Rakib SHEIKH
I1 EISI, IA Big Data et Cybersécurité
noobzik@pm.me

Projet : ATL-Datamart Version Python

Projet pour :
- Cours d'atelier Architecture décisionnel Datamart (TRDE704) pour les I1 de l'EPSI Paris et Arras.
- Lien vers le template du github : https://github.com/Noobzik/ATL-Datamart

Le sujet est à disposition dans le dossier docs ET le sujet à jour dans votre espace learning.

L'objectif de ce TP sera de déployer une architecture de type CAC40 afin de vous familariser avec les termes d'une architecture Big Data.

Pour cela, nous allons tout d'abord collecter des données. Mais ces données doivent être stockés quelque part. Comme nous ne savons pas de la valeur qu'elle peut générer, nous devons alors le stocker dans un **Datalake** représenté ici par le service Minio.

Une fois que les données stockés sur Minio, nous supposons que nous avons trouvé une valeur potentielle des données à exploiter, nous devons donc préparer les données brutes depuis un Minio vers une base de données représantant un **Data Warehouse**. Cette partie définie donc une intégration de données.

Maintenant, vous vous rendez compte que votre équipe a rédigé un contrat d'intégration de données sous la forme d'un modèle en flocon. Comme vous n'êtes pas sûr que cette donnée sera réutilisé par les équipe externe à la votre, vous allez donc vous l'approprier pour votre équipe qui possède sa propre base de donnée. Cette base de donnée est en réalité un sous-ensemble du Data Warehouse, qui est physiquement détaché. C'est donc l'appelation d'un **Data Mart**.

Enfin, les données sont nettoyés et prêtes à être visualisés, vous allez donc faire appelle à un outil de visualisation de données de type Tableau voire PowerBI.

Le descriptif des 5 TPs :

- **Pour le TP 1 : La collecte des données vers un datalake**
  - L'objectif du TP1 est de récupérer les données des taxi jaunes de la ville de New York, actualisés à chaque mois, directement sur le site d'état de New York. Afin de récupérer les données, nous téléchargeons un fichier parquet du mois en question qui sera stocké vers un Datalake. Le rôle du datalake sera joué par un service Docker s'appelant Minio. Minio est un service permettant de stocker des objets visant à remplacer l'architecture de stockage distribué Apache Hadoop. Comme c'est une copie parfaite du service AWS S3, l'ensemble des fonctions que vous utiliserez pour Minio seront également applicable pour AWS S3.
  - Afin de simplifier l'acheminement du fichier parquet vers le bucket Minio, le TP 1 sera réalisé en deux temps.
    1. Téléchargez un fichier parquet et stockez le dans le dossier data/raw (depuis la racine du projet template).
    2. Uploadez un fichier parquet que vous venez de télécharger dans le dossier data/raw/ vers votre bucket Minio.
  - Il faudra utiliser le fichier qui se situe à `src/data/grab_parquet.py` et compléter les fonctions qui sont vides.
  - Remarque : Ne vous cassez pas la tête à effacer les fonctions. Sinon, vous allez augmenter exponentiellement la difficulté de réaliser ce TP.
- **Pour le TP 2 : Du datalake vers un datawarehouse.**
  - L'entreprise du CAC40 fictive a finalement pu trouver une finalité amenant à utiliser les fichiers parquets dormants dans le Datalake. Nous allons devoir commencer à récupérer ces fichiers parquets qui sera intégré vers un nouveau service de base donnée postgresql jouant le rôle de Data Warehouse.

- Pour réaliser cette intégration de donnée, il vous est proposé deux approches :
  - Vous pouvez concevoir un scripting python, prêt à être mis en production afin de récupérer un fichier parquet et permettant de réaliser une ingestion de données vers postgresql Data Warehouse. Il faudra utiliser le fichier `src/data/dump_to_sql.py`.
    - *L'implémentation actuel permet de prendre les fichiers parquets sauvegardés en local. Vous devriez modifier le programme pour qu'il récupère les fichiers parquets que vous avez stockés dans Minio.*
  - La deuxième approche consiste à utiliser un logiciel de type ETL. Cependant, il n'existe que 3 ETL permettant de réaliser cette tâche : **KNIME, Talend (Remplacé par Talaxie), Amphi.ai**.
- **Pour le TP 3: Datamart et création d'un modèle multi-dimensionnelle.**
  - À ce stade, nos données sont prêtes à être requêtés par notre équipe métier, mais un problème se pose : notre schéma de donnée actuel n'est pas optimiser pour l'analyse de données. En effet, en l'état, notre table est orientée OLTP, ce qui signifie qu'elle est optimisé pour ajouter une course. Nous allons devoir transformer notre table vers un modèle en étoile, flocon ou constellation, dont vous justifierez votre choix dans le rapport finale. Une contrainte : vous êtes dans l'obligation d'utiliser le langage SQL du SGBD de votre choix.
  - Par soucis de simplicité du sujet, vous êtes libre utiliser le SGBD de votre choix sans tenir compte des propriété OLAP.
  - Vous aurez donc un script SQL pour chaque tâche distinct :
    - `creation.sql` pour la création des tables en flocons avec les contraintes associés.
    - `insertion.sql` pour insérer les données depuis votre base de donnée `Data Warehouse` vers votre base de donnée `Data Mart`.
      - Remarque : C'est bien DEUX SERVEURS SGBD distinct ET NON DEUX BASES DE DONNEES !
      - *Le non respect de ce dernier point vaudra 0 pour ce TP.*
- **Pour le TP 4 : Visualisation de données**
  - Lorsque vous avez fait le TP3, vous devriez normalement avoir une idée sur la restitution des données que vous souhaitez faire dans la partie visualisation de données.
    - Si ce n'est pas le cas, vous pouvez ouvrir un Notebook qui sera sauvegardé dans le dossier `notesbooks` pour réaliser votre Analyse Exploratoire de Données (EDA) (*Renseignez vous sur le cours de Data Science & Machine Learning pour en savoir plus*).
    - Pour les plus chaud d'entre vous, vous pouvez concevoir un tableau de bord à l'aide de Streamlit.
  - Vous devez connecter votre outil de Data Visualisation à votre base de donnée `Data Mart` afin de produire les visualisations.
- **Pour le TP 5 : Introduction à l'orchestration d'une pipeline Data Engineering via Airflow**
  - Cette partie du TP vous servira d'introduction à l'orchestration des tâches d'un projet Big Data. C'est-à-dire de lancer des scripts python de manière totalement automatisée sur un interval définie.
  - Pour le moment, je vous demande de réaliser une dag qui permet de télécharger un parquet du dernier mois en vigueur (TP 1) et de le stocker vers Minio. Elle est à compléter dans le fichier `airflow/dags/minio.py`.
  - Une fois que vous avez compris le fonctionnement des dags, vous pouvez vous amuser à automatiser le TP 2 et 3 afin de rendre le TP 4 totalement autonome.

Conseils :

- Afin d'assurer la reproductibilité du code au niveau des chemins fichiers, vous devez utiliser la librairie `os.path.lib`.
- La gestion des dates peut sembler fastidieuse sous python, l'astuce ici est d'utiliser la librairie `pendulum` permettant de simplifier la manipulation des dates. Elle est obligatoire pour l'utilisation d'Airflow à titre d'information.

Rendu :

- Un lien github vers votre projet
- **Un pull request fera office de rendu officiel. L'absence d'un pull request donnera lieu à un 0 systématique.**

- Par mail ainsi que dans le dossier docs, un rapport écrit décrivant l'ensemble des actions menés pour réaliser les 5 TP ainsi que des captures d'écrans

Règle de retard :

- Le rendu sera fixé par votre intervenant, à 6h00 du matin.
- Chaque heure de retard entamé sera sanctionné de −5 points. La sanction est appliqué dès 6h00m00s.

Annexes

```
    ├── airflow
    │   ├── config       <- Configuration files related to the Airflow Instance
    │   ├── dags         <- Folder that contains all the dags
    │   ├── logs         <- Contains the logs of the previously dags run
    │   └── plugins      <- Should be empty : Contains all needed plugins to make the dag work
```

```
Project Organization
------------
    ├── airflow
    │   ├── config       <- Configuration files related to the Airflow Instance
    │   ├── dags         <- Folder that contains all the dags
    │   ├── logs         <- Contains the logs of the previously dags run
    │   └── plugins      <- Should be empty : Contains all needed plugins to make the dag work
    ├── LICENSE
    ├── Makefile         <- Makefile with commands like `make data` or `make train`
    ├── README.md        <- The top-level README for developers using this project.
    ├── data
    │   ├── external     <- Data from third party sources.
    │   ├── interim      <- Intermediate data that has been transformed.
    │   ├── processed    <- The final, canonical data sets for modeling.
    │   └── raw          <- The original, immutable data dump.
    ├── docs             <- A default Sphinx project; see sphinx-doc.org for details
    ├── models           <- Trained and serialized models, model predictions, or model summaries
    ├── notebooks        <- Jupyter notebooks. Naming convention is a number (for ordering),
    │                       the creator's initials, and a short `-` delimited description, e.g.
    ├── references       <- Data dictionaries, manuals, and all other explanatory materials.
    │
    ├── reports          <- Generated analysis as HTML, PDF, LaTeX, etc.
    │   └── figures      <- Generated graphics and figures to be used in reporting
    │
    ├── requirements.txt <- The requirements file for reproducing the analysis environment,
    │                       e.g. generated with `pip freeze > requirements.txt`
    │
    ├── setup.py         <- makes project pip installable (pip install -e .) so src can be
    │                       imported
    ├── src              <- Source code for use in this project.
    │   ├── __init__.py  <- Makes src a Python module
    │   │
    │   ├── data         <- Scripts to download or generate data
    │   │   └── make_dataset.py
    │   │
    │   ├── features     <- Scripts to turn raw data into features for modeling
    │   │   └── build_features.py
    │   │
    │   ├── models       <- Scripts to train models and then use trained models to make
    │   │   │               predictions
    │   │   ├── predict_model.py
    │   │   └── train_model.py
    │   │
    │   └── visualization <- Scripts to create exploratory and results oriented visualizations
    │       └── visualize.py
    │
    └── tox.ini          <- tox file with settings for running tox; see tox.readthedocs.io
```