Rapport Big Data

Etudiants: Clément Delamotte, Julien Michaud
Encadrant: Rakib Sheikh

Année 2025-2026

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

Pour le premier tp, nous créons d'abord un contexte Spark et téléchargeons dans un bucket Minio les données de courses de new york. Au début, nous devions changer l'url à partir de laquelle nous téléchargeons les fichiers *parquets*. Comme nous pouvons copier le lien sans avoir à cliquer dessus, nous avons décidé d'optimiser le processus en récupérant l'url via les arguments d'exécution.
Ainsi la commande pour télécharger les données de janvier 2025 devient :
> **sbt "run https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2025-01.parquet"**

Exercice 2 : Nettoyage de données et ingestion multi branche

Fait après l'exercice 3, le but est de remplir les tables PostgreSQL à l'aide des données réelles fournies par nyc.gov. Comme nous avions introduit des contraintes au niveau sql, nous n'avions pas forcément besoin d'en introduire dans le script Scala.

Exercice 3 : Data Warehouse et création d'un modèle multidimensionnel

Nous avons choisi le modèle en étoiles car les données de Yellow Taxi décrivent un processus avec un fait central (fact_trip) qui contient des mesures (montants, distance, durée), et une multitude de dimensions à faible cardinalité (vendor, rate_code, etc.).

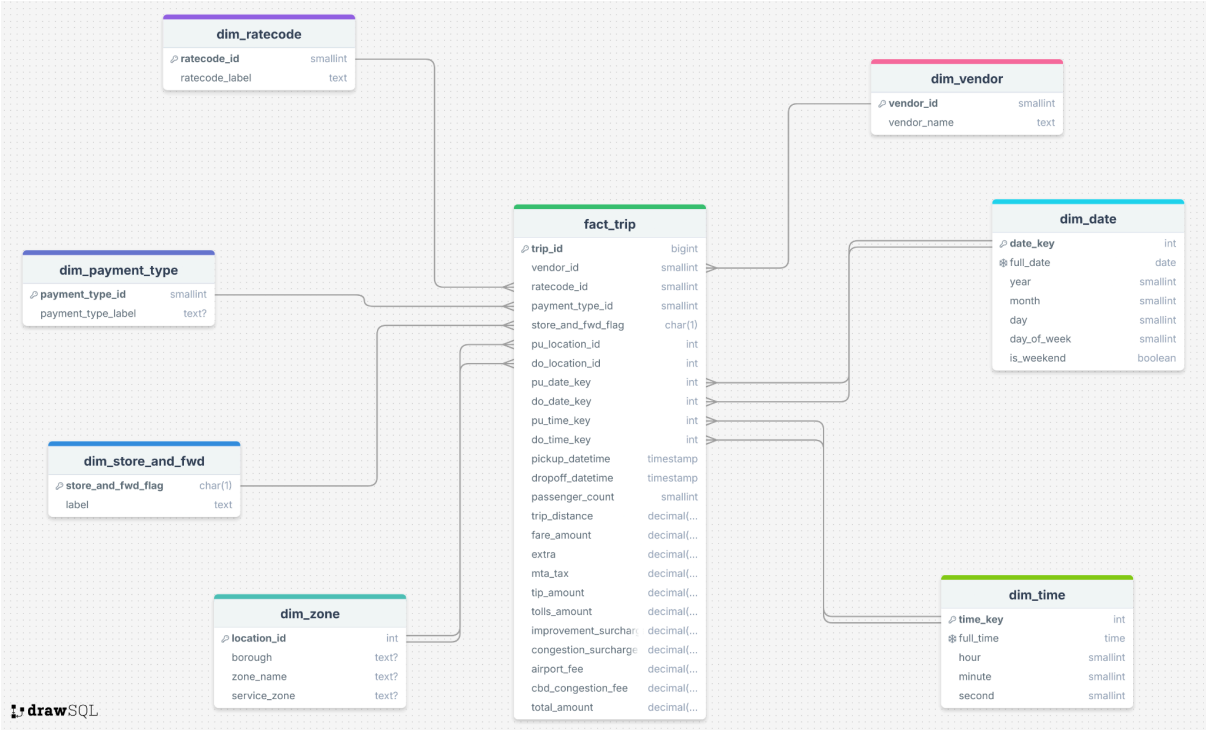


Diagramme de la base de donnée

Exercice 4 : Visualisation de données

En utilisant Streamlit, nous avons réalisé une interface web dans laquelle un utilisateur rentre des données liées à une course de taxi et où notre modèle essaie de produire une estimation du prix de la course.

Exercice 5: Implémentation d'un modèle de Machine Learning

En récupérant les données dans le bucket minIO (sur la branche de Machine Learning, dans le dossier data/raw/), on effectue un filtrage avec un initialisation lazy du jeu de données.

On commence par récupérer les features qui nous intéressent et on construit des features plus intéressantes et moins lourdes en mémoire. Dans le modèle que nous avons développé, nous avons négligé la construction d'une nouvelle feature basée sur le temps de trajet effectué par le taxi pour une course donnée.

Parmi toutes les features, nous avons choisi les features suivantes:

- **Passenger_count**: On pourrait supposer que plus il y a de passagers, plus le coût du trajet est cher.

- **PULocationID**: C'est le point de rendez-vous entre le client et le chauffeur. En fonction de la classification de certaines zones, cela pourrait amener à des différences tarifaires.
- **DOLocationID**: C'est la destination du client. En fonction de la classification de certaines zones, cela pourrait amener à des différences tarifaires.
- **Fare_amount**: Le montant initial qui va evidemment influencer le prix total.
- **Extra**: Des services fournis aux clients qui peuvent augmenter la facture et donc le prix total.
- **Tip_amount**: Il est possible que certaines zones donnent plus ou moins de bonus au chauffeur afin qu'il puisse être rémunérer et c'est pour cela que c'est une caractéristique vraiment intéressante à prendre en compte, surtout qu'il est difficile de savoir à l'avance combien un chauffeur va recevoir de la part d'un client.
- **Trip_distance**: Il semble clair que plus le chauffeur conduit, plus il consomme de carburant ce qui a un coût et peut donc influencer fortement le prix final donné au client.

La raison pour laquelle nous avons pas choisi de continuer en ajoutant le temps de chaque course en tant que features est que nous avions déjà atteint une RMSE (Root Mean Square Error) inférieure à 10 comme voulu et que sa valeur était **proche de 1 pour le mois de Janvier** uniquement (RMSE=1.30).

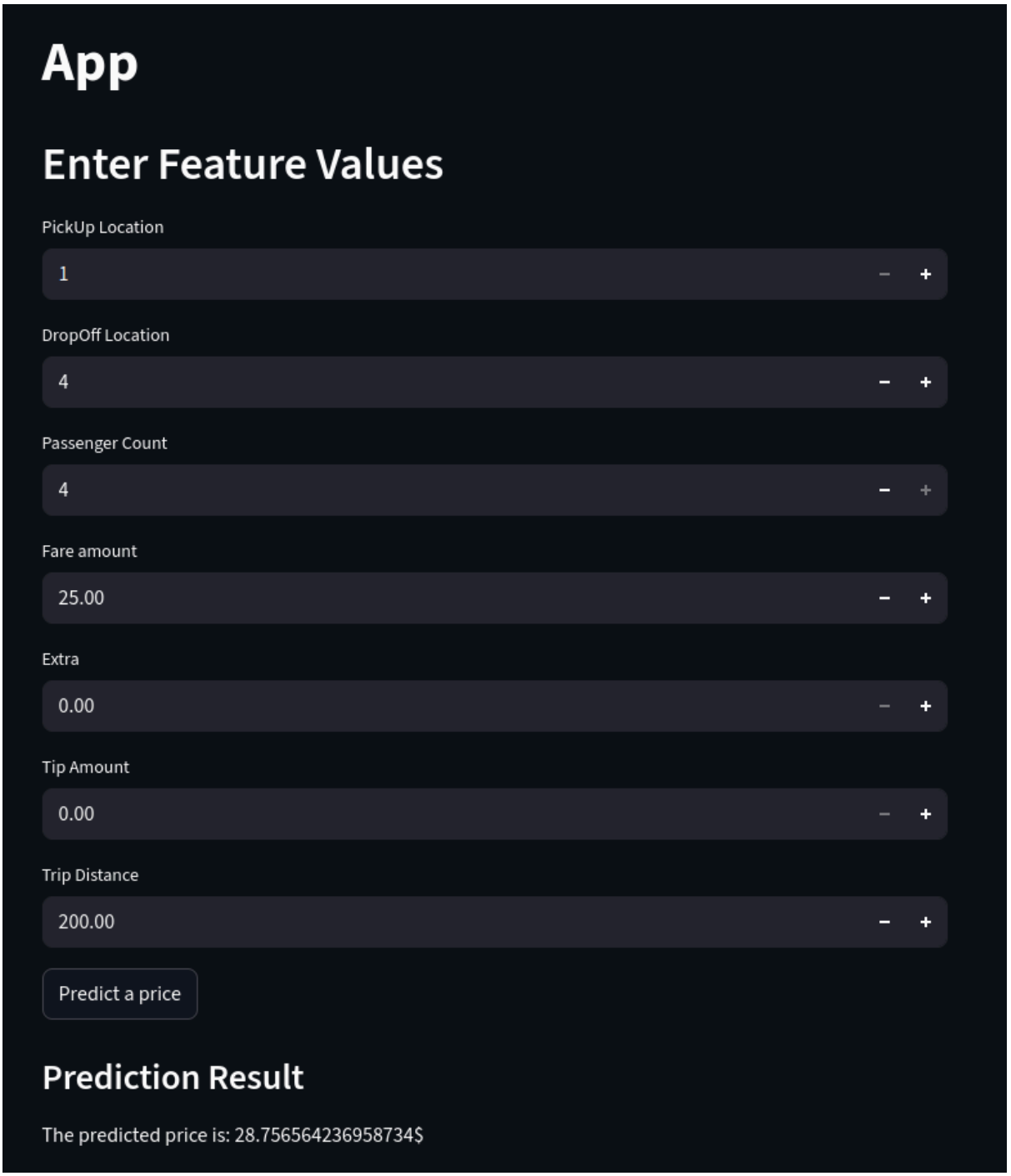
Le modèle qui a été choisi est une **régression avec Random Forest** qui est une approche simpliste mais efficace dans ce cas d'exemple car nous avons un nombre limité de features / d'inputs ce qui rend le calcul d'une prédiction relativement plus court.

Pour faire démarrer le programme:
> **uv run machine_learning.py**
Ne pas oublier de synchroniser l'environnement virtuel
> **uv sync pyproject.toml**

Pour tester l'application streamlit:
> **streamlit run app.py**

P.S: Ne pas oublier de faire **docker compose up** afin d'avoir accès à minIO qui va être appelé pour la gestion et le traitement des données brutes.

Annexe


App
Enter Feature Values
PickUp Location
1
DropOff Location
4
Passenger Count
4
Fare amount
25.00
Extra
0.00
Tip Amount
0.00
Trip Distance
200.00
Predict a price
Prediction Result
The predicted price is: 28.756564236958734$