Nous allons nous intéresser au traitement d'un flux de données simplifié:

- Nous disposons d'un jeu de données au format .csv fourni avec ce document (*transactions.csv*); il s'agit d'une version simplifiée de transactions bancaires à traiter.
- Nous disposons également d'un algorithme basique dont la vocation est d'effectuer deux traitements **distincts** et **indépendants** (détaillés plus bas). Le cœur de cet algorithme est également fourni avec ce document (*algorithm.py*) et est écrit en utilisant le framework **metaflow** (un orchestrateur de tâches organisées en Directed Acyclic Graph).
- Le **premier objectif** sera de permettre l'exécution de cet algorithme en complétant le graphe des tâches dans le fichier joint (on pourra librement s'inspirer des différents tutoriels et exemples présentés dans la documentation de **metaflow** à cette fin). La manière de découper les tâches manquantes, leurs noms etc... est laissée libre. En revanche, toutes les données manipulées et produites devront être stockées dans une structure de données robuste et adaptée au problème (plus bas suivront des suggestions à ce propos).
- Le **second objectif** sera de proposer une solution de monitoring du processus de traitement, permettant de suivre son bon déroulement et ayant la capacité de relever des erreurs, anomalies, ou alertes qu'il aura été jugé bon de suivre. Essentiellement, il faut s'imaginer que ce flux de données s'inscrit dans un contexte business critique d'un volume important et où chaque transaction est censée être traitée correctement.

La logique de l'algorithme en question est très simple:

- En entrée, des transactions avec certaines grandeurs d'intérêt (en somme: un montant et un libellé bancaire), **qu'il va s'agir de récupérer de la structure de données retenue pour le stockage**, qui vont être **taguées** d'une part, et **annotées** d'autre part (de manière indépendante).
- Pour ce qui est de l'apposition d'un **tag:** on dispose d'une liste de noms de tags donnée au préalable, l'algorithme cherche dans le libellé s'ils sont présents et renvoie un résultat s'il n'y en a qu'un seul identifié; **lorsque c'est le cas, on enregistrera l'identifiant de la transaction ainsi que le nom du tag en question**.
- Pour ce qui est de l'**annotation**, l'algorithme renvoie:
    - "LARGE SALE" si le montant est strictement supérieur à 300€;
    - "SALE" si le montant est entre 0 et 300€;
    - "SMALL EXPENSE" si le montant est entre 0 et -200€;
    - "EXPENSE" si le montant est strictement inférieur à -200€.

  Au même titre que pour l'apposition d'un tag, **on enregistrera alors l'identifiant de la transaction ainsi que l'annotation en question**.
- Chaque flux de résultats produit devra donc être stocké dans la structure de données utilisée pour les besoins du projet.

Suggestion concernant cette structure de données:

On pourra mettre en place une base de données basique à l'aide de **sqlite3**, dans laquelle insérer les données issues du .csv fourni avec ce document, puis utiliser cette dernière afin de gérer tout l'écosystème de données.
Si tel est le choix retenu, on fournit avec ce document (*sqlite.py*) un exemple très simpliste d'une fonction qui exécute des instructions à la suite, les commit, et ferme la connexion; on pourra librement s'inspirer de cet exemple pour les besoins du projet.