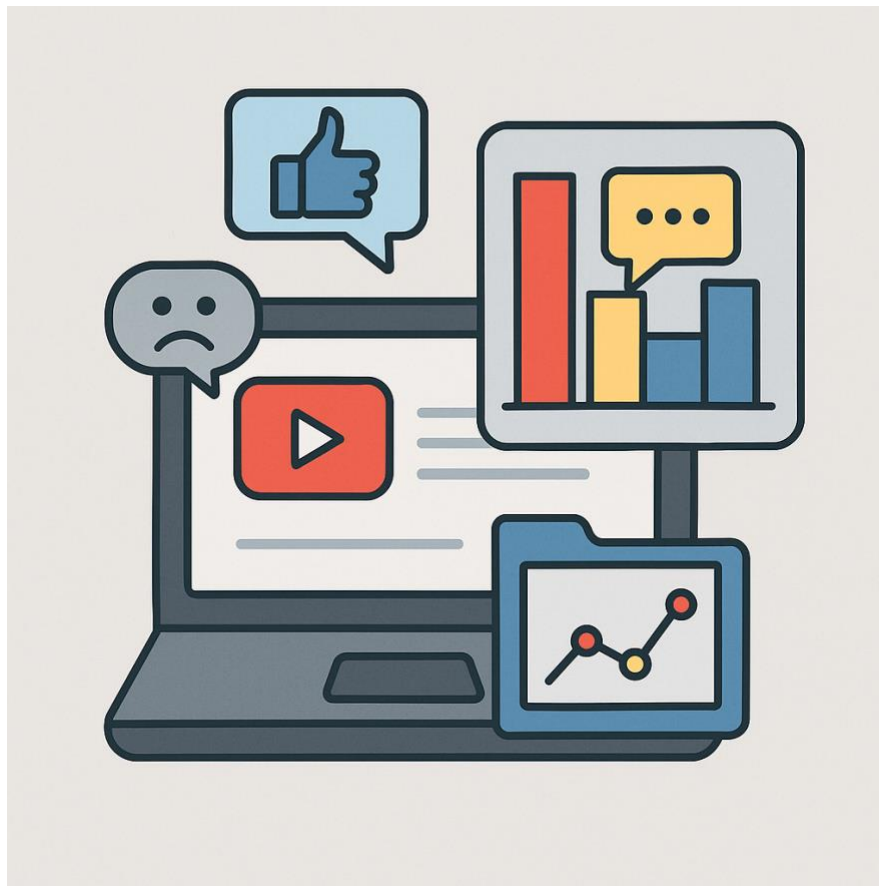


Titre RNCP 39586

**INGENIEUR EN SCIENCE DES DONNÉES SPECIALISÉ
EN APPRENTISSAGE AUTOMATIQUE**

**Bloc 1 :
COLLECTER TRANSFORMER T SÉCURISER DES DONNÉES**



**Sujet :
Développement d'un outil d'analyse automatisé des
commentaires YouTube**

**Auteur :
FURTADO LEAL Carla**

Sommaire

<i>I.</i>	<i>Collecte de données structurées et non structurées.....</i>	<i>3</i>
A.	Une stratégie de collecte de données.....	3
B.	Un exemple de collecte de données	4
C.	Une méthode d'automatisation de collecte	6
<i>II.</i>	<i>Stockage des données structurées et non structurées</i>	<i>7</i>
A.	Une stratégie de stockage et un modèle de données	7
B.	Une base de données et une solution de stockage Big Data	9
<i>III.</i>	<i>Structuration, transformation et enrichissement des données</i>	<i>9</i>
A.	Outils et technologies de traitement des données	9
B.	Transformation des données	11
C.	Processus ETL, automatisation et orchestration du traitement des données.....	12
<i>IV.</i>	<i>Sécurisation des données</i>	<i>13</i>
A.	Une politique de sécurisation des données	13
B.	Un schéma d'architecture de sécurité	15
	<i>Annexe</i>	<i>16</i>
	<i>Table des figures.....</i>	<i>33</i>

I. Collecte de données structurées et non structurées

A. Une stratégie de collecte de données

Pour rappel, nous cherchons à répondre à la question :

Comment valoriser les retours clients via l'analyse automatisé des commentaires YouTube ?

Pour ce faire nous avons décidé de développer un outil d'analyse automatisé des commentaires You Tube. Ce qui implique l'élaboration d'une stratégie de collecte de données textuelles, indispensable à l'analyse de sentiments et Topic Modeling. Le Topic modeling fonctionne sur la base de données non labellisées et détermine les sujets sous-jacents latents dans ces données en se basant sur la co-occurrence des mots. L'analyse de sentiment, lorsqu'elle fait appel à un modèle d'apprentissage automatique supervisé, nécessite des données d'entraînement, des commentaires labellisés. Si on mène cette analyse en utilisant des modèles pré-entraînés, des lexiques ou des règles, aucune donnée n'est nécessaire, sauf dans le cas où on fait du fine tuning.

Ainsi, notre stratégie de collecte, doit nous permettre de construire et d'incrémenter une base de données de commentaires selon les besoins d'analyse et de construire un corpus d'entraînement représentatif pour les performances et la pertinence des analyses, basées sur des modèles d'apprentissage supervisé.

La collecte doit être ciblée et automatisée sur les vidéos renseignées par l'utilisateur, il ne s'agit pas de collecter tous les commentaires existant sur la plateforme.

Les données utiles et nécessaires appartiennent donc à deux catégories :

- **Les données d'analyses** : commentaires YouTube extraits via l'API officielle
- **Les données d'entraînement** : corpus de commentaires ou textes courts français labellisé

Pour une vidéo donnée, nous interrogeons l'API YouTube V3 de Google Cloud, pour récupérer diverses métadonnées relatives à une vidéo donnée, dont les commentaires. Cette source nous permet de garantir la fiabilité des données et la conformité réglementaire.

Il s'agit d'une source de données structurée, suivant le schéma de type JSON.

Lors de l'extraction seules les données utiles et nécessaires sont collectées :

- `Id` : l'identifiant unique du commentaire
- `channelId` : l'identifiant de la chaîne YouTube
- `publishedAt` : date de publication
- `textOriginal` : le texte du commentaire tel qu'il a été publié ou mis à jour.
- `likeCount` : le nombre de like du commentaire
- `videoId` : identifiant de la vidéo
- `authorChannelId` : l'identifiant de la chaîne de la personne qui écrit le commentaire

- `title` : titre de la vidéo
- `description` : description de la vidéo
- `commentCount` : le nombre de commentaires de la vidéo

Les identifiants permettent de rattacher chaque commentaire à une vidéo précise d'une chaîne donnée et de les identifier de manière unique. En ce qui concerne l'identifiant de la chaîne de l'utilisateur, elle est collectée uniquement pour supprimer le commentaire du youtubeur. A ces données nous ajoutons la date d'extraction et l'url de la vidéo.

Afin de ne collecter que les données pertinentes, nous avons mis en place un filtre pour s'assurer qu'il s'agit de vidéos françaises (dont les commentaires sont essentiellement rédigés en français) et avec plus de 200 commentaires, grâce à `commentCount`. Le titre et la description sont utilisés pour s'assurer qu'il s'agit bien d'une vidéo française, seul le titre est stocké. Ces données sont ensuite stockées pour être analysées.

Ensuite, l'analyse automatisée nécessite également des données d'entraînement pour le calibrage du modèle d'analyse de sentiment. Ainsi, le meilleur modèle sélectionné, sera appliqué sur les données réelles, les commentaires. Les données d'entraînement doivent être similaires aux données réelles : langue, longueur, vocabulaire ...

En effet, plus les données ont des caractéristiques similaires aux données réelles meilleurs seront les performances. Kaggle est une plateforme qui met à disposition des jeux de données relatifs à sujets divers et variés, pour des compétitions, recherche scientifiques... C'est donc sur cette plateforme que nous avons trouvé des données d'entraînement potentielles. Il s'agit de commentaires YouTube et de tweets traduits de l'anglais vers le français à partie d'un modèle pré-entraîné, ainsi que des reviews du site « Allociné ».

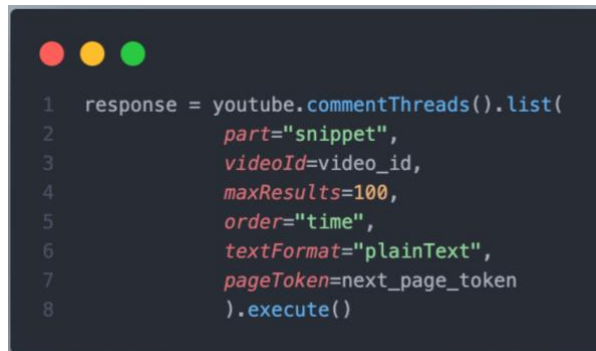
Le web scrapping a également été envisagé. Nous n'aurions pas été soumis à un quota comme c'est le cas lorsque nous utilisons l'API. Nous avons tout de même privilégié l'API pour des raisons réglementaires et pour gain de temps. En effet, le scrapping des commentaires n'est pas autorisé, de plus développer un code de scrapping peut prendre du temps et le scrapping en lui-même est chronophage. Notons que nous somme également dépendant de la structure même du site, si elle vient à changer il faudra nécessairement revoir le programme. Ainsi, l'API allie simplicité et gain de temps, mais implique le respect de quotas.

B. Un exemple de collecte de données

Nous avons sélectionné une vidéo d'un youtubeur populaire en France, dont les vidéos rencontrent généralement un fort engagement en termes de vues et de commentaires (plus de 200 commentaires).

Tout d'abord l'utilisation de l'API de google YouTube v3 nécessite d'avoir un compte Google cloud developer et un projet associé pour lequel nous avons généré des clés d'accès / des droit d'accès pour l'API.

L'interrogation de l'API se fait en précisant la ressource que l'on veut exploiter et la méthode à utiliser. Ici il s'agit de la ressource du fil de discussion, « commentThreads » exploité selon la méthode « list » pour extraire les données. Ensuite le paramètre « videoId », obtenu à partir de l'url, permet d'identifier la vidéo ciblée.



```
1 response = youtube.commentThreads().list(  
2     part="snippet",  
3     videoId=video_id,  
4     maxResults=100,  
5     order="time",  
6     textFormat="plainText",  
7     pageToken=next_page_token  
8 ).execute()
```

Figure 1: Exemple de structure d'appel d'API



```
1 # Ajouter les commentaires récupérés à la liste  
2 extraction_date = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())  
3 for item in response.get("items", []):  
4     comment_info = item["snippet"]["topLevelComment"]["snippet"]  
5     comments_data.append({  
6         "id": item["id"],  
7         "channelId": comment_info.get("channelId"),  
8         "videoId": comment_info.get("videoId"),  
9         "author": comment_info.get("authorDisplayName"),  
10        "publishedAt": comment_info.get("publishedAt"),  
11        "comment": comment_info.get("textDisplay"),  
12        "extractedAt": extraction_date  
13    })
```

Figure 2 : Extrait de code permettant de sélectionner les variables utiles

Le script *extraction.py* en annexe, permet d'interroger l'API via une classe qui prend l'url de la vidéo comme paramètre d'entrée.

Les commentaires récupérés sont initialement au format JSON, comme dans le schéma présenté précédemment. Nous récupérons une liste de dictionnaires correspondant chacun à un commentaire et aux métadonnées associées. Pour l'analyse, les données sont ensuite transformées au format dataframe.

En termes d'exhaustivité de la collecte, le nombre de commentaires affiché est plus important que le nombre de commentaires extraits. Cela s'explique par l'absence des réponses aux commentaires dans l'extraction. Il est possible de récupérer les commentaires en ajoutant le paramètre « replies » dans « part », mais nous avons de ne pas les récupérer. En effet, les réponses aux commentaires permettent de capter les interactions entre les spectateurs, qui tendent à s'éloigner du sujet de la vidéo. Or nous cherchons à analyser les « réactions directes » au contenu et non les interactions entre les individus qui risquent de biaiser nos résultats ou de les faire diverger.

De plus, les commentaires modérés par la plateforme ne sont pas collectés. Seuls les commentaires publiés et donc approuvés par la plateforme sont collectés et analysés. Ce qui assure la qualité des données. En effet, la plateforme garantit un premier niveau de qualité en filtrant de type de commentaires indésirables, évitant doublons (tous les commentaires ont un id unique qu'ils conservent en cas de mise à jour) et en gérant le format des données.

De la même façon, l'utilisation de l'API implique le respect et la conformité aux principes du règlement générale sur la protection des données.


En ce qui concerne les jeux de données d'entraînement issus de Kaggle, ils sont téléchargés directement sur le site et se présentent sous la forme d'un tableau. Il est nécessaire d'avoir un compte afin de télécharger.

C. Une méthode d'automatisation de collecte

Ici nous avons décidé d'opter pour une mise à jour planifier plutôt qu'en temps réel. La mise à jour en temps réel permet de garantir la cohérence des données et de l'analyse à tout moment, mais dans notre cas cela n'est pas optimal. En effet, cela impliquerait d'interroger en continue l'api, or les appels sont limités. La limite quotidienne est fixée à 10 000 unités par jours, sachant que la méthode « list » consomme 1 unité par appel. Nous pouvons dire que nous faisons du « quasi-temps réel » puisque dès lors que les données existent dans la base, elles sont mises à jour toutes les heures. D'autant plus que le moment où le nombre de commentaires évolue significativement c'est dans les premières heures / jours après la publication de la vidéo.

Lors de la première analyse, l'utilisateur entre une url dans l'application, ce qui conduit à une première extraction et à la création d'une base de données. C'est cette action qui automatise la collecte des données pour chaque analyse. Tout au long de la vie de la vidéo, les données associées sont automatiquement mises à jour, indépendamment du fait que l'utilisateurs interagisse avec l'application ou non. Ainsi, si la première extraction est soumise à une action, la mise à jour est totalement automatique. Elle s'appuie sur les différentes classent qui constituent le processus ETL. La première étape consiste à recenser l'ensemble des url analysés stockés dans nos bases de données. Ensuite le processus ETL est appelé pour chaque url.

Cette fonction est déployée dans Prefect flow, un framework de gestion de workflows data, via la commande suivante et tourne en parallèle de l'application :



```
1 .serve(name="mise_a_jour", schedule=Interval(timedelta(minutes=60), anchor_date=datetime(2025, 8, 16, 19, 0), timezone="Europe/Paris"))
```

Figure 3 : Extrait de code pour la programmation de la mise à jour

Ainsi, nous pouvons planifier son exécution tous les jours à intervalles d'une heure entre chaque exécution à partir d'une date donnée. Le code complet *synchronistaion.py* est disponible en annexe.

Les flows exécutés pendant les mises à jour sont monitorés via l'interface de prefect flow.

Il était initialement prévu pour la mise à jour de n'ajouter que les nouveaux commentaires dans les bases de données. Cependant, à chaque commentaire est associé à un nombre de likes qui peut évoluer et le commentaire lui-même peut être modifié ou supprimé. Par conséquent, à chaque mise à jour nous faisons une update de ce qui existe déjà et ajoutons les nouvelles données.

II. Stockage des données structurées et non structurées

A. Une stratégie de stockage et un modèle de données

La stratégie de stockage adoptée repose une architecture conçue pour répondre aux besoins du client souhaitant analyser son contenu. Chaque chaîne y est identifiée comme une entité indépendante, ce qui permet de compartimenter les données selon une approche documentaire. Plus simplement, il y a une base par chaîne YouTube et chacune contient un ensemble de collections qui recensent les données relatives à chaque vidéo. Cette approche garantit l'isolation entre les éventuels futurs utilisateurs et optimise l'accessibilité aux données. Elle s'inscrit dans une logique d'analyse séquentielle des vidéos, tout en conservant une certaine flexibilité pour des évolutions futures. Comme l'analyse de groupes de vidéos notamment pour les vidéos qui portent sur le même concept, les playlists. Dans cette éventualité nous pourrions regrouper tous les commentaires de plusieurs vidéos dans une base contenant 1 seule collection avec l'ensemble des documents. Cependant puisque nous sommes soumis à un quota pour la collecte cette option n'est pas encore possible.

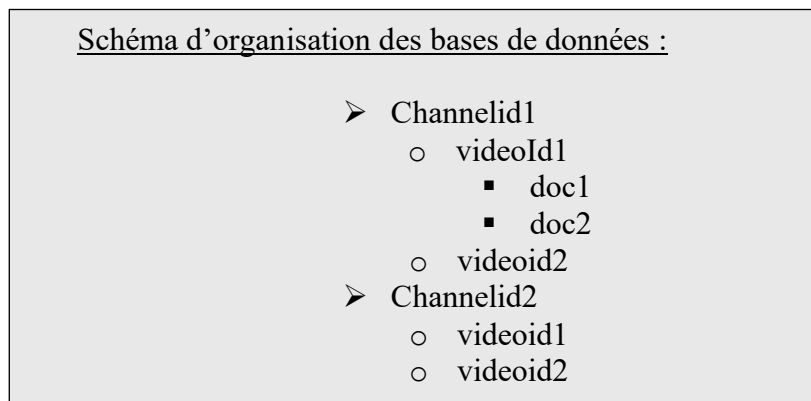
Cette organisation permet l'utilisation des données selon 2 axes :

- **Analytique** : la méthode de stockage permet de stocker des données brutes et de faire des enrichissements successifs de méta données et des résultats d'analyse (sentiments, futures métriques de YouTube, autres idées d'analyse...).
- **L'accessibilité** : les données sont disponibles et accessibles pour chaque utilisateur sans problème de partitionnement. En effet, puisque l'analyse est menée par un client sur son contenu, il n'est pas nécessaire et même exclu de pouvoir croiser les informations entre les différentes chaînes. Il est plus simple ainsi de sécuriser les données.

L'idée est que toutes les informations soient disponibles au même endroit pour une chaîne et une vidéo donnée.

Notre stratégie de stockage s'articule donc autour de plusieurs bases de données qui correspondent chacune à des chaînes différentes (toujours dans l'éventualité des plusieurs utilisateurs). Dans une base on trouve une collection avec autant de documents que de commentaires analysés, appartenant à une vidéo. C'est dans les documents que l'on trouve les données. Si une base contient plusieurs collections, cela signifie pour cette chaîne plusieurs vidéos ont été analysées.

Les bases sont nommées par rapport à l'identifiant de la chaîne correspondante et les collections par rapport à l'identifiant de la vidéo. Ainsi, l'organisation est illustrée par le schéma suivant :



La structure / le modèle de chaque document suit la structure d'un document JSON avec des clés et des valeurs comme le montre le schéma suivant :

<i>Nom de la clé</i>	<i>Exemple de valeur</i>
<i>Id</i>	Kjdchvdhv77 vb 67sdk
<i>Titre</i>	On a traversé la Méditerranée
<i>ChannelId</i>	Hbhjvhg67gcv58bhjv4454
<i>videoId</i>	Ld69n46V
<i>publishedAt</i>	2025-07-30T12:00:15Z
<i>Comment</i>	vous étiez dans ma ville natale 😞
<i>likecount</i>	0
<i>extractedAt</i>	2025-08-02T22:28:15Z
<i>Token_clean_lem</i>	['etre', 'ville', 'natal']
<i>Comment_clean_lem</i>	etre ville natal
<i>W2vec_vector</i>	Un vecteur
<i>Tfidf_vector</i>	Un vecteur
<i>Sentiment</i>	Neutre

En ce qui concerne la manipulation, les données sont essentiellement manipulées avant d'être stockées dans la base de données lors du processus ETL. Une fois stockées, elles se font via python grâce à un connecteur qui permet la connexion avec la base de données. Les seules manipulations effectuées sont l'écriture / mise à jour dans la base de données, la lecture pour l'analyse et l'agrégation pour l'affichage des résultats. Ce sont des manipulations simples et qui ne nécessitent pas de jointures.

B. Une base de données et une solution de stockage Big Data

Nous avons sélectionné MongoDB, un système de gestion de base de données (SGBD) orienté document (NoSQL), pour son affinité naturelle avec les documents JSON et la flexibilité du schéma qui permet l'enrichissement successif des collections et documents. En effet, dans notre cas il n'est pas nécessaire d'utiliser une base de données relationnelle car les chaînes YouTube sont indépendantes les unes des autres et un commentaire ne peut appartenir qu'à une seule chaîne. Nous n'avons donc pas besoin de faire des jointures, qui sont des opérations coûteuses. Même si une personne peut avoir plusieurs chaînes, les contenus sont généralement différents et donc une analyse croisée ne serait pas pertinente. De plus, les bases de données relationnelles sont soumises à des schémas de données stricts, incompatible avec des données textuelles hétérogènes.

Notons que dans le classement des solutions existantes, MongoDB est classé 5^{ème} en termes de popularité et est le seul SGBD orienté document (de manière native) sur les 10 premiers.

De plus, les données sont facilement manipulables avec Python grâce au connecteur officiel PyMongo. Il permet d'effectuer les opérations CRUD (create, read, update, delete) et les requêtes d'agrégation directement depuis le code Python.

Cette solution de stockage a été développée à partir de l'image Docker officielle de MongoDB (version 6). Nos bases sont donc hébergées dans un conteneur Docker accessible via le port "27017". Le conteneur est rattaché à un volume qui permet la persistance des données dans les bases même si le conteneur s'étend. L'accès à la base est également sécurisé par un nom utilisateur et un mot de passe.

Nous avons donc une solution de stockage Big Data qui fonctionne avec MongoDB conteneurisé et déployée via Docker pour garantir la portabilité et la reproductibilité.

III. Structuration, transformation et enrichissement des données

A. Outils et technologies de traitement des données

Le traitement des données textuelles s'appuie sur un ensemble de bibliothèques Python spécialisées dans le traitement de texte et l'apprentissage automatique. Nos données étant déjà structurées, seuls les traitements nécessaires à l'analyse sont effectués.

Dans un premier temps, afin d'analyser les commentaires, il est nécessaire d'appliquer certaines transformations au texte brut. Ces techniques de prétraitement sont déjà implémentées dans plusieurs bibliothèques comme Spacy et NLTK. Ces 2 bibliothèques sont assez complètes en termes de traitements de données (stopwords, modèles de words embedding...). Cependant, nous avons privilégié l'utilisation de Spacy pour la

majorité des traitements, pour sa rapidité et sa plus grande affinité avec la langue française par rapport à NLTK. Nous avons associé Spacy à d'autres bibliothèques :

- Re - regex pour optimiser le nettoyage avec le traitement d'expressions régulières (certains patterns comme les liens, les hastag, dates...)
- Unidecode pour convertir les caractères spéciaux en équivalent ASCII
- Langdetect permet de détecter la langue et de filtrer les données
- String : pour la ponctuation

Ensuite, pour les vectoriser, c'est-à-dire représenter le texte sous forme numérique, il y a 3 types de méthodes :

- Les méthodes bag of words (BOW)
- Les techniques d'embedding
- L'utilisation de modèles pré-entraînés

La méthode choisie dépend du type de modèle utilisé pour l'analyse de sentiment. Pour les transformers elle n'est pas nécessaire, mais pour les modèles de machine learning classique elle est indispensable. Elle permet de représenter le texte dans un espace numérique. Les méthodes de type BOW se basent sur la fréquence des mots et compte le nombre d'occurrence dans le corpus (l'espace de représentation). Les méthodes de type word embedding représentent chaque mot sous forme de vecteur et permettent de capturer les similitudes sémantiques, de conserver la sémantique. Pour pouvoir tester différents modèles d'analyse de sentiment, nous avons sélectionné 2 méthodes de vectorisation : le TF-IDF pour la catégorie BOW et Word2vec pour la catégorie word embedding. La méthode TF-IDF ou *terme frequency-inverse document frequency* permet d'intégrer l'importance du mot dans sa représentation numérique et ne se limite pas à un simple comptage. La méthode Word2vec se base sur des réseaux de neurones, pour vectoriser les mots. Ces 2 modèles sont respectivement disponibles dans les bibliothèques sklearn et gensim. Nous n'avons pas sélectionné FastText, un dérivé de Word2vec pour le word embedding car bien qu'il soit plus adapté pour les données issues de réseaux sociaux, il est plus lent et plus lourd.

Enfin, nous avons décidé d'intégrer l'analyse de sentiment en tant que transformation / enrichissement des données. Encore une fois, les méthodes et modèles nécessaires sont déjà implémentées dans certaines bibliothèques python. Là aussi, il y a plusieurs méthodes possibles :

- Méthode qui se base sur le lexique : Vader, textblob téléchargeable via pip
- Machine learning classique : tous disponible avec sklearn
- Modèles pré-entraînés de type transformers : disponible dans le catalogue d'hugging face
- Réseau de neurones

Ici notre choix sera déterminé par une analyse des performances des différents modèles. Toutefois, nous pouvons déjà dire que Vader n'est pas initialement adapté pour la langue française, mais s'il est toujours possible de traduire le texte, à la différence de Textblob. De plus, le choix du modèle peut également influencer le choix de la méthode de vectorisation.

Enfin, en ce qui concerne la manipulation de données nous avons privilégié pandas car nous utilisons des dataframes.

B. Transformation des données

L'ensemble des traitements sont détaillés dans le script *transformation.py* en annexe et se divisent donc en 3 parties.

- **Nettoyage textuel**

La première étape est le découpage du texte en token, ici en mots, tout en conservant l'ordre. Le token est notre unité d'analyse et est généralement identifié grâce aux espaces, séparateur naturel entre les mots. On filtre l'ensemble des tokens par rapport à une liste de mots et caractères indésirables qui comprend des stop-words, la ponctuation et les chiffres. Il s'agit de retirer les tokens qui ne portent aucune information tel que : *le, la, de, une, « , », ?, !...* On en profite également pour regrouper certaines expressions (ex : noms propres) qui sont porteuses de sens et en supprimer d'autres, tel que les url.

La seconde étape consiste à uniformiser nos tokens en mettant tout en minuscule, en retirant les accents et en réduisant certains mots à une racine. Cette dernière étape permet de réduire le nombre de tokens différents. En effet « Internet » et « internet » sont traités comme 2 tokens différents alors qu'il s'agit du même mot / de variables. En ce qui concerne la racinisation des mots, il existe 2 méthodes la « lemmatisation » qui réduit le mot à sa véritable racine et le « stemming » qui retire les terminaisons des mots, ce qui peut donner des mots qui n'ont pas de sens. Nous appliquons donc la lemmatisation qui réduit / regroupe les tokens sous une racine commune qui a du sens.

Une fois ces étapes terminées nous avons ajouté une étape qui filtre selon la taille du token, si le token ne contient que 3 caractères ou moins il est supprimé. Nous partons du principe qu'un token aussi court n'apportera pas de valeur à l'analyse.

Exemple de données transformées	
vous étiez dans ma ville natale 😞	etre ville natal

- **La vectorisation**

Comme expliqué précédemment, les méthodes TF-IDF et Word2vec produisent des matrices. Dans le cas du TF-IDF cette matrice est souvent très grande et avec beaucoup de 0 elle ne sera donc pas présentée dans ce document. De même pour Word2vec la présentation d'un vecteur exéplerait le document.

Chaque transformation ajoute une colonne dans la base MongoDB. Pour le topic modeling nous avons décidé de ne pas l'intégrer au pipeline ETL en tant que transformation.

C. Processus ETL, automatisation et orchestration du traitement des données

Notre processus ETL extrait, transforme et enrichie les données extraites de l'api, comme expliqué précédemment, puis charge les données dans mongodb. Il repose sur les scripts python *extraction.py*, *transformation.py* et *load.py* qui contiennent toutes les fonctions essentielles. C'est le script *etl.py* qui permet d'orchestrer les différentes étapes et qui est en réalité le flow principal exécuté lorsque l'on utilise l'outil. L'interface streamlit est le point d'entrée où l'utilisateur entre l'url d'une vidéo YouTube (qui appartient à sa chaine). Suite à cette action si les données relatives à la vidéo n'existent pas déjà, les étapes s'exécutent successivement. Cette condition permet de gagner du temps en interrogeant directement la base de données et si les données ont déjà été collectées. Le processus complet suit le schéma ETL automatisé suivant :

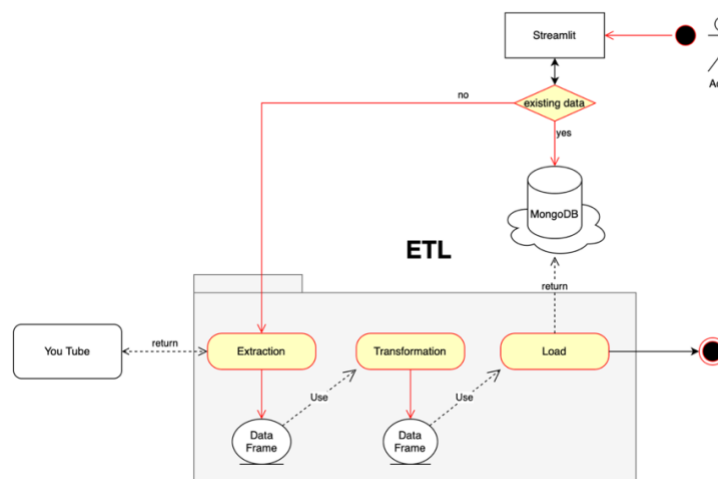


Figure 4 : Schéma de l'architecture de l'outil

Nous avons également programmé ce flow pour qu'il s'exécute toutes les heures de manière automatique et mette à jour la base de données.

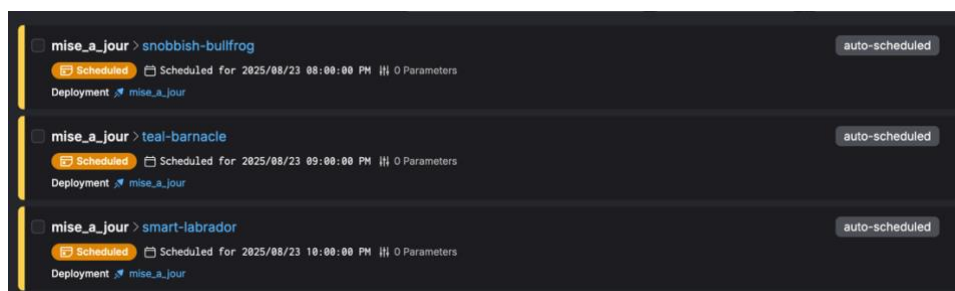


Figure 5 : Capture de l'UI de Prefect avec les mises à jour programmées

L'exécution programmée est implémentée grâce à Prefect dont l'interface utilisateur permet de suivre l'évolution des tâches via différents indicateurs :

- Le statut : late, running, failed, completed, cancelled
- L'heure et la date d'exécution

- Le temps d'exécution
- Les paramètres
- Les tâches associées

Le graphe d'ordonnancement des tâches permet de visualiser en temps l'évolution et le statut des tâches/ flow appelés pendant l'exécution. Dans notre cas l'extraction, la transformation et le chargement sont des flows qui font intervenir plusieurs tasks. Ainsi ce graphe permet d'identifier rapidement des causes d'échec.

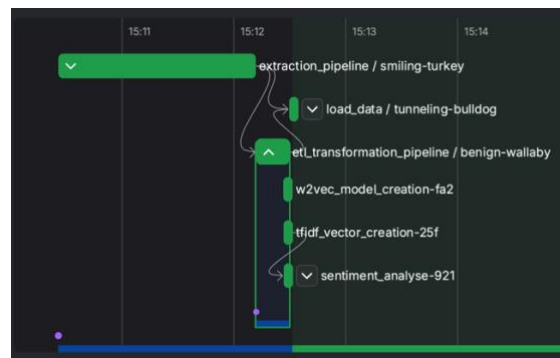


Figure 6 : Capture de l'UI de Prefect avec un exemple d'exécution de flow

IV. Sécurisation des données

A. Une politique de sécurisation des données

Les enjeux de sécurité sont inhérents au type de données traités et à l'architecture du projet. En effet, les commentaires sont des données « sensibles » dont le traitement doit être adapté en conséquence. Ils sont soumis à des contraintes légales (RGPD et conditions d'utilisation de l'API) auxquelles nous sommes soumis. Ainsi, notre politique de sécurité définit l'ensemble des mesures techniques et organisationnelles permettant de garantir l'intégrité du système (de bout en bout) et la confidentialité des données.

Sécurité dans les données

Afin de préserver l'identité des auteurs de commentaires, nous avons limité la collecte des données aux données strictement nécessaires. Donc la base de données ne contient aucune donnée d'identification (tel que le pseudo), mis à part l'identifiant du commentaire. De plus, étant donné le caractère sensible des données il est essentiel de mettre en place une modération pour supprimer les commentaires jugés trop sensibles. Cette fonction de modération est déjà assurée par YouTube en tant que plateforme hébergeant les commentaires.

Sécurité dans l'usage

Dans la même logique, les dashboard de résultats ne seront en mesure d'afficher que quelques commentaires et en aucun cas les données d'identification ne seront présentées ou accessibles pour l'utilisateur. Dans le même temps, l'analyse se fait au niveau macro, c'est-à-dire qu'il n'est pas possible de cibler un individu ou un groupe/minorité pour en faire l'objet de l'analyse. Les conditions générales d'utilisations ou une charte d'utilisation sera soumise à l'approbation de l'utilisateur lors de sa première utilisation de l'application. Elle présentera les mises en garde vis-à-vis de l'utilisation de ces données et les règles à respecter. Notamment que les données ne doivent être utilisées uniquement à des fins professionnelles, pour l'orientation stratégique de l'activité de l'utilisateur.

Sécurité dans le stockage

De plus, puisque les données sont mises à jour toutes les heures elles restent conformes à la réglementation RGPD tant YouTube la respecte. Si les données sont mises à jour sur YouTube elles le seront également dans nos bases. Il y a une mise à jour horaire qui garantit que les données ne sont pas stockées plus que nécessaire.

Sécurité des flux

L'ensemble des flux de données, notamment entre l'application streamlit et la base de données, doivent également être sécuriser. D'abord en s'assurant que ce qui est entré dans l'application est bien un url You Tube valide. Pour cela nous avons mis en place une vérification qui bloque l'analyse si l'url n'est pas valide. Ensuite, nos données transitent entre 3 zones distinctes : la base de données, l'application et la zone de traitement des données. La mise en place d'une api gateway accessible via token entre le différent service permettrait de sécuriser les flux. Cette mesure ne sera pas mise car un peu excessive au regard de la nature données que nous traitons. En effet, nous ne stockons aucune information permettant d'identifier les émetteurs des commentaires. Néanmoins nous pouvons mettre en place un système d'alerting en cas d'échec de l'exécution du pipeline ETL.

Sécurité dans l'organisation

De plus, l'architecture de l'outil doit également garantir que seul l'auteur du contenu soit en mesure de lancer les analyses et accéder aux résultats. Donc la mise en place d'une authentification au niveau de l'interface streamlit. En ce qui concerne la sécurisation de l'architecture, elle passe par la création des rôles et de droits différents. L'application streamlit à la base de données avec uniquement des droits de lecture peut l'utilisateur. Seule la fonction de mise à jour a le droit d'écrire dans la base de données. Afin de limiter les accès et les actions possibles. Dans cette logique, l'ensemble des activités doit être surveillée et consignées. Ajoutons que le temps d'utilisation de la session streamlit est limité.

B. Un schéma d'architecture de sécurité

Le schéma suivant illustre les mesures de sécurité appliquées.

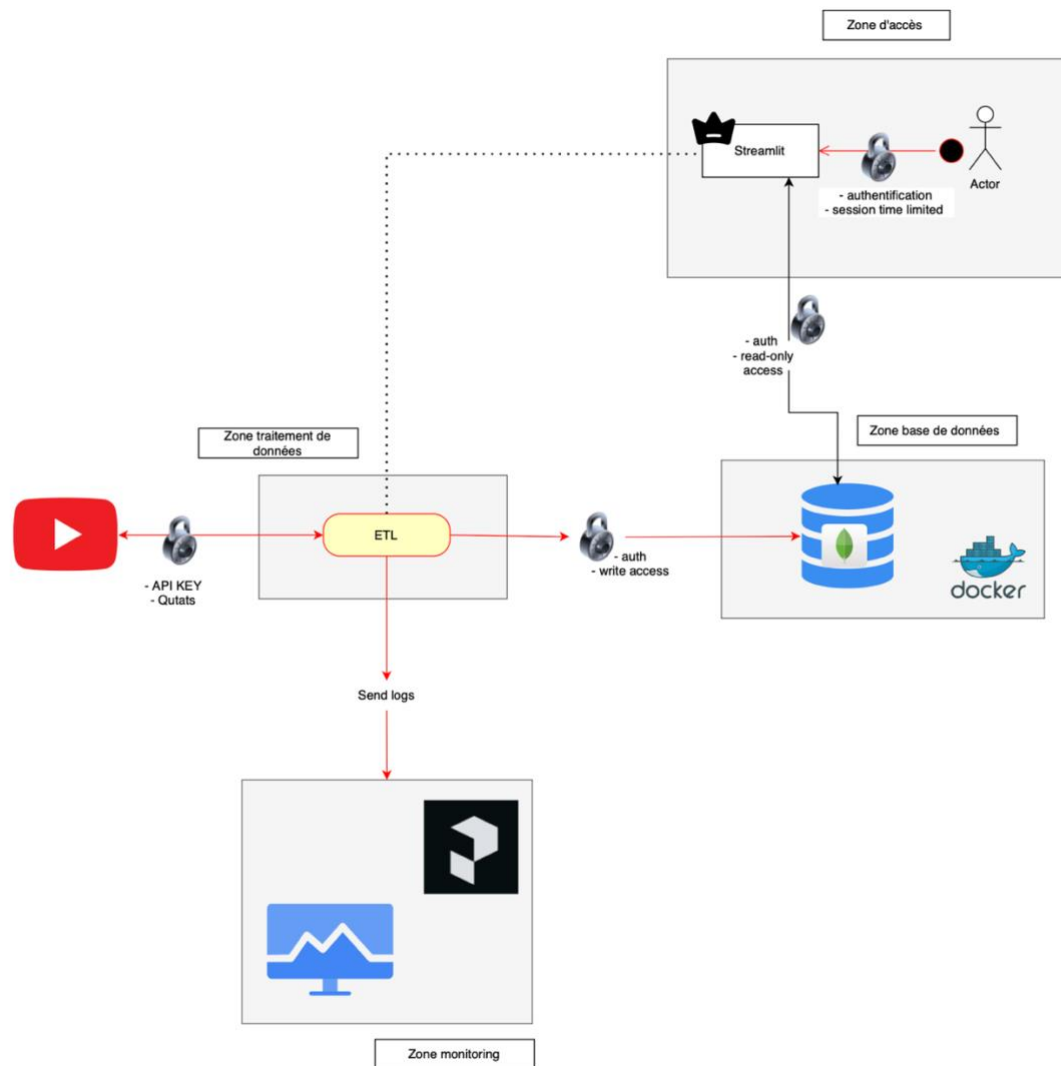


Figure 7 : Schéma de l'architecture sécurité

Annexe

[1] *extraction.py*

```
""" _____ Library
_____ """

import re
import os
import time
import pandas as pd
from dotenv import load_dotenv
from prefect import flow, task
from googleapiclient.discovery import build
from langdetect import detect
from .load import Load
from prefect.logging import get_run_logger

scopes = ["https://www.googleapis.com/auth/youtube.readonly"]
load_dotenv()

""" _____ Fonctions
_____ """

class Extraction :
    def __init__(self, video_url:str = ""):
        self.api_key = os.getenv("DEVELOPER_KEY")
        self.video_url = video_url
        #self.last_comment = last_comment # id du commentaire le plus récent
        self.video_id = self.url2id()
        self.channel_id = None
        self.existing_comments_id = None

    # récupérer l'id de la vidéo à partir de l'URL
    def url2id(self):
        pattern = r'^((https?:\V)?(www\.)?(youtube\.com\/(watch?v=|embed\/|v\/)|youtu\.be\/)([w-]{11})(\S+)?)$'
        if re.match(pattern, self.video_url) :
            print("Valid YouTube URL")
        else:
            raise ValueError('Error : invalid YouTube URL')

        video_id = self.video_url.split("v=")[-1][:11]
        return video_id
```



```

# récupère le secret id pour l'appel d'API : faire os.getenv('DEVELOPER_KEY') ou lire dans un fichier
# def get_key(self):
#     with open('secret_clientid.txt', 'r') as file:
#         DEVELOPER_KEY = file.read()
#     return DEVELOPER_KEY

@task(name='get_data_task', description="Tâche d'extraction des données YouTube")
# appel de l'API
def get_data(self):
    #logger = get_run_logger()

    api_key = self.api_key # .get_key() #
    youtube = build("youtube", "v3", developerKey=api_key)

    # Paramètres initiaux pour la requête
    video_id = self.url2id()
    comments_data = []
    next_page_token = None

    # Récupérer les infos de la vidéo
    try:
        video_response = youtube.videos().list(
            part="snippet,statistics",
            id=video_id
        ).execute()

        video_info = video_response["items"][0]["snippet"]
        self.channel_id = video_info["channelId"]
        self.exisitng_comments_id = Load().check_exisitng_data(self.channel_id, self.video_id)

    except Exception as e:
        raise RuntimeError(f"Impossible de récupérer les infos de la vidéo : {e}")

    # Vérifier la langue de la chaîne
    try:
        video_title = video_info["title"]
        video_description = video_info["description"]
        lang = detect(video_title + " " + video_description)
        print(f"Langue de la chaîne : {lang}")

```

```

    if lang != "fr":
        raise ValueError(f"Langue détectée : {lang.upper()}, ce script ne traite que les chaînes françaises.")
except Exception as e:
    raise RuntimeError(f"Impossible de vérifier la langue de la chaîne : {e}")

# Vérifier le nombre de commentaires
try:
    video_nb_comments = int(video_response["items"][0]["statistics"]["commentCount"])
    if video_nb_comments < 200:
        raise ValueError("Error : not enough comments")
except Exception as e:
    raise RuntimeError(f"Impossible de récupérer le nombre de commentaires : {e}")

# Étape 3 – Collecte des commentaires
print("Langue valide \n"
      "Nombre de commentaires suffisant \n"
      "Récupération des commentaires en cours...")

while True:
    response = youtube.commentThreads().list(
        part="snippet",
        videoId=video_id,
        maxResults=100,
        order="time",
        textFormat="plainText",
        pageToken=next_page_token
    ).execute()

    # Ajouter les commentaires récupérés à la liste
    extraction_date = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
    for item in response.get("items", []):
        # logger.info(f'on compare : {item["id"]} et {self.last_comment}')
        comment_info = item["snippet"]["topLevelComment"]["snippet"]
        # if item["id"] not in self.existing_comments_id:
        # Vérifier si le commentaire a déjà été extrait :
        if comment_info.get("channelId") == comment_info.get("authorChannelId"):
            continue

```

```

        comments_data.append({
            "url": self.video_url,
            "id": item["id"],
            "titre": video_title,
            "channelId": comment_info.get("channelId"),
            "videoid": comment_info.get("videoid"),
            "author": comment_info.get("authorDisplayName"),
            "publishedAt": comment_info.get("publishedAt"),
            "comment": comment_info.get("textOriginal"),
            "likeCount": comment_info.get("likeCount"),
            "extractedAt": extraction_date
        })

    # Vérifier s'il y a une page suivante
    time.sleep(5)
    next_page_token = response.get("nextPageToken")
    if not next_page_token:
        break
    time.sleep(1)

    print(f'data uploaded')
    return comments_data

# Fonction pour créer un DataFrame à partir des données collectées
def get_data_table(self) -> pd.DataFrame:
    # Création du DataFrame à partir des données collectées
    df = pd.DataFrame(self.get_data())
    # print(df.head())
    print(f'Total de commentaires récupérés : {df.shape[0]}')
    return df

@flow(name='extraction_pipeline', description="Pipeline d'extraction des données YouTube")
def main_extraction(self):
    df = self.get_data_table()
    return df, self.video_id, self.channel_id

"""
_____ Main
_____
"""

if __name__ == "__main__":

```

```
data, id_video, id_channel = Extraction.main_extraction()

# data = get_data_table(get_data())
```

[2] *transformation.py*

```
""" _____ Library
                                     """

import spacy
import re
import json
import pandas as pd
import numpy as np
import nltk
import pickle
from nltk.stem.snowball import SnowballStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
import string
from nltk.probability import FreqDist
import matplotlib.pyplot as plt
from unidecode import unidecode
from gensim.models import Word2Vec
from gensim.utils import simple_preprocess
from prefect import task, flow, get_run_logger
from transformers import pipeline

# from prefect.cache_policies import NO_CACHE

nlp = spacy.load("fr_core_news_sm")
stopwords = list(nlp.Defaults.stop_words)
punctuation = list(string.punctuation)
s_stemmer = SnowballStemmer("french")

""" _____ nettoyage
                                     """

# ÉCRIRE UN FICHER AVEC LES EXTRAS STOPWORDS ET LES EXPRESSIONS que l'utilisateur pourra
agrémenter :
```

```

# appel de la fonction s'il y a un input, ouverture et écriture des lignes supplémentaires, sauvegarde du document
pour utilisation

## @task(name='expressions_frequentes', cache_policy=NO_CACHE, description="Remplace les expressions
fréquentes par leur version normalisée")
def expressions_frequentes(text, path='extra_expressions.txt'):
    with open(path, 'r') as file:
        expressions = json.load(file)
    for key, value in expressions.items():
        text = re.sub(key, value, text.lower())
    return text

## @task(name='reduire_repetitions', cache_policy=NO_CACHE, description="Réduit les répétitions de lettres
dans les mots")
def reduire_repetitions(mot):
    """
    nettoyage des répétitions de lettres
    On remplace les répétitions de lettres par une seule occurrence
    Ex: "loooove" devient "love"
    """
    return re.sub(r'(\.){2,}', r'\1', mot)

# # @task(name='preprocessing_task', description="Tâche de prétraitement des données textuelles")
def preprocessing(text, join=True, methode='lemma', extra_stopwords : list = None, extra_punctuation : list =
None, extra_pattern : re.Pattern=None, path='extra_expressions.txt'):
    if not isinstance(text, str):
        raise ValueError(f"Expected a string, but got {type(text)} : {text}")
    extra_stopwords = extra_stopwords or []
    extra_punctuation = extra_punctuation or []
    nondesiredtokens = set(punctuation + stopwords + extra_punctuation + extra_stopwords)

    text = expressions_frequentes(text, path=path) # Remplacer les expressions fréquentes par leur version
normalisée

    # pattern = re.compile(r"(http://\S+|# @\S+|\.\d.*|\#.*)")
    pattern = re.compile(r'http.+')

    # Tokenisation et nettoyage en une seule passe
    if methode == 'lemma':
        tokens = [unicode(pattern.sub("", token.lemma_lower())) for token in nlp(text) if not token.is_stop and not
token.is_digit and token.text.lower() not in nondesiredtokens and len(token) > 3]

```

```

elif methode == 'stem':
    tokens = [unicode(pattern.sub("", s_stemmer.stem(token.text.lower())) for token in nlp(text) if not
token.is_stop and not token.is_digit and token.text.lower() not in nondesiredtokens and len(token) > 3]

# Filtrer les tokens vides après suppression des liens/mentions
cleaned_tokens = [reduire_repetitions(token) for token in tokens if len(token)>3]
# re.sub(r"\-{1,}|\{1,}|\V{1,}|\_{1,}|\~{1,}|\^{1,}|\?{1,}|\={1,}|\:{1,}|\|{1,}|\{1,}'|'",

return " ".join(cleaned_tokens) if join else cleaned_tokens

"""
_____ Vectorisation
_____

@task(name='w2vec_model_creation', description="Création du modèle Word2Vec à partir des données
prétraitées")
def make_w2vec_model(df:pd.DataFrame, text='comment_clean_lem'):
    """
    Obtenir le vecteur d'un mot spécifique
    """
    try:
        corpus = df[text].dropna().tolist()
        tokenized_corpus = [simple_preprocess(sent) for sent in corpus]
        model_w2v = Word2Vec(sentences=tokenized_corpus, vector_size=100, window=5, min_count=1, sg=1)
        return model_w2v
    except KeyError:
        raise ValueError("Error : le modèle Word2Vec n'a pas été entraîné correctement.")

def get_sentence_vector(text, model):
    words = simple_preprocess(text)
    word_vectors = [model.wv[word] for word in words if word in model.wv]
    if len(word_vectors) > 0:
        return np.mean(word_vectors, axis=0) # moyenne des vecteurs
    else:
        return np.zeros(model.vector_size)

# # @task(name='w2vec_vector_creation', description="Extraction des vecteurs de phrases à partir du modèle
Word2Vec")
def get_w2vec_vector(df, text='comment_clean_lem'):

```

```

"""
Obtenir le vecteur d'une phrase en utilisant le modèle Word2Vec
"""

try:
    model = make_w2vec_model(df, text=text)
    df['w2vec_vector_np'] = df[text].apply(lambda x: get_sentence_vector(x, model))
    df['w2vec_vector'] = df['w2vec_vector_np'].apply(lambda x: x.tolist() if isinstance(x, np.ndarray) else x)
    df.drop(columns=['w2vec_vector_np'], inplace=True) # Optionnel : supprimer la colonne intermédiaire

    return df
except ValueError as e:
    raise ValueError(f"Erreur lors de la création du modèle Word2Vec : {e}")

@task(name='tfidf_vector_creation', description="Extraction des vecteurs TF-IDF à partir des données
prétraitées")
def get_tfidf_vector(df, text='comment_clean_lem'):
    """
    Obtenir le vecteur TF-IDF d'une phrase
    """

    if not all(isinstance(x, str) for x in df[text].dropna()):
        raise ValueError(f"La colonne '{text}' ne contient pas que des chaînes de caractères")

    vectorizer = TfidfVectorizer()
    corpus = df[text].dropna().tolist()
    tfidf_matrix = vectorizer.fit_transform(corpus)
    df['tfidf_vector_np'] = list(tfidf_matrix.toarray())
    df['tfidf_vector'] = df['tfidf_vector_np'].apply(lambda x: x.tolist() if isinstance(x, np.ndarray) else x)
    df.drop(columns=['tfidf_vector_np'], inplace=True) # Optionnel : supprimer la colonne intermédiaire
    return df

""" _____ analyse
_____ """

@task(name='sentiment_model_creation', description="Création du modèle d'analyse des sentiments")
def get_sentiment_model(path='/Users/carla/Desktop/GitHub/Projet-RNCP/src/utls/bestmodel.pkl'):
    # with open(path, 'rb') as fichier_modele:
    #     model = pickle.load(fichier_modele)

    model = pipeline(
        "sentiment-analysis",
        model="cardiffnlp/twitter-xlm-roberta-base-sentiment-multilingual"
    )

```

```

return model

@task(name='sentiment_analyse', description="Analyse des sentiments des commentaires")
def get_sentiment(df,model, text='comment'):
    model = get_sentiment_model()
    # df['sentiment'] = df[text].apply(lambda x: model.predict([x])[0])
    df['sentiment']= df[text].apply(lambda x: model(x)[0].get('label'))
    df['sentiment_score'] = df[text].apply(lambda x: model(x)[0].get('score'))
    return df

""" _____ Main
_____ """

@flow(name='etl_transformation_pipeline', description="Pipeline de transformation des données pour l'ETL")
def main_transformation(df, comment = 'comment', path='extra_expressions.txt'):
    logger= get_run_logger()
    try:
        # df, video_id, channel_id = Extraction.main_extraction() # Assuming main() returns a DataFrame with a
        # 'comment' column

        df = df.dropna(subset=[comment]) # Drop rows where 'comment' is NaN
        df['tokens_clean_lem'] = df[comment].apply(lambda x: preprocessing(x, join=False, path=path))
        df['comment_clean_lem'] = df[comment].astype(str).apply(lambda x: preprocessing(x, join=True, path=path))
        logger.info("Nettoyage des données terminée avec succès.")

        # Vectorisation
        df = get_w2vec_vector(df, text='comment_clean_lem')
        logger.info("Vectorisation w2vec des données terminée avec succès.")

        df = get_tfidf_vector(df, text='comment_clean_lem')
        logger.info("Vectorisation tfidf des données terminée avec succès.")

        # Sentiment Analysis
        df = get_sentiment(df,model=None, text='comment') # Remplacez 'model' par votre modèle de sentiment
        return df # video_id, channel_id
    except Exception as e:
        logger.error(f"Erreur lors de la transformation des données : {e}")
        raise e

# if __name__ == "__main__":
#     df = pd.read_csv('database.csv',parse_dates=['publishedAt','extractedAt'])

```



```

# df = main_extraction() # Assuming main() returns a DataFrame with a 'comment' column

# # pattern = re.compile(r"(http://\S+|# @\S+|\d.*\d.*\#.*)")

# df['tokens_clean_lem'] = df['comment'].astype(str).apply(lambda x: preprocessing(x, join=False,
extra_stopwords=['vidéo', 'vidéos', 'video', 'videos', '.', 'jsuis', 'hey', 'faire'], extra_punctuation=[]))

# df['comment_clean_lem'] = df['comment'].astype(str).apply(lambda x: preprocessing(x, join=True,
extra_stopwords=['vidéo', 'vidéos', 'video', 'videos', '.', 'jsuis', 'hey', 'faire'], extra_punctuation=[]))

# df, video_id, channel_id = Extraction.main_transformation()

# Sauvegarder le DataFrame transformé

# df.to_csv('transformed_comments.csv', index=False)

```

[3] load.py

```

"""
_____ Library
_____

from pymongo import MongoClient
import json
import pandas as pd
import os
from dotenv import load_dotenv
from prefect import flow, task
from prefect import get_run_logger
from pymongo import UpdateOne
# from .transformation import main_transformation
# from .extraction import Extraction

load_dotenv()

"""
_____ Fonctions
_____

class Load:
    def __init__(self, maj=False):
        """
        Initialisation de la classe Load.
        """

        self.maj = maj # Indique si c'est une mise à jour ou un chargement initial
        self.username = str(os.getenv('MONGO_USERNAME'))
        self.password = str(os.getenv('MONGO_PASSWORD'))
        self.host = str(os.getenv('MONGO_HOST'))
        self.port = str(os.getenv('MONGO_PORT'))
        self.authSource = str(os.getenv('MONGO_AUTH_SOURCE'))

```

```

@task(name='authentification_task', description="Tâche d'authentification pour la connexion à MongoDB")
def authentification(self):
    """
    Fonction pour récupérer les informations d'authentification depuis un fichier de configuration.
    """
    try:
        #with open('/Users/carla/Desktop/GitHub/Projet-RNCP/auth.json', 'r') as file:
        #lines = json.load(file)

        username = self.username # Encodage du nom d'utilisateur lines['username'] #
        password = self.password # Encodage du mot de passe lines['password'] #
        host = self.host # Adresse du serveur MongoDB lines['host'] #
        port = self.port # Port du serveur MongoDB lines['port'] #
        authSource = self.authSource # Base de données d'authentification lines['authSource'] #
        return username, password, host, port, authSource
    except FileNotFoundError:
        raise FileNotFoundError("Le fichier de configuration est introuvable.")

@task(name='data_base_connexion_task', description="Tâche de connexion à la base de données MongoDB",
retries=3, retry_delay_seconds=5)
def data_base_connexion(self):
    """
    Fonction pour se connecter à la base de données MongoDB.
    """

    username, password, host, port, auth_db = self.authentification()
    # Création de l'URI de connexion MongoDB
    uri = f"mongodb://{username}:{password}@{host}:{port}?authSource={auth_db}"
    print(f"Connecting to MongoDB at {uri}")
    client = MongoClient(uri)
    # Vérification de la connexion
    try:
        client.admin.command('ping') # Ping pour vérifier la connexion
        print("Connexion à MongoDB réussie.")
        return client
    except Exception as e:
        raise RuntimeError(f"Erreur de connexion à MongoDB : {e}")

```

```

def data_base_deconnexion(self, client):
    """
    Fonction pour fermer la connexion à la base de données MongoDB.
    Args:
        client (MongoClient): Instance de connexion à MongoDB.
    Returns:
        None
    """
    client.close()
    print("Connexion à MongoDB fermée.")

# @task(name='check_exisitng_data_task', description="Tâche de vérification de l'existence des données dans
MongoDB")
def check_exisitng_data(self, db_name:str, collection_name:str):
    """
    Vérifie si une collection existe déjà dans la base de données.
    Args:
        client (MongoClient): Instance de connexion à MongoDB.
        db_name (str): Nom de la base de données / id de la chaîne.
        collection_name (str): Nom de la collection à vérifier (identifiant de la vidéo).
    Returns:
        None si la collection n'existe pas.
        une liste d'IDs si la collection existe.
    """
    client = self.data_base_connexion()
    # Lister les bases de données
    try:
        # Lister les bases de données
        db_list = client.list_database_names()

        if db_name not in db_list:
            print(f"Il n'existe pas encore de données pour cette chaîne : {db_name}.")
            self.data_base_deconnexion(client)
            return None

        # Lister les collections dans la base de données spécifiée
        collection_list = client[db_name].list_collection_names()

        if collection_name not in collection_list:
            print(f"La collection '{collection_name}' n'existe pas dans la base de données '{db_name}'.")

```

```

        self.data_base_deconnexion(client)

        return None

    print(f"La collection '{collection_name}' existe déjà dans la base de données '{db_name}'.")

    # Récupérer les IDs uniques de la collection
    ids = client[db_name][collection_name].distinct("id")

    # Récupérer les colonnes/champs disponibles
    # Méthode 1: Examiner un document échantillon pour obtenir les champs
    # sample_doc = client[db_name][collection_name].find_one()
    # columns = list(sample_doc.keys()) if sample_doc else []

    return ids#, columns

except Exception as e:
    print(f"Erreur lors de la vérification des données : {e}")
    return None

finally:
    self.data_base_deconnexion(client)

@flow(name='load_data', description="Chargement des données dans MongoDB")
def load(self, df, video_id:str, channel_id:str):
    logger = get_run_logger()

    client = self.data_base_connexion()
    # création de la base
    db = client[channel_id] # idenentifiant de la chaine
    # création de la table dans la base
    collection = db[video_id] # indentifiant de la video

    if self.maj == True:
        # logger.info("Mise à jour des données dans MongoDB...")
        comments_data = df.to_dict('records')
        operations = [UpdateOne(
            {"id": comment["id"]},
            {"$set": comment},
            upsert=True)
            for comment in comments_data

```

```

    ]

    if operations:
        collection.bulk_write(operations)
        self.data_base_deconnexion(client)

    else :
        # Insertion
        # logger.info("Insertion des données dans MongoDB...")
        collection.insert_many(df.to_dict(orient="records"))
        logger.info("Données insérées avec succès dans MongoDB.")
        self.data_base_deconnexion(client)

    # Mise à jour de la collection

""" _____ Main
_____ """

# if __name__ == "__main__":
#     Load.load()

# def check_existing_data(self, db_name:str, collection_name:str):

#     client = self.data_base_connexion()

#     try:
#         # Lister les bases de données
#         db_list = client.list_database_names()

#         if db_name not in db_list:
#             print(f"Il n'existe pas encore de données pour cette chaîne : {db_name}.")
#             return None

#         # Lister les collections dans la base de données spécifiée
#         collection_list = client[db_name].list_collection_names()

#         if collection_name not in collection_list:
#             print(f"La collection '{collection_name}' n'existe pas dans la base de données '{db_name}'.")
#             return None

```

```

#     print(f"La collection '{collection_name}' existe déjà dans la base de données '{db_name}'.")

#     # Récupérer la collection
#     collection = client[db_name][collection_name]

#     # Récupérer les IDs uniques
#     ids = collection.distinct("id")

#     # Récupérer les colonnes/champs disponibles
#     # Méthode 1: Examiner un document échantillon pour obtenir les champs
#     sample_doc = collection.find_one()
#     columns = list(sample_doc.keys()) if sample_doc else []

#     # Méthode alternative: Examiner plusieurs documents pour avoir tous les champs possibles
#     # (utile si tous les documents n'ont pas exactement les mêmes champs)
#     """
#     all_fields = set()
#     for doc in collection.find().limit(100): # Limite pour éviter de charger trop de données
#         all_fields.update(doc.keys())
#     columns = list(all_fields)
#     """

#     finally:
#         self.data_base_deconnexion(client)

```

[4]synchronisation.py

```

# Imports pour la configuration
import sys
sys.path.append("/Users/carla/Desktop/GitHub/Projet-RNCP")
from src.utils.load import Load
from datetime import datetime, timedelta
from prefect.schedules import Interval
from src.utils.extraction import Extraction
from src.Pipeline1.etl import main_etl
from prefect import flow, get_run_logger

# écraser l'ancienne base et la mettre à jours avec les nouvelles données

```

```

@flow(name='mise_a_jour', description="Pipeline de mise à jour des données YouTube", retries=1,
retry_delay_seconds=60)
def maj():
    # etape 0 : se connecter à la base de données MongoDB
    logger = get_run_logger()
    logger.info("Début du processus de mise à jour des données YouTube")
    loader = Load()
    client = loader.data_base_connexion()

    # etape 2 : récupérer les urls des vidéos de la base de données
    list_urls = []
    # id_lists = [] # pour stocker les derniers ids de commentaires

    for db_name in client.list_database_names():
        # logger.info(f"traitement base {db_name}")
        if db_name in ("admin", "config", "local", "test"): # filtrer les bases internes MongoDB
            # logger.info(f"bases trouvées {db_name}")
            continue
        db = client[db_name]
        # Lister toutes les collections de cette base
        for collection in db.list_collection_names():
            # logger.info(f"traitement de la collection {collection}")
            db_collection = db[collection]
            # Récupérer l'URL de la vidéo à partir de la collection
            video_url = db_collection.find_one({}, {"url": 1, "_id": 0})
            logger.info(f"taille de video_url {len(video_url)}")
            if len(video_url)>0:
                # logger.info(f"I url {video_url}, la collection {collection}, la base {db_name}")
                #last_comment_id = db_collection.find_one(sort=[("publishedAt", -1)], projection={"_id":1,"_id": 0}) #
dernier commentaire
                #logger.info(f"le dernier commentaire qui foire tout {last_comment_id}")

                list_urls.append(video_url['url'])

                #id_list = db_collection.distinct("id")
                #id_lists.append(id_list)
                # last_comment_ids.append(last_comment_id['id'])
            else:
                pass # ajouter un truc pour ajouter les info s'il n'y a pas d'url ajouter un paramètre a main_etl

```

```

loader.data_base_deconnexion(client)

for url in list_urls:

    logger.info(f"main_etl s'exécute avec les paramètres :{url}")
    main_etl(url, maj=True) # Appel de la fonction main_etl pour chaque URL

if __name__ == "__main__":
    maj.serve(name="mise_a_jour", schedule=Interval(timedelta(minutes=60), anchor_date=datetime(2025, 8, 23, 15, 0), timezone="Europe/Paris")) #cron="0 10 * * MON") # Lancer la mise à jour

```

[5] etl.py

```

import sys

sys.path.append("/Users/carla/Desktop/GitHub/Projet-RNCP") # Ajoute le répertoire parent au chemin de
recherche des modules

from src.utils.extraction import Extraction
from src.utils.transformation import main_transformation
from src.utils.load import Load
from prefect import flow, task
from prefect.logging import get_run_logger

@flow(name='main_etl_flow', description="Pipeline ETL principal pour l'analyse des vidéos YouTube")
def main_etl(url:str, last_id=None, with_channel_id:bool=False, maj=False):
    """
    """

    logger = get_run_logger()

    try:

        #logger.info("Début du pipeline ETL pour l'URL : %s", url)
        # last_id = None

        input_video_url = url

        # Étape 1 : Extraction des données
        extraction = Extraction(video_url=input_video_url)
        data, video_id, channel_id = extraction.main_extraction()
        logger.info(f"fin de l'extraction des {data.shape[0]} données")

        # Étape 2 : Transformation des données

```



```

transformed_data = main_transformation(data)

logger.info("Transformation des données terminée.")

# Étape 3 : Chargement des données
loader = Load(maj)
loader.load(transformed_data, video_id, channel_id)
logger.info("Chargement des données terminé.")
logger.info("Pipeline ETL terminé avec succès.")

if with_channel_id:
    return channel_id
except Exception as e:
    logger.info(f"Une erreur s'est produite dans le pipeline ETL : {e}")
    #raise ValueError(f"Erreur dans le pipeline ETL : {e}")

"""
_____ Main
_____
"""

# if __name__ == "__main__":
#     main_etl.serve(name="main_etl_flow", cron="0 10 * * MON") # Exemple d'URL YouTube

```

Table des figures

Figure 1 : Exemple de structure d'appel d'API.....	5
Figure 2 : Extrait de code permettant de sélectionner les variables utiles	5
Figure 3 : Extrait de code pour la programmation de la mise à jour	6
Figure 4 : Schéma de l'architecture de l'outil	12
Figure 5 : Capture de l'UI de Prefect avec les mises à jour programmées	12
Figure 6 : Capture de l'UI de Prefect avec un exemple d'exécution de flow.....	13
Figure 7 : Schéma de l'architecture sécurité.....	15