**3. Machine Learning: Matching candidats ↔ offres**

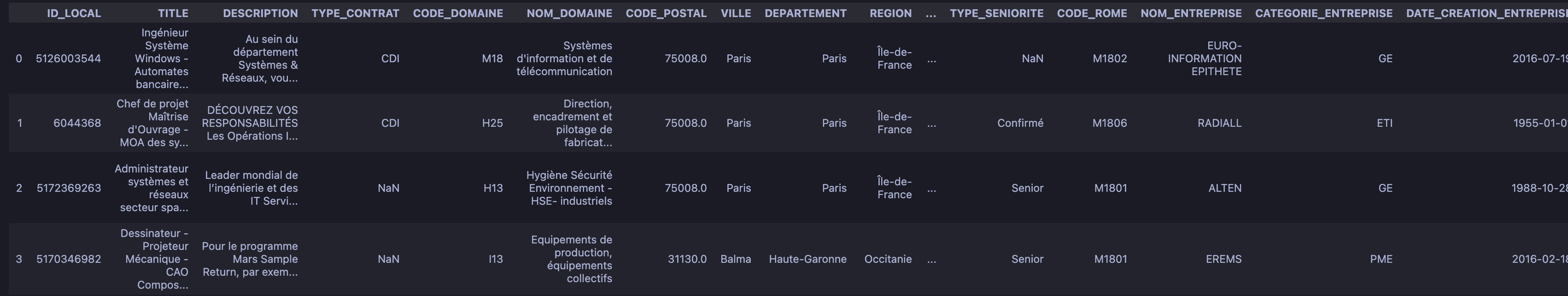
1. **Introduction**

L’objectif principal de cette phase est de développer un système de recommandation permettant de faire correspondre efficacement des profils candidats à des offres d’emploi. Contrairement à une approche de classification supervisée classique, notre démarche s’inscrit dans un cadre de recommandation de contenu non supervisé. Cela implique de travailler sans étiquettes explicites de “bon” ou “mauvais” matching, et de s’appuyer sur la similarité sémantique entre les textes descriptifs des profils et des offres.

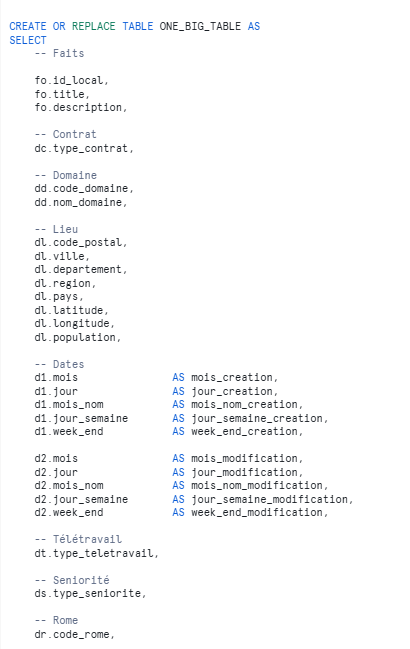
Dans cette optique, nous avons constitué deux jeux de données : une base de 35 000 offres (BigTable Offre) utilisée pour entraîner notre modèle d’analyse textuelle, et un second ensemble de 10 000 profils candidats pour le tester. Les différentes étapes de la construction du pipeline Machine Learning — de l’extraction à la mise en production — ont été orchestrées via MLflow, assurant ainsi un suivi rigoureux des expériences et une reproductibilité du modèle.

Ce rapport détaille la préparation des données, l’exploration, la sélection de caractéristiques, la modélisation sémantique des textes (TF-IDF vs SBERT), le tuning des paramètres (k, batch size, seuil de similarité), et l’évaluation des performances. Enfin, nous abordons les perspectives d’amélioration à travers des stratégies d’hybridation et l’intégration de feedback utilisateur.

1. **Préparation des données (hors EDA)**

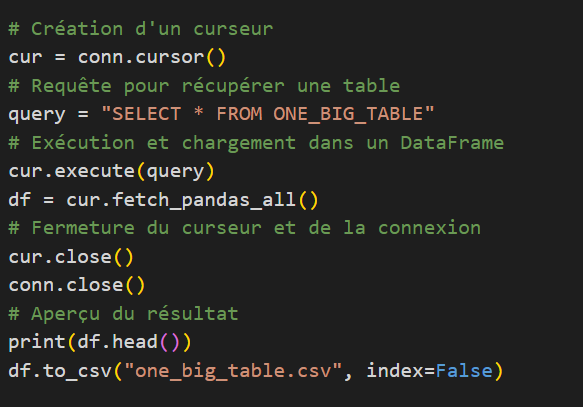
*Schema 1 : Aperçu de la table BigTable Offre.*

Reconstitution d’une « BigTable » en faisant un JOIN de la table de FAIT avec toutes les tables de dimensions :

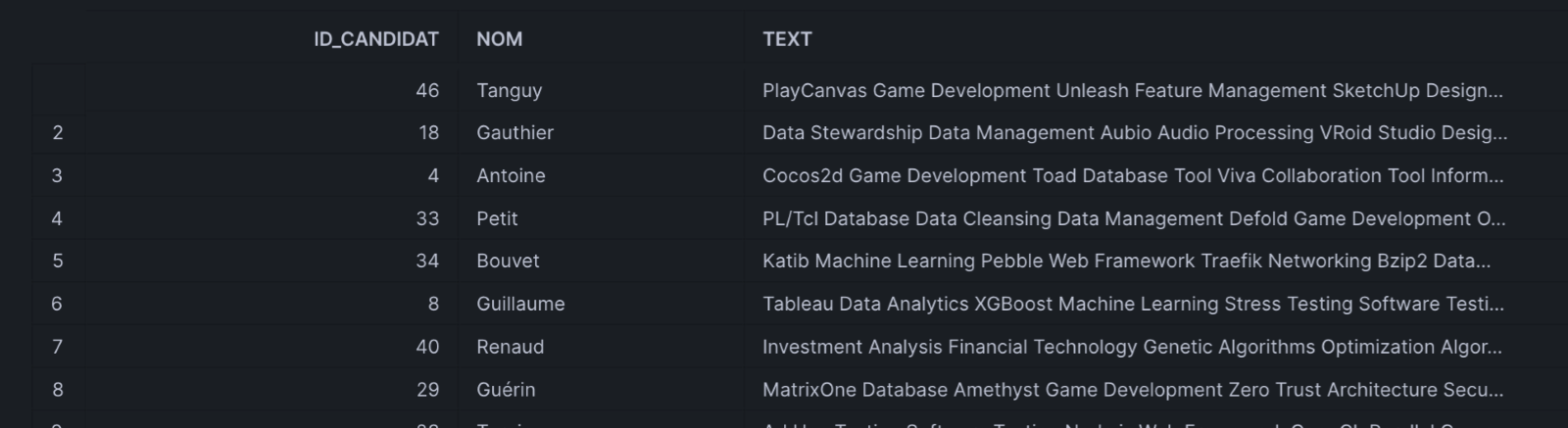
 

*Schema 2 : Requetes Snowflake pour constituer notre « BigTable Offre »*

Puis, nous avons constitué un script python pour aller chercher cette table grâce à snowflake connector :



*Schema 3 : Script python pour récupérer notre BigTable, avec snowflake\_connector*

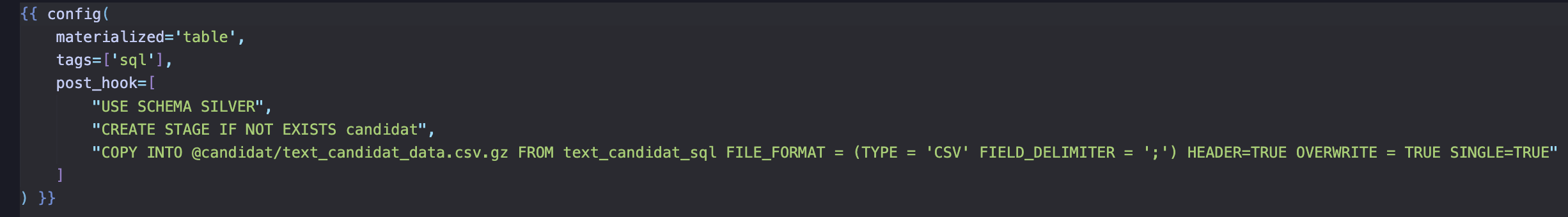
*Schema 4: Aperçu de la table TEXT\_CANDIDAT*

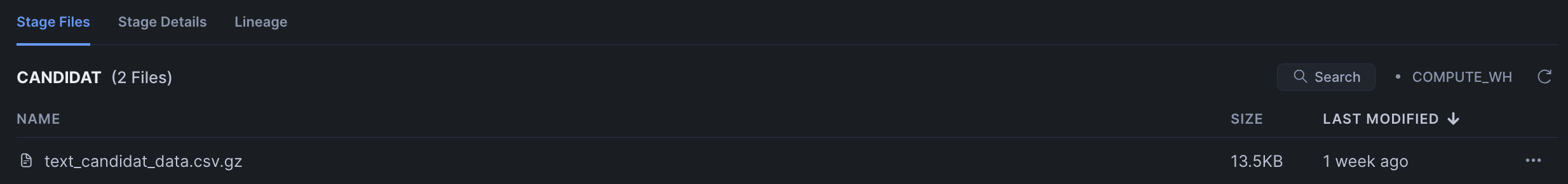
Reconstitution de la table “TEXT\_CANDIDAT” avec la jointure et concaténation de toutes les tables de dimensions en une seul colonne ‘text’



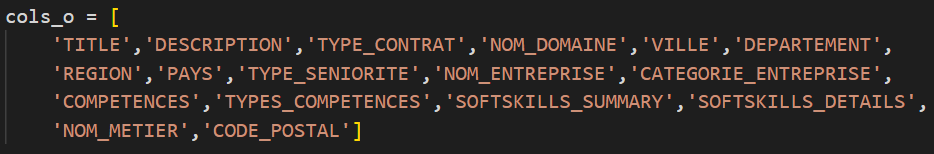
*Schema 5: Requetes DBT dans snowflake pour constituer notre « TEXT\_CANDIDAT»*

Par suite de cela un “post\_hook” vient faire l’export de cette table dans le stage pour avoir notre table text faker candidat.

*Schema 6 : Post Hook DBT pour la récupération en fin de requête la table « TEXT\_CANDIDAT»*

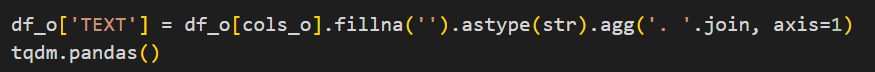
*Schema 7 : Fichier text\_candidat dans le stage.*

On a donc fichier candidat à disposition dans le stage pour tester notre model.

Finalement, voici les colonnes que nous avons décidé de garder pour notre modèle :

*Schema 8 : Choix des colonnes stratégiques pour notre Modèle de ML*

Puis, nous avons regroupé la table Candidat ainsi que la table d’offre sur une seule colonne, afin d’améliorer les performances de notre modèle.

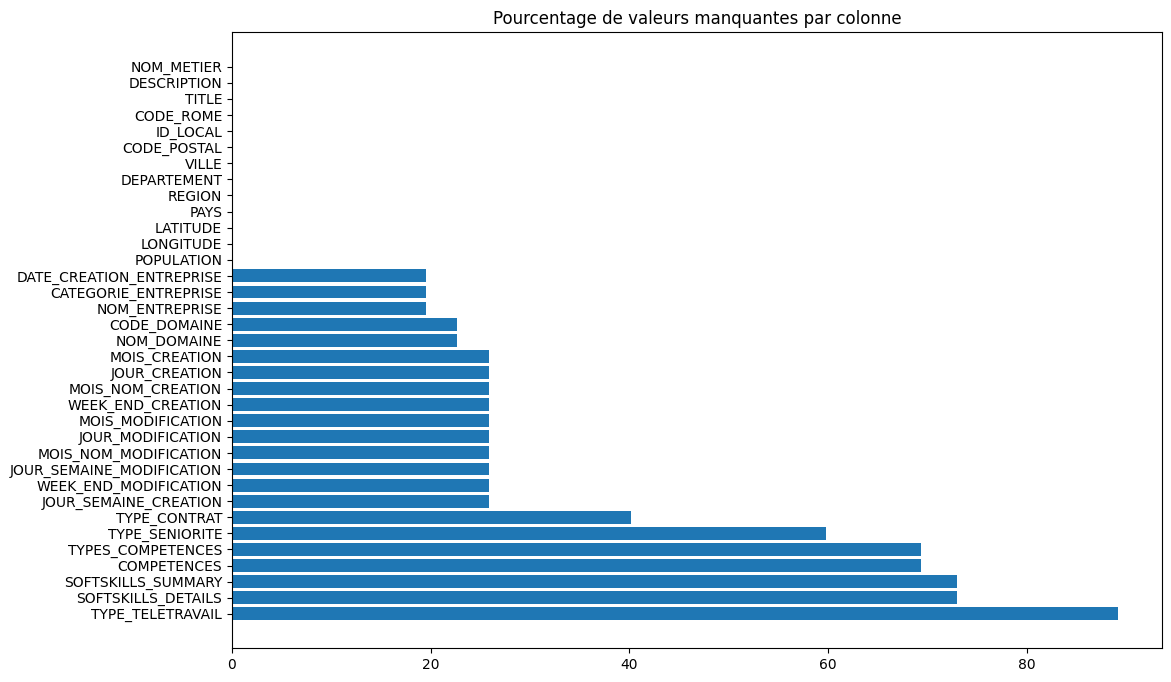


*Schema 9 : Script de concaténation des colonnes*

1. **Exploration et enrichissement (EDA)**

Visualisation de la table des offres :

1. Valeurs manquantes



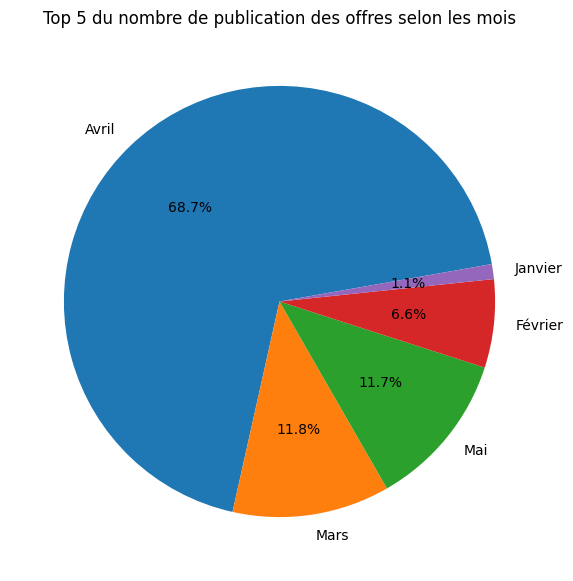
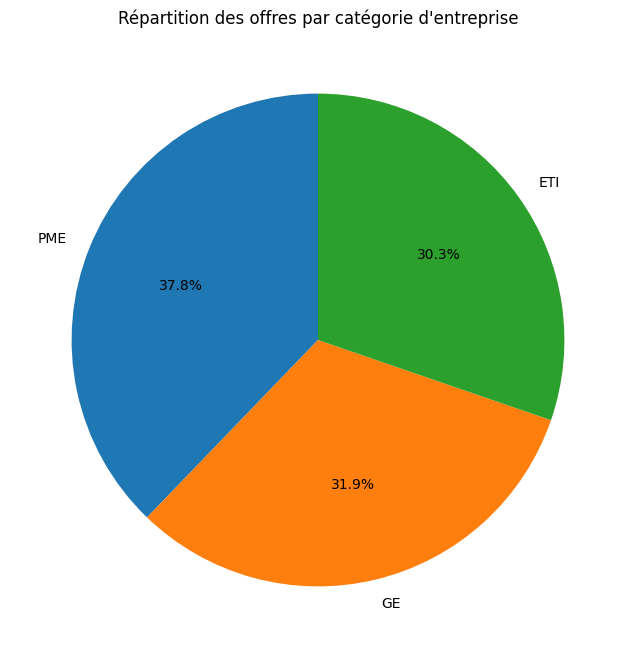


*Schema 10 : Visualisations des valeurs manquantes de la table BigTable Offre.*

Suite à l’analyse des données, nous avons observé que la colonne Type\_Teletravail présentait plus de 80 % de valeurs manquantes. Nous avons donc pris la décision de la **supprimer**, car elle ne contenait pas suffisamment d’informations exploitables pour le modèle.

Puis, nous avons fait un choix stratégique sur les colonnes que nous trouvions pertinentes pour notre modèle de Machine Learning pour le matching. Nous avons donc fait une analyse des différentes colonnes :

Exemple :

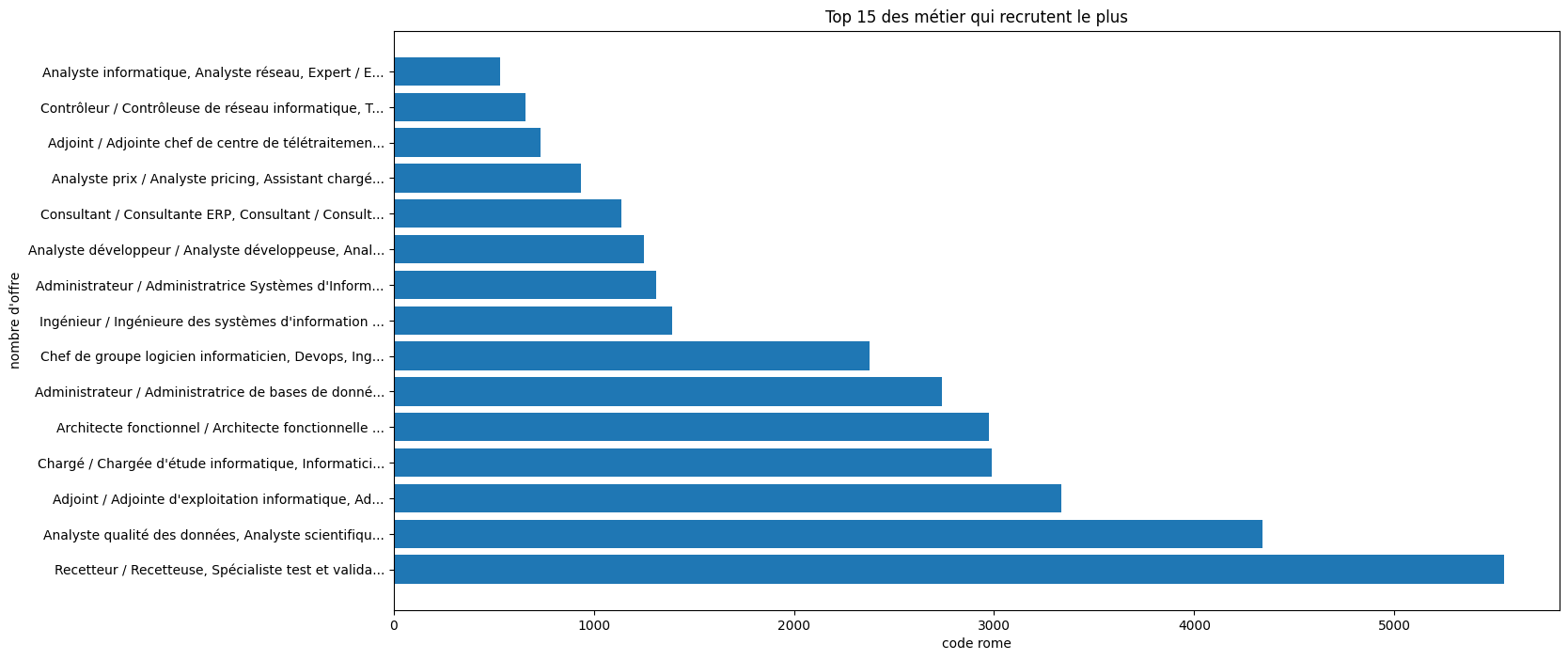
 

*Schema 11: Visualisation du Top 5 du nombre de publication des offres selon les mois et de la répartition des offres par catégorie d’entreprise.*

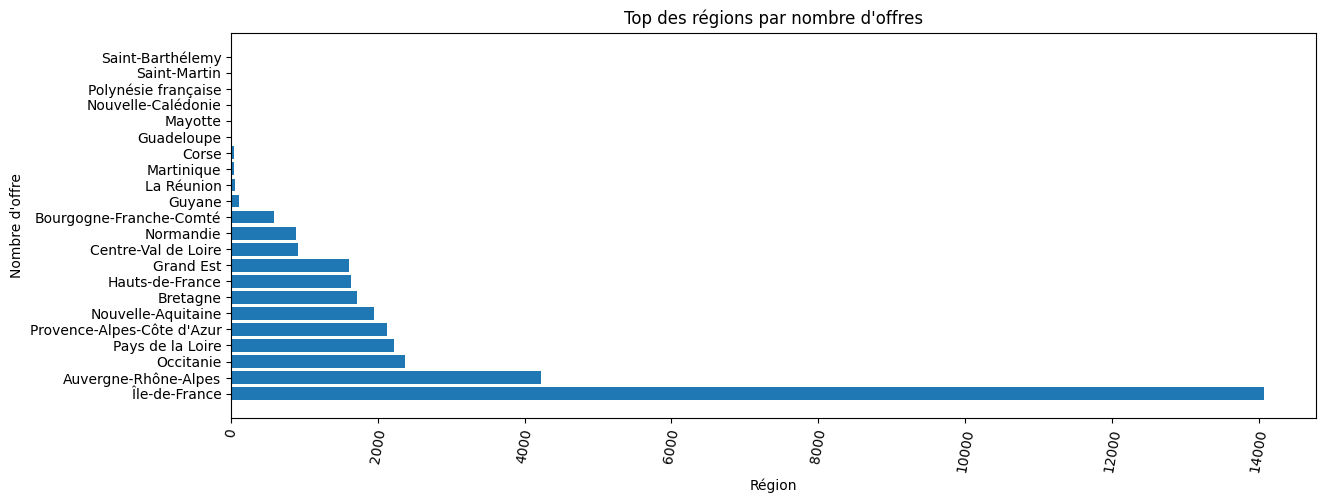
Remarque : le Top 5 est basé sur les données disponibles à partir de mars. Une extraction plus complète sera utilisée dans les prochaines phases du projet. Néanmoins, on constate que le mois **d’avril** concentre le plus grand nombre d’offres publiées.

En ce qui concerne la répartition des offres par catégorie d’entreprise, celle-ci s’avère relativement homogène, sans qu’une catégorie ne domine significativement.

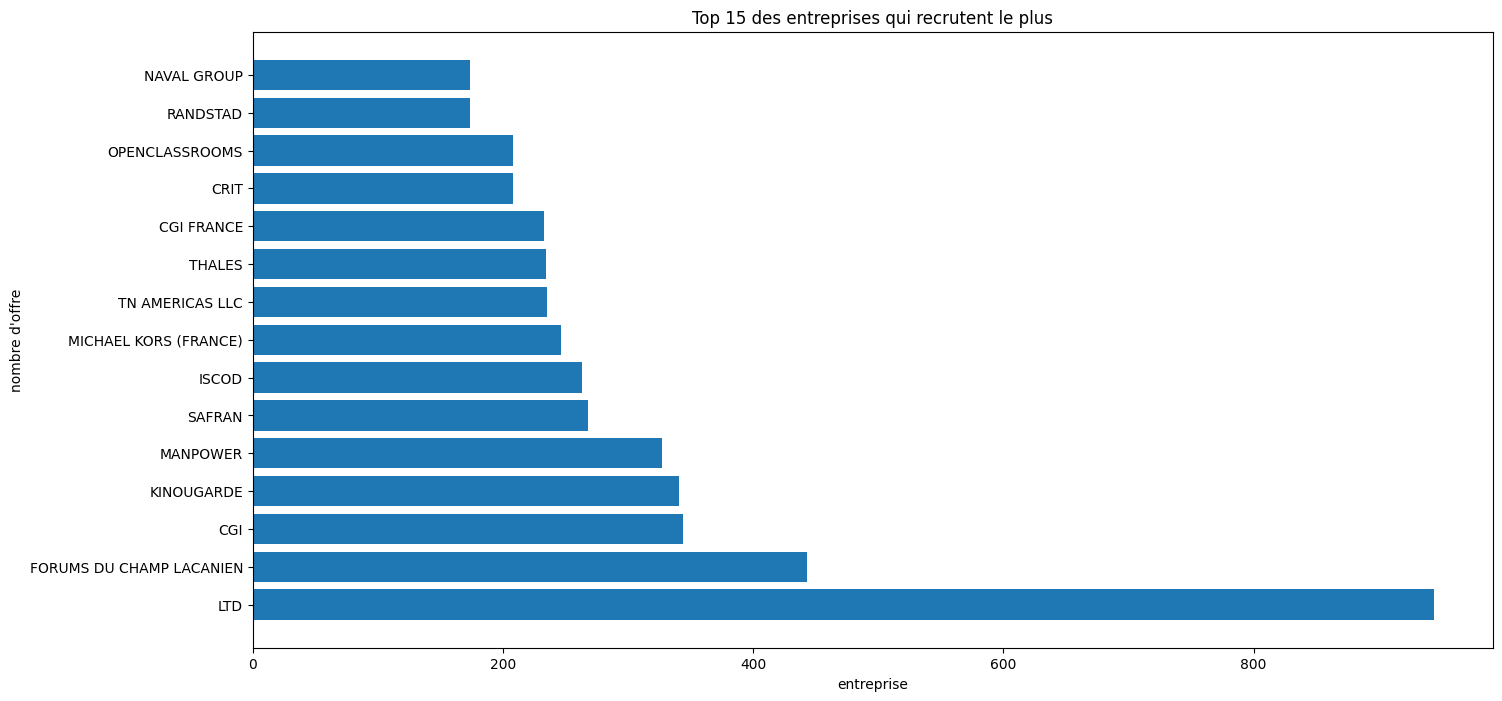
Nous avons ensuite réalisé une analyse plus détaillée en identifiant les **Top 15** selon différents axes :

*Schema 12: Top 15 des métiers les plus représentés.*

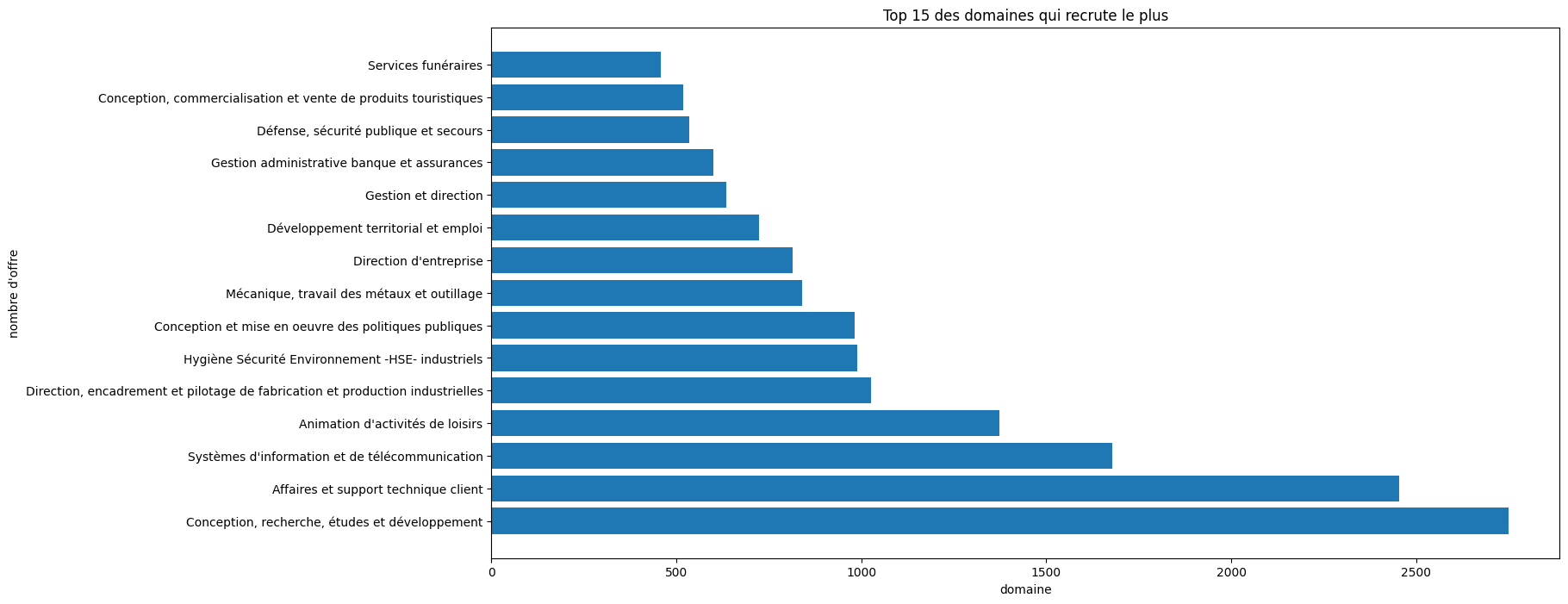
On observe que le métier de recetteur apparaît le plus souvent dans les offres. Les métiers liés à la data (ex. analyste qualité des données, architecte big data, etc.) figurent aussi dans les premières positions.

*Schema 13: Visualisation Top des régions qui recrutent le plus.*

Sans surprise, l’Île-de-France est la région la plus dynamique en termes d’offres publiées. À l’inverse, les DOM-TOM sont sous-représentés dans notre extraction.

*Schema 14: Visualisation du Top 15 des entreprises qui recrutent le plus*

Parmi les plus actives, on retrouve notamment LTD, Forum du Champ Lacanien et CGI.

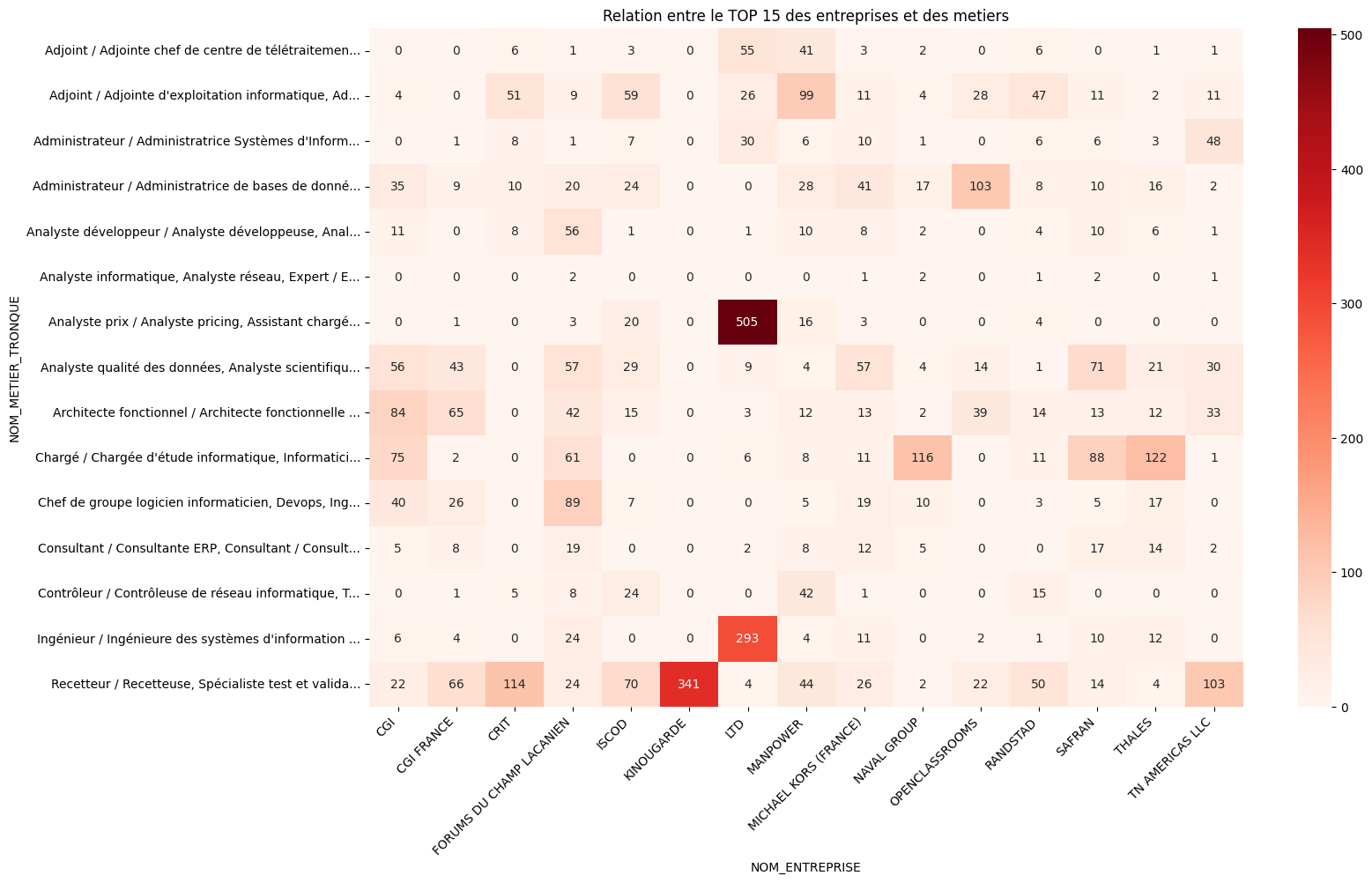
*Schema 15: Visualisation du Top 15 des domaines les plus représentés.*

Les domaines tels que conception, recherche, études et développement sont ceux qui regroupent le plus d’offres.



*Schema 16: Visualisation du Tableau de contingence entre les Top 15 des domaines et les métiers*.

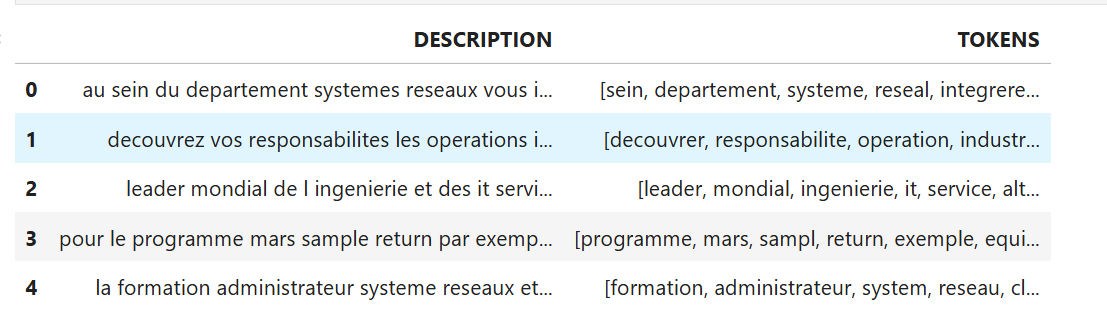
Par la suite, nous avons établi un tableau de contingence entre le Top 15 des domaines et des métiers, où l’on peut observer une concentration des offres selon les domaines d’activité. Par exemple, une part importante des offres liées à la data se regroupe dans les domaines de la conception, recherche, études et développement, ainsi que dans celui des affaires et du support technique client.

*Schema 17: Visualisation du Tableau de contingence entre les Top 15 des entreprises et les métiers.*

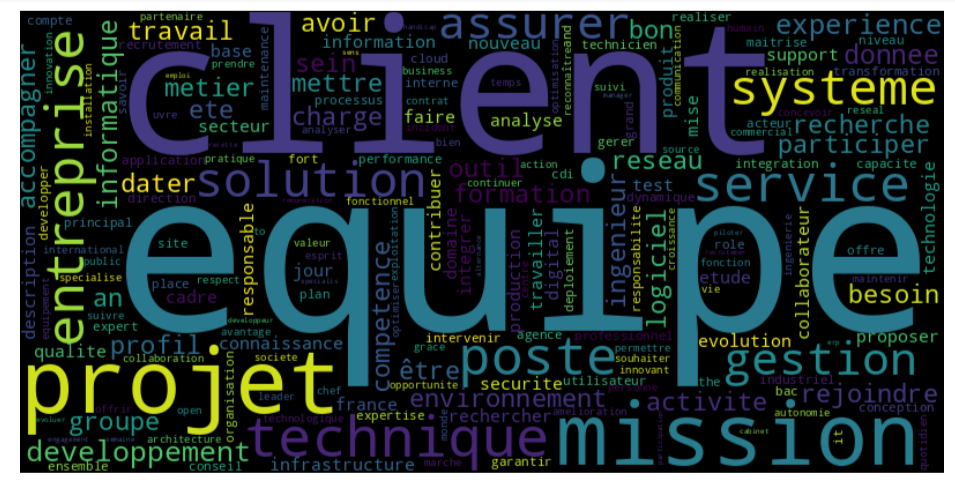
Cette visualisation permet d’identifier quelles entreprises recrutent spécifiquement pour quels types de métiers, offrant ainsi une vue stratégique.

* + Ajustement des stop-words et impact sur le corpus

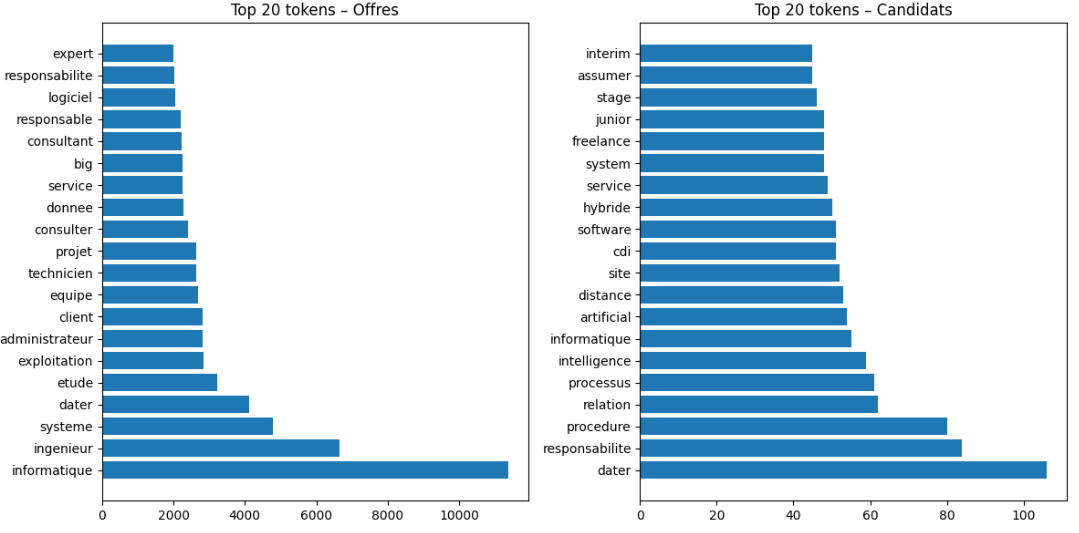
Ain de pouvoir faire une analyse plus poussée de notre modèle, nous allions utiliser les stop -words. Après tokenisation, nous avons pu visualiser les tokens les plus présentes dans notre jeu de données :



*Schema 18 : Tokenisation de notre jeu de données*

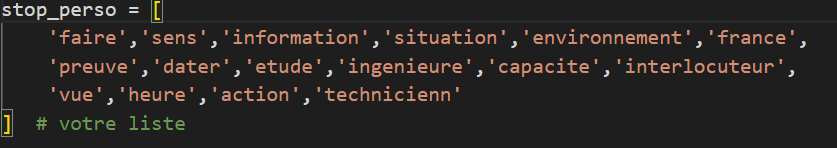


*Schema 19 : CloudWords de notre jeu de données*



*Schema 20 : Top 20 Token Table Offre/Candidat*

Nous sommes arrivés à cette liste de stop words à supprimer :



*Schema 21 : Liste stop words*

1. **Représentation des textes**

TF-IDF vs Berth. Nous avons choisi de partir sur un model Berth plutôt que sur un TF IDF car la principale de notre macing allait se faire sur des phrases, ce que ne fait pas TF\_IDF. AU niverau du modele celui «     sbert\_model = "paraphrase-multilingual-MiniLM-L12-v2" allait nous permettre de faire de pouvoir gérer le fait que nos offres peuvent etre francaise OU anglaise.

1. **Construction de notre pipeline**

Afin de tester nos modèles, nous avons voulu jouer sur 3 paramètres : le nombre de voisin k (knn), le taille de batch et le seul de similarité.

**1. Le nombre de voisins *k* (KNN)**

* **Intérêt** : détermine combien d’offres vous considérez « proches » d’un profil donné avant de rendre un verdict.
* **Effet** :
  + *Petit k* (ex. 1–5) → décisions très localisées, mais sensibles au bruit (une offre aberrante peut fausser le résultat).
  + *Grand k* (ex. 20–50) → vote plus « majoritaire », lissé, mais risque de proposer des offres trop génériques.
* **Trade-off** : trouver le juste milieu pour capturer la diversité sans diluer la pertinence.

**2. La taille de batch (batch size)**

* **Intérêt** : nombre de profils/offres traités simultanément lors du calcul d’embeddings ou du scoring.
* **Effet** :
  + *Batchs petits* → plus de passes mémoire, latence unitaire élevée, mais consomment moins de RAM GPU/CPU.
  + *Batchs grands* → meilleures performances globales grâce à l’optimisation matricielle (vectorisation), mais risque d’OOM si vous manquez de mémoire.
* **Trade-off** : maximiser le débit (throughput) tout en restant dans les limites de votre infrastructure.

**3. Le seuil de similarité (threshold)**

* **Intérêt** : valeur de coupure au-delà de laquelle vous considérez qu’une offre est suffisamment « similaire » à un profil.
* **Effet** :
  + *Seuil haut* (ex. cosinus > 0,8) → recommandations très strictes, forte précision mais faible rappel (peu d’offres proposées).
  + *Seuil bas* (ex. cosinus > 0,3) → plus d’offres, meilleur rappel, mais risque d’inclure des propositions peu pertinentes.
* **Trade-off** : équilibre précision vs. rappel selon l’usage (ex. « Top 5 » vs. « Toutes les offres > seuil »).

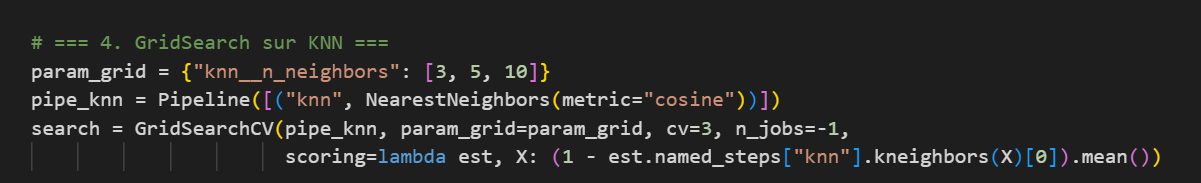
**En pratique**

* **Grid search rapide** : tester k ∈ {5, 10, 20}, batch size ∈ {16, 64, 256}, threshold ∈ {0,4; 0,6; 0,8}.
* **Évaluation** : mesurer precision@k ou recall@k sur un jeu de validation (simulateur de candidats) pour repérer la combinaison optimale.

Ces trois paramètres agissent sur des volets différents du pipeline :

* *k* sur la logique même du matching,
* batch size sur la performance d’exécution,
* threshold sur la qualité finale des recommandations.

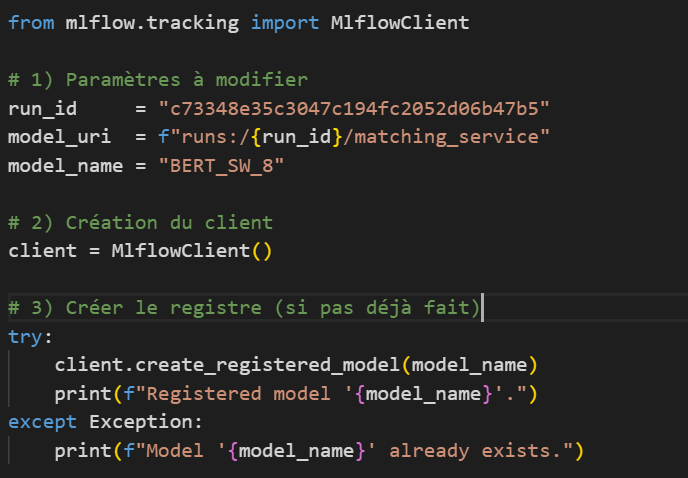
Ainsi, à l’aide du module GridSearchCV, nous avons fait tourner notre modèle sur ces 3 paramètres, et celui-ci nous à trouvé le plus performant .



*Schema 22: Script de GridSearch sur knn*

1. **Intégration MLFlow**

Après implémentation sur MLflow, nous avons pu comparer les différents modèles. Nous avons mis en production celui qui était le plus pertinent. Afin de mettre en production, nous devons choisir le nom du modèle ainsi que le nom de l’id présent dans les expérimentation d’ML Flow.

****

*Schema 23 : Script de mise en production*

1. **Tests de bout en bout**
   * Exemple de requête “profil fictif → top-5 offres”
   * Extraits de code ou requêtes via l’API de test
2. **Conclusion et perspectives**
   * Bilan de la phase ML (performances, robustesse)
   * Prochaines améliorations envisagées (hybridation plus poussée, feedback utilisateur…)