Python

Data Science, manipuler et visualiser les données

OCTOBRE 2025

Retour sur vos questions du troisième jour

Matplotlib : gray et Greys_r

Tous les deux suivent la convention 0 -> noir, 1 -> blanc (Greys suit la convention 0 -> blanc, 1 -> noir), gray

Mais gray suit une interpolation « linéaire » des niveaux de gris, tandis que Greys_r en suit une « améliorée ».

CPU et GPU pour calculs distribués

	CPU	GPU
Architecture	Quelques cœurs puissants	Des milliers de petits cœurs
Type de tâches	Séries / logiques complexes	Massivement parallèles / répétitives
Mémoire	RAM plus grande, latence plus faible	Mémoire vidéo (VRAM) plus rapide mais limitée
Flexibilité	Très généraliste	Très spécialisé (calculs matriciels, flottants)
Idéal pour	Préparation de données, contrôle, I/O, calculs séquentiels	Calculs intensifs, traitement d'images, deep learning

CPU et GPU pour calculs distribués

CPU

- Tâches sont fortement séquentielles (chaîne d'opérations dépendantes).
- Le traitement implique beaucoup de logique conditionnelle (if/else, branchements).

OU

- Tâche IO bound/memory-bound, par exemple un prétraitement d'images plutôt simple mais sur de multiples images (lecture puis redimensionnement/conversions)

OU

- Coordonner des sous-tâches sur plusieurs machines (distribution, orchestration).

Exemples :

- Chargement et, annotnormalisation d'un grand nombre d'images.
- Calculs distribués via Dask, Ray ou Spark avant l'entraînement d'un modèle.
- Compression, filtrage de datasets.

CPU et GPU pour calculs distribués

GPU

- Calcul est massivement parallèle : chaque pixel, matrice ou tenseur peut être traité indépendamment.
- Calculs matriciels à grande échelle.
- Traitement d'images intensif : convolution, filtrage, transformations, Deep Learning.

Exemples :
- Entraînement de modèles de vision (TensorFlow, PyTorch, etc.).
- Inférence rapide sur de grandes images ou vidéos.

Les bibliothèques qui exploitent le GPU :
- CUDA / cuDNN (bas niveau, Nvidia)
- PyTorch,
- TensorFlow,
- JAX
- OpenCV CUDA,
- cuPy,
- Numba,
- RAPIDS cuDF/cuML

CPU et GPU pour calculs distribués

I/O-bound	L'opération est limitée par l'entrée/sortie : lecture ou écriture sur disque, SSD, réseau ou stockage cloud. Même si le CPU/GPU est rapide, il attend que les données arrivent.
Memory-bound	L'opération est limitée par la vitesse de la mémoire RAM (lecture/écriture dans des tableaux en mémoire). Si le CPU ou GPU doit attendre que les données soient chargées depuis la RAM, c'est un goulot d'étranglement.
Compute-bound	L'opération est limitée par la puissance de calcul du CPU ou GPU. Les calculs eux-mêmes prennent le temps principal.

CPU et GPU pour calculs distribués

Dans un environnement distribué, on combine souvent les deux :
- Le CPU orchestre les tâches et répartit les données.
- Le GPU exécute les calculs lourds sur chaque nœud.

Exemple de pipeline d'imagerie distribué :
- CPU (cluster) : lecture du dataset, partitionnement avec Dask.
- GPU (nœuds) : traitement d'images (CNN, débruitage, segmentation).
- CPU : agrégation des résultats, sauvegarde, analyse.

Frameworks utiles :
- Dask + cuDF / cuML
- PyTorch DistributedDataParallel
- HorovodRay + GPUs
- Spark RAPIDS Accelerator

Plateformes Cloud

	Plateformes	Détails
Prototypage rapide	Google Colab, PythonAnywhere	Environnements prêts à l'emploi avec GPU/TPU gratuits
Fonctions événementielles	AWS Lambda, Google Cloud Functions, Azure Functions	Exécution serverless avec intégration cloud native
Applications web	Heroku, Anaconda Cloud	Déploiement facile, support de bibliothèques Python
Data science & machine learning	*Slides suivants*	Support étendu des bibliothèques et outils spécialisés

Plateformes Cloud

1. Google Cloud Platform (GCP) – Vertex AI

Points forts :

- Classé leader dans le Magic Quadrant 2024 de Gartner pour les plateformes de data science et de machine learning.
- Intégration transparente avec des outils tels que TensorFlow, Vertex AI Workbench, et BigQuery.
- Infrastructure GPU avancée pour l'entraînement et l'inférence de modèles.

Cas d'usage : Idéal pour les entreprises recherchant une plateforme unifiée pour le ML, l'IA générative et l'analyse de données massives.

Plateformes Cloud

2. Amazon Web Services (AWS) – SageMaker

Points forts :

- Plateforme complète pour le cycle de vie du machine learning, de la préparation des données au déploiement des modèles.
- Large écosystème de services intégrés, y compris SageMaker Studio, SageMaker Pipelines, et SageMaker JumpStart.

Cas d'usage : Convient aux entreprises nécessitant une flexibilité maximale et une intégration avec d'autres services AWS.

Plateformes Cloud

3. Microsoft Azure – Azure Machine Learning

Points forts :

- Intégration étroite avec les outils Microsoft tels que Power BI, Azure Databricks, et Azure Synapse Analytics.
- Plateforme hybride permettant le déploiement sur site ou dans le cloud.

Cas d'usage : Adapté aux organisations utilisant déjà l'écosystème Microsoft et nécessitant une solution ML hybride.

Plateformes Cloud

4. Oracle Cloud Infrastructure (OCI)

Points forts :

- Investissements significatifs dans l'infrastructure GPU, avec des projets visant à déployer 50 000 GPU MI450 Instinct d'AMD d'ici 2026.
- Partenariats stratégiques, y compris un investissement de 500 milliards de dollars avec OpenAI.

Cas d'usage : Convient aux entreprises nécessitant une infrastructure haute performance pour des charges de travail d'IA intensives.

Plateformes Cloud

5. Databricks

Points forts :

- Plateforme collaborative basée sur Apache Spark, idéale pour le traitement de données massives et le machine learning.
- Intégration avec des outils open-source tels que MLflow, Delta Lake, et Koalas.

Cas d'usage : Parfaite pour les équipes de data science collaborant sur des projets de grande envergure nécessitant une gestion avancée des données.

Plateformes Cloud

« Snowflake fait du python en le traduisant en SQL » :

Snowflake supporte Python via Snowpark for Python qui permet de manipuler des DataFrames côté serveur : filtrage, jointures, agrégations, etc. Ensuite, on peut éventuellement récupérer le résultat final en Pandas si nécessaire.
En d'autres termes, Snowflake compile les opérations en SQL et les exécute dans son moteur de base de données.

Snowflake UDFs en Python : on peut aussi créer des User-Defined Functions (UDFs) en Python, mais elles s'exécutent dans un environnement isolé côté serveur (Snowflake Python runtime).Ces UDFs ne sont pas converties en SQL ; elles sont exécutées comme du code Python dans un conteneur.

Plateformes Cloud

« Snowflake fait du python en le traduisant en SQL » :

L'idée de **"Python traduit ou optimisé pour exécuter sur un moteur de données"** n'est pas unique à Snowflake.

	Fonction équivalente	Fonctionnement
BigQuery (GCP)	BigQuery ML & BigQuery Python client	BigQuery ML permet certaines transformations ML en SQL. Avec pandas-gbq ou BigQuery Python Client, on écrit en Python, mais en arrière-plan, des requêtes SQL sont envoyées.
Databricks / Spark	PySpark	Les transformations DataFrame PySpark (filter, join, select) sont traduites en plan d'exécution Spark, équivalent à SQL distribuée, mais optimisé pour Spark.
Redshift (AWS)	Redshift Spectrum + Python UDF	Les UDF Python s'exécutent côté serveur dans un conteneur, mais les DataFrame-style APIs (via Redshift ML) traduisent certaines opérations en SQL.
Azure Synapse / SQL Pool	SQL + Python UDF	Les Python UDF sont exécutées côté serveur pour des traitements spécifiques, les opérations sur tables peuvent être optimisées via SQL.

Plateformes Cloud

« Snowflake fait du python en le traduisant en SQL » :

L'idée de **"Python traduit ou optimisé pour exécuter sur un moteur de données"** n'est pas unique à Snowflake.

Librairie	Mécanisme	Remarque
pandas	Tout s'exécute en Python (CPU), pas de traduction SQL automatique	Adapté aux données petites à moyennes.
Modin	API pandas compatible, utilise Ray ou Dask en backend	Les opérations pandas sont parallélisées sur CPU et parfois distribuées sur cluster, mais pas traduites en SQL.
Dask DataFrame	API pandas-like, plan distribué	Les transformations sont traduites en graph de tâches exécuté en parallèle, concept similaire à Snowpark mais pour CPU/cluster local.
Polars	API Rust/Python DataFrame	Très rapide, multi-thread, certaines opérations SQL-like optimisées, pas besoin de SQL explicite.
Vaex	Lazy evaluation, out-of-core	Transformations sur gros datasets hors mémoire → planification comme Snowpark mais tout reste en Python local.