Rapport du TP : TP : Mise en place d'un Pipeline Big Data avec Dask sur le Dataset NYC Taxi

Fait par : Khadija Nachid Idrissi && Rajae Fdili

1. Introduction

Ce TP avait pour objectif de concevoir une application web permettant l'upload d'un fichier de données, son nettoyage automatique, la consultation de statistiques et d'informations sur ce fichier, ainsi que le téléchargement du fichier nettoyé. Le backend

est développé en FastAPI, utilisant Dask pour la gestion efficace des données, tandis que le frontend en React permet une interface utilisateur simple et intuitive.

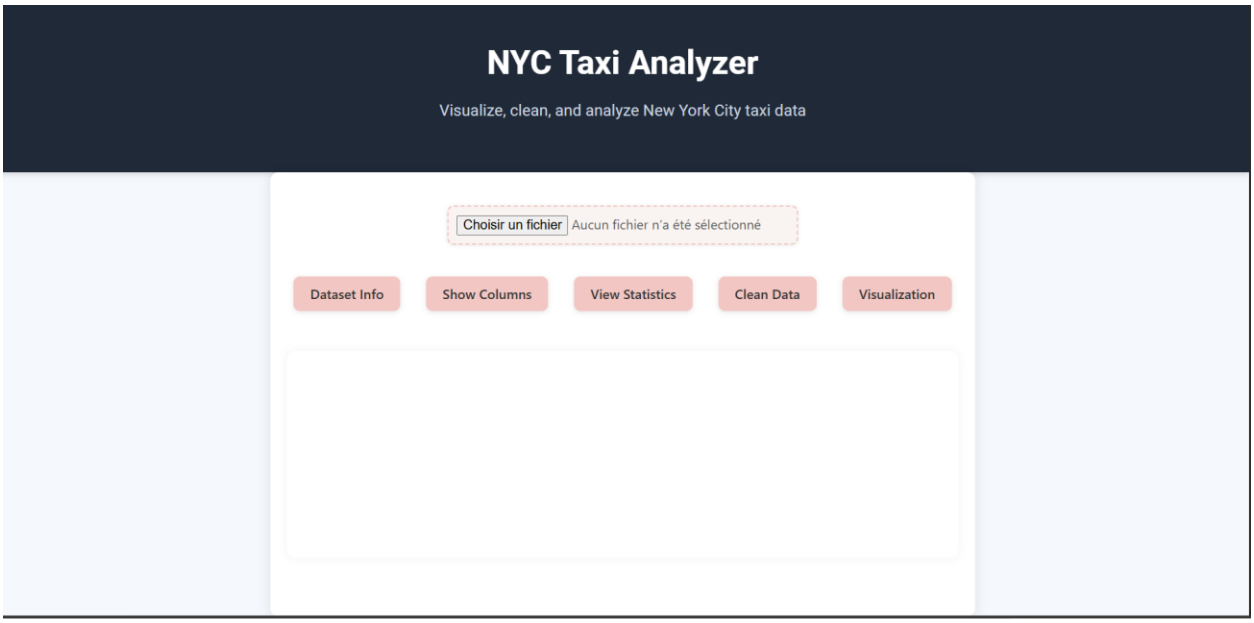
2. Technologies utilisées

- **FastAPI** : framework backend Python rapide, asynchrone, support natif de la validation.
- **Dask** : manipulation de grands datasets, parallélisation, traitement paresseux.
- **Pandas** : manipulation de données tabulaires (via Dask).
- **React** : interface utilisateur moderne, gestion des états, appels API.
- **CORS Middleware** : pour permettre la communication frontend/backend locale.
- **FileResponse / JSONResponse** : gestion des réponses fichiers et JSON.

3. Architecture générale

- **Backend FastAPI** expose plusieurs routes (endpoints) :
  - `/upload` : upload d'un fichier (csv, excel, parquet)
  - `/dataset_info` : infos globales sur le dataset (nb lignes, colonnes, valeurs manquantes, doublons)
  - `/columns` : liste colonnes + types
  - `/statistics` : statistiques descriptives (moyenne, écart-type, compte)
  - `/clean` : nettoyage + sauvegarde du fichier nettoyé
  - `/trip-peaks` : analyse du nombre de trajets par heure (sur un dataset taxi)
  - `/download_cleaned` : téléchargement du fichier nettoyé
- Gestion locale des fichiers dans les dossiers `uploaded_files` et `output_files`.
- Utilisation de variables globales pour stocker le chemin des fichiers uploadés et nettoyés.

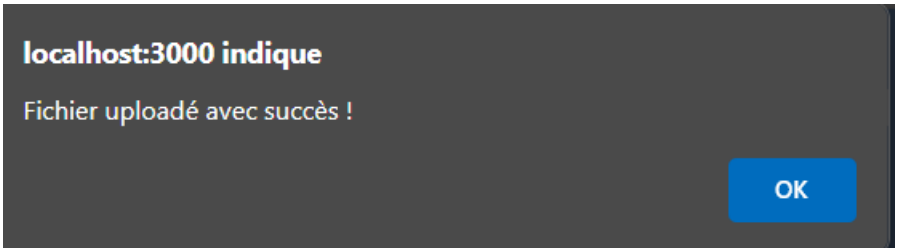
4. Description détaillée des fonctionnalités et frontend associé



4.1 Upload du fichier

- **Backend** : Endpoint POST `/upload` qui accepte un fichier (csv/xlsx/parquet).
- **Frontend** :
    - Un bouton `Choisir un fichier` (input file) et un bouton `Uploader`.

    - Après sélection, le fichier est envoyé au backend via fetch POST.
    - Affichage du nom du fichier uploadé ou message d'erreur.



4.2 Affichage des infos globales (`/dataset_info`)

- **Backend** : GET retourne nombre de lignes, colonnes, doublons, valeurs manquantes.
- **Frontend** :
    - Bouton `Afficher infos dataset`.
    - Après clic, fetch GET `/dataset_info`.
    - Affichage dans une zone texte ou tableau lisible.

Informations générales sur le dataset

Catégorie	Valeur
Nombre total de lignes	3 582 628
Nombre total de colonnes	19
Nombre de doublons	0
Valeurs manquantes par colonne	
VendorID	0
tpep_pickup_datetime	0
tpep_dropoff_datetime	0
passenger_count	426190
trip_distance	0
RatecodeID	426190
store_and_fwd_flag	426190
PULocationID	0
DOLocationID	0
payment_type	0
fare_amount	0
extra	0

extra	0
mta_tax	0
tip_amount	0
tolls_amount	0
improvement_surcharge	0
total_amount	0
congestion_surcharge	426190
Airport_fee	426190

4.3 Liste des colonnes (`/columns`)

- **Backend** : GET qui retourne noms et types.
- **Frontend** :
  - Bouton `Afficher colonnes`.
  - Liste des colonnes affichée sous forme tableau ou liste.

Nom de la colonne	Type de données
VendorID	int32
tpep_pickup_datetime	datetime64[us]
tpep_dropoff_datetime	datetime64[us]
passenger_count	int64
trip_distance	float64
RatecodeID	int64
store_and_fwd_flag	string
PULocationID	int32
DOLocationID	int32
payment_type	int64
fare_amount	float64
extra	float64
mta_tax	float64
tip_amount	float64
tolls_amount	float64
improvement_surcharge	float64
total_amount	float64
total_amount	float64
congestion_surcharge	float64
Airport_fee	float64

4.4 Statistiques descriptives (`/statistics`)

- **Backend** : Moyenne, écart-type, compte des colonnes numériques.
- **Frontend** :
    - Bouton `Afficher statistiques`.
    - Affichage clair par colonne.

Statistiques numériques

mean

Colonne	Valeur
VendorID	1.758
passenger_count	1.338
trip_distance	4.517
RatecodeID	2.256
PULocationID	164.211
DOLocationID	163.156
payment_type	1.076
fare_amount	18.683
extra	1.364
mta_tax	0.483
tip_amount	3.191
tolls_amount	0.542
improvement_surcharge	0.974
total_amount	27.121

total_amount	27.121
congestion_surcharge	2.252
Airport_fee	0.143

std_dev

Colonne	Valeur
VendorID	0.431
passenger_count	0.840
trip_distance	302.412
RatecodeID	10.678
PULocationID	64.471
DOLocationID	69.570
payment_type	0.647
fare_amount	18.121
extra	1.809
mta_tax	0.121
tip_amount	4.033
tolls_amount	2.157

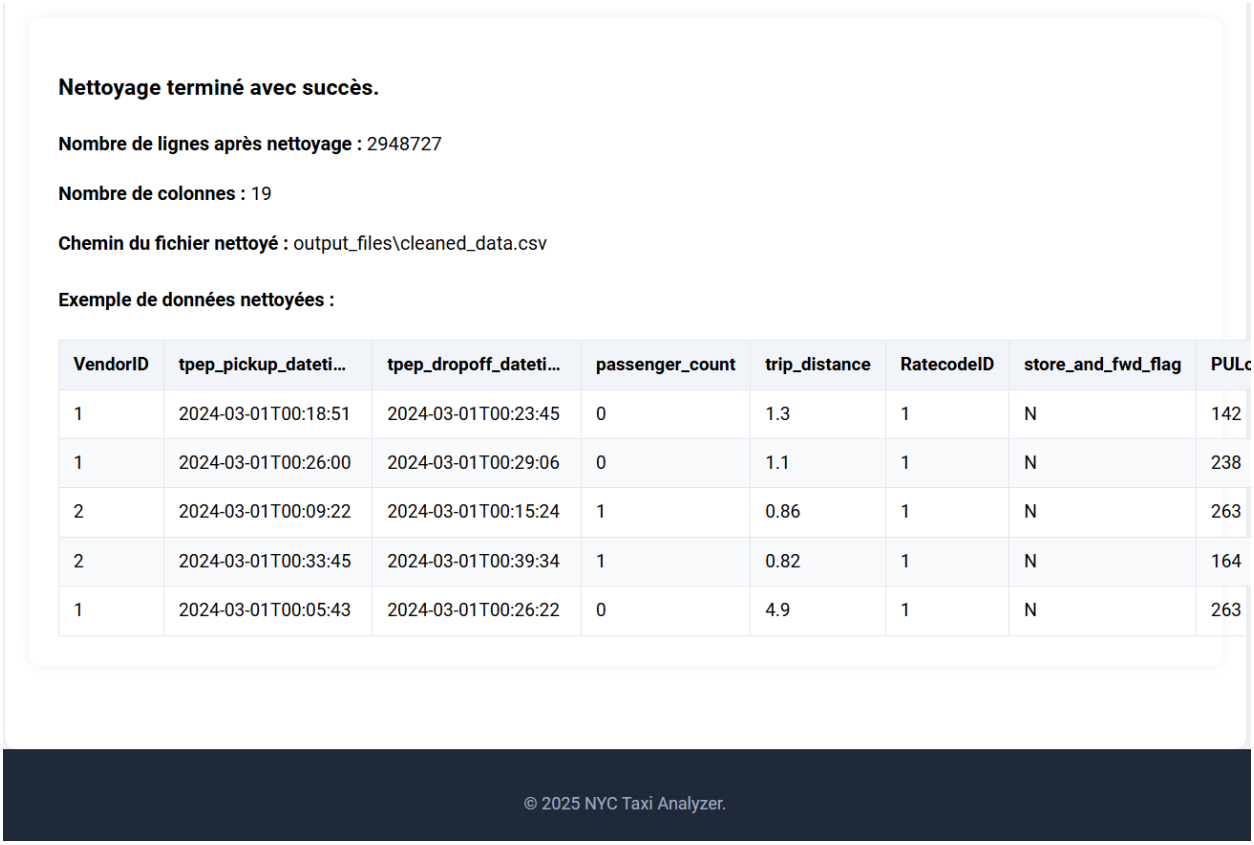
improvement_surcharge	0.223
total_amount	22.832
congestion_surcharge	0.838
Airport_fee	0.492

count

Colonne	Valeur
VendorID	3582628.000
passenger_count	3156438.000
trip_distance	3582628.000
RatecodeID	3156438.000
PULocationID	3582628.000
DOLocationID	3582628.000
payment_type	3582628.000
fare_amount	3582628.000
extra	3582628.000
mta_tax	3582628.000
tip_amount	3582628.000
tip_amount	3582628.000
tolls_amount	3582628.000
improvement_surcharge	3582628.000
total_amount	3582628.000
congestion_surcharge	3156438.000
Airport_fee	3156438.000

4.5 Nettoyage des données (`/clean`)

- **Backend** : POST, nettoyage (dropna, filtres), sauvegarde, renvoie résumé.
- **Frontend** :
    - Bouton `Nettoyer dataset`.
    - Après clic, POST `/clean`.
    - Affiche résumé : nb lignes, colonnes, aperçu des 5 premières lignes, chemin du fichier nettoyé.



Nettoyage terminé avec succès.

Nombre de lignes après nettoyage : 2948727

Nombre de colonnes : 19

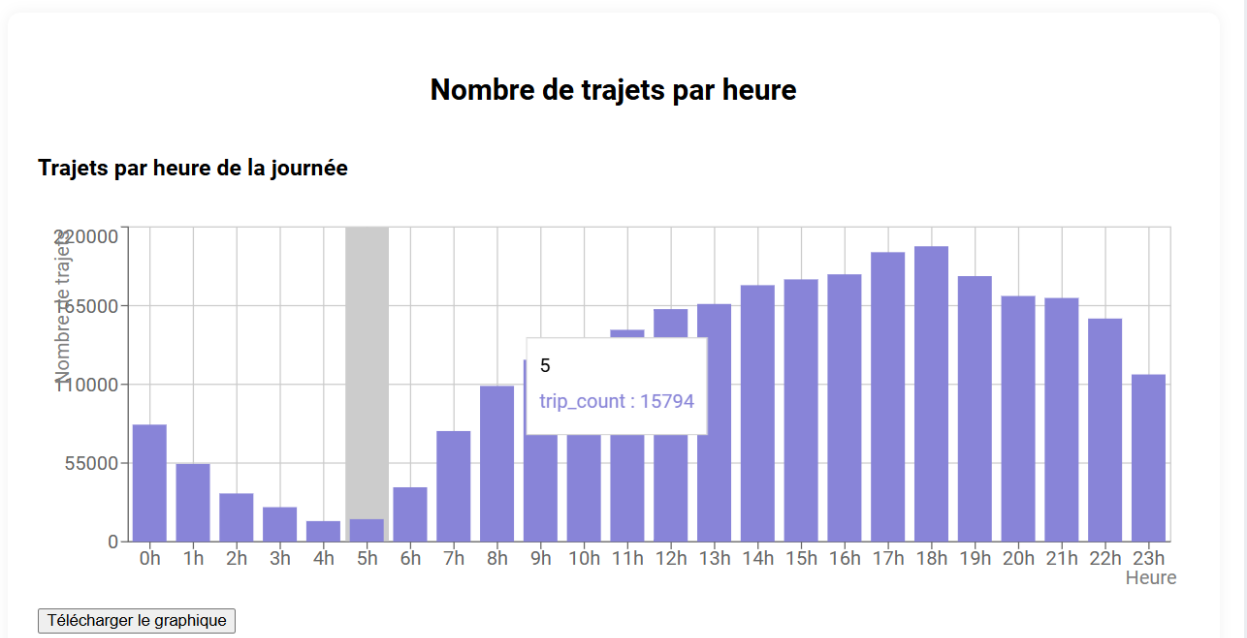
Chemin du fichier nettoyé : output_files\cleaned_data.csv

Exemple de données nettoyées :

VendorID	tpep_pickup_dateti...	tpep_dropoff_dateti...	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULo
1	2024-03-01T00:18:51	2024-03-01T00:23:45	0	1.3	1	N	142
1	2024-03-01T00:26:00	2024-03-01T00:29:06	0	1.1	1	N	238
2	2024-03-01T00:09:22	2024-03-01T00:15:24	1	0.86	1	N	263
2	2024-03-01T00:33:45	2024-03-01T00:39:34	1	0.82	1	N	164
1	2024-03-01T00:05:43	2024-03-01T00:26:22	0	4.9	1	N	263

4.6 Analyse des pics de trajets (`/trip-peaks`)

- **Backend** : GET, calcul trajets par heure.
- **Frontend** :
    - Bouton `Afficher pics trajets`.
    - Affichage graphique simple (bar chart) ou liste.

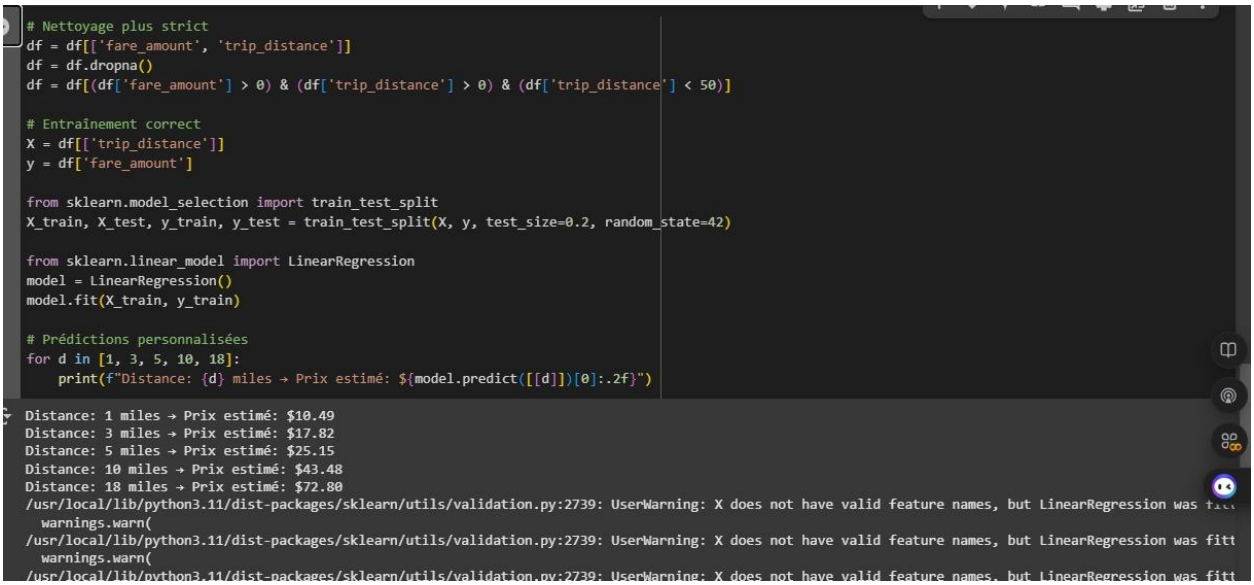


5-Module de machine learning : LinearRegression

LinearRegression est un module de machine learning utilisé pour prédire une valeur en fonction d'une autre.

Il cherche à tracer une droite qui passe le plus près possible des points d'un graphique, pour modéliser la relation entre une variable indépendante (X) et une variable dépendante (Y).

- Exemple : prédire le prix d'un trajet en taxi à New York en fonction de la distance parcourue, grâce à une régression linéaire simple.



```
# Nettoyage plus strict
df = df[['fare_amount', 'trip_distance']]
df = df.dropna()
df = df[(df['fare_amount'] > 0) & (df['trip_distance'] > 0) & (df['trip_distance'] < 50)]

# Entraînement correct
X = df[['trip_distance']]
y = df['fare_amount']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)

# Prédictions personnalisées
for d in [1, 3, 5, 10, 18]:
    print(f"Distance: {d} miles → Prix estimé: ${model.predict([[d]])[0]:.2f}")
```

```
Distance: 1 miles → Prix estimé: $10.49
Distance: 3 miles → Prix estimé: $17.82
Distance: 5 miles → Prix estimé: $25.15
Distance: 10 miles → Prix estimé: $43.48
Distance: 18 miles → Prix estimé: $72.80
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LinearRegression was fitt
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LinearRegression was fitt
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LinearRegression was fitt
```

6. Défis rencontrés

- Gestion des fichiers uploadés et chemins en variable globale — problème si plusieurs utilisateurs.
- Dask est paresseux, il faut faire `.compute()` pour récupérer les résultats.
- Certaines colonnes absentes ou erreurs format datetime.
- Correction d'erreurs 500 notamment dans endpoint `/dataset_info` (ajout de variable globale `uploaded_df`).
- Synchronisation frontend/backend et gestion CORS.

7. Améliorations possibles

- Passage à une gestion utilisateur pour éviter les conflits globaux.

- Frontend plus sophistiqué avec formulaires validés.
- Plus d’analyses statistiques avancées et graphiques interactifs.
- Ajout d’un système d’authentification.
- Dockerisation de l’application pour déploiement.

8. Conclusion

Ce TP a permis de combiner la puissance de FastAPI et Dask pour traiter des datasets volumineux, tout en proposant une interface frontend simple et réactive. J’ai amélioré mes compétences en API REST, manipulation de fichiers, et communication asynchrone frontend/backend.