Salto

Tweets - Analyse de sentiments

[**Objectifs du projet**](#_hk309br4n8nt) **5**

[Objectif Principal](#_aj6biyxqr3mo) 5

[Objectifs Secondaires](#_9tlzznohqjqi) 5

[**API Twitter v2**](#_lglxinuuo7hb) **6**

[Présentation](#_s10g8l8z2euj) 6

[Recherche récente](#_ed4bj462l9qx) 7

[Recherche dans les archives complètes](#_p58t5tfky4hu) 7

[Principales fonctionnalités concernant la recherche de tweets](#_jri19jcel3p7) 8

[Timeline mention Endpoint](#_xpi0j4445mjr) 9

[Prise en main de l’API Twitter v2](#_v1qek9xgd8tq) 10

[Inscription](#_to2cqukkrq6a) 10

[Création d’un projet et connection à l’application](#_vdejbfcbhmzf) 10

[Requêtage](#_50z26dc8rk35) 12

[URL](#_6uw5i19pcieb) 12

[Identifiants](#_5yimnclf1s6s) 12

[Paramètres de recherche](#_iwtk0dzakqw1) 13

[**Web scraping**](#_37vrd46prqh8) **14**

[Présentation](#_4u2s9r1w8yg0) 14

[Octoparse](#_l0unw4pn719s) 14

[Prise en main de Octoparse](#_3loehcp2i6xx) 15

[Inscription](#_e4y1hodpfzod) 15

[Installation](#_ihc5j8b8r3zy) 15

[Création du Crawler](#_qjxdnjg6sesu) 15

[Limite d’Octoparse](#_t5foswmbv3dm) 19

[**BigQuery**](#_du057k5aywh7) **20**

[Présentation](#_qdl3rs7egnqr) 20

[Prise en main](#_euj7tc7nry61) 20

[Activation des API](#_dl9qdd5w60oz) 20

[Création d’un compte de service](#_2go7jkb6eoyr) 20

[Création d’une database et de la table associée](#_pqo1dcrfehhh) 20

[Connexion au client BigQuery](#_fqsrtppaz2s2) 21

[Principales requêtes](#_zgxhw5dn5236) 21

[**Analyse de sentiment**](#_ipzpvil5as6m) **23**

[Analyse de la structure NLP](#_o2wjrtuw1hoy) 23

[Google Natural Language](#_j9bc1p4qon8n) 23

[**Reconnaissance d’entité nommée**](#_vbdrelvmn7hb) **25**

[Spacy](#_1vgoj4dztmbt) 25

[Google Natural Language](#_egkmgigkm206) 25

[Libraire FuzzyWuzzy](#_1owbreldusyl) 25

[**Nettoyage des tweets**](#_iip2p4kuyhjj) **28**

[Explication et intérêt de la démarche](#_fm9zlbp0ra44) 28

[Fonctionnement de la fonction de Preprocessing](#_dbp9fjhyn6rj) 28

[WordCloud](#_mqtdr7v8efz8) 29

[**Automatisation**](#_4xh43jx5b07b) **30**

[Prise en main](#_pfk8a2jbirhw) 30

[Création d’un sujet](#_pnpwvh91lewm) 31

[Création d’une fonction](#_xs4dc111rqqn) 31

[**Logiciel**](#_7sepnot8690k) **34**

[Architecture](#_xotxtka1n2tp) 34

[Mise à jour de la table de tweets](#_otbq2edjxmm4) 35

[Requêtage de l’API Twitter v2](#_meputys5uzrc) 36

[Génération des clés](#_w4wm6yw6mcqt) 37

[Analyse de sentiments](#_1xcicttusar1) 37

[Analyse d’entités](#_1g8m0eaw8zz1) 37

[Nettoyage des Tweet et Wordcloud](#_d1hg5sjh3tex) 38

[Import vers BigQuery](#_acyo9xmvh37q) 38

[**Limites et améliorations envisageables**](#_f151gktpu827) **39**

[Base de films Salto](#_f1qgxkx81n7f) 39

[Recherche de films](#_zd5dh1um982n) 39

[Base de films complémentaire](#_rwpavbvmbtwc) 39

[Secret Management](#_nxrbdj40mavm) 39

# Objectifs du projet

## Objectif Principal

Dans le but d’améliorer son offre et sa connaissance client, SALTO souhaite analyser les sentiments des tweets à son attention. Et ainsi en extraire la polarité (positive, négative, neutre), les irritants et points forts de son offre. Par la même occasion, SALTO souhaite détecter les programmes générant le plus de tweets et ainsi pouvoir les suivre au fil du temps en fonction des actions marketing et d’acquisition mises en place.

## Objectifs Secondaires

Détecter les futures pépites (programmes) dans le but d’améliorer notre stratégie d’acquisition de contenu. A ce titre, l’objectif ici est au niveau global (pas seulement les tweet à l’attention de SALTO) de détecter les programmes générant de l'intérêt.

Détecter les communautés d’utilisateurs autour d’un contenu (ex : fan des Marseillais..) dans le but de pouvoir les isoler et les adresser (actions marketing et acquisition)

# 

# 

# 

# API Twitter v2

## Présentation

L’utilisation de l’API Twitter v2 nécessite un compte twitter ainsi qu’un compte développeur. Ce dernier est nécessaire afin de créer les identifiants permettant de requêter l’API.

3 niveaux d’accès sont proposés :

* Essential
* Elevated
* Academic research

Ci-dessous, un récapitulatif des limites pour chaque niveau d’accès :

|  | Essential | Elevated | Academic Research |
| --- | --- | --- | --- |
| Tweets Caps | Retrieve up to 500k Tweets per month | Retrieve up to 2 million Tweets per month | Retrieve up to 10 million Tweets per month |
| Filtered stream rule limit | 5 rules | 25 rules | 1000 rules |
| Filtered stream POST rules rate limit | 25 requests per 15 minutes | 50 requests per 15 minutes | 100 requests per 15 minutes |

*Niveaux d’accès et limites du compte développeur - Twitter API*

La recherche de Tweets est une fonctionnalité importante utilisée pour afficher les conversations Twitter sur un sujet ou un événement spécifique. Bien que cette fonctionnalité soit présente dans Twitter, ces “end points’ offrent une plus grande flexibilité et puissance lors du filtrage et de l'ingestion de Tweets afin que vous puissiez trouver plus facilement des données pertinentes pour votre recherche ; créer des applications d'écoute » en temps quasi réel ; ou généralement explorer, analyser et/ou agir sur les Tweets liés à un sujet d'intérêt.

Twitter propose 2 end points permettant la recherche de tweets :

* la recherche récente
* la recherche d'archives complètes

Ces deux points de terminaison REST partagent une conception et des fonctionnalités communes, notamment leur utilisation d'une requête de recherche unique pour filtrer les Tweets autour d'un sujet spécifique. Ces requêtes de recherche sont créées avec un ensemble d'opérateurs qui correspondent aux attributs du Tweet et de l'utilisateur, tels que les mots clés du message, les hashtags et les URL. Les opérateurs peuvent être combinés dans des requêtes avec une logique booléenne et des parenthèses pour aider à affiner le comportement de correspondance des requêtes.

Une fois que vous avez configuré votre requête et commencé à recevoir des Tweets, ces points de terminaison prennent en charge la navigation dans les résultats à la fois par plages d'heure et d'ID de Tweet. Ceci est conçu pour prendre en charge deux cas d'utilisation courants :

1. Obtenir l'historique : les demandes concernent une période d'intérêt, sans se concentrer sur la nature en temps réel des données. Une seule demande est faite et toutes les données correspondantes sont livrées en utilisant la pagination selon les besoins. Il s'agit du mode par défaut pour la recherche de tweets.
2. Sondage ou écoute : " y a-t-il de nouveaux Tweets depuis ma dernière demande ?". Les requêtes sont réalisées de manière continue.

### Recherche récente

Elle permet d'accéder par programmation aux Tweets publics filtrés publiés au cours de la semaine dernière, et est disponible pour tous les développeurs qui ont un compte de développeur et utilisent des clés et des jetons d'une application dans un projet.

Ce end point peut diffuser **jusqu'à 100 Tweets par requête** dans **l'ordre chronologique inverse.**

Lorsque vous utilisez un projet avec un accès essentiel ou élevé, vous pouvez utiliser l'ensemble d'opérateurs de base et effectuer des requêtes contenant jusqu'à 512 caractères. Lorsque vous utilisez un projet avec accès à la recherche universitaire, vous avez accès à des opérateurs supplémentaires et pouvez effectuer des requêtes jusqu'à 1024 caractères.

### Recherche dans les archives complètes

Accès à la recherche universitaire uniquement, non traitée dans le cadre de ce projet.

### 

### Principales fonctionnalités concernant la recherche de tweets

La table ci-dessous présente les principaux endpoints disponibles via l’API Twitter v2. Pour la liste complète, veuillez-vous référer à la documentation twitter au lien suivant <https://developer.twitter.com/en/portal/products/essential>

| Feature | Details | Endpoints | Rate limit (per app) |
| --- | --- | --- | --- |
| Tweets lookup | Retrieve detailed, up-to-date information about a specific Tweet, or Tweets. | GET /2/tweets/:id | 900 requests / 15 min |
| Manage Tweets | Create and delete Tweets on behalf of a user. | POST /2/tweets  DELETE /2/tweets/:id | 200 request / 15min  50 requests / 15min |
| Timelines | Retrieve the Tweets created by a user, or the Tweets mentioning a user. | GET /2/users/:id/tweets  GET /2/users/:id/mentions | 1500 requests / 15 mins  450 requests / 15 mins |
| Search Tweets | Search for historical Tweets, using advanced filtering tools to refine your search criteria. | GET /2/tweets/search/recent | 450 requests / 15 mins |
| Tweet counts | Understand and retrieve the volume of Tweet data for a search query, using the same advanced filtering tools available on the Search Tweets endpoints. | GET /2/tweets/counts/recent | 300 requests / 15 mins |
| Filtered stream | Filter the real-time stream of publicly available Tweets, using advanced filtering tools to refine your criteria. | POST/2/tweets/search/stream/rules  GET /2/tweets/search/stream/rules  GET /2/tweets/search/stream | 25 requests / 15 mins  450 requests / 15 mins  50 requests / 15 mins |
| Volume stream | Retrieve a 1% random sample of publicly available Tweets in real time. | GET /2/tweets/sample/stream | 50 requests / 15 mins |
| Retweets | Retweet and undo Retweets, or retrieve users that have Retweeted a Tweet | GET /2/tweets/:id/retweeted\_by | 75 requests / 15 mins |
| Likes | Like and un-like Tweets, or retrieve a user's Likes, or a list of users that have liked a Tweet. | GET /2/tweets/:id/liking\_users  GET /2/users/:id/liked\_tweets  POST /2/users/:id/likes  DELETE /2/users/:id/likes/:tweet\_id | 75 requests / 15 mins  75 requests / 15 mins  1000 requests / 24 hours  50 requests / 15 mins |
| Hide replies | Hide or unhide replies to improve the quality of the conversation | PUT /2/tweets/:tweet\_id/hidden | 50 requests / 15 mins |

*Twitter API v2 - liste des endpoints*

### Timeline mention Endpoint

Afin de répondre à la demande de Salto, nous avons utilisé le endpoint *GET/2/users/:id/mentions*.

Ce endpoint permet d'accéder aux Tweets mentionnant un utilisateur spécifique . Cela inclut également les réponses aux Tweets de l'utilisateur demandé. Il n’est toutefois possible que de récupérer les 800 derniers tweets mentionnant l’utilisateur. Chaque requête renvoie (selon les paramètres) l’identifiant du tweet, sa date de publication ainsi que son texte. Les tweets sont **diffusés dans l'ordre chronologique inverse**, en commençant par le plus récent. Les résultats sont paginés jusqu'à 100 Tweets par page sous format JSON.

La table ci-dessous précise les principaux paramètres de la requête.

Pour une description exhaustive, [ici](https://developer.twitter.com/en/docs/twitter-api/tweets/timelines/api-reference/get-users-id-mentions)

| Nom | Type | Description |
| --- | --- | --- |
| id (required) | String | Identifiant unique de l'utilisateur pour lequel on souhaite retourner les Tweets |

*Timeline Mention - Paramètre de chemin*

| Nom | Type | Description |
| --- | --- | --- |
| end\_time | date (ISO 8601) | AAAA-MM-JJTHH:mm:ssZ (ISO 8601/RFC 3339). Date limite jusqu’à laquelle les tweets seront recherchés. Veuillez noter que ce paramètre ne prend pas en charge une valeur en millisecondes. |
| max\_results | integer | Spécifie le nombre de Tweets à essayer de récupérer, jusqu'à un maximum de 100 par requête. Par défaut, 10 résultats sont retournés si ce paramètre n'est pas renseigné. La valeur minimale autorisée est 5. |
| start\_time | integer | AAAA-MM-JJTHH:mm:ssZ (ISO 8601/RFC 3339).Date limite à partir de laquelle les tweets seront recherchés. Veuillez noter que ce paramètre ne prend pas en charge une valeur en millisecondes. |
| tweet.fields | enum (attachments, author\_id, context\_annotations, conversation\_id, created\_at, entities, geo, id, in\_reply\_to\_user\_id, lang, non\_public\_metrics, public\_metrics, organic\_metrics, promoted\_metrics, possibly\_sensitive, referenced\_tweets, reply\_settings, source, text, withheld) | Ce paramètre permet de sélectionner les informations relatives aux tweets à récupérer. Spécifiez les champs souhaités dans une liste séparée par des virgules sans espaces entre les virgules et les champs. |

*Timeline Mention - Paramètre de requête*

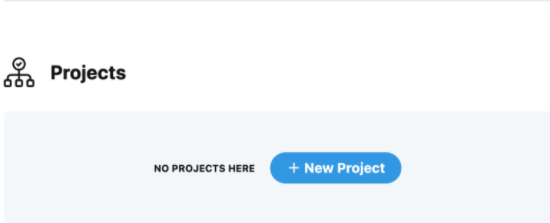
## Prise en main de l’API Twitter v2

### Inscription

L’utilisation de l’API Twitter v2 nécessite la création d’un compte twitter et d’un compte développeur. Cette opération peut se faire [ici](https://developer.twitter.com/en/portal/dashboard)

### Création d’un projet et connection à l’application

Ensuite, dans le portail des développeurs, créez un nouveau projet.

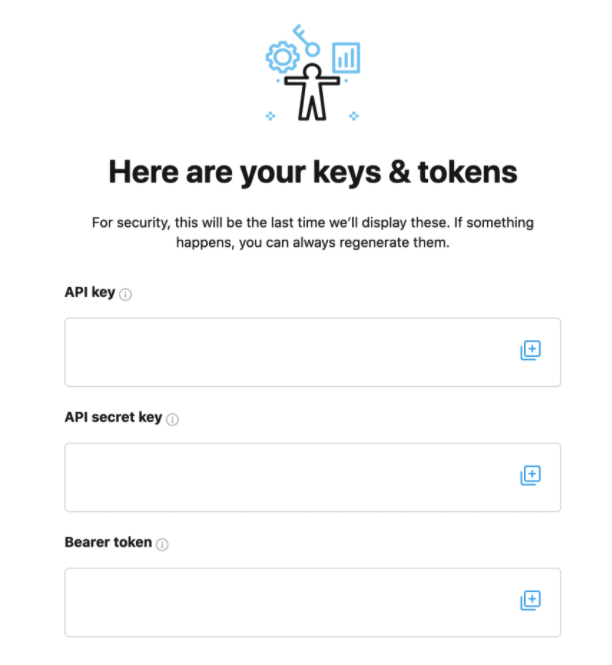


*Twitter API v2 - Création d’un projet*

Donnez-lui un nom, sélectionnez le cas d'utilisation approprié, fournissez une description du projet. Ensuite, vous pouvez soit créer une nouvelle application, soit connecter une application existante (une application est un conteneur pour vos clés API dont vous avez besoin pour faire une requête HTTP à l'API Twitter).

Cliquez sur "créer une nouvelle application à la place" et donnez un nom à votre application afin de créer une nouvelle application.

Une fois que vous avez cliqué sur terminé, vous obtiendrez vos clés d’authentification que vous pourrez ensuite utiliser pour vous connecter aux différents endpoints proposés par l'API Twitter v2.



*Twitter API v2 - Obtention des clés utilisateur*

### Requêtage

La libraire *requests* permet de requêter un serveur http:// à l’aide de la fonction get.

Elle requiert 3 arguments à savoir :

* un url
* des identifiants / clés (si besoin)
* Les paramètres de recherche



*Requêtage Twitter API v2*

#### URL

Le ‘Endpoint’ mention est accessible via l’url suivant :

*https://api.twitter.com/2/users/user\_id/mentions*

ou ‘user\_id’ fait référence à l’identifiant Twitter de l'utilisateur, ici Salto.

Twitter propose un outil de conversion via son site <https://tweeterid.com/> à partir du nom de compte utilisateur comme présenté ci-dessous.



*Salto Twitter ID*

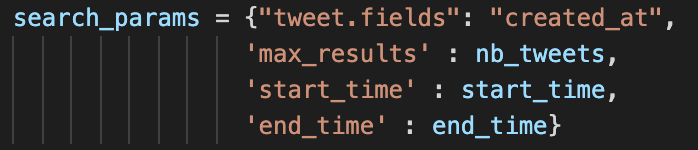
#### Identifiants

Ils sont propres au compte développeur et sont générés lors de la création du projet via le site <https://developer.twitter.com/en/portal/dashboard>

#### Paramètres de recherche

La liste des paramètres est précisée au sein du chapitre *Timeline mention Endpoint.*

Dans notre cadre applicatif, nous nous intéressons plus particulièrement au paramètres suivants :



*Timeline Mention - Paramètres*

* created\_at : fait référence à la date de publication du tweet
* max\_results : correspond au nombre maximum de tweet récupéré par requête (limite=100)
* start\_time : date limite à partir de laquelle récupérer les tweets
* end\_time : date limite jusqu’à laquelle récupérer les tweets

# Web scraping

## Présentation

Les limites de l’API twitter (800 tweets les plus récents) nous ont conduits à la solution du web scraping pour récupérer l’historique des tweets mentionnant Salto. En premier lieu nous avons tenté une approche directement via un compte Twitter mais les ban successifs nous ont amenés vers des solutions alternatives dont Octoparse.

## Octoparse

Octoparse nécessite l’installation du logiciel et la création d’un compte pour celui-ci.

3 niveaux d’accès sont proposés :

* Gratuit
* Standard (payant)
* Professionnelle (payant)

Ci-dessous, un récapitulatif des limites pour chaque niveau d’accès :

|  | Gratuit | Standard | Professionnelle |
| --- | --- | --- | --- |
| Crawler | 10 | 100 | 250 |
| Exportation | 10000 enregistrement de données | illimité | illimité |
| Cloud | Non | Oui | Oui |
| Template | Non | Oui | Oui |
| Rotation IP | Non | Oui | Oui |

*Niveaux d’accès pour Octoparse*

## Prise en main de Octoparse

### Inscription

L’utilisation de Octoparse nécessite la création d’un compte twitter et d’un compte développeur. Cette opération peut se faire [ici](https://www.octoparse.fr/signup)

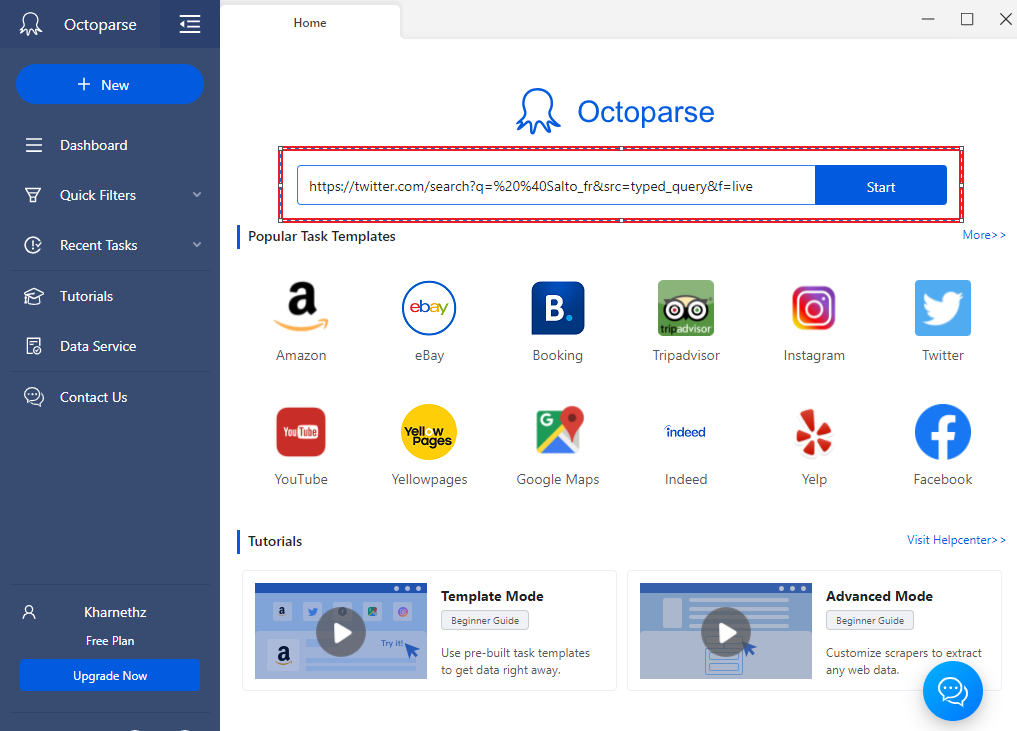
### Installation

Il est aussi nécessaire de télécharger leur logiciel. Cette opération peut se faire [ici](https://www.octoparse.fr/download/windows)

### Création du Crawler

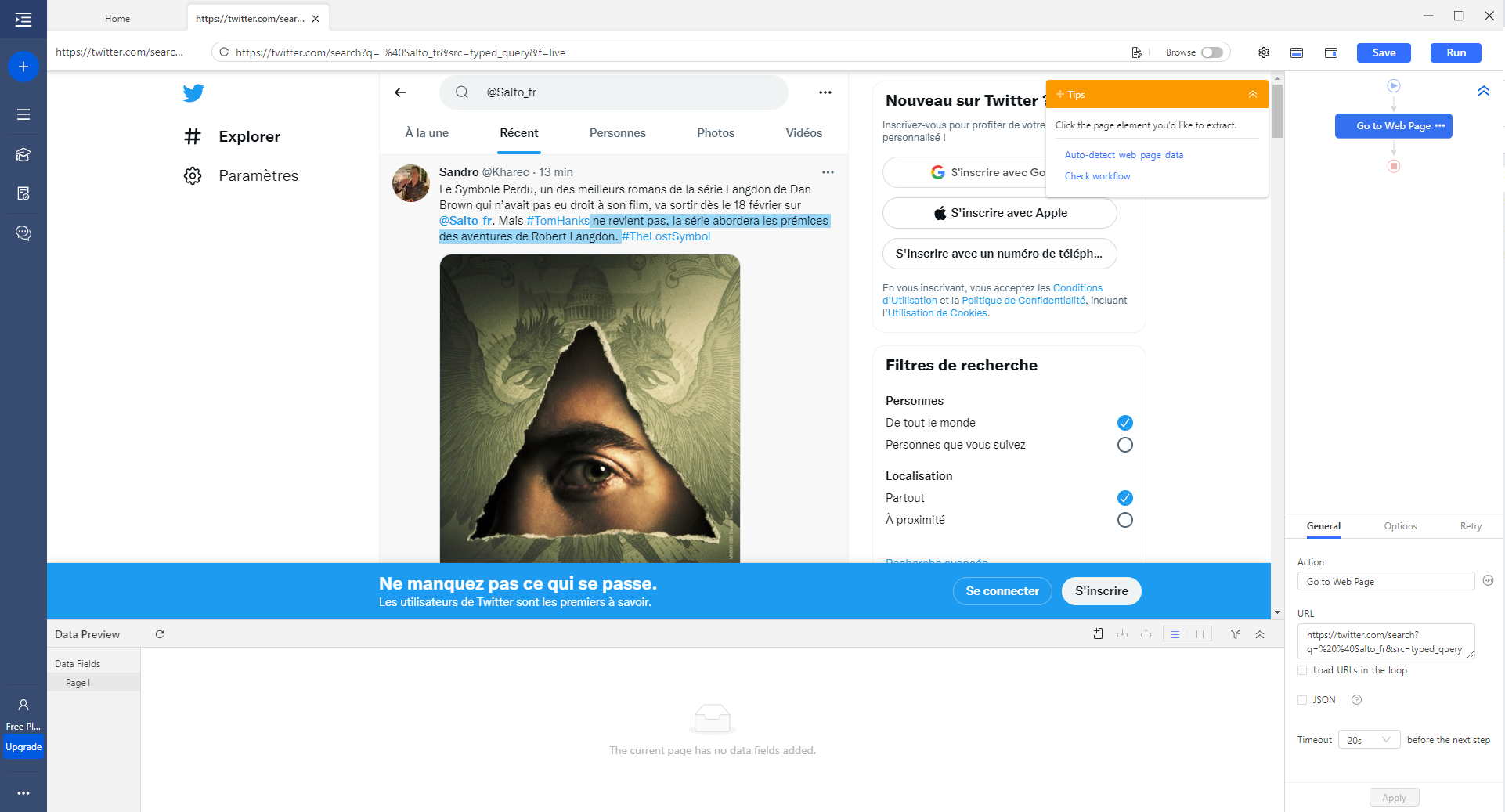
Rendez vous sur twitter et récupérez le lien de la page contenant les tweets que vous voulez récupérer.

Saisissez ce lien dans Octoparse:



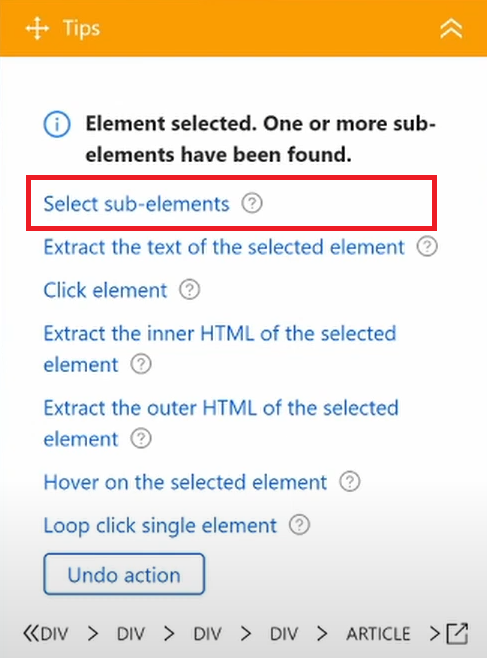
*Page d'accueil Octoparse*

Cliquez sur start, vous serez donc sur la page des tweet que vous souhaitez récupérer.



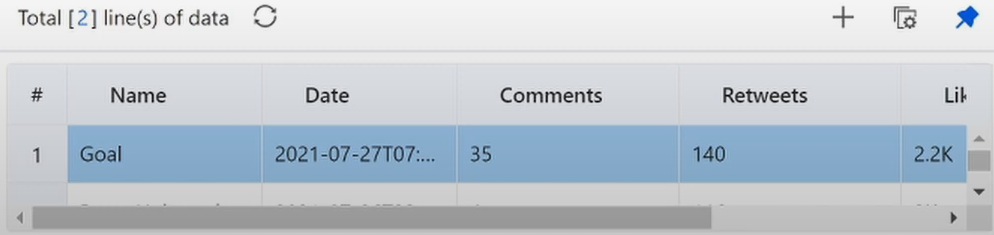
*Page d'accueil Twitter*

Il faut donc maintenant cliquer sur le premier tweet, après avoir sélectionné ce dernier, il faudra cliquer sur “Select sub-elements”



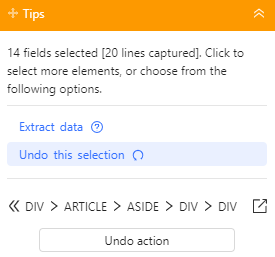
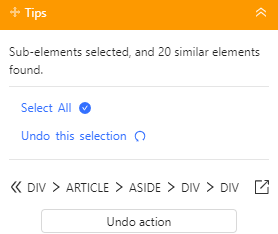
*Actions Octoparse*

Cela créera un tableau temporaire sur lequel vous pourrez observer les données récupérées et renommer les colonnes si besoin.



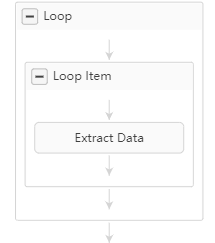
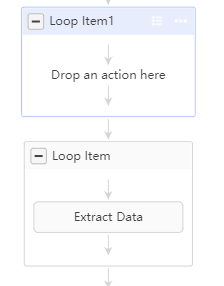
*Table de données*

Il vous faudra ensuite cliquer sur ‘Select All’, puis ‘Extract data’



*Extraction des données sous Octoparse*

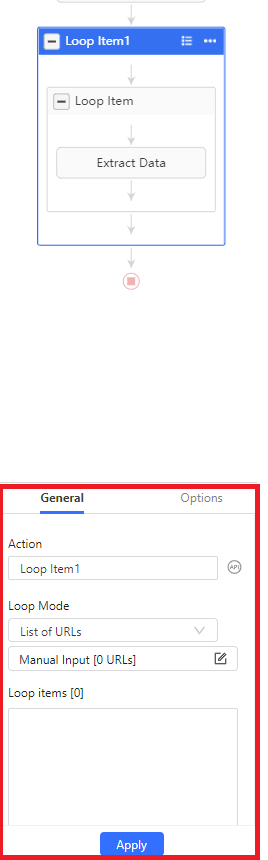
Maintenant que vous avez les informations pour une vingtaine de tweets, une ‘Loop item’ apparaît dans le workflow. Il faut maintenant rajouter une boucle par dessus celle-ci, pour récupérer tous les tweets. Passer votre souris sur le ‘+’ et sélectionner ‘Loop’. Ensuite glisser votre premier ‘Loop item’ dans votre nouvelle ‘Loop.

’

*Bouclage Octoparse*

Dernière étape, paramétrage de cette dernière Loop. Cliquez dessus et vous aurez l’onglet général ouvert en bas du workflow.

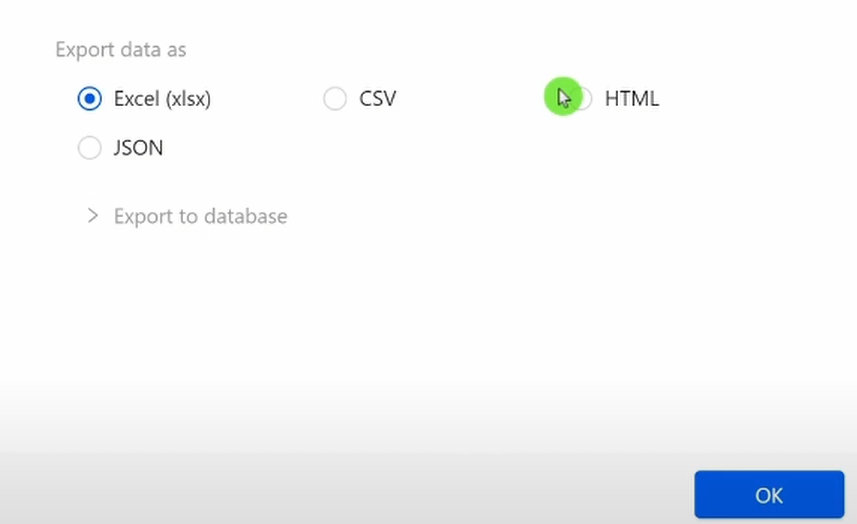
Dans Loop mode, sélectionnez ‘Scroll Down’ et ‘Apply’.



*Bouclage Octoparse - configuration*

Votre Crawler est maintenant prêt, il ne vous reste plus qu'à cliquer sur Save, puis Run.

Une nouvelle fenêtre s’ouvrira vous montrant l’avancement de la récupération de données, une fois fini cela vous demandera sous quel format l’importer



*Exportation des données*

## Limite d’Octoparse

Octoparse aura certaines limites avec twitter, notamment récupérer les tweet contenant des images. Le logiciel a une difficulté à récupérer les tweets contenant un média. Ce qui occasionne une perte de ⅓ des tweets.

# 

# BigQuery

## Présentation

Afin de suivre les recommandations de Salto, l’ensemble des données est stocké sur BigQuery afin d’assurer l’homogénéité dans les outils/plateformes cloud utilisés par la société.

## Prise en main

### Activation des API

Le fonctionnement de BigQuery dans le cadre du projet nécessite l’activation des API suivantes, directement via Google Cloud Platform:

* BigQuery API
* BigQuery Connection API

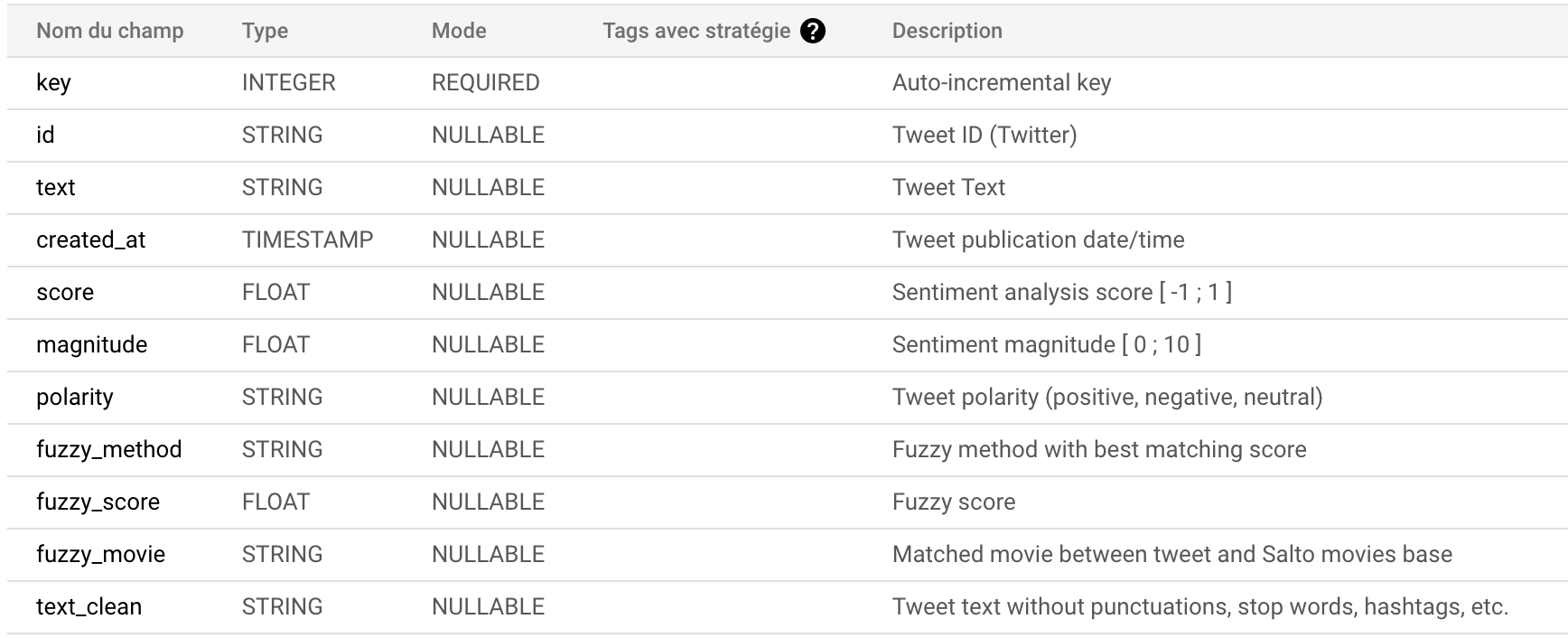
### Création d’un compte de service

Création du compte de service “[wcs-remote@salto-datalab-pid2.iam.gserviceaccount.com](mailto:wcs-remote@salto-datalab-pid2.iam.gserviceaccount.com)” et génération des clés d’authentification associées sous format JSON

### Création d’une database et de la table associée

La base de données “Twitter\_database” a été créée afin d’héberger l’ensemble des données du projet sur des serveurs européens. La sécurité des données est basée sur une clé de chiffrement gérée par Google.

La table “Tweets” contient les données selon la structure suivante :



*BigQuery - Schéma de la table*

* key : elle correspond à un identifiant unique généré automatiquement et auto-incrémenté lors de l’ajout de nouveaux tweets au sein de la table. Ainsi le tweet le plus ancien possède la clé numéro 0 et le tweet le plus récent est associé à la clé la plus grande.
* id : identifiant du tweet (géré par Twitter). Il permet, entre autre, de réaliser des requêtes spécifiques par rapport à un tweet.
* text : texte brut du tweet
* created\_at : date et heure de publication du tweet (heure UTC)
* score : résultat de l’analyse de sentiment via Google Natural Language. Ce score est compris entre - 1 et 1 , -1 correspondant à un résultat extrêment négatif et +1 à un résultat extrêment positif. La note de 0 correspond à un texte pour lequel aucun sentiment n’est détecté.
* magnitude : indique l'intensité générale de l'émotion (positive ou négative) exprimée dans le texte en question. Elle est comprise entre 0.0 et +inf
* fuzzy\_score : score de matching (0 -100) entre les tweets et la base de donnée Salto (liste de films sur la plateforme de streaming)
* fuzzy\_method : correspond à la méthode de matching ayant donné les meilleurs résultats (si résultats > 90/100). 4 méthodes ont été implémentées à date : *fuzz.ratio, fuzz.partial\_ratio, fuzz.token\_sort\_ratio, fuzz.set\_ratio*
* fuzzy\_movie : nom du film retrouvé dans la base Salto, si score >90
* text\_clean : texte du tweet après suppression des caractères/mots inutiles

### Connexion au client BigQuery

La connexion se fait à partir des identifiants du compte de service au format JSON via la syntaxe suivante :



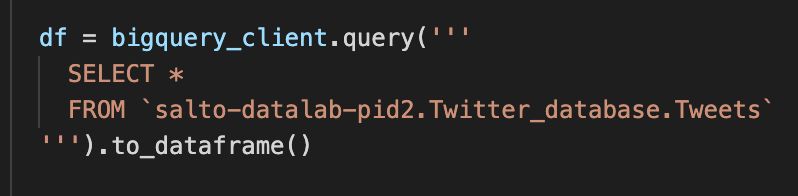
*BigQuery - connexion client*

### Principales requêtes

L’application développée dans le cadre de ce projet utilise principalement deux types de requêtes visant la gestion de la base de données BigQuery.

La première permet d'interroger la base de données actuelle pour en extraire la donnée.

Elle utilise la fonction *query* associée au client BigQuery. Ci-dessous est donné un exemple pour récupérer l’ensemble des tweets de la table “Tweets”, de la base de données “Twitter\_database” du projet salto-datalab-pid2:



*BigQuery = Requête de l’ensemble de la table*

La seconde requête permet quant à elle de mettre à jour la table. Elle utilise la fonction *load\_table\_from\_dataframe* avec pour argument :

* les nouvelles données à injecter dans la table (format DataFrame Pandas)
* les paramètres d’import avec :
  + Le schéma des données (doit être identique au schéma de la table)
  + Le mode de copie des données (append, overwrite…)

Dans le cadre du projet on choisira le mode “Append” permettant d’ajouter de nouvelle lignes sans modification des données déjà existantes

* le chemin (path) de la table avec pour format “nom\_projet.nom\_database.nom\_table”

Exemple ci-dessous:

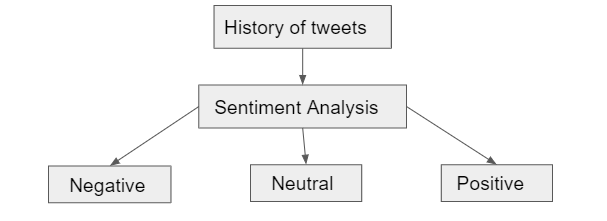


*BigQuery - Mise à jour de la table*

# Analyse de sentiment

## Analyse de la structure NLP

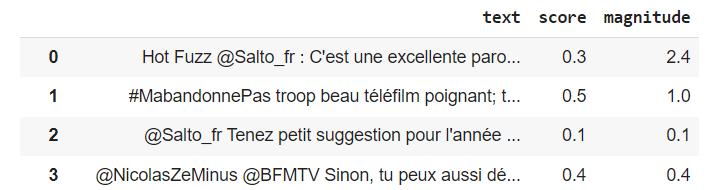
Dans notre étude, nous avons développé et désigné une structure NLP en classifiant notre historique de tweets en 3 groupes: positive, négative, et neutre à travers l’analyse des sentiments. La figure ci-dessous montre un résumé de notre structure. Cette structure peut identifier les différents types de tweets, les sujets mentionnés associés. Ainsi, nous pouvons identifier les différents opinions, afin de prévenir, ou d’orienter le public sur l’amélioration de la plateforme.



*Analyse NLP pour les tweets*

## Google Natural Language

L’analyse des sentiments inspecte le tweet donné et identifie l’opinion émotionnelle dominante ressortant du tweet afin de déterminer le comportement utilisateur comme positif, négatif ou neutre. L’analyse des sentiments utilisée dans notre étude est Google Cloud sentiment analysis. Google Cloud sentiment analysis appartient à l’API Cloud Natural Language de Google. Les utilisateurs ont besoin d’obtenir les crédits Google Cloud, afin d’utiliser ce service Google. En fournissant un texte, Google Cloud sentiments analysis produit les scores et les valeurs des magnitudes du sentiment. Ces scores s’établissent sur une valeur entre -1 et 1, pour laquelle -1 traduit un sentiment extrêmement négatif, 0 pour neutre, et 1 pour extrêmement positive. Chaque score possède une magnitude pour un tweet, et une valeur élevée de la magnitude indique l’ampleur du sentiment. En donnant un tweet qui inclut de multiples phrases, l’algorithme calcule le sentiment du score et la magnitude associée à chaque phrase, et fournit donc un score global du sentiment et la magnitude globale pour notre tweet en entrée.



*Exemple d’analyses de sentiments sur 5 tweets, et d’un score de magnitude*

Nous pouvons voir sur le premier tweet, une bonne magnitude par rapport aux autres

caractérisée par le terme ‘excellente’, pour les autres tweets les mots semblent moins

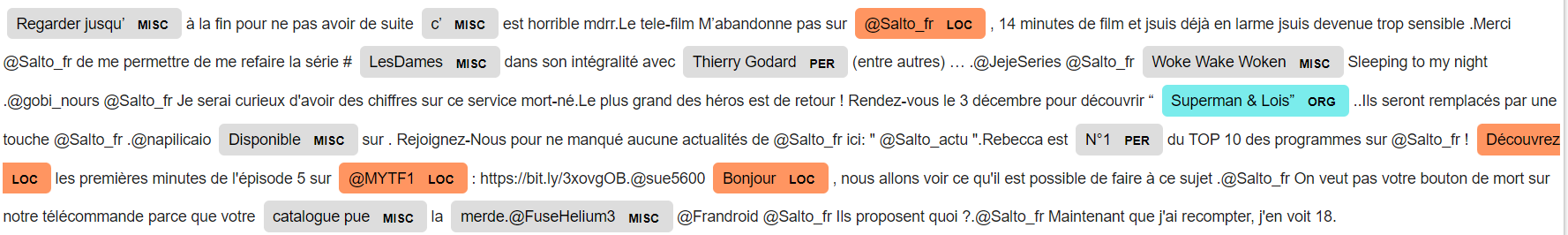
marquants émotionnellement, traduisant donc une magnitude plus faible.

# 

# Reconnaissance d’entité nommée

## Spacy

L'approche de l'extraction et de l'analyse des entités ne se limite pas à la recherche de noms, mais consiste également à inspecter le texte donné pour y trouver des entités connues telles que des personnalités publiques, des points de repère et des noms communs tels que restaurant, stade, etc. Il renvoie également des informations sur ces entités. Dans notre étude, nous avons utilisé l'analyse d'entités de spaCy. L'analyse d'entités de spaCy a identifié les entités dans notre texte de tweets et les a classées dans différentes catégories, dont les suivantes : personne, lieu, organisation, divers. La ci-dessous montre un exemple d'analyse d'entité en utilisant l'analyse d'entité de spaCy. L'absence d’entité relative à l’art, les films et le manque de performance au global nous a encouragé à abandonner cette approche.



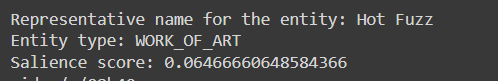
**Figure 5. Exemple d’analyse et d’extraction d’entités**

**Figure 5. Exemple d’analyses d**

**Figure 5. Exemple d’analyses d’entités avec SpaCy**

## Google Natural Language

Nous avons testé l’API Google Natural Language en comparaison avec SpaCy. Celui-ci délivre une catégorie d’entités, plus spécifiques pour la dénomination des programmes avec l’entité ‘WORK\_OF\_ART’, à travers notre exemple ci-dessous.

****

*Dénomination d’entité sur Google Natural Language*

Cependant, la limite de cette méthode réside sur l’ordre d’apparition de la catégorie d’entité concerné, impactant le score de pertinence ‘Salience score’. Cette approche nuirait donc à la détection du programme le plus pertinent pour un tweet donné.

## Libraire FuzzyWuzzy

Au sein du package Fuzzy Wuzzy, nous avons comparé les 4 méthodes de matching, qui sont les méthodes ‘fuzz.ratio’, ‘ratio’, ‘partial\_ratio’, et ‘token\_sort\_ratio’. La méthode ‘ratio’ calcule uniquement le ratio de similarité entre 2 séquences de même longueur, en se basant sur le rapport de similitude de la distance de Levenshtein, tandis que celle de la fuzz.partial\_ratio() est capable de détecter que les deux chaînes de caractères faisant référence à un mot particulier. La façon dont cela fonctionne est l'utilisation d'une logique de "partiel optimal". En d'autres termes, si la chaîne courte a une longueur k et la chaîne plus longue une longueur m, l'algorithme recherche le score de la sous-chaîne de longueur k qui correspond le mieux.

En outre, les fonctions fuzz.token présentent un avantage important par rapport à ratio et 'partial ratio'. Elles tokenisent les chaînes de caractères et les pré-traitent en les transformant en minuscules et en supprimant la ponctuation. Dans le cas de fuzz.token\_sort\_ratio(), les chaînes de caractères sont triées par ordre alphabétique et ensuite assemblées. Après cela, une simple fuzz.ratio() est appliquée pour obtenir le pourcentage de similarité.

En particulier, fuzz.token\_set\_ratio() adopte une approche plus flexible que fuzz.token\_sort\_ratio(). Au lieu de simplement tokenizer les chaînes, de les trier puis de les recoller, token\_set\_ratio() effectue une opération d'ensemble qui retire les tokens communs (l'intersection) et effectue ensuite des comparaisons par paires fuzz.ratio() entre les nouvelles chaînes suivantes :

s1 = Tokens triés en intersection

s2 = Chaînes de tokens en intersection + restes de la chaîne 1 tokenisés

s3 = Intersection des éléments triés + reste des éléments triés de la chaîne 2.

La logique qui sous-tend ces comparaisons est que, comme la première chaîne s1 est toujours la même, le score aura tendance à augmenter au fur et à mesure que ces mots représentent une plus grande partie des chaînes originales ou que les tokens restants sont plus proches les uns des autres.

Dans notre exemple, on ressort des listes de scores, en prenant le score le plus haut parmi les différentes méthodes, avec une tolérance fixé à 90 %, afin de ressortir des programmes adaptés, par rapport à notre liste initiale. Nous gardons uniquement le premier programme qui est le plus cohérent sur le contenu du tweet.

De plus, nous avons retenu une liste de bad words personnalisés, à travers cet exemple:

list\_bad\_words = ['Vu','Face-à-face']

*Liste des termes nuisibles*

Parmi l’ensemble des méthodes, la fonction fuzz.token\_set\_ratio et ratio présentent les scores les plus élevés et sont majoritairement retenus. Toutefois, nous avons souhaité conserver l’ensemble des fonctions pour s’assurer une meilleure exhaustivité quant à l’identification des films au sein des tweets.

# 

# Nettoyage des tweets

## Explication et intérêt de la démarche

Nous avons effectué une étape de preprocessing des tweets bruts, qui est une étape importante dans notre analyse des sentiments. En effet, cette partie est importante car nos tweets bruts, sans ‘cleaning’, contenaient des informations non structurées, et redondantes, souvent non pertinentes.

Par exemple, les hashtags, les liens, et les mentions de référence @ ne sont pas nécessaires pour notre thème de recherche, en fournissant aucun contexte significatif pour l’exploration de nouveaux thèmes provenant des tweets.

## Fonctionnement de la fonction de Preprocessing

Nous avons pour cela développé pour notre module 'manage tweets', une fonction permettant le ‘cleaning’ des tweets, via la bibliothèque NLTK qui est un outil de traitement naturel de Python. Pour chaque ajout de tweets, celui-ci effectue le traitement brut tels que la suppression, remplacement textuelle des émojis, la liste des stopwords, la ponctuation, ou encore les liens urls.

Nous avons également défini une liste de symboles des ponctuations et caractères que nous voulions retirer pour nos tweets.

En utilisant ces modules, nous avons facilement créé une fonction applicable pour notre tâche spécifique de preprocessing. De cette façon, nous pouvons le réutiliser pour n’importe quel dataset.

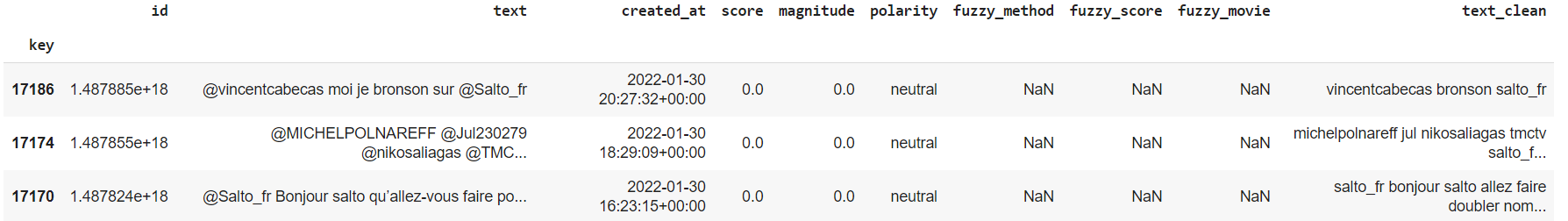
En retirant les liens, les hashtags, ou encore les tags, ponctuations, nous pouvons utiliser les expressions régulières (Regex).

Afin d’appliquer nécessairement les techniques de NLP(Natural Language Processing), tels que la tokenisation, par exemple nous pouvons utiliser ces fonctions suivantes:

La tokenisation est un processus de séparation d’un document (un tweet) en plusieurs mots, ponctuations, valeurs numériques, etc. Dans le cadre de la tokenisation, la meilleure pratique est de retirer les stopwords qui sont essentiellement un ensemble de mots communs ne fournissant aucun contexte significatif en lui-même. Ceci, nous permet de nous concentrer sur les mots importants par exemple. Un exemple serait des mots comme ‘comment, parmi, le’ et d’autres, nous avons donc utilisé NLTK.

En outre, notre fonction 'clean tweets' applique toutes les fonctions définies ci-dessus en appliquant les termes en minuscules, séparant la ponctuation, les espaces et les nombres d’un tweet. Ceci retourne une liste de tokens cleanés.

Par la suite, nous pouvons appliquer toutes ces étapes de preprocessing, via notre fonction sur notre dataframe et voir les résultats.



*Résultats du nettoyage des tweets*

A partir de cette étape, nous pourrons effectuer une analyse des tweets et voir les différentes tendances: tweets positifs, négatifs (WordCloud) sur notre collection de tweets.

## WordCloud

Un wordcloud est basé sur une fréquence de mots d’un document, signifiant que plus le terme en question sera présent, plus celui-ci sera imposant. Ceci peut être très utile afin de se fixer un aperçu sur des avis.

Suite à la création de notre fonction 'clean tweets', une extraction sur l’ensemble des tweets nettoyés fut effectuée, en les assemblant sur une seule chaîne de caractères, séparés d’espaces pour notre wordcloud.

Sur l’exemple ci-dessous, nous voyons apparaître des mots tels que ‘saison’, ‘série’, ‘merci’, ou 'épisode' pour l’ensemble de nos tweets Salto, ce sont les mots qui apparaissent le plus souvent. Certains mots, s’ils devaient apparaître seuls n’auraient aucun sens. En effet, celles-ci possèdent une valeur ajoutée en les plaçant dans leurs contextes.



*Exemple de wordcloud pour les tweets Salto*

Bien entendu, nous pouvons également faire ressortir les sentiments de nos tweets, à travers notre champ ’polarity’ (positif, négatif, neutre), que nous avons intégré via Google Data Studio.

# Automatisation

L'automatisation a été réalisée sous GCP, à travers l’utilisation d’API dédiées.

Google Cloud Functions permet un déploiement du code sur le cloud supprimant le fardeau de la gestion de l'infrastructure sous-jacente lors de la mise à jour et du déploiement du code. D’autre part, Google Schedule permet d’automatiser une tâche à une fréquence donnée.

## Prise en main

Les API suivantes ont été activées :

* Cloud Function
* Cloud Scheduler

Pub/Sub vous permet de créer des systèmes de producteurs et de consommateurs d'événements, appelés éditeurs et abonnés. Les éditeurs communiquent de manière asynchrone avec les abonnés en diffusant des événements plutôt que par des appels de procédure à distance (RPC) synchrones.

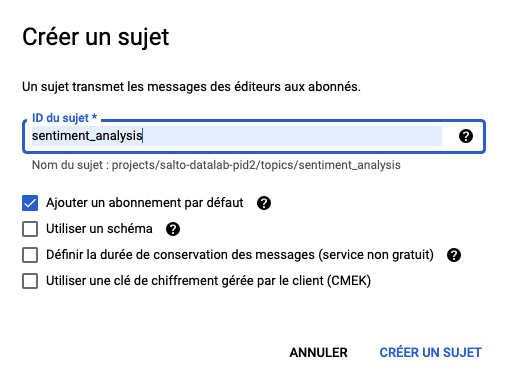
Les éditeurs envoient des événements au service Pub/Sub, sans tenir compte du mode ou du moment de leur traitement. Pub/Sub fournit ensuite des événements à tous les services qui doivent y réagir. Par rapport aux systèmes communiquant via des RPC, où les éditeurs doivent attendre que les abonnés reçoivent les données, une intégration asynchrone augmente la flexibilité et la robustesse du système global.

Les applications d'éditeur peuvent envoyer des messages dans un "sujet" et d'autres applications peuvent s'abonner à ce sujet pour recevoir les messages. Elle permet de créer un lien entre les subscribers (Sub) et les Publishers (Pub).

Dans ce cadre, la création d’une fonction nécessite la réalisation des tâches suivantes:

* La création d’un sujet
* La création de la fonction
* La création d’une tâche (scheduler)

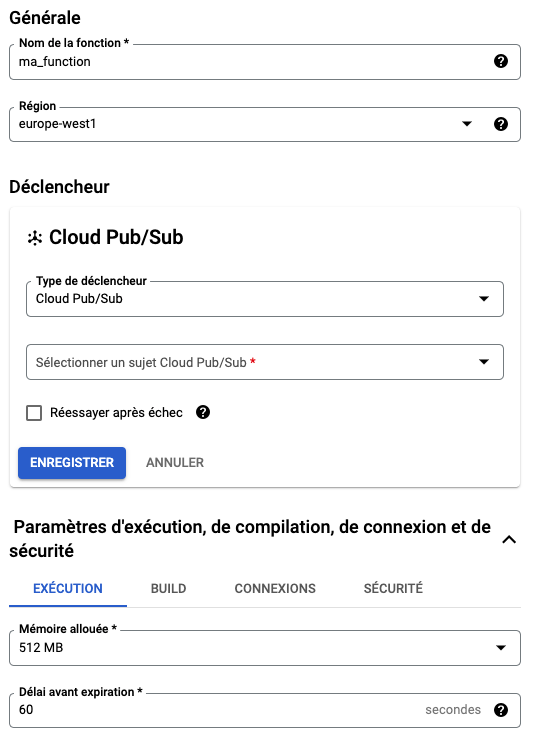
#### Création d’un sujet



*Création d’un sujet sous GCP*

#### Création d’une fonction

Dans un premier temps, vous devez renseigner le nom de votre fonction ainsi que la région, ici “europe-west1 pour la zone UTC+1. Sélectionner le type de déclencheur souhaité (dans notre cas Pub/Sub) et les paramètres de la fonction comme présenté ci-dessous:



*Création d’une fonction sous GCP*

La page suivante vous permet d’importer votre fonction à partir de votre ordinateur, de sélectionner la version de python ainsi que de définir le point d’entrée.

Plusieurs contraintes sont imposées par GCP pour valider l’import de la fonction:

* Le fichier principal (main.py) doit impérativement être à la racine du dossier zip
* Le fichier main doit être développé sous la forme d’une fonction prenant 2 arguments (ici data et context), même si ces derniers ne sont pas utilisés dans le code. Cette fonction est référencée dans le point d’entrée comme présenté dans l’exemple ci-dessous
* L’ensemble des fichiers importés ne doit pas contenir de bytes nuls. Leur suppression peut être réalisée via le terminal avec la commande sed suivi d’une expression regex tel que “sed -i '' 's/\x0//g' fichier.py” (sous mac). Il conviendra, si besoin, de l’appliquer à l’ensemble des scripts python.

# 

*Import de la fonction à partir d’un fichier zip*

Une fois la fonction importée avec succès, elle peut être exécutée.

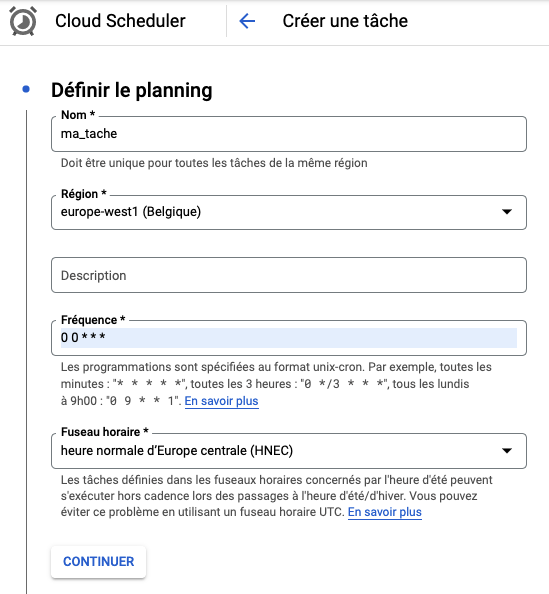
Création d’une tâche

Renseignez le nom de la tâche, la région ainsi que la fréquence sous format cron (mm hh jj MMM JJJ ) ou:

* mm représente les minutes (de 0 à 59)
* hh représente l'heure (de 0 à 23)
* jj représente le numéro du jour du mois (de 1 à 31)
* MMM représente l'abréviation du nom du mois (jan, feb, ...) ou bien le numéro du mois (de 1 à 12)
* JJJ représente l'abréviation du nom du jour ou bien le numéro du jour dans la semaine :
  + 0 = dimanche
  + 1 = lundi
  + 2 = mardi
  + …
  + 6 = samedi
  + 7 = dimanche (représenté deux fois pour les deux types de semaine)

Pour chaque valeur numérique (mm, hh, jj, MMM, JJJ) les notations possibles sont :

* \* : à chaque unité (0, 1, 2, 3, 4…)
* 5,8 : les unités 5 et 8
* 2-5 : les unités de 2 à 5 (2, 3, 4, 5)
* \*/3 : toutes les 3 unités (0, 3, 6, 9…)
* 10-20/3 : toutes les 3 unités, entre la dixième et la vingtième (10, 13, 16, 19)
* toutes les heures à la minute : 0 \*/2 \* \* \*

Dans le cadre de notre fonction, nous souhaitons qu’elle soit exécutée tous les jours à 00h00.

*Création d’une tâche sous GCP*

L’étape suivante consiste à définir le planning et le type d'exécution comme ci-dessous:

# 

*Définition du planning*

# Logiciel

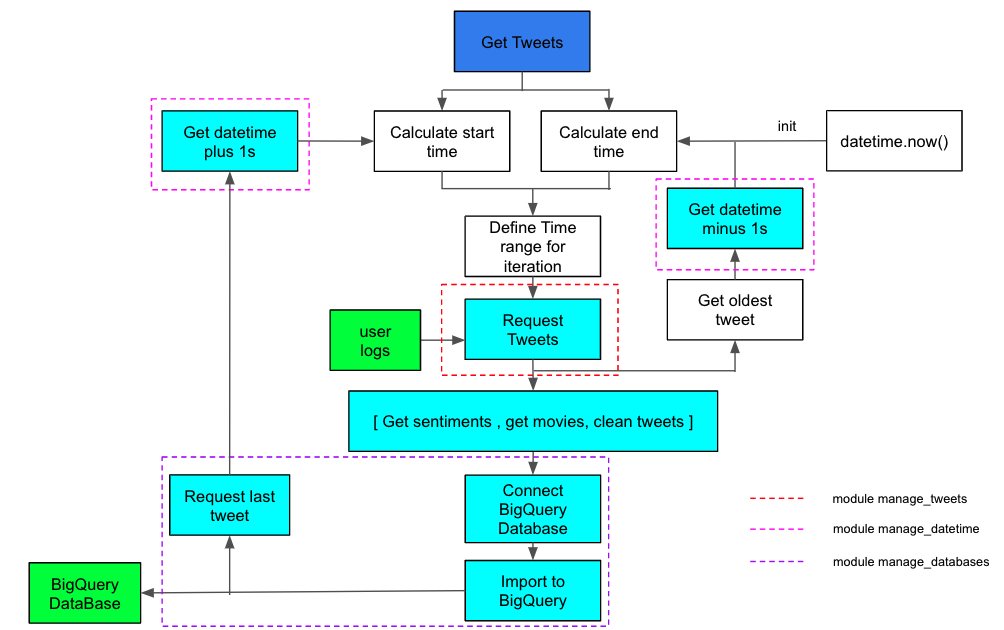
Les détails donnés ci-dessous concernent la méthodologie mise en œuvre dans le cadre du processus de captation des tweets via **l’API Twitter v2**. Les données récupérées par Octoparse ont fait l’objet d’un traitement indépendant nous permettant de constituer la première table sous BigQuery. Toutefois, les différentes étapes de processing étant les mêmes, il n’y a pas de partie dédiée dans ce document.

## Architecture

Le logiciel a été découpé en différents modules :

1. Manage\_tweets : gestion des tweets à travers le requêtage (API Twitter v2) ou le traitement des tweets via regex ou nltk
2. Manage\_bigquery\_database : permet le requêtage et le stockage des tweets sur BigQuery
3. Manage\_datetime : module de conversion des données temporelles
4. Manage\_cloud\_natural\_language : permet de requêter l’API GNL
5. Manage\_movie\_entity : module dédié à la reconnaissance d’entités nommées

Ci-dessous l’architecture globale :



*Architecture logiciel*

## Mise à jour de la table de tweets

Les différentes étapes présentées ci-dessous sont réalisées par l’algorithme de façon séquentielle. Chaque nouvelle étape de calcul (représentée par une nouvelle section) intervient donc après la réalisation de la précédente:

1. Requêtage de l’API Twitter v2
2. Génération des clés
3. Analyse de sentiment
4. Analyse d’entités
5. Nettoyage des tweets et Wordcloud
6. Import vers BigQuery

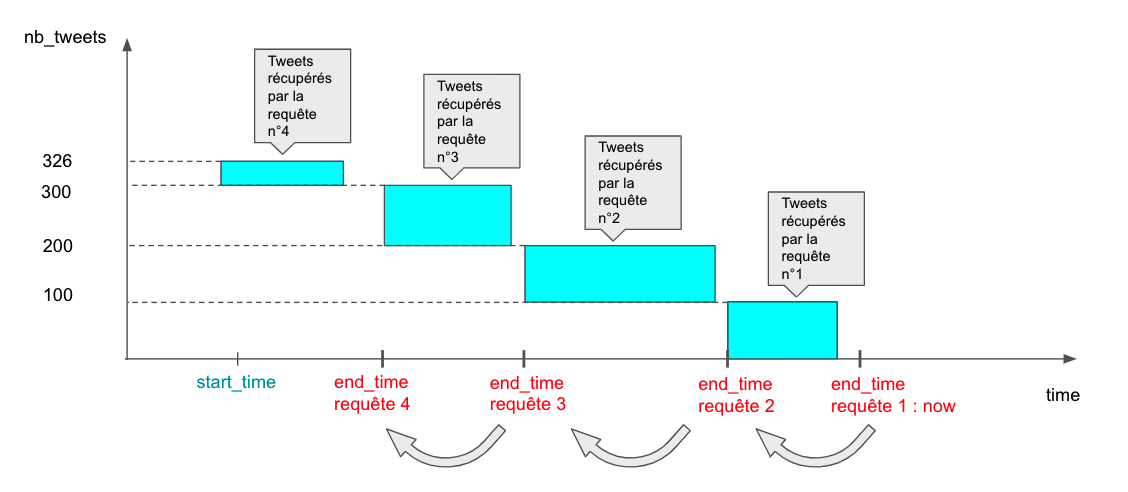
### Requêtage de l’API Twitter v2

La stratégie de requête a été définie en fonction des limites imposées par celle-ci. Pour rappel, il n’est possible de récupérer que les 800 tweets les plus récents (ce qui impose un requêtage régulier et automatisé) et chaque requête permet de récupérer au maximum 100 tweets (ce qui suppose plusieurs requêtes successives pour assurer de capter l’ensemble des tweets sur une plage de temps donnée), les derniers publiés.

Le schéma ci-dessous présente la stratégie de requêtage. Dans cet exemple, nous devons récupérer 326 nouveaux tweets, non présents dans notre database.

La plage de recherche [ start\_time , end\_time ] correspond à la période écoulée entre le dernier tweet capté et l’heure de la requête. Le start\_time est calculé à l’aide d’une requête BigQuery correspondant au plus récent tweet de la database.

1. La première requête permet de récupérer 100 tweets les plus récents sur notre plage de temps.
2. Le end\_time est fixé à la date du dernier tweet récupéré (le plus ancien parmis les 100 tweets)
3. La seconde requête est effectuée et permets de récupérer 100 nouveaux tweets
4. Le end\_time est réajusté
5. La 3ème requête est réalisée
6. Le end\_time est réajusté
7. La 4ème requête permet de récupérer seulement 26 tweets pour un maximum de 100 tweets => il n’y a pas d’autre nouveau tweet sur la plage de temps considérée.



*Twitter API v2 - boucle de requêtes*

L’ensemble des tweets et des informations inhérentes est stocké sous forme de DataFrame Pandas.

### Génération des clés

Afin de repérer les tweets à l’aide d’un identifiant unique, une clé unique est créée de façon incrémentale selon la date de publication des tweets. La clé du tweet le plus récent dans la database sert de référence à la construction de la nouvelle liste de clés pour les nouveaux tweets récupérés.

### Analyse de sentiments

Pour chaque tweet, le texte est soumis à une analyse de sentiment via l’API Google Natural Language permettant de ressortir le score et la magnitude de l’analyse. Le texte est envoyé vers l’API de façon brute, sans traitement dans la mesure ou Google utilise ses propres outils et technologie de preprocessing. Ces données viennent s'ajouter au DataFrame existant.

### Analyse d’entités

Cette étape de processing vise à chercher un “match” entre les tweets et une liste de films communiquée par Salto. Cette démarche vise in fine, à identifier les principaux centres d’intérêt des utilisateurs de la plateforme de streaming, définir les programmes qui font l’objet de commentaires, qu’ils soient positifs ou négatifs.

Cette analyse est réalisée à l’aide de la libraire Fuzzy selon différentes méthodes de matching. Seuls les résultats présentant un taux de matching de plus de 90% (paramétrable par l’utilisateur) sont intégrés à la base de données.

### Nettoyage des Tweet et Wordcloud

Afin de proposer à Salto un Wordcloud, nous procédons à un traitement des tweets permettant de supprimer la ponctuation, les stopwords, les hashtags, les mentions, etc., faisant ressortir uniquement les mots pouvant représenter un intérêt dans le cadre de notre analyse. L’ensemble du texte, dépourvu des caractères/mots indésirables, est stocké dans une nouvelle colonne pour chaque tweet.

### Import vers BigQuery

L’ensemble du DataFrame est importé sous BigQuery et s’ajoute à la table existante.

# Limites et améliorations envisageables

## Base de films Salto

A l’heure actuelle, la détection de films est réalisée à travers la librairie FuzzyWuzzy sur la base d’un fichier .csv contenant un total de 356 films ou références cinématographiques présents sur la plateforme Salto. Il conviendra de compléter cette base de données dans le futur pour une analyse plus exhaustive.

## Recherche de films

Nous nous sommes heurtés à certains problèmes de matching lors des premiers essais. Ces derniers sont liés au nommage de certains films tels que “vu” ou “face-à-face”. Ces mots apparaissent un très grand nombre de fois dans les tweets et génèrent des erreurs de matching qu’il convient d’éliminer. Dans ce sens, nous avons généré une liste de films à ne pas prendre en compte lors du processus d’identification. Cela implique en revanche l’impossibilité future d’identifier ces films au sein des tweets.

Une importante lacune selon nous concerne les hashtags faisant souvent référence aux films. FuzzyWuzzy n’est pas en mesure de détecter un film sous un format comme #LesMarseillaisaMiami. Une évolution future pourrait notamment viser à dissocier au sein d’un hashtag les différents mots et reconstruire une phrase compréhensible par l’algorithme.

Enfin, la recherche de matchs via Fuzzy Wuzzy est appliquée sur les tweets ‘brut’, sans traitement. Il pourrait être intéressant de tester l’approche sur des tweets ayant déjà fait l’objet d’un preprocessing.

## Base de films complémentaire

Une approche plus globale peut être envisagée, notamment à travers l’exploitation de la base ImdB contenant plusieurs milliers de références de films/séries français. Cela donnerait à Salto des informations importantes sur les films non intégrés sur la plateforme et qui représente un centre d’intérêt pour l’utilisateur. Cette approche a été testée sur la base d'environ 6000 films français, filtrés sur la période [2000 - 2022]. Les résultats obtenus étaient très décevants et pour cause, un grand nombre de films dont le nom contient ou est uniquement constitué de mots courants.

## Secret Management

Dans l’avenir, la gestion des clés pourrait être faite à l’aide de secret Manager.