Rapport : Atelier Qualité des données

Participants : Axel Depoitre, Paul-Louis Mignotte, Sadish Senthilkumar, Ethan Bensaid

Encadré par : Rakib SHEIKH
Durée : 14 heures
Cours dispensé pour : I1 EISI, I1 ECDPIA, I1 ESI Cyber
Date : 13/01/2025

1. Introduction

Ce projet vise à établir un pipeline de contrôle de la qualité des données en utilisant les outils suivants :

- **Soda Core** pour les checks automatiques.
- **PostgreSQL** comme base de données.
- **Streamlit** pour la visualisation des résultats dans un tableau de bord interactif.

Nous avons suivi les étapes nécessaires pour configurer l'environnement, exécuter les checks et afficher les résultats.

---

2. Configuration de l'environnement

2.1. Prérequis

- **Python 3.8+** installé.
- Gestionnaire de paquets **pip**.
- Une instance de **PostgreSQL** fonctionnelle.

2.2. Création de l'environnement virtuel

Pour isoler les dépendances :

```
bash
python3 -m venv env
source env/bin/activate
```

2.3. Installation des dépendances

Voici les bibliothèques Python nécessaires :

- **Soda Core** pour l'exécution des checks :

```
bash
pip install soda-core-postgres
```

- **psycopg2** pour connecter Python à PostgreSQL :

```
bash
pip install psycopg2
```

- **pandas** pour manipuler les données :

```
bash
pip install pandas
```

- **Streamlit** pour le tableau de bord :

```
bash
pip install streamlit
```

- **SQLAlchemy** pour simplifier la connexion avec la base de données :

```
bash
pip install sqlalchemy
```

---

3. Configuration de Soda Core

3.1. Création des fichiers nécessaires

Fichier `configuration.yml`

Ce fichier configure l'accès à la base de données :

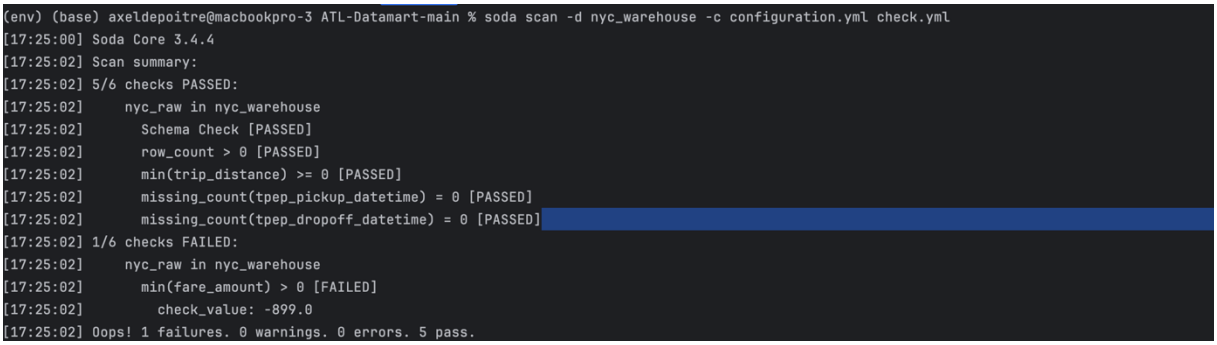
```
yaml

data_source nyc_warehouse:
  type: postgres
  host: localhost
  port: 15432
  username: postgres
  password: admin
  database: nyc_warehouse
```

3.2. Fichier check.yml : Définition des règles de qualité

Le fichier `check.yml` contient les règles de qualité qui ont été appliquées aux données dans la table `nyc_raw`. Ces règles ont été élaborées en se basant sur les spécifications de la source de données (Yellow Taxi Trip Data Dictionary). Voici les détails de ce fichier :

```
yaml
checks for nyc_raw:
  # Vérification du schéma attendu
  - schema:
      warn:
        when required column missing: [
          Airport_fee, tpep_pickup_datetime, tpep_dropoff_datetime,
          passenger_count, trip_distance, RatecodeID, VendorID,
          PULocationID, DOLocationID, payment_type, fare_amount,
          extra, mta_tax, tip_amount, tolls_amount,
          improvement_surcharge, total_amount, congestion_surcharge,
          store_and_fwd_flag
        ]

  # Vérification que la table contient des lignes
  - row_count > 0

  # Vérification des valeurs minimales dans certaines colonnes
  - min(fare_amount) > -1
  - min(trip_distance) >= 0

  # Vérification qu'aucune valeur nulle n'est présente
```

```
  - missing_count(tpep_pickup_datetime) = 0
  - missing_count(tpep_dropoff_datetime) = 0
```



3.3. Résultats initiaux et corrections des données

Lors de l'exécution des checks à l'aide de la commande :

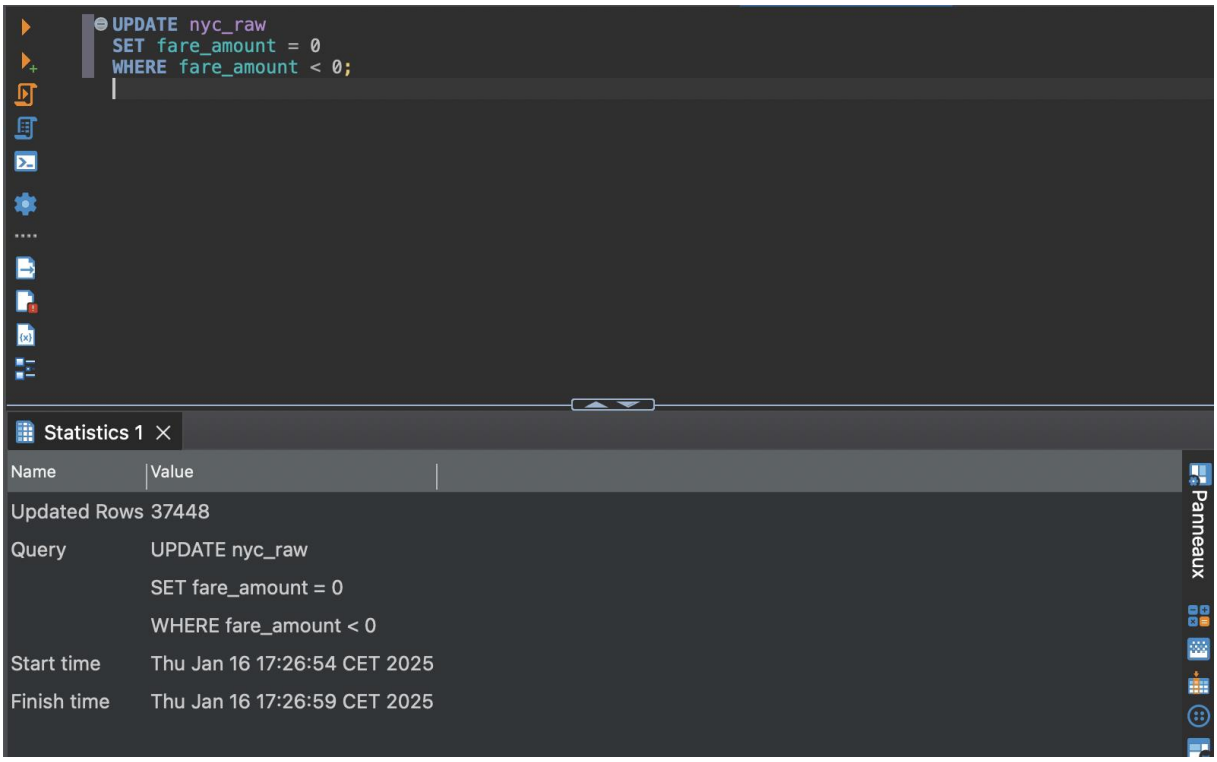
```
bash
soda scan -d nyc_warehouse -c configuration.yml check.yml
```

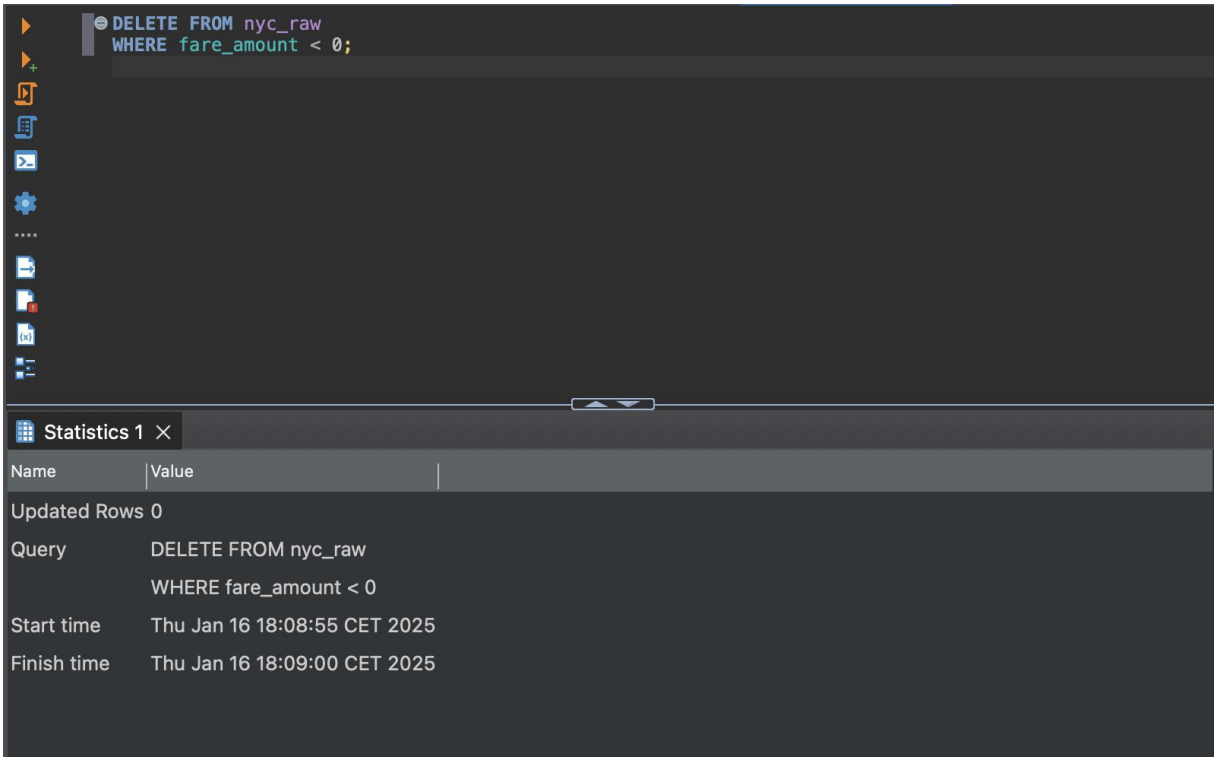
Le rapport a montré que certains tests échouaient. Par exemple :

- La règle `min(fare_amount) > -1` a échoué car certaines lignes contenaient une valeur négative pour `fare_amount`. Cela indiquait des anomalies dans les données d'entrée.

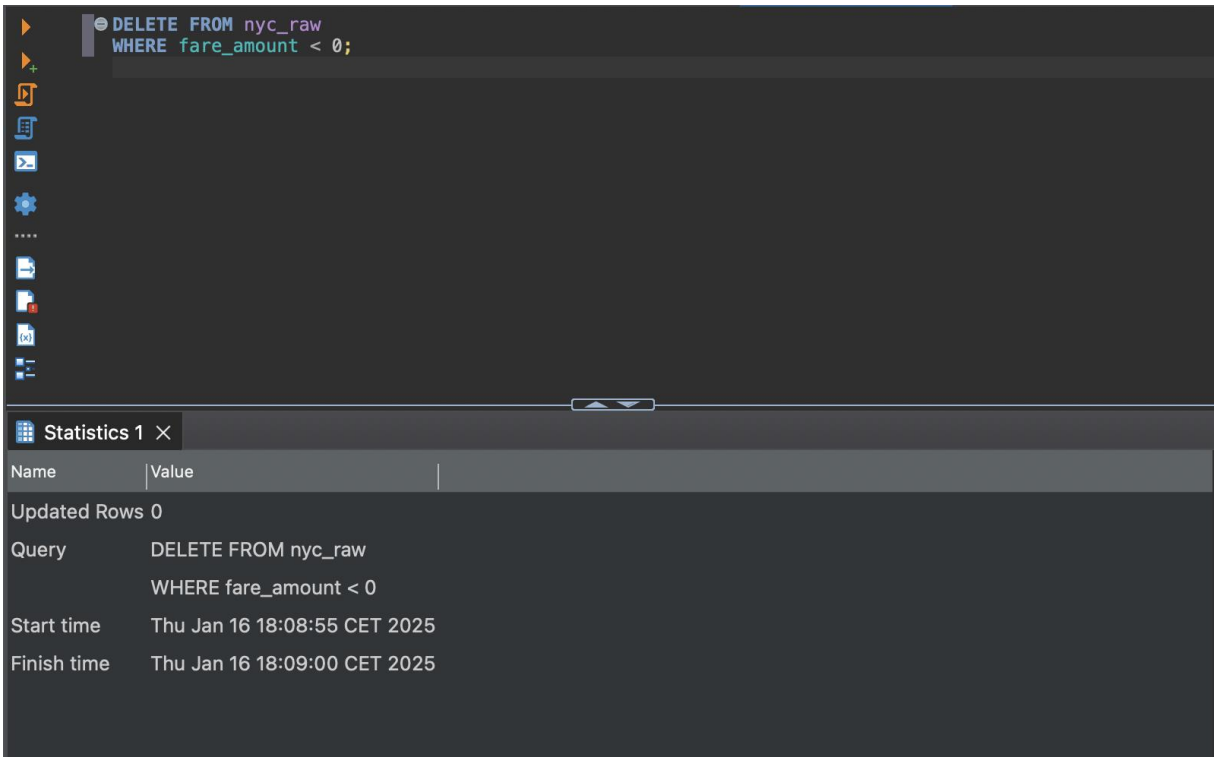
Étapes pour corriger les anomalies

Pour résoudre ces problèmes, nous avons appliqué un filtrage des données incohérentes directement dans la base de données :





Après nettoyage, nous avons relancé les checks, et tous les tests sont passés avec succès. Ce nettoyage garantit que les données analysées respectent les standards de qualité.



---

4. Vérification des données avec Soda Core

4.1. Lancement des checks

Après nous avons créé un nouveau fichier check2.yml :

```
checks for nyc_raw:

  # Vérification que le schéma contient les colonnes attendues

  - schema:

      warn:

        when required column missing: [

          "Airport_fee", tpep_pickup_datetime, tpep_dropoff_datetime,

          passenger_count, trip_distance, "RatecodeID", "VendorID",

          "PULocationID", "DOLocationID", payment_type, fare_amount,

          extra, mta_tax, tip_amount, tolls_amount,

          improvement_surcharge, total_amount, congestion_surcharge,

          store_and_fwd_flag

        ]
```

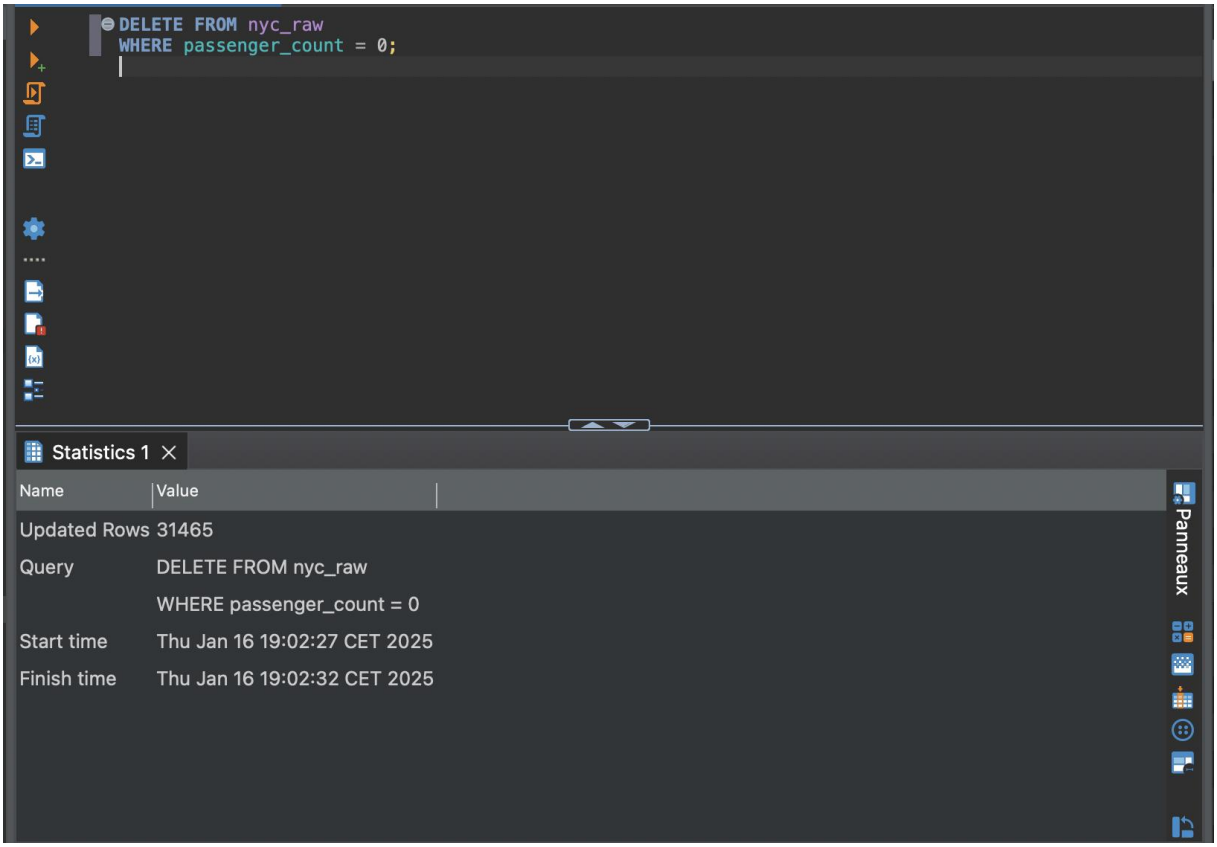
```
  # Vérification du nombre de lignes
  - row_count > 0

  # Vérification des valeurs minimales pour certaines colonnes
  - min(fare_amount) >= 0
  - min(trip_distance) >= 0

  # Vérification des nullités
  - missing_count(tpep_pickup_datetime) = 0
  - missing_count(tpep_dropoff_datetime) = 0

  # Vérification des plages pour passenger_count
  - min(passenger_count) >= 1
  - max(passenger_count) <= 8

  # Vérification des IDs de localisation
  - min("PULocationID") >= 1
  - min("DOLocationID") >= 1
  - missing_count("PULocationID") = 0
  - missing_count("DOLocationID") = 0
```

Commande pour exécuter Soda Core :

```
Base
soda scan -d nyc_warehouse -c configuration.yml check2.yml
```

```
[19:00:34] Scan summary:
[19:00:34] 11/12 checks PASSED:
[19:00:34]     nyc_raw in nyc_warehouse
[19:00:34]       Schema Check [PASSED]
[19:00:34]       row_count > 0 [PASSED]
[19:00:34]       min(fare_amount) >= 0 [PASSED]
[19:00:34]       min(trip_distance) >= 0 [PASSED]
[19:00:34]       missing_count(tpep_pickup_datetime) = 0 [PASSED]
[19:00:34]       missing_count(tpep_dropoff_datetime) = 0 [PASSED]
[19:00:34]       max(passenger_count) <= 9 [PASSED]
[19:00:34]       min("PULocationID") >= 1 [PASSED]
[19:00:34]       missing_count("PULocationID") = 0 [PASSED]
[19:00:34]       min("DOLocationID") >= 1 [PASSED]
[19:00:34]       missing_count("DOLocationID") = 0 [PASSED]
[19:00:34] 1/12 checks FAILED:
[19:00:34]     nyc_raw in nyc_warehouse
[19:00:34]       min(passenger_count) >= 1 [FAILED]
[19:00:34]         check_value: 0
[19:00:34] Oops! 1 failures. 0 warnings. 0 errors. 11 pass.
```

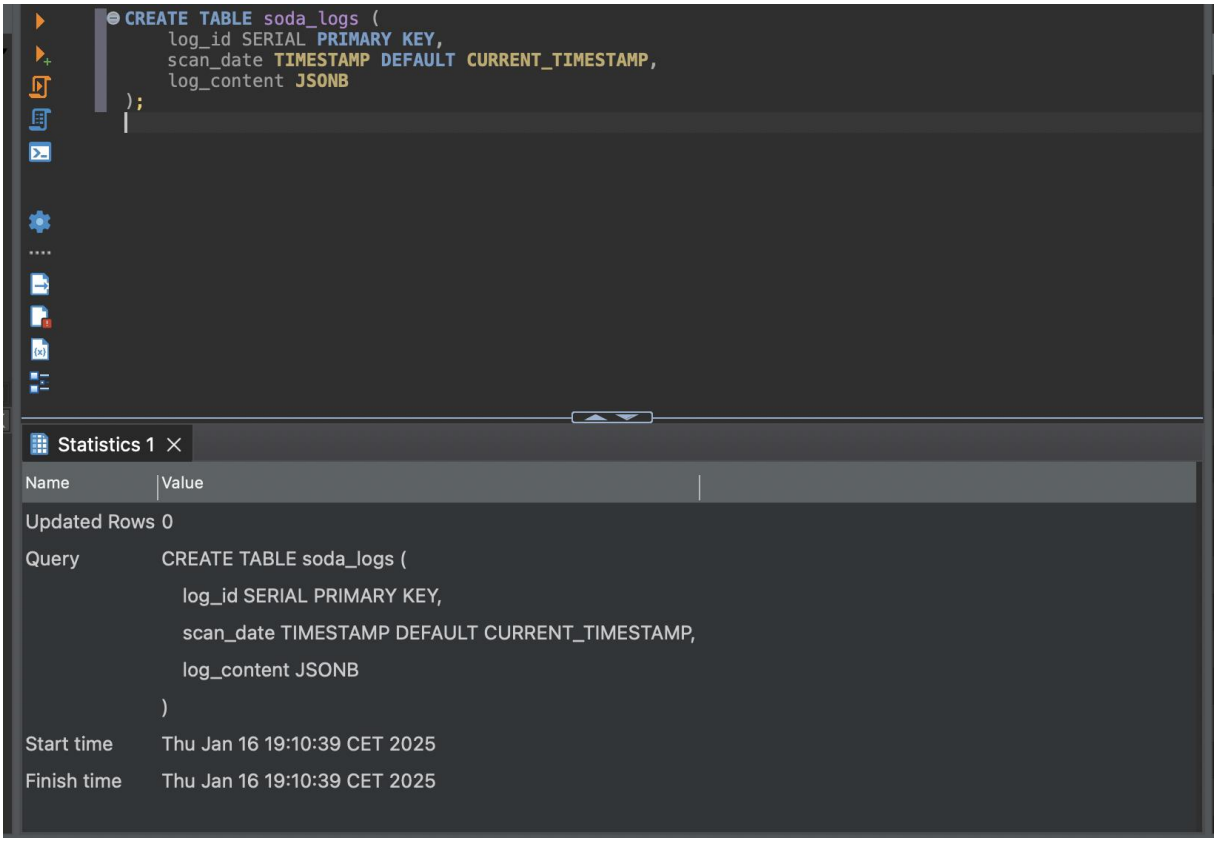
Nous avons toujours u problème de données dans passenger_count , en effet un taxi ne peux pas avoir 0 personnes, voici comment nous avons réglé le problème :



```
[19:03:49] Soda Core 3.4.4
[19:03:53] Scan summary:
[19:03:53] 12/12 checks PASSED:
[19:03:53]     nyc_raw in nyc_warehouse
[19:03:53]       Schema Check [PASSED]
[19:03:53]       row_count > 0 [PASSED]
[19:03:53]       min(fare_amount) >= 0 [PASSED]
[19:03:53]       min(trip_distance) >= 0 [PASSED]
[19:03:53]       missing_count(tpep_pickup_datetime) = 0 [PASSED]
[19:03:53]       missing_count(tpep_dropoff_datetime) = 0 [PASSED]
[19:03:53]       min(passenger_count) >= 1 [PASSED]
[19:03:53]       max(passenger_count) <= 9 [PASSED]
[19:03:53]       min("PULocationID") >= 1 [PASSED]
[19:03:53]       missing_count("PULocationID") = 0 [PASSED]
[19:03:53]       min("DOLocationID") >= 1 [PASSED]
[19:03:53]       missing_count("DOLocationID") = 0 [PASSED]
[19:03:53] All is good. No failures. No warnings. No errors.
(env) (base) axeldepoitre@macbookpro-3 ATL-Datamart-main %
```

4.2. Export des logs

Pour sauvegarder les logs des checks dans une base PostgreSQL, un script Python a été créé.



---

5. Chargement et stockage des logs

Un script a été utilisé pour insérer les logs Soda Core dans la table `soda_logs` de la base `nyc_warehouse`. Voici un exemple de code utilisé :

Code d'upload des logs :

```
python

import psycopg2
from datetime import datetime

# Connexion à la base de données
conn = psycopg2.connect(
    host="localhost",
    port=15432,
    database="nyc_warehouse",
    user="postgres",
    password="admin"
)

# Insertion des logs dans la base
def insert_logs(scan_date, log_content):
    cursor = conn.cursor()
    query = "INSERT INTO soda_logs (scan_date, log_content) VALUES (%s,
%s);"
    cursor.execute(query, (scan_date, log_content))
    conn.commit()
    cursor.close()

# Exemple d'usage
scan_date = datetime.now()
log_content = '{"checks": [{"name": "row_count > 0", "outcome": "pass"}]}'
# Exemple JSON
insert_logs(scan_date, log_content)
conn.close()
```

---

6. Tableau de bord avec Streamlit

6.1. Objectif

Afficher les résultats des logs dans une interface utilisateur interactive.

6.2. Code du tableau de bord

Nous créons un nouveau fichier dashboard.py qui contient notre Dashboard

```
python

import streamlit as st
import psycopg2
import pandas as pd
import json
from sqlalchemy import create_engine

# Connexion à la base PostgreSQL
def connect_to_db():
    engine =
create_engine("postgresql+psycopg2://postgres:admin@localhost:15432/nyc_war
ehouse")
    return engine

# Charger les logs depuis la base
def fetch_logs():
```

```
    engine = connect_to_db()
    query = "SELECT scan_date, log_content FROM soda_logs ORDER BY
scan_date DESC;"
    logs = pd.read_sql(query, engine)
    return logs

# Extraire les checks et leurs statuts
def extract_checks(log_content):
    try:
        log_data = json.loads(log_content)
        checks = log_data.get("checks", [])
        extracted_data = []
        for check in checks:
            extracted_data.append({
                "Nom du Check": check.get("name", "Inconnu"),
                "Statut": check.get("outcome", "Inconnu"),
                "Table": check.get("table", "Inconnu"),
                "Colonne": check.get("column", "Inconnu")
            })
        return extracted_data
    except Exception as e:
        return [{"Nom du Check": "Erreur", "Statut": str(e), "Table": "-",
"Colonne": "-"}]

# Interface utilisateur
st.title("Dashboard de Qualité de Données")
st.markdown("Suivi des indicateurs de qualité.")

try:
    logs = fetch_logs()
    st.success("Données chargées avec succès !")

    st.subheader("Logs de Qualité")
    st.dataframe(logs)

    st.subheader("Indicateurs de Performance")
    for index, row in logs.iterrows():
        scan_date = row["scan_date"]
        log_content = row["log_content"]

        st.markdown(f"### Résultats du scan du {scan_date}")
        checks_data = extract_checks(log_content)
        checks_df = pd.DataFrame(checks_data)
        st.dataframe(checks_df)
except Exception as e:
    st.error(f"Erreur lors du chargement des données : {e}")
```

6.3. Lancement du tableau de bord

Pour exécuter le tableau de bord :

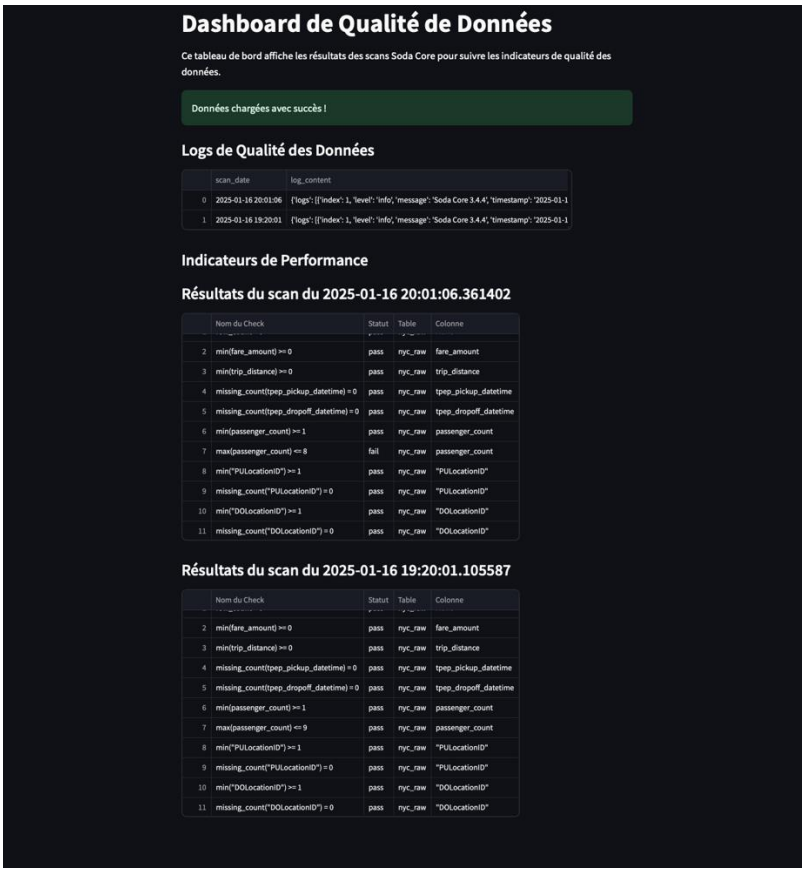
```
bash
streamlit run dashboard.py
```

---

7. Résultats finaux

Le tableau de bord a permis de visualiser les résultats des checks exécutés avec Soda Core. Chaque log inclut :

- La date du scan.
- Les résultats des checks (succès ou échec).
- Les métriques associées.



---

8. Conclusion

Ce projet a démontré comment :

1. Exécuter des checks de qualité des données avec Soda Core.
2. Stocker les résultats dans PostgreSQL.
3. Visualiser les résultats à l'aide de Streamlit.

Pour améliorer le pipeline, on pourrait :

- Automatiser l'exécution des checks à l'aide d'un scheduler (par ex., `cron`).
- Ajouter des alertes pour les checks échoués.