PROJET DS
“Système de recommandation de films”

—

Ana Vivas Occhipinti
Ariel Assayag
Charles Yahi
Lam Paturle

Github : https://github.com/DataScientest-Studio/SEP24-BDS-RECO_FILM

Table des matières

Étape 1 : rapport d’exploration, de data visualisation et de pre-processing des données

Introduction au projet

Contexte

Les systèmes de recommandation occupent une place centrale dans le domaine du machine learning (apprentissage automatique), car ils sont largement utilisés pour personnaliser l'expérience utilisateur dans divers secteurs, tels que le commerce électronique, les plateformes de streaming, les réseaux sociaux et bien d'autres.

Ils se basent sur diverses techniques du machine learning (filtrage collaboratif, apprentissage supervisé, non supervisé, deep learning) pour fournir des recommandations personnalisées. Leur efficacité repose sur leur capacité à traiter des volumes importants de données et à s'adapter aux préférences des utilisateurs, tout en gérant des défis comme le démarrage à froid, la scalabilité, et les biais algorithmiques.

Objectifs

- Apprendre à travailler en équipe autour d’une problématique classique de la data science
- Apprendre à préparer une base de données : riche, propre, utile, optimisée, etc.
- Apprendre à sélectionner et ajuster un modèle de machine learning en fonction de la problématique, des variables explicatives et de la cible
- Apprendre à créer concrètement un système de recommandation en fonction de l’utilisateur et du contenu (e.g. prédire de manière pertinente et précise le rating d’un film)
- Apprendre à présenter les résultats de nos recherches avec les outils et selon les standards de la data science

Défis

Problème du démarrage à froid (Cold Start Problem)

L'un des principaux défis des systèmes de recommandation est le démarrage à froid, qui survient lorsqu'un nouveau produit ou un nouvel utilisateur n'a pas encore d'historique d'interactions. Les systèmes doivent alors s'appuyer sur des informations externes (basées sur le contenu ou des données démographiques) pour faire des recommandations.

Scalabilité

Les systèmes de recommandation doivent souvent traiter un grand volume de données, en particulier sur les grandes plateformes. Les algorithmes de recommandation doivent donc être scalables pour pouvoir fournir des résultats en temps réel, même avec des millions d'utilisateurs et d'articles.

Biais et diversité

Les recommandations peuvent parfois conduire à un biais ou à un manque de diversité, ce qui signifie que les utilisateurs peuvent être limités à des recommandations très similaires à leurs comportements passés, sans être exposés à de nouveaux contenus. Les systèmes de recommandation doivent donc équilibrer la personnalisation avec la diversité pour maximiser l'engagement.

Enjeux éthiques et confidentialité

Les systèmes de recommandation soulèvent également des questions importantes en matière de confidentialité des données et d'éthique. Les recommandations sont basées sur la collecte de grandes quantités de données personnelles, et il est important que les entreprises respectent la vie privée des utilisateurs et évitent de créer des systèmes qui renforcent les biais ou créent des bulles de filtrage.

Méthode de travail

Composition de l’équipe

L’équipe est composée de quatre membres aux profils complémentaires :

- Ana : actuaire de formation
- Ariel : ingénieur en génie civil
- Charles : développeur de formation, intégré dans l’équipe à partir du 31 octobre 2024
- Lam : product manager dans la production et l'édition de jeux vidéo

Répartition des tâches

Chaque membre contribue au projet à toutes ses étapes et selon ses compétences spécifiques. Dans les grandes lignes, les responsabilités ont été partagées comme suit (pour simplifier) :

- Ana : responsable de l’analyse des données, incluant l'exploration, le nettoyage et la préparation des datasets.
- Ariel et Lam : chargés de concevoir et de mettre en place différentes modélisations pour la compréhension et l’intégration des données. En charge également de l’organisation et de la mise au format du dépôt GitHub.
- Charles : apporte des idées en lien avec les actions déjà entamées dans le projet. Il est prévu qu'il développe l'application Streamlit avec Ariel.

Outils utilisés

- WhatsApp et Slack pour la communication quotidienne
- Google Meet et Zoom pour les réunions en visioconférence
- Google Docs et Google Slides pour la rédaction des rapports en mode collaboratif
- PowerBI : utilisé pour visualiser les unions des bases de données, permettant une meilleure compréhension des relations et des structures des données consolidées
- Python et Jupyter Notebooks : pour l’analyse des données, notamment pour l’exploration, le nettoyage et la manipulation des datasets
- Python, Jupyter Notebooks et VS Code : transition vers VS Code pour structurer et écrire du code en Python, tout en conservant l’utilisation des notebooks. Cette approche a permis une meilleure organisation et un suivi clair des avancées des modélisations étape par étape
- GitHub : utilisé pour gérer les versions du code, collaborer en équipe, documenter les différents éléments du projet et communiquer
- Streamlit : utilisé pour la présentation de nos résultats durant la soutenance (démo)

Principaux modules de formation utilisés

- Exploration et préparation des données
    - 101 - Python pour la Data Science : ce module a permis de maîtriser les bases de Python pour l'analyse des données, essentiel pour explorer les datasets et effectuer les premières manipulations.
    - 104 - Statistiques Exploratoires : a été utilisé pour analyser et comprendre les caractéristiques des données initiales, détecter les anomalies et évaluer leur distribution.
    - Méthodologie en Data Science : fournit le cadre nécessaire pour structurer les étapes du projet, depuis la collecte des données jusqu'à la modélisation et l'évaluation.

- Visualisation et compréhension des données
    - 111 - DataViz avec Matplotlib : permet de créer des visualisations claires et détaillées pour interpréter les tendances et relations dans les données.
    - 112 - DataViz avec Seaborn et 114 - Matplotlib - Compléments : ces modules ont renforcé les capacités à générer des visualisations avancées et interactives, facilitant l'analyse des corrélations et la sélection des variables importantes.

- Modélisation et systèmes de recommandation
    - Systèmes de recommandation : a fourni des bases théoriques et pratiques sur les algorithmes utilisés (collaborative filtering, content-based, hybrid systems) et leur mise en œuvre.
    - 125 - Méthodes de réduction de dimension : essentiel pour simplifier les données en réduisant leur complexité tout en conservant l'information pertinente, particulièrement utile pour optimiser les systèmes de recommandation.

- Développement et intégration
    - 117 - Streamlit : bien que prévu pour une utilisation future, ce module est destiné à faciliter la création d'applications interactives pour présenter les résultats du système de recommandation.

Expertise “Système de recommandation”

Ana

(sep24_bootcamp_ds)

Actuaire de formation depuis 2009, j'ai décidé d'entamer une reconversion professionnelle, guidée par ma passion croissante pour le domaine de la Data Science. En effet, c’est la manipulation des données et la prise de décisions éclairées par celles-ci qui m'ont toujours attirées dans l’actuariat. Ce domaine évolue désormais fortement vers l’intégration de la Data Science, rendant indispensables des compétences techniques que je souhaite renforcer.

Après plus de dix années d’expérience dans le secteur des assurances, je ressens aujourd’hui le besoin d’élargir mes perspectives professionnelles. La Data Science ouvre des horizons fascinants en permettant d’appliquer mes compétences à des secteurs variés, ce qui correspond pleinement à ma volonté de diversifier mon champ d’expertise et d’explorer de nouveaux défis.

Dans cette optique, un sujet qui me passionne particulièrement est l’analyse des comportements sociaux. Parmi les projets fil rouge proposés au cours de la formation, le développement d’un système de recommandation de films correspondait en partie à ce que je recherchais. Ce projet aborde l’analyse des comportements sociaux à travers l’étude des préférences et des interactions humaines. Il m’offre l’occasion d’explorer un domaine totalement distinct de celui des assurances tout en m’initiant à des dynamiques sociales.

Expertise initiale : 2/5

Ariel

(aug24_bootcamp_mle)

Professionnel de la gestion et de la construction intelligente :

Avec plus de 10 ans d’expérience dans la gestion de projets immobiliers et la mise en œuvre de solutions innovantes dans le secteur de la construction, j’ai décidé de réorienter ma carrière vers l’intégration des nouvelles technologies, guidé par ma passion croissante pour l’IoT et l’analyse des données. En tant qu’ingénieur civil de formation et expert en gestion immobilière, j’ai toujours été fasciné par l’optimisation des processus et la résolution de problématiques complexes à l’aide de solutions technologiques.

Dans mon rôle actuel de Smart Construction Solutions Manager, j’ai dirigé l’adoption de technologies IoT dans des projets de construction, réduisant les temps de 50 % grâce à des systèmes de maintenance prédictive et optimisant l’utilisation des ressources. Ces initiatives m’ont permis d’explorer des applications concrètes de la science des données et de l’ingénierie numérique, renforçant ma conviction que l’avenir réside dans l’interconnexion des disciplines.

Aujourd'hui, je souhaite approfondir mes compétences techniques en Data Science pour diversifier mes perspectives professionnelles et répondre aux défis d'un secteur en constante évolution. Une thématique qui me passionne particulièrement est l'analyse prédictive appliquée à la gestion des ressources et au suivi en temps réel des projets.

Expertise initiale : 3/5

Charles

(avr24_Bootcamp_DS)
https://www.linkedin.com/in/davansys/

Avec deux décennies d'expérience dans l'informatique, notamment dans la gestion de projets IT liés aux solutions e-Banking et à l'Intranet décisionnel, je suis diplômé en ingénierie des Applications Réparties de l'Université Paris 8.

Passionné par l'analyse de données, j'ai souhaité me spécialiser en data science. Cette formation me permet de concrétiser un projet d'innovation interne qui vise à proposer des produits personnalisés à nos clients, en exploitant leurs profils. Ce projet, lauréat d'un concours interne, est une opportunité unique de mettre en pratique mes compétences techniques et d'apporter une réelle valeur ajoutée à notre entreprise.

Expertise initiale : 3/5

Lam

(aug24_bootcamp_mle)
https://www.linkedin.com/in/lampaturle/

En reconversion professionnelle après 20 ans dans la production et l'édition de jeux vidéo et applications mobile. Habitué à intégrer la data dans l'analyse et la prise de décision en tant que Product Manager et à travailler avec des data scientists et data analysts au quotidien, mais peu de compétences techniques (code, visualisation, modélisation, etc.).

Grand consommateur de films (et de séries) et utilisateur de différentes plateformes s'appuyant sur des systèmes de recommandation (e.g. Le Monde, YouTube et Netflix, Amazon et autres sites d'e-commerce, Spotify, Facebook et Instagram, etc.). Avant de choisir ce projet de groupe dans le cadre de la formation Datascientest, j'avais envisagé de créer un système de recommandation de séries coréennes comme projet personnel.

Expertise initiale : 2/5

Compréhension et manipulation des données

Cadre

Nous avons travaillé avec deux jeux de données distincts, disponibles librement et suggérés par Datascientest. Avoir deux jeux de données, chacun séparés en plusieurs fichiers, nous a permis d'avoir une base de données unifiée riche, mais nous a posé de nombreuses difficultés : fusion des jeux de données, décisions à prendre sur les variables à garder ou non, etc.

Pour plus de détails, le rapport d'exploration des données est disponible ici au format .xlsx : https://docs.google.com/spreadsheets/d/1vgNMtYaLKitcrDfo4GNbjIM65D6oeO10/edit?usp=sharing&ouid=105672209662968344116&rtpof=true&sd=true

Dataset "MovieLens 20M"

- Plus de 138 000 utilisateurs uniques
- Plus 27 000 films uniques
- Plus de 20 millions de ratings
- Plus de 465 000 tags (génome)
- Près de 12 millions de scores de pertinence pour plus de 1100 tags (génome)
- 6 fichiers : genome-scores.csv, genome-tags.csv, links.csv, movies.csv, ratings.csv et tags.csv
- C'est le jeu de données principal dans le cadre de notre projet "système de recommandation" puisque c'est celui qui contient les 'userId' et les ratings par utilisateur
- Mis à disposition par GroupLens, un laboratoire de recherche appartenant à l'Université du Minnesota

Dataset "IMDb Non-Commercial"

- Plus de 5 millions de titres uniques
- Près de 1.5 millions de ratings (moyenne de notes)
- Nombre de votes (avec notamment un titre ayant reçu près de 3 millions de votes)
- Plus de 18 millions d'entrées sur les crews (réalisateurs, scénaristes, etc.)
- Près de 14 millions d'entrées sur les casts (acteurs, etc.)
- 7 fichiers : name.basics.tsv, title.akas.tsv, title.basics.tsv, title.crew.tsv, title.episode.tsv, title.principals.tsv, title.ratings.tsv
- Ce dataset est beaucoup plus large que le dataset MovieLens non seulement parce qu'il ne se limite pas aux films, mais aussi parce qu'il couvre une période beaucoup plus large
- IMDb est également beaucoup plus riche puisqu'il contient de nombreuses variables supplémentaires notamment sur l'équipe, le cast, etc.
- En revanche, ce jeu de données ne contient aucune information sur les utilisateurs
- Mis à disposition par IMDb, une entreprise appartenant au groupe Amazon

Pertinence

Dans le cadre de notre projet Data Scientist et suite à la documentation présentée lors de notre apprentissage, nous avons sélectionné deux types de systèmes de recommandation avec lesquels expérimenter en priorité :

1. Filtrage collaboratif

Principe :
Ce système s'appuie sur les préférences d'utilisateurs similaires pour recommander des éléments. Si des utilisateurs ont aimé des éléments similaires par le passé, il est probable qu'ils apprécient les mêmes éléments à l'avenir.

Variables explicatives :

- Matrice utilisateur-élément : Représente les interactions entre les utilisateurs et les éléments (notes, achats, vues, etc.).
- Similarité entre utilisateurs : Mesure la proximité des goûts entre utilisateurs (ex: coefficient de corrélation de Pearson, similarité cosinus).
- Voisinage : Ensemble d'utilisateurs similaires utilisés pour prédire les préférences d'un utilisateur cible.

2. Filtrage basé sur le contenu

Principe :
Ce système analyse les caractéristiques des éléments pour recommander des éléments similaires à ceux qu'un utilisateur a appréciés par le passé.

Variables explicatives :

- Profil utilisateur : représente les préférences de l'utilisateur en fonction des caractéristiques des éléments qu'il a aimés.
- Description des éléments : ensemble des caractéristiques des éléments (ex: genre d'un film, auteur d'un livre, mots-clés d'un article).
- Similarité entre éléments : mesure la ressemblance entre les éléments en fonction de leurs caractéristiques (ex: distance cosinus, TF-IDF).

Distinction des variables explicatives :
Le filtrage collaboratif se concentre sur les “interactions entre utilisateurs”, tandis que le filtrage par contenu se concentre sur les “caractéristiques des éléments”. Le premier utilise des données de comportement utilisateur, le second des données descriptives des éléments.

Cible(s)

Pour le filtrage collaboratif, notre cible initiale est très claire :

- ‘rating’ (MovieLens) = mesure de la “popularité par utilisateur”

Ensuite nous avons évolué vers une autre cible :

- ‘score_pertinence’ (New), un score d'agrégation plus pertinent que nous avons appelé score de pertinence et qui tient compte aussi du rating et du nombre de votes sur IMDb

Si nous en avons le temps, pour pallier au problème de démarrage à froid (“cold start”), nous pourrions aussi envisager un troisième système de recommandation qui chercherait à proposer aux utilisateurs pour lesquels nous n’avons aucun profil (nouveaux utilisateurs sans historique) des films en s’appuyant sur leur popularité générale. Dans ce système, les cibles pourraient être :

- ‘numVotes’ (IMDb) et/ou ‘averageRating’ (IMDb) = des mesures de la “popularité générale” d’un film (par opposition à la popularité d’un film pour un utilisateur en particulier)

Variables prioritaires

Nous avons identifié les variables prioritaires suivantes afin de créer la matrice de notations à la fois pour le filtrage collaboratif “user-based” et “item-based”:

- ‘userId’ (MovieLens)
- ‘imdbId’ (MovieLens)
- ‘rating’ (MovieLens)

Autres variables pertinentes

Pour le filtrage par contenu, si le temps le permet, nous envisageons d’utiliser les variables suivantes :

- timestamp (MovieLens) = utilisé pour une analyse graphique (biais)
- ‘super_genre’ (New) = 20 nouvelles colonnes créées à partir de ‘genres’ (IMDb)
- ‘category_Profession’ (New) = 9 nouvelles colonnes pour le cast créées à partir de ‘primaryProfession’ (IMDb)
- ‘region’ (IMDb)
- ‘language’ (IMDb)
- ‘isAdult’ (IMDb)
- ‘runtimeMinutes’ (IMDb)
- ‘directors’ (IMDb)
- ‘writers’ (IMDb)
- ‘jobs’ (IMDb)
- ‘characters’ (IMDb)
- ‘primaryName’ (IMDb)
- ‘birthYear’ (IMDb)

Nous avons encore des doutes sur la meilleure manière d'utiliser ces variables dans le cadre d'un système de filtrage par contenu, mais la première piste que nous voulons suivre est de concaténer toutes ces variables dans la même colonne afin de pouvoir l'utiliser ensuite dans un processus de tokenisation et de vectorisation (TfidfVectorizer).
Sinon, nous pourrions aussi effectuer une étude de significativité par variable.

Variables de fusion

Lors de notre exploration, nous avons identifié des variables utiles pour la fusion des fichiers en dataset et des datasets en database, mais jugées inutiles pour l'analyse et la modélisation. En voici la liste :

- 'movieId' (MovieLens)
- 'titleId' (IMDb)
- 'tconst' (IMDb)
- 'nconst' (IMDb)

Variables redondantes

Les variables suivantes n'ont pas été retenues, en effet, nous utilisons 'imdbId' (MovieLens) comme identifiant unique pour les films en combinaison avec 'title_name' (New), donc les autres variables se rapportant au titre des films ne sont pas utiles :

- 'title' (MovieLens)
- 'title' (IMDb)
- 'originalTitle' (IMDb)
- 'primaryTitle' (IMDb)

'startYear' (IMDb) = nous utilisons release_year (New)

De même, nous pensons utiliser les 'super_genres' (NEW) créés à partir de 'genres' (IMDb) donc les autres variables se rapportant au genre des films ne sont pas utiles :

- 'genres' (MovieLens) = séparé en 20 genres et dichotomisé (un film pouvait avoir plusieurs genre dans la même colonne)
- 'genres' (IMDb)

Variables ignorées

Nous n'avons pas récupéré de données autres que celles de MovieLens et IMDb, par conséquent, nous n'avons pas établi de lien avec TMDB (i.e. The Movie Database). La variable suivante n'est donc pas utilisée :

- 'tmdbId' (MovieLens)

Les variables qui concernaient les séries (et non les films) ont également été éliminées :

- 'parentTconst' (IMDb)

- ‘seasonNumber’ (IMDb)
- ‘episodeNumber’ (IMDb)
- ‘endYear’ (IMDb)

Enfin d’autres variables nous ont parues inutiles ou pouvaient être liées à d’autres variables non-utilisées :

- ‘types’ (IMDb)
- ‘ordering’ (IMDb)
- ‘attributes’ (IMDb)
- ‘isOriginalTitle’ (IMDb) = lié à une variable redondante car on utilise ‘primaryTitle’ (IMDb)
- ‘titleType’ (IMDb)
- ‘deathYear’ (IMDb)
- ‘category’ (New) = nouvelles variables créées à partir de ‘primaryProfession’ (IMDb)

Variables à reconsidérer plus tard

Nous avons éliminé les variables ci-dessous dans un premier temps car elles ne sont pas utiles pour le filtrage collaboratif. Elles sont également difficiles à traiter et nous ne sommes pas certains de leur corrélation avec la variable cible.

Toutefois, nous nous réservons le choix de changer d’avis et pourrons les ré-intégrer, notamment pour le filtrage par contenu :

- ‘primaryProfession.1’ (IMDb), ‘primaryProfession.2’ (IMDb), ‘primaryProfession.3’ (IMDb)
- ‘knownForTitles.1’ (IMDb), ‘knownForTitles.2’ (IMDb), ‘knownForTitles.3’ (IMDb), ‘knownForTitles.4’ (IMDb)

Pour l’instant, nous avons décidé de ne pas utiliser le fichier génome qui nous a semblé peu exploitable et non-pertinent en tous cas pour notre première étape de modélisation :

- ‘tag’ (MovieLens)
- ‘tagId’ (MovieLens)
- ‘relevance’ (MovieLens)

Particularités

Fichiers .tsv : nous n’avions jamais rencontré de fichier utilisant ce format avant, il a donc fallu nous renseigner pour savoir comment les utiliser et comprendre pourquoi ils sont utilisés :

- Moins de conflits avec les séparateurs
- Lisibilité dans les éditeurs de texte
- Compatibilité avec des outils spécialisés
- Moins de besoin d'échappement de caractères
- Meilleure gestion des données multi-lignes
- Etc.

Limitations

- La taille énorme des jeux de données de départ rend le travail d'exploration et de préparation très fastidieux. Nous avons vraiment lutté pour obtenir une base de données unifiée lisible.
- La taille de la matrice semble ingérable (138 000 utilisateurs et 20 millions de ratings) donc nous allons nous efforcer de réduire ses dimensions pour pouvoir faire nos calculs.
- La mémoire de nos machines ne nous permet pas de travailler avec une base de données si vaste: nous sommes parfois obligés de trouver des moyens de contournement (e.g. Dask).
- Période de temps très précise et étroite : notre dataset MovieLens a été créé entre janvier 1995 et mars 2015 et mis à jour en octobre 2016, il ne contient donc pas de films sortis après 2016.
- Seulement deux plateformes sont utilisées pour notre projet (MovieLens principalement et IMDb dans une moindre mesure). On pourrait légitimement se poser la question suivante: MovieLens est-elle la plateforme la plus pertinente (vs Netflix, RottenTomatoes, TMDB, etc.) ?
- MovieLens est spécialisé dans les films, par conséquent, on perd un pan entier de l'offre de contenu vidéo de longue durée, et notamment les séries
- Biais lié à la date (volume de ratings, ratings moyens) : nous avons voulu l'illustrer dans la partie visualisation

Pre-processing et Feature Engineering

Nettoyage

- La première étape a été la consolidation des fichiers MovieLens en une base unique (merge)
    - Clé de fusion : ‘movieId’ (MovieLens)
- L’ajout des informations contenues dans le jeu IMDb i.e enrichissement de la base de données MovieLens par les données IMDb (merge) est venu dans un second temps :
    - Clés de fusion :
        - ‘imdbId’ (MovieLens) et ‘tconst’ (IMDb)
        - ‘tconst’ (IMDb) pour consolider les différents fichiers IMDb
- Certaines variables ont été retravaillées ou ont vu certaines de leurs entrées corrigées. Voici deux exemples frappants :
    - ‘isAdult’ (IMDb) = la variable “isAdult” était corrompue (valeurs aberrantes probablement dues à une mauvaise importation dans le dataset originel). Nous avons choisi de supprimer les 630 lignes concernées dans title.basics.tsv.
    - Des valeurs aberrantes dans ‘birthYear’ (IMDb) et ‘deathYear’ (IMDb) avec certaines dates avant JC, ce qui pouvait donner des âges négatifs dans ‘age’ (New)

[52]:

	tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes	genres
1096792	tt10233364	tvEpisode	Rolling in the Deep Dish\tRolling in the Deep ...	0	2019.0	NaN	NaN	Reality-TV	NaN
1506347	tt10970874	tvEpisode	Die Bauhaus-Stadt Tel Aviv - Vorbild für die M...	0	2019.0	NaN	NaN	Talk-Show	NaN
1893588	tt11670006	tvEpisode	...ein angenehmer Unbequemer...\t...ein angene...	0	1981.0	NaN	NaN	Documentary	NaN
2004398	tt11868642	tvEpisode	GGN Heavyweight Championship Lungs With Mike T...	0	2020.0	NaN	NaN	Talk-Show	NaN
2157754	tt12149332	tvEpisode	Jeopardy! College Championship Semifinal Game ...	0	2020.0	NaN	NaN	Game-Show	NaN
...	...	...	...	...	...	...	...	...	...
7992665	tt33058849	tvEpisode	The Deadly Bees\tThe Deadly Bees	0	1986.0	NaN	NaN	Fantasy,Horror,Mystery	NaN
8116462	tt33376462	tvEpisode	A Bruxa tá Solta\tA Bruxa tá Solta	0	2016.0	NaN	NaN	Animation,Comedy,Family	NaN
8464006	tt3984412	tvEpisode	I'm Not Going to Come Last, I'm Just Going to ...	0	2014.0	NaN	NaN	Game-Show,Reality-TV	NaN
11093483	tt9822816	tvEpisode	Zwischen Vertuschung und Aufklärung - Missbrau...	0	2019.0	NaN	NaN	Talk-Show	NaN
11133239	tt9909210	tvEpisode	Politik und/oder Moral - Wie weit geht das Ver...	0	2005.0	NaN	NaN	Talk-Show	NaN

630 rows × 9 columns

    - ‘birthYear’ (IMDb) = Correction de l’entrée pour Rosita Royce qui contenait une valeur aberrante
- Variables ré-encodées:
    - ‘timestamp’ (MovieLens) = changement au format datetime
- Création de nouvelles variables :

  Les variables suivantes ont été créées lors de l’exploration, mais nous n’allons finalement pas les utiliser pour la modélisation (en tous cas dans un premier temps):
    - ‘title_name (New) et ‘release_year’ (New) créées à partir de ‘title’ (MovieLens)
    - ‘timestamp_readable’ (New), ‘date_rating’ (New), ‘time_rating’ (New), ‘year’ (New), ‘month’ (New) à partir de ‘timestamp’ (MovieLens) dans ratings.csv
    - ‘timestamp_readable’ (New), ‘date_tag’ (New), ‘time_tag’ (New), ‘year_tag’ (New), ‘month_tag’ (New) à partir de ‘timestamp’ (MovieLens) dans tags.csv

- ‘age’ (New) à partir de birthYear dans name.basics.tsv
- ‘category’ (New) : 9 professions séparées créées à partir de ‘primaryProfession’ (IMDb)
- ‘super_genres’ (New) : 20 genres séparés créés à partir de ‘genres’ (IMDb)

Union des bases de données

1. Création du dataset pour la **matrice de notations**

Tout d’abord, cette étape s’est concentrée sur la préparation de la base de données nécessaire pour le filtrage collaboratif. Les variables clés identifiées pour atteindre cet objectif étaient :

- movieId
- userId
- rating

Après analyse, nous avons constaté que la base de données **MovieLens** identifie clairement les évaluations (**rating**) en les associant aux utilisateurs (**userId**) et aux films (**movieId**). Par conséquent, nous avons décidé d’utiliser cette base de données comme principale source pour construire la **matrice des notations.**

En parallèle, nous avons également exploré les bases de données IMDb. Bien que celles-ci fournissent des évaluations sous forme de moyennes pour chaque film, elles incluent également le nombre d’utilisateurs ayant contribué à ces moyennes.

Afin d’enrichir notre projet, nous avons intégré ces moyennes par film issues de **IMDb** à la base de données principale. Cette étape nous permettra d’explorer si ces informations supplémentaires peuvent être exploitées de manière pertinente dans le cadre de notre analyse et du projet global.

Pour cela, les bases de données unifiées ont été :

- MovieLens (unifiées par la variable ‘movieId’)
  - Movies.csv
  - Links.csv
  - Ratings.csv
- IMDb (unifiée par la variable ‘imdbId’ des datasets MovieLens et ‘tconst’ du dataset IMDb) :
  - title.basics.tsv

Visualisation du dataset :

[65]:

	userId	imdbId	rating	title_name	imdb_averageRating	imdb_numVotes
960	11	114709	4.5	Toy Story	8.3	1088953
961	11	113189	2.5	GoldenEye	7.2	273041
962	11	112281	3.5	Ace Ventura: When Nature Calls	6.4	237008
963	11	114746	5.0	Twelve Monkeys (a.k.a. 12 Monkeys)	8.0	654005
964	11	112697	4.5	Clueless	6.9	253424

Dataset utilisé pour :

- Filtrage collaboratif basé sur les utilisateurs (User-Based Collaborative Filtering)
- Filtrage collaboratif basé sur les éléments (Item-Based Collaborative Filtering)
- Filtrage collaboratif (approche modèle)

Nous avons pris en compte l'élimination de plusieurs variables du dataset afin de le rendre pertinent pour les modèles mentionnés ci-dessus.

2. Création du dataset pour utiliser le filtrage basé sur le contenu (Content-Based Filtering)

Pour cela nous avons décidé de récupérer des informations des bases de données IMDb pour les ajouter à la base de données de MovieLens.

Voici la visualisation du dataset précédent à la réduction des variables reprenant le dataset précédent mais avec toutes les informations :

[5]:

	userId	movieId	rating	date_rating	time_rating	year_rating	month_rating	genres	title_name	release_year	...	imdb_isAdult	imdb_sta
0	1	2	3.5	2005-04-02	23:53:47	2005	4	Adventure|Children|Fantasy	Jumanji	1995	...	0	
1	1	29	3.5	2005-04-02	23:31:16	2005	4	Adventure|Drama|Fantasy|Mystery|Sci-Fi	City of Lost Children, The (Cité des enfants p...	1995	...	0	
2	1	32	3.5	2005-04-02	23:33:39	2005	4	Mystery|Sci-Fi|Thriller	Twelve Monkeys (a.k.a. 12 Monkeys)	1995	...	0	
3	1	47	3.5	2005-04-02	23:32:07	2005	4	Mystery|Thriller	Seven (a.k.a. Se7en)	1995	...	0	
4	1	50	3.5	2005-04-02	23:29:40	2005	4	Crime|Mystery|Thriller	Usual Suspects, The	1995	...	0	

5 rows × 26 columns

[11]:
```
MovieLens_Imdb.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000263 entries, 0 to 20000262
Data columns (total 26 columns):
 #   Column               Dtype
---  ------               -----
 0   userId               int64
 1   movieId              int64
 2   rating               float64
 3   date_rating          datetime64[ns]
 4   time_rating          string
 5   year_rating          int32
 6   month_rating         int32
 7   genres               string
 8   title_name           string
 9   release_year         Int64
 10  imdbId               int64
 11  tmdbId               float64
 12  imdb_tconst          string
 13  imdb_titleType       string
 14  imdb_primaryTitle    string
 15  imdb_originalTitle   string
 16  imdb_isAdult         Int64
 17  imdb_startYear       Int64
 18  imdb_endYear         Int64
 19  imdb_runtimeMinutes  float64
 20  imdb_genres          string
 21  imdb_tconst_numeric  float64
 22  imdb_directors       string
 23  imdb_writers         string
 24  imdb_averageRating   float64
 25  imdb_numVotes        Int64
dtypes: Int64(5), datetime64[ns](1), float64(5), int32(2), int64(3), string(10)
memory usage: 3.8 GB
```

Nous souhaitions récupérer le nom des réalisateurs ainsi que les films associés à ces réalisateurs.

Cependant, la variable disponible dans le dataset indique uniquement un code correspondant aux noms des réalisateurs :

ıtes	imdb_genres	imdb_tconst_numeric	imdb_directors	imdb_writers	imdb_averageRating	imdb_numVotes
)4.0	Adventure,Comedy,Family	113497.0	nm0002653	nm0378144,nm0852430,nm0833164,nm0885575	7.1	384991
12.0	Adventure,Drama,Fantasy	112682.0	nm0001988,nm0000466	nm0012496,nm0000466,nm0001988,nm0491011	7.5	72399
29.0	Mystery,Sci-Fi,Thriller	114746.0	nm0000416	nm0003408,nm0672459,nm0672466	8.0	654005
27.0	Crime,Drama,Mystery	114369.0	nm0000399	nm0001825	8.6	1839918
)6.0	Crime,Drama,Mystery	114814.0	nm0001741	nm0003160	8.5	1161903

Pour cette raison, on recherche :

- Les noms de réalisateurs
- Les films liés aux directeurs de la base de données Name_basics

Et également, pour les inclure :

- Les noms des acteurs
- Les films liés aux acteurs du film

Pour finalement obtenir un dataset avec les variables suivantes :

[3]:

	userId	movieId	rating	genres	title_name	release_year	imdbId	imdb_averageRating	imdb_numVotes	directors_primaryName	director
0	1	2	3.5	Adventure|Children|Fantasy	Jumanji	1995	113497	7.1	384991	Joe Johnston	Captai
1	1	29	3.5	Adventure|Drama|Fantasy|Mystery|Sci-Fi	City of Lost Children, The (Cité des enfants p...	1995	112682	7.5	72399	Marc Caro,Jean-Pierre Jeunet	
2	1	32	3.5	Mystery|Sci-Fi|Thriller	Twelve Monkeys (a.k.a. 12 Monkeys)	1995	114746	8.0	654005	Terry Gilliam	E
3	1	47	3.5	Mystery|Thriller	Seven (a.k.a. Se7en)	1995	114369	8.6	1839918	David Fincher	S
4	1	50	3.5	Crime|Mystery|Thriller	Usual Suspects, The	1995	114814	8.5	1161903	Bryan Singer	X-M

[7]:

```
print(Union.columns)
```

```
Index(['userId', 'movieId', 'rating', 'genres', 'title_name', 'release_year',
       'imdbId', 'imdb_averageRating', 'imdb_numVotes',
       'directors_primaryName', 'directors_knownForTitles_primaryTitle',
       'actors_characters', 'actors_primaryName',
       'actors_knownForTitles_primaryTitle'],
      dtype='object')
```

Transformations

- Dichotomisation pour les genres et pour les professions (‘category’ (New)): nous ne sommes pas encore sûrs de pouvoir utiliser ces variables
- Normalisation / Standardisation (e.g. MinMaxScaler, RobustScaler, StandardScaler): pas nécessaire pour le filtrage collaboratif qui sera notre première étape, nous pourrons y revenir plus tard selon les modélisations envisagées

Réduction de dimension

Voici les pistes que nous explorons pour réduire la taille de notre matrice de notations :

Sur les films :

- Sélection de films en fonction du nombre de notes reçues (élimination des extrêmes) : ne sélectionner que des films avec un niveau de popularité générale minimum (> Xk votes)
- Sélection de films en fonction de la moyenne de notes reçues (élimination des extrêmes) : ne sélectionner que des films avec un niveau de popularité générale minimum (>7?, >8?)

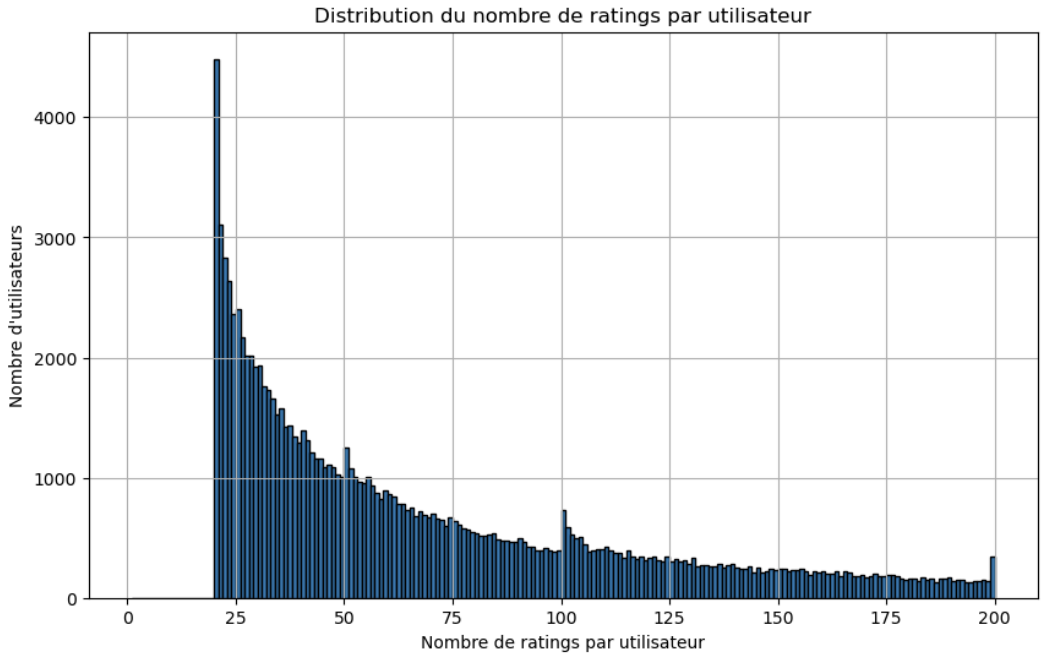
Sur les users :

- Sélection de users en fonction du nombre de notes données (élimination des extrêmes)
- Sélection de users en fonction de la moyenne de notes données (élimination des extrêmes): il semble que la plupart sont déjà éliminés automatiquement lorsqu’on élimine les users en fonction du nombre de notes

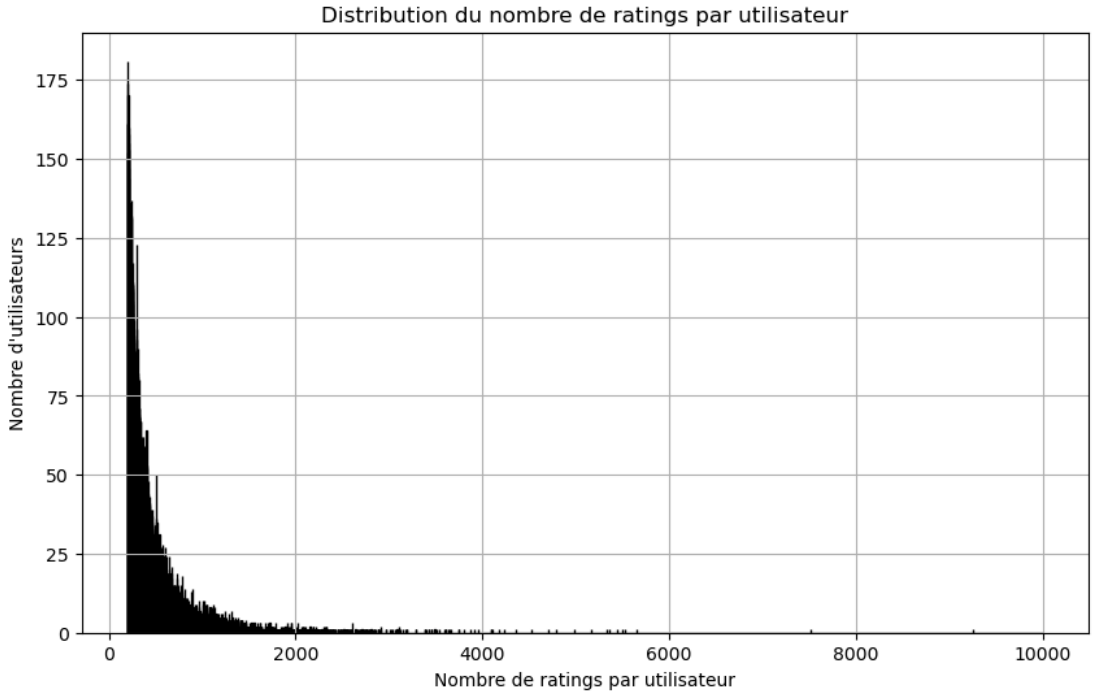
Nous ajusterons ces critères en fonction de la taille de matrice qui nous permettra de faire des calculs sans qu’ils soient trop lourds pour nos machines.

Quelques visualisations pour nous aider à prendre les bonnes décisions :

Distribution du nombre de notes par utilisateur (histogramme) (1/2) :



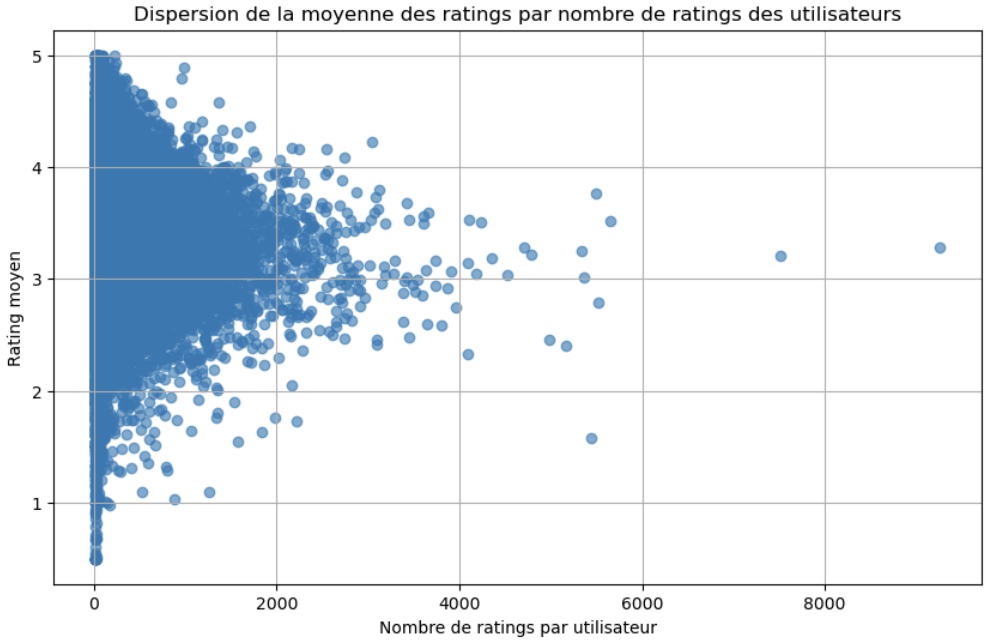
Distribution du nombre de notes par utilisateur (histogramme) (2/2) i.e. continuation du graphique :



La concentration des ratings se trouve majoritairement chez les utilisateurs ayant donné un nombre de ratings inférieur à 200. Cependant, ces utilisateurs peuvent également introduire du bruit dans les données. Il serait donc utile de filtrer ou de pondérer leurs contributions afin d'améliorer la robustesse du modèle de filtrage collaboratif.

Nous pensons qu'il serait judicieux d'éliminer ces informations (les ratings des utilisateurs très peu actifs), ce qui pourrait réduire significativement la taille de la base de données. Cela permettrait de mieux gérer la mémoire sur nos ordinateurs, rendant les opérations de traitement des données plus rapides et plus fluides.

Nuage de points :



Ce graphique nous montre que l'élimination des ratings des utilisateurs très peu actifs permet de maintenir la moyenne des notes par utilisateur, sans perdre de données essentielles pouvant radicalement influencer le modèle. De même, on peut éliminer les utilisateurs trop actifs.

Tableau récapitulatif des réductions de dimensions possibles (en cours d'évaluation, pour nous donner des ordres de grandeur) :

Obs	1	2	3	4
Min number ratings	500	1,000	...	...
Max number ratings	700	2,000		

Matrice originale serait de :

Userid (matrix rows)	138,493	138,493		
Films (matrix columns)	26,744	26,744		

Matrice envisagée

Userid (matrix rows)	3,502	1,639		
Films (matrix columns)	16,346	21,080		

% d'information possible à garder

Userids %	2.53%	1.18%		
Films %	61.12%	78.82%		

Pour affiner les recommandations, nous avons décidé de ne conserver que les utilisateurs ayant donné entre 500 et 700 évaluations dans la base de données MovieLens.

Ce filtrage permet d'éliminer les utilisateurs ayant très peu de notations ainsi que ceux en ayant donné un nombre excessif, chacun présentant des effets indésirables sur les performances et la fiabilité du modèle de filtrage collaboratif. Voici les raisons principales :

Utilisateurs ayant très peu de notes (pas assez actifs)

Lorsqu'un utilisateur a donné très peu de notes (par exemple, 30 ou 50), le modèle dispose d'informations insuffisantes pour bien comprendre ses préférences. Cela entraîne des recommandations basées sur des corrélations faibles ou non représentatives, risquant ainsi de produire des résultats peu pertinents. Par ailleurs, en filtrage collaboratif, les modèles basés sur la similarité comparent les utilisateurs entre eux en fonction de leurs notes. Les utilisateurs ayant peu de notes n'ont pas assez de points communs avec les autres pour calculer des similarités fiables, ce qui fausse les résultats.

Utilisateurs ayant trop de notes (trop actifs)

À l'inverse, les utilisateurs ayant évalué un très grand nombre d'éléments affichent souvent des goûts trop génériques, ce qui peut biaiser les similarités, car ils partagent des items avec presque tous les autres utilisateurs. Cela réduit leur utilité pour le modèle, qui cherche des préférences plus spécifiques et distinctives.

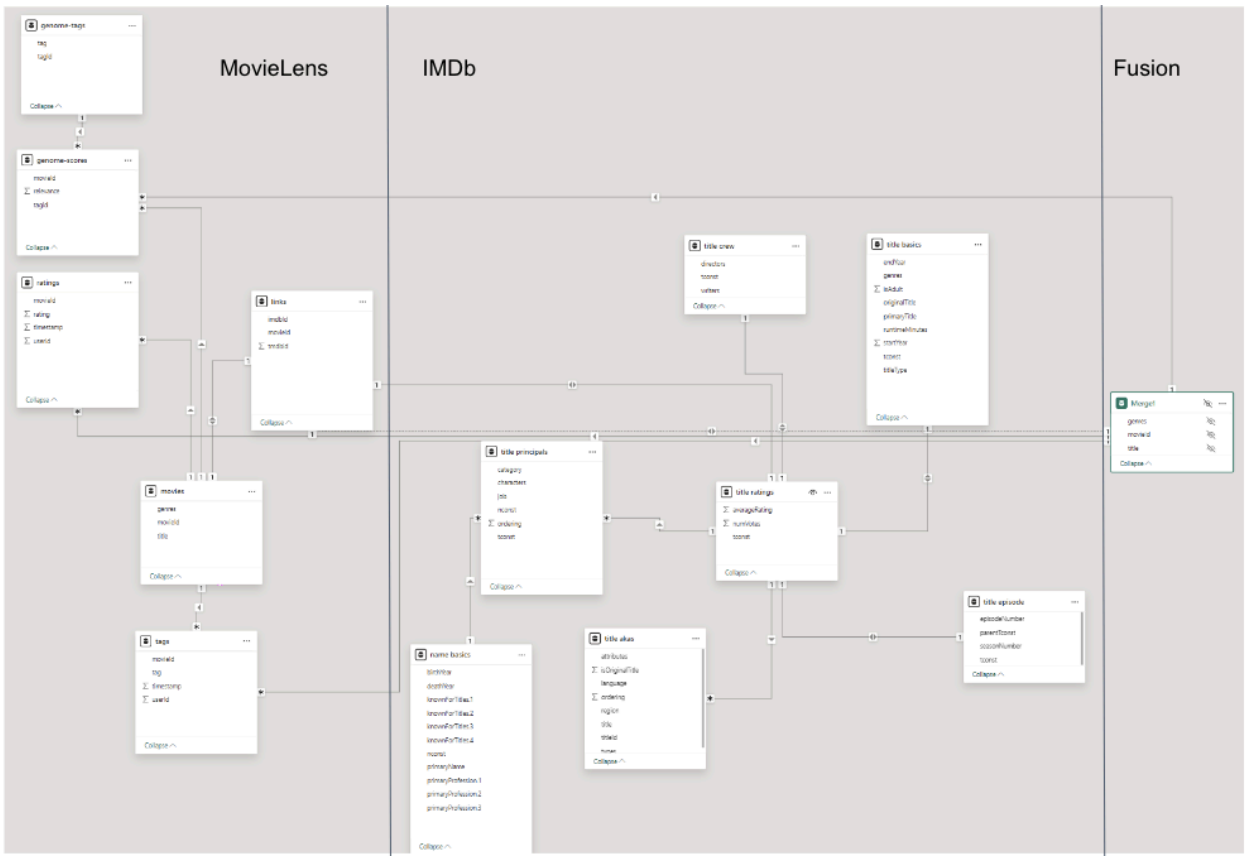
Optimisation des performances et réduction des biais

Filtrer les utilisateurs ayant donné trop ou trop peu de notes réduit la quantité de données traitée, augmentant ainsi l'efficacité des calculs et minimisant le bruit dans les données. Ce nettoyage permet au modèle de se concentrer sur les utilisateurs ayant un profil de notation plus représentatif, réduisant ainsi le risque de résultats incohérents. Par exemple, un utilisateur ayant noté seulement un ou deux items pourrait recevoir des recommandations trop spécifiques (parfois incorrectes), tandis qu'un utilisateur ayant noté un nombre excessif d'items pourrait se voir proposer des recommandations trop génériques.

Visualisations et Statistiques

Les graphiques que vous allez voir dans les pages suivantes sont des modèles de visualisations créés pour illustrer la démarche analytique à suivre. Ces exemples montrent comment exploiter les données pour concevoir un outil de recommandation performant. Nous allons parcourir chaque graphique et comprendre en quoi ils nous apportent des informations cruciales.

Map (Power BI)



I

Ceci est une visualisation des variables et de leurs liens par fichier et par jeu de données.

Base de données proposée pour le modèle de filtrage collaboratif

Nous avons besoin des variables suivantes :

- userId
- films (donc nous utiliserons les ‘imdbId’ afin d'éviter des problèmes liés au texte)
- ratings

Vous pouvez voir également que nous avons intégré les ratings (moyenne) et le nombre de votes de la base de données IMDb.

[11]:
```
# Lire base de données
FC1_movieLens_Imdb = pd.read_feather(FC1_movieLens_Imdb_path)
FC1_movieLens_Imdb.info()
FC1_movieLens_Imdb.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000263 entries, 0 to 20000262
Data columns (total 6 columns):
 #   Column              Dtype
---  ------              -----
 0   userId              int64
 1   imdbId              int64
 2   rating              float64
 3   title_name          string
 4   imdb_averageRating  float64
 5   imdb_numVotes       Int64
dtypes: Int64(1), float64(2), int64(2), string(1)
memory usage: 934.6 MB
```

[11]:

	userId	imdbId	rating	title_name	imdb_averageRating	imdb_numVotes
0	1	113497	3.5	Jumanji	7.1	384991
1	1	112682	3.5	City of Lost Children, The (Cité des enfants p...	7.5	72399
2	1	114746	3.5	Twelve Monkeys (a.k.a. 12 Monkeys)	8.0	654005
3	1	114369	3.5	Seven (a.k.a. Se7en)	8.6	1839918
4	1	114814	3.5	Usual Suspects, The	8.5	1161903

Nous préparons la base de données pour l'application du modèle de filtrage basé sur le contenu.
Voici une visualisation des variables en cours de traitement.
Nous avons également des idées pour dichotomiser certaines variables et éliminer des colonnes de la base de données, comme nous l'avons appris pendant le cours. Cependant, ces dichotomisations doivent être appliquées après la séparation de la base de données entre les jeux de test et d'entraînement.

[9]:
```
movieLens_Imdb.head()
```

[9]:

	userId	movieId	rating	date_rating	time_rating	year_rating	month_rating	genres	title_name	release_year	...	imdb_isAdult	imdb_st
0	1	2	3.5	2005-04-02	23:53:47	2005	4	Adventure|Children|Fantasy	Jumanji	1995	...	0	
1	1	29	3.5	2005-04-02	23:31:16	2005	4	Adventure|Drama|Fantasy|Mystery|Sci-Fi	City of Lost Children, The (Cité des enfants p...	1995	...	0	
2	1	32	3.5	2005-04-02	23:33:39	2005	4	Mystery|Sci-Fi|Thriller	Twelve Monkeys (a.k.a. 12 Monkeys)	1995	...	0	
3	1	47	3.5	2005-04-02	23:32:07	2005	4	Mystery|Thriller	Seven (a.k.a. Se7en)	1995	...	0	
4	1	50	3.5	2005-04-02	23:29:40	2005	4	Crime|Mystery|Thriller	Usual Suspects, The	1995	...	0	

5 rows × 26 columns

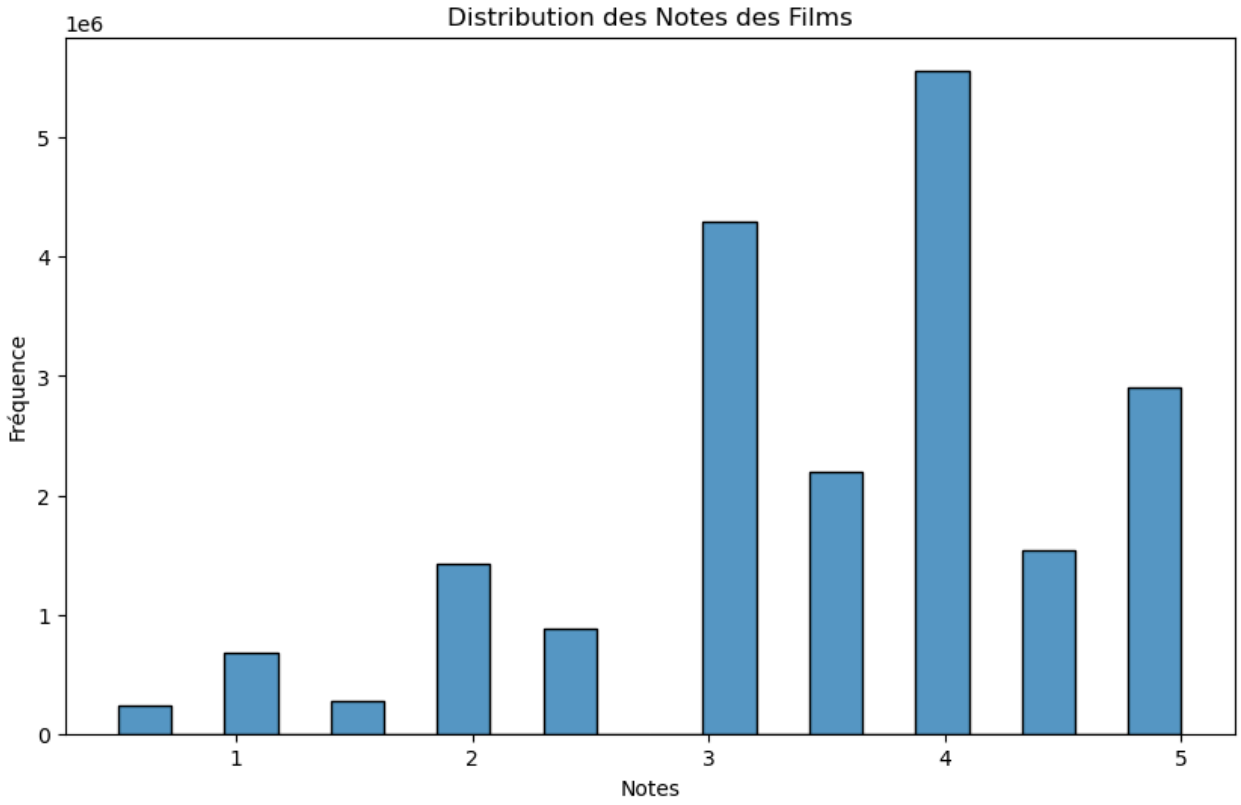
[23]:
```
movieLens_Imdb.compute().info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000263 entries, 0 to 20000262
Data columns (total 26 columns):
 #   Column               Dtype
---  ------               -----
 0   userId               int64
 1   movieId              int64
 2   rating               float64
 3   date_rating          datetime64[ns]
 4   time_rating          string
 5   year_rating          int32
 6   month_rating         int32
 7   genres               string
 8   title_name           string
 9   release_year         Int64
 10  imdbId               int64
 11  tmdbId               float64
 12  imdb_tconst          string
 13  imdb_titleType       string
 14  imdb_primaryTitle    string
 15  imdb_originalTitle   string
 16  imdb_isAdult         Int64
 17  imdb_startYear       Int64
 18  imdb_endYear         Int64
 19  imdb_runtimeMinutes  float64
 20  imdb_genres          string
 21  imdb_tconst_numeric  float64
 22  imdb_directors       string
 23  imdb_writers         string
 24  imdb_averageRating   float64
 25  imdb_numVotes        Int64
dtypes: Int64(5), datetime64[ns](1), float64(5), int32(2), int64(3), string(10)
memory usage: 6.5 GB
```

Visualisations réalisées

Ces graphiques ont été créés pour mieux comprendre notre jeu de données et mieux intégrer les variables dans les modélisations prévues :

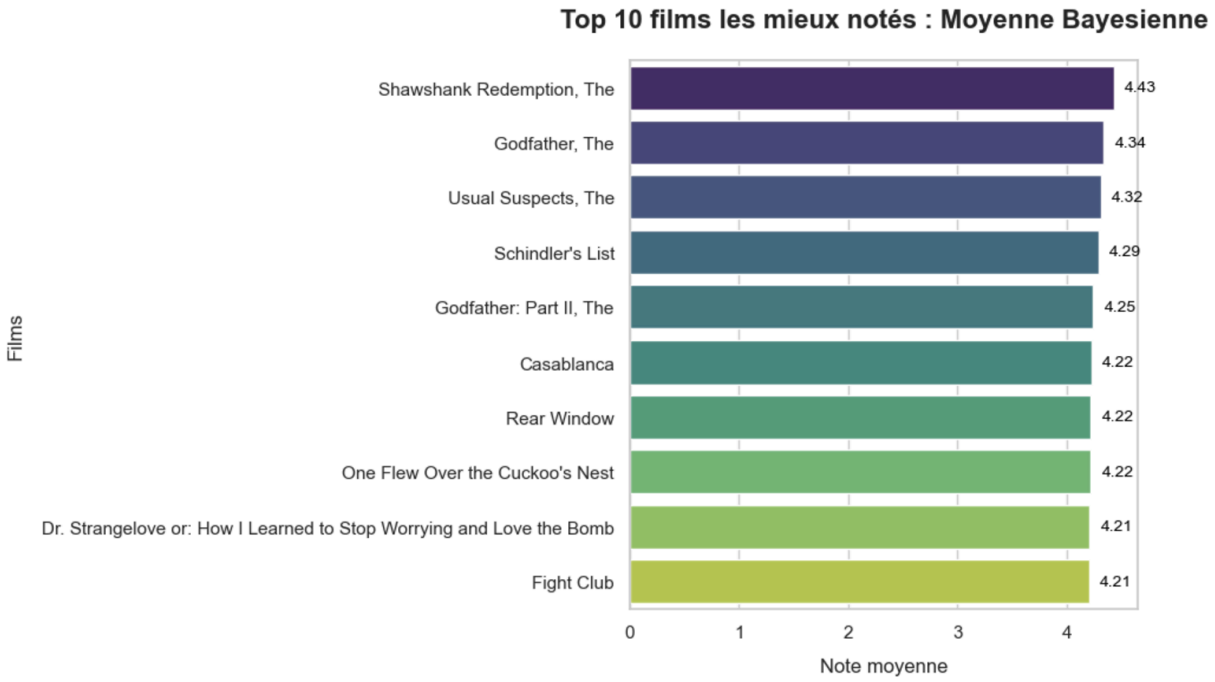
Distribution des notes



Ce premier graphique de visualisation des données est un histogramme des notes attribuées aux films par les utilisateurs de MovieLens. Ce modèle illustre la distribution des évaluations, qui tend à se concentrer autour de 3 à 4 étoiles. Cela indique que la majorité des utilisateurs attribue des notes moyennes à positives, ce qui reflète une tendance à donner des avis modérés sur les films.

Utilisation pour les recommandations : En utilisant cette distribution, nous pouvons adapter les recommandations. Par exemple, si un utilisateur a tendance à attribuer des notes élevées, l'outil de recommandation pourrait privilégier des films qui sont globalement bien notés par la communauté.

Top 10 des films avec les meilleures notes dans la base de données MovieLens compte tenu de la moyenne bayésienne

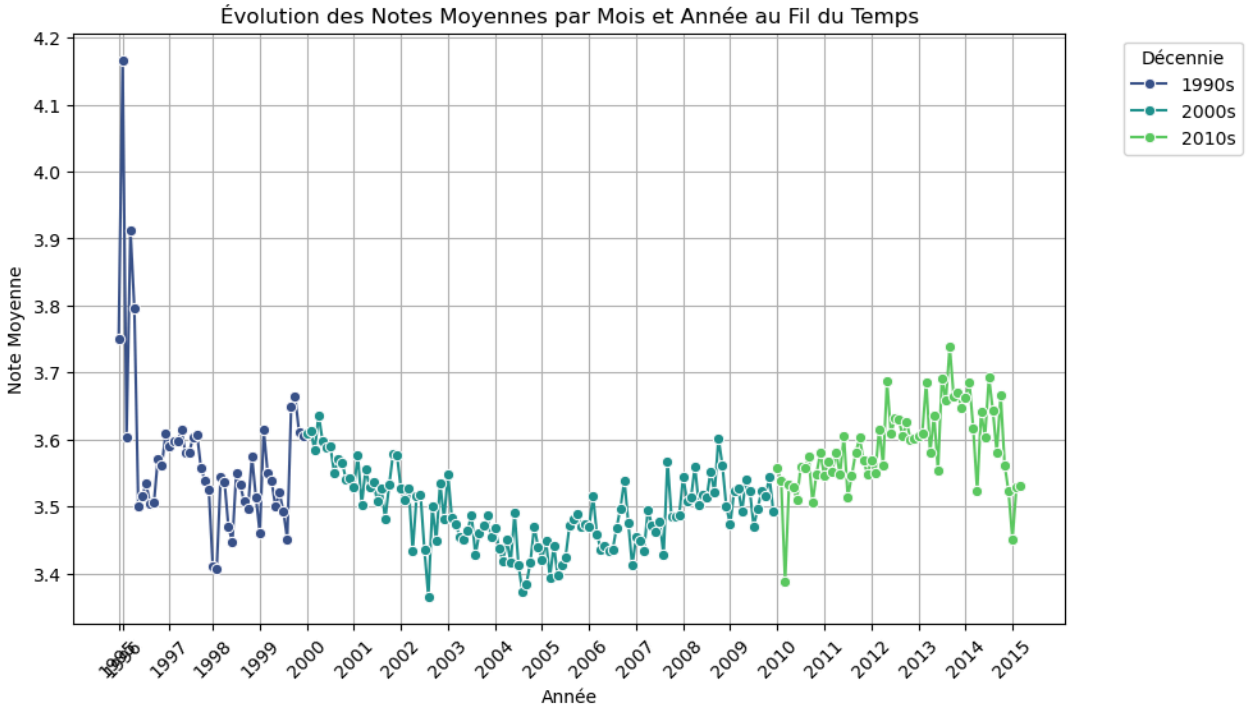


Le graphique présente les **10 films les mieux notés** selon une **moyenne bayésienne**, qui pondère les évaluations pour plus d'objectivité.

- **"The Shawshank Redemption"** se classe en tête (**4.43**), suivi de **"The Godfather"** (**4.34**) et **"The Usual Suspects"** (**4.32**).
- Des classiques comme **"Schindler's List"**, **"Casablanca"**, et **"Rear Window"** confirment leur place parmi les favoris du cinéma.
- Ce classement reflète la diversité des genres (drame, suspense) et la rigueur d'une méthode statistique qui valorise qualité et quantité des évaluations.

En somme, ce top 10 célèbre des chefs-d'œuvre intemporels, reconnus à la fois par les critiques et le public.

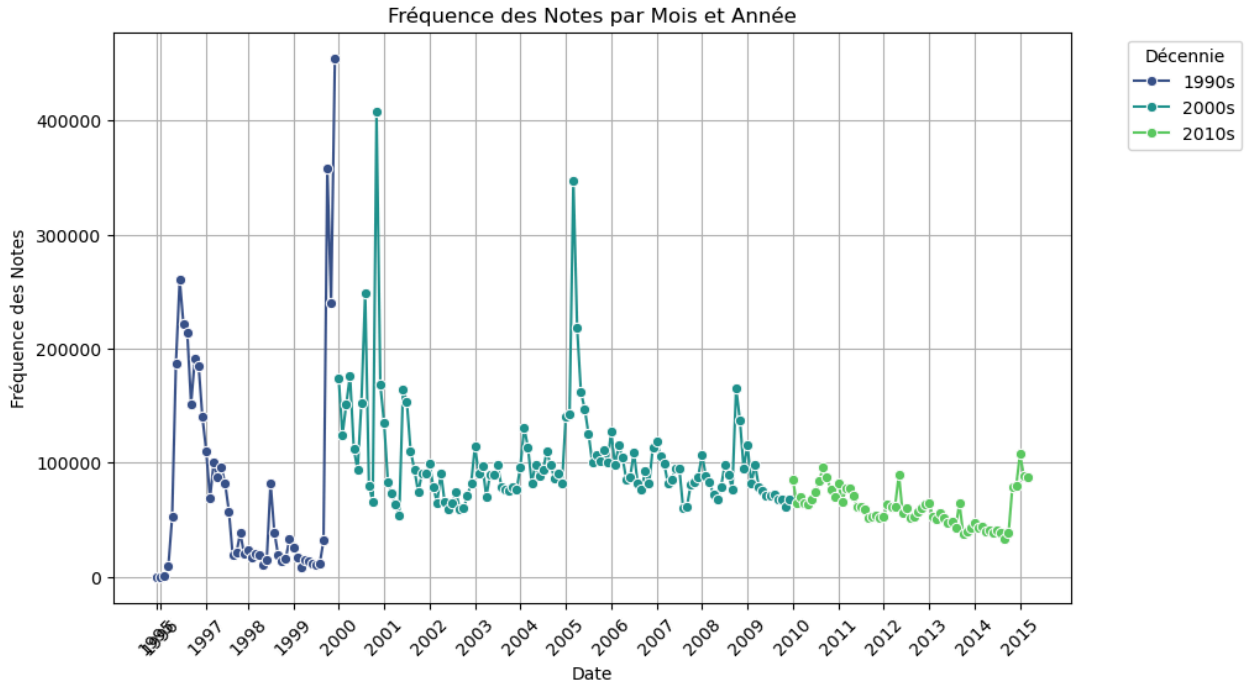
Évolution des notes dans le temps (moyenne)



Ce graphique illustre l'évolution des notes moyennes attribuées aux films dans MovieLens, montrant une tendance à la baisse dans les années 1990, une stagnation autour de 3.5 dans les années 2000, et une remontée progressive dans les années 2010. Cette évolution reflète l'évolution des préférences des utilisateurs au fil du temps.

Utilisation pour les recommandations : En exploitant ces données, un système de recommandation peut s'adapter aux préférences temporelles. Par exemple, si un utilisateur privilégie des films récents bien notés, l'outil pourrait proposer davantage de films produits après 2010.

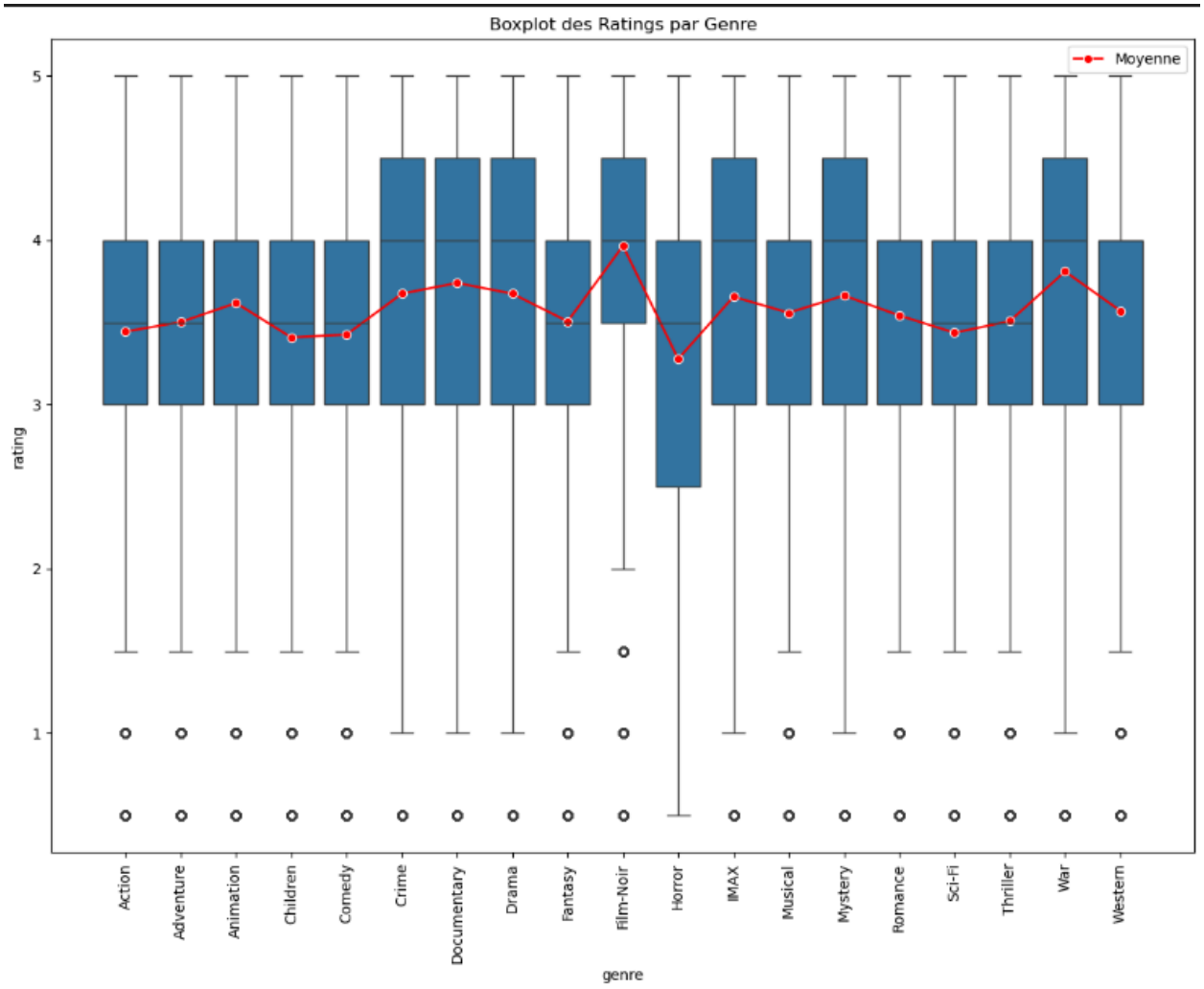
Évolution des notes dans le temps (fréquence)



Ce graphique montre la fréquence des notes attribuées aux films par décennie. On observe une augmentation marquée de l'activité autour des années 2000, suivie d'une stabilisation pendant les années 2000 et une légère reprise dans les années 2010. Cette évolution reflète probablement l'élargissement progressif de la base d'utilisateurs et l'essor des plateformes de recommandation.

Utilisation pour les recommandations : En intégrant la fréquence des notes, le système peut mieux adapter ses suggestions. Par exemple, une forte activité à une période donnée peut indiquer une tendance ou un engouement pour certains genres, utile pour affiner les recommandations à ces périodes.

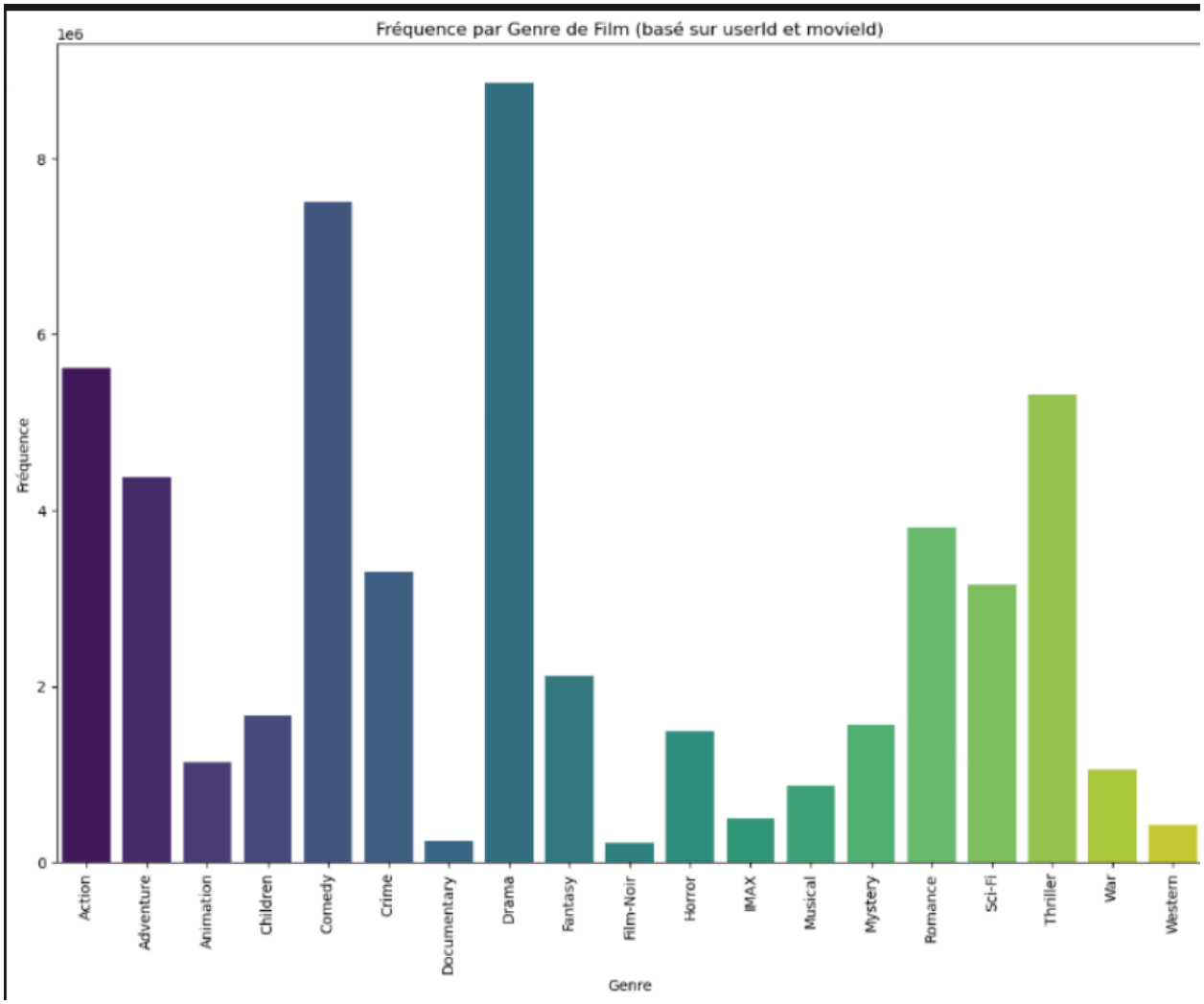
Distribution des genres



Ce boxplot montre la distribution des notes attribuées pour chaque genre de film. L'analyse révèle que la majorité des genres ont une médiane proche de 3 à 4 étoiles, ce qui reflète une cohérence dans les évaluations globales. Cependant, il existe une forte dispersion des notes pour tous les genres, avec des valeurs extrêmes, ce qui indique une diversité d'opinions parmi les utilisateurs.

Interprétation pour les recommandations : Cette absence de différences significatives entre les genres suggère que les goûts des utilisateurs ne peuvent pas être facilement prédits en se basant uniquement sur le genre. Un système de recommandation efficace devra donc prendre en compte d'autres paramètres, comme les préférences individuelles et l'historique de visionnage, pour offrir des recommandations plus personnalisées.

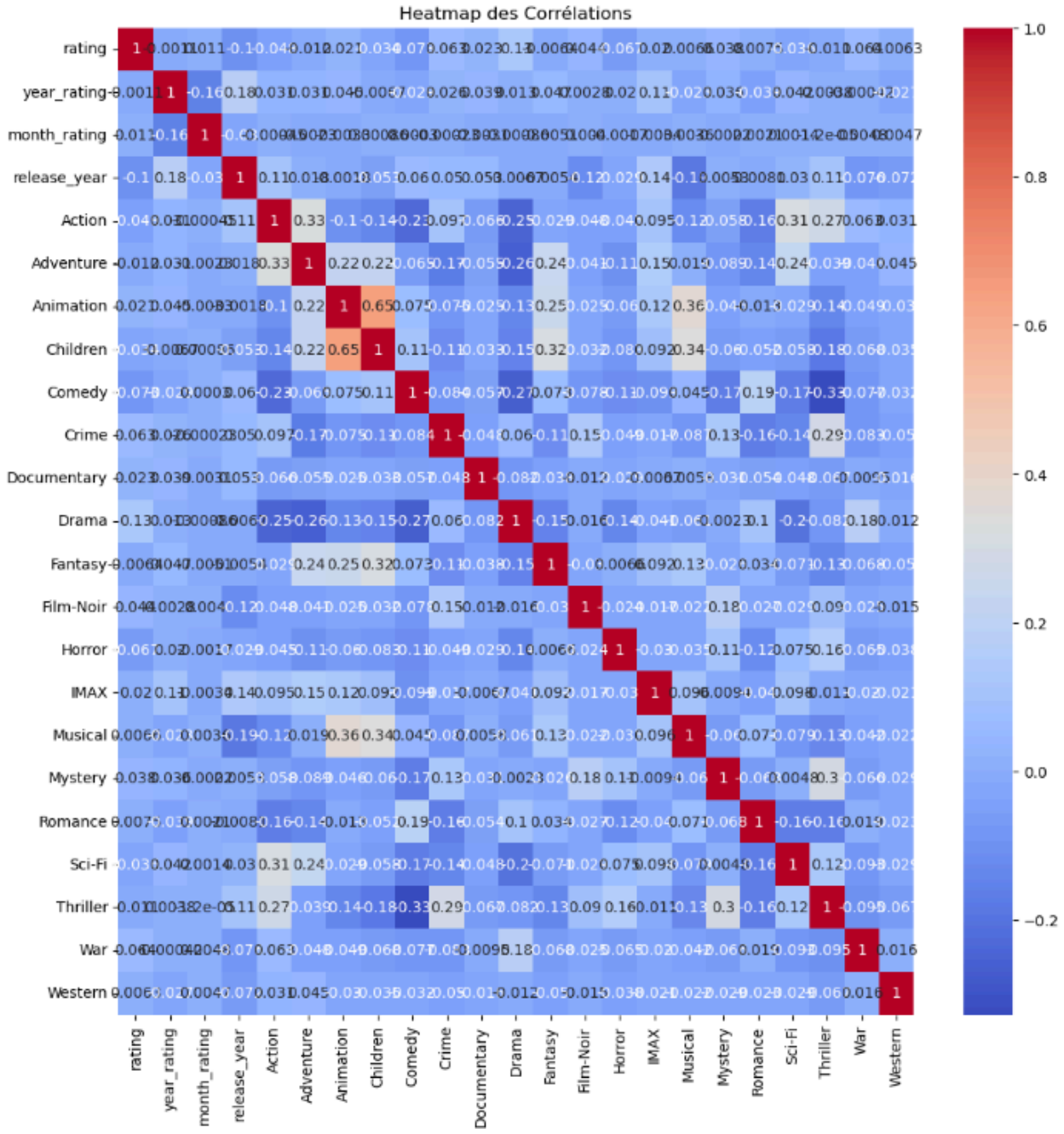
Fréquence par genre



Ce graphique montre la fréquence des évaluations par genre de film. On observe que certains genres, comme Drama et Comedy, reçoivent un nombre élevé de notations, ce qui indique leur popularité auprès des utilisateurs. En revanche, des genres tels que Documentary, IMAX, et Film-Noir sont moins notés, reflétant probablement une audience plus restreinte ou une moindre disponibilité dans la base de données.

Interprétation pour les recommandations : Cette distribution suggère que certains genres dominent en termes de fréquence d'évaluations. Un système de recommandation doit tenir compte de cette surreprésentation afin d'éviter un biais vers les genres les plus populaires. En parallèle, il est important d'explorer des recommandations plus diversifiées, notamment pour des genres moins fréquemment évalués, afin d'offrir une expérience plus riche et personnalisée aux utilisateurs.

Heatmap des corrélations



Cette heatmap montre les corrélations entre les notes et les genres de films. L'analyse révèle que peu de variables présentent des corrélations fortes entre elles, que ce soit positivement ou négativement. La majorité des coefficients de corrélation sont proches de 0, ce qui indique qu'il existe peu de relations significatives entre les différentes catégories des bases de données de movieLens (par exemple, entre les genres et les notes attribuées).

Interprétation pour les recommandations : Cette faible corrélation suggère que les goûts des utilisateurs sont diversifiés et peu prévisibles à partir d'une seule variable. Un système de recommandation doit donc être multidimensionnel, combinant plusieurs facteurs (comme l'historique de l'utilisateur et les tendances récentes) pour offrir des suggestions pertinentes.

Cela montre aussi qu'un modèle de Deep Learning pouvant capter les signaux faibles pourrait se révéler plus adapté pour notre problématique.

Analyses statistiques

Test ANOVA, chi 2, V de Cramer : les tests habituels d'apprentissage supervisé ne sont pas spécifiquement conçus pour le filtrage collaboratif.

Le filtrage collaboratif repose plutôt sur des techniques de similarité et de matrice de notation pour prédire les préférences des utilisateurs. Les métriques couramment utilisées pour évaluer ces modèles incluent la précision, le rappel, le score F1, l'erreur quadratique moyenne (MSE) et l'erreur absolue moyenne (MAE). Nous allons observer ces deux dernières plus précisément dans l'étape de modélisation (et même la RMSE plutôt que la MSE pour être exact).

Étape 2 : rapport de modélisation

Classification du problème

Type de problème

Les systèmes de recommandation reposent sur deux principaux types de problèmes en Machine Learning: le problème de filtrage collaboratif (apprentissage supervisé ou non supervisé) et le problème de filtrage basé sur le contenu (apprentissage supervisé). Il existe aussi des approches dites hybrides (apprentissage supervisé ou non supervisé) ou encore contextuelles (apprentissage supervisé ou apprentissage par renforcement). Vous pouvez trouver ci-dessous une liste exhaustive des approches possibles :

Filtrage Collaboratif

Description :
Le filtrage collaboratif se base sur les interactions passées entre les utilisateurs et les éléments (par exemple, les films regardés, les produits achetés).
Il vise à prédire les préférences d'un utilisateur en fonction des comportements d'autres utilisateurs similaires.

Exemple :
Un utilisateur qui a aimé des films que d'autres utilisateurs similaires ont appréciés se verra recommander les mêmes films.

Techniques courantes :

- Factorisation matricielle : Décomposition de matrices utilisateur-élément pour identifier des dimensions latentes.
- KNN (k-nearest neighbors) : Recherche d'utilisateurs ou d'éléments similaires.
- Réseaux neuronaux : Apprentissage des embeddings pour les utilisateurs et les éléments.

Filtrage basé sur le contenu

Description :
Ce type repose sur les caractéristiques des éléments et sur les préférences passées des utilisateurs.
Il vise à recommander des éléments similaires à ceux qu'un utilisateur a appréciés.

Exemple :
Si un utilisateur aime un film d'action avec un acteur spécifique, il se verra recommander d'autres films d'action similaires avec des caractéristiques comparables.

Techniques courantes :

- Classification : régression logistique, SVM.
- Apprentissage supervisé sur les caractéristiques : Modèles qui apprennent à prédire les préférences en fonction des attributs des éléments.

Apprentissage hybride

Description :
Combine les deux approches (filtrage collaboratif et basé sur le contenu) pour tirer parti des interactions utilisateur-élément et des caractéristiques des éléments.

Exemple :
Netflix combine les évaluations passées des utilisateurs (collaboratif) et des caractéristiques des films comme les genres, les réalisateurs, ou les acteurs (contenu).

Techniques courantes :

- Factorisation matricielle hybride.
- Deep Learning : Auto-encoders, modèles attentionnels.

Approches contextuelles

Description :
Prend en compte le contexte (temps, lieu, humeur) pour ajuster les recommandations.

Exemple :
Spotify recommande une playlist différente en fonction de l'heure de la journée.

Technique courante :
Apprentissage par renforcement (Reinforcement Learning) : Bandit algorithms, qui adaptent les recommandations en temps réel pour maximiser l'engagement.

En résumé, les systèmes de recommandation couvrent des problématiques variées de classification, de régression, de clustering, ou encore d'apprentissage par renforcement, selon l'approche utilisée.

Finalement nous avons décidé de nous concentrer sur certaines approches en particulier et de tester en priorité les modèles suivants :

- **Filtrage Collaboratif : approche mémoire**
    - **User-based**
    - **Item-based**
- **Filtrage Collaboratif : approche modèle**
    - **Item-based + SVD**
    - **Surprise (SVD)**
- **Filtrage basé sur le contenu**

Tâches de Machine Learning

Le sujet principal de notre projet est la création d’un système de recommandation.
Comme nous l’avons vu dans la partie précédente, il existe beaucoup d’approches possibles. Les systèmes de recommandation s'apparentent à plusieurs tâches spécifiques en Machine Learning, en fonction de l'approche utilisée.

Voici un aperçu détaillé des correspondances :

Classification

Description :
Lorsqu'il s'agit de prédire si un utilisateur aimera ou n'aimera pas un élément (binaire), ou si un utilisateur appartient à une catégorie de comportement spécifique.

Exemple :
Prédire si un utilisateur évaluera un film comme "positif" ou "négatif".

Techniques courantes :
Régression logistique, SVM, arbres de décision, Random Forests.

Régression

Description :
Dans les cas où l'on prédit une valeur continue, comme une note (ex. : de 1 à 5 étoiles) que l'utilisateur donnera à un produit ou film.

Exemple :
Estimer que l'utilisateur X donnera 4,3 étoiles à un restaurant.

Techniques courantes :
Régressions linéaires ou non linéaires, réseaux neuronaux.

Clustering

Description :
Regrouper des utilisateurs ou des éléments similaires en fonction de caractéristiques communes.

Exemple :
Identifier des groupes d'utilisateurs ayant des préférences similaires ou des clusters d'éléments populaires auprès de profils similaires.

Techniques courantes :
K-Means, DBSCAN, algorithmes de clustering hiérarchique.

Réduction de dimensionnalité

Description :
Extraire des représentations latentes des utilisateurs ou des éléments à partir d'une grande matrice utilisateur-élément pour simplifier les calculs et capturer des similarités.

Exemple :
Découvrir que deux films sont similaires car ils partagent une "dimension latente" liée à leur genre ou à leur public cible.

Techniques courantes :
Factorisation matricielle (SVD, NMF), PCA, auto-encodeurs.

Apprentissage par renforcement

Description :
Adapter les recommandations en temps réel en maximisant l'engagement de l'utilisateur (ex. : clics, temps passé, achats).

Exemple :
Afficher des suggestions basées sur l'interaction récente d'un utilisateur avec le système.

Techniques courantes :
Bandit algorithms, Q-Learning, Deep Q-Networks (DQN).

Apprentissage supervisé

Description :
Utilisé pour prédire les préférences d'un utilisateur en fonction des interactions passées ou des caractéristiques des éléments.

Exemple :
Utiliser un modèle supervisé pour apprendre les préférences utilisateur à partir de données labellisées.

Techniques courantes :
Régressions, réseaux neuronaux, arbres de décision.

Apprentissage non supervisé

Description :
Trouver des relations implicites dans les données sans avoir de labels explicites.

Exemple :
Identifier automatiquement des groupes d'utilisateurs avec des comportements similaires.

Techniques courantes :
Clustering, factorisation matricielle, réseaux neuronaux non supervisés.

Tâches hybrides

Description :
Combinaison des tâches ci-dessus pour maximiser la précision des recommandations.

Exemple :
Associer le filtrage collaboratif et le filtrage basé sur le contenu.

En résumé, les systèmes de recommandation s'apparentent principalement aux tâches de Machine Learning suivantes :

- **Classification (prédire un choix binaire ou multi-catégorique).**
- **Régression (prédire une note ou une probabilité).**
- **Clustering (regrouper des utilisateurs ou des éléments similaires).**
- **Réduction de dimensionnalité (extraire des représentations latentes).**
- **Apprentissage par renforcement (adapter les recommandations dynamiquement).**

Le choix de la tâche dépend de l'objectif spécifique du système de recommandation et des données disponibles.

Avec les modèles que nous allons détailler dans les pages suivantes, nous avons principalement travaillé sur les tâches de régression et de réduction de dimensionnalité (filtrage collaboratif) et dans une moindre mesure sur des tâches de clustering (similarité des éléments dans le filtrage basé sur le contenu).

Métrique de performance principale

La principale métrique utilisée pour évaluer nos modèles est le Root Mean Squared Error (RMSE). Cette métrique a été choisie car elle est particulièrement adaptée aux systèmes de recommandation qui prédisent explicitement les notes des utilisateurs (par exemple, sur une échelle de 1 à 5 étoiles).

Le RMSE a l'avantage de pénaliser davantage les erreurs importantes, ce qui permet de minimiser les prédictions très éloignées des valeurs réelles, un aspect essentiel pour garantir la fiabilité des recommandations.

Autres métriques

En complément du RMSE, nous avons également utilisé la métrique Mean Absolute Error (MAE). Contrairement au RMSE, la MAE présente plusieurs avantages spécifiques :

- Facilité d'interprétation : La MAE mesure directement l'erreur moyenne en termes d'unités de la variable prédite. Elle ne donne pas un poids disproportionné aux grandes erreurs, ce qui la rend facile à comprendre et adaptée lorsque toutes les erreurs doivent être traitées de manière égale.
- Robustesse face aux valeurs aberrantes : La MAE est moins influencée par les écarts importants (outliers) que le RMSE, ce qui permet une évaluation plus équilibrée des performances globales du modèle.

Ces deux métriques (RMSE et MAE) offrent une vision complémentaire des performances du système, en permettant à la fois d'évaluer la précision globale et de comprendre l'impact des grandes erreurs.

Choix du modèle et optimisation

Algorithmes essayés

Nous n’avons pas eu le temps d’essayer tous les modèles possibles ni d'expérimenter avec toutes les techniques listées précédemment. Mais voici les différentes modélisations que nous avons pu effectuer:

Filtrage collaboratif: approche mémoire

User-based

On cherche à prédire la note d’un film grâce à une matrice de notation et au calcul d’une similarité cosinus sur ses vecteurs horizontaux: les utilisateurs.

```python
## Création de la matrice de notations
mat_ratings = filtered_data.pivot_table(columns = 'imdbId', index = 'userId', values = 'rating')

# Affichage de la taille de la matrice
print(f"Nombre d'utilisateurs : {mat_ratings.shape[0]}")
print(f"Nombre de films : {mat_ratings.shape[1]}")

# Augmentation de la notation de 1
mat_ratings = mat_ratings + 1

# Remplacement des valeurs manquantes par des 0
mat_ratings.fillna(0, inplace=True)

# Création de la matrice CSR
sparse_ratings = csr_matrix(mat_ratings)

# Stockage des identifiants utilisateurs et films
user_ids = mat_ratings.index.tolist()
films_ids = mat_ratings.columns.tolist()
```

✓ 2.3s Python

```python
# Utilisation de la fonction 'cosine_similarity' du module 'dist' pour calculer la similarité cosinus entre les utili
user_similarity = dist.cosine_similarity(sparse_ratings)

# Création d'un DataFrame à partir de la matrice de similarité entre utilisateurs.
# Les index et les colonnes du DataFrame sont les identifiants des utilisateurs.
user_similarity = pd.DataFrame(user_similarity, index=user_ids, columns=user_ids)
```

✓ 2.9s Python

On choisit un utilisateur. On sélectionne des films qui n'ont pas encore été vus par l'utilisateur ainsi que les k utilisateurs les plus similaires en excluant l'utilisateur lui-même. On obtient la note prédite grâce au calcul du produit scalaire entre ratings et similar_users.

```python
# Création d'une fonction de prédiction optimisée pour des calculs plus rapides
def pred_user_optimized(mat_ratings, user_similarity, k, userId):
    # Films non vus
    to_predict = mat_ratings.loc[userId][mat_ratings.loc[userId] == 0]

    # k utilisateurs les plus similaires
    similar_users = user_similarity.loc[userId].sort_values(ascending=False)[1:k+1]

    # Normalisation
    norm = np.sum(np.abs(similar_users))

    # Calcul matriciel des prédictions
    ratings_matrix = mat_ratings.loc[similar_users.index, to_predict.index]
    pred = np.dot(similar_users.values, ratings_matrix.values) / norm

    # Retour des prédictions sous forme de série
    return pd.Series(pred, index=to_predict.index)
```

✓ 0.0s Python

L'objectif est d'arriver à une liste de recommandations du type "Les utilisateurs qui ont noté les mêmes films que vous, ont aussi noté…"

Après application du modèle, on obtient bien 10 films:

```
Recommandations user similarity (title_name) : imdbId
Pleasantville          5.500000
Monsters, Inc.         5.347657
Dogma                  5.347657
Sin City               5.337072
American History X     5.336862
Truman Show, The       5.184730
Shrek 2                5.174145
From Dusk Till Dawn    5.173934
Green Mile, The        5.173723
Prestige, The          5.173723
Name: 11, dtype: float64
```

Les films recommandés sont plutôt populaires et on remarque que certains ont des notes identiques. On peut deviner une vague similitude, mais le lien entre les films préférés d'un utilisateur et la liste recommandée n'est pas forcément évident a priori.

Pour illustrer ce propos, voici la liste des films les mieux notés par le même utilisateur:

	userId	imdbId	rating	title_name	imdb_averageRating	imdb_numVotes
1207	11	268695	5.0	Time Machine, The	6.0	131665
1236	11	183649	5.0	Phone Booth	7.1	289665
1215	11	145487	5.0	Spider-Man	7.4	897455
1216	11	276751	5.0	About a Boy	7.1	193028
1221	11	120912	5.0	Men in Black II (a.k.a. MIIB) (a.k.a. MIB 2)	6.2	410409
1225	11	310793	5.0	Bowling for Columbine	8.0	149594
1227	11	295297	5.0	Harry Potter and the Chamber of Secrets	7.4	705854
1228	11	133240	5.0	Treasure Planet	7.2	136269
1229	11	238380	5.0	Equilibrium	7.3	350544
1230	11	167261	5.0	Lord of the Rings: The Two Towers, The	8.8	1819156

Après avoir créé deux ensembles Train et Test, nous avons pu évaluer le modèle et obtenir un RMSE de 3.27 et un MAE de 2.85 pour k=3 (3 plus proches voisins utilisateurs).

```python
# Définition d'une fonction pour évaluer la performance
def evaluate_model(train_ratings, test_ratings, user_similarity, k):
    y_true = []
    y_pred = []

    for user in test_ratings.index:
        # Prédictions pour l'utilisateur
        predictions = pred_user_optimized(train_ratings, user_similarity, k, user)

        # Notes réelles dans l'ensemble de test
        true_ratings = test_ratings.loc[user][test_ratings.loc[user] > 0]

        # Alignement des indices pour comparer les prédictions et les vraies valeurs
        common_indices = true_ratings.index.intersection(predictions.index)
        y_true.extend(true_ratings[common_indices].values)
        y_pred.extend(predictions[common_indices].values)

    # Calcul de la RMSE (Root Mean Squared Error)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))

    # Calcul de la MAE (Mean Absolute Error)
    mae = mean_absolute_error(y_true, y_pred)

    return rmse, mae

# Choix du nombre de voisins (k)
k = 3

# Évaluation du modèle
rmse, mae = evaluate_model(train_ratings, test_ratings, user_similarity_train, k)
print(f"RMSE du modèle avec k={k} : {rmse:.4f}")
print(f"MAE du modèle avec k={k} : {mae:.4f}")
```

✓ 3m 35.3s Python

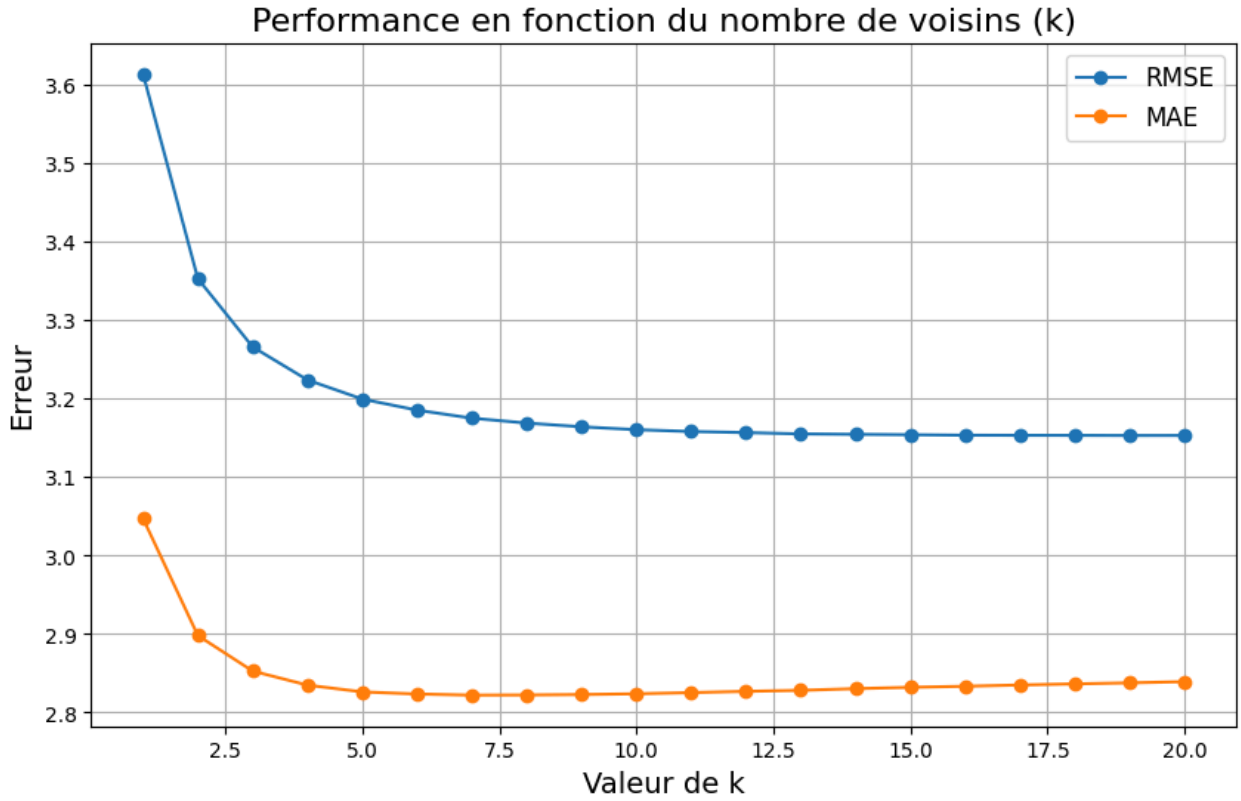
Nous avons ensuite lancé un GridSearch afin de trouver la valeur de k optimisant la RMSE (k=19) pour une RMSE de 3.15.

```python
# Liste des valeurs de k à tester
k_values = range(1, 21)  # Tester k entre 1 et 20

# Stocker les résultats
results = []

# Boucle sur chaque valeur de k
for k in k_values:
    rmse, mae = evaluate_model(train_ratings, test_ratings, user_similarity_train, k)
    results.append((k, rmse, mae))
    print(f"k={k}, RMSE={rmse:.4f}, MAE={mae:.4f}")

# Conversion des résultats en DataFrame
results_df = pd.DataFrame(results, columns=["k", "RMSE", "MAE"])

# Trouver la valeur optimale de k pour le RMSE
best_k_rmse = results_df.loc[results_df['RMSE'].idxmin(), 'k']
print(f"Meilleure valeur de k pour RMSE : {best_k_rmse}, RMSE : {results_df['RMSE'].min():.4f}")

# Trouver la valeur optimale de k pour la MAE
best_k_mae = results_df.loc[results_df['MAE'].idxmin(), 'k']
print(f"Meilleure valeur de k pour MAE : {best_k_mae}, MAE : {results_df['MAE'].min():.4f}")
```

✓ 89m 1.1s Python

Pour information, le k optimisant le MAE est différent (k=7) pour un MAE de 2.82.

Vous pouvez observer l'évolution des deux courbes dans le graphique ci-dessous:



En relançant la fonction de prédiction avec le k optimal, on obtient une liste de films recommandés différente:

```
Recommandations user similarity (title_name) : imdbId
Truman Show, The                                          4.772451
Monsters, Inc.                                            4.767596
Incredibles, The                                          4.705387
Sin City                                                  4.637971
Seven (a.k.a. Se7en)                                      4.384220
Kill Bill: Vol. 2                                         4.377030
Léon: The Professional (a.k.a. The Professional) (Léon)   4.310374
Ocean's Eleven                                            4.294483
Shrek 2                                                   4.244641
Indiana Jones and the Temple of Doom                      4.241282
dtype: float64
```

A noter que l'on n'observe plus le problème de notes strictement égales en utilisant ce k optimisant le RMSE.

Ces métriques nous ont paru très élevés pour une note sur 5 à tel point que nous avons décidé d'effectuer une vérification ultérieure et de recalculer les métriques de performance. En les calculant à nouveau avec un code différent et en alignant la base de données et la cible pour tous les modèles (utilisation du score de pertinence sur 100), nous avons pu finalement obtenir un RMSE et un MAE bien meilleurs et plus en ligne avec les autres modèles, autour de 15%. Cela nous a semblé beaucoup plus cohérent comme résultats.

Item-based

On cherche à prédire la note d'un film grâce à une matrice de notation et au calcul d'une similarité cosinus mais cette fois sur ses vecteurs verticaux (matrice transposée): les films.

```python
# Le calcul de la similarité se fait sur les colonnes cette fois.
# Il suffit de transposer notre matrice creuse `sparse_ratings` et de faire le calcul de la similarité cosinus dessus
item_similarity = dist.cosine_similarity(sparse_ratings.T)

item_similarity = pd.DataFrame(item_similarity, index = films_ids, columns = films_ids)
```

✓ 3.5s Python

Comme dans la méthode précédente, on choisit un utilisateur. On sélectionne des films qui n'ont pas encore été vus par l'utilisateur ainsi que les k films les plus similaires en excluant le film lui-même. On obtient la note prédite grâce au calcul du produit scalaire entre ratings et similar_items.

```python
# Création de la fonction de prédiction
def pred_item(mat_ratings, item_similarity, k, userId):
    # Films non vus par l'utilisateur
    to_predict = mat_ratings.loc[userId][mat_ratings.loc[userId] == 0]

    # Stockage des prédictions
    predictions = {}

    for i in to_predict.index:
        # Sélection des k films les plus similaires
        similar_items = item_similarity.loc[i].sort_values(ascending=False)[1:k+1]

        # Calcul de la norme
        norm = np.sum(np.abs(similar_items))

        # Vérification pour éviter la division par zéro
        if norm == 0:
            predictions[i] = 0  # Si la norme est zéro, prédire 0
            continue

        # Récupération des notes données par l'utilisateur aux k plus proches voisins
        ratings = mat_ratings[similar_items.index].loc[userId]

        # Calcul du produit scalaire
        scalar_prod = np.dot(ratings, similar_items)

        # Prédiction normalisée
        predictions[i] = scalar_prod / norm

    # Retour des prédictions sous forme de série
    return pd.Series(predictions)
```

✓ 0.0s Python

L'objectif est d'arriver à une liste de recommandations du type "Ces films ont des notes similaires aux films que vous avez notés..."

Après application du modèle, on obtient bien 10 films:

```
Recommandations item similarity (title_name) : imdbId
Green Mile, The                                    6.0
Buffy the Vampire Slayer                           6.0
Arachnophobia                                      6.0
Mummy, The                                         6.0
Gift, The                                          6.0
Planet of the Apes                                 6.0
Groundhog Day                                      6.0
Terminator Salvation                               6.0
Erin Brockovich                                    6.0
Master and Commander: The Far Side of the World    6.0
Name: 11, dtype: float64
```

On remarque que ces films sont différents des films obtenus par le filtrage collaboratif user-based.

Et on voit ici apparaître un problème auquel nous nous attaquerons plus tard: les 10 films recommandés obtiennent tous la note maximum.

Il pourrait être intéressant de voir combien de films obtiennent cette même note... En tous cas, cela ne nous permet pas de les proposer aux utilisateurs dans un ordre de priorité satisfaisant.

Dans un premier temps, nous n'avons pas pu implémenter de mesure de la performance pour ce modèle. Nous avons écrit un code pour cela, mais les calculs étaient trop longs (similarité à calculer pour trop de vecteurs). Il semble que le temps de calcul pour une seule itération (k=3) était au-delà de 24 heures pour l'évaluation du modèle (nous avons interrompu le calcul avant de pouvoir aller au bout).

Dans ces conditions, nous avons préféré éviter de lancer la cellule pour un GridSearch sur plus de 20 valeurs de k, ce qui n'aurait fait que multiplier les calculs par 20...

Cela dit, augmenter le nombre de voisins (k) a quand même permis d'avoir des notes plus différenciées. En effet, nous avons utilisé k=19 qui était l'optimum pour le modèle user-based précédent et avons obtenu la liste suivante :

```
Recommandations item similarity (title_name) : Flatliners                    5.792071
Arachnophobia                          5.765309
Groundhog Day                          5.764323
Thirteenth Floor, The                  5.764270
Indiana Jones and the Temple of Doom   5.733754
X-Files: Fight the Future, The         5.712600
Mummy, The                             5.688687
Fugitive, The                          5.631752
Running Man, The                       5.629523
Jumanji                                5.605842
dtype: float64
```

Avec un peu plus de temps à la fin du projet, nous avons réussi à calculer les métriques de performance avec un code différent et en alignant la base de données et la cible pour tous les modèles (score de pertinence sur 100), nous avons pu finalement obtenir un RMSE et un MAE en ligne avec les autres modèles, autour de 15%. Le modèle item-based a obtenu des performances légèrement supérieures au modèle user-based (amélioration de l'ordre de 1%). Encore une fois, cela nous a semblé cohérent.

Filtrage collaboratif: approche modèle (factorisation matricielle)

Item-based + SVD

On ajoute l'utilisation d'un modèle TruncatedSVD à la méthode vue précédemment pour réduire les dimensions de notre matrice. Il s'agit d'une factorisation qui fait apparaître des facteurs latents.

```python
## Application du modèle SVD

# Décomposition en Valeurs Singulières Tronquée
svd = TruncatedSVD(n_components=12)

# Transformation de la matrice transposée de sparse_ratings
ratings_red = svd.fit_transform(sparse_ratings.T)

# Affichage des dimensions de la matrice pour vérification
print("Dimensions de la matrice réduite :", ratings_red.shape)

# La transformation SVD tronquée a réduit notre matrice et nous a renvoyé une matrice de dimension (l*c) où
# l correspond aux nombre de films et c au nombre de facteurs latents. Soit 16346*12 dans cet exemple.
```

✓ 0.5s Python

```python
# On peut calculer la similarité entre les films directement sur la matrice réduite ou
# en reconstruisant notre matrice de notations à travers la méthode inverse_transform.
# Cette nouvelle matrice aura les estimations des notes manquantes dans notre matrice originale.

# Reconstruction de la matrice originale
new_ratings_red = svd.inverse_transform(ratings_red)
```

✓ 0.0s Python

On peut ensuite calculer la similarité entre les films de deux manières différentes: soit directement sur la matrice réduite, soit sur une matrice reconstituée.

```python
# Similarité sans reconstruction de la matrice originale
item_similarity = dist.cosine_similarity(ratings_red)

# Similarité avec reconstruction de la matrice originale
item_similarity2 = dist.cosine_similarity(new_ratings_red)

# Conversion des matrices de similarité en DataFrame
item_similarity = pd.DataFrame(item_similarity, index = films_ids, columns = films_ids)
item_similarity2 = pd.DataFrame(item_similarity2, index = films_ids, columns = films_ids)
```

✓ 6.2s Python

```python
## Utilisation de la fonction de prédiction pour faire des recommandations

# Sélection de l'utilisateur
user = 11

# Utilisation d'item_similarity pour faire 10 recommandations à l'utilisateur
reco_user = pred_item(mat_ratings, item_similarity, 3, user).sort_values(ascending=False).head(10)

# Utilisation d'item_similarity2 pour faire 10 recommandations à l'utilisateur
reco_userbis = pred_item(mat_ratings, item_similarity2, 3, user).sort_values(ascending=False).head(10)
```

✓ 42.7s Python

Après application du modèle, on obtient les deux listes suivantes :

```
Recommandations item similarity + SVD (title_name) : imdbId
Top Gun                                              6.000000
Yamakasi - Les samouraïs des temps modernes          6.000000
Honey, I Shrunk the Kids                             6.000000
Groundhog Day                                        6.000000
Metropolis                                           5.833840
Monsters, Inc.                                       5.833000
Da Vinci Code, The                                   5.667278
Kingdom of Heaven                                    5.666859
Charlie's Angels                                     5.666660
K-PAX                                                5.666485
Name: 11, dtype: float64
```

```
Recommandations item similarity 2 + SVD (title_name) : imdbId
Groundhog Day                                        6.000000
Top Gun                                              6.000000
Honey, I Shrunk the Kids                             6.000000
Yamakasi - Les samouraïs des temps modernes          6.000000
Metropolis                                           5.833840
Monsters, Inc.                                       5.833000
Da Vinci Code, The                                   5.667278
Kingdom of Heaven                                    5.666859
Charlie's Angels                                     5.666660
K-PAX                                                5.666485
Name: 11, dtype: float64
```

Dans cet exemple, les films recommandés sont les mêmes selon les deux méthodes, mais ils sont différents des films trouvés par les deux variantes du filtrage collaboratif de base (user-based et item-based de l'approche mémoire).

On voit ici apparaître le même problème que pour le filtrage collaboratif item-based de base: beaucoup de films obtiennent la même note, et notamment la note maximum.

On peut éviter ce problème dans une moindre mesure en changeant la valeur de k. En effet, si l'on utilise le k optimal trouvé avec notre premier modèle, pour k=19, il n'y a plus de films avec les mêmes notes. Et la liste est différente de celle obtenue avec k=3, même si certains films sont communs.:

```python
# On choisit maintenant k=19 comme avec les modèles précédents

# Sélection de l'utilisateur
user = 11

# Utilisation d'item_similarity pour faire 10 recommandations à l'utilisateur
reco_user_k19 = pred_item(mat_ratings, item_similarity, 19, user).sort_values(ascending=False).head(10)

# Utilisation d'item_similarity2 pour faire 10 recommandations à l'utilisateur
reco_userbis_k19 = pred_item(mat_ratings, item_similarity2, 19, user).sort_values(ascending=False).head(10)
```

✓ 44.9s Python

```
Recommandations item similarity avec k=19 + SVD (title_name) : imdbId
Indiana Jones and the Temple of Doom                5.398427
Groundhog Day                                       4.821858
Ice Age                                             4.607218
Truman Show, The                                    4.501755
Mummy, The                                          4.478065
Who Framed Roger Rabbit?                            4.475699
Crouching Tiger, Hidden Dragon (Wo hu cang long)    4.342115
Monsters, Inc.                                      4.282679
Sleepy Hollow                                       4.127065
Green Mile, The                                     4.102033
Name: 11, dtype: float64

Recommandations item similarity 2 avec k=19 + SVD (title_name) : imdbId
Indiana Jones and the Temple of Doom                5.398427
Groundhog Day                                       4.821858
Ice Age                                             4.607218
Truman Show, The                                    4.501755
Mummy, The                                          4.478065
Who Framed Roger Rabbit?                            4.475699
Crouching Tiger, Hidden Dragon (Wo hu cang long)    4.342115
Monsters, Inc.                                      4.282679
Sleepy Hollow                                       4.127065
Green Mile, The                                     4.102033
Name: 11, dtype: float64
```

Avec un peu plus de temps à la fin du projet, nous avons réussi à calculer les métriques de performance pour ce modèle également en alignant la base de données et la cible pour tous les modèles (score de pertinence sur 100), nous avons pu obtenir un RMSE et un MAE en ligne avec les autres modèles, autour de 15%. Le modèle item-based avec SVD a obtenu des performances légèrement supérieures au modèle item-based “simple”. Cela nous a semblé cohérent. On y reviendra plus tard en présentant notre benchmark des modèles.

Filtrage basé sur le contenu

Filtrage à partir des bases de données produits de l’étape de nettoyage, contenant une version enrichie de MovieLens, intégrant des informations issues d'IMDB ainsi qu'une base de données sur les acteurs. L'objectif initial était de fusionner ces différentes sources afin de créer une base cohérente et exploitable pour des analyses ultérieures.

Nous avons suivi les étapes suivantes :

Partie I. Nettoyage et Simplification des Données

Pour préparer la base de données à des analyses avancées, des traitements ont été appliqués pour simplifier certaines colonnes, éliminer les informations superflues et organiser la structure.

1. Limitation des Informations sur les Acteurs

Les noms des acteurs ont été limités aux quatre premiers pour chaque film. Cela a permis de simplifier la colonne actors_primaryName en ne conservant que les noms les plus pertinents, facilitant ainsi la manipulation et la lisibilité des données.

Ce choix a été motivé par la nécessité de réduire la complexité des données. En effet, certains films avaient des distributions d'acteurs extrêmement longues, rendant la colonne peu pratique à manipuler. En limitant le nombre d'acteurs, nous avons réduit la taille des données sans sacrifier l'information essentielle pour les analyses futures.

Cependant, ce processus n'a pas été sans difficulté. Dans certains cas, les listes d'acteurs étaient incomplètes ou mal formatées, ce qui a nécessité des ajustements manuels et l'utilisation de fonctions robustes pour gérer ces anomalies. La fonction d'application utilisée a été spécifiquement paramétrée pour être tolérante aux erreurs de format, garantissant ainsi la cohérence du résultat.

```
# Limiter les noms d'acteurs aux 4 premiers dans chaque cellule de la colonne
'actors_primaryName'

df['actors_primaryName'] = df['actors_primaryName'].apply(

      lambda x: ','.join(x.split(',')[:4]) if isinstance(x, str) else x,

      meta=('x', 'str')
```

2. Nettoyage des Personnages

Les informations relatives aux personnages ont été épurées afin de supprimer les caractères non pertinents (crochets, guillemets) et de limiter le nombre de personnages à quatre par film.

Ce nettoyage était crucial pour assurer une meilleure lisibilité des données, car les caractères superflus rendaient certaines analyses difficiles à mener, en particulier lors des opérations de filtrage ou d'exploration visuelle des données. En supprimant ces caractères, nous avons simplifié la colonne et amélioré sa cohérence.

Le principal défi de cette étape résidait dans la manipulation de chaînes de caractères très longues et souvent mal structurées. De plus, les différences de format entre les films, en raison de la diversité des sources de données, ont nécessité l'utilisation de méthodes robustes de nettoyage basées sur des expressions régulières.

```
# Nettoyer la colonne 'actors_characters' en supprimant les crochets et
guillemets

df['actors_characters'] = df['actors_characters'].str.replace(r'[\[\]" ]', '',
regex=True)



# Limiter chaque cellule à 4 personnages seulement

df['actors_characters'] = df['actors_characters'].apply(lambda x:
','.join(x.split(',')[:4]) if isinstance(x, str) else x)
```

3. Suppression des Colonnes Non Pertinentes

Certaines colonnes techniques, telles que **`date_rating`**, **`release_year`**, **`imdb_isAdult`**, etc., ont été supprimées afin de réduire la surcharge inutile et de simplifier le DataFrame.

Ces colonnes, bien qu'initialement incluses, ne contribuaient pas de manière significative à l'objectif principal de la recommandation de films. Par conséquent, leur suppression a permis de réduire la taille du DataFrame et d'optimiser les performances des analyses futures.

Il a également été nécessaire de vérifier que la suppression de ces colonnes n'affecterait pas d'autres aspects de l'analyse. Pour cela, des tests ont été menés sur des sous-ensembles de données afin de garantir que les informations clés étaient toujours présentes après cette étape.

```
# Supprimer les colonnes spécifiées

df = df.drop(columns=['date_rating', 'release_year', 'imdb_isAdult',
'imdb_tconst', 'imdb_runtimeMinutes'])
```

Dans cette partie, les filtres ont été appliqués de manière itérative afin de réduire le nombre de lignes tout en maximisant la qualité des films retenus, en explorant différents seuils et critères de filtrage.

1. Filtrage des Notes Utilisateurs

Ce premier filtre a été essentiel pour garantir que les recommandations seraient basées sur des films appréciés par les utilisateurs, en éliminant les titres ayant reçu des notes faibles qui auraient pu altérer la pertinence des recommandations.

```
# Suppression des lignes avec une note_utilisateur inférieure à 3.5

df = df[df['note_utilisateur'] >= 3.5]
```

2. Filtrage par Nombre de Votes IMDb

Une série de filtres itératifs a été appliquée pour ne conserver que les films les plus populaires :

- **50 000 votes** sur IMDb, puis **100 000 votes**, ensuite **250 000 votes**, et enfin **350 000 votes**. Ces seuils ont été choisis progressivement afin d'évaluer leur impact sur la quantité des données et de ne retenir que les films ayant fait l'objet d'un **fort consensus populaire**.

L'application progressive de ces filtres a permis de maintenir un équilibre entre la quantité des données et leur pertinence. En augmentant le seuil de votes à chaque étape, nous avons réduit le volume tout en augmentant la fiabilité des recommandations potentielles.

```
# Filtrer les films ayant au moins 50 000 votes sur IMDb

df = df[df['nombre_votes_imdb'] >= 50000]



# Filtrer les films ayant au moins 100 000 votes sur IMDb

df = df[df['nombre_votes_imdb'] >= 100000]



# Filtrer les films ayant au moins 250 000 votes sur IMDb

df = df[df['nombre_votes_imdb'] >= 250000]
```

```
# Filtrer les films ayant au moins 350 000 votes sur IMDb

df = df[df['nombre_votes_imdb'] >= 350000]
```

3. Filtrage par Note IMDb

Seuls les films ayant une **note moyenne IMDb ≥ 7** ont été conservés, garantissant ainsi que les recommandations incluent uniquement des films largement appréciés.

Ce filtre final a été mis en place pour garantir une qualité élevée des recommandations. En fixant un seuil de 7 pour la note moyenne IMDb, nous avons éliminé les films médiocres ou polarisants, assurant ainsi une expérience utilisateur optimale.

```
# Filtrer le DataFrame pour ne conserver que les films avec une note moyenne
IMDb de 7 ou plus

df = df[df['note_moyenne_imdb'] >= 7]
```

4. Limitation des Films Connus des Acteurs

Les **films connus des acteurs** ont été limités à **trois titres** par acteur afin de simplifier les données tout en maintenant l'essentiel de l'information utile pour la recommandation.

Ce choix visait à réduire la complexité des informations présentes, car chaque acteur pouvait être lié à un grand nombre de films, rendant la base de données inutilement volumineuse. En limitant le nombre de films connus à trois par acteur, nous avons conservé l'information essentielle permettant de contextualiser la carrière de chaque acteur tout en évitant les redondances.

```
# Limiter les films connus des acteurs à trois titres dans la colonne
'films_connus_acteurs'

df['films_connus_acteurs'] = df['films_connus_acteurs'].apply(

      lambda x: ','.join(x.split(',')[:3]) if isinstance(x, str) else x,

      meta=('films_connus_acteurs', 'object')

)
```

5. Ajustement des Partitions

Après chaque étape de filtrage, le **nombre de partitions** a été ajusté pour optimiser les performances et la gestion de la mémoire lors de la manipulation de données volumineuses avec **Dask**.

Le réajustement du nombre de partitions est une opération cruciale pour garantir une exécution fluide des différents traitements. En effet, à mesure que la taille du DataFrame diminuait, il devenait nécessaire de réduire le nombre de partitions pour limiter la surcharge en mémoire et améliorer la vitesse des calculs.

```
# Répartir le DataFrame en 10 partitions pour optimiser les performances

df = df.repartition(npartitions=10)  # Ajustez le nombre de partitions en
fonction de votre machine
```

Partie III. Filtrage par Contenu pour le Système de Recommandation

Cette section se concentre sur l'application du filtrage par contenu, une stratégie visant à limiter la taille de l'ensemble de films utilisés pour les calculs de similarité, afin de rendre le processus computationnel plus accessible avec des ressources informatiques limitées. Le filtrage par contenu est essentiel, en particulier lorsque l'on manipule des volumes de données massifs, car les calculs de similarité peuvent rapidement devenir impraticables sur une machine conventionnelle.

1. Justification du Choix de 100 000 Films

Le choix de **100 000 lignes** repose sur des contraintes de ressources matérielles. En effet, le calcul des similarités entre les films était devenu **inaccessible sur l'ordinateur utilisé**, compte tenu du volume initial des données, qui comportait plus de quatre millions de lignes. Pour atténuer cette problématique, il était nécessaire de réduire la taille du jeu de données afin de permettre un traitement faisable, tout en préservant la qualité de l'analyse.

Pour ce faire, nous avons extrait les **100 000 premières lignes** du DataFrame initial à l'aide de la fonction head(). Cette approche permet de travailler sur un sous-ensemble représentatif des films disponibles, tout en réduisant les ressources requises pour les calculs ultérieurs.

```
# Créer un nouveau DataFrame avec seulement les 100 000 premières lignes en
utilisant head()

df_short = df.head(100000)

# Afficher un aperçu pour vérifier
```

```
print(df_short)
```

2. Sauvegarde du Sous-Ensemble

Le sous-ensemble ainsi obtenu a ensuite été sauvegardé dans un fichier CSV pour une utilisation ultérieure, évitant ainsi de devoir répéter cette opération de filtrage. Cette étape garantit une plus grande efficacité au cours des prochaines phases de développement du système de recommandation.

```
# Enregistrer le DataFrame df_short en CSV pour une utilisation ultérieure

output_path = r'C:\Users\724me\OneDrive\Desktop\dossier ANA\df_short.csv'

df_short.to_csv(output_path, index=False)

print(f"Le DataFrame df_short a été enregistré avec succès dans {output_path}")
```

3. Analyse Statistique du Sous-Ensemble

Pour mieux appréhender les caractéristiques du sous-ensemble, des statistiques descriptives ont été générées. Ces statistiques offrent un aperçu des principales propriétés de l'ensemble de données réduit, notamment la distribution des notes attribuées par les utilisateurs, les moyennes des notes IMDb, ainsi que le nombre de votes reçus.

```
df_short.describe()
```

L'objectif de cette analyse est de s'assurer que le sous-ensemble sélectionné reste représentatif des films populaires et largement évalués. Ce filtrage par contenu simplifie considérablement la construction du système de recommandation, en réduisant le volume de données à traiter sans compromettre la qualité des recommandations qui pourront être générées.

Filtrage collaboratif avec Surprise

Partie I. Traitement de la Base de Données dans le Cadre du Système de Recommandation Collaboratif par Surprise

Dans cette partie, nous explorons la préparation de la base de données pour l'application d'une méthode de recommandation collaborative en utilisant la bibliothèque **Surprise**. Cette méthode nécessite un volume de données significatif pour être efficace, d'où l'utilisation d'un échantillon plus grand par rapport au filtrage par contenu réalisé précédemment.

1. Extraction de 1 000 000 de Lignes

Pour commencer, nous avons tenté d'extraire un sous-ensemble de **1 000 000 de lignes** de la base de données initiale à l'aide de la fonction head(). Cette étape vise à rendre les calculs de similarité entre utilisateurs plus robustes, en augmentant le nombre d'interactions utilisateurs-films disponibles pour les modèles de recommandation collaborative.

```
# Créer un nouveau DataFrame avec seulement les 1 000 000 premières lignes en utilisant head()

df_short_Million = df.head(1000000)

# Afficher un aperçu pour vérifier

print(df_short_Million)
```

Cependant, en raison du volume disponible après les filtres appliqués précédemment, un avertissement a été émis indiquant que le nombre de lignes disponibles était insuffisant pour atteindre l'objectif initial d'un million de lignes. Nous avons donc obtenu **407 610 lignes**, ce qui constitue néanmoins un échantillon quatre fois plus grand que celui utilisé pour le filtrage par contenu.

```
c:\Users\724me\anaconda3\Lib\site-packages\dask\dataframe\core.py:8393: UserWarning: Insufficient elements for `head`. 1000000 elements requested, only 407610 elements available. Try passing larger `npartitions` to `head`.

  warnings.warn(
```

Malgré cette limitation, l'échantillon obtenu est largement suffisant pour entamer le développement du système de recommandation collaboratif. En effet, disposer de **407 610 interactions** permet de capter des motifs et des corrélations significatives entre les utilisateurs, ce qui est essentiel pour la qualité des recommandations.

2. Sauvegarde de l'Échantillon

Pour assurer la réutilisation de cet échantillon lors des étapes suivantes, le DataFrame a été sauvegardé dans un fichier CSV. Cela permet de gagner du temps en évitant de refaire l'extraction à chaque itération de l'algorithme de recommandation.

```
# Enregistrer le DataFrame df_short en CSV

output_path = r'C:\Users\724me\OneDrive\Desktop\dossier
ANA\base_finale_filtree_description_million.csv'

df_short_Million.to_csv(output_path, index=False)

print(f"Le DataFrame df_short a été enregistré avec succès dans {output_path}")

print(type(df_short_Million))

<class 'pandas.core.frame.DataFrame'>
```

3. Analyse de l'Échantillon Obtenu

Le sous-ensemble final, composé de **407 610 lignes**, présente une taille bien plus conséquente que celle utilisée lors de la phase de filtrage par contenu (100 000 lignes). Cet échantillon plus grand est destiné à renforcer la robustesse du modèle collaboratif, permettant d'améliorer la capacité du système à trouver des similarités entre utilisateurs en fonction des films notés.

Voici un aperçu des premières lignes du sous-ensemble obtenu :

```
      id_utilisateur  id_film   id_imdb  note_utilisateur  note_moyenne_imdb
nombre_votes_imdb  genres                          ...

0           1.0         2.0   113497.0          3.5               7.1
384991.0    Adventure|Children|Fantasy

2           1.0    32.0   114746.0         3.5               8.0
654005.0    Mystery|Sci-Fi|Thriller

...

60711       13577.0     47.0   114369.0         5.0               8.6
1839918.0         Mystery|Thriller
```

Ce sous-ensemble contient des informations pertinentes, notamment les **notes des utilisateurs**, les **genres des films**, ainsi que le **nombre de votes** sur IMDb, qui seront exploités dans le cadre de la modélisation collaborative.

Dans cette section, nous analysons les résultats obtenus en appliquant une approche de filtrage collaboratif à l'aide de la bibliothèque **Surprise**. Cette méthode vise à implémenter un modèle de recommandation collaboratif fondé sur les interactions utilisateur-film, permettant ainsi de fournir des recommandations personnalisées basées sur les similarités dans les préférences des utilisateurs.

Partie II. Préparation des données

1. Chargement des Données Préparées

La première étape consiste à charger la base de données filtrée précédemment, composée de **407 610 entrées**, qui a été sauvegardée au format CSV. Cette base de données est utilisée dans le cadre de la modélisation collaborative, et il est essentiel de vérifier que le chargement des données s'effectue correctement pour garantir l'intégrité des analyses futures.

```
import pandas as pd



# Chemin du fichier CSV

file_path = r'C:\Users\724me\OneDrive\Desktop\dossier
ANA\base_finale_filtree_description_million.csv'



# Charger le DataFrame depuis le fichier CSV

df_short = pd.read_csv(file_path)



# Vérifier si le chargement s'est bien passé

print(df_short.head())
```

Après le chargement, un aperçu des premières lignes confirme que les données ont été chargées avec succès, incluant des informations cruciales telles que les **notes des utilisateurs**, les **genres des films**, ainsi que les **réalisateurs**.

2. Génération de Noms d'Utilisateurs Aléatoires

Pour rendre l'analyse plus accessible, des noms d'utilisateurs fictifs ont été générés à l'aide de la bibliothèque **Faker**. Cette anonymisation sert à diversifier les données tout en facilitant la présentation des résultats et l'interprétation des recommandations.

```
# Installer Faker pour la génération de noms aléatoires

!pip install faker

from faker import Faker

import random



# Initialiser Faker avec différentes langues pour diversifier les noms

fake_us = Faker('en_US')

fake_fr = Faker('fr_FR')

fake_latino = Faker('es_MX')

fake_german = Faker('de_DE')



# Générer des noms pour chaque id_utilisateur unique dans df_short

user_ids = df_short['id_utilisateur'].unique()

names = {}



# Associer chaque `id_utilisateur` unique à un nom aléatoire selon un style
différent

for user_id in user_ids:

      choice = random.choice(['us', 'fr', 'latino', 'german'])

      if choice == 'us':

      first_name = fake_us.first_name()

      last_name = fake_us.last_name()
```

```
        elif choice == 'fr':
        first_name = fake_fr.first_name()
        last_name = fake_fr.last_name()
        elif choice == 'latino':
        first_name = fake_latino.first_name()
        last_name = fake_latino.last_name()
        else:  # German names
        first_name = fake_german.first_name()
        last_name = fake_german.last_name()

        names[user_id] = f"{first_name} {last_name}"

# Ajouter la colonne `nom_utilisateur` en mappant les noms aux `id_utilisateur`
df_short['nom_utilisateur'] = df_short['id_utilisateur'].map(names)
```

3. Calcul du Score de Pertinence

Pour renforcer la qualité des recommandations, un **score de pertinence ajusté** a été calculé pour chaque film. Ce score prend en compte à la fois les **notes attribuées par les utilisateurs** et les **notes IMDb**, tout en pondérant l'influence du nombre de votes de manière à éviter un biais disproportionné envers les films les plus populaires.

```
import numpy as np

# Convertir la note utilisateur sur une échelle de 10
df_short['note_utilisateur_sur_10'] = df_short['note_utilisateur'] * 2

# Calculer le score de pertinence ajusté en diminuant l'influence du nombre de votes
df_short['pertinence_brute'] = (
```

```
      (0.4 * df_short['note_utilisateur_sur_10'] + 0.6 *
df_short['note_moyenne_imdb']) *

      (np.log10(df_short['nombre_votes_imdb']) ** (1 / 3))  # Racine cubique du
log pour une influence minimale

)



# Normaliser le score de pertinence sur une échelle de 0 à 100

min_pertinence = df_short['pertinence_brute'].min()

max_pertinence = df_short['pertinence_brute'].max()

df_short['score_pertinence'] = 100 * (df_short['pertinence_brute'] -
min_pertinence) / (max_pertinence - min_pertinence)
```

Ce score de pertinence normalisé permet de comparer les films de manière plus objective et constitue la base de l'élaboration de recommandations plus pertinentes pour les utilisateurs.

4. Optimisation des Paramètres de l'Algorithme de Recommandation

Pour la mise en place du système de recommandation, l'algorithme **SVD (Singular Value Decomposition)** de la bibliothèque **Surprise** a été utilisé. Une recherche de grille a été réalisée pour optimiser les hyperparamètres du modèle, tels que le nombre de facteurs latents (n_factors), le taux d'apprentissage (lr_all), et la régularisation (reg_all). Cette recherche est un processus itératif, parfois qualifié de **pérégrination** dans l'espace des hyperparamètres, qui vise à minimiser l'erreur de prédiction et à améliorer la qualité des recommandations.

```
from surprise import SVD

from surprise.model_selection import GridSearchCV, cross_validate



# Définir la grille d'hyperparamètres

param_grid = {

      'n_factors': [50, 100, 120],         # Dimensions latentes

      'n_epochs': [20, 30, 40],            # Nombre d'époques

      'lr_all': [0.003, 0.005, 0.01],      # Taux d'apprentissage

      'reg_all': [0.1, 0.2, 0.25]          # Régularisation
```

```
}

# Effectuer la recherche avec parallélisme
grid_search = GridSearchCV(
      SVD,
      param_grid,
      measures=['rmse', 'mae'],
      cv=3,
      n_jobs=-1  # Utilise tous les cœurs disponibles
)

# Lancer la recherche
grid_search.fit(dataset)

# Meilleurs paramètres
best_params = grid_search.best_params['rmse']
print("Meilleurs paramètres trouvés : ", best_params)

# Réentraînement avec les meilleurs paramètres
algo = SVD(**best_params)

# Construire l'ensemble d'entraînement complet
trainset = dataset.build_full_trainset()

# Entraîner le modèle
algo.fit(trainset)
```

```
print("Modèle entraîné avec les meilleurs paramètres.")
```

Les **meilleurs paramètres** identifiés par la recherche de grille ont été utilisés pour entraîner le modèle sur l'ensemble complet des données, garantissant ainsi que l'algorithme est optimisé pour minimiser l'erreur quadratique moyenne (**RMSE**).

5. Validation Croisée du Modèle Optimisé

Après l'entraînement, une validation croisée a été réalisée pour évaluer les performances du modèle en utilisant les **meilleurs paramètres** obtenus lors de la recherche. Cette validation croisée permet de vérifier la robustesse du modèle et de s'assurer qu'il est capable de fournir des recommandations précises.

```
# Validation croisée avec les meilleurs paramètres

results = cross_validate(algo, dataset, measures=['RMSE', 'MAE'], cv=3,
verbose=True)



# Afficher les résultats

print("Résultats finaux : ", results)
```

Les résultats de la validation croisée indiquent les valeurs moyennes des erreurs **RMSE** et **MAE** sur trois plis. Ces mesures permettent d'évaluer la qualité des prédictions, et les itérations successives ont permis d'affiner le modèle pour atteindre des valeurs d'erreur de plus en plus faibles, traduisant une amélioration progressive des performances.

Partie III. Génération de Recommandations Personnalisées pour les Utilisateurs

Dans cette section, nous poursuivons le processus en appliquant l'algorithme de filtrage collaboratif optimisé afin de générer des recommandations personnalisées pour les utilisateurs. Cette approche repose sur les résultats des étapes précédentes et illustre une progression logique et structurée dans la conception d'un système de recommandation sophistiqué. Elle démontre également la manière dont chaque étape a permis de renforcer la pertinence des suggestions pour une expérience utilisateur enrichie.

L'objectif de cette partie est de tirer pleinement parti des données préparées pour élaborer des recommandations spécifiques, en utilisant une combinaison de techniques de filtrage collaboratif, d'évaluation des préférences passées et de suggestions inattendues qui encouragent l'utilisateur à explorer de nouvelles œuvres. Cette approche multimodale garantit

un engagement soutenu et un enrichissement du catalogue de chaque utilisateur, en équilibrant entre les films qu'ils sont susceptibles d'apprécier et ceux qui les surprendront.

1. **Préparation des Données et Entraînement du Modèle**

Pour générer des recommandations personnalisées, nous avons d'abord rechargé les données filtrées et normalisées, puis entraîné notre modèle **SVD** optimisé en utilisant les meilleurs paramètres identifiés lors de l'étape de recherche de grille. Ces paramètres ont permis de minimiser les erreurs (RMSE et MAE), garantissant ainsi une meilleure précision des recommandations. Le modèle a été conçu pour être à la fois performant en termes de précision de prédiction et adaptable, permettant de capturer les préférences complexes des utilisateurs.

```
from surprise import Dataset, Reader, SVD

from termcolor import colored

import pandas as pd

import random



# Définir le lecteur pour la bibliothèque Surprise

reader = Reader(rating_scale=(data['score_pertinence'].min(),
data['score_pertinence'].max()))



# Charger l'ensemble de données entier pour l'entraînement

dataset = Dataset.load_from_df(data[['nom_utilisateur', 'titre_film',
'score_pertinence']], reader)



# Construire l'ensemble d'entraînement à partir de toutes les données

trainset = dataset.build_full_trainset()



# Initialiser l'algorithme SVD avec les meilleurs paramètres trouvés

best_params = {

      'n_factors': 120,
```

```
        'n_epochs': 20,
        'lr_all': 0.003,
        'reg_all': 0.85
}
algo = SVD(**best_params)


# Entraîner le modèle sur l'ensemble complet des données
algo.fit(trainset)
```

2. Génération des Recommandations Personnalisées

Avec le modèle entraîné, nous avons implémenté une fonction permettant de générer des recommandations pour chaque utilisateur. Cette fonction, get_recommendations_for_user(), prédit la pertinence de chaque film non encore noté par l'utilisateur et génère une liste des **10 meilleurs films** à recommander. L'algorithme repose sur une analyse des similarités entre les utilisateurs ainsi que sur les préférences spécifiques de chaque individu, assurant ainsi que les recommandations soient pertinentes et alignées avec les goûts de chaque utilisateur.

```
# Fonction pour générer des recommandations pour un utilisateur spécifique
def get_recommendations_for_user(user_name, algo, data, n=10):
        # Liste de tous les films uniques
        all_films = data['titre_film'].unique()


        # Films déjà notés par l'utilisateur
        rated_films = data[data['nom_utilisateur'] ==
user_name]['titre_film'].unique()


        # Films non encore notés par l'utilisateur
        films_to_predict = [film for film in all_films if film not in
rated_films]
```

```
      # Prédire le score de pertinence pour chaque film non encore noté

      predictions = [algo.predict(user_name, film) for film in
films_to_predict]



      # Trier les prédictions par score décroissant et sélectionner les top-N

      recommendations = sorted(predictions, key=lambda x: x.est, reverse=True)



      # Ajouter une vérification explicite pour éviter les doublons

      recommendations = [(pred.iid, pred.est) for pred in recommendations if
pred.iid not in rated_films][:n]



      return recommendations
```

3. **Suggestions Complémentaires et Diversification**

Pour enrichir davantage l'expérience utilisateur, nous avons également implémenté deux fonctionnalités additionnelles :

- **Obtenir les films les mieux notés par un utilisateur** (à l'aide de get_top_rated_films()). Cette fonctionnalité permet à l'utilisateur de visualiser ses préférences passées et de mieux comprendre les recommandations qui lui sont faites. Cela offre une transparence dans le système de recommandation et favorise la confiance des utilisateurs dans les suggestions fournies.
- **Suggestions aléatoires ("Hors des sentiers battus")** (à l'aide de get_random_discovery_films()). Ces suggestions visent à offrir à l'utilisateur la possibilité de découvrir des films qu'il n'aurait peut-être pas envisagé de regarder, diversifiant ainsi son catalogue personnel et ajoutant un facteur de surprise qui peut mener à des découvertes intéressantes. Cette diversité est cruciale pour maintenir l'intérêt et favoriser une exploration au-delà des zones de confort de l'utilisateur.

```
# Fonction pour obtenir les films les mieux notés par un utilisateur

def get_top_rated_films(user_name, data, top_n=10):

      # Obtenir les films déjà notés par l'utilisateur et trier par score
décroissant

      user_ratings = data[data['nom_utilisateur'] == user_name]
```

```
      if user_ratings.empty:

      print(colored(f"Aucun film noté trouvé pour l'utilisateur {user_name}.",
"red", attrs=["bold"]))

      return []

      top_rated = user_ratings.sort_values(by='score_pertinence',
ascending=False).head(top_n)

      return list(zip(top_rated['titre_film'], top_rated['score_pertinence']))



# Fonction pour générer des suggestions aléatoires (hors sentiers battus)

def get_random_discovery_films(data, n=5):

      all_films = data['titre_film'].unique()

      random_films = random.sample(list(all_films), min(n, len(all_films)))

      return random_films
```

4. **Présentation des Résultats à l'Utilisateur**

Pour améliorer l'interaction avec les utilisateurs, une fonction afficher_recommandations() a été créée. Celle-ci permet de présenter non seulement les recommandations personnalisées, mais aussi les films préférés de l'utilisateur, ainsi qu'une sélection de suggestions "hors des sentiers battus". Cette approche vise à enrichir l'expérience utilisateur en proposant des options variées, basées sur les préférences existantes, les recommandations générées par le modèle, et des suggestions exploratoires.

L'idée derrière cette présentation segmentée est de fournir une structure claire aux utilisateurs, leur permettant de comprendre les différentes catégories de suggestions disponibles :

- **Les films préférés** sont ceux que l'utilisateur a déjà notés très positivement et qu'il est susceptible de revoir.
- **Les recommandations** sont des films alignés sur leurs préférences passées, utilisant l'algorithme SVD pour identifier les titres les plus pertinents.
- **Les suggestions "Hors des sentiers battus"** visent à surprendre l'utilisateur en lui offrant des films auxquels il n'aurait peut-être pas pensé spontanément.

```
# Fonction pour afficher les recommandations et les films préférés

def afficher_recommandations(nom_utilisateur, algo, data, n=10):
```

```
        # Obtenir les 10 films préférés de l'utilisateur

        films_aimes = get_top_rated_films(nom_utilisateur, data, top_n=10)


        # Obtenir les 10 recommandations pour l'utilisateur

        recommandations = get_recommendations_for_user(nom_utilisateur, algo,
data, n)


        # Obtenir les films "Hors des sentiers battus"

        random_discovery = get_random_discovery_films(data, n=5)


        # Filtrer pour éviter les doublons entre les films aimés et les
recommandations

        films_aimes_set = set([film[0] for film in films_aimes])

        recommandations_filtrees = [(film, score) for film, score in
recommandations if film not in films_aimes_set][:n]


        # Afficher les films préférés de l'utilisateur

        print(colored(f"\n💖 Top 10 des films préférés de {nom_utilisateur} :\n",
'magenta', attrs=['bold']))

        for i, (film, score) in enumerate(films_aimes, 1):

        etoiles = "⭐" * (int(score) // 20)

        print(f"{i}. {film}")

        print(f"   Score donné : {score:.2f} {etoiles}\n")


        # Afficher les recommandations

        print(colored(f"\n🎬 Top 10 des recommandations pour {nom_utilisateur}
:\n", 'cyan', attrs=['bold']))

        for i, (film, score) in enumerate(recommandations_filtrees, 1):
```

```
        etoiles = "⭐" * (int(score) // 20)

        couleur = 'green' if score >= 75 else 'yellow' if score >= 50 else 'red'

        print(colored(f"{i}. {film}", couleur, attrs=['bold']))

        print(f"   Score de pertinence prédit : {score:.2f} {etoiles}\n")


        # Afficher les films "Hors des sentiers battus"

        print(colored(f"\n🌟 Top 5 Hors des sentiers battus :\n", 'blue',
attrs=['bold']))

        for i, film in enumerate(random_discovery, 1):

        print(f"{i}. {film}")

 # Exemple d'utilisation

nom_utilisateur = "Siegbert Gunpf"  # Remplacez par un utilisateur valide

afficher_recommandations(nom_utilisateur, algo, data)
```

Partie IV. Évaluation des Performances du Modèle

Exemple d'utilisation et Présentation des Résultats

Afin d'illustrer la pertinence et l'efficacité du système de recommandation mis en œuvre, nous avons pris un utilisateur fictif, **Jillian Cervantes**, et généré des recommandations personnalisées à partir du modèle préalablement entraîné. Cette illustration fournit une démonstration pratique du fonctionnement du système, mettant en lumière sa capacité à fournir des suggestions diversifiées et pertinentes, en adéquation avec les préférences spécifiques de chaque utilisateur. Les résultats obtenus montrent l'efficacité du modèle pour adapter les recommandations à des goûts individuels tout en encourageant la découverte de nouveaux contenus. Voici les résultats présentés pour cet utilisateur particulier :

```
# Exemple d'utilisation

nom_utilisateur = "Jillian Cervantes"  # Remplacez par un utilisateur valide

afficher_recommandations(nom_utilisateur, algo, data)
```

💖 **Top 10 des films préférés de Jillian Cervantes :**

1. Star Wars: Episode V - The Empire Strikes Back Score donné : 82.38 ⭐⭐⭐⭐
2. Saving Private Ryan Score donné : 80.96 ⭐⭐⭐⭐
3. Star Wars: Episode IV - A New Hope Score donné : 80.71 ⭐⭐⭐⭐
4. Gladiator Score donné : 79.56 ⭐⭐⭐
5. Terminator 2: Judgment Day Score donné : 79.20 ⭐⭐⭐
6. Star Wars: Episode VI - Return of the Jedi Score donné : 72.83 ⭐⭐⭐
7. Casablanca Score donné : 72.29 ⭐⭐⭐
8. Jurassic Park Score donné : 70.50 ⭐⭐⭐
9. Sixth Sense, The Score donné : 70.39 ⭐⭐⭐
10. Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb Score donné : 67.18 ⭐⭐⭐

Ces films illustrent les préférences de Jillian Cervantes, notamment son intérêt pour des classiques de la science-fiction, de l'aventure, et des drames de guerre. Ces informations permettent au modèle de recommander des contenus similaires, tout en cherchant à surprendre l'utilisateur en l'exposant à des contenus qu'il n'a pas encore explorés.

🎬 **Top 10 des recommandations pour Jillian Cervantes :**

1. Shawshank Redemption, The Score de pertinence prédit : 87.75 ⭐⭐⭐⭐
2. Godfather, The Score de pertinence prédit : 82.66 ⭐⭐⭐⭐
3. Schindler's List Score de pertinence prédit : 79.45 ⭐⭐⭐
4. Band of Brothers Score de pertinence prédit : 78.81 ⭐⭐⭐
5. Matrix, The Score de pertinence prédit : 78.07 ⭐⭐⭐
6. Dark Knight, The Score de pertinence prédit : 73.88 ⭐⭐⭐
7. Lord of the Rings: The Return of the King, The Score de pertinence prédit : 72.55 ⭐⭐⭐
8. 12 Angry Men Score de pertinence prédit : 72.03 ⭐⭐⭐
9. Pulp Fiction Score de pertinence prédit : 70.89 ⭐⭐⭐
10. Usual Suspects, The Score de pertinence prédit : 70.00 ⭐⭐⭐

Les recommandations générées sont le fruit de l'analyse collaborative des préférences de Jillian Cervantes, s'appuyant sur des similarités avec d'autres utilisateurs partageant des goûts similaires. La diversité des films recommandés, incluant des drames profonds, des thrillers et des classiques du cinéma, est conçue pour à la fois satisfaire les préférences établies de Jillian tout en l'exposant à des œuvres qui pourraient élargir ses intérêts cinématographiques.

🌟 Top 5 Hors des sentiers battus :

1. Transformers
2. Stand by Me
3. Hotel Rwanda
4. Fury
5. Dallas Buyers Club

La section "Hors des sentiers battus" offre une série de suggestions moins attendues, visant à encourager une exploration active de genres ou de contenus que l'utilisateur n'aurait peut-être pas envisagés spontanément. Ces recommandations ajoutent un élément de surprise et sont conçues pour équilibrer les suggestions plus familières, incitant ainsi l'utilisateur à découvrir de nouveaux horizons cinématographiques. Cette approche mixte favorise à la fois le confort des choix familiers et la stimulation de la curiosité.

Évaluation de la Diversité des Utilisateurs

Outre la personnalisation des recommandations, une autre évaluation a porté sur la diversité et l'étendue des utilisateurs impliqués dans ce système. Nous avons calculé le nombre total de personnes uniques présentes dans le jeu de données pour mieux comprendre l'ampleur de l'audience couverte :

```
# Compter le nombre total de personnes uniques dans la colonne
'nom_utilisateur'

nombre_utilisateurs_uniques = data['nom_utilisateur'].nunique()

print("Nombre total de personnes différentes :", nombre_utilisateurs_uniques)
```

Nombre total de personnes différentes : 13193

Ce résultat démontre l'étendue de la base d'utilisateurs considérée dans le cadre du système de recommandation. Une diversité aussi importante d'utilisateurs permet au modèle de capter une large variété de préférences, rendant ainsi les recommandations générées plus pertinentes et applicables à un public diversifié. Cette diversité est cruciale pour s'assurer que le système de recommandation soit adaptable et généralisable, offrant des suggestions de qualité à des utilisateurs aux goûts et attentes variés.

Évaluation des Performances du Modèle

Dans cette section, nous nous concentrons sur l'évaluation des performances du modèle de recommandation optimisé. L'évaluation des performances est une composante essentielle pour valider la fiabilité et la précision des recommandations proposées. Pour ce faire, nous avons recours aux métriques standard telles que le **Root Mean Squared Error (RMSE)** et le **Mean Absolute Error (MAE)**, qui sont largement utilisés dans le domaine pour mesurer la qualité de la prédiction des scores d'utilisateurs.

Afin de garantir que les performances du modèle ne soient pas simplement le résultat d'un sur-apprentissage sur un sous-ensemble des données, nous avons utilisé une méthode de validation croisée à trois plis (cross_validate()). Cette méthode permet de vérifier la robustesse des résultats et de s'assurer que le modèle est bien généralisable, c'est-à-dire qu'il fonctionne de manière cohérente sur différents segments du jeu de données.

```
from surprise.model_selection import cross_validate

# Validation croisée avec les meilleurs paramètres

results = cross_validate(algo, dataset, measures=['RMSE', 'MAE'], cv=3,
verbose=True)

# Afficher les résultats

print("Résultats finaux : ", results)
```

Les résultats de cette validation croisée sont les suivants :

```
Meilleurs paramètres trouvés :  {'n_factors': 120, 'n_epochs': 20, 'lr_all':
0.003, 'reg_all': 0.85}

Modèle entraîné avec les meilleurs paramètres.

Evaluating RMSE, MAE of algorithm SVD on 3 split(s).


                  Fold 1  Fold 2  Fold 3  Mean   Std

RMSE (testset)    12.8164 12.7830 12.7607 12.7867 0.0229

MAE (testset)     10.7737 10.7522 10.7344 10.7534 0.0160

Fit time          2.63  2.68    2.69 2.66  0.02

Test time         0.85    0.87 0.89  0.87    0.02
```

```
Résultats finaux :  {'test_rmse': array([12.81637425, 12.78301723,
12.76074886]), 'test_mae': array([10.77366958, 10.75216608, 10.73441529]),
'fit_time': (2.6305088996887207, 2.6763250827789307, 2.686274290084839),
'test_time': (0.8496055603027344, 0.865408182144165, 0.8865530490875244)}
```

Les métriques obtenues indiquent que les valeurs de **RMSE** et **MAE** sont acceptables pour un système de recommandation de films. Bien que les erreurs puissent sembler relativement élevées, il est important de noter qu'un système de recommandation ne vise pas nécessairement à prédire chaque préférence avec une exactitude parfaite. Au contraire, une certaine marge d'erreur est souhaitable dans un contexte de recommandation culturelle, car elle favorise l'introduction de suggestions surprenantes qui peuvent enrichir l'expérience de l'utilisateur. En somme, le modèle est capable de maintenir un équilibre satisfaisant entre la prédictibilité et l'exploration.

Cette évaluation des performances par validation croisée permet également de mieux ajuster les paramètres du modèle afin de maximiser sa généralisabilité. Trouver un compromis entre la précision des recommandations et leur capacité à offrir des options diversifiées est essentiel pour maintenir l'engagement des utilisateurs et répondre à une grande variété de préférences individuelles.

Conclusion

Plusieurs algorithmes ont été explorés, notamment la factorisation matricielle et le KNN (K-Nearest Neighbors). Le KNN a été testé pour évaluer les similarités directes entre utilisateurs ou films. Les réseaux de neurones ont été envisagés pour leur potentiel à modéliser des relations non linéaires plus complexes, mais leur coût computationnel et leur performance n'ont pas justifié leur utilisation par rapport aux méthodes plus simples.

Modèle retenu

Nous avons décidé de retenir le **modèle de filtrage collaboratif SVD (Singular Value Decomposition)** utilisant la bibliothèque Python **Surprise**, car il surpasse les approches classiques (User-Based et Item-Based) qui reposent généralement sur des calculs de similarité (cosinus, Pearson, etc.) appliqués directement aux données des utilisateurs ou des items.

Surprise, en particulier avec le modèle SVD, offre plusieurs avantages significatifs :

1. **Algorithmes avancés** : Contrairement aux simples approches User-Based et Item-Based, qui comparent directement les utilisateurs ou les items en fonction de leurs évaluations, les algorithmes comme SVD explorent les **relations latentes** présentes dans les données. Ces relations capturent des informations implicites sur les

préférences des utilisateurs et les caractéristiques des items, permettant une compréhension plus profonde des interactions.
2. **Gestion améliorée des données manquantes** : Les méthodes implémentées dans Surprise, et notamment le SVD, sont adaptées aux matrices utilisateur-item qui sont généralement très creuses (beaucoup de valeurs manquantes). En effet, ces modèles n'exigent pas que toutes les interactions soient renseignées et sont capables de fonctionner efficacement même avec des données partielles.
3. **Performances accrues grâce à la factorisation matricielle** : Le SVD décompose la matrice utilisateur-item en plusieurs facteurs latents représentant les préférences des utilisateurs et les attributs des items. Cette décomposition réduit la dimensionnalité des données tout en conservant leur structure sous-jacente. En conséquence, le SVD est souvent plus précis que les approches traditionnelles de filtrage collaboratif, qui ne tiennent pas compte de ces dimensions latentes.

Justifications techniques supplémentaires :

- **SVD** est une méthode de **factorisation matricielle**, qui est aujourd'hui une pierre angulaire des systèmes de recommandation modernes.
- Contrairement aux méthodes de similarité directe (comme celles utilisées dans User-Based ou Item-Based), SVD prédit une note en utilisant les facteurs latents plutôt qu'en calculant explicitement les similitudes.
- Surprise comprend des méthodes et fonctions qui permettent de mesurer la performance du modèle facilement
- Surprise comprend des méthodes et fonctions qui permettent d’optimiser les hyperparamètres facilement

Optimisation de paramètres

Une optimisation des hyperparamètres a été réalisée en utilisant Grid Search et la validation croisée. Ces techniques ont permis de déterminer les paramètres optimaux, tels que le taux d'apprentissage, le nombre de facteurs latents et le nombre d'époques, afin de maximiser la performance du modèle. La validation croisée a été essentielle pour garantir la robustesse du modèle et éviter le surapprentissage.

Modèles avancés

Nous n'avons pas testé de modèles avancés comme le Bagging, le Boosting ou des techniques de Deep Learning. Cela s'explique principalement par des contraintes de temps liées à la préparation des données et à la modélisation déjà réalisées, ainsi que par des limitations matérielles (hardware), comme la mémoire disponible de nos ordinateurs. Ces facteurs ont

considérablement affecté le temps disponible pour explorer ces approches, rendant leur implémentation irréalisable dans le cadre du projet.

Interprétation des résultats

Analyse des erreurs et améliorations du dataset et modèle

Notre stratégie a plutôt consisté à anticiper les erreurs potentielles et éviter les valeurs aberrantes. Cette analyse s'est articulée autour des points suivants :

Certaines erreurs auraient potentiellement pu être introduites par la rareté des données pour certains films ou par des biais de popularité. Par exemple, les films très populaires avaient tendance à être sur-représentés, tandis que les films moins connus mais bien notés étaient souvent sous-estimés :

Filtrage des Données pour Améliorer la Pertinence

Les filtres ont été appliqués de manière itérative afin de réduire le nombre de lignes tout en maximisant la qualité des films retenus, en explorant différents seuils et critères de filtrage.

Filtrage des Notes Utilisateurs

Nous avons commencé par filtrer les **films ayant une note utilisateur inférieure à 3,5**.
Ce premier filtre a été essentiel pour garantir que les recommandations seraient basées sur des films appréciés par les utilisateurs, en éliminant les titres ayant reçu des notes faibles ou aberrantes qui auraient pu altérer la pertinence des recommandations.

```
# Suppression des lignes avec une note_utilisateur inférieure à 3.5

df = df[df['note_utilisateur'] >= 3.5]
```

Filtrage par Nombre de Votes IMDb

Une série de filtres itératifs a été appliquée pour ne conserver que les films les plus populaires :

50 000 votes sur IMDb, puis **100 000 votes**, ensuite **250 000 votes**, et enfin **350 000 votes**.

Ces seuils ont été choisis progressivement afin d'évaluer leur impact sur la quantité des données et de ne retenir que les films ayant fait l'objet d'un **fort consensus populaire**.

L'application progressive de ces filtres a permis de maintenir un équilibre entre la quantité des données et leur pertinence.

En augmentant le seuil de votes à chaque étape, nous avons réduit le volume tout en augmentant la fiabilité des recommandations potentielles.

```
# Filtrer les films ayant au moins 50 000 votes sur IMDb

df = df[df['nombre_votes_imdb'] >= 50000]

# Filtrer les films ayant au moins 100 000 votes sur IMDb

df = df[df['nombre_votes_imdb'] >= 100000]

# Filtrer les films ayant au moins 250 000 votes sur IMDb

df = df[df['nombre_votes_imdb'] >= 250000]

# Filtrer les films ayant au moins 350 000 votes sur IMDb

df = df[df['nombre_votes_imdb'] >= 350000]
```

Filtrage par Note IMDb

Seuls les films ayant une **note moyenne IMDb ≥ 7** ont été conservés, garantissant ainsi que les recommandations incluent uniquement des films largement appréciés.

Ce filtre final a été mis en place pour garantir une qualité élevée des recommandations. En fixant un seuil de 7 pour la note moyenne IMDb, nous avons éliminé les films médiocres ou polarisants, assurant ainsi une expérience utilisateur optimale.

```
# Filtrer le DataFrame pour ne conserver que les films avec une note moyenne
IMDb de 7 ou plus

df = df[df['note_moyenne_imdb'] >= 7]
```

Au final, nous avons obtenu un dataset de **407 610 lignes.** Ce dernier sera utilisée dans le cadre du filtrage collaboratif.

Création d’un score de pertinence brute et net

Par suite, nous avons notamment créé le score de pertinence ci dessous :

```
# Convertir la note utilisateur sur une échelle de 10
```

```
df_short['note_utilisateur_sur_10'] = df_short['note_utilisateur'] * 2

# Calculer le score de pertinence ajusté en diminuant l'influence du nombre de
votes

df_short['pertinence_brute'] = (

    (0.4 * df_short['note_utilisateur_sur_10'] + 0.6 *
df_short['note_moyenne_imdb']) *

    (np.log10(df_short['nombre_votes_imdb']) ** (1 / 3))  # Racine cubique du
log pour une influence minimale

)

# Normaliser le score de pertinence sur une échelle de 0 à 100

min_pertinence = df_short['pertinence_brute'].min()

max_pertinence = df_short['pertinence_brute'].max()

df_short['score_pertinence'] = 100 * (df_short['pertinence_brute'] -
min_pertinence) / (max_pertinence - min_pertinence)
```

Cette formule calcule un **score de pertinence** pour un film en prenant en compte plusieurs facteurs, tout en réduisant l'influence d'un grand nombre de votes pour éviter des biais.

- `df_short['pertinence_brute']` : C'est le score de pertinence final qu'on calcule.
- `note_utilisateur_sur_10` : La note donnée par les utilisateurs sur 10.
- `note_moyenne_imdb` : La note moyenne globale issue d'IMDb.
- `nombre_votes_imdb` : Le nombre total de votes enregistrés pour un film sur IMDb.

Moyenne pondérée des notes

$$0.4 \times \text{note utilisateur} + 0.6 \times \text{note moyenne IMDb}$$

Cela signifie que la note moyenne IMDb est jugée plus importante (60%) que la note individuelle des utilisateurs (40%) pour déterminer la pertinence brute.

Réduction de l'impact du nombre de votes

$$(\log_{10}(\text{nombre\_votes\_imdb}))^{1/3}$$

Formule global :

(0.4×note utilisateur+0.6×note moyenne IMDb)*(log10(nombre_votes_imdb))**(⅓)

Le score final est obtenu en divisant la moyenne pondérée des notes par cette valeur réduite du nombre de votes. Cela garantit :

- Une bonne balance entre **qualité (note)** et **quantité (nombre de votes)**.
- Une protection contre l'effet des films avec beaucoup de votes mais une qualité douteuse.

Techniques d'interprétabilité

Dans le cadre de ce projet, nous avons créé un score de pertinence, utilisé pour classer les meilleurs films les uns par rapport aux autres.

Le score de pertinence comme métrique d'interprétation utilisé pour classer les films offre une source numérique directement interprétable :

- Il permet de comparer les films entre eux.
- Il donne une mesure quantitative de la "proximité" entre les préférences d'un utilisateur et les caractéristiques des films.

Amélioration significative

L'utilisation d'un score de pertinence a permis non seulement de consolider de manière équitable les différentes sources de notation des films, mais également de réduire l'impact disproportionné du nombre de votes provenant d'IMDB. Par ailleurs, une fois normalisé, ce score a fourni un classement suffisamment discriminant pour établir une liste de recommandations personnalisées par utilisateur, répondant ainsi à l'objectif principal de notre projet.

Étape 3 : rapport final

Difficultés rencontrées lors du projet

Taille des bases de données

La gestion des bases de données représentait un défi majeur en raison de leur taille conséquente. Nous avons opté pour l'utilisation de Dask afin de gérer ces volumes, mais malgré les avantages qu'offre cette bibliothèque, elle s'est révélée insuffisante pour traiter efficacement les données à la hauteur de nos attentes.

Limitations logicielles et matérielles

Les performances limitées des logiciels et des équipements informatiques à notre disposition ont représenté un frein majeur. Ces limitations étaient particulièrement évidentes lorsque nous avons tenté de traiter les bases de données pour tester le Filtrage basé sur le contenu (Content-Based Filtering). Nous souhaitions exploiter les informations d'IMDb, telles que les acteurs, les réalisateurs, les films associés à ces professionnels, et bien d'autres. Cependant, même avec ces variables essentielles, le traitement des données s'est avéré extrêmement complexe en raison de la mémoire RAM insuffisante.

Pour contourner ces contraintes, nous avons été contraints de supprimer certaines informations, ce qui a réduit la richesse et la précision de notre modèle. Par ailleurs, nous aurions souhaité inclure des données supplémentaires, comme les langues ou les régions associées aux films, mais cette ambition n'a pas pu être réalisée en raison des limites matérielles. Ainsi, même les informations de base que nous considérions indispensables étaient déjà difficiles à traiter dans le cadre des ressources disponibles.

Contraintes de temps

Le temps restreint dédié au projet a également été un obstacle important. En parallèle du projet, nous devions suivre des cours, étudier et préparer des examens, ce qui réduisait notre capacité à nous investir pleinement. Cette contrainte temporelle nous a également empêchés de mieux planifier certaines étapes clés, ce qui aurait pu nous permettre de les aborder de manière plus efficace et sereine.

Planification des cours et apprentissage progressif

L'apprentissage progressif des modules tout au long du projet a rendu difficile une planification anticipée efficace. Ce manque de capacité de prévoyance a limité notre capacité à identifier et à mettre en œuvre des solutions optimales dès le départ. Avec plus de temps et une vision claire des étapes à venir, nous aurions pu affiner notre méthodologie et obtenir de meilleurs résultats.

De plus, certains modules sont arrivés un peu tard par rapport à l'avancement du projet, comme ceux portant sur la réduction de dimensions et le text mining. Ces sujets, bien que pertinents, auraient pu être mieux exploités si nous avions eu accès à ces connaissances plus tôt. Nous avons pris des décisions qui se sont heureusement avérées pertinentes, mais une compréhension préalable de ces concepts aurait probablement permis des prises de décision plus rapides et plus éclairées.

Streamlit

Streamlit présente plusieurs limitations :

1. **Limitation de taille** : il est impossible d'inclure des bases de données dépassant 200 Mb.
2. **Incompatibilité avec certaines bibliothèques** : Certaines bibliothèques ne sont pas compatibles, notamment *Surprise*. Cela a nécessité des recherches supplémentaires pour garantir son intégration et la présentation correcte des résultats. Il est important de souligner que cette bibliothèque est essentielle pour notre projet, car le modèle retenu repose sur son utilisation
3. **Adaptation du code Python initial** : le code Python doit être modifié pour être compatible avec Streamlit, ce qui peut nécessiter des ajustements significatifs.

Bilan

Contributions principales dans l'atteinte des objectifs du projet

Ana

Elle a joué un rôle dans l'atteinte des objectifs du projet en assurant l'organisation et **le traitement des données**, une étape fondamentale pour la réussite des modélisations. Elle a utilisé ses compétences en **exploration des données et en statistiques descriptives** pour structurer et analyser les bases nécessaires à la mise en œuvre des modèles.

Lors de l'élaboration des filtrages collaboratifs basés sur l'utilisateur et sur les éléments, Ana a pris en compte les variables principales nécessaires à ces modèles (UserId, MovieId et Rating). Elle a **préparé les bases de données** MovieLens, enrichies avec les évaluations (ratings) provenant d'IMDb. Parallèlement, elle a mené une analyse approfondie pour **réduire la taille des données** tout en préservant leur représentativité, ce qui a permis d'exécuter les modèles malgré les contraintes de ressources. Cette optimisation a été cruciale pour garantir que les modélisations puissent être réalisées de manière efficace.

En parallèle, Ana a préparé une base de données enrichie contenant des informations complémentaires sur les films (réalisateurs, acteurs, films associés), ouvrant ainsi la voie à l'expérimentation avec le Filtrage basé sur le Contenu. Cette démarche a contribué à explorer d'autres approches potentielles pour améliorer les recommandations, étendant ainsi les possibilités d'analyse pour le projet.

Enfin, Ana a participé activement à la **prise de décisions en équipe** concernant les modélisations et a soutenu la rédaction des résultats finaux. Sa contribution principale réside donc dans sa capacité à structurer les données, à permettre la mise en œuvre des modèles collaboratifs et à ouvrir de nouvelles perspectives avec des approches complémentaires, tout en soutenant les analyses finales nécessaires à l'atteinte des objectifs du projet.

Ariel

Ariel a concentré sa contribution sur le test et l'évaluation de plusieurs modélisations, notamment :

- **Filtrage Collaboratif : Approche modèle** (Surprise avec SVD)
- **Filtrage basé sur le contenu**

Ces modélisations ont permis de comparer leurs performances et leur pertinence par rapport à d'autres approches testées. Finalement, le modèle **Filtrage Collaboratif : Approche modèle (Surprise avec SVD)** a été retenu comme la solution principale pour notre étude.

Dans le cadre de ce modèle, Ariel a créé un score de pertinence innovant, qui a permis :

1. De consolider de manière équitable les différentes sources de notation des films.
2. De réduire l'impact disproportionné du nombre de votes provenant d'IMDb, garantissant ainsi une évaluation plus équilibrée des données.

Par ailleurs, Ariel a également mis en place l'interface **Streamlit** pour le projet, fournissant un outil pratique et interactif pour visualiser les résultats et soutenir les démonstrations.

Enfin, Ariel a participé activement aux prises de décision concernant les modélisations et leur mise en œuvre, jouant un rôle clé dans l'orientation finale du projet.

Charles

Depuis son arrivée le 31 octobre, Charles a pris le temps de s'approprier les spécificités et le fonctionnement du projet. Il s'est activement impliqué dans la mise en place de l'application **Streamlit** aux côtés d'Ariel, contribuant ainsi à la structuration et à la présentation des résultats.

De plus, il a proposé des idées pour améliorer le projet, renforçant sa qualité et sa pertinence. Charles a également participé de manière active aux décisions stratégiques concernant l'orientation globale du projet, démontrant un esprit collaboratif et une vision orientée vers les objectifs.

Lam

Ses principales contributions ont été le soutien de l'équipe dans la vision, l'organisation et la réalisation des tâches clés du projet, ainsi que la mise en œuvre, le test et l'évaluation des premières modélisations. Il a veillé à ce que les formats demandés pour les livrables soient respectés et pris en compte, tout en rédigeant les différents rapports nécessaires à chaque étape du projet libérant un temps précieux pour les tâches les plus chronophages. Il a aussi été responsable du maintien du projet sur GitHub.

En termes techniques, il a mis en place plusieurs modélisations, notamment :

- **Filtrage Collaboratif : approche mémoire**
  - **User-based**
  - **Item-based**
- **Filtrage Collaboratif : approche modèle**
  - **Item-based + SVD**

Ces modélisations ont permis d'obtenir de premières recommandations de films et d'établir un benchmark pour pouvoir ensuite comparer les performances des différents modèles et leur pertinence, contribuant ainsi à la sélection des solutions les plus adaptées aux objectifs du projet.

En plus des aspects techniques, il a également participé à la vision stratégique du projet et aux prises de décision concernant le choix et la priorisation des différentes tâches à toutes les étapes du projet. Son rôle a été de collaborer avec l'équipe pour évaluer les avancées, ajuster les priorités, et garantir que les résultats répondent aux attentes initiales tout en respectant les contraintes et les délais.

Modifications apportées au modèle depuis la dernière itération

Nous n'avons pas modifié les modèles de manière significative depuis la dernière itération du rapport, principalement en raison du manque de temps. Nous avons quelques améliorations en tête d'ici la soutenance, mais à ce stade, nous ne pouvons pas garantir qu'elles seront finalisées à temps. Si tel est le cas, nous intégrerons les modifications apportées et leur impact sur les résultats lors de notre soutenance et dans une version finale de ce rapport (ainsi que sur le repo GitHub).

En revanche, nous avons réussi à aligner les modèles lors de la phase Streamlit, c'est-à-dire que nous avons unifié la base de données et la cible (score de pertinence sur 100) pour tous les modèles. Nous avons également réussi à calculer le RMSE et le MAE pour tous les modèles de filtrage collaboratif, ce qui n'était pas le cas lors de la remise précédente de notre rapport.

Comme mentionné précédemment dans la section sur les contraintes et difficultés rencontrées lors du projet, mener simultanément la formation et le projet a considérablement réduit le temps disponible pour tester et ajuster les modélisations. Bien que nous aurions souhaité explorer et tester d'autres modèles pour enrichir nos résultats, les contraintes temporelles ne nous ont pas permis de le faire.

Résultats obtenus

Nous avons testé cinq approches:

- **Filtrage Collaboratif : approche mémoire**
  - **User-based**
  - **Item-based**
- **Filtrage Collaboratif : approche modèle**
  - **Item-based + SVD**
  - **Surprise (SVD)**
- **Filtrage basé sur le contenu**

Les performances ont été évaluées à l'aide des métriques RMSE et MAE :

Voici le benchmark avant réalignement de la base de données et l'utilisation unifiée du score de pertinence:

	Modèle	k	RMSE	MAE	Commentaires
Filtrage Collaboratif : approche mémoire	User-based	k=3	3.27 / 5	2.85 / 5	Pour k=3 (3 plus proches voisins utilisateurs)
		k=7	3.18 / 5	2.82 / 5	A la recherche d'optimiser la métrique MAE
		k=19	3.15 / 5	2.84 / 5	Suite à lancer une GridSearch, afin de trouver la valeur de k optimisant la RMSE
	Item-based	k =19	-	-	Nous avons rencontré des difficultés pour obtenir les métriques, le temps de calcul pour une itération (k=3) dépassant 24 heures. Le calcul a été interrompu avant son achèvement.
Filtrage Collaboratif : approche modèle	Item-based +SVD	k=19	-	-	Nous avons rencontré des difficultés pour obtenir les métriques, car le modèle, basé sur le vecteur des films, exige un temps de calcul important. Cela s'observe également dans le modèle basé sur les items (Item-based), qui présente des contraintes similaires.
	Surprise (SVD)		12.78 / 100	10.75 /100	Métriques basé sur 100
Filtrage basé sur le contenu			-	-	Aucune métrique spécifique n'a été identifiée pour évaluer la similarité textuelle, car il n'y a pas de référence pour évaluer la similarité obtenu

* Les métriques du filtrage collaboratif, basé sur l'approche modèle **Surprise SVD**, sont exprimées sur une échelle de 100, car les évaluations (ratings) reposent sur un score de pertinence défini sur 100.
En revanche, les scores utilisés dans les autres modèles sont exprimés sur une échelle de 5.

Voici le benchmark après réalignement de la base de données et l'utilisation unifiée du score de pertinence, prenant aussi en compte les temps de calcul:

	Modèle	RMSE	MAE	Temps de calcul de l'evaluation
0	SVD (Surprise)	15.2936	12.3635	Quelques secondes
1	Item-based + SVD (modele)	15.5542	12.2443	Plusieurs heures
2	User-based (approche mémoire)	16.5321	14.0516	30 min
3	Item-based (approche mémoire)	15.8114	13.1578	30 min
4	Filtrage basé sur le contenu	Non applicable	Non applicable	RAS

A noter: le modèle SVD (Surprise) obtient de peu la meilleure performance en terme d'erreur, mais celle-ci pourrait encore être améliorée légèrement grâce à l'optimisation des hyper-paramètres.

On peut retenir quelques informations importantes :

- Les gains de performance sont de l'ordre de 1-2% (on peut encore améliorer du même ordre de grandeur le modèle SVD + surprise avec l'optimisation des hyper-paramètres)
- Dans le contexte d'un système de recommandation, nous n'avons pas besoin d'une précision très élevée (parce que nous ne sommes pas dans un contexte de vie ou de mort et parce que l'on a besoin de garder de la diversité dans les recommandations)
- Donc au final, toutes choses égales par ailleurs, on privilégie la solution avec le meilleur temps de calcul ET on préfère ne pas se lancer dans les modélisations avancées (beaucoup de travail en plus à la toute fin du projet pour un gain incertain)

Commentaires :

Le modèle SVD a démontré une performance supérieure par rapport aux autres approches évaluées, avec un **RMSE** de **12,78** sur une échelle de 100 et un **MAE** de **10,75** sur 100. Ces résultats indiquent une précision élevée dans les prédictions de notes, reflétant une capacité du modèle à minimiser efficacement l'écart entre les notes prédites et les notes réelles.

Comparativement, ces métriques surpassent celles des autres méthodes testées, confirmant que le modèle Surprise SVD est mieux adapté pour capturer les interactions latentes et complexes entre les utilisateurs et les éléments dans le cadre du filtrage collaboratif. Cela met en évidence sa robustesse et sa pertinence dans le contexte des données utilisées, où l'échelle et les scores sont particulièrement exigeants.

- **Apprendre à travailler en équipe autour d'une problématique classique de la data science**

Le travail en équipe s'est avéré efficace grâce à une collaboration active de tous les membres. Nous avons rapidement identifié la répartition des tâches, ce qui nous a permis de respecter les délais impartis tout en garantissant une qualité optimale du résultat final.

Processus métier concernés :
Cet objectif a été pleinement atteint et illustre une approche applicable à la gestion de projets collaboratifs en data science, ainsi qu'à la coordination entre différentes équipes techniques. L'expérience a permis de développer des compétences essentielles, telles que l'identification des points forts de chaque membre de l'équipe et l'optimisation de ces compétences dans un contexte métier. Par ailleurs, la gestion efficace du temps a été un facteur clé dans la réussite de ce projet, renforçant ainsi notre capacité à travailler sur des problématiques complexes tout en respectant les contraintes opérationnelles.

- **Préparer une base de données : riche, propre, utile, optimisée, etc.**

La base de données a été préparée en respectant les critères de qualité attendus : complète, pertinente et adaptée aux besoins des modèles à tester. Une exploration approfondie des données a permis d'identifier les variables exploitables ainsi que les relations entre les tables et les variables nécessaires à l'analyse.

Dans le processus de préparation, nous avons découvert les défis et limitations qu'une grande base de données peut poser dans un projet de data science. Cela nous a conduit à maîtriser des bibliothèques Python adaptées à ces tâches et à prendre des décisions éclairées en matière de réduction dimensionnelle. Ces étapes se sont révélées cruciales pour optimiser les données et tester efficacement les différents modèles prévus dans le cadre du projet.

Processus métier concernés :
Cet objectif est applicable aux étapes clés de la préparation des données, incluant le nettoyage, l'exploration et la réduction dimensionnelle, indispensables pour la réussite de la modélisation et l'analyse dans un contexte métier.

- **Sélectionner et ajuster un modèle de machine learning en fonction de la problématique, des variables explicatives et de la cible**

Nous avons identifié plusieurs modèles de machine learning à tester afin d'évaluer leurs performances et leur pertinence pour la problématique posée. Cette démarche nous a permis de comparer les résultats et de sélectionner le modèle le plus adapté.

Les modèles testés incluent :

- Filtrage Collaboratif : approche mémoire
    - User-based
    - Item-based
- Filtrage Collaboratif : approche modèle
    - Item-based + SVD
    - Surprise (SVD)
- Filtrage basé sur le contenu

Ces expérimentations nous ont permis de comprendre les spécificités et les avantages de chaque approche. Finalement, nous avons retenu le modèle **Surprise SVD**, qui s'est avéré le plus stable et performant parmi ceux testés. En termes de métriques, ce modèle a obtenu des scores de RMSE et MAE plus significatifs que les autres, démontrant une meilleure précision dans la prédiction des notes des utilisateurs. Grâce à son approche supervisée, il a surpassé les autres modèles dans la qualité des prédictions et la pertinence des résultats.

Processus métier concernés :
Cette étape s'inscrit dans un contexte métier où il est essentiel de savoir identifier et sélectionner le modèle le plus pertinent pour répondre aux attentes et aux objectifs, tout en évaluant les performances des différentes approches disponibles.

- **Créer concrètement un système de recommandation en fonction de l'utilisateur et du contenu (e.g. prédire de manière pertinente et précise le rating d'un film)**

Nous avons conçu un système de recommandation capable de prédire de manière précise le rating qu'un utilisateur pourrait attribuer à des films qu'il n'a pas encore visionnés. Ce système repose sur deux approches principales :

Filtrage collaboratif basé sur le modèle (Surprise SVD) :
Après une analyse comparative, ce modèle a été retenu pour sa robustesse et sa précision dans la prédiction des notes, ce qui le rend particulièrement adapté à notre objectif.

Filtrage basé sur le contenu :
Bien que cette méthode ait été développée, le temps a manqué pour implémenter un modèle hybride combinant les deux approches. Néanmoins, le filtrage collaboratif basé sur le modèle a été identifié comme la solution la plus pertinente pour proposer un système de recommandation efficace et cohérent.

Processus métier concernés :
Un tel système de recommandation pourrait être intégré dans des plateformes en ligne pour personnaliser l'expérience utilisateur, comme dans le domaine du streaming (films, séries), du

e-commerce (produits), ou même des applications éducatives (cours ou ressources). Il constitue un levier stratégique pour améliorer l'engagement des utilisateurs, renforcer leur satisfaction et augmenter la fidélité à la plateforme.

- **Présenter les résultats de nos recherches avec les outils et selon les standards de la data science**

Nous avons développé une application interactive avec **Streamlit** pour présenter les résultats de nos analyses de manière claire et dynamique. Cet outil nous a permis de structurer nos visualisations, nos métriques et nos modèles de manière intuitive, offrant ainsi une base solide pour la présentation des résultats.

Cependant, au cours de cette implémentation, nous avons également identifié certaines limitations inhérentes à la bibliothèque Streamlit, que nous avons documentées dans la section des difficultés rencontrées durant le projet. Ces observations pourraient être utiles pour orienter des choix futurs vers des outils complémentaires ou alternatifs en fonction des besoins spécifiques.

Processus métier concernés :
Ce processus est particulièrement utile dans des contextes de soutenance de projet, de communication de résultats à des parties prenantes ou encore de présentation à des décideurs ou collaborateurs souhaitant explorer les détails du projet. Il s'agit d'une étape clé pour valoriser le travail accompli et faciliter la prise de décision sur la base des résultats obtenus.

- **Au-delà des objectifs établis :**

Dans la première partie du rapport, nous avons mentionné que les **préférences temporelles** auraient pu être intégrées en tant que variables dans les modèles de recommandation de films. Cependant, nous avons choisi de ne pas inclure cette variable, estimant que les meilleurs films à recommander sont **intemporels** et que leur pertinence ne dépend pas nécessairement de facteurs temporels.

De plus, nous avons également indiqué que la variable **genre** aurait pu être davantage exploitée. Cette variable a effectivement été prise en compte dans le modèle basé sur le contenu, mais pas dans le modèle retenu, à savoir le **SVD de Surprise**. Nous avons estimé que le **score de pertinence** calculé par ce modèle atténuait déjà l'influence des genres ou du nombre de vues, rendant cette variable moins critique dans ce contexte.

Conclusion : Temps forts et principaux enseignements du projet

Ce projet de data science nous a permis de développer un système de recommandation de films à travers une exploration méthodique des différentes approches de filtrage collaboratif et de filtrage basé sur le contenu. Voici les temps forts et les principaux enseignements de cette expérience :

Difficultés initiales et consolidation des données

- L'intégration de deux jeux de données distincts (IMDb et MovieLens) a représenté un défi important.
- Le volume massif de données et la présence de nombreuses valeurs inutilisables ont nécessité un nettoyage approfondi avant de pouvoir démarrer la modélisation.

Premières itérations : exploration des modèles simples

1. Filtrage collaboratif User-User :
    - En utilisant les similarités entre utilisateurs, nous avons généré des recommandations du type : *"Les utilisateurs ayant des goûts similaires aux vôtres ont aimé ces films."*
    - Les résultats, bien que prometteurs, montrent une similarité faible et une MAE initiale élevée (>50%).
    - Une optimisation des hyperparamètres via GridSearch a permis des résultats légèrement améliorés, avec un choix optimal de k=19 voisins.
2. Filtrage collaboratif Item-Item :
    - En calculant les similarités entre les films, nous avons généré des recommandations : *"Ces films sont similaires à ceux que vous avez notés."*
    - Les résultats se sont avérés différents de ceux du modèle User-User, et les notes initiales toujours peu discriminantes.
    - L'ajout d'une factorisation matricielle via SVD a permis de réduire la complexité des calculs et de faire apparaître des facteurs latents intéressants.
3. Embryon de filtrage basé sur le contenu :
    - Une première version, limitée à l'analyse des titres des films, a été mise en place.
    - Cette approche, bien que simpliste, a ouvert la voie à des itérations plus riches basées sur des données enrichies (acteurs, réalisateurs, genres, etc.).

Évolutions majeures et solutions performantes

1. Filtrage basé sur le contenu (v2) :
    - En ajoutant des colonnes enrichies sur les films, nous avons pu créer une description plus complète des contenus.
    - Le modèle a démontré une pertinence accrue pour recommander des films partageant des acteurs, des réalisateurs, et dans une moindre mesure, des genres.

    - L’utilisation d’un score de similarité a permis un classement clair des recommandations.
2. Filtrage collaboratif avec la librairie *Surprise* :
    - La factorisation plus puissante de Surprise a offert un gain significatif en termes de performance (MAE réduite à environ 10%).
    - Un score de pertinence intégrant les notes et le nombre de votes IMDb a permis un ranking plus discriminant.
    - Bien que l'optimisation des hyperparamètres n'ait apporté que des gains marginaux, ce modèle s’est avéré le plus performant et le plus adapté.

Décisions finales et présentation utilisateur

- Nous avons décidé d’implémenter un modèle **“Surprise”**, à la fois performant et user-friendly :
    - Recommandations diversifiées incluant des choix "hors des sentiers battus".
    - Gestion du problème de *cold start*.
    - Présentation visuelle des recommandations avec des faux noms d’utilisateurs et un rappel des films préférés.

Perspectives et axes d'amélioration

- Enrichir les descriptions de films par du web scraping pour affiner le filtrage basé sur le contenu.
- Enrichir la base de données afin d’améliorer la confiance et la satisfaction des utilisateurs (intégration de films plus récents et des séries, ajout d’autres sources représentant la voix du Peuple e.g. Rotten Tomatoes)
- Explorer les méthodes de Deep Learning pour un système plus sophistiqué, bien que cela relève d’un projet futur.

Enseignement principal

Bien qu'il soit possible d'explorer des approches plus complexes, la simplicité et l'efficacité du modèle **Surprise** en font une solution de choix, démontrant qu’un équilibre entre performance et expérience utilisateur peut surpasser des gains marginaux en précision brute.

Bibliographie

Modules de la formation

- 101 - Python pour la Data Science
- 104 - Statistiques Exploratoires
- Méthodologie en Data Science
- 111 - DataViz avec Matplotlib
- 112 - DataViz avec Seaborn

- 114 - Matplotlib - Compléments
- Systèmes de recommandation
- 125 - Méthodes de réduction de dimension
- 117 - Streamlit

Sites webs

- https://www.alldatascience.com/recommender-systems/simple-recipe-recommender-system-with-scikit-surprise/
- https://medium.com/@sumanadhikari/building-a-movie-recommendation-engine-using-scikit-learn-8dbb11c5aa4b
- https://realpython.com/build-recommendation-engine-collaborative-filtering/
- https://abhaysinghr.hashnode.dev/building-a-simple-recommendation-system-with-scikit-learn
- https://medium.com/@paul0/collaborative-filtering-25d181107082
- https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0

Annexes

Références

Jeux de données

Dataset MovieLens : https://grouplens.org/datasets/movielens/20m/ - source principale
Dataset IMDb : https://www.imdb.com/interfaces/ - source complémentaire
Rapport d'exploration des données (.xlsx) :
https://docs.google.com/spreadsheets/d/1vgNMtYaLKitcrDfo4GNbjIM65D6oeO10/edit?usp=sharing&ouid=105672209662968344116&rtpof=true&sd=true
Explication du code : disponible sur notre Github

Autres Liens

Github : https://github.com/DataScientest-Studio/SEP24-BDS-RECO_FILM
The Movie Database : (https://www.themoviedb.org/) - source non-utilisée