# EPSI : école privée des sciences informatiques

DEPARTEMENT INFORMATIQUE

Première Année

# ATL-Datamart

## Rapport de projet


Rapport réalisé par :

**BAZIZ Ilhem**

**LEKHAL Massinissa**

**BELABBAS Madani Wassim**

Responsable de l'UE :

**SHEIKH Rakib**


*Année universitaire :* 2024/2025

# ***Introduction :***

Dans le cadre de notre formation en architecture décisionnelle, nous avons mené à bien un projet visant à concevoir et déployer une architecture Datamart complète. Ce projet s'inscrit dans une approche pratique et immersive, avec pour objectif de maîtriser les différentes étapes du traitement des données, depuis leur extraction jusqu'à leur restitution sous forme de tableaux de bord interactifs.

Le projet, centré sur une entreprise de VTC basée à New York, avait pour but de transformer des données brutes disponibles sur des plateformes publiques en un ensemble de connaissances exploitables. Pour ce faire, nous avons mis en œuvre une architecture reposant sur les principes suivants :

- L'automatisation de la récupération des données via un Data Lake.
- La transformation et le stockage des données dans un Data Warehouse.
- La création de Data Marts optimisés pour la visualisation.
- La conception de tableaux de bord dynamiques à l'aide d'outils de Business Intelligence tels que Tableau ou Power BI.

Ce document retrace les différentes étapes du projet, de la mise en place des flux ETL (Extract, Transform, Load) à l'élaboration d'une infrastructure performante et optimisée. Il met également en lumière les défis rencontrés et les solutions adoptées, ainsi que les résultats obtenus à travers les visualisations développées. Enfin, il explore les perspectives d'évolution et les bonnes pratiques pour assurer la pérennité de ce type d'architecture.

# ***Objectifs principaux :***

Ce projet vise à répondre à des problématiques concrètes liées à l'exploitation de données massives pour la prise de décision. Les objectifs principaux du projet sont les suivants :

**1- Automatiser la récupération des données**

- Intégrer des jeux de données provenant de sources publiques (site de l'État de New York) dans un Data Lake à l'aide d'outils tels que MinIO.
- Développer des scripts capables d'automatiser cette collecte.

**2- Mettre en œuvre des flux ETL efficaces**

- Transformer les données brutes issues du Data Lake en données propres et prêtes à être stockées dans un Data Warehouse.
- Automatiser les flux d'extraction, de transformation et de chargement (ETL) pour garantir la cohérence et la fiabilité des données.

**3- Concevoir une architecture de Data Mart performante**

- Mettre en place une structure en flocon pour le stockage des données, adaptée à des requêtes analytiques rapides.
- Exploiter une base OLAP pour optimiser les performances des analyses décisionnelles.

**4- Développer des tableaux de bord dynamiques**

- Identifier les indicateurs clés de performance (KPI) à partir des données disponibles.
- Créer des visualisations interactives à l'aide d'outils de BI (Tableau, Power BI) pour fournir des insights pertinents.

**5- Introduire des mécanismes d'automatisation avancés**

- Implémenter une DAG avec Apache Airflow pour automatiser l'ensemble du processus, en assurant une exécution planifiée et reproductible.

Ces objectifs s'inscrivent dans une démarche visant à acquérir les compétences nécessaires à la conception et au déploiement d'une architecture décisionnelle complète, en mettant l'accent sur l'automatisation et la performance.

# Lancement du projet

# Méthodologie

## Étape 1 : Récupération des données en local (TP1) :

La première étape du projet consiste à récupérer les données des courses de taxis de New York à partir du site officiel de l'État de New York, puis les stocker localement dans le répertoire data/raw. Toutes les fonctionnalités nécessaires à cette tâche ont été développées dans un unique fichier : **src/data/grab_parquet.py**.

**grab_data** :
Cette fonction récupère les données correspondant aux deux premiers mois (janvier et février ) de l'année 2024.

- Une itération a été effectuée sur chaque mois de l'année pour générer dynamiquement les URL des fichiers d'enregistrements des taxis de NYC.
- La bibliothèque **urllib** a été utilisée pour télécharger les fichiers depuis les URLs générées.
- Avant chaque téléchargement, la fonction vérifie si le fichier existe déjà localement afin d'éviter les téléchargements redondants.

Le fichier src/data/grab_parquet.py centralise ainsi l'ensemble du processus de récupération des données, garantissant une organisation claire et cohérente du code. Le stockage des données dans le répertoire data/raw local permet de simplifier la gestion initiale des fichiers et d'assurer un contrôle total avant leur intégration dans le Data Lake basé sur MinIO.

La deuxième étape consiste à envoyer tous les fichiers Parquet du dossier 'raw' vers MinIO. (Vers le data lake) en utilisant la fonction **write_data_minio()** qui automatise le processus de transfert des fichiers Parquet locaux vers un bucket MinIO, tout en vérifiant :

- Que la connexion est bien établie entre le client et le serveur minio.
- Que le bucket existe (ou en le créant si nécessaire).
- Que le dossier local contient bien des fichiers Parquet à uploader.
- Les éventuelles erreurs pour les gérer et afficher des messages appropriés.

## Étape 2 : Stockage des données locales en base de données (TP2) :

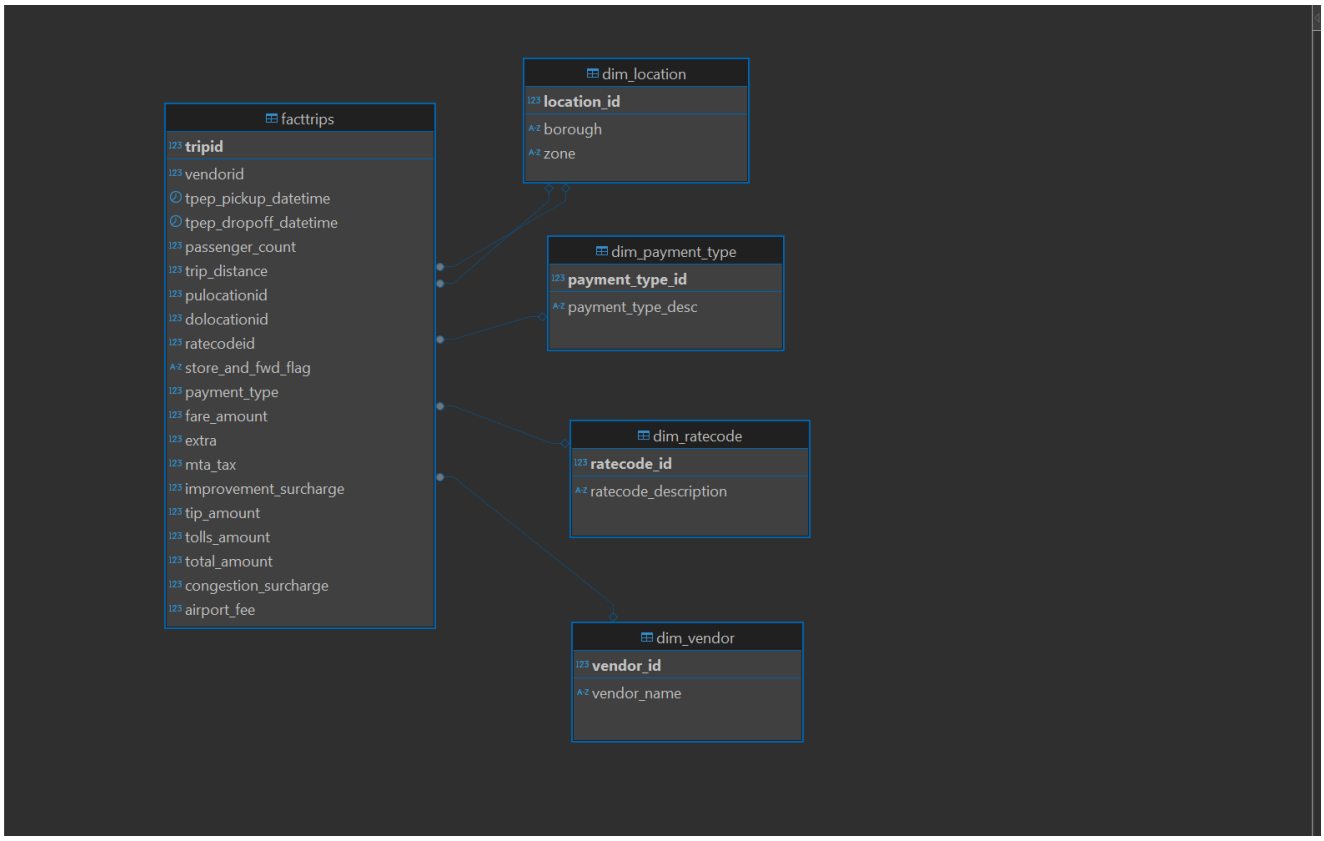
La seconde étape consistait à stocker les données récupérées localement lors de l'étape précédente dans une base de données **"nyc_warehouse"** (créer avec la commande **CREATE DATABASE nyc_warehouse**). Donc en lançant la script **dump_to_sql.py** qui est un pipeline complet permettant de traiter des fichiers **Parquet** depuis un serveur **MinIO** et de les enregistrer dans une base de données **PostgreSQL**. Il commence par la création d'un client MinIO pour interagir avec un bucket spécifique nommé **"yellowtaxibucket"**. Ensuite, la fonction principale, **process_parquet_files**, orchestre les étapes suivantes : (1) lister les fichiers Parquet présents dans le bucket, (2) récupérer chaque fichier et le charger dans un **DataFrame Pandas**, (3) nettoyer les noms des colonnes pour les standardiser (convertir en minuscules et remplacer les espaces par des underscores) et (4) insérer les données nettoyées dans une table PostgreSQL nommée **"nyc_raw"**. L'ensemble du processus est sécurisé par une gestion des erreurs à chaque étape, avec des vérifications pour détecter et traiter tout problème, comme une erreur de connexion ou de données. Enfin, des mécanismes de libération de la mémoire sont intégrés pour optimiser les ressources lors du traitement de gros volumes de données.

## Étape 3 : Création des tables en modèle en Flocon (TP3) :

Dans cette étape, j'ai mis en place un modèle en Flocon en utilisant des requêtes SQL dans un SGBD PostgreSQL. Voici les principales étapes réalisées :

1. **Analyse des données** :
   J'ai analysé les données disponibles afin d'identifier les **faits** principaux et les **dimensions** pertinentes nécessaires à la conception du modèle en Flocon.
2. **Conception du modèle en Flocon** :
   Sur la base des résultats de l'analyse, j'ai conçu un modèle en Flocon composé d'une table de faits centralisant les données principales et de plusieurs tables de dimensions décrivant les attributs connexes.
3. **Écriture des requêtes SQL** :
   Les requêtes nécessaires à la création des tables du modèle en Flocon ont été rédigées dans le script **src/data/creation.sql**.
4. **Test et validation** :
   Après avoir exécuté les scripts SQL pour créer les tables, j'ai validé leur structure en vérifiant que chaque table avait été correctement créée et que les relations entre la table de faits et les tables de dimensions respectaient le modèle en Flocon.

Cette approche m'a permis de structurer les données de manière optimale pour les analyses décisionnelles tout en garantissant la performance des requêtes.




## Étape 4 : Insertion des données dans le Data Mart (TP3) :

Cette étape a consisté à transférer les données depuis la base de données **Data Warehouse** vers le **Data Mart**. Voici les différentes étapes réalisées :

1. **Analyse des données** :
   J'ai analysé les données disponibles dans la base de données Data Warehouse afin d'identifier les informations pertinentes à insérer dans les tables du Data Mart.
2. **Écriture des requêtes SQL** :
   J'ai rédigé les requêtes nécessaires pour extraire les données du Data Warehouse et les insérer dans les tables correspondantes du Data Mart. Ces requêtes, regroupées dans le script **insertion.sql**, utilisaient principalement des instructions **INSERT INTO SELECT** pour effectuer la copie des données entre les deux bases.
3. **Exécution des requêtes** :
   J'ai exécuté les requêtes SQL pour insérer les données dans le Data Mart tout en surveillant le bon déroulement de l'opération. Cela m'a permis de m'assurer que l'ensemble des données a été correctement transféré.

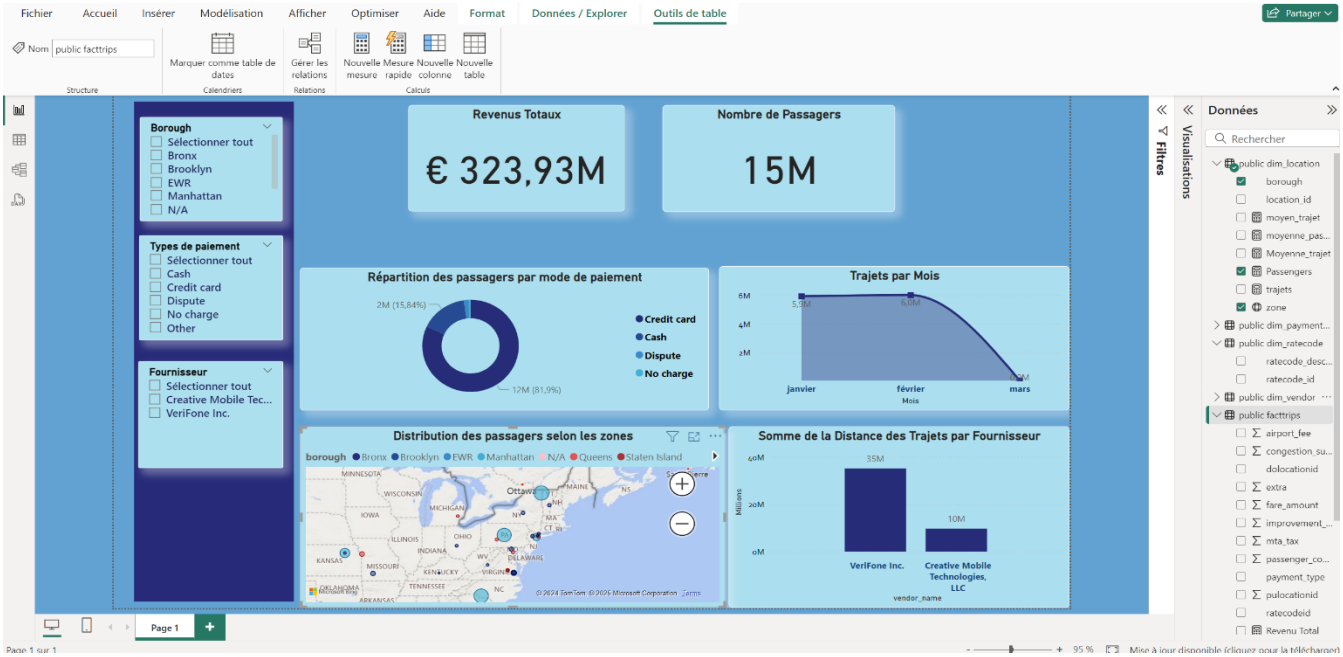
4. **Test et validation** :
   Après l'insertion, j'ai effectué une vérification approfondie pour comparer les données présentes dans le Data Mart avec celles du Data Warehouse. Cette étape a permis de confirmer l'exactitude et l'intégrité des données transférées.

Ce processus a permis de structurer le Data Mart avec des données prêtes à être utilisées pour des analyses et visualisations, tout en garantissant une cohérence totale avec la base source.

## Étape 5: Visualisation de données (TP4) :

Cette étape a consisté à explorer, transformer et visualiser les données afin de mieux comprendre leur structure, de préparer les informations pour l'analyse et d'obtenir des insights clairs. Voici les étapes détaillées :

1. Exploration des Données :
   - J'ai examiné les données brutes pour identifier les variables clés et les relations entre elles.
   - J'ai détecté les valeurs aberrantes, les anomalies et les données manquantes.
2. Transformation des Données :
   - J'ai nettoyé les données, imputé les valeurs manquantes et normalisé certaines variables.
   - J'ai créé de nouvelles variables et effectué des agrégations pour adapter les données à l'analyse.
3. Visualisation des Données :
   - J'ai créé des graphiques (histogrammes, diagrammes circulaires, etc.) pour explorer la distribution des données.
   - J'ai utilisé des visualisations linéaires et géographiques pour analyser les tendances et les répartitions.

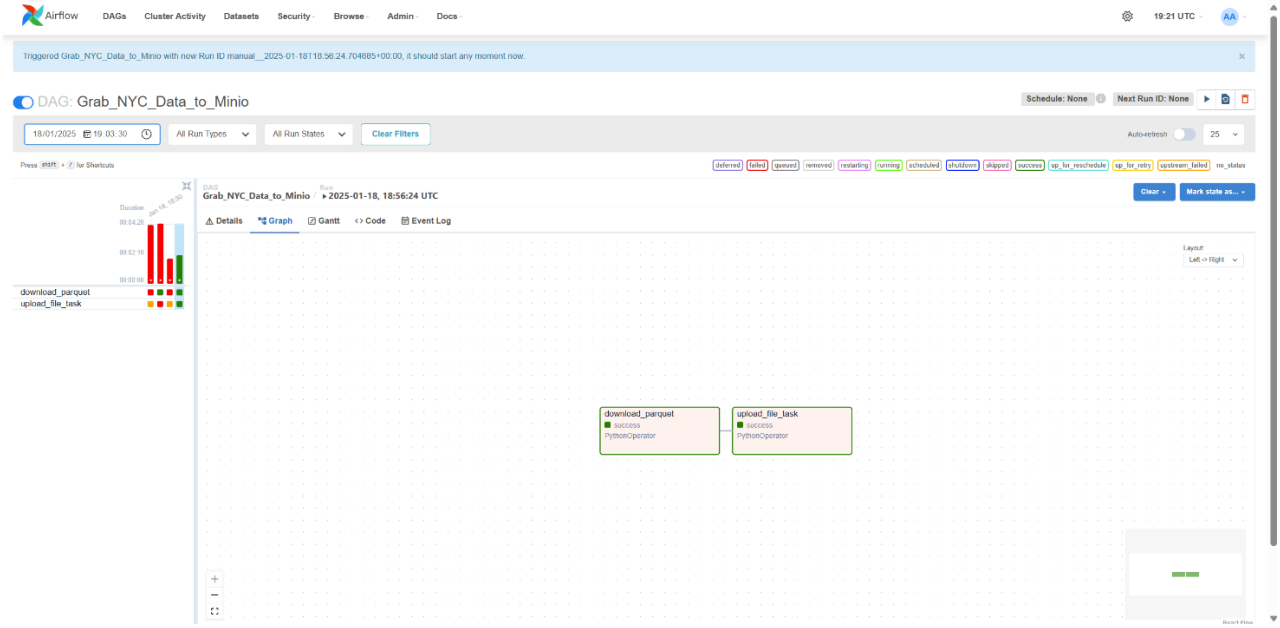



Ce processus m'a permis de mieux comprendre la structure des données et de préparer des visualisations claires pour l'analyse.

## Étape 6: Automatisation des tâches avec Apache Airflow (TP5) :

Cette étape a consisté à automatiser le processus de récupération du dernier mois de données en utilisant Apach Airflow . Voici les étapes que j'ai suivies :

- Création d'une DAG: J'ai créé une DAG **('Grab_NYC_Data_to_Minio')** dans Apache Airflow pour définir le flux de travail automatisé. La DAG comprend des tâches pour télécharger le dernier mois (par rapport à la date actuelle)de données et les stocker dans Minio.
- Planification des tâches: J'ai planifié l'exécution de la DAG pour qu'elle se déclenche automatiquement tous les premiers du mois, en utilisant des expressions cron (**schedule_interval='0 0 1 * *'**)pour définir l'horaire de déclenchement.
- Configuration des connexions: J'ai configuré les connexions Airflow pour se connecter au système de stockage Minio et accéder aux données.
- Test et validation: J'ai testé la DAG pour m'assurer qu'elle fonctionnait correctement et qu'elle récupérait les données attendues chaque mois.

- 


## Partie 2 : Qualité des Données avec Soda

Cette partie a consisté à  configurer, tester et appliquer des règles de qualité des données à une Base des données , en utilisant l'outil Soda.

**Configuration de la Connexion à la Base de Données** : une connexion à la base de données nyc_warehouse a été configurée à l'aide du fichier configuration.yml. Le fichier de configuration contient les paramètres nécessaires à la connexion à la base de données PostgreSQL.

- Une fois le fichier créé, la commande (soda test-connection -d nyc_warehouse -c configuration.yml)a été exécutée pour tester la validité de la connexion.
- **Résultat** :
  La connexion a été réussie et validée.
- ```
  PS C:\Users\massi\Documents\ATL-Datamart> soda test-connection -d nyc_war
  [20:58:49] Soda Core 3.4.4
  Successfully connected to 'nyc_warehouse'.
  Connection 'nyc_warehouse' is valid.
  PS C:\Users\massi\Documents\ATL-Datamart>
  ```
- Le fichier **check.yml** a été créé pour spécifier les règles de qualité des données à appliquer. Ces règles incluent des vérifications sur la présence de colonnes nécessaires et l'absence des colonnes.
- Les tests ont été effectués en utilisant la commande (soda scan -d nyc_warehouse -c configuration.yml check.yml)
  ```

```
PS C:\Users\massi\Documents\ATL-Datamart> soda scan -d nyc_warehouse -c c
[21:02:19] Soda Core 3.4.4
[21:02:20] Scan summary:
[21:02:20] 1/1 check PASSED:
[21:02:20]     nyc_raw in nyc_warehouse
[21:02:20]       Schema Check [PASSED]
[21:02:20] All is good. No failures. No warnings. No errors.
PS C:\Users\massi\Documents\ATL-Datamart>
```

- Ajout de l'Option **-V** Pour obtenir une sortie plus détaillée des résultats des tests , Cela a permis d'obtenir des informations supplémentaires, telles que les détails des erreurs ou des avertissements, ainsi que les colonnes concernées.

```
PS C:\Users\massi\Documents\ATL-Datamart> soda scan -d nyc_warehouse -c configuration.yml check.yml -V
[21:02:41] Soda Core 3.4.4
[21:02:41] Reading configuration file "configuration.yml"
[21:02:42] Reading SodaCL file "check.yml"
[21:02:42] Scan execution starts
[21:02:42] Postgres connection properties: host="localhost", port="15432", database="nyc_warehouse", user=
n_timeout="None"
[21:02:42] Query 1.nyc_warehouse.nyc_raw.schema[nyc_raw]:
SELECT column_name, data_type, is_nullable
FROM information_schema.columns
WHERE lower(table_name) = 'nyc_raw'
  AND lower(table_catalog) = 'nyc_warehouse'
  AND lower(table_schema) = 'public'
ORDER BY ORDINAL_POSITION
[21:02:42] Scan summary:
[21:02:42] 1/1 query OK
[21:02:42]   1.nyc_warehouse.nyc_raw.schema[nyc_raw] [OK] 0:00:00.007982
[21:02:42] 1/1 check PASSED:
[21:02:42]     nyc_raw in nyc_warehouse
[21:02:42]       Schema Check [check.yml] [PASSED]
[21:02:42]         schema_measured = [vendorid integer, tpep_pickup_datetime timestamp without time zone,
senger_count double precision, trip_distance double precision, ratecodeid double precision, store_and_fwd_
 payment_type bigint, fare_amount double precision, extra double precision, mta_tax double precision, tip_
, improvement_surcharge double precision, total_amount double precision, congestion_surcharge double preci
[21:02:42] All is good. No failures. No warnings. No errors.
PS C:\Users\massi\Documents\ATL-Datamart>
```

- **Tester le nombre de lignes :** La règle a été définie de manière à émettre un avertissement si le nombre de lignes est supérieur à 90 et une erreur si le nombre de lignes est égal à 0.

```
PS C:\Users\massi\Documents\ATL-Datamart> soda scan -d nyc_warehouse -c
[21:04:17] Soda Core 3.4.4
[21:04:17] Scan summary:
[21:04:17] 1/1 check WARNED:
[21:04:17]     nyc_raw in nyc_warehouse
[21:04:17]       row_count warn when > 90 fail when = 0 [WARNED]
[21:04:17]         check_value: 5972150
[21:04:17] Only 1 warning. 0 failure. 0 errors. 0 pass.
PS C:\Users\massi\Documents\ATL-Datamart>
```

- Ce processus d'application des règles de qualité des données à l'aide de Soda a permis de garantir la validité et la qualité des données présentes dans la base des données **nyc_warehouse.**

**Exercice 2 :**

Dans cette partie nous allons définir des règles de qualité des données pour garantir que les données sont conformes aux spécifications de la Yellow Trips Data Dictionary et aux règles générales de qualité des données applicables à un modèle en flocons.

- **<u>Compréhension des données :</u>**
- L'utilisation des documents de référence (Yellow Trips Data Dictionary et Taxi Zone LookUp Table) permet de s'assurer que les données respectent des normes spécifiques et que toutes les valeurs sont correctes.
- **Configuration de la Connexion à la Base de Données** : une connexion à la base de données nyc_datamart a été configurée à l'aide du fichier configuration.yml. Le fichier de configuration contient les paramètres nécessaires à la connexion à la base de données PostgreSQL.
- **<u>Règles de Qualité des Données</u>**

- **Table facttrips :**
  - Assurer l'unicité de l'ID du trajet (tripid).
  - Vérifier que les identifiants de fournisseur (vendorid), zones (pulocationid, dolocationid), et codes tarifaires (ratecodeid) sont valides.
  - Les dates de prise en charge et de dépose doivent être cohérentes (la date de dépose ne peut pas être avant la date de prise en charge).
  - Les montants doivent être positifs et correspondre à la somme des montants individuels (par exemple, fare_amount, tip_amount, etc.).
- **Table dim_location :**

    - Garantir l'unicité de location_id et la validité des zones géographiques (borough, zone), par exemple Manhattan, Brooklyn, etc.
- **Table dim_payment_type :**
    - Vérifier que l'ID (payment_type_id) et la description du type de paiement (payment_type_desc) sont valides.
- **Table dim_ratecode :**
    - Vérifier que l'ID du code tarifaire (ratecode_id) et la description du tarif (ratecode_description) sont valides.
- **Table dim_vendor :**
    - Vérifier l'unicité de l'ID du fournisseur (vendor_id) et la validité de son nom (vendor_name).