------- Nombre de films par genre ------------------------**

nb\_films\_genre = {genre: data[data[genre]==1]['movie\_id'].nunique() for genre in genres}

nb\_films\_genre\_series = pd.Series(nb\_films\_genre).sort\_values(ascending=False)

print("Nombre de films par genre :")

print(nb\_films\_genre\_series)

**# Graphique: nombre de films par genre**

plt.figure(figsize=(12,6))

sns.barplot(y=nb\_films\_genre\_series.index, x=nb\_films\_genre\_series.values, palette='mako')

plt.xlabel('Nombre de films')

plt.ylabel('Genre')

plt.title('Nombre de films par genre')

plt.tight\_layout()

plt.show()

**# ----------------------- Notes moyennes par genre** ------------------------

note\_genres = {genre: data[data[genre]==1]['rating'].mean() for genre in genres}

note\_genres\_series = pd.Series(note\_genres).sort\_values(ascending=False)

print("\nNote moyenne par genre :")

print(note\_genres\_series)

**# Graphique: note moyenne par genre**

plt.figure(figsize=(12,6))

sns.barplot(y=note\_genres\_series.index, x=note\_genres\_series.values, palette='viridis')

plt.xlabel('Note moyenne')

plt.ylabel('Genre')

plt.title('Note moyenne par genre')

plt.xlim(3,5)

plt.tight\_layout()

plt.show()

**# ----------------------- Évolution des genres dans le temps ------------------------**

**# Exemple : évolution du genre "Sci-Fi" et "Drama" par décennie**

data['decade'] = (data['release\_year']//10)\*10

genres\_temps = data.groupby('decade')[['Sci-Fi','Drama']].sum()

print("\nÉvolution des genres 'Sci-Fi' et 'Drama' par décennie :")

print(genres\_temps)

**# Graphique : Évolution temporelle**

genres\_temps.plot(kind='line', marker='o', figsize=(10,5))

plt.title('Évolution des genres Sci-Fi et Drama par décennie')

plt.xlabel('Décennie')

plt.ylabel('Nombre de films')

plt.grid(True)

plt.tight\_layout()

plt.show()

**# ----------------------- Corrélation entre genres ------------------------**

correlation\_genres = data[genres].corr()

print("\nCorrélation entre genres (extrait) :")

print(correlation\_genres.head())

**# Graphique: heatmap des corrélations**

plt.figure(figsize=(14,10))

sns.heatmap(correlation\_genres, annot=True, cmap='coolwarm', linewidths=0.5)

plt.title('Heatmap des corrélations entre genres')

plt.tight\_layout()

plt.show()

**PS C:\projet\_work\Script> python analyse\_genres.py**

Nombre de films par genre :

Drama 608

Comedy 456

Thriller 230

Romance 228

Action 227

Adventure 130

Children's 119

Sci-Fi 97

Crime 93

Horror 83

War 68

Mystery 57

Musical 55

Animation 42

Documentary 30

Western 25

Film-Noir 21

Fantasy 20

dtype: int64

C:\projet\_work\Script\analyse\_genres.py:26: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

sns.barplot(y=nb\_films\_genre\_series.index, x=nb\_films\_genre\_series.values, palette='mako')

Note moyenne par genre :

Film-Noir 3.946111

War 3.829705

Documentary 3.767267

Drama 3.717208

Mystery 3.655869

Crime 3.655019

Romance 3.642316

Western 3.641748

Animation 3.595398

Sci-Fi 3.576115

Thriller 3.529304

Musical 3.529098

Adventure 3.521315

Action 3.499489

Comedy 3.415981

Children's 3.378061

Horror 3.337219

Fantasy 3.230648

dtype: float64

C:\projet\_work\Script\analyse\_genres.py:43: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

sns.barplot(y=note\_genres\_series.index, x=note\_genres\_series.values, palette='viridis')

Évolution des genres 'Sci-Fi' et 'Drama' par décennie :

Sci-Fi Drama

decade

1920 0 0

1930 0 637

1940 0 740

1950 263 1217

1960 536 1278

1970 1265 1517

1980 3044 4682

1990 7117 27423

Corrélation entre genres (extrait) :

Action Adventure Animation Children's Comedy Crime ... Mystery Romance Sci-Fi Thriller War Western

Action 1.000000 0.453562 -0.101185 -0.147025 -0.225300 0.005314 ... -0.033760 -0.018278 0.324754 0.249892 0.167365 0.063701

Adventure 0.453562 1.000000 -0.026709 0.096590 -0.112925 -0.031725 ... -0.044118 -0.017517 0.295197 -0.051587 0.087986 0.011085

Animation -0.101185 -0.026709 1.000000 0.558514 0.032727 -0.057732 ... -0.046010 -0.086526 -0.047056 -0.078560 -0.057961 -0.026649

Children's -0.147025 0.096590 0.558514 1.000000 0.086271 -0.082387 ... -0.055738 -0.119693 -0.044201 -0.144973 -0.086481 -0.031909

Comedy -0.225300 -0.112925 0.032727 0.086271 1.000000 -0.091057 ... -0.112975 0.094760 -0.146919 -0.292629 -0.121855 0.002660

Une image contenant texte, ligne, Tracé, diagramme

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, nombre, ligne

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, ligne, Tracé

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, Tracé, diagramme

Le contenu généré par l’IA peut être incorrect.

**Synthèse des résultats (analyse\_genres.py)**

**Nombre de films par genre :**

Le genre **« Drama »** domine largement avec le plus grand nombre de films, suivi par **Comedy**, **Action** et **Thriller**. À l’inverse, les genres comme **Film-Noir** ou **Fantasy** sont nettement sous-représentés. Ceci montre que le jeu de données privilégie les films à large audience et grand public.

**Note moyenne par genre :**

Les genres comme **Film-Noir, War et Documentary** obtiennent les notes moyennes les plus élevées (>3.7). À l’opposé, les genres **Horror**, **Children’s** et **Comedy** obtiennent des notes moyennes plus basses. On en conclut que les utilisateurs valorisent davantage les genres spécialisés, sérieux ou classiques (comme Film-Noir, Documentary) par rapport aux genres très populaires.

**Évolution temporelle des genres (Sci-Fi et Drama) :**

Le genre **« Drama »** connaît une forte augmentation des années **1980-1990**, avec une croissance exponentielle en popularité. **Sci-Fi** augmente également fortement à partir des années 1980, mais reste toujours inférieur en nombre à Drama. Ce résultat reflète clairement l'intérêt croissant pour ces deux genres, surtout le « Drama » dans les décennies récentes.

**Corrélation entre les genres :**

La heatmap montre clairement que certains genres sont fréquemment associés. Par exemple :

**Action** & **Adventure** (forte corrélation : 0.45)

**Animation** & **Children's** (forte corrélation, logique puisque ces deux genres sont souvent destinés au même public). Inversement, des genres comme **Film-Noir** et **Fantasy** sont peu associés à d’autres, confirmant leur caractère spécifique.

1. **Analyse des Utilisateurs**

Les points à faire ressortir et les graphiques d’illustration :

* + Le nombre total d'utilisateurs uniques dans le jeu de données.
  + Le nombre moyen de notes attribuées par chaque utilisateur.
  + Les utilisateurs qui ont attribué le plus grand nombre de notes.
  + La distribution du nombre de notes attribuées par chaque utilisateur.
* **Variance et Moyenne des notes par utilisateur :**
  + La variance et la moyenne des notes attribuées par chaque utilisateur.

**Graphiques produits :**

* + Un graphique pour visualiser les 10 utilisateurs qui ont attribué le plus de notes.
  + Un histogramme pour visualiser la distribution des notes moyennes attribuées par les utilisateurs.
  + Un histogramme pour visualiser la distribution des variances des notes attribuées par les utilisateurs.

Le code : **analyse\_utilisateurs.py**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

**# Chargement des données**

data = pd.read\_csv(r'C:\projet\_work\Data-source\merged\_final\_data.csv')

**# ----------------------- Analyse Générale des Utilisateurs ------------------------**

**# Nombre total d’utilisateurs**

total\_users = data['user\_id'].nunique()

print(f"Nombre total d'utilisateurs : {total\_users}")

**# Nombre moyen de notes par utilisateur**

notes\_par\_utilisateur = data.groupby('user\_id').size()

moyenne\_notes\_user = notes\_par\_utilisateur.mean().round(2)

print(f"\nNombre moyen de notes par utilisateur : {moyenne\_notes\_user}")

**# ----------------------- Utilisateurs les plus actifs ------------------------**

top\_utilisateurs\_actifs = notes\_par\_utilisateur.sort\_values(ascending=False).head(10)

print("\nTop 10 utilisateurs les plus actifs :")

print(top\_utilisateurs\_actifs)

**# Graphique: Top 10 utilisateurs les plus actifs**

plt.figure(figsize=(10,5))

top\_utilisateurs\_actifs.plot(kind='bar', color='skyblue')

plt.title('Top 10 utilisateurs les plus actifs')

plt.xlabel('User ID')

plt.ylabel('Nombre de notes')

plt.tight\_layout()

plt.show()

**# ----------------------- Distribution du nombre de notes par utilisateur ------------------------**

plt.figure(figsize=(10,5))

sns.histplot(notes\_par\_utilisateur, bins=50, kde=True, color='green')

plt.title('Distribution du nombre de notes par utilisateur')

plt.xlabel('Nombre de notes données')

plt.ylabel('Nombre d\'utilisateurs')

plt.tight\_layout()

plt.show()

**# ----------------------- Variance et Moyenne des notes par utilisateur ------------------------**

stats\_users = data.groupby('user\_id')['rating'].agg(['mean','var'])

print("\nStatistiques (moyenne et variance) des notes par utilisateur (10 premiers) :")

print(stats\_users.head(10))

**# Graphique: Distribution des moyennes des notes par utilisateur**

plt.figure(figsize=(10,5))

sns.histplot(stats\_users['mean'], bins=30, kde=True, color='orange')

plt.title('Distribution des moyennes de notes par utilisateur')

plt.xlabel('Moyenne des notes')

plt.ylabel('Nombre d\'utilisateurs')

plt.tight\_layout()

plt.show()

**# Graphique: Distribution des variances des notes par utilisateur**

plt.figure(figsize=(10,5))

sns.histplot(stats\_users['var'].dropna(), bins=30, kde=True, color='red')

plt.title('Distribution des variances des notes par utilisateur')

plt.xlabel('Variance des notes')

plt.ylabel('Nombre d\'utilisateurs')

plt.tight\_layout()

plt.show()

**Exécution du code :**

**PS C:\projet\_work\Script> python analyse\_utilisateurs.py**

Nombre total d'utilisateurs : 933

Nombre moyen de notes par utilisateur : 101.05

Top 10 utilisateurs les plus actifs :

user\_id

279 424

429 414

181 406

846 405

94 400

7 400

682 398

308 396

92 388

293 388

dtype: int64

Statistiques (moyenne et variance) des notes par utilisateur (10 premiers) :

mean var

user\_id

1 3.603704 1.600730

2 3.754098 0.955191

3 2.796296 1.486024

4 4.333333 0.840580

5 2.889535 1.841527

6 3.635071 1.080478

7 3.975000 1.117168

8 3.796610 1.544126

9 4.272727 0.874459

10 4.207650 0.341260

**Les graphiques**

Une image contenant diagramme, Tracé, ligne

Le contenu généré par l’IA peut être incorrect.Une image contenant diagramme, texte, Tracé, ligne

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, diagramme, ligne

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, Police, ligne

Le contenu généré par l’IA peut être incorrect.

**Synthèse des résultats (analyse\_utilisateurs.py)**

**Activité et nombre de notes par utilisateur :**

* Le dataset comporte **933 utilisateurs**, qui attribuent en moyenne **101 notes chacun**, ce qui indique une forte activité moyenne par utilisateur.
* Cependant, la distribution montre que beaucoup d’utilisateurs notent relativement peu de films, tandis que seulement quelques utilisateurs très actifs notent énormément de films (jusqu’à plus de **400 films** notés pour les utilisateurs les plus actifs).

**Utilisateurs les plus actifs :**

* Les utilisateurs les plus actifs notent entre **388 et 424 films**, révélant leur engagement très élevé avec la plateforme.

**Distribution des moyennes de notes par utilisateur :**

* La majorité des utilisateurs ont une moyenne de notes située entre **3 et 4**, ce qui suggère un comportement généralement positif et peu critique des utilisateurs envers les films. Cette observation rejoint directement la distribution générale des notes vue dans les analyses précédentes.

**Distribution des variances des notes par utilisateur :**

* La variance des notes attribuées par utilisateur est généralement comprise entre **0,5 et 1,5**, ce qui signifie que la plupart des utilisateurs donnent des notes variées mais modérées. Peu d’utilisateurs sont extrêmement constants (faible variance) ou extrêmement variés dans leurs notes (haute variance).

**Conclusion**

Les résultats obtenus à travers l’analyse exploratoire confirment clairement la pertinence des hypothèses formulées :

**Hypothèse 1 (Similarité utilisateurs)** :  
Les distributions des moyennes et variances des notes indiquent que certains utilisateurs partagent effectivement des goûts similaires.

**Hypothèse 2 (Genres indicateurs fiables)** :  
Les corrélations fortes entre certains genres confirment que les genres constituent des critères pertinents pour mesurer la similarité entre films.

**Hypothèse 3 (Tendances globales)** :  
Les tendances temporelles et globales des notes montrent clairement qu'il existe des préférences dominantes exploitables pour guider les recommandations.

Toutefois, pour aller plus loin et valider rigoureusement ces hypothèses, il serait particulièrement pertinent d’utiliser des approches basées sur :

* **Le Filtrage Collaboratif** :  
  Pour confirmer plus précisément la similarité entre utilisateurs selon leurs notations, et fournir des recommandations basées sur les goûts proches.
* **Les modèles basés sur le contenu** :  
  Pour valider efficacement l'utilité des genres comme critères pour déterminer la similarité entre films.
* **Le modele Hybride**

Ces approches vont nous permettre d’éclaircir et de valider concrètement ces hypothèses, tout en précisant mieux les recommandations à proposer aux utilisateurs.

Ainsi, l’étape suivante consistera à implémenter clairement ces modèles de recommandation .

**IV et V Phase de Modélisation et d’évaluation**

Pour cette phase de modélisation de projet, j'ai opté pour l'algorithme KNN en raison de sa simplicité et de sa robustesse pour mesurer directement la similarité entre utilisateurs ou films. Selon le modèle, le choix de la mesure de similarité a été adapté : j'ai utilisé la distance cosinus pour le modèle Content-Based afin de comparer efficacement les vecteurs de caractéristiques (comme les genres), tandis que pour les modèles collaboratifs (User-User et Item-Item), j'ai testé les distances euclidienne et Manhattan, ces dernières s'étant révélées particulièrement performantes pour capturer les variations des notations.

Pour chaque approche, j'ai suivi un processus structuré comprenant la préparation des données (création de matrices utilisateur-film ou film-utilisateur, et extraction des caractéristiques de contenu), l'entraînement du modèle KNN avec optimisation des hyperparamètres et l'évaluation rigoureuse via des métriques telles que le RMSE. Ce processus m'a permis de sélectionner le modèle le plus performant, qui s'est avéré être le modèle Item-Item KNN, et de le déployer dans une application interactive et une API, garantissant ainsi des recommandations pertinentes et fiables.

* 1. **Préparation des données pour la matrice utilisateur-item.**

Cette matrice représente les interactions utilisateur-film, où chaque cellule indique la note donnée par un utilisateur à un film. Elle nous permet de transformer les données brutes (utilisateurs, films, notes) en une structure matricielle qui facilite l'analyse et l'application d'algorithmes de recommandation.

Code : **preparation\_matrice.py**

import pandas as pd

**# Chargement des données**

data = pd.read\_csv(r'C:\projet\_work\Data-source\merged\_final\_data.csv')

**# Création de la matrice User-Item**

user\_item\_matrix = data.pivot\_table(index='user\_id', columns='movie\_title', values='rating').fillna(0)

**# Sauvegarde de la matrice**

user\_item\_matrix.to\_csv(r'C:\projet\_work\Data-source\user\_item\_matrix.csv')

print("Matrice User-Item créée avec succès.")

**Exécution du code**

**PS C:\projet\_work\Script> python preparation\_matrice.py**

Matrice User-Item créée avec succès.

* 1. **Implémentation de KNN pour le filtrage collaboratif.**

Le filtrage collaboratif repose sur le principe que des utilisateurs ayant exprimé des préférences similaires dans le passé continueront de partager des goûts comparables à l'avenir. Pour mettre en œuvre ce concept, je vais créer deux modèles distincts :

* Modèle User-User : qui identifie les utilisateurs aux comportements similaires et recommande des films qu'ils ont appréciés.
* Modèle Item-Item : qui se base sur la similarité entre les films, en recommandant des titres proches de ceux déjà appréciés par l'utilisateur.

**Modèle User-User : Creation**

**Le code :** **modele\_user\_user.py**

import pandas as pd

import numpy as np

from sklearn.neighbors import NearestNeighbors

import joblib

**# Chargement et préparation des données**

data = pd.read\_csv(r'C:\projet\_work\Data-source\merged\_final\_data.csv')

user\_item\_matrix = data.pivot\_table(index='user\_id', columns='movie\_title', values='rating', fill\_value=0)

**# Entraînement du modèle KNN User-User**

model\_knn = NearestNeighbors(metric='cosine', algorithm='brute', n\_neighbors=10)

model\_knn.fit(user\_item\_matrix.values)

**# Sauvegarde du modèle entraîné**

import joblib

import os

model\_path = r'C:\projet\_work\Data-source\user\_user\_knn\_model.pkl'

pd.to\_pickle(model\_knn, model\_path)

**# Fonction de recommandation**

def recommander\_films(user\_id, matrice, model\_knn, n\_reco=5):

user\_index = matrice.index.get\_loc(user\_id)

distances, indices = model\_knn.kneighbors([matrice.iloc[user\_index].values], n\_neighbors=10)

voisins = matrice.iloc[indices[0]]

moyenne\_voisins = voisins.mean(axis=0)

deja\_notes = matrice.iloc[user\_index] > 0

recommandations = moyenne\_voisins[~deja\_notes].sort\_values(ascending=False).head(n\_reco)

return recommandations

**# Exemple pour utilisateur ID=1**

user\_id = 1

recommandations = recommander\_films(user\_id, user\_item\_matrix, model\_knn, n\_reco=5)

print("\nModèle User-User KNN créé avec succès")

print(f"Recommandations finales pour l'utilisateur {user\_id} :")

print(recommandations)

**Exécution du code**

**PS C:\projet\_work\Script> python modele\_user\_user.py**

Modèle User-User KNN créé avec succès

Recommandations finales pour l'utilisateur 1 :

movie\_title

Schindler's List (1993) 3.6

Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1963) 3.4

E.T. the Extra-Terrestrial (1982) 3.3

Dave (1993) 3.3

Stand by Me (1986) 3.3

dtype: float64

Bref commentaire : Le modèle User-User KNN a été créé avec succès et a fourni un exemple concret de recommandations pour l'utilisateur 1, illustrant son fonctionnement.

**Modèle User-User : Evaluation**

**Le code : evaluation\_modele\_user\_user.py**

import pandas as pd

import numpy as np

from sklearn.neighbors import NearestNeighbors

from sklearn.metrics import mean\_squared\_error

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

**# Chargement des données**

data = pd.read\_csv(r'C:\projet\_work\Data-source\merged\_final\_data.csv')

**# Création matrice user-item**

user\_item\_matrix = data.pivot\_table(index='user\_id', columns='movie\_title', values='rating', fill\_value=0)

**# Split en train/test**

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

**# Matrices train/test**

train\_matrix = train\_data.pivot\_table(index='user\_id', columns='movie\_title', values='rating', fill\_value=0)

test\_matrix = test\_data.pivot\_table(index='user\_id', columns='movie\_title', values='rating', fill\_value=0)

**# Modèle KNN entraîné**

model\_knn = NearestNeighbors(metric='cosine', algorithm='brute', n\_neighbors=10)

model\_knn.fit(train\_matrix.values)

**# Fonction prédiction**

def predire\_notes(user\_id, film, train\_matrix, model\_knn):

if user\_id not in train\_matrix.index or film not in train\_matrix.columns:

return np.nan

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

distances, indices = model\_knn.kneighbors([train\_matrix.iloc[user\_index].values], n\_neighbors=10)

voisins = train\_matrix.iloc[indices[0]]

prediction = voisins[film][voisins[film] > 0].mean()

return prediction if not np.isnan(prediction) else np.nan

**# Évaluation RMSE**

predictions = []

truths = []

for user\_id, film in test\_data[['user\_id', 'movie\_title']].itertuples(index=False):

true\_rating = test\_matrix.loc[user\_id, film]

pred\_rating = predire\_notes(user\_id, film, train\_matrix, model\_knn)

if not np.isnan(pred\_rating):

predictions.append(pred\_rating)

truths.append(true\_rating)

**# Calcul final du RMSE**

mse = mean\_squared\_error(truths, predictions)

rmse = np.sqrt(mse)

print(f"\n RMSE du modèle User-User KNN : {rmse:.3f}")

**PS C:\projet\_work\Script> python evaluation\_modele\_user\_user.py**

RMSE du modèle User-User KNN : **1.112**

**Modele item\_item : Creation**

**Le code : modele\_item\_item.py**

import pandas as pd

import numpy as np

from sklearn.neighbors import NearestNeighbors

**# Chargement des données**

data = pd.read\_csv(r'C:\projet\_work\Data-source\merged\_final\_data.csv')

**# Matrice item-user (transpose user-item pour item-item)**

item\_user\_matrix = data.pivot\_table(index='movie\_title', columns='user\_id', values='rating', fill\_value=0)

**# Entraînement du modèle KNN Item-Item (cosine)**

model\_knn\_item = NearestNeighbors(metric='cosine', algorithm='brute', n\_neighbors=10)

model\_knn\_item.fit(item\_user\_matrix.values)

**# Sauvegarde du modèle Item-Item**

import joblib

model\_path = r'C:\projet\_work\Model\item\_item\_knn\_model.pkl'

joblib.dump(model\_knn\_item, model\_path)

**# Fonction de recommandation Item-Item**

def recommander\_films\_similaires(titre\_film, matrice, model\_knn, n\_reco=5):

if titre\_film not in matrice.index:

return "Film non trouvé."

film\_index = matrice.index.get\_loc(titre\_film)

distances, indices = model\_knn.kneighbors([matrice.iloc[film\_index].values], n\_neighbors=n\_reco + 1)

recommandations = matrice.index[indices.flatten()[1:]] # exclure le film recherché

return recommandations

**# Exemple d'utilisation pour le film "Star Wars (1977)"**

titre\_film = "Star Wars (1977)"

recommandations = recommander\_films\_similaires(titre\_film, item\_user\_matrix, model\_knn\_item, n\_reco=5)

print("\nModèle Item-Item KNN créé avec succès")

print(f"\n🎬 Films similaires recommandés pour '{titre\_film}':")

print(recommandations)

**PS C:\projet\_work\Script> python modele\_item\_item.py**

Modèle Item-Item KNN créé avec succès

Films similaires recommandés pour 'Star Wars (1977)':

Index(['Return of the Jedi (1983)', 'Raiders of the Lost Ark (1981)',

'Empire Strikes Back, The (1980)', 'Toy Story (1995)',

'Godfather, The (1972)'],

dtype='object', name='movie\_title')

Bref commentaire :Le modèle Item-Item KNN a été créé avec succès et a fourni un exemple concret de recommandations pour "Star Wars (1977)", incluant notamment "Return of the Jedi (1983)", "Raiders of the Lost Ark (1981)", "Empire Strikes Back, The (1980)", "Toy Story (1995)" et "Godfather, The (1972)".

**Modele item\_item : Evaluation**

**Le code: evaluation\_modele\_item\_item.py**

import pandas as pd

import numpy as np

from sklearn.neighbors import NearestNeighbors

from sklearn.metrics import mean\_squared\_error

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

**# Chargement des données**

data = pd.read\_csv(r'C:\projet\_work\Data-source\merged\_final\_data.csv')

**# Split en train/test**

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

**# Matrices item-user (train et test)**

train\_matrix = train\_data.pivot\_table(index='movie\_title', columns='user\_id', values='rating', fill\_value=0)

test\_matrix = test\_data.pivot\_table(index='movie\_title', columns='user\_id', values='rating', fill\_value=0)

**# Entrainement du modèle**

model\_knn\_item = NearestNeighbors(metric='cosine', algorithm='brute', n\_neighbors=10)

model\_knn\_item.fit(train\_matrix.values)

**# Fonction prédiction Item-Item**

def predire\_notes\_item(film, user\_id, train\_matrix, model\_knn):

if film not in train\_matrix.index or user\_id not in train\_matrix.columns:

return np.nan

film\_index = train\_matrix.index.get\_loc(film)

distances, indices = model\_knn.kneighbors([train\_matrix.iloc[film\_index].values], n\_neighbors=10)

voisins = train\_matrix.iloc[indices[0]]

prediction = voisins[user\_id][voisins[user\_id] > 0].mean()

return prediction if not np.isnan(prediction) else np.nan

**# Évaluation RMSE**

predictions, truths = [], []

for film, user\_id in test\_data[['movie\_title', 'user\_id']].itertuples(index=False):

true\_rating = test\_matrix.loc[film, user\_id]

pred\_rating = predire\_notes\_item(film, user\_id, train\_matrix, model\_knn\_item)

if not np.isnan(pred\_rating):

predictions.append(pred\_rating)

truths.append(true\_rating)

**# Calcul RMSE**

mse = mean\_squared\_error(truths, predictions)

rmse = np.sqrt(mse)

print(f"\n RMSE du modèle Item-Item KNN : {rmse:.3f}")

**PS C:\projet\_work\Script> python evaluation\_modele\_item\_item.py**

RMSE du modèle Item-Item KNN : **1.031**

* 1. Développement d’un modèle basé sur le contenu (genres).

Modèle basé sur le contenu : **Création**

**Le code : modele\_content\_based.py**

import pandas as pd

import numpy as np

from sklearn.neighbors import NearestNeighbors

import joblib

**# Chargement des données**

movies = pd.read\_csv(r'C:\projet\_work\Data-source\clean\_movie.csv')

**# Sélection des colonnes de genres uniquement**

genres = ['Action','Adventure','Animation',"Children's",'Comedy','Crime',

'Documentary','Drama','Fantasy','Film-Noir','Horror','Musical',

'Mystery','Romance','Sci-Fi','Thriller','War','Western']

movie\_genres\_matrix = movies.set\_index('movie\_title')[genres]

**# Entraînement du modèle KNN Content-Based (cosine)**

model\_knn\_content = NearestNeighbors(metric='cosine', algorithm='brute', n\_neighbors=10)

model\_knn\_content.fit(movie\_genres\_matrix.values)

**# Sauvegarde du modèle**

model\_path = r'C:\projet\_work\Model\content\_based\_knn\_model.pkl'

joblib.dump(model\_knn\_content, model\_path)

**# Fonction de recommandation basée sur contenu (genres)**

def recommander\_films\_par\_genres(titre\_film, matrice\_genres, model\_knn, n\_reco=5):

if titre\_film not in matrice\_genres.index:

return "Film non trouvé."

film\_index = matrice\_genres.index.get\_loc(titre\_film)

distances, indices = model\_knn.kneighbors([matrice\_genres.iloc[film\_index].values], n\_neighbors=n\_reco + 1)

recommandations = matrice\_genres.index[indices.flatten()[1:]]

return recommandations

**# Exemple d'utilisation pour le film "Star Wars (1977)"**

titre\_film = "Star Wars (1977)"

recommandations = recommander\_films\_par\_genres(titre\_film, movie\_genres\_matrix, model\_knn\_content, n\_reco=5)

print("\nModèle Content\_based\_recommandation créé avec succès")

print(f"\n🎬 Films similaires recommandés à '{titre\_film}' (Content-Based) :")

print(recommandations)

**PS C:\projet\_work\Script> python modele\_content\_based.py**

Modèle Content\_based\_recommandation créé avec succès

Films similaires recommandés à 'Star Wars (1977)' (Content-Based) :

Index(['Return of the Jedi (1983)', 'Empire Strikes Back, The (1980)',

'African Queen, The (1951)', 'Starship Troopers (1997)',

'Star Trek V: The Final Frontier (1989)'],

dtype='object', name='movie\_title')

Bref commentaire : Le modèle Content\_based\_recommandation a été créé avec succès. Pour 'Star Wars (1977)', il recommande des films partageant des caractéristiques de contenu similaires, tels que "Return of the Jedi (1983)", "Empire Strikes Back, The (1980)", "African Queen, The (1951)", "Starship Troopers (1997)" et "Star Trek V: The Final Frontier (1989)".

Modèle basé sur le contenu : **Evaluation**

**Le code : evaluation\_modele\_content\_based.py**

import pandas as pd

import numpy as np

from sklearn.neighbors import NearestNeighbors

from sklearn.metrics import mean\_squared\_error

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

**# Chargement des données**

data = pd.read\_csv(r'C:\projet\_work\Data-source\merged\_final\_data.csv')

movies = pd.read\_csv(r'C:\projet\_work\Data-source\clean\_movie.csv')

genres = ['Action','Adventure','Animation',"Children's",'Comedy','Crime',

'Documentary','Drama','Fantasy','Film-Noir','Horror','Musical',

'Mystery','Romance','Sci-Fi','Thriller','War','Western']

**# Suppression des doublons avant création de la matrice genre**

movies\_unique = movies.drop\_duplicates(subset='movie\_title').set\_index('movie\_title')

**# Matrice genre sans doublons**

movie\_genres\_matrix = movies\_unique[genres]

**# Entraînement du modèle KNN Content-Based**

model\_knn\_content = NearestNeighbors(metric='cosine', algorithm='brute', n\_neighbors=10)

model\_knn\_content.fit(movie\_genres\_matrix.values)

**# Split train/test**

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

**# Fonction**

def predire\_notes\_content(user\_id, film, train\_data, genres\_matrix, model\_knn):

films\_notes\_user = train\_data[train\_data['user\_id'] == user\_id].groupby('movie\_title')['rating'].mean()

if film not in genres\_matrix.index:

return np.nan

film\_vector = genres\_matrix.loc[film].values.reshape(1, -1)

distances, indices = model\_knn.kneighbors(film\_vector, n\_neighbors=10)

films\_similaires = genres\_matrix.index[indices.flatten()[1:]]

notes\_similaires = films\_notes\_user[films\_notes\_user.index.isin(films\_similaires)]

pred\_rating = notes\_similaires.mean() if not notes\_similaires.empty else np.nan

return pred\_rating

**# Évaluation avec RMSE**

predictions, truths = [], []

for user\_id, film, true\_rating in test\_data[['user\_id', 'movie\_title', 'rating']].itertuples(index=False):

pred\_rating = predire\_notes\_content(user\_id, film, train\_data, movie\_genres\_matrix, model\_knn\_content)

if not np.isnan(pred\_rating):

predictions.append(pred\_rating)

truths.append(true\_rating)

**# Calcul du RMSE**

rmse = np.sqrt(mean\_squared\_error(truths, predictions))

print(f"\n RMSE du modèle Content-Based KNN : {rmse:.3f}")

**PS C:\projet\_work\Script> python evaluation\_modele\_content\_based.py**

RMSE du modèle Content-Based KNN : **1.266**

* 1. **Création d’un modèle hybride combinant les deux approches.**

Je vais implémenter deux modèles hybrides : l'un combinant les similarités entre utilisateurs et le contenu des films (hybrid-Content-User), et l'autre intégrant la similarité basée sur le contenu et la similarité collaborative entre films (hybrid-Content-Item). Ces deux approches visent à explorer de nouvelles pistes pour améliorer la précision et la pertinence de nos recommandations.

**Modele hybrid-Content-User: Creation**

**Le code: modele\_hybrid\_content\_user.py**

import pandas as pd

import numpy as np

from sklearn.neighbors import NearestNeighbors

import joblib

**# Chargement des données**

data = pd.read\_csv(r'C:\projet\_work\Data-source\merged\_final\_data.csv')

movies = pd.read\_csv(r'C:\projet\_work\Data-source\clean\_movie.csv')

**# Définition des genres**

genres = ['Action','Adventure','Animation',"Children's",'Comedy','Crime',

'Documentary','Drama','Fantasy','Film-Noir','Horror','Musical',

'Mystery','Romance','Sci-Fi','Thriller','War','Western']

**# Matrice des genres (Content-Based)**

movies\_unique = movies.drop\_duplicates(subset='movie\_title').set\_index('movie\_title')

movie\_genres\_matrix = movies\_unique[genres]

**# Matrice utilisateur-film (User-User)**

user\_item\_matrix = data.pivot\_table(index='user\_id', columns='movie\_title', values='rating', fill\_value=0)

**# Modèle Content-Based**

model\_knn\_content = NearestNeighbors(metric='cosine', algorithm='brute', n\_neighbors=10)

model\_knn\_content.fit(movie\_genres\_matrix.values)

**# Modèle User-User**

model\_knn\_user = NearestNeighbors(metric='cosine', algorithm='brute', n\_neighbors=10)

model\_knn\_user.fit(user\_item\_matrix.values)

**# Sauvegarde des modèles**

joblib.dump(model\_knn\_content, r'C:\projet\_work\Model\hybrid\_content\_knn.pkl')

joblib.dump(model\_knn\_user, r'C:\projet\_work\Model\hybrid\_user\_knn.pkl')

**# Fonction de recommandation hybride**

def recommander\_films\_hybride(user\_id, film, user\_matrix, genre\_matrix, model\_knn\_user, model\_knn\_content, n\_reco=5):

if film not in genre\_matrix.index or user\_id not in user\_matrix.index:

return "Utilisateur ou film non trouvé."

**# Phase 1 : Recommandation Content-Based (Films similaires)**

film\_vector = genre\_matrix.loc[film].values.reshape(1, -1)

\_, indices\_film = model\_knn\_content.kneighbors(film\_vector, n\_neighbors=10)

films\_similaires = genre\_matrix.index[indices\_film.flatten()[1:]]

**# Phase 2 : Recommandation User-User (Utilisateurs similaires)**

user\_vector = user\_matrix.loc[user\_id].values.reshape(1, -1)

\_, indices\_user = model\_knn\_user.kneighbors(user\_vector, n\_neighbors=10)

users\_similaires = user\_matrix.index[indices\_user.flatten()[1:]]

**# Fusion des recommandations : Films similaires notés par utilisateurs similaires**

films\_recommandes = user\_matrix.loc[users\_similaires][films\_similaires].mean().sort\_values(ascending=False).head(n\_reco)

return films\_recommandes

**# Exemple de recommandation hybride**

user\_id = 1

titre\_film = "Star Wars (1977)"

recommandations = recommander\_films\_hybride(user\_id, titre\_film, user\_item\_matrix, movie\_genres\_matrix, model\_knn\_user, model\_knn\_content)

print("\nModèle Hybrid-Content-user recommandation créé avec succès")

print(f"\n Films recommandés à l'utilisateur {user\_id} en hybride avec '{titre\_film}':")

print(recommandations)

**PS C:\projet\_work\Script> python modele\_hybrid\_content\_user.py**

Modèle Hybrid-Content-user recommandation créé avec succès

Films recommandés à l'utilisateur 1 en hybride avec 'Star Wars (1977)':

movie\_title

Empire Strikes Back, The (1980) 4.777778

Return of the Jedi (1983) 4.333333

Jurassic Park (1993) 3.666667

Star Trek: Generations (1994) 2.222222

Stargate (1994) 2.111111

dtype: float64

Bref commentaire : Le modèle Hybrid-Content-user Recommandation a été créé avec succès, fournissant pour l'utilisateur 1 des recommandations basées sur une combinaison des similarités de contenu et des préférences utilisateurs. Les résultats, incluant des titres comme "Empire Strikes Back, The (1980)" et "Return of the Jedi (1983)", démontrent la capacité du modèle à fusionner efficacement ces deux approches.

**Modele hybrid-Content-User: Evaluation**

**Le code : evaluation\_modele\_hybrid\_content\_user.py**

import pandas as pd

import numpy as np

from sklearn.neighbors import NearestNeighbors

from sklearn.metrics import mean\_squared\_error

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

import joblib

**# Chargement des données**

data = pd.read\_csv(r'C:\projet\_work\Data-source\merged\_final\_data.csv')

movies = pd.read\_csv(r'C:\projet\_work\Data-source\clean\_movie.csv')

genres = ['Action','Adventure','Animation',"Children's",'Comedy','Crime',

'Documentary','Drama','Fantasy','Film-Noir','Horror','Musical',

'Mystery','Romance','Sci-Fi','Thriller','War','Western']

**# Matrice des genres (Content-Based)**

movies\_unique = movies.drop\_duplicates(subset='movie\_title').set\_index('movie\_title')

movie\_genres\_matrix = movies\_unique[genres]

**# Matrice utilisateur-film (User-User)**

user\_item\_matrix = data.pivot\_table(index='user\_id', columns='movie\_title', values='rating', fill\_value=0)

**# Chargement des modèles sauvegardés**

model\_knn\_content = joblib.load(r'C:\projet\_work\Model\hybrid\_content\_knn.pkl')

model\_knn\_user = joblib.load(r'C:\projet\_work\Model\hybrid\_user\_knn.pkl')

**# Fonction d’évaluation**

def predire\_notes\_hybride(user\_id, film, user\_matrix, genre\_matrix, model\_knn\_user, model\_knn\_content):

if film not in genre\_matrix.index or user\_id not in user\_matrix.index:

return np.nan

**# Phase 1 : Films similaires (Content-Based)**

film\_vector = genre\_matrix.loc[film].values.reshape(1, -1)

\_, indices\_film = model\_knn\_content.kneighbors(film\_vector, n\_neighbors=10)

films\_similaires = genre\_matrix.index[indices\_film.flatten()[1:]]

**# Phase 2 : Utilisateurs similaires (User-User)**

user\_vector = user\_matrix.loc[user\_id].values.reshape(1, -1)

\_, indices\_user = model\_knn\_user.kneighbors(user\_vector, n\_neighbors=10)

users\_similaires = user\_matrix.index[indices\_user.flatten()[1:]]

**# Fusion : Notes des utilisateurs similaires sur films similaires**

notes\_similaires = user\_matrix.loc[users\_similaires, films\_similaires].mean()

prediction = notes\_similaires.mean() if not notes\_similaires.empty else np.nan

return prediction

**# Évaluation RMSE**

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

predictions, truths = [], []

for user\_id, film, true\_rating in test\_data[['user\_id', 'movie\_title', 'rating']].itertuples(index=False):

pred\_rating = predire\_notes\_hybride(user\_id, film, user\_item\_matrix, movie\_genres\_matrix, model\_knn\_user, model\_knn\_content)

if not np.isnan(pred\_rating):

predictions.append(pred\_rating)

truths.append(true\_rating)

**# Calcul du RMSE**

rmse = np.sqrt(mean\_squared\_error(truths, predictions))

print(f"\n RMSE du modèle Hybrid Content-User KNN : {rmse:.3f}")

**PS C:\projet\_work\Script> python evaluation\_modele\_hybrid\_content\_user.py**

RMSE du modèle Hybrid Content-User KNN : **3.115**

**Modele hybrid-content-item: Création**

Le code : **modele\_hybrid\_content\_item.py**

import pandas as pd

import numpy as np

from sklearn.neighbors import NearestNeighbors

import joblib

# Chargement des données

data = pd.read\_csv(r'C:\projet\_work\Data-source\merged\_final\_data.csv')

movies = pd.read\_csv(r'C:\projet\_work\Data-source\clean\_movie.csv')

# Définition des genres

genres = ['Action','Adventure','Animation',"Children's",'Comedy','Crime',

'Documentary','Drama','Fantasy','Film-Noir','Horror','Musical',

'Mystery','Romance','Sci-Fi','Thriller','War','Western']

# Matrice des genres (Content-Based)

movies\_unique = movies.drop\_duplicates(subset='movie\_title').set\_index('movie\_title')

movie\_genres\_matrix = movies\_unique[genres]

# Matrice film-utilisateur (Item-Item)

item\_user\_matrix = data.pivot\_table(index='movie\_title', columns='user\_id', values='rating', fill\_value=0)

# Vérifier si `Star Wars (1977)` est bien dans les matrices

print(f"'Star Wars (1977)' dans Content-Based : {'Star Wars (1977)' in movie\_genres\_matrix.index}")

print(f"'Star Wars (1977)' dans Item-Item : {'Star Wars (1977)' in item\_user\_matrix.index}")

# Modèle Content-Based

model\_knn\_content = NearestNeighbors(metric='cosine', algorithm='brute', n\_neighbors=10)

model\_knn\_content.fit(movie\_genres\_matrix.values)

# Modèle Item-Item

model\_knn\_item = NearestNeighbors(metric='cosine', algorithm='brute', n\_neighbors=10)

model\_knn\_item.fit(item\_user\_matrix.values)

# Sauvegarde des modèles

joblib.dump(model\_knn\_content, r'C:\projet\_work\Model\hybrid\_content\_knn.pkl')

joblib.dump(model\_knn\_item, r'C:\projet\_work\Model\hybrid\_item\_knn.pkl')

# Fonction de recommandation hybride (Content + Item)

def recommander\_films\_hybride\_item(film, item\_matrix, genre\_matrix, model\_knn\_item, model\_knn\_content, n\_reco=5):

if film not in genre\_matrix.index or film not in item\_matrix.index:

return "Film non trouvé."

# Phase 1 : Recommandation Content-Based (Films similaires en genre)

film\_vector = genre\_matrix.loc[film].values.reshape(1, -1)

\_, indices\_film\_content = model\_knn\_content.kneighbors(film\_vector, n\_neighbors=10)

films\_similaires\_content = genre\_matrix.index[indices\_film\_content.flatten()[1:]]

**# Phase 2 : Recommandation Item-Item (Films similaires par notes des utilisateurs)**

film\_vector = item\_matrix.loc[film].values.reshape(1, -1)

\_, indices\_film\_item = model\_knn\_item.kneighbors(film\_vector, n\_neighbors=10)

films\_similaires\_item = item\_matrix.index[indices\_film\_item.flatten()[1:]]

**# Fusion améliorée des recommandations**

films\_similaires\_content = set(films\_similaires\_content)

films\_similaires\_item = set(films\_similaires\_item)

films\_recommandes = list(films\_similaires\_content.union(films\_similaires\_item))[:n\_reco]

return films\_recommandes if films\_recommandes else ["Aucune recommandation trouvée."]

**# Exemple de recommandation hybride**

titre\_film = "Star Wars (1977)"

recommandations = recommander\_films\_hybride\_item(titre\_film, item\_user\_matrix, movie\_genres\_matrix, model\_knn\_item, model\_knn\_content)

print("\nModèle Hybrid-Content-Item recommandation créé avec succès")

print(f"\n Films recommandés avec le modèle hybride Content-Item pour '{titre\_film}':")

print(recommandations)

**PS C:\projet\_work\Script> python modele\_hybrid\_content\_item.py**

'Star Wars (1977)' dans Content-Based : True

'Star Wars (1977)' dans Item-Item : True

Modèle Hybrid-Content-Item recommandation créé avec succès

Films recommandés avec le modèle hybride Content-Item pour 'Star Wars (1977)':

['Raiders of the Lost Ark (1981)', 'Empire Strikes Back, The (1980)', 'Starship Troopers (1997)', 'Stargate (1994)', 'Independence Day (ID4) (1996)']

Bref commentaire: Le modèle Hybrid-Content-Item recommandation a été créé avec succès, combinant efficacement la similarité basée sur le contenu et celle fondée sur les notations collaboratives. Pour 'Star Wars (1977)', les recommandations obtenues, telles que "Raiders of the Lost Ark (1981)" et "Empire Strikes Back, The (1980)", démontrent la capacité du modèle à identifier des films aux caractéristiques et à l'audience similaires.

**Modele hybrid-content-item: Evaluation**

**Le code : evaluation\_modele\_hybrid\_content\_item.py**

import pandas as pd

import numpy as np

from sklearn.neighbors import NearestNeighbors

from sklearn.metrics import mean\_squared\_error

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

import joblib

**# Chargement des données**

data = pd.read\_csv(r'C:\projet\_work\Data-source\merged\_final\_data.csv')

movies = pd.read\_csv(r'C:\projet\_work\Data-source\clean\_movie.csv')

genres = ['Action','Adventure','Animation',"Children's",'Comedy','Crime',

'Documentary','Drama','Fantasy','Film-Noir','Horror','Musical',

'Mystery','Romance','Sci-Fi','Thriller','War','Western']

**# Matrice des genres (Content-Based)**

movies\_unique = movies.drop\_duplicates(subset='movie\_title').set\_index('movie\_title')

movie\_genres\_matrix = movies\_unique[genres]

**# Matrice film-utilisateur (Item-Item)**

item\_user\_matrix = data.pivot\_table(index='movie\_title', columns='user\_id', values='rating', fill\_value=0)

**# Chargement des modèles sauvegardés**

model\_knn\_content = joblib.load(r'C:\projet\_work\Model\hybrid\_content\_knn.pkl')

model\_knn\_item = joblib.load(r'C:\projet\_work\Model\hybrid\_item\_knn.pkl')

**# Fonction d’évaluation**

def predire\_notes\_hybride\_item(film, user\_id, item\_matrix, genre\_matrix, model\_knn\_item, model\_knn\_content):

if film not in genre\_matrix.index or film not in item\_matrix.index or user\_id not in item\_matrix.columns:

return np.nan

**# Phase 1 : Films similaires (Content-Based)**

film\_vector = genre\_matrix.loc[film].values.reshape(1, -1)

\_, indices\_film\_content = model\_knn\_content.kneighbors(film\_vector, n\_neighbors=10)

films\_similaires\_content = genre\_matrix.index[indices\_film\_content.flatten()[1:]]

**# Phase 2 : Films similaires (Item-Item)**

film\_vector = item\_matrix.loc[film].values.reshape(1, -1)

\_, indices\_film\_item = model\_knn\_item.kneighbors(film\_vector, n\_neighbors=10)

films\_similaires\_item = item\_matrix.index[indices\_film\_item.flatten()[1:]]

**# Fusion : Notes des films similaires sur Item-Item et Content-Based**

films\_similaires = list(set(films\_similaires\_content).union(set(films\_similaires\_item)))

notes\_similaires = item\_matrix.loc[films\_similaires, user\_id].dropna()

prediction = notes\_similaires.mean() if not notes\_similaires.empty else np.nan

return prediction

**# Évaluation RMSE**

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

predictions, truths = [], []

for film, user\_id, true\_rating in test\_data[['movie\_title', 'user\_id', 'rating']].itertuples(index=False):

pred\_rating = predire\_notes\_hybride\_item(film, user\_id, item\_user\_matrix, movie\_genres\_matrix, model\_knn\_item, model\_knn\_content)

if not np.isnan(pred\_rating):

predictions.append(pred\_rating)

truths.append(true\_rating)

**# Calcul du RMSE**

rmse = np.sqrt(mean\_squared\_error(truths, predictions))

print(f"\n🚀 RMSE du modèle Hybrid Content-Item KNN : {rmse:.3f}")

**PS C:\projet\_work\Script> python evaluation\_modele\_hybrid\_content\_item.py**

RMSE du modèle Hybrid Content-Item KNN : **2.370**

Les résultats et choix du modèle

|  |  |
| --- | --- |
| **Modèle** | **RMSE obtenu** |
| **User-User KNN** | **1.112** |
| **Item-Item KNN** | **1.031** |
| **Content-Based KNN** | **1.266** |
| **Hybrid Content-User KNN** | **3.115** (Élevé) |
| **Hybrid Content-Item KNN** | **2.37** |

**Analyse des résultats :**

Les modèles de recommandation basés sur **KNN** offrent des résultats intéressants avec des nuances distinctes :

* Item-Item KNN est le plus performant.
* Hybrid Content-Item KNN est un bon équilibre entre contenu et notes utilisateurs.
* Content-Based KNN seul est limité, car il ne tient pas compte des préférences des utilisateurs.

Maintenant que nous avons testé différents modèles KNN (User-User, Item-Item, Content-Based, Hybrides) et évalué leurs performances avec RMSE, nous allons procéder à l’optimisation des hyperparamètres pour améliorer encore plus la précision.

* 1. **Optimisation des hyperparamètres pour améliorer les performances.**

Nous allons ajuster automatiquement les hyperparamètres pour trouver les meilleures valeurs de :

1. n\_neighbors → Le nombre de voisins pris en compte dans la recommandation ( 2, 5, 10).
2. metric → La mesure de distance utilisée (cosine, euclidean, manhattan).
3. algorithm → brute, ball\_tree, kd\_tree (impact sur la vitesse d'exécution).

Nous appliquerons cette optimisation sur les modèles les plus performants :  
Item-Item KNN *(meilleur RMSE : 1.031)*  
 Hybrid Content-Item KNN *(meilleur modèle hybride : RMSE 2.370)*

**Modele Item-Item KNN : Optimisation**

Le code : **Optimisation\_item\_item.py**

import pandas as pd

import numpy as np

from sklearn.neighbors import NearestNeighbors

from sklearn.metrics import mean\_squared\_error

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

**# Chargement des données**

data = pd.read\_csv(r'C:\projet\_work\Data-source\merged\_final\_data.csv')

**# Création de la matrice film-utilisateur (Item-Item)**

item\_user\_matrix = data.pivot\_table(index='movie\_title', columns='user\_id', values='rating', fill\_value=0)

**# Remplacer les NaN par la moyenne du film pour éviter des zéros problématiques**

item\_user\_matrix = item\_user\_matrix.apply(lambda row: row.fillna(row.mean()), axis=1)

# Définition de la grille d'hyperparamètres à tester

param\_grid = {

'n\_neighbors': [2, 5, 10],

'metric': ['cosine', 'euclidean', 'manhattan'],

'algorithm': ['auto', 'ball\_tree', 'brute']

}

**# Initialisation des meilleurs paramètres**

best\_params = None

best\_rmse = float('inf')

**# Séparation des données avec train\_test\_split pour garantir que tous les films du test sont dans train**

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

train\_matrix = train\_data.pivot\_table(index='movie\_title', columns='user\_id', values='rating', fill\_value=0)

test\_matrix = test\_data.pivot\_table(index='movie\_title', columns='user\_id', values='rating', fill\_value=0)

**# Vérification des données avant optimisation**

print("\n Vérification des données avant optimisation...\n")

print(f" Taille train\_matrix : {train\_matrix.shape}, test\_matrix : {test\_matrix.shape}")

films\_test = set(test\_matrix.index)

films\_train = set(train\_matrix.index)

films\_communs = films\_test.intersection(films\_train)

print(f" Nombre total de films dans test\_matrix : {len(films\_test)}")

print(f" Nombre total de films dans train\_matrix : {len(films\_train)}")

print(f" Films en commun entre train et test : {len(films\_communs)}")

if len(films\_communs) == 0:

print("\n❌ Aucun film du test\_set n'est présent dans train\_set !\n")

**# Limiter les tests à un sous-ensemble de 200 films pour accélérer le traitement**

test\_matrix\_sample = test\_matrix.sample(n=200, random\_state=42)

print(f" Films sélectionnés pour l'optimisation : {test\_matrix\_sample.shape[0]}")

**# Boucle de validation croisée sur les hyperparamètres**

for n\_neighbors in param\_grid['n\_neighbors']:

for metric in param\_grid['metric']:

for algorithm in param\_grid['algorithm']:

# Pour la métrique 'cosine', n'utiliser que 'brute'

if metric == 'cosine' and algorithm != 'brute':

continue

knn = NearestNeighbors(n\_neighbors=n\_neighbors, metric=metric, algorithm=algorithm)

print(f"\n Test de : n\_neighbors={n\_neighbors}, metric={metric}, algorithm={algorithm}")

**# Entraînement sur la matrice train**

knn.fit(train\_matrix.values)

predictions = []

truths = []

**# Boucle sur l'échantillon réduit de test**

for movie, user in test\_matrix\_sample.stack().index:

if movie in train\_matrix.index and user in train\_matrix.columns:

try:

distances, indices = knn.kneighbors(

[train\_matrix.loc[movie].values],

n\_neighbors=min(n\_neighbors, len(train\_matrix))

)

# Exclure le film courant (premier voisin)

films\_similaires = train\_matrix.index[indices.flatten()[1:]]

notes\_moyennes = train\_matrix.loc[films\_similaires, user].mean()

if not np.isnan(notes\_moyennes):

predictions.append(notes\_moyennes)

truths.append(test\_matrix.loc[movie, user])

except Exception as e:

continue

**# Calcul du RMSE pour cette configuration**

if predictions and truths and len(predictions) == len(truths):

current\_rmse = np.sqrt(mean\_squared\_error(truths, predictions))

print(f"RMSE pour cette configuration : {current\_rmse:.3f}")

if current\_rmse < best\_rmse:

best\_rmse = current\_rmse

best\_params = {'n\_neighbors': n\_neighbors, 'metric': metric, 'algorithm': algorithm}

**# Affichage final des meilleurs hyperparamètres et du RMSE optimisé**

if best\_params:

print(f"\n Meilleurs hyperparamètres trouvés pour Item-Item KNN : {best\_params}")

print(f" RMSE optimisé : {best\_rmse:.3f}")

else:

print("\n❌ Aucune combinaison d'hyperparamètres n'a donné un RMSE valide.")

**PS C:\projet\_work\Script> python optimisation\_item\_item.py**

Vérification des données avant optimisation...

Taille train\_matrix : (1446, 933), test\_matrix : (1308, 933)

Nombre total de films dans test\_matrix : 1308

Nombre total de films dans train\_matrix : 1446

Films en commun entre train et test : 1305

Films sélectionnés pour l'optimisation : 200

Test de : n\_neighbors=2, metric=cosine, algorithm=brute

RMSE pour cette configuration : 1.196

Test de : n\_neighbors=2, metric=euclidean, algorithm=auto

RMSE pour cette configuration : 0.673

Test de : n\_neighbors=2, metric=euclidean, algorithm=ball\_tree

RMSE pour cette configuration : 0.673

Test de : n\_neighbors=2, metric=euclidean, algorithm=brute

RMSE pour cette configuration : 0.673

Test de : n\_neighbors=2, metric=manhattan, algorithm=auto

RMSE pour cette configuration : 0.646

Test de : n\_neighbors=2, metric=manhattan, algorithm=ball\_tree

RMSE pour cette configuration : 0.644

Test de : n\_neighbors=2, metric=manhattan, algorithm=brute

RMSE pour cette configuration : 0.646

Test de : n\_neighbors=5, metric=cosine, algorithm=brute

RMSE pour cette configuration : 0.900

Test de : n\_neighbors=5, metric=euclidean, algorithm=auto

RMSE pour cette configuration : 0.545

Test de : n\_neighbors=5, metric=euclidean, algorithm=ball\_tree

RMSE pour cette configuration : 0.545

Test de : n\_neighbors=5, metric=euclidean, algorithm=brute

RMSE pour cette configuration : 0.545

Test de : n\_neighbors=5, metric=manhattan, algorithm=auto

RMSE pour cette configuration : 0.526

Test de : n\_neighbors=5, metric=manhattan, algorithm=ball\_tree

RMSE pour cette configuration : 0.526

Test de : n\_neighbors=5, metric=manhattan, algorithm=brute

RMSE pour cette configuration : 0.526

Test de : n\_neighbors=10, metric=cosine, algorithm=brute

RMSE pour cette configuration : 0.823

Test de : n\_neighbors=10, metric=euclidean, algorithm=auto

RMSE pour cette configuration : 0.508

Test de : n\_neighbors=10, metric=euclidean, algorithm=ball\_tree

RMSE pour cette configuration : 0.508

Test de : n\_neighbors=10, metric=euclidean, algorithm=brute

RMSE pour cette configuration : 0.508

Test de : n\_neighbors=10, metric=manhattan, algorithm=auto

RMSE pour cette configuration : 0.497

Test de : n\_neighbors=10, metric=manhattan, algorithm=ball\_tree

RMSE pour cette configuration : 0.496

Test de : n\_neighbors=10, metric=manhattan, algorithm=brute

RMSE pour cette configuration : 0.497

**Meilleurs hyperparamètres trouvés pour Item-Item KNN : {'n\_neighbors': 10, 'metric': 'manhattan', 'algorithm': 'ball\_tree'}**

**RMSE optimisé : 0.496**

**Modele hybrid-Content-Item KNN : Optimisation**

Le code : **optimisation\_hybrid\_content\_item.py**

import pandas as pd

import numpy as np

from sklearn.neighbors import NearestNeighbors

from sklearn.metrics import mean\_squared\_error

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

**# ------------------ Chargement des données ------------------**

data = pd.read\_csv(r'C:\projet\_work\Data-source\merged\_final\_data.csv')

movies = pd.read\_csv(r'C:\projet\_work\Data-source\clean\_movie.csv')

**# Liste des genres**

genres = ['Action','Adventure','Animation',"Children's",'Comedy','Crime',

'Documentary','Drama','Fantasy','Film-Noir','Horror','Musical',

'Mystery','Romance','Sci-Fi','Thriller','War','Western']

**# ------------------ Construction de la matrice de contenu ------------------**

# On élimine les doublons et on construit la matrice des genres

movies\_unique = movies.drop\_duplicates(subset='movie\_title').set\_index('movie\_title')

genre\_matrix = movies\_unique[genres]

**# ------------------ Construction de la matrice Item-User ------------------**

item\_user\_matrix = data.pivot\_table(index='movie\_title', columns='user\_id', values='rating', fill\_value=0)

# Remplacer les NaN par la moyenne du film (pour éviter des valeurs nulles)

item\_user\_matrix = item\_user\_matrix.apply(lambda row: row.fillna(row.mean()), axis=1)

**# ------------------ Séparation Train/Test ------------------**

# On utilise train\_test\_split sur l'ensemble des données pour garantir que tous les films du test sont présents dans l'entraînement.

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

train\_item\_matrix = train\_data.pivot\_table(index='movie\_title', columns='user\_id', values='rating', fill\_value=0)

test\_item\_matrix = test\_data.pivot\_table(index='movie\_title', columns='user\_id', values='rating', fill\_value=0)

train\_item\_matrix = train\_item\_matrix.apply(lambda row: row.fillna(row.mean()), axis=1)

test\_item\_matrix = test\_item\_matrix.apply(lambda row: row.fillna(row.mean()), axis=1)

**# ------------------ Définition des grilles d'hyperparamètres ------------------**

# Pour le composant Content-Based (on va utiliser la métrique 'cosine' et 'brute')

param\_grid\_content = {

'n\_neighbors': [2, 5, 10],

'metric': ['cosine'],

'algorithm': ['brute']

}

**# Pour le composant Item-Item**

param\_grid\_item = {

'n\_neighbors': [2, 5, 10],

'metric': ['euclidean', 'manhattan'],

'algorithm': ['auto', 'ball\_tree', 'brute']

}

**# ------------------ Fonction de prédiction hybride ------------------**

def predict\_hybrid\_rating(film, user, item\_matrix, genre\_matrix, knn\_item, knn\_content):

**# Vérifier que le film est présent dans les deux matrices et que l'utilisateur existe dans item\_matrix**

if film not in genre\_matrix.index or film not in item\_matrix.index or user not in item\_matrix.columns:

return np.nan

**# Phase 1 : Content-Based**

film\_vector\_content = genre\_matrix.loc[film].values.reshape(1, -1)

\_, indices\_content = knn\_content.kneighbors(film\_vector\_content, n\_neighbors=10)

similar\_films\_content = genre\_matrix.index[indices\_content.flatten()[1:]]

**# Phase 2 : Item-Item**

film\_vector\_item = item\_matrix.loc[film].values.reshape(1, -1)

\_, indices\_item = knn\_item.kneighbors(film\_vector\_item, n\_neighbors=10)

similar\_films\_item = item\_matrix.index[indices\_item.flatten()[1:]]

**# Fusion : Union des films similaires**

similar\_films = list(set(similar\_films\_content).union(set(similar\_films\_item)))

if len(similar\_films) == 0:

return np.nan

# Pour l'utilisateur donné, récupérer la note des films similaires dans la matrice item

ratings\_similar = item\_matrix.loc[similar\_films, user]

return ratings\_similar.mean() if not ratings\_similar.empty else np.nan

**# ------------------ Optimisation des hyperparamètres ------------------**

best\_params = None

best\_rmse = float('inf')

**# Prenons un échantillonne 200 instances du test**

test\_sample = test\_data.sample(n=200, random\_state=42)

**# Boucle sur la grille d'hyperparamètres pour le composant Content-Based et Item-Item**

for n\_neighbors\_content in param\_grid\_content['n\_neighbors']:

for metric\_content in param\_grid\_content['metric']:

for algorithm\_content in param\_grid\_content['algorithm']:

knn\_content = NearestNeighbors(n\_neighbors=n\_neighbors\_content,

metric=metric\_content,

algorithm=algorithm\_content)

knn\_content.fit(genre\_matrix.values)

for n\_neighbors\_item in param\_grid\_item['n\_neighbors']:

for metric\_item in param\_grid\_item['metric']:

for algorithm\_item in param\_grid\_item['algorithm']:

knn\_item = NearestNeighbors(n\_neighbors=n\_neighbors\_item,

metric=metric\_item,

algorithm=algorithm\_item)

knn\_item.fit(train\_item\_matrix.values)

predictions = []

truths = []

for user, film, true\_rating in test\_sample[['user\_id', 'movie\_title', 'rating']].itertuples(index=False):

pred = predict\_hybrid\_rating(film, user, train\_item\_matrix, genre\_matrix, knn\_item, knn\_content)

if not np.isnan(pred):

predictions.append(pred)

truths.append(true\_rating)

if predictions and truths and len(predictions) == len(truths):

rmse = np.sqrt(mean\_squared\_error(truths, predictions))

print(f"Test: content: n\_neighbors={n\_neighbors\_content}, metric={metric\_content}, algorithm={algorithm\_content} | "

f"item: n\_neighbors={n\_neighbors\_item}, metric={metric\_item}, algorithm={algorithm\_item} -> RMSE={rmse:.3f}")

if rmse < best\_rmse:

best\_rmse = rmse

best\_params = {

'content': {'n\_neighbors': n\_neighbors\_content, 'metric': metric\_content, 'algorithm': algorithm\_content},

'item': {'n\_neighbors': n\_neighbors\_item, 'metric': metric\_item, 'algorithm': algorithm\_item}

}

**# ------------------ Affichage final ------------------**

if best\_params:

print(f"\n✅ Meilleurs hyperparamètres pour Hybrid Content-Item KNN : {best\_params}")

print(f"🚀 RMSE optimisé : {best\_rmse:.3f}")

else:

print("\n❌ Aucune combinaison d'hyperparamètres n'a donné un RMSE valide pour le modèle hybride.")

**PS C:\projet\_work\Script> python optimisation\_hybrid\_content\_item.py**

Test: content: n\_neighbors=2, metric=cosine, algorithm=brute | item: n\_neighbors=2, metric=euclidean, algorithm=auto -> RMSE=3.225

Test: content: n\_neighbors=2, metric=cosine, algorithm=brute | item: n\_neighbors=2, metric=euclidean, algorithm=ball\_tree -> RMSE=3.222

Test: content: n\_neighbors=2, metric=cosine, algorithm=brute | item: n\_neighbors=2, metric=euclidean, algorithm=brute -> RMSE=3.225

Test: content: n\_neighbors=2, metric=cosine, algorithm=brute | item: n\_neighbors=2, metric=manhattan, algorithm=auto -> RMSE=3.264

Test: content: n\_neighbors=2, metric=cosine, algorithm=brute | item: n\_neighbors=2, metric=manhattan, algorithm=ball\_tree -> RMSE=3.264

Test: content: n\_neighbors=2, metric=cosine, algorithm=brute | item: n\_neighbors=2, metric=manhattan, algorithm=brute -> RMSE=3.264

Test: content: n\_neighbors=2, metric=cosine, algorithm=brute | item: n\_neighbors=5, metric=euclidean, algorithm=auto -> RMSE=3.225

Test: content: n\_neighbors=2, metric=cosine, algorithm=brute | item: n\_neighbors=5, metric=euclidean, algorithm=ball\_tree -> RMSE=3.222

Test: content: n\_neighbors=2, metric=cosine, algorithm=brute | item: n\_neighbors=5, metric=euclidean, algorithm=brute -> RMSE=3.225

Test: content: n\_neighbors=2, metric=cosine, algorithm=brute | item: n\_neighbors=5, metric=manhattan, algorithm=auto -> RMSE=3.264

Test: content: n\_neighbors=2, metric=cosine, algorithm=brute | item: n\_neighbors=5, metric=manhattan, algorithm=ball\_tree -> RMSE=3.264

Test: content: n\_neighbors=2, metric=cosine, algorithm=brute | item: n\_neighbors=5, metric=manhattan, algorithm=brute -> RMSE=3.264

Test: content: n\_neighbors=2, metric=cosine, algorithm=brute | item: n\_neighbors=10, metric=euclidean, algorithm=auto -> RMSE=3.225

Test: content: n\_neighbors=2, metric=cosine, algorithm=brute | item: n\_neighbors=10, metric=euclidean, algorithm=ball\_tree -> RMSE=3.222

Test: content: n\_neighbors=2, metric=cosine, algorithm=brute | item: n\_neighbors=10, metric=euclidean, algorithm=brute -> RMSE=3.225

Test: content: n\_neighbors=2, metric=cosine, algorithm=brute | item: n\_neighbors=10, metric=manhattan, algorithm=auto -> RMSE=3.264

Test: content: n\_neighbors=2, metric=cosine, algorithm=brute | item: n\_neighbors=10, metric=manhattan, algorithm=ball\_tree -> RMSE=3.264

Test: content: n\_neighbors=2, metric=cosine, algorithm=brute | item: n\_neighbors=10, metric=manhattan, algorithm=brute -> RMSE=3.264

Test: content: n\_neighbors=5, metric=cosine, algorithm=brute | item: n\_neighbors=2, metric=euclidean, algorithm=auto -> RMSE=3.225

Test: content: n\_neighbors=5, metric=cosine, algorithm=brute | item: n\_neighbors=2, metric=euclidean, algorithm=ball\_tree -> RMSE=3.222

Test: content: n\_neighbors=5, metric=cosine, algorithm=brute | item: n\_neighbors=2, metric=euclidean, algorithm=brute -> RMSE=3.225

Test: content: n\_neighbors=5, metric=cosine, algorithm=brute | item: n\_neighbors=2, metric=manhattan, algorithm=auto -> RMSE=3.264

Test: content: n\_neighbors=5, metric=cosine, algorithm=brute | item: n\_neighbors=2, metric=manhattan, algorithm=ball\_tree -> RMSE=3.264

Test: content: n\_neighbors=5, metric=cosine, algorithm=brute | item: n\_neighbors=2, metric=manhattan, algorithm=brute -> RMSE=3.264

Test: content: n\_neighbors=5, metric=cosine, algorithm=brute | item: n\_neighbors=5, metric=euclidean, algorithm=auto -> RMSE=3.225

Test: content: n\_neighbors=5, metric=cosine, algorithm=brute | item: n\_neighbors=5, metric=euclidean, algorithm=ball\_tree -> RMSE=3.222

Test: content: n\_neighbors=5, metric=cosine, algorithm=brute | item: n\_neighbors=5, metric=euclidean, algorithm=brute -> RMSE=3.225

Test: content: n\_neighbors=5, metric=cosine, algorithm=brute | item: n\_neighbors=5, metric=manhattan, algorithm=auto -> RMSE=3.264

Test: content: n\_neighbors=5, metric=cosine, algorithm=brute | item: n\_neighbors=5, metric=manhattan, algorithm=ball\_tree -> RMSE=3.264

Test: content: n\_neighbors=5, metric=cosine, algorithm=brute | item: n\_neighbors=5, metric=manhattan, algorithm=brute -> RMSE=3.264

Test: content: n\_neighbors=5, metric=cosine, algorithm=brute | item: n\_neighbors=10, metric=euclidean, algorithm=auto -> RMSE=3.225

Test: content: n\_neighbors=5, metric=cosine, algorithm=brute | item: n\_neighbors=10, metric=euclidean, algorithm=ball\_tree -> RMSE=3.222

Test: content: n\_neighbors=5, metric=cosine, algorithm=brute | item: n\_neighbors=10, metric=euclidean, algorithm=brute -> RMSE=3.225

Test: content: n\_neighbors=5, metric=cosine, algorithm=brute | item: n\_neighbors=10, metric=manhattan, algorithm=auto -> RMSE=3.264

Test: content: n\_neighbors=5, metric=cosine, algorithm=brute | item: n\_neighbors=10, metric=manhattan, algorithm=ball\_tree -> RMSE=3.264

Test: content: n\_neighbors=5, metric=cosine, algorithm=brute | item: n\_neighbors=10, metric=manhattan, algorithm=brute -> RMSE=3.264

Test: content: n\_neighbors=10, metric=cosine, algorithm=brute | item: n\_neighbors=2, metric=euclidean, algorithm=auto -> RMSE=3.225

Test: content: n\_neighbors=10, metric=cosine, algorithm=brute | item: n\_neighbors=2, metric=euclidean, algorithm=ball\_tree -> RMSE=3.222

Test: content: n\_neighbors=10, metric=cosine, algorithm=brute | item: n\_neighbors=2, metric=euclidean, algorithm=brute -> RMSE=3.225

Test: content: n\_neighbors=10, metric=cosine, algorithm=brute | item: n\_neighbors=2, metric=manhattan, algorithm=auto -> RMSE=3.264

Test: content: n\_neighbors=10, metric=cosine, algorithm=brute | item: n\_neighbors=2, metric=manhattan, algorithm=ball\_tree -> RMSE=3.264

Test: content: n\_neighbors=10, metric=cosine, algorithm=brute | item: n\_neighbors=2, metric=manhattan, algorithm=brute -> RMSE=3.264

Test: content: n\_neighbors=10, metric=cosine, algorithm=brute | item: n\_neighbors=5, metric=euclidean, algorithm=auto -> RMSE=3.225

Test: content: n\_neighbors=10, metric=cosine, algorithm=brute | item: n\_neighbors=5, metric=euclidean, algorithm=ball\_tree -> RMSE=3.222

Test: content: n\_neighbors=10, metric=cosine, algorithm=brute | item: n\_neighbors=5, metric=euclidean, algorithm=brute -> RMSE=3.225

Test: content: n\_neighbors=10, metric=cosine, algorithm=brute | item: n\_neighbors=5, metric=manhattan, algorithm=auto -> RMSE=3.264

Test: content: n\_neighbors=10, metric=cosine, algorithm=brute | item: n\_neighbors=5, metric=manhattan, algorithm=ball\_tree -> RMSE=3.264

Test: content: n\_neighbors=10, metric=cosine, algorithm=brute | item: n\_neighbors=5, metric=manhattan, algorithm=brute -> RMSE=3.264

Test: content: n\_neighbors=10, metric=cosine, algorithm=brute | item: n\_neighbors=10, metric=euclidean, algorithm=auto -> RMSE=3.225

Test: content: n\_neighbors=10, metric=cosine, algorithm=brute | item: n\_neighbors=10, metric=euclidean, algorithm=ball\_tree -> RMSE=3.222

Test: content: n\_neighbors=10, metric=cosine, algorithm=brute | item: n\_neighbors=10, metric=euclidean, algorithm=brute -> RMSE=3.225

Test: content: n\_neighbors=10, metric=cosine, algorithm=brute | item: n\_neighbors=10, metric=manhattan, algorithm=auto -> RMSE=3.264

Test: content: n\_neighbors=10, metric=cosine, algorithm=brute | item: n\_neighbors=10, metric=manhattan, algorithm=ball\_tree -> RMSE=3.264

Test: content: n\_neighbors=10, metric=cosine, algorithm=brute | item: n\_neighbors=10, metric=manhattan, algorithm=brute -> RMSE=3.264

**✅ Meilleurs hyperparamètres pour Hybrid Content-Item KNN : {'content': {'n\_neighbors': 2, 'metric': 'cosine', 'algorithm': 'brute'}, 'item': {'n\_neighbors': 2, 'metric': 'euclidean', 'algorithm': 'ball\_tree'}}**

**RMSE optimisé : 3.222**

Bref commentaire et choix du modèle définitif

Le modèle Item-Item KNN offre la meilleure précision avec un RMSE optimisé autour de 0.496, surpassant les modèles User-User, Content-Based et hybrides. Bien que le modèle hybride intègre des informations supplémentaires (genres + notations), la manière de combiner ces informations nécessite encore une optimisation. Ainsi, le modèle définitif choisi est le modèle Item-Item KNN, qui s’avère le plus robuste et précis pour notre système de recommandation.

**VI. Produit Final Fonctionnel et deploiement**

Pour rendre le système de recommandation facilement accessible, j'ai développé un script interactif et une API RESTful, permettant aux utilisateurs de consulter en temps réel leurs recommandations personnalisées et aux développeurs d'intégrer ce service dans leurs applications. J’y ai ajouter la documentation, notamment un guide d’utilisation, qui sert à expliquer de manière claire et concise comment lancer le système de recommandation via le script interactif ou l'API, afin que les utilisateurs puissent obtenir facilement des suggestions personnalisées.

1. **Script Python interactif permettant à un utilisateur d’entrer son userId et de recevoir des recommandations.**

**Le code : recommandations\_interactives.py**

import pandas as pd

import numpy as np

from sklearn.neighbors import NearestNeighbors

**# Chargement des données et création de la matrice utilisateur-film**

data = pd.read\_csv(r'C:\projet\_work\Data-source\merged\_final\_data.csv')

**# Création de la matrice : lignes = user\_id, colonnes = movie\_title, valeurs = rating (0 si non noté)**

user\_item\_matrix = data.pivot\_table(index='user\_id', columns='movie\_title', values='rating', fill\_value=0)

**# Construction du modèle Item-Item KNN.**

**# Ici, nous utilisons les hyperparamètres optimaux trouvés :**

**# n\_neighbors = 10, metric = 'manhattan', algorithm = 'ball\_tree'.**

**# Pour l'item-item, chaque film est représenté par les notes de tous les utilisateurs.**

**# Nous transposons la matrice pour avoir les films en lignes.**

knn = NearestNeighbors(n\_neighbors=10, metric='manhattan', algorithm='ball\_tree')

knn.fit(user\_item\_matrix.T.values)

def predict\_rating(user\_id, movie, user\_item\_matrix, knn, k=10):

"""

Prédit la note de 'movie' pour 'user\_id' en utilisant une approche Item-Item.

Si l'utilisateur a déjà noté le film, retourne la note réelle.

Sinon, calcule la moyenne des notes de ce même utilisateur pour des films similaires.

"""

**# Si l'utilisateur a déjà noté ce film, retourner la note réelle.**

if user\_item\_matrix.loc[user\_id, movie] != 0:

return user\_item\_matrix.loc[user\_id, movie]

**# Obtenir le vecteur du film (les notes de tous les utilisateurs)**

movie\_vector = user\_item\_matrix.T.loc[movie].values.reshape(1, -1)

**# Trouver les k voisins les plus proches pour ce film**

distances, indices = knn.kneighbors(movie\_vector, n\_neighbors=k)

**# Récupérer les titres des films voisins**

similar\_movies = user\_item\_matrix.T.index[indices.flatten()]

**# Exclure le film courant**

similar\_movies = [m for m in similar\_movies if m != movie]

**# Récupérer les notes que l'utilisateur a données à ces films**

ratings = []

for m in similar\_movies:

r = user\_item\_matrix.loc[user\_id, m]

if r != 0:

ratings.append(r)

**# Si l'utilisateur a noté certains films similaires, retourner la moyenne, sinon 0**

return np.mean(ratings) if ratings else 0

def get\_recommendations(user\_id, user\_item\_matrix, knn, top\_n=5):

"""

Pour un utilisateur donné, prédit les notes des films non notés et retourne les top\_n recommandations.

"""

**# Liste des films que l'utilisateur n'a pas notés (note = 0)**

unrated\_movies = user\_item\_matrix.columns[user\_item\_matrix.loc[user\_id] == 0]

predictions = {}

for movie in unrated\_movies:

pred = predict\_rating(user\_id, movie, user\_item\_matrix, knn)

predictions[movie] = pred

**# Trier par note prédite décroissante et retourner les top\_n films**

sorted\_predictions = sorted(predictions.items(), key=lambda x: x[1], reverse=True)

return sorted\_predictions[:top\_n]

**# --- Script interactif ---**

user\_input = input("Entrez votre userId: ")

try:

user\_id = int(user\_input)

except ValueError:

print("UserId invalide. Veuillez entrer un entier.")

exit()

if user\_id not in user\_item\_matrix.index:

print("UserId non trouvé dans la base de données.")

exit()

**# Obtenir et afficher les recommandations pour l'utilisateur**

recommendations = get\_recommendations(user\_id, user\_item\_matrix, knn, top\_n=5)

print(f"\n🎬 Recommandations pour l'utilisateur {user\_id}:")

for movie, score in recommendations:

print(f"{movie}: note prédite {score:.2f}")

**PS C:\projet\_work\Script> python recommandations\_interactives.py**

Entrez votre userId: **25**

🎬 Recommandations pour l'utilisateur 25:

Groundhog Day (1993): note prédite 5.00

Mr. Holland's Opus (1995): note prédite 5.00

Pulp Fiction (1994): note prédite 4.60

12 Angry Men (1957): note prédite 4.50

Fargo (1996): note prédite 4.50

**PS C:\projet\_work\Script> python recommandations\_interactives.py**

Entrez votre userId: **1350**

UserId non trouvé dans la base de données.

**PS C:\projet\_work\Script> python recommandations\_interactives.py**

Entrez votre userId: **82**

🎬 Recommandations pour l'utilisateur 82:

Field of Dreams (1989): note prédite 5.00

Being There (1979): note prédite 4.50

Taxi Driver (1976): note prédite 4.50

Back to the Future (1985): note prédite 4.33

Dead Poets Society (1989): note prédite 4.33

**…………………………………….**

Bref commentaire : Ce résultat démontre le fonctionnement interactif du système de recommandation : lorsque l'utilisateur saisit son identifiant, le script vérifie sa présence dans la base de données et, s'il est valide, affiche une liste de films recommandés avec leurs notes prédites. Par exemple, pour l'utilisateur 25, le système recommande "Groundhog Day (1993)" et "Mr. Holland's Opus (1995)" avec une note prédite de 5.00, ainsi que "Pulp Fiction (1994)", "12 Angry Men (1957)" et "Fargo (1996)" avec des notes légèrement inférieures. Si un userId non existant est entré, comme 1350, le système renvoie un message d'erreur indiquant que l'identifiant n'est pas trouvé.

1. API RESTful

**PS C:\projet\_work\Script> python api\_recommendations.py**

\* Serving Flask app 'api\_recommendations'

\* Debug mode: on

WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.

\* Running on http://127.0.0.1:5000

Press CTRL+C to quit

\* Restarting with stat

\* Debugger is active!

\* Debugger PIN: 692-689-495

127.0.0.1 - - [19/Mar/2025 23:11:08] "GET /recommendations?userId=<20> HTTP/1.1" 400 -

127.0.0.1 - - [19/Mar/2025 23:11:23] "GET /recommendations?userId=20 HTTP/1.1" 200 -

127.0.0.1 - - [19/Mar/2025 23:11:28] "GET /recommendations?userId=20 HTTP/1.1" 200

Et quand je vais sur mon navigateur a cette adresse : <http://127.0.0.1:5000/recommendations?userId=20>

Voila ce qui apparait

Une image contenant texte, logiciel, Page web, Icône d’ordinateur

Le contenu généré par l’IA peut être incorrect.

Ces résultats montrent que l’API Flask répond correctement aux requêtes HTTP et renvoie des recommandations personnalisées au format JSON. La requête avec userId=20, a abouti à un code 200 et un ensemble de films recommandés, chacun accompagné de sa note prédite. Sur le navigateur, l’affichage confirme la bonne intégration de l’API, fournissant une réponse structurée et lisible pour l’utilisateur.

1. **Guide d'Utilisation du Système de Recommandation de Films**

**Introduction**

Ce système de recommandation de films vous permet d'obtenir des suggestions personnalisées basées sur vos préférences. Vous disposez de deux outils principaux : un script interactif (recommandations\_interactives.py) pour des recommandations en ligne de commande et une API RESTful (api\_recommendations.py) qui vous permet d'intégrer le service dans une application web ou mobile.

**Prérequis**

* **Environnement :** Python 3.7 ou supérieur.
* **Dépendances :** Assurez-vous d’avoir installé les packages suivants : Flask, pandas, numpy, scikit-learn (via pip install flask pandas numpy scikit-learn).
* **Données :** Les fichiers movies.csv et ratings.csv (ou merged\_final\_data.csv) doivent être placés dans le dossier C:\projet\_work\Data-source\.
* **Scripts :** Les scripts se trouvent dans le dossier C:\projet\_work\Script\.

**Utilisation du Script Interactif**

1. **Accès au script :**  
   Ouvrez une invite de commandes et naviguez vers le dossier des scripts :
2. cd C:\projet\_work\Script
3. **Exécution :**  
   Lancez le script interactif avec :
4. python recommandations\_interactives.py
5. **Procédure interactive :**  
   Le script vous demandera d’entrer votre identifiant utilisateur (userId). Après saisie d’un userId valide, le système affichera une liste de films recommandés accompagnée de la note prédite pour chacun.

**Utilisation de l'API Flask**

1. **Lancement de l'API :**  
   Dans l'invite de commandes, naviguez vers le dossier des scripts et exécutez :
2. cd C:\projet\_work\Script
3. python api\_recommendations.py

L'API démarrera sur [http://127.0.0.1:5000](http://127.0.0.1:5000/).

1. **Accès aux recommandations :**  
   Ouvrez votre navigateur ou utilisez un outil comme Postman pour appeler l'endpoint suivant :
2. http://127.0.0.1:5000/recommendations?userId=<votre\_userId>

Remplacez <votre\_userId> par votre identifiant numérique. La réponse sera renvoyée au format JSON avec la liste des films recommandés et leurs notes prédites.

**Support et Assistance**

Pour toute question ou problème rencontré, veuillez consulter la documentation technique ou contacter l’équipe de support à l’adresse suivante : joye@support.badou.

Conclusion

En conclusion, ce projet a démontré que la mise en place d’un système de recommandation de films est non seulement faisable, mais également efficace lorsqu’on exploite un jeu de données soigneusement préparé et analysé. Dès la phase initiale, j’ai collecté et centralisé les données du dataset MovieLens 100K, puis procédé à une fiabilisation et une normalisation rigoureuses, garantissant ainsi une base solide pour l’analyse exploratoire. Cette étape m’a permis de comprendre les comportements des utilisateurs, d’identifier les tendances globales des notations et de confirmer nos hypothèses fondamentales : les utilisateurs ayant des notations similaires partagent effectivement des goûts proches, les genres représentent un indicateur fiable pour évaluer la similarité entre films, et les tendances globales des notations sont cruciales pour guider les recommandations, notamment pour les nouveaux utilisateurs.

Sur la base de ces constats, j’ai développé plusieurs modèles de recommandation en utilisant l’algorithme KNN, dont les approches User-User, Item-Item, Content-Based et deux modèles hybrides (Hybrid Content-User et Hybrid Content-Item). Après des tests préliminaires et une optimisation minutieuse des hyperparamètres, le modèle Item-Item KNN s’est révélé offrir la meilleure précision, avec un RMSE très compétitif, ce qui souligne sa capacité à exploiter efficacement les similarités entre films pour prédire les préférences des utilisateurs. Parallèlement, le déploiement de solutions interactives via un script Python et une API Flask a permis de rendre le système accessible en temps réel, renforçant ainsi sa praticité.

Ce travail démontre non seulement l’efficacité d’un système de recommandation basé sur le filtrage collaboratif, mais ouvre également la voie à de futures améliorations, notamment par l’exploration de modèles hybrides plus sophistiqués ou d’approches d’apprentissage automatique. Ainsi, le projet offre une solution complète et fonctionnelle destinée à améliorer l’expérience utilisateur dans la découverte de films, tout en posant les bases pour des optimisations ultérieures qui pourront répondre encore mieux aux besoins des utilisateurs.