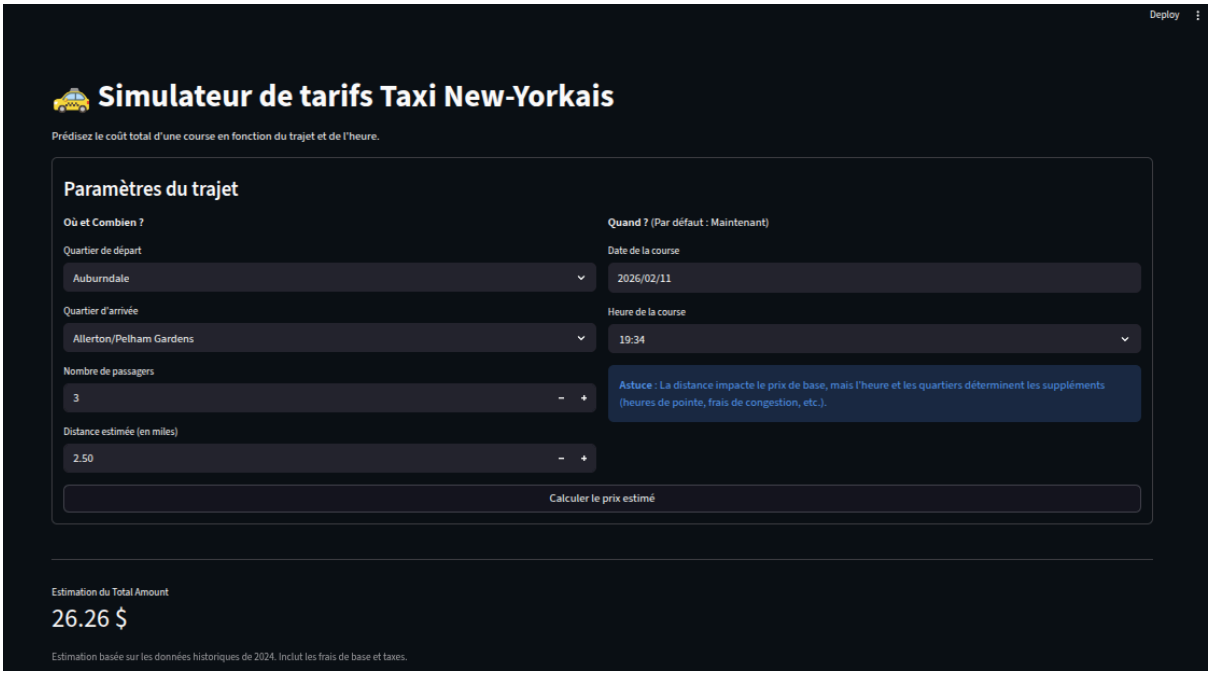
BIG DATA



14/02/2026

Amine AIT MOUSSA
Siham DAANOUNI
Haoxin LIU
Wangjiachen ZHAO
ING3 IAC

Professeur : Rakib SHEIKH

SOMMAIRE

1. Introduction

Notre projet du module Big Data s'inscrit dans la mise en œuvre d'une architecture complète, visant à traiter et analyser les données officielles de transport de la ville de New York. L'objectif est de construire un pipeline industriel capable de transformer des données brutes massives en informations exploitables via un dashboard décisionnel et un service de prédiction.

L'architecture repose sur une segmentation en trois couches :

Data Lake (Couche Bronze/Silver) : Utilisation de MinIO pour le stockage objet des fichiers Parquet bruts.

ETL & Data Warehouse (Couche Gold) : Emploi d'Apache Spark (Scala) pour le nettoyage et l'ingestion dans PostgreSQL, structuré en schéma en étoile.

Service de Prédiction : Module de Machine Learning en Python pour l'estimation des tarifs, validé par des tests unitaires et la norme PEP 8.

Ce rapport détaille les défis de traitement de volumes importants et les performances de notre modèle (RMSE de 4,25), dépassant les exigences initiales (RMSE < 10).

2. Infrastructure et Flux d'Ingestion

2.1 Environnement Conteneurisé

L'ensemble de l'infrastructure est orchestrée via **Docker Compose** pour garantir la portabilité. MinIO simule un Data Lake industriel pour le stockage des fichiers Parquet. PostgreSQL pour un serveur de base de données relationnelle faisant office de Data Warehouse.

2.2 Collecte et Ingestion Multi-Branche (Exercices 1 et 2)

Le processus a été automatisé en Scala pour assurer un flux direct depuis la source vers le Cloud local.

Nous commençons avec une acquisition des données au format Parquet depuis le site de l'État de New York directement vers le bucket nyc-raw de MinIO. Nous enchaînons ensuite avec un traitement Spark (Data Ingestion)**.** Un job Spark qui traite les données brutes avec une logique de "branching":

Branche 1 : Sauvegarde des données nettoyées en format Parquet dans MinIO pour l'entraînement ML.

Branche 2 : Ingestion vers le Data Warehouse PostgreSQL après transformation en mémoire.

```
aminedesel@aminedesel-B550-AORUS-ELITE-AX-V2:~/Desktop/BigData_proje
t/ex02_data_ingestion$ docker exec -it postgres-db psql -U myuser -d
 taxidb -c "SELECT count(*) FROM dwh.fact_trip;"
  count
---------
 8479488
(1 row)
```

2.3 Gestion des volumes massifs

Le défi majeur a été l'ingestion du dataset 2024. L'utilisation de Spark a permis de partitionner les données, évitant les erreurs de mémoire (OOM) rencontrées avec des librairies classiques.

3. Modélisation du Data Warehouse (Exercice 3)

3.1 Choix du schéma en étoile

Nous avons implémenté un schéma en étoile dans PostgreSQL. Bien que le contrat initial suggérait un modèle en flocon, nous avons privilégié l'étoile pour la performance et la simplicité. Cela implique moins de jointures complexes pour les requêtes analytiques, ainsi qu'une compatibilité directe et performante avec les outils de Data Visualization.

3.2 Architecture et Intégrité

Le Datamart s'articule autour d'une table de faits centrale (fact_trip) et de tables de dimensions (dim_date, dim_location, dim_rate_code, dim_payment_type). L'intégrité est garantie par des scripts SQL de création de contraintes et un mécanisme de "lookup" lors de l'ingestion Spark.

```sql
CREATE SCHEMA IF NOT EXISTS dwh;

-- Dimensions
CREATE TABLE IF NOT EXISTS dwh.dim_vendor (
                                            vendor_key   SERIAL PRIMARY KEY,
                                            vendor_id    INT UNIQUE NOT NULL
);

CREATE TABLE IF NOT EXISTS dwh.dim_rate_code (
                                               rate_code_key SERIAL PRIMARY KEY,
                                               rate_code_id  INT UNIQUE NOT NULL
);

CREATE TABLE IF NOT EXISTS dwh.dim_payment_type (
                                                  payment_type_key SERIAL PRIMARY KEY,
                                                  payment_type_id  INT UNIQUE NOT NULL
);

CREATE TABLE IF NOT EXISTS dwh.dim_location (
                                              location_key SERIAL PRIMARY KEY,
                                              location_id  INT UNIQUE NOT NULL
);

CREATE TABLE IF NOT EXISTS dwh.dim_datetime (
                                              datetime_key SERIAL PRIMARY KEY,
                                              ts           TIMESTAMP UNIQUE NOT NULL,
                                              date         DATE NOT NULL,
                                              year         INT NOT NULL,
                                              month        INT NOT NULL,
                                              day          INT NOT NULL,
                                              hour         INT NOT NULL,
                                              dow          INT NOT NULL
);

-- Fact table: 1 row = 1 trip
CREATE TABLE IF NOT EXISTS dwh.fact_trip (
                                           trip_key SERIAL PRIMARY KEY,

                                           pickup_datetime_key  INT NOT NULL,
                                           dropoff_datetime_key INT NOT NULL,

                                           vendor_key       INT,
```

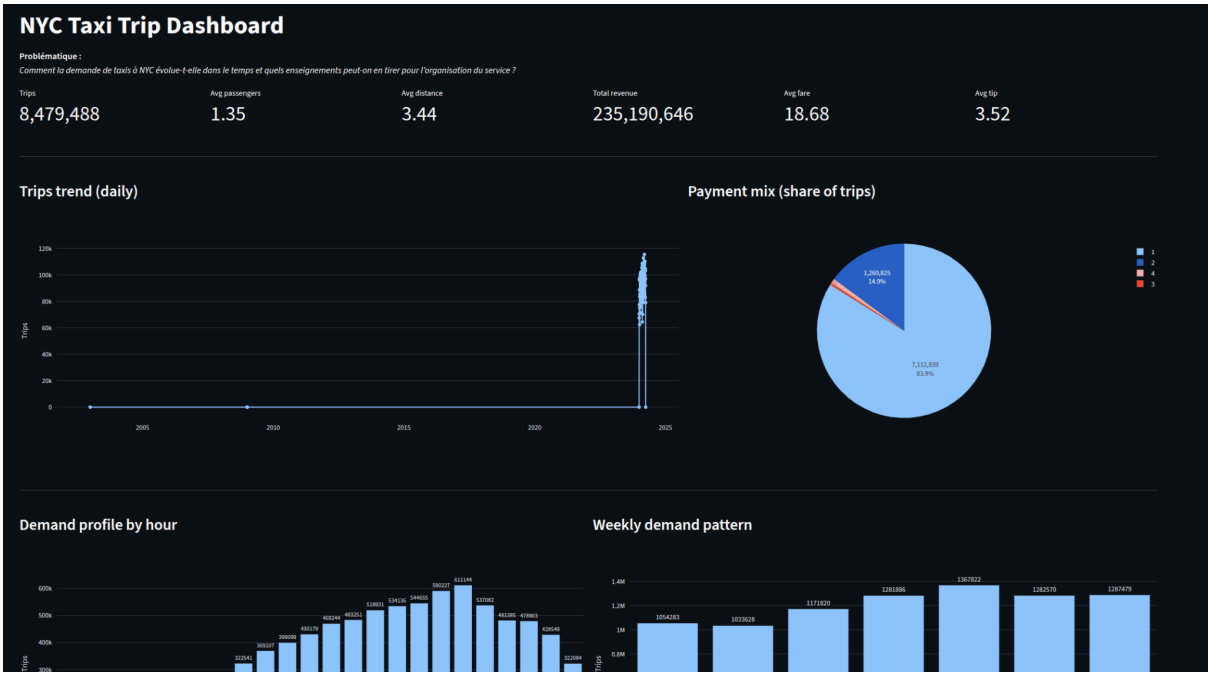
4. Visualisation de Données (Exercice 4)

Une fois les données structurées dans le Data Warehouse, nous avons développé des outils de restitution pour répondre aux besoins métiers.

Analyse Exploratoire (EDA) : Utilisation de dashboard.py pour visualiser les volumes de courses, les revenus totaux et les comportements de paiement des usagers.

Indicateurs clés (KPIs) : Affichage en temps réel de la distance moyenne des trajets et des zones de prise en charge les plus rentables.

Accessibilité : Le choix de Streamlit permet une interaction dynamique avec la base PostgreSQL, facilitant la lecture des données pour des utilisateurs non-techniques



5. Machine Learning Qualité Logicielle (Exercice 5)

5.1 Modélisation et Résultats

Pour prédire le total_amount, nous avons utilisé l'algorithme HistGradientBoostingRegressor. Notre modèle affiche un RMSE de 4,25, soit une précision bien supérieure à l'objectif de 10 fixé par le cahier des charges. Les principaux intérêts de ce modèle sont la gestion native des catégories et la rapidité sur de gros volumes de données.

5.2 Industrialisation et Tests

Le développement respecte les standards de production:

Qualité : Audit via flake8 et documentation au format NumpyDoc.

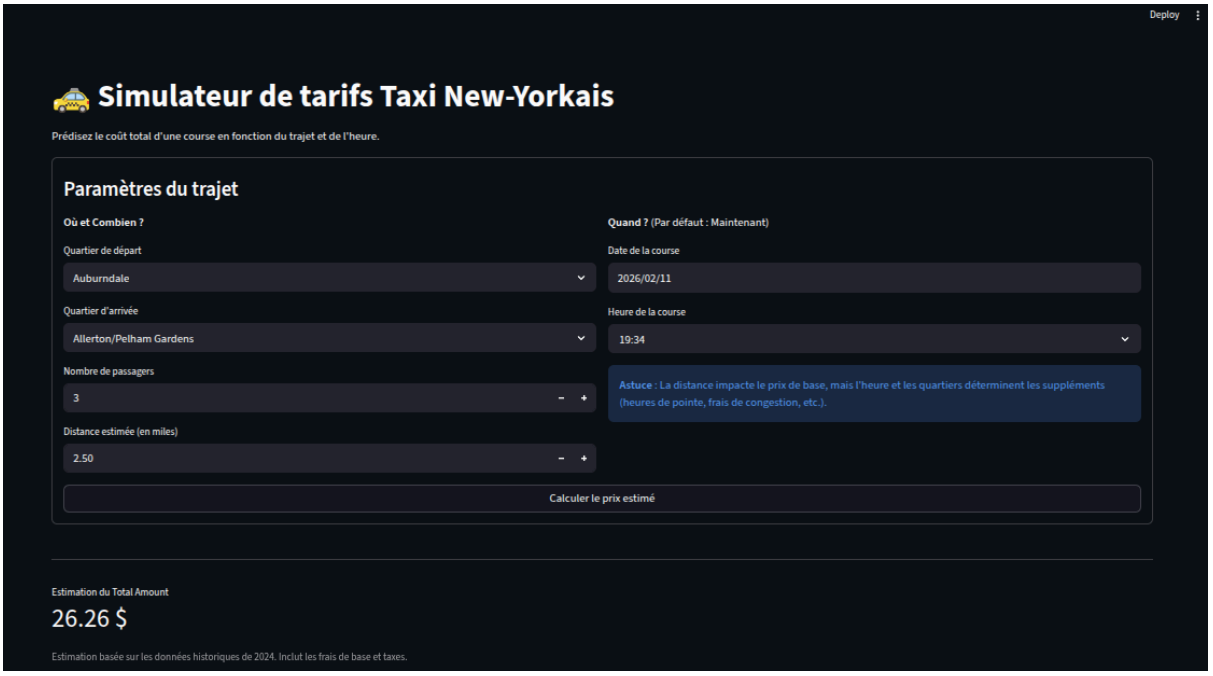
Tests : Suite de tests unitaires validant l'intégrité des données à l'entrée de l'entraînement et lors de l'inférence.

```
ta_projet/ex05_ml_prediction_service$ python -m src.train
[EX5] RMSE: 4.2550
[EX5] Saved model -> artifacts/model.joblib
[EX5] Saved spec  -> artifacts/feature_spec.json
```

```
(.venv) aminedesel@aminedesel-B550-AORUS-ELITE-AX-V2:~/Desktop/BigData_projet/ex05_ml_prediction_service$ flake8 --max-line-length=100 src/
(.venv) aminedesel@aminedesel-B550-AORUS-ELITE-AX-V2:~/Desktop/BigData_projet/ex05_ml_prediction_service$ export MINIO_BUCKET_CLEAN="nyc-cleaned"
export YEAR="2024"
export MONTH="01"
python -m pytest tests/test_pipeline.py
========================= test session starts ==========================
platform linux -- Python 3.11.14, pytest-9.0.2, pluggy-1.6.0
rootdir: /home/aminedesel/Desktop/BigData_projet/ex05_ml_prediction_service
configfile: pytest.ini
collected 3 items

tests/test_pipeline.py ...                                        [100%]

========================== 3 passed in 32.63s ==========================
```



6. Orchestration (Bonus Exercice 6)

6.1 Automatisation via Apache Airflow

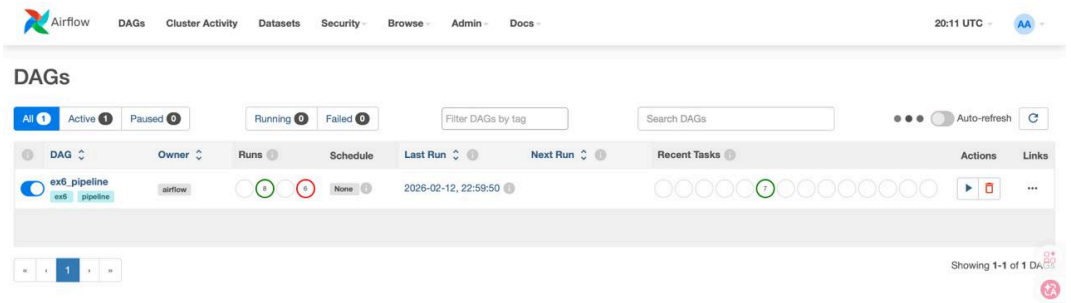
Pour transformer ces scripts en un pipeline industriel, nous avons déployé Apache Airflow. L'ensemble du workflow est géré par un DAG (Directed Acyclic Graph) :

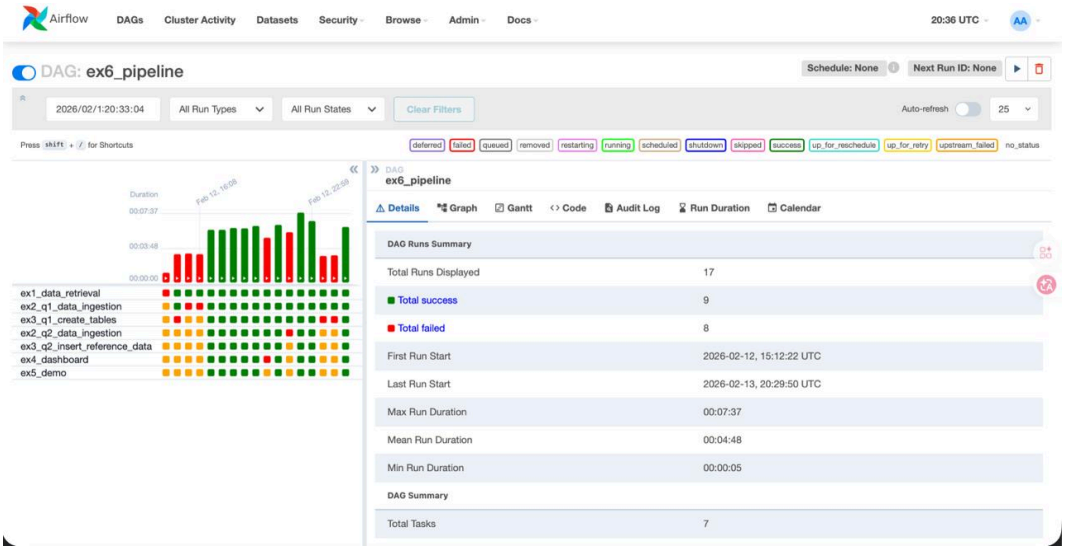
Séquençage : La collecte MinIO déclenche le job Spark, qui lui-même précède l'ingestion SQL et l'entraînement ML.

Robustesse : Airflow gère les reprises sur erreur (retries) en cas de défaillance réseau lors du téléchargement des fichiers Parquet.

6.2 Monitoring "CAC 40"

L'interface d'Airflow permet un suivi en temps réel de l'état de santé du pipeline, offrant une visibilité complète sur les temps d'exécution et les éventuels goulots d'étranglement.





7. Conclusion

Ce projet démontre la robustesse d'une stack Big Data moderne : Spark pour la puissance de calcul, MinIO/PostgreSQL pour le stockage hybride, et Airflow pour l'orchestration globale. Le respect des normes logicielles et l'atteinte d'un RMSE de 4,25 confirment la viabilité de la solution pour un déploiement en production. L'interface Streamlit finale transforme cette infrastructure technique en un véritable outil d'aide à la décision.