Big Data

Déploiement d'une Architecture Big Data : Analyse et Prédiction des flux de Taxis à New York

ING3 IAC Semaine A, CY-Tech
13 février 2026

Auteurs : Andre Victor
Bogaer Youenn
Dano Ewen

Implémentation du projet : https://github.com/Dnewe/projet_big_data_cytech_25.git

1 Introduction

1.1 Contexte

Ce Projet s'inscrit dans le module Big Data, l'objectif principal est de déployer une architecture CAC40 afin de manipuler un flux de données massif.

Pour cela nous allons utiliser le jeu de données ouvert de la ville de New York, il porte sur les registres de trajets des taxis jaunes. Ces données sont stockées sous forme de parquet, nous ferons la collecte au traitement machine learning de ces données

1.2 Objectif

L'objectif est d'appliquer la théorie vu en cours sous la forme de 5 exercices :

— Exercice 1 : Collecte et Stockage – Mise en place d'un Data Lake via le service Minio pour l'ingestion des fichiers bruts.
— Exercice 2 : Ingestion et Transformation – Développement d'un pipeline Spark/Scala pour le nettoyage des données et leur intégration dans un Data Warehouse (PostgreSQL).
— Exercice 3 : Modélisation Multi-dimensionnelle – Structuration des données selon un modèle en étoile ou en flocon pour optimiser les requêtes analytiques.
— Exercice 4 : Data Visualisation – Conception d'un tableau de bord (via Streamlit) pour l'exploration de données.
— Exercice 5 : Machine Learning – Implémentation d'un modèle prédictif du prix des courses et mise en place de tests unitaires.

1.3 Présentation rapport

Dans ce rapport, nous allons traiter chaque exercice indépendamment et justifier nos choix d'implémentation. Enfin, dans certaines parties, nous présenterons nos résultats.

2 Exercice 1 : La collecte des données et l'intégration de données

L'objectif était de récupérer les fichiers Parquet des "Yellow Taxis" pour les stocker dans notre Data Lake représenté par Minio.

2.1 Ajustements techniques

— **Docker** : Nous avons renommé le dossier "Docker" en "docker" (minuscule) pour assurer la compatibilité avec `docker-compose`.
— **Minio** : Le bucket a été nommé `nyc-raw` au lieu de `nyc_raw`, car Minio refuse les underscores dans les noms de buckets.

2.2 Stratégie de téléchargement

Nous avons testé deux approches pour automatiser le flux :

— **Essai 1** : Passage par un fichier temporaire en local avant lecture par Spark, mais la lecture directe de l'URL par Spark a échoué.
— **Essai 2 (Solution finale)** : Utilisation du `FileSystem` de Hadoop pour streamer les données directement de l'URL vers Minio sans stockage local.

Contrainte rencontrée : Nous avons limité le téléchargement à un seul mois car le site nous bloquait après plusieurs requêtes successives (protection Cloudflare). Nous avons donc ajouté des `User-Agent` dans notre code Scala pour simuler une navigation humaine et éviter le bannissement.

3 Exercice 2 : Le nettoyage de données et l'ingestion multi-branche

3.1 Branche 1 : Nettoyage des données

Dans cette partie nous allons voir le nettoyage des données en utilisant Spark :

Nous avons tout d'abord récuperer le parquet enregistré dans notre cluster minio.

Tout d'abord nous avons voulu récupérer plusieurs parquets puis tous les traiter en même temps, sauf que nous avons eu un problème c'est que Spark plantait car certains parquets ne stockait pas les données de manière identique donc c'est à ce moment là que nous avons décidé de traiter seulement un seul parquet pour simplifier notre projet. Donc nous traiterons seulement un mois de données de "Yellow taxis".

Pour le nettoyage nous avons utilisé le document mis à disposition sur le site où l'on télécharge les parquets qui présente les données (ref 1). Notre premier nettoyage portait seulement sur la logique, c'est à dire avoir des valeurs positives et des dates de début et de fin dans le bon ordre (fin > début).

Puis après la visualisation des données, mais aussi lors de la conception de notre modèle, nous avons identifié d'autres problèmes liés au dataset et nous sommes arrivés à ce nettoyage :

- **Cohérence temporelle** : Validation que la date de fin de course (*tpep_dropoff_datetime*) est bien postérieure à la date de début (*tpep_pickup_datetime*).
- **Volume de transport** : Suppression des trajets affichant un nombre de passagers nul ou négatif (*passenger_count* > 0).
- **Validation physique** : Filtrage des courses ayant une distance nulle (*trip_distance* > 0), ces dernières ne représentant pas des trajets réels.
- **Intégrité financière** : Élimination des anomalies de facturation en s'assurant que le tarif de base (*fare_amount*) et le montant total (*total_amount*) sont strictement positifs.
- **Géolocalisation** : Restriction des identifiants de zone (*PULocationID* et *DOLocationID*) aux bornes valides (entre 1 et 265).
- **Conformité au dictionnaire de données** :
  - Vérification des vendeurs (*VendorID* : 1, 2, 6, 7).
  - Validation des codes de tarifs (*RateCodeID* : de 1 à 6, et 99).
  - Validation des modes de paiement (*payment_type* : de 0 à 6).

Résultats :
Avant filtrage : 4 181 444 lignes
Après filtrage : 3 001 197 lignes

Enfin nous avons enregistré nos données valides sous forme d'un nouveau parquet dans un nouveau bucket minio.

3.2 Branche 2

Cette branche a été faite après la création de nos tables SQL.

L'objectif de cette branche était d'insérer nos données dans une base de données, plus précisément dans notre table de fait.

Nous somme partis de nos parquets déjà filtrés lors de la branche 1 et nous avons inséré nos données dans la table de fait puis dans la table de temps (Dim_time). L'ordre d'insertion était important car quand on était en dévellopement, et pour utiliser la commande overwrite lors de l'insertion des données, il fallait drop d'abord la table de fait car elle contenait une clé étrangère de Dim_time Le fait d'avoir réutilisé les mêmes noms dans la table sql et dans le parquet nous a simplifié le travail dans spark.

4 SQL

L'objectif de cette étape est de transformer les données brutes issues du Data Lake en une structure optimisée pour l'analyse. Contrairement au format initial orienté OLTP.

4.1 Choix du Modèle : Le Modèle en Flocon

Pour ce projet, nous avons fait le choix d'implémenter un modèle en flocon . Bien qu'un modèle en étoile aurait pu suffire pour traiter ces données de manière plus simple, nous nous sommes conformés aux exigences du projet qui imposait une structure en flocon.

On a choisi le flocon pour bien séparer les localisations des quartiers, même si l'étoile aurait été plus simple à faire :

— La table `Dim_Location` est physiquement détachée et reliée à une table `Dim_Borough` afin de respecter la hiérarchie demandée.
— Cette structure permet de garantir une meilleure intégrité des données en évitant la redondance des noms de quartiers (Manhattan, Bronx ...).

4.2 Création

Nous avons utilisé le langage SQL du SGBD PostgreSQL pour définir la structure du Data Warehouse. Le script suivant présente la création de la table de faits (`Fact_Trips`) ainsi que ses dimensions associées.

— **Table de Faits** : Elle centralise les mesures quantitatives comme les montants, les distances ou le nombre de passagers, ainsi que les clés étrangères vers les dimensions.
— **Dimensions** : Nous avons extrait les données temporelles (`Dim_Time`), les modes de paiement (`Dim_Payment`), les prestataires (`Vendor`), et la géographie (`Dim_Location` & `Dim_Borough`).

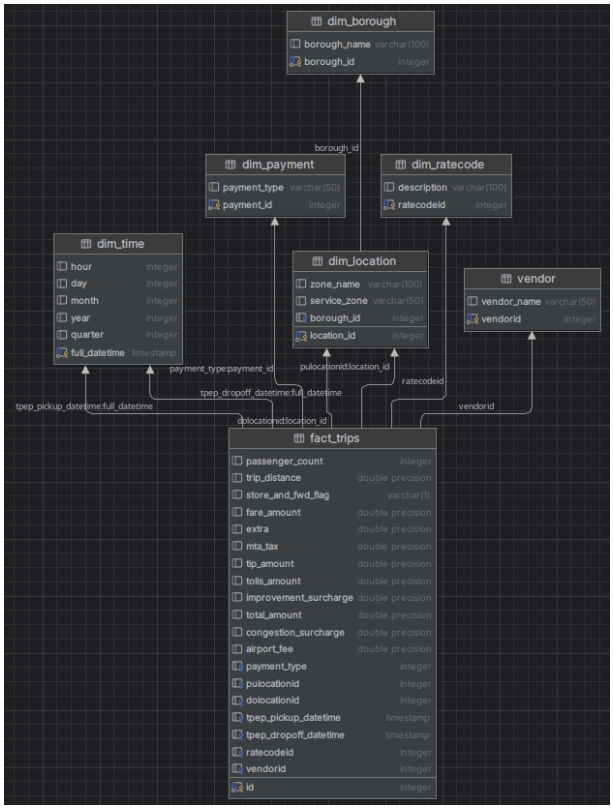

FIGURE 1 – Diagramme SQL

4.3 Insertion

L'alimentation du Data Warehouse s'est faite en deux temps : l'insertion des données de référence et l'ingestion des données traitées par Spark.

Les données de référence insérées concernent :

— Les types de paiement (ex : Credit card, Cash).
— Les codes tarifaires (ex : Standard rate, JFK).
— Les prestataires de services (Vendor).
— Les 264 zones présentes dans les parquets, pour les insérer nous avons utiliser le fichier csv présent sur le site `NYC.gouv` (`Taxi Zone Lookup Table`)

5 Exercice 4 : Visualisation de données

L'objectif de cette étape est de transformer les données nettoyées et structurées en informations décisionnelles via un tableau de bord interactif.

5.1 Méthodologie

Avant le développement, une phase d'Analyse Exploratoire des Données a été réalisée sous forme de Notebook. Cette étape a permis de confirmer la structure des données.

Pour le dashboard, nous avons utilisé la bibliothèque Streamlit, connectée directement à notre base de données PostgreSQL. L'interface est structurée en quatre sections principales via une barre de navigation latérale :

— **Vue d'ensemble** : Affichage de la data importante tel que le nombre total de courses, le prix moyen et la distance moyenne.
— **Analyse Temporelle** : Visualisation de l'affluence et de la rentabilité horaire.
— **Analyse Géographique** : Cartographie interactive des flux.
— **Dimensions Métiers** : Analyse par type de tarif, vendeur et mode de paiement.

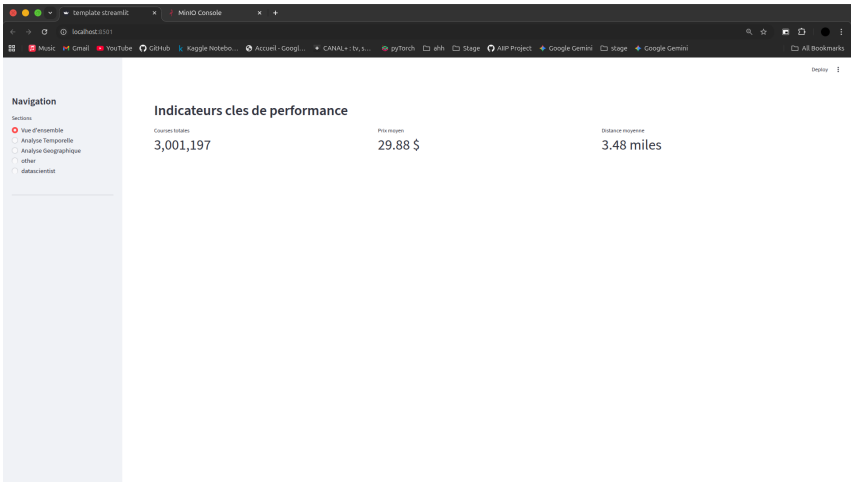

FIGURE 2 – KPI

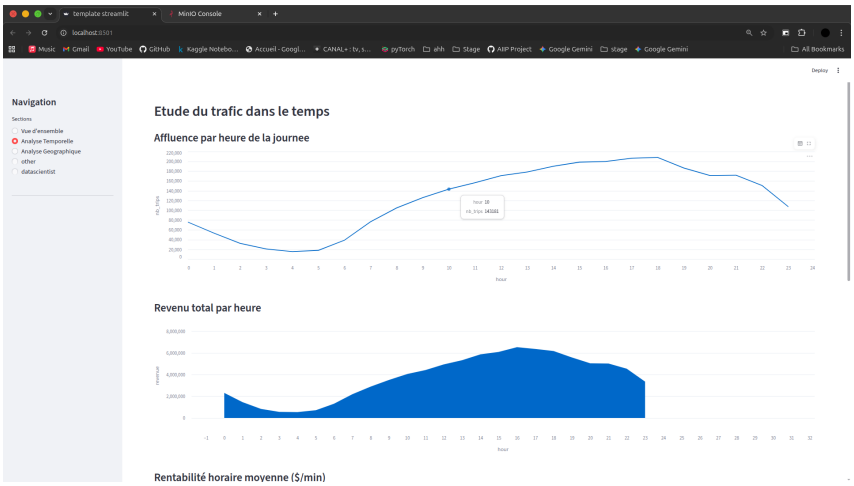

FIGURE 3 – Analyse temporelle 1.1

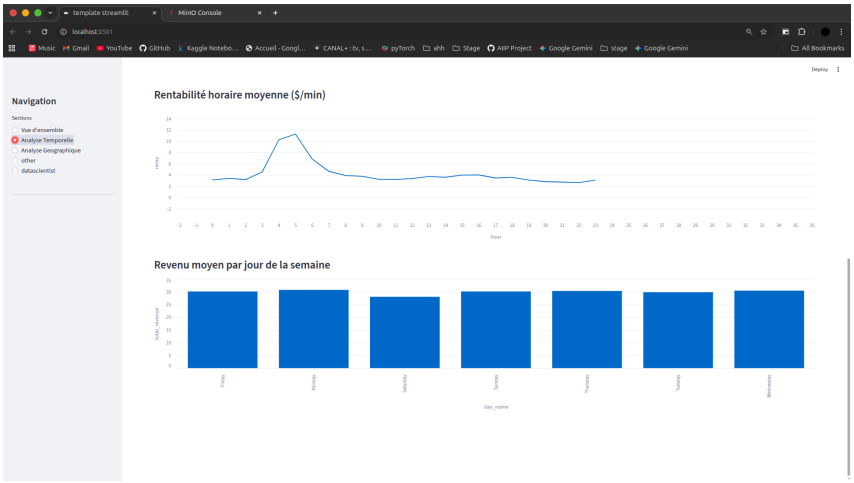

FIGURE 4 – Analyse temporelle 2.2

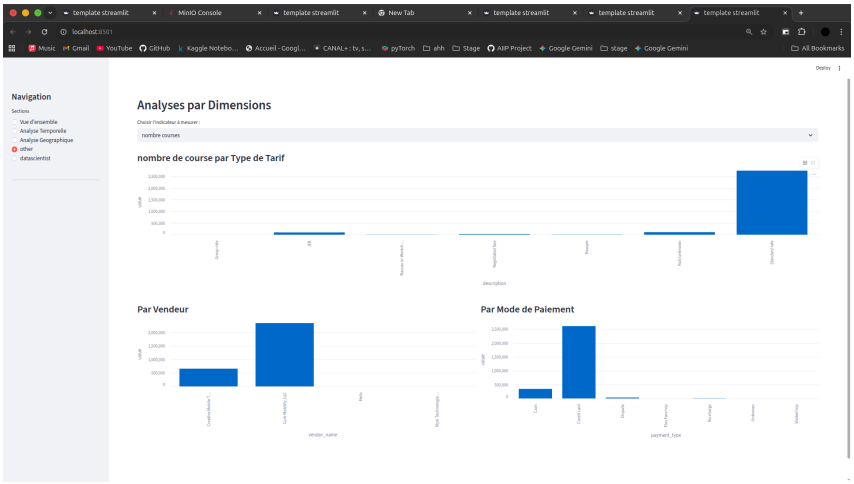

FIGURE 5 – Analyse métier

5.2 Map et technique

Pour la visualisation spatiale, nous avons implémenté une carte choroplèthe interactive avec Plotly.

— **Développement** : La création de cette carte a nécessité l'intégration d'un fichier **GeoJSON** spécifique aux zones de taxi de New York afin de tracer les limites précises de chaque secteur.
— **Fonctionnalité** : L'utilisateur peut basculer entre une vue par zone ou par quartier (Une échelle logarithmique a été ajouté pour la visualisation)

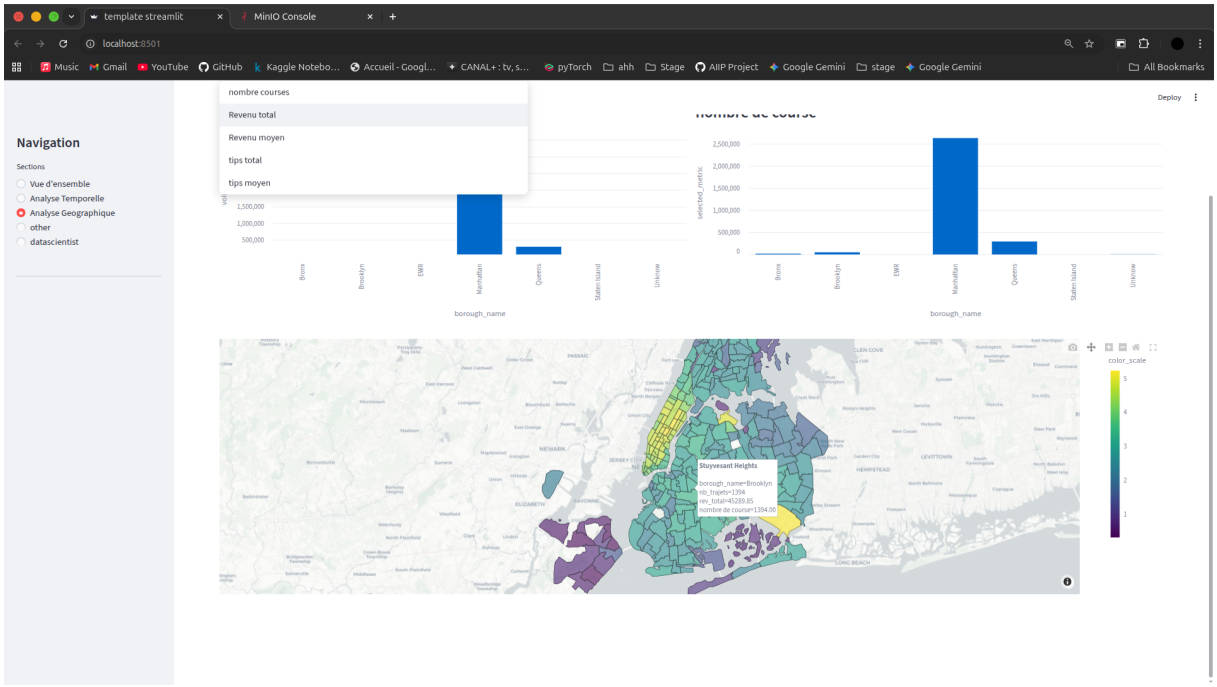

FIGURE 6 – Map dashboard

A la fin nous nus somme rendu compte que le fiichier `app.py` qui lançait notre app streamlit était trop long donc pour le raccourcir, nous avons fait un fichier `querries.py` qui contient toutes les requêtes faites sur notre base de données. Nous avons rencontrés beaucoup de problèmes avec ces reqêtes en passant sous un environement uv, la version de sql alchemy avait changé et nous avions des problèmes de case.

5.3 Analyses

Le dashboard nous a permis de tirer plusieurs conclusions importantes sur le marché des taxis jaunes :

- **Distribution Géographique** : L'analyse confirme que les taxis jaunes opèrent quasi exclusivement à Manhattan et vers l'aéroport JFK. Les autres secteurs sont très peu desservis, car ils sont réglementairement réservés aux "Green Taxis".
- **Analyse des Prestataires et Tarifs** : Nous avons observé une absence totale de données pour certains fournisseurs (*VendorID* 6 et 7), qui correspondent à des services de chauffeurs privés. De même, le tarif "Group Ride" est devenu quasi inexistant, ce type de tarif a disparu dans ces dernières années.
- **Facteurs d'influence du prix** : La visualisation montre que le prix dépend principalement de la **distance** et du **temps de trajet**. Le nombre de passagers n'influence pas ou peu le tarif, conformément à la politique tarifaire de New York (sauf Van, plus de 4 personnes).
- **Analyse des marges** : Dans l'onglet datascientist nous avons mis une map de corrélation des features numériques et calculer la moyenne des frais pour le chauffeur.

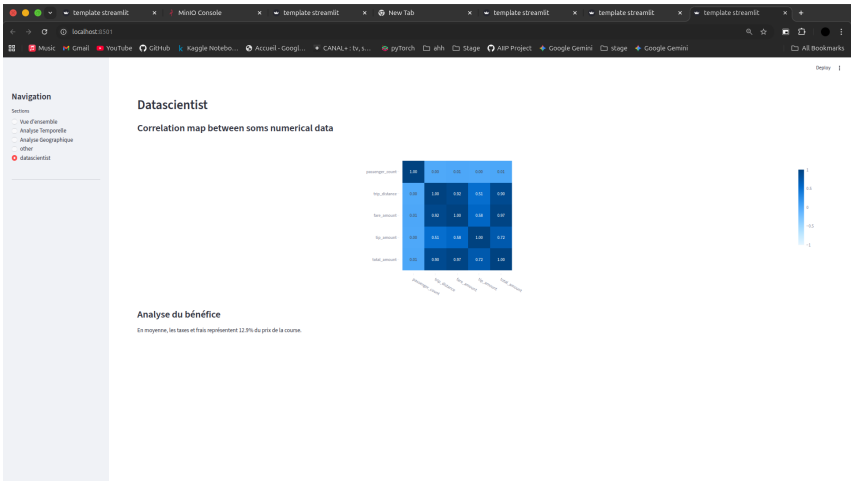

FIGURE 7 – data

L'objectif final est de prédire le prix d'une course en fonction des caractéristiques du trajet.

5.4 Choix du modèle et préparation

Nous avons utilisé XGBoost (*XGBRegressor*), un modèle connu pour ses performances sur les données numérique. Avant de faire notre entraînement, un Notebook a permis de réaliser une Analyse Exploratoire et une recherche d'hyperparamètres par Validation Croisée.

XGBoost est un modèle amélioré d'arbre de décision qui va exécuté plusieurs fois l'arbre en regardant ou la précédente exécution a fait une erreur, c'est un des meilleurs modèles de machine learning mais qui reste très rapide (l'éxécution a pris quelques secondes sur nos machines) Pour optimiser le modèle, nous avons créé de nouvelles variables à partir des données brutes : l'heure de la journée, le jour de la semaine et surtout la durée du trajet (*duration*). Le nettoyage a été affiné par rapport à la branche 1 :

— **Gestion des classes rares** : Le tarif "Group Ride" (*RatecodeID* 6) n'avait que 3 occurrences. Nous l'avons fusionné avec la catégorie 99 (Unknown) pour stabiliser l'apprentissage.
— **Définition de la Target** : Notre cible est le montant total hors pourboires (*total_amount - tip_amount*). Nous avons exclu les pourboires car ils dépendent de facteurs humains imprévisibles (qualité du service, humeur du client) qui ajouteraient du bruit au modèle.

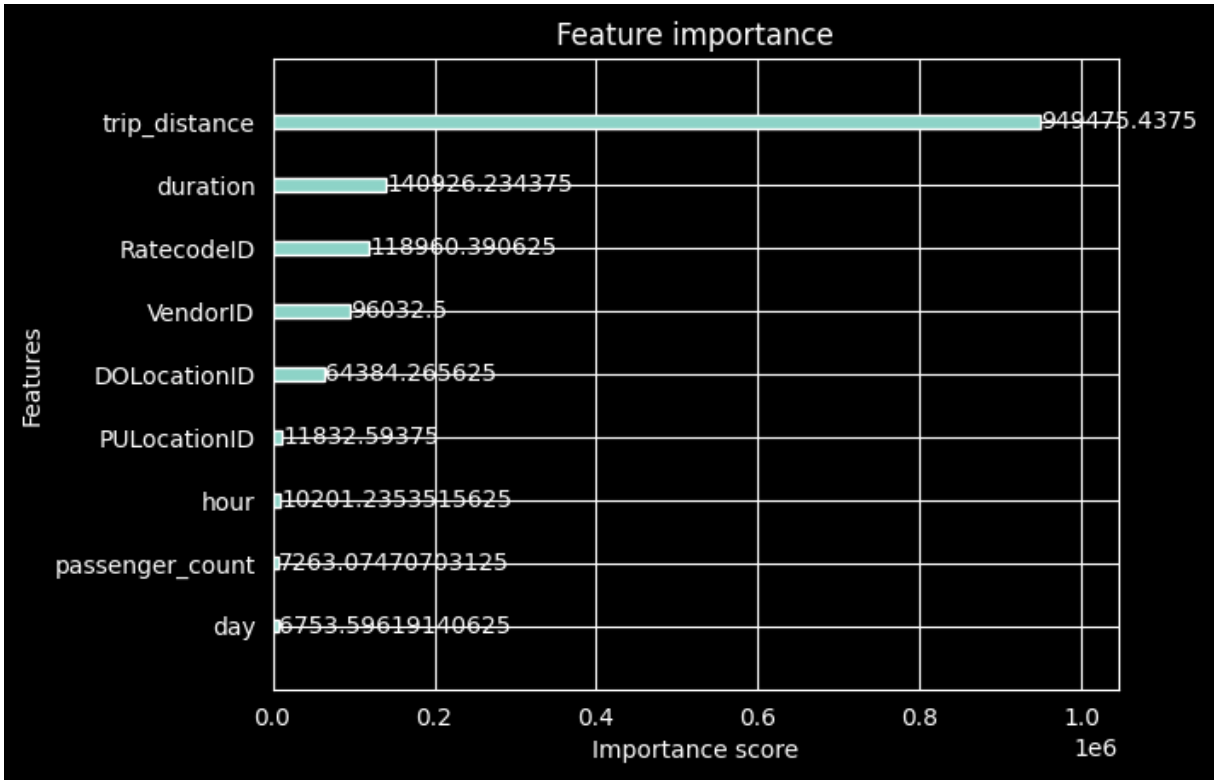

FIGURE 8 – feature importance

5.5 Entraînements

Les données sont récupérées directement depuis le bucket `nyc-raw` sur Minio. Nous avons activé l'option `enable_categorical=True` de XGBoost pour traiter les IDs de zone et les variables temporelles comme des catégories.

Pour garantir la fiabilité de l'inférence, nous sauvegardons un artifact contenant le modèle, mais aussi les `dtypes` et l'ordre des colonnes. Cela assure que les données envoyées pour une prédiction individuelle respectent exactement le format attendu par le modèle.

5.6 Inférence et Tests

Le script d'inférence intègre une fonction `check_data` qui valide les entrées avant la prédiction :

— **Types** : Vérification des entiers et flottants.
— **Plages de valeurs** : Heure (0-23), jour (0-6), et IDs de zone (1-265).

L'objectif était un RMSE inférieur à 10. Notre modèle obtient un score de 2.74, ce qui représente une erreur très faible par rapport au prix moyen d'une course (environ 24 $).

5.7 Démonstration : L'application Streamlit

Une interface interactive permet de simuler une course en temps réel.

Fonctionnement :

— **Interface Latérale** : Choix du nombre de passagers, de l'heure et du tarif.
— **Carte Folium** : Sélection visuelle des points de départ et d'arrivée. L'ID de zone (*LocationID*) est extrait automatiquement via un fichier **GeoJSON**.
— **API OSRM** : Calcul automatique de la distance réelle et du temps de trajet estimé. Un "Mode manuel" est disponible en cas de panne de l'API.
— **Prédiction** : Un bouton lance l'estimation du prix. Un menu déroulant permet de vérifier la validité des données envoyées au modèle.

Améliorations possibles : Le modèle pourrait inclure les mois pour être plus précis. Cependant, comme nous travaillons actuellement sur un échantillon d'un seul mois pour simplifier nous ne pouvons pas le faire

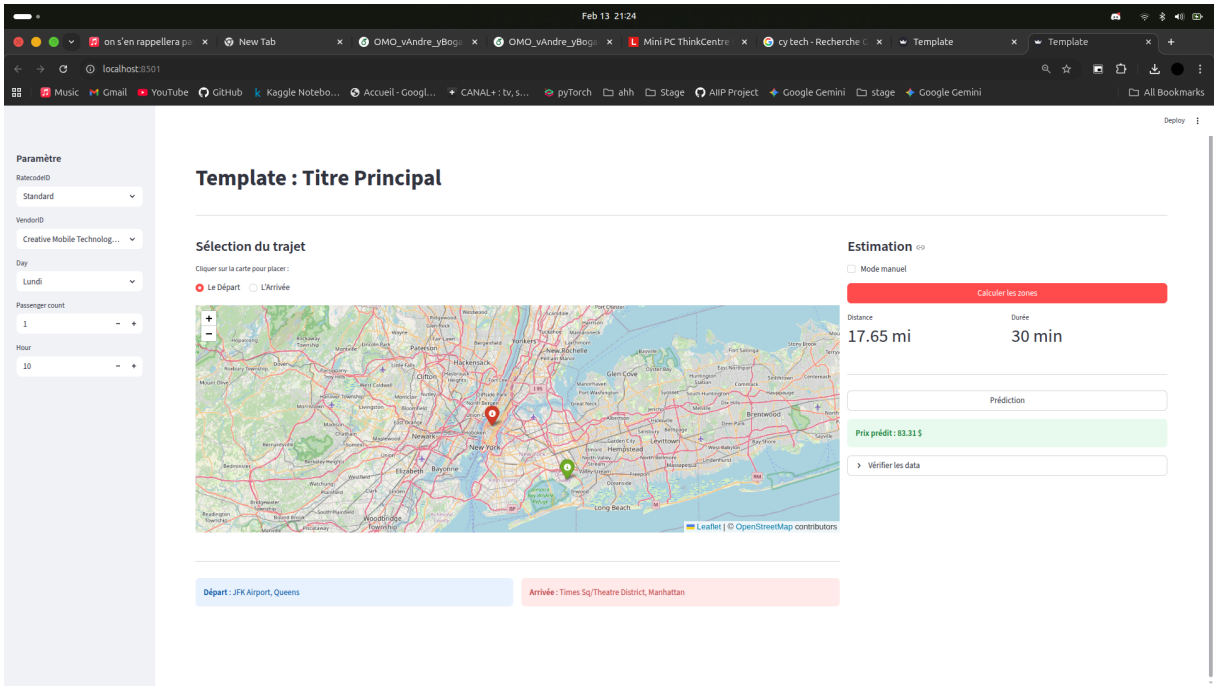

FIGURE 9 – App de prédiction avec API (distance et temps)

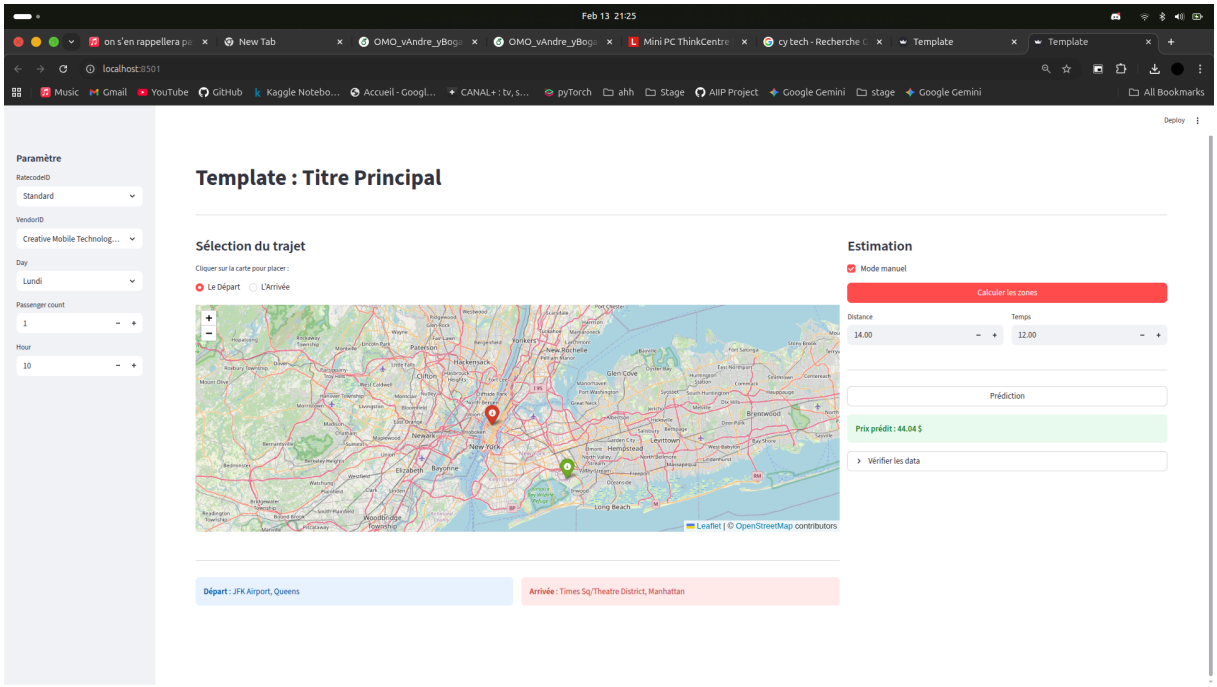

FIGURE 10 – App de prédiction mode manuelle

5.8 Références

· Yellow Trips Data Dictionary,

https ://www.nyc.gov/assets/tlc/downloads/pdf/data_ dictionary_ trip_ records_ yellow.pdf