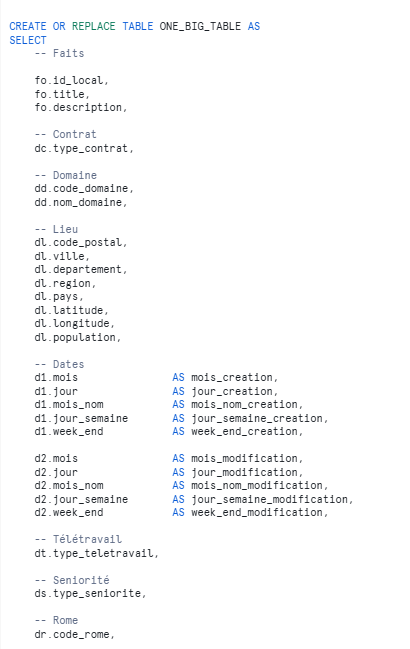
**3. Machine Learning: Matching candidats ↔ offres**

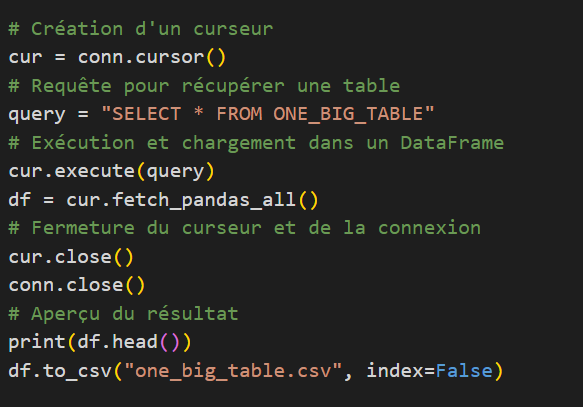
1. **Introduction**
   * Rappel rapide de l’objectif : proposer des offres pertinentes à un profil candidat
   * Limites et périmètre (cas de recommandation de contenu, pas de target supervisée classique)
2. **Préparation des données (hors EDA)**
   * Snapshot à l’instant de notre schéma étoile gold sur Snowflake .

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



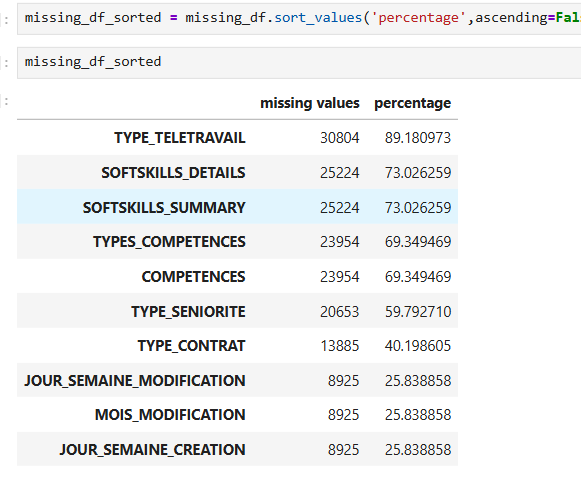
*Schema 1 : Requetes Snowflake pour constituer notre « BigTable Offre »*

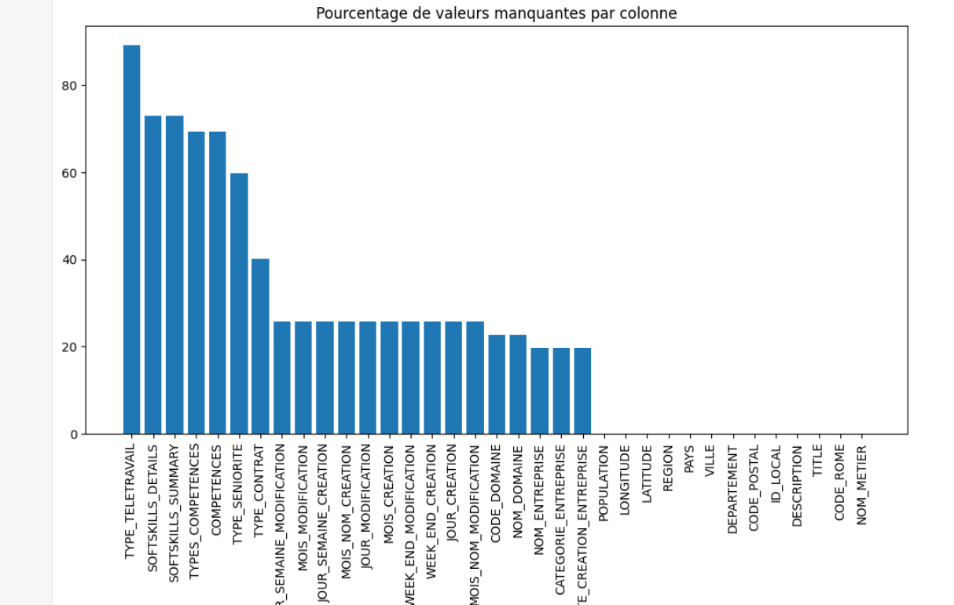
Puis, nous avons constitué un script python pour aller chercher cette table grâce à snowflake connector :



*Schema 2 : Script python pour récupérer notre BigTable, avec snowflake\_connector*

* Visualisation de la table des offres :

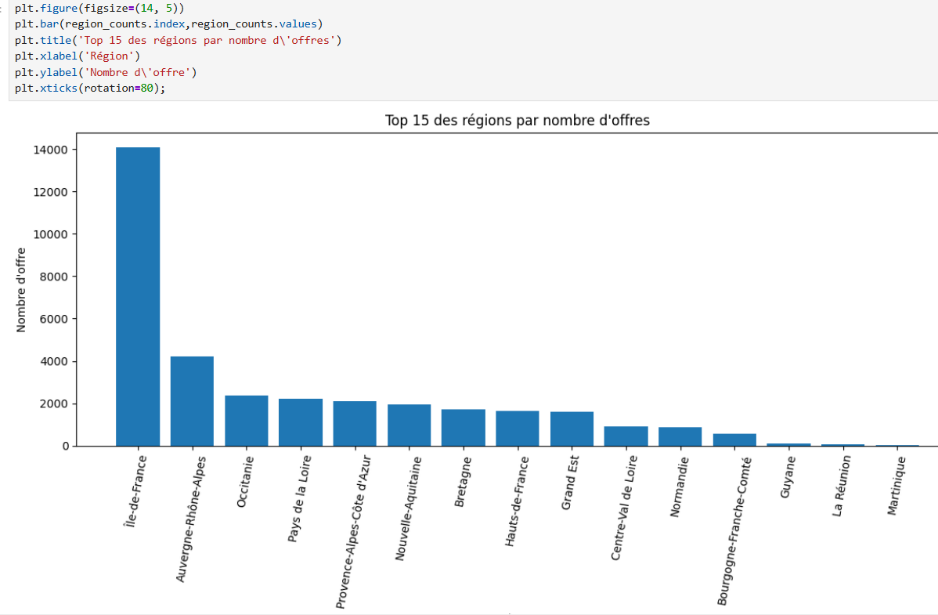
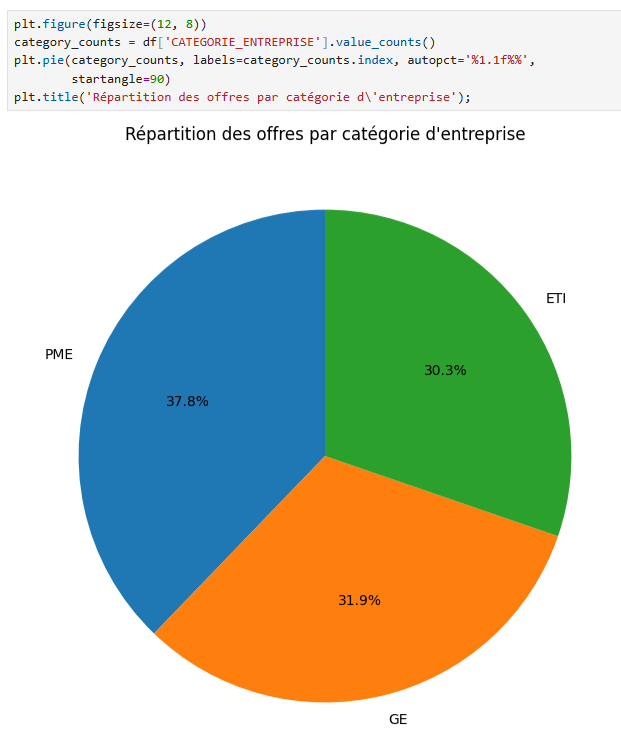
1. Valeures manquantes

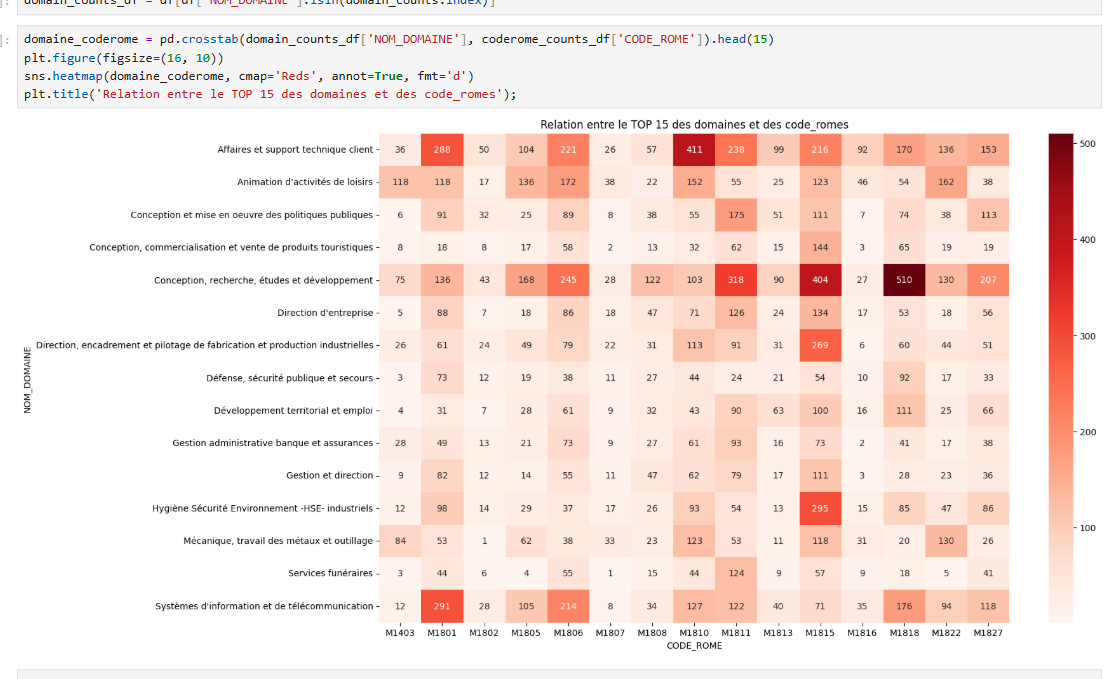


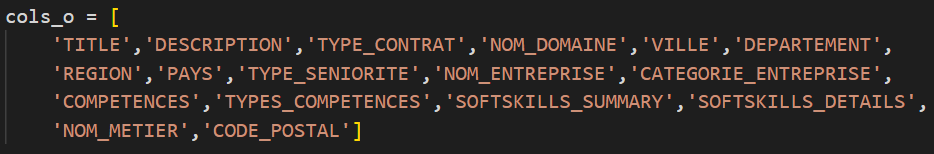
*Schema 3 : Notebook analyse du profil de notre Table*

Conclusion : Suppression de la colonne « Type\_Teletravail » car +80% de valeures manquantes

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

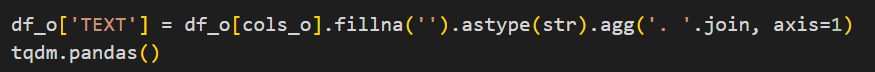
Exemple :



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

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

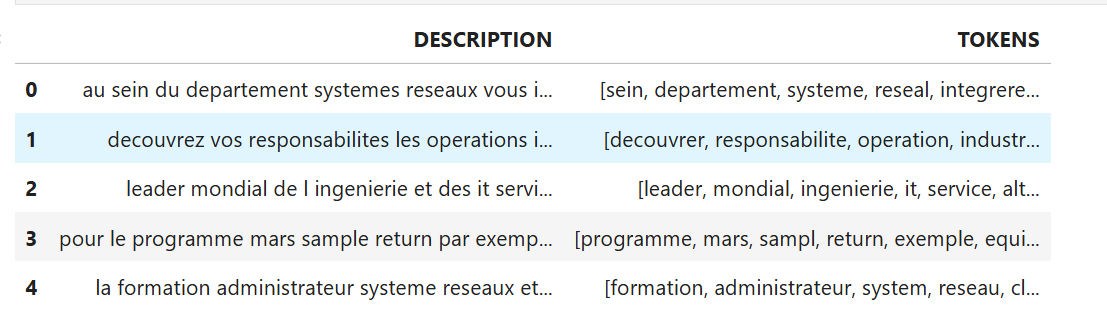
Puis, nous avons regroupé la table Candidat ainsi que la table d’offre sur une seule colonne, afin d’améliorer les performances de notre modèle.



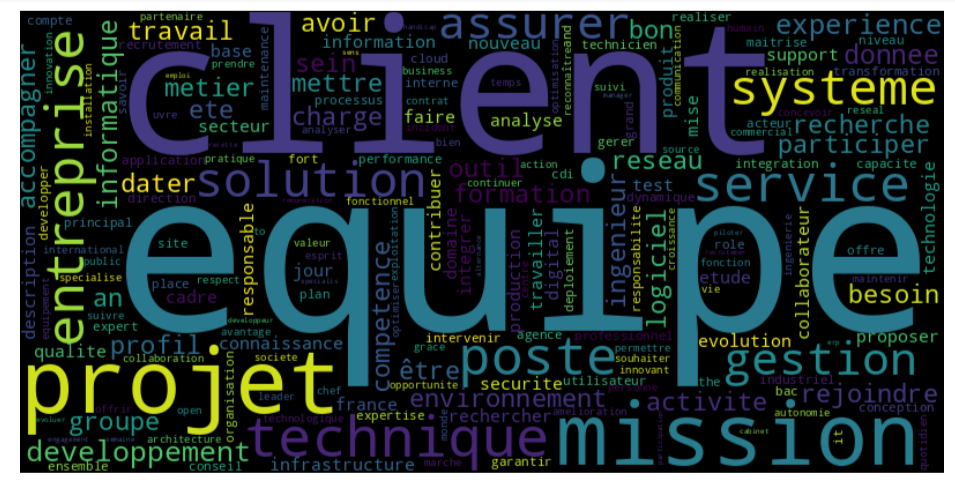
*Schema x : Script de concaténation des colonnes*

1. **Exploration et enrichissement (EDA)**
   * Ajustement des stop-words et impact sur le corpus

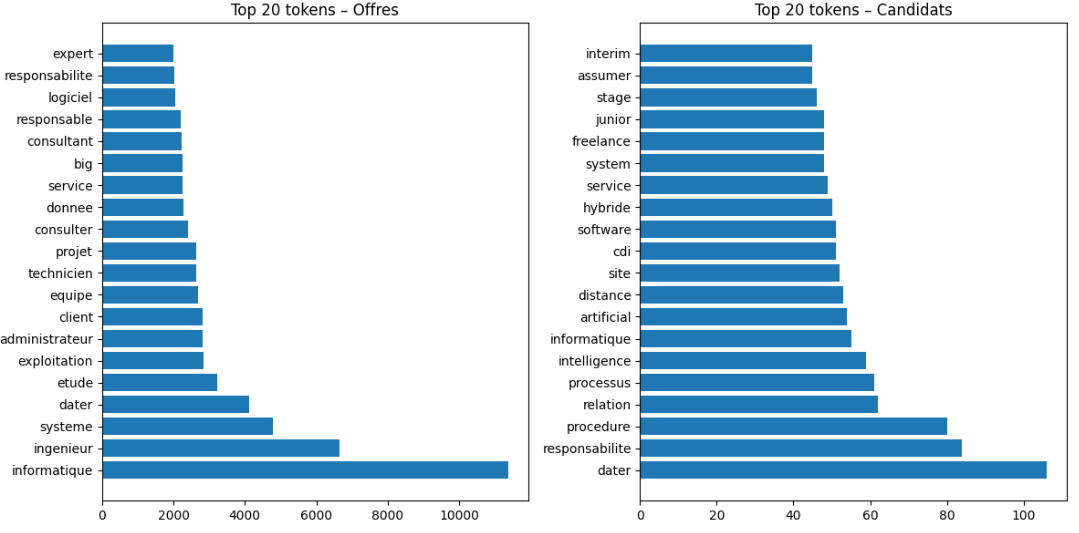
Ain de pouvoir faire une analyse plus poussé de notre modèle, nous allions utiliser les stop -words. Après tokenisation, nous avons pu visualiser les tokens les plus présentes dans notre jeu de données :



*Schema x : Tokenisation de notre jeu de données*

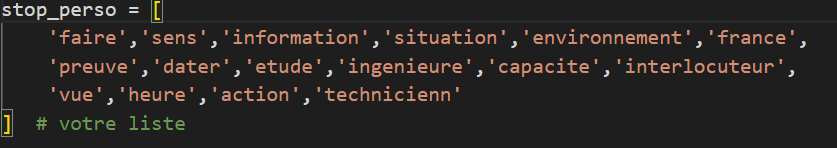


*Schema x : CloudWords de notre jeu de données*



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

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



*Schema x : 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 »).

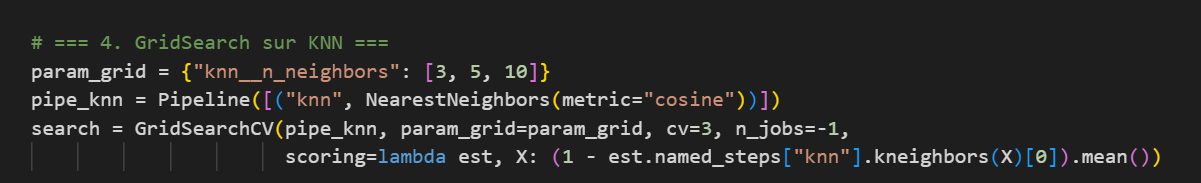
**En pratique**

* **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.

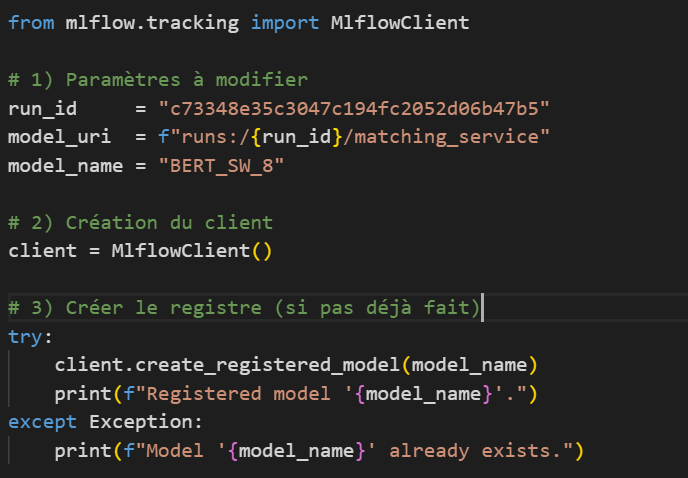
Ainsi, à l’aide du module GridSearchCV, nous avons fait tourner notre modèle sur ces 3 paramètres, et celui-ci nous à trouvé le plus performant .



*Schema x : 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 x : 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…)