Projet Netfloox

Contexte du projet

L'objectif de notre projet est de développer une interface de streaming vidéo capable de proposer aux clients des contenus susceptible de les intéresser. Et doté d'un système de prédiction de popularité afin d'aider les équipes de production de contenu vidéo à connaitre les potentielles caractéristiques susceptibles d'influencer la popularité / satisfaction d'un contenu.

Résumé des étapes effectuées

- Création d'un trello
- Récupération des données
- Insertion des données
- Exploration et analyse des données
- Nettoyage et pré processing des données
- Développement d'un système de recommandation
- Développement d'un système de prédiction de popularité
- Développement d'une interface web pour l'utilisation de ces services

Livrables

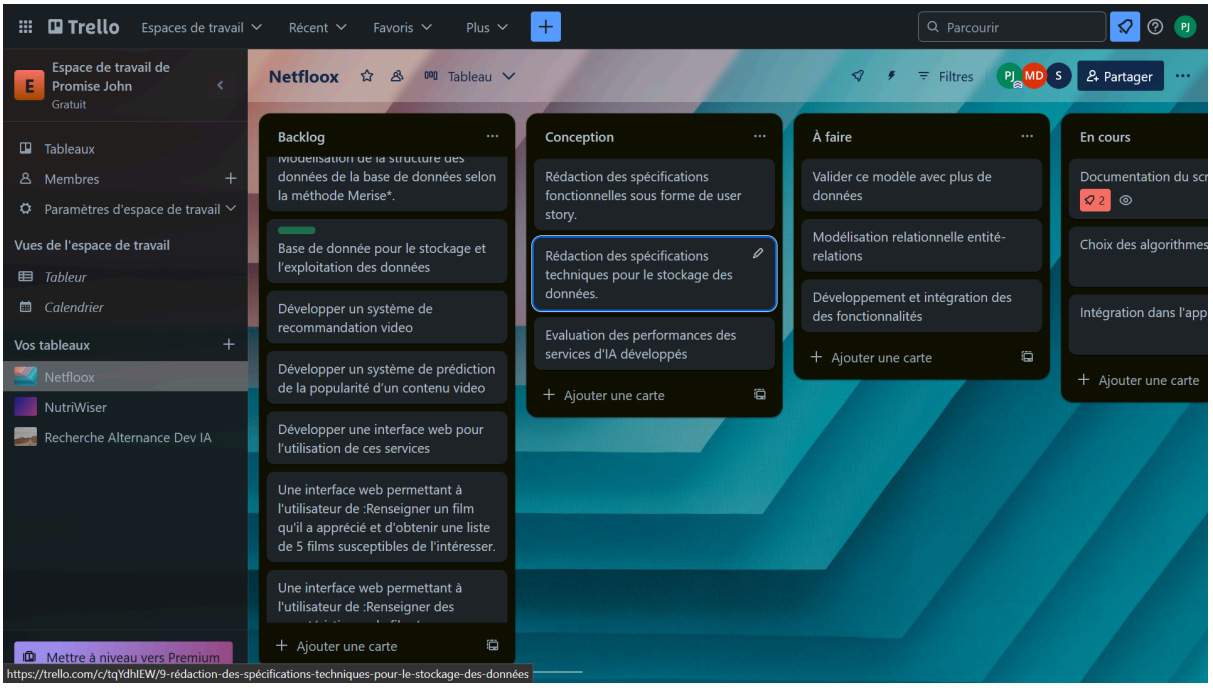
- Un modèle relationnel de données
- Une base de données alimentées
- Un algorithme de recommandation
- Un algorithme de prédiction de la popularité
- Une interface web permettant à l'utilisateur de :
    - Observer les analyses graphiques et statistiques de la base de données.
    - Renseigner un film qu'il a apprécié et d'obtenir une liste de 5 films susceptibles de l'intéresser.
    - Renseigner des caractéristiques de film (genre, acteur, producteur, etc) est d'en estimer la popularité potentielle.
- Une présentation écrite et orale résumant vos travaux.
- Un dépôt Github.
- Un Trello du projet

Users Story

- En tant qu'utilisateur qui souhaite regarder un film, je souhaite rentrer un film que j'ai aimé et obtenir une suggestion de 5 films à regarder en rentrant le nom dans un champ de texte dans une application web
- En tant que producteur de film, je souhaite rentrer les caractéristiques d'un film que je souhaite produire et obtenir une estimation de sa popularité, en rentrant les caractéristiques dans des champs dans une application web
- En tant qu'utilisateur, je souhaite observer les caractéristiques des films de la base de données IMDB via des graphiques sur une application web

Création d'un Trello

Afin d'organiser notre travail, nous avons utilisé Trello pour créer un Kanban, une méthode issue des approches agiles. Cet outil nous a permis de planifier les différentes étapes du projet, de définir et de répartir nos tâches, ainsi que de suivre efficacement l'avancement. Nous avons régulièrement mis à jour ce tableau tout au long du projet afin d'assurer une gestion fluide et structurée.



Récupération des données

Nous avions plusieurs jeux de données à notre disposition : un premier contenant environ 8 000 films et un second, bien plus volumineux, avec près de 8 millions de films, disponible via ce lien : https://datasets.imdbws.com/.

En travaillant avec le jeu de données de 10 000 films, nous avons rapidement rencontré un dilemme. Ce dataset, de nature relationnelle, contenait des colonnes dépendantes d'autres tables, notamment via des clés étrangères. Par exemple, la table title_basics répertorie les titres de films, tandis que la table title_ratings contient les notes attribuées à ces films, les deux étant liées par un identifiant unique (tconst).
Cependant, dans cet échantillon, très peu de films disposaient de toutes leurs informations associées. Seulement une petite poignée. Rendant impossible l'entraînement efficace d'un modèle sur une base aussi incomplète. Nous pensons que lors de la création de cet échantillon, certaines tables ou lignes ont été extraites sans prendre en compte les dépendances avec les clés étrangères.

Pour pallier ce problème, nous avons opté pour l'importation du jeu de données complet composé de plusieurs millions de lignes, garantissant ainsi une couverture plus cohérente des relations entre les tables.

Exploration et analyse des données

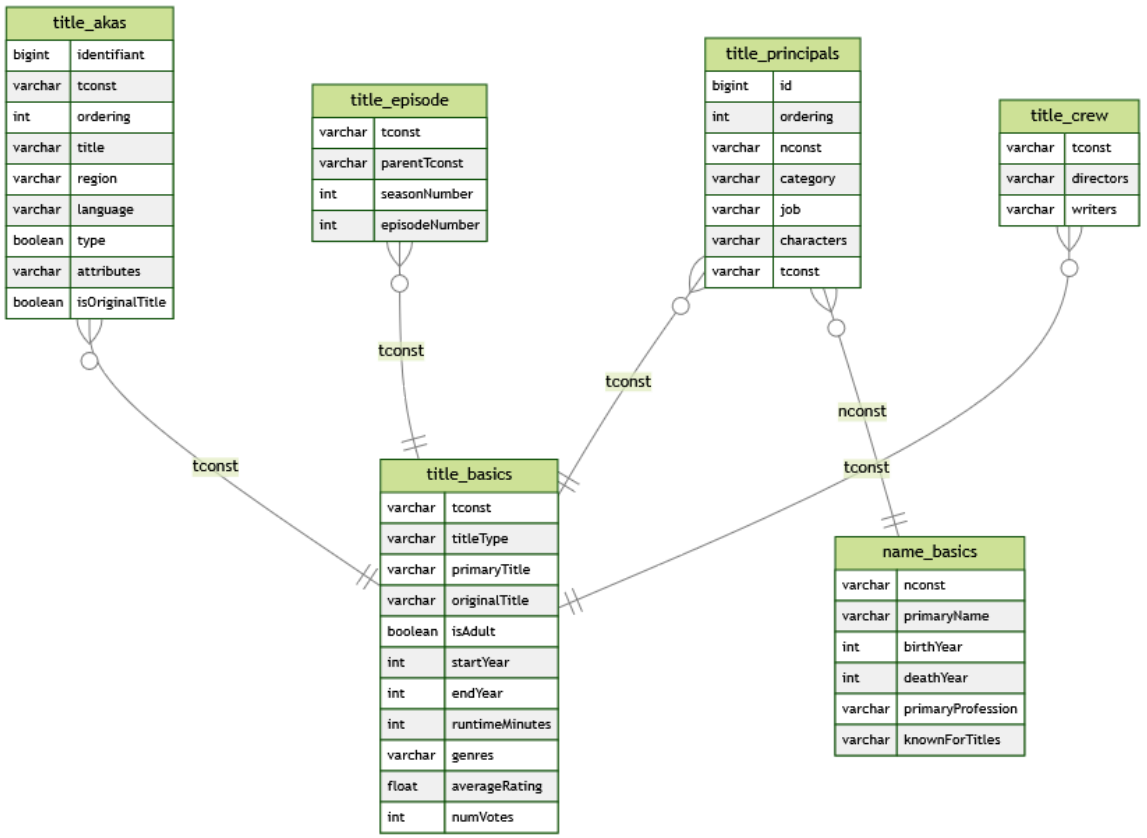
Avant de travailler avec le grand dataset, le petit dataset nous a tout de même été utile pour une première exploration. Nous avons utilisé Pandas pour examiner localement les tables, comprendre leur structure, et analyser les données qu'elles contiennent. Cette étape nous a permis de répondre à plusieurs questions essentielles :

- À quoi correspond chaque table et quelles informations contient-elle ?
- Quelles sont les relations entre les tables ?
- Quelles colonnes jouent le rôle de clés étrangères ?
- Existe-t-il des tables de jointures ?

Voici les tables que nous avons étudiées :

- **title_akas** : contient les différentes versions ou traductions des titres.
- **title_basics** : regroupe les informations principales sur les films et séries (type de contenu, titre principal, genre, année, etc.).C’est la table principale.
- **title_crew** : répertorie les réalisateurs et scénaristes associés à chaque titre.
- **title_ratings** : fournit les notes attribuées aux films et séries ainsi que le nombre de votes.
- **name_basics** : contient les informations sur les personnes (acteurs, réalisateurs, etc.) associées aux productions.
- **title_principals** : relie les personnes aux titres en détaillant leur rôle ou fonction (acteur, réalisateur, etc.).
- **title_episode** : fournit des détails spécifiques aux épisodes de séries (numéro de saison et d’épisode, parent de la série).

Grâce à cette exploration, nous avons pu concevoir un Modèle Conceptuel de Données (MCD), ce qui nous a aidés à visualiser la structure globale du dataset et les relations entre les tables.



Cette analyse approfondie nous a également permis d'identifier les colonnes pouvant servir de features potentielles pour nos modèles de prédiction et de recommandation.

```
## features à utiliser

- primarytitle
- titletype
- isadult
- startyear
- endyear ?
- genres
- agrégation / pondération de la colonne averagerating et numvotes
- runtimeminutes
- seasonnumber
- episodenumber
- nombre de régions dispo
- liste des régions dispo
- category (acteur, self, actrice, producteur ?) des primary name
```

Au cours de notre exploration, nous avons également constaté que certaines colonnes contenaient un grand nombre de valeurs manquantes. Par exemple, plusieurs films n'avaient pas de notes associées dans la table title_ratings ou étaient dépourvus de certaines caractéristiques importantes telles que les genres ou les réalisateurs.

Pour éviter de compliquer inutilement le prétraitement, nous avons décidé de ne pas imputer artificiellement ces données manquantes. À la place, nous avons choisi de laisser le modèle pénaliser automatiquement les films ou séries incomplets.

Par exemple, un film sans note ou sans informations essentielles sera moins susceptible d'être recommandé.

Cette approche présente un avantage : elle incite les utilisateurs ou les producteurs de contenus à mieux renseigner les informations sur les films, afin de maximiser leur visibilité et leur popularité.

Ce choix est en cohérence avec nos objectifs, car un contenu peu renseigné ou peu documenté est généralement moins pertinent ou moins populaire pour les utilisateurs. Il est donc naturel que le modèle reflète cette réalité dans ses recommandations.

Insertion des données

Un **pré-nettoyage** minimal a été appliqué sur certaines colonnes, adapté à chaque fichier, avant leur chargement dans la base de données. Après un temps d'attente conséquent, les données ont été intégrées à la BDD avec succès.

Nous avons ensuite pris soin de compter le nombre de lignes présentes dans chaque table, mais également de vérifier combien de lignes restaient lorsque les **clés étrangères** étaient respectées, c'est-à-dire lorsque chaque ligne était liée à des enregistrements valides dans d'autres tables. Voici les résultats de ces vérifications :

```
## Lignes en moins si ajout des clés étrangères

title_principals -> title_basics : 67 215 123 / 90 000 000
title_principals-> name_basics : 90 0000 000 / 90 000 000
title_princpals -> title_basics & name_basics : 67 000 000 / 90 000 000
title_crew -> title_basics : 11 000 000 / 11 000 000
title_akas -> title_basics : 40 000 000 / 47 000 000
title_episode -> title_basics : 7 000 000 / 8 000 000
```

Cette étape de vérification des clés étrangères a été utile, mais elle était pertinente surtout dans le cadre où nous aurions décidé de créer des contraintes de clés étrangères dans notre base de données. Cependant, nous avions décidé de ne pas créer de clés étrangères, pour cause de plusieurs contraintes.

Notre alternative a été d’extraire un sous-ensemble des 10 000 films les plus récents, en incluant toutes les features pertinentes. Les données provenant des différentes tables liées ont été jointes dans un unique dataset.

Cette approche nous a permis de :

- Regrouper toutes les informations nécessaires dans un seul DataFrame,
- Simplifier les opérations de prétraitement et de nettoyage,
- Réduire les temps d’exécution des analyses en travaillant sur un volume de données plus maniable.

Avec cette configuration, nous avons optimisé l'équilibre entre flexibilité, performance et facilité d’utilisation, tout en maintenant une bonne structure logique des données pour nos modèles.

Nettoyage et pré-processing des données

Pour nos données catégorielles, principalement sous forme de texte, nous avons d'abord pris le temps d'explorer les colonnes afin d'identifier les éventuels **biais** ou **incohérences**. Selon les observations faites dans chaque colonne, nous avons défini une liste de transformations adaptées au contexte, tout en anticipant les conséquences de chaque opération.

Exemple 1 : Colonne title_type (type de contenu)

Observation : L'exploration des données nous a montré que les valeurs dans cette colonne sont cohérentes et normalisées (ex. : "movie", "series", "short"), sans anomalies ni doublons problématiques.

Décision : Cette colonne n’a pas besoin de transformations complexes telles que la tokenisation, le stemming, ou la suppression des stop words, car il s'agit de catégories prédéfinies.

Nous avons choisi d’appliquer un encodage (par exemple, one-hot encoding ou encodage ordinal) afin de rendre ces données exploitables par les modèles.

Exemple 2 : Colonne primary_title (titre principal)

Observation : L’examen des premiers titres a montré une grande variété de valeurs. Les titres sont souvent composés de plusieurs mots, et certains titres sont définis en partie par des stop words (ex. : "The Good Place", "The Walking Dead").

Décision : Une tokenisation des titres est nécessaire pour isoler chaque mot constitutif.

Les stop words ne doivent pas être supprimés, car ils peuvent être essentiels au sens du titre. Par exemple, "The Good Place" perdrait sa signification si le mot "The" était supprimé.

Nous avons choisi d’utiliser un stemming ou une lemmatisation pour regrouper les différentes variantes d’un même mot (ex. : "run", "running", "runs").

Pour un modèle de recommandation basé sur les titres, nous avons opté pour une vectorisation via TF-IDF, afin d'attribuer davantage de poids aux mots rares et significatifs tout en réduisant l’impact des mots trop fréquents.

Exemple 3 : Colonne genres

Observation : Nous avons constaté que cette colonne contient plusieurs combinaisons de genres, telles que "Action|Adventure|Fantasy". Au total, 354 combinaisons différentes ont été identifiées.

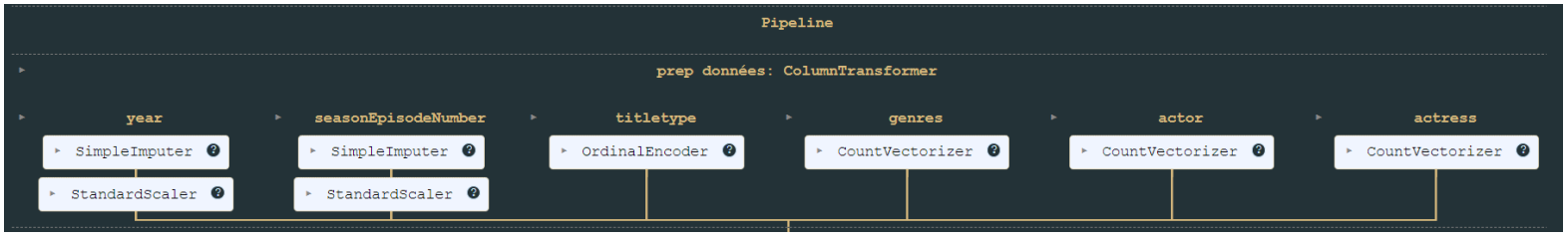
Décision : Il est nécessaire d’appliquer une tokenisation pour séparer les différents genres au sein d’un même enregistrement.

Chaque genre doit ensuite être encodé (multi-label ou one-hot encoding) afin d’être exploitable dans les modèles d’apprentissage automatique.

Choix et entraînement des modèles

Pipeline de base

Nous avons commencé par choisir nos features et définir pour chacun d’eux le traitement adapté.
L’année est imputée de la moyenne et scalé avec standard scaler. Le nombre de saisons et d’épisode sont imputés de 1 (un film sans mention de nombre d’épisodes ou de saison est un film unique dont avec une saison d’1 épisode). On labelise ensuite le titletype avec ordinal encoder. Pour finir, on a commencé par utiliser MultiLabelBinarizer sur les colonnes contenant des listes de valeurs (genres, actor, actress, self, etc.). Mais cette méthode n’était pas correctement intégrée dans les pipelines. Nous avons donc choisi d’utiliser en remplacement Count Vectorizer qui sur des listes de valeur donne un résultat similaire.



Système de recommandation de film

Nous avons choisi d’utiliser la fonction cosine_similarity de scikit-learn pour notre système de recommandation, ainsi en créant une matrice de similarité entre les films.
Nous avons ainsi utilisé notre pipeline pour traiter les features, puis appliqué la fonction cosine_simiarity pour obtenir notre matrice.

On utilise ensuite cette matrice pour récupérer les cinq films ayant le plus haut score de similarité sur la ligne du film choisit (en excluant lui-même).
Comme nous n'avons pas de données de test, il est impossible d'évaluer de manière efficace les performances de ce système, mais les résultats obtenus à la main avec quelques films connus nous ont parus plutôt pertinent.

Système de prédiction de popularité d'un film

Nous avons commencé par la bibliothèque Pycaret pour évaluer de manière théorique la performances de plusieurs modèles sur un échantillon de données.
Le modèle qui en est ressorti le plus performant était catboostregressor (mais non intégré dans scikit-learn), puis RandomForestRegressor.

Dans un deuxième temps, nous avons repris la pipeline de base, et intégré le modèle après le préprocessing.
Nous avons dans un premier temps utilisé LinearRegression qui est le modèle le plus simple a interprété. Nous obtenions un résultat négatif (donc qui explique moins bien les features que la moyenne du score). Nous avons ensuite testé RandomForestRegressor et KNeighboursRegressor. Le plus haut score était obtenu avec KNeighboursRegressor et non RandomForestRegressor, avec en moyenne 0.3 de R^2, 0.0003 de MSE (Mean Square Error) et 0.003 de MAE (Mean Absolute Error).
Nous avons donc choisi de garder ce dernier modèle.

Réalisation de l'application

Objectifs et Fonctionnalités

Cette application a été conçue dans une optique analytique afin de structurer et d'exploiter efficacement un ensemble de données cinématographiques. Elle intègre trois fonctionnalités majeures :

- **Exploration des données** : Visualisation des informations issues de bases de données nettoyées et standardisées.
- **Système de recommandation avancé** : Application d'algorithmes de similarité vectorielle pour suggérer des films comparables en termes de contenu et de popularité.
- **Modèle prédictif de popularité** : Évaluation automatique de l'impact potentiel d'un film sur le public, sur la base de critères précis saisis par l'utilisateur.

Structure et Optimisation du Code

Configuration et Préparation des Données

L'architecture logicielle repose sur des bibliothèques de traitement des données telles que Pandas et NumPy, en complément d'un stockage optimisé sous PostgreSQL. Un mécanisme de mise en cache Parquet réduit les latences en évitant les requêtes répétitives vers la base de données.

Extraction et Nettoyage

Une phase d'ingestion structurée permet d'unifier les sources de données, suivie de processus avancés de transformation :

- **Conversion et homogénéisation** des types de données pour assurer une cohérence analytique.
- **Nettoyage et structuration** des champs textuels.
- **Calcul de métriques avancées**, notamment un score pondéré intégrant la distribution des votes et les évaluations du public.

Approche Modulaire pour la Recommandation

L'algorithme de recommandation repose sur l'analyse de la similarité cosinus entre les caractéristiques vectorielles des films, obtenues après transformation et normalisation des données textuelles et numériques.

Modèle Prédictif de Popularité

Le modèle de régression repose sur un pipeline d'apprentissage automatique combinant KNeighborsRegressor et des transformations spécifiques aux données d'entrée.

Interface Utilisateur

L'intégration avec Streamlit offre une interaction intuitive permettant d'accéder aux différentes fonctionnalités à travers un menu ergonomique et des visualisations interactives.

Optimisations et Perspectives

Des axes d'amélioration ont été identifiés :

- **Affinement du score pondéré** en intégrant une correction logarithmique.
- **Optimisation des performances** en exploitant des techniques de vectorisation avancée. (BERT/SBERT utilisé pour rechercher de la similarité entre texte)
- **Enrichissement de l'expérience utilisateur** par l'ajout d'un système de personnalisation dynamique.