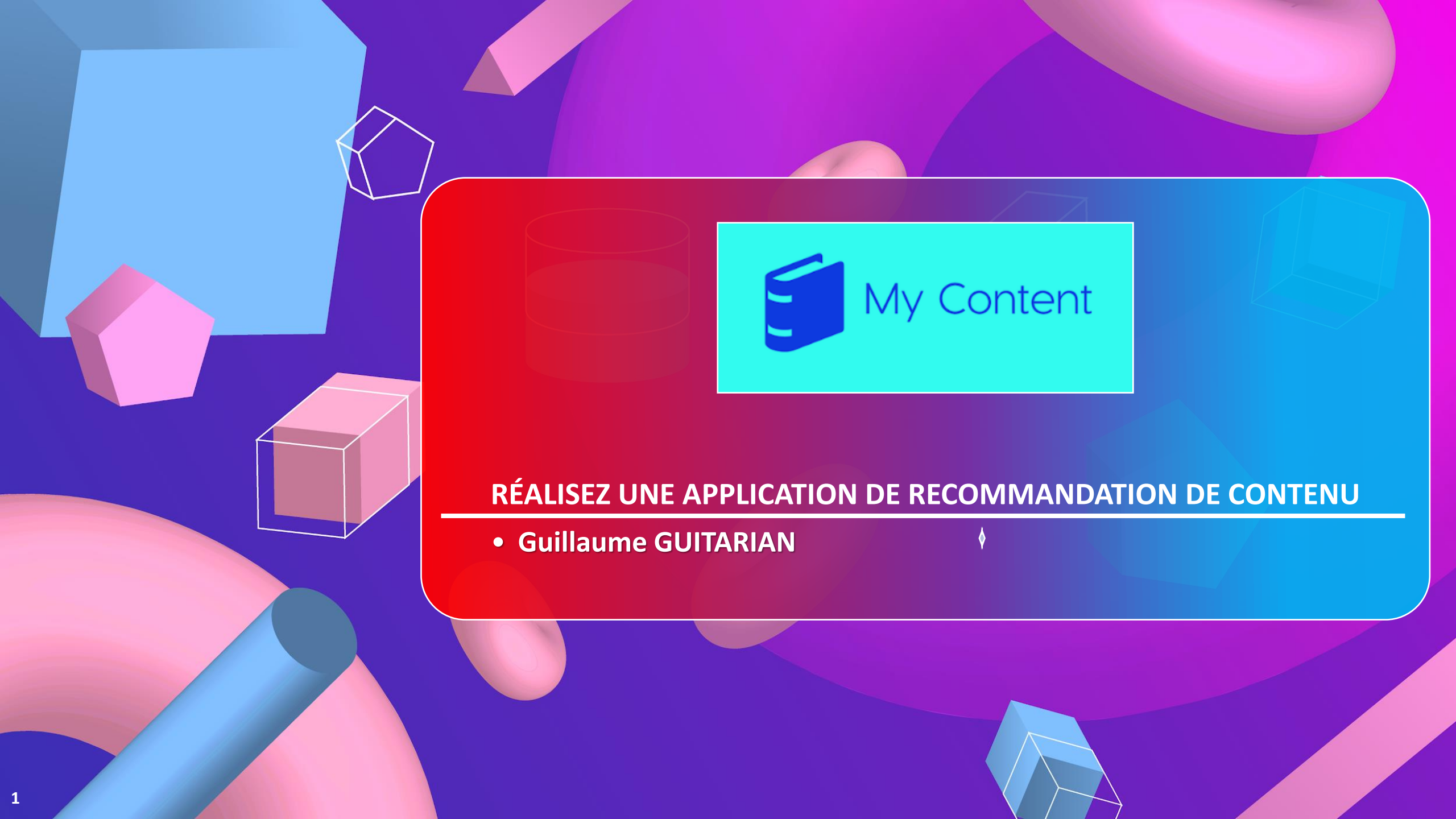


RÉALISEZ UNE APPLICATION DE RECOMMANDATION DE CONTENU

- Guillaume GUITARIAN

Sommaire

Présentation du projet

My Content

Objectif

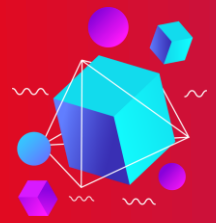
- Encourager la lecture
- Recommander du contenu pertinent

Solution à développer

- Minimum Viable Product (application)
- API en ligne :
  ID d'utilisateur ➔ 5 articles

Gestion des données

LES DONNÉES : SESSIONS



DESCRIPTION

- Issue de Globo.com
- Interactions des utilisateurs avec les articles :
  - user_id
  - click_article_id
  - session_size
  - session_id
  - Etc...

DETAILS

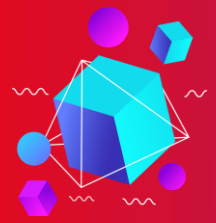
- Dataframe **Clicks** (385 fichiers CSV)
- 3 Millions de sessions
- 320 000+ utilisateurs uniques
- 46 000 articles uniques (lus)

GESTION

- Regroupement par utilisateur
- Regroupement par article
- Distribution nombre de lecture par article
- Distribution nombre d'article lus par utilisateur
- Suppression des entrées « non utiles »
- Gestion des valeurs aberrantes

Gestion des données

LES DONNÉES : ARTICLES



DESCRIPTION

- Données article tel que :
  - vecteurs d'embedding (profil de contenu)
  - article_id
  - nombre de mots
  - etc...

DETAILS

- 2 dataframes :
  articles_embeddings (pickle)
  articles_metadata (csv)
- 360 000+ articles
- Dimension des vecteurs = 250

GESTION

- Données filtrées par rapport au dataframe précédent (clicks)

Gestion des données

RESTRUCTURATION : PRINCIPE

REGROUPEMENTS

1/ Nombre de lectures par **article**

2/ Nombre de lectures par **utilisateur**

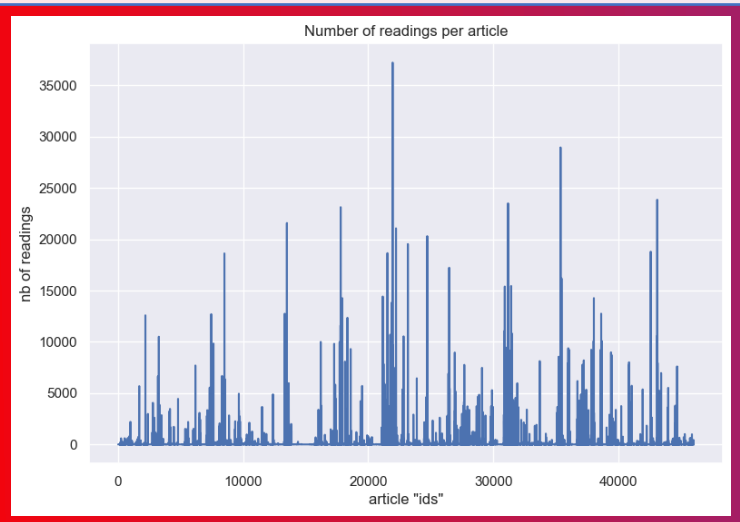
TRAITEMENTS

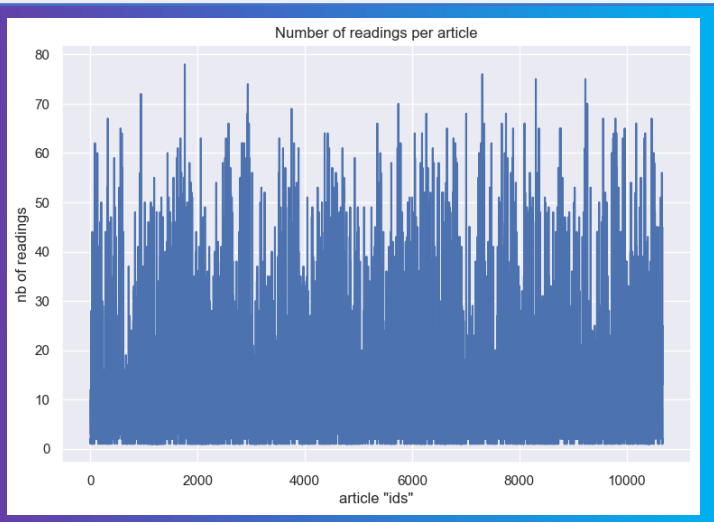
Suppression des données de **basse qualité** (nb lectures insuffisants)

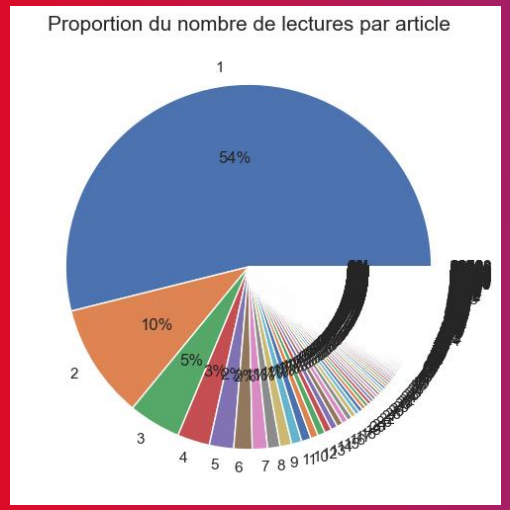
Suppression des **valeurs aberrantes** (robots)

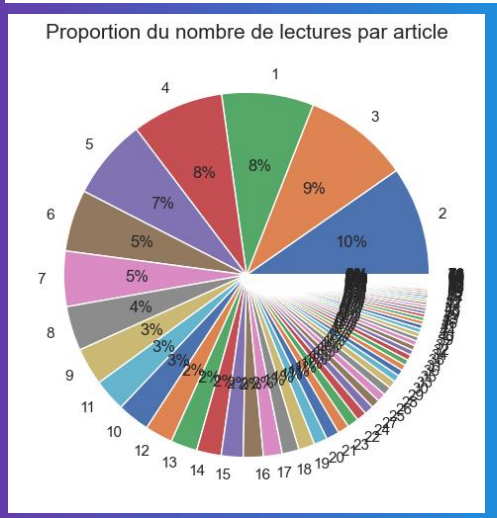
Gestion des données

RÉSULTATS : LECTURES PAR ARTICLE





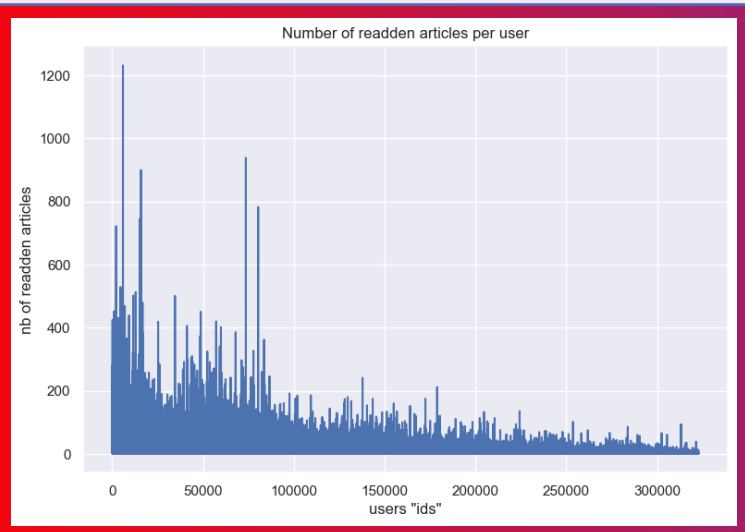


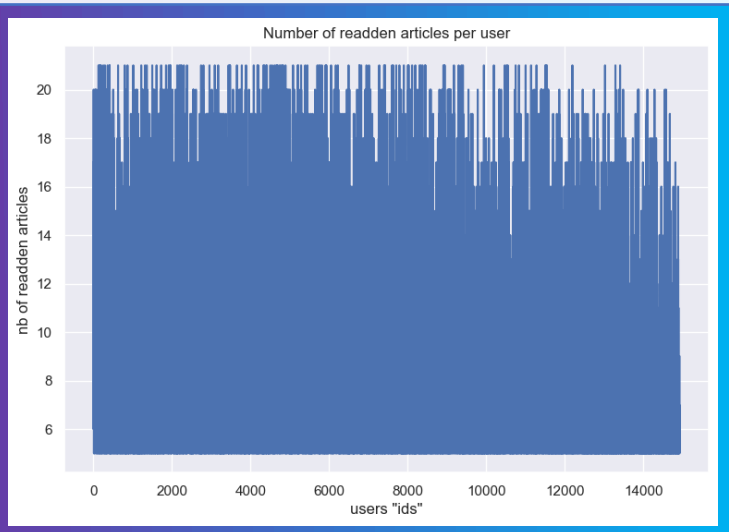


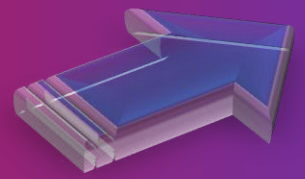
~ 11 000 ARTICLES

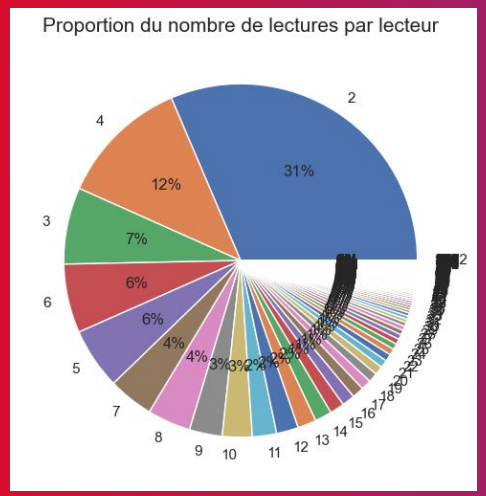
Gestion des données

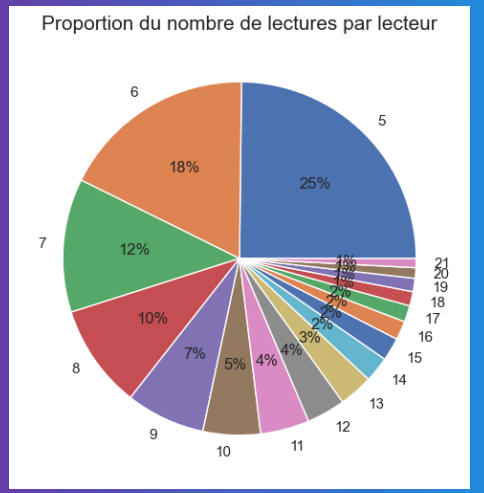
RÉSULTATS : LECTURES PAR UTILISATEUR











~ 15 000 USERS

RECOMMANDATION DE CONTENU : INTRODUCTION

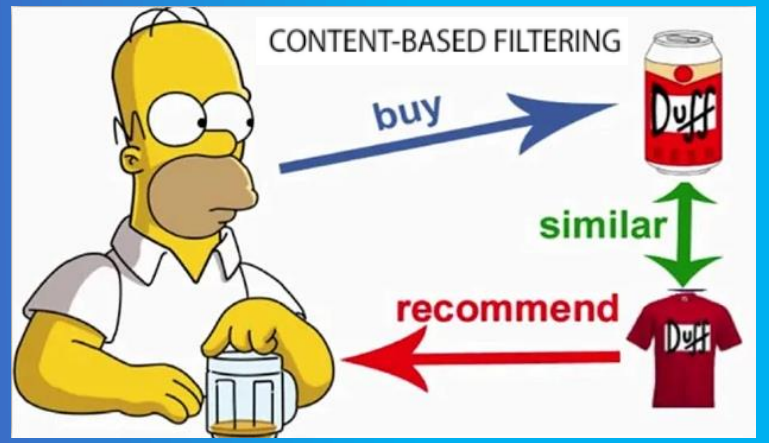
https://miro.medium.com/v2/resize:fit:1100/format:webp/1*Y_QG3Kvfk0fSnCirLBHZ7w.jpeg

Recommandation de contenu

LES 2 PRINCIPALES FAMILLES : PRINCIPES

CONTENT BASED FILTERING

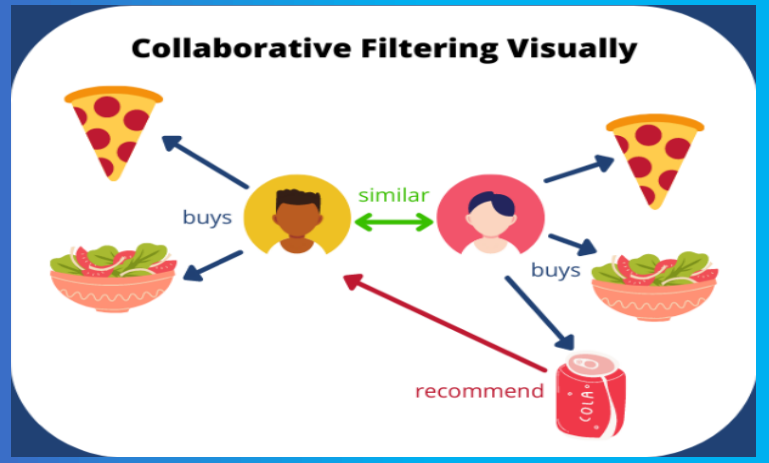
- Préférences utilisateur / caractéristiques du **contenu** (articles_embeddings)
- Déduction du **profil utilisateur**
- REC. = **contenu** similaire (caractéristiques) / profil utilisateur
- AV : Pas besoin d'autres utilisateur pour recommander du contenu
- INC. : Précision des recommandations dépend de la qualité/quantité d'information du **contenu** (contenu **jamais** recommandé)



1*oYpMnPQFZaiZQizgVWBpoA.png (667×403) (medium.com)

COLLABORATIVE FILTERING

- Basé sur la similarités des **profils utilisateur**
- REC. = **contenu exclusif** d'un **profil utilisateur** similaire
- AV : Peut proposer un **genre nouveau** de contenu
- INC. : Nécessité d'autres utilisateurs.



collaborative-filtering-shown-visually.png (800×600) (wp.com)

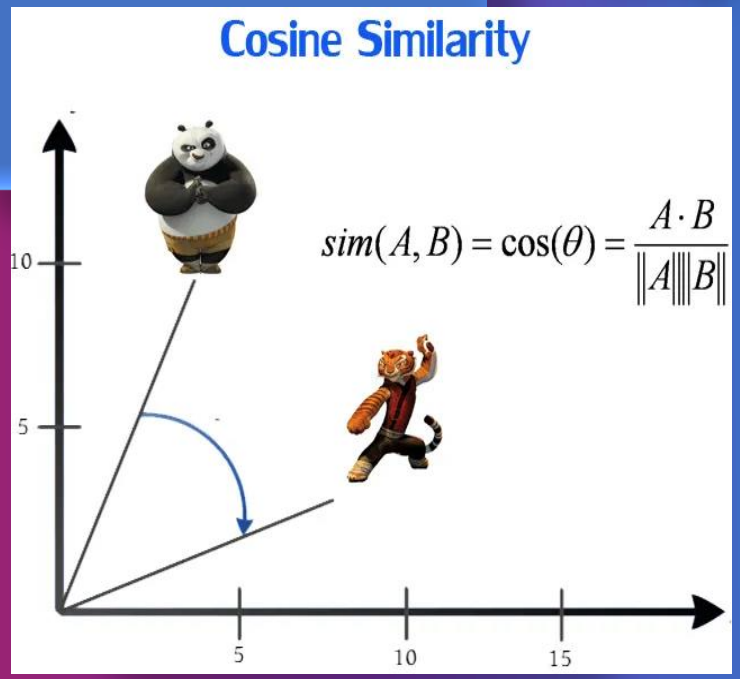
Recommandation de contenu

MOTEUR CUSTOM : CONTENT BASED FILTERING

PROFILS UTILISATEUR de CONTENU

- 1 vecteur d'embedding = caractéristiques d'1 article
- Récupération de la liste des articles lus pour chaque utilisateur
- Moyenne des vecteurs correspondant ➔ Préférences utilisateur

COMPARAISON



1*Q4xQoV8k_7S7xB-NfvFdrw.png (697×535) (medium.com)

RECOMMANDATIONS

- 1 article pour 1 utilisateur
- 7 heures de calcul
- Sauvegarde dans un fichier pickle

Recommandation de contenu

MOTEUR CUSTOM : COLLABORATIVE FILTERING

PROFILS UTILISATEUR de LECTURE

- Matrice USER / ITEM (sparse vectors) en 2 étapes :
- A/ Dataframe du nombre de lecture d'un article pour chaque couple user_id / article_id
- B/ Méthode « pivot_table » avec en index « user_id » et en colonne « article_id »

COMPARAISON

- **Cosinus similarity** : entre les différents vecteurs de lecture des utilisateurs
- Exclusion du résultat de la comparaison d'un vecteur à lui-même
- Récupération des articles non lus à partir du vecteur ayant le meilleur score

RECOMMANDATIONS

- N articles pour 1 utilisateur
- 9 heures de calcul
- Sauvegarde dans un fichier pickle

Recommandation de contenu

MOTEUR CUSTOM : RECOMMANDATIONS FINALES

COLLABORATIVE FILTERING

article_id	1779	1799	1843	1844	1873	1877	1885	1895	1932	1933	...	363921
user_id												
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
17	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
321743	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
321991	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
322334	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
322538	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
322666	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0

CONTENT BASED FILTERING

user_id	click_article_id	ref_vect
3	[236065, 236294, 234686, 235665, 236671, 235745]	[-0.633888045946757, -0.957632948954001, -0.4...
5	[286413, 59929, 60253, 348132, 282785, 202763,...	[0.03758495080503909, -0.9648358085576225, 0.1...
10	[198420, 198322, 142116, 5341, 198321, 199227,...	[-0.42235957831144333, -0.9588247289260229, -0...
16	[162286, 206233, 256163, 29828, 331622, 107289...	[-0.26575193447726114, -0.9233078871454511, 0....
17	[264013, 157861, 348103, 88912, 88911, 161907,...	[-0.15422873795032502, -0.9321221619844436, -0...
...	...	...
321743	[294107, 294111, 121785, 353415, 173546]	[-0.4039882570505142, -0.9399216771125793, -0....
321991	[158158, 159927, 156158, 168377, 76393]	[-0.1290243998169899, -0.9631192803382873, -0....
322334	[84911, 84288, 83549, 84100, 84681, 83406, 83770]	[-0.63419588362532, -0.9585439903395516, -0.48...
322538	[72336, 72335, 72334, 72333, 72329, 72330]	[0.7768468260765076, -0.8668341636657715, 0.44...
322666	[331149, 361585, 330990, 273348, 270958, 1973,...	[-0.7071492142147489, -0.9736078580220541, 0.0...

SORTED RECOMMANDATIONS
➔ PICKLE

```
{
    "user_id 1": {article_id : CS score , ..., article_id : CS score }, # Max list size = 5
    "user_id 2": {article_id : CS score , ..., article_id : CS score }, # Max list size = 5
    ...
    "user_id n": {article_id : CS score , ..., article_id : CS score }, # Max list size = 5
}
```

Recommandation de contenu

LIBRAIRIE SURPRISE : PRÉSENTATION ET RATING

CARACTÉRISTIQUES

- Modèle entrainable de recommandation de contenu (KNN + SVD + Aléatoire)
- Nécessite une variable de **RATING**
- Métriques RMSE, MAE...

```
# Creation of the dataframe. Column names are irrelevant.
ratings_dict = {
    "itemID": [1, 1, 1, 2, 2],
    "userID": [9, 32, 2, 45, "user_foo"],
    "rating": [3, 2, 4, 3, 1],
}
```

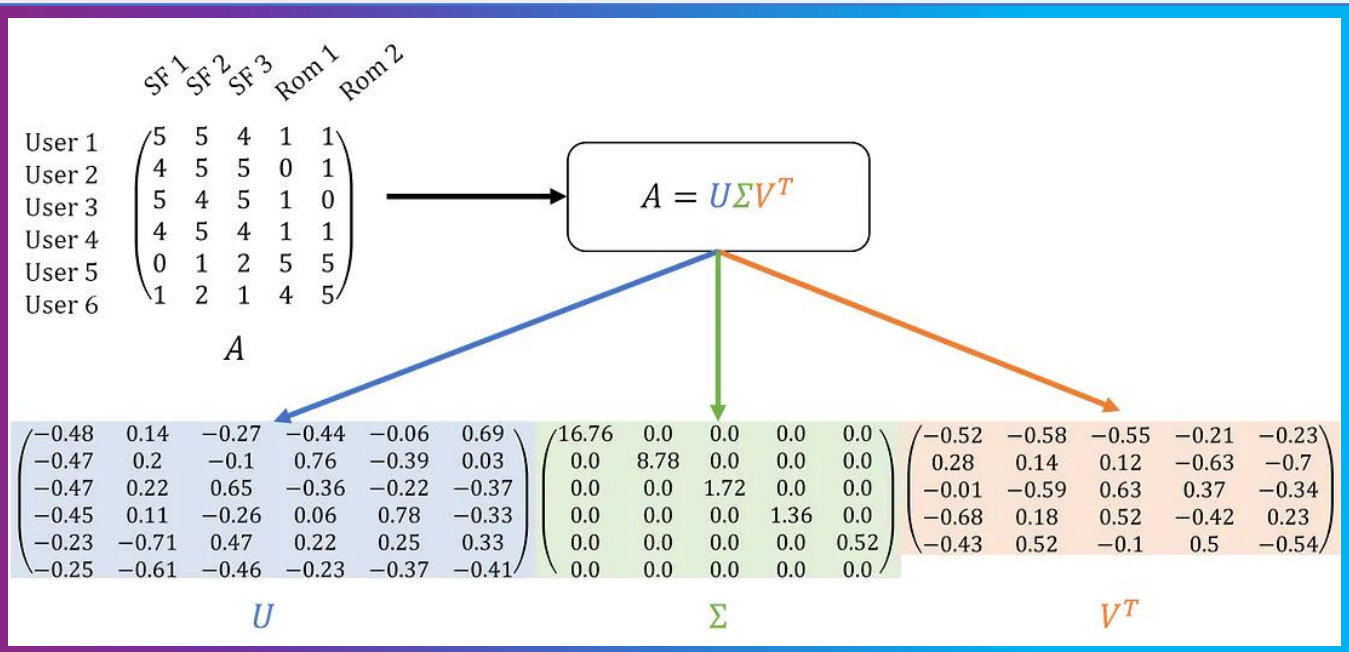
	user_id	click_article_id	nb_readdings	mean_session_size	rating_raw
0	3	234686	1	3.0	3.0
1	3	235665	1	3.0	3.0
2	3	235745	1	4.0	4.0
3	3	236065	1	2.0	2.0
4	3	236294	1	2.0	2.0
...	...	...	...	...	...
123843	322666	313556	1	14.0	14.0
123844	322666	330990	1	14.0	14.0
123845	322666	331149	1	14.0	14.0
123846	322666	360547	1	14.0	14.0
123847	322666	361585	1	14.0	14.0

Recommandation de contenu

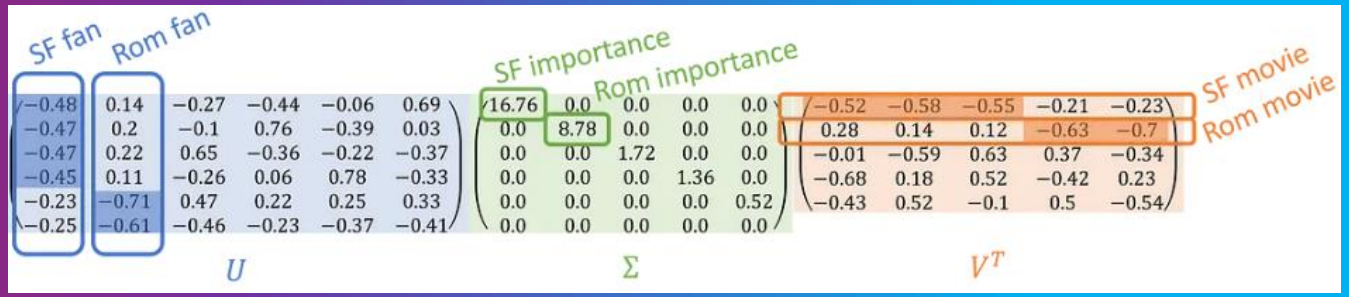
LIBRAIRIE SURPRISE : MODÈLE SVD

CARACTÉRISTIQUES

- **S**ingular **V**alue **D**ecomposition
- Factorisation matricielle
- Nouvelle famille de recommandation de contenu



Singular Value Decomposition in a Movie Recommender System | by Yeunun Choo | Medium



Recommandation de contenu

LIBRAIRIE SURPRISE : MÉTRIQUES

Root Mean Squared Error
Erreur quadratique moyenne

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Mean Absolute Error
Erreur absolue moyenne

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

CARACTÉRISTIQUES

- Toutes 2 mesurent l'ampleur moyenne de l'erreur
- Toutes 2 sont de la même unité que celle du **rating**
- PLAGE = [0 à Max(rating)-Min(rating)] avec :
  - 0 ➔ Aucune erreur
  - Max(rating)-Min(rating) ➔ Tout faux
- RMSE plus sensible que MAE aux grandes erreurs (^{2})

Recommandation de contenu

LIBRAIRIE SURPRISE : RÉSULTATS

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    3.0749  3.0078  3.0494  3.1334  3.1140  3.0759  0.0449
MAE (testset)     2.1920  2.1771  2.1969  2.2058  2.2050  2.1954  0.0105
Fit time          0.70    0.72    0.69    0.70    0.73    0.71    0.02
Test time         0.09    0.09    0.11    0.11    0.11    0.10    0.01
```

```
Evaluating RMSE, MAE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    4.0565  4.0692  4.0768  4.0570  4.0474  4.0614  0.0104
MAE (testset)     2.6874  2.6895  2.6920  2.6813  2.6868  2.6874  0.0036
Fit time          1.53    1.63    1.66    1.73    1.64    1.64    0.06
Test time         0.40    0.41    0.40    0.42    0.40    0.41    0.01
Evaluating RMSE, MAE of algorithm NormalPredictor on 5 split(s).
```

```
Evaluating RMSE, MAE of algorithm NormalPredictor on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    5.1093  5.1388  5.1383  5.1360  5.1323  5.1309  0.0110
MAE (testset)     3.7980  3.7904  3.7828  3.7821  3.7720  3.7850  0.0087
Fit time          0.08    0.08    0.09    0.08    0.09    0.08    0.01
Test time         0.06    0.20    0.06    0.20    0.06    0.12    0.07
```

PRED SVD

```
top_n = get_top_n(predictions, n=5)
get_top_n_for_user(top_n, 101975)
```

```
Recomandation for user : 101975
article: 51230 (est. 5.0568489121633915 - true. 6.0)
article: 226097 (est. 4.905211602682222 - true. 2.0)
article: 100914 (est. 4.756790245009111 - true. 6.0)
```

PRED KNN

```
top_n2 = get_top_n(predictions2, n=5)
get_top_n_for_user(top_n2, 101975)
```

```
Recomandation for user : 101975
article: 51230 (est. 5.326359030258988 - true. 6.0)
article: 226097 (est. 5.326359030258988 - true. 2.0)
article: 100914 (est. 5.326359030258988 - true. 6.0)
```

PRED Random

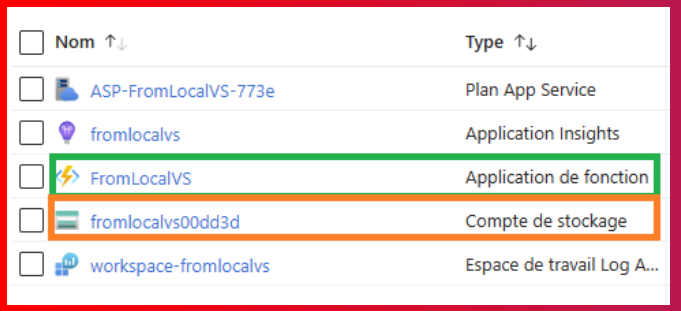
```
top_n3 = get_top_n(predictions3, n=5)
get_top_n_for_user(top_n3, 101975)
```

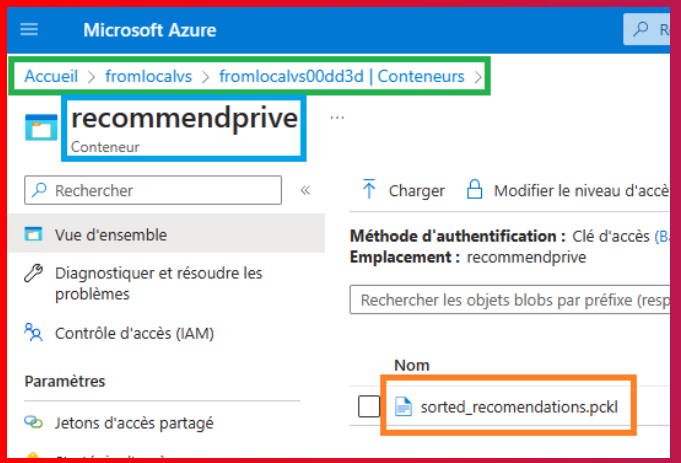
```
Recomandation for user : 101975
article: 100914 (est. 9.97663132935576 - true. 6.0)
article: 51230 (est. 7.159046445033018 - true. 6.0)
article: 226097 (est. 6.782024134999867 - true. 2.0)
```

Inférence

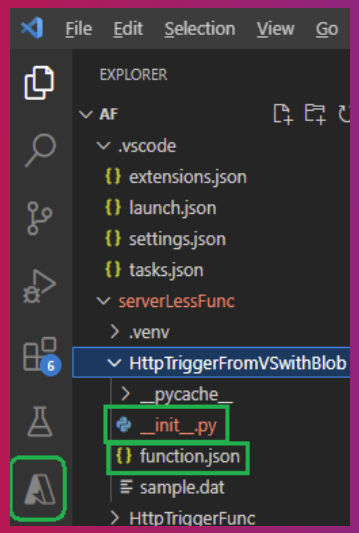
DÉPLOIEMENT : API / SERVEUR

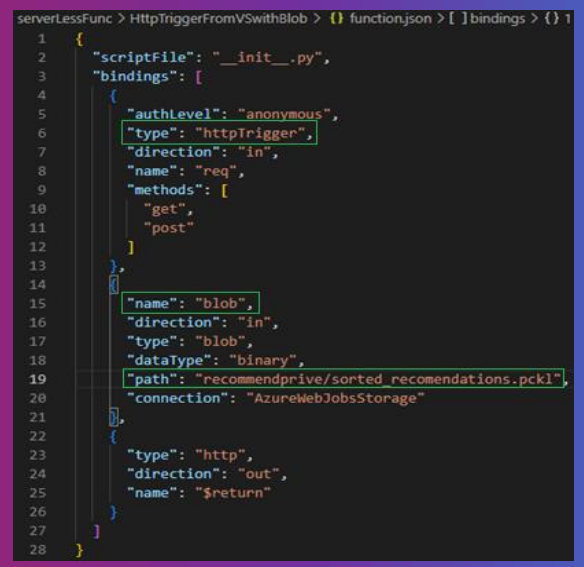
1/ AZURE CLOUD





2/ VS CODE + EXTENSION AZURE





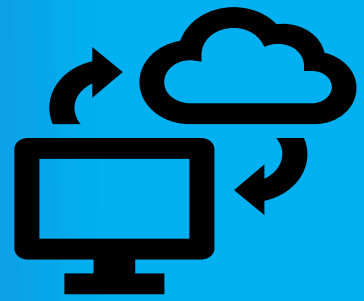

```
def main(req: func.HttpRequest, blob: func.InputStream) -> func.HttpResponse:
    user_id = req.params.get('user_id')
    pickled_data = blob.read()
    sorted_recomendations = pickle.loads(pickled_data)
    recomendations_to_display = {}
    for key, value in sorted_recomendations[user_id].items():
        recomendations_to_display['Article N° ' + str(key)] = '(similiraty score = ' + str(round(value, 3)) + ')'

    reco_json = json.dumps(recomendations_to_display, indent = 4)
    return func.HttpResponse(reco_json)
```

3/ DEPLOIEMENT

Via VS CODE

4/ AZURE CLOUD (test)



5 / SERVICE DISPONIBLE VIA L'URL :

https://fromlocalvs.azurewebsites.net/api/HttpTriggerFromVSwithBlob?user_id=

Pour consommation par le client ➔ Diapo suivante

Inférence

DÉPLOIEMENT : API / CLIENT + DEMO

5/ Client

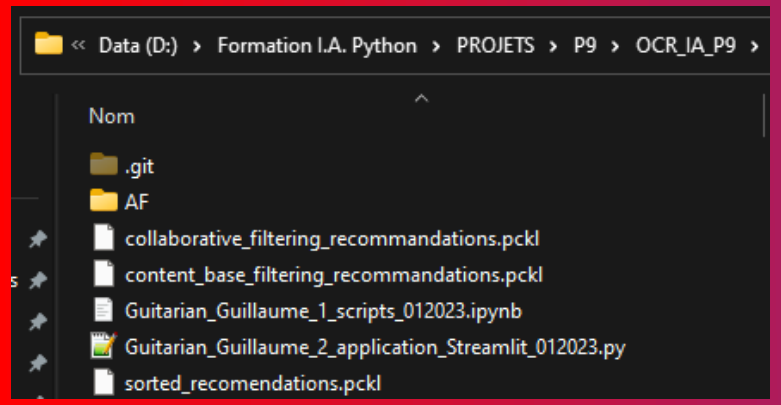
- Librairie Streamlit
- Choix de l'utilisateur
- Appel à l'Azure fonction avec **user_id** en paramètre
- Lecture du retour de l'Azure fonction
- Affichage des recommandations

DEMO

SAUVEGARDE ET VERSIONNING

GIT: AJOUTER DU CODE LOCAL A GITHUB

DOSSIER LOCAL SUR L'ORDINATEUR



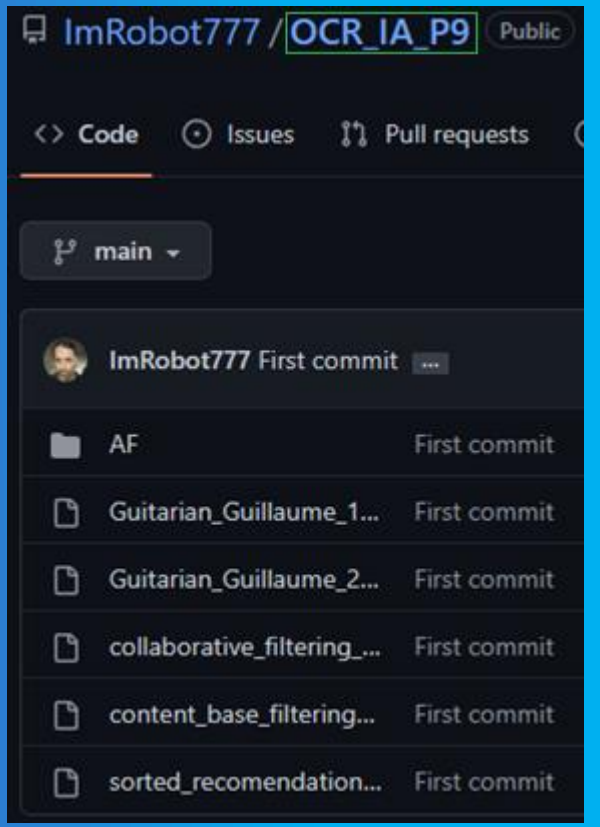
CONSOLE GITBASH DANS LE DOSSIER LOCAL

```
$ git init -b main
$ git add .
$ git commit -m "First commit"
$ git remote add origin <REMOTE_URL>
```

<REMOTE URL> = URL du REPOSITORY sur GITHUB

```
$ git push origin main
```

GITHUB REPOSITORY : BRANCH MAIN



Synthèse

- Une recommandation de contenu a pu être réalisée:
  - En isolant la partie la plus qualitative des données
  - En élaborant un moteur de type Content Based Filtering et Collaborative Filtering from scratch
  - En combinant les 2 approches

- Nous avons également pu faire une requête API de prédiction:
  - En élaborant une fonction Azure qui se déclenche via un appel HTTP et qui utilise un blob storage
  - En élaborant une application web client via streamlit qui exploite l'API (fonction Azure)

Perspectives

- Points d'amélioration :
  - **Rating** : Nous aurions pu peut-être obtenir de meilleurs résultats en prenant en compte le nombre de mots des articles et non pas uniquement le nombre de lecture * temps de session moyen
  - **Content base filtering** : Même si non utilisé en soit, nous aurions pu éviter (comme nous l'avons d'ailleurs fait pour le collaborative filtering) de recommander des articles déjà lus.
  - **MISE A JOUR** : Intégration continue via des **crons** qui :
    - Mettrait à jour les tables de sessions utilisateur et articles
    - Referait tourner les moteurs/modèles avec les nouvelles données
    - Regénérerait le fichier pickle de recommandations
    - Ferait une mise à jour du blob storage dans Azure avec ce fichier
- Limites :
  - La qualité du jeu de données étant assez faible (uniquement environ 2000 articles lus plus d'1 fois par le même utilisateur) ➔ difficile d'avoir un score de **Rating** qualitatif à fournir aux modèles de la librairie SURPRISE et donc d'avoir de bons modèles dans ces conditions !
  - Calculs des moteurs CUSTOM très longs ! (cosin_sim custom plus rapide que celui de scikit-learn)