

# Cours d'introduction à la Data Science

**Chargé du cours : Jules DEGILA, PhD, MCF**  
**Assistante : Aurélie KPOZE, PhD**

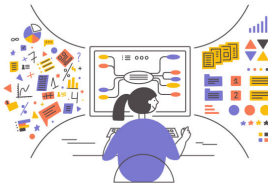
jules.degila@imsp-uac.org, satou.kpoze@imsp-uac.org

18 Novembre 2025



# Chapitre 2

## De la collecte au stockage des données



Collecter, transformer, normaliser, stocker les données.

# Plan du Chapitre 2

- 1 Sources de données
- 2 Collecte des données
- 3 Ingestion (batch / streaming)
- 4 Validation et qualité
- 5 Nettoyage et normalisation
- 6 Transformation
- 7 Stockage (SQL, NoSQL, Data Lake)
- 8 Bonnes pratiques

# Objectifs du Chapitre 2 — De la collecte au stockage des données

**À la fin du chapitre, vous serez capables de :**

- Identifier les principales sources de données
- Expliquer les différentes méthodes de collecte et d'ingestion (batch, streaming)
- Comprendre la notion de schéma et définir correctement les types de données
- Appliquer les étapes clés du nettoyage et de la normalisation des données
- Choisir un format de stockage adapté (SQL, NoSQL, Data Lake)
- Décrire le rôle d'un pipeline de données dans l'architecture



- Les données sont au cœur des systèmes modernes, mais elles arrivent souvent sous des formes brutes et hétérogènes.
- Comprendre comment elles sont collectées et préparées est essentiel pour toute analyse fiable.
- Nous verrons comment les données circulent : de leur source, à leur ingestion, jusqu'à leur stockage.
- L'objectif : mettre en place une base solide pour les chapitres suivants

- **Pipeline de données :**

- Ensemble d'étapes automatisées qui font circuler les données depuis la source jusqu'au système de stockage ou d'analyse.
- Étapes typiques : collecte → ingestion → validation → nettoyage → transformation → stockage.

- **ETL (Extract–Transform–Load) :**

- **Extract** : extraction des données depuis les sources.
- **Transform** : nettoyage et transformation avant le stockage.
- **Load** : chargement des données transformées dans la base.
- Approche classique, transformation réalisée en dehors du stockage.

- **ELT (Extract–Load–Transform) :**

- **Extract** : extraction des données brutes.
- **Load** : chargement direct dans le data lake/warehouse.
- **Transform** : transformations réalisées **dans** le système de stockage.

- La collecte consiste à récupérer les données depuis leurs sources d'origine.
- Elle peut être manuelle ou automatisée selon les besoins.
- Première étape du cycle de vie de la donnée : la qualité de la collecte conditionne tout le reste.
- Enjeux : fiabilité, exhaustivité, régularité, conformité légale (RGPD).

- **Sources internes** : bases SQL, ERP, CRM, fichiers métier.
- **Sources externes** : open data, partenaires, API publiques.
- **Méthodes** :
  - collecte manuelle (exports, fichiers) ;
  - collecte automatisée (API, ETL/ELT, ingestion de logs).
- **Formats courants** : CSV, Excel, JSON, XML, logs.

- Sources hétérogènes, formats variés, qualité parfois faible.
- Données manquantes, bruit, incohérences, délais d'accès variables.
- Contraintes légales : RGPD, sécurité des échanges.
- Bonnes pratiques :
  - automatiser au maximum ;
  - valider les données dès la collecte ;
  - documenter les sources et la fréquence de collecte.

- L'ingestion consiste à faire entrer les données dans le système interne (base, data lake, data warehouse).
- Étape qui suit la collecte : on fait rentrer les données dans l'infrastructure.
- Deux grandes approches : **ingestion en batch** et **ingestion en streaming**.
- Objectif : garantir un flux régulier, fiable et traçable de données.

## ● Ingestion en batch

- Les données sont chargées par lots à intervalles réguliers.
- Exemple : importer un fichier chaque nuit ; mise à jour toutes les heures.
- Adaptée aux besoins non urgents et aux gros volumes.

## ● Ingestion en streaming

- Les données arrivent en continu, presque en temps réel.
- Exemple : flux de capteurs, logs serveurs, transactions financières.
- Adaptée aux applications temps réel (alertes, détection de fraude).
- Le choix dépend des besoins métier : fréquence, volume, rapidité.

- Vérifier que les données ingérées respectent un **schéma** :
  - types corrects (dates, nombres, catégories) ;
  - valeurs attendues (listes, plages, formats).
- Détecter les **anomalies** :
  - valeurs manquantes ou aberrantes ;
  - doublons ;
  - incohérences (âges négatifs, dates futures...).
- Contrôler la **qualité globale** :
  - complétude ;
  - cohérence ;
  - unicité ;
  - exactitude.
- Objectif : garantir que seules des données fiables passent aux étapes suivantes (nettoyage, transformation, stockage).

- **Complétude** : vérifier l'absence de valeurs manquantes.
- **Exactitude** : données correctes et proches de la réalité.
- **Cohérence** : absence de contradictions entre variables ou systèmes.
- **Unicité** : pas de doublons (ID, enregistrements, valeurs).
- **Validité** : respect des formats, des règles et des valeurs autorisées.
- **Actualité** : données suffisamment récentes pour l'usage prévu.

- **Traitement des valeurs manquantes :**
  - suppression (lignes/colonnes) ;
  - imputation simple (moyenne, médiane, valeur par défaut).
- **Gestion des doublons :**
  - détection et suppression des enregistrements en double.
- **Correction des incohérences :**
  - âges négatifs, dates impossibles, formats invalides.
- **Nettoyage des valeurs aberrantes :**
  - détection d'outliers, erreurs de saisie, valeurs extrêmes.
- Objectif : obtenir des données fiables, propres et cohérentes avant transformation.

- **Pandas (Python) :**
  - gestion des valeurs manquantes : `isna()`, `fillna()`, `dropna()`
  - doublons : `drop_duplicates()`
  - types : `astype()`, `to_datetime()`
  - nettoyage texte : `str.lower()`, `str.strip()`, `replace()`
- **NumPy :**
  - opérations mathématiques pour détecter outliers et incohérences ;
  - gestion de tableaux numériques propres et rapides.
- **SQL :**
  - filtrage et correction : `WHERE`, `CASE WHEN`
  - nettoyage des doublons : `DISTINCT`, `GROUP BY`
  - conversion des formats (dates, nombres).
- **Outils spécialisés :**
  - **Great Expectations** : tests automatiques de qualité
  - **Pandera** : validation de schémas Pandas
  - **OpenRefine** : nettoyage manuel (typos, clusters de valeurs)
- **ETL/ELT (Airflow, Talend, Dataiku, etc.) :**
  - automatisation du nettoyage à grande échelle ;
  - règles métiers, détection d'erreurs, corrections répétables.

- **Standardisation des formats :**
  - dates (AAAA-MM-JJ), unités (€, kg, m), casse (minuscules).
- **Harmonisation des catégories :**
  - regroupement des variantes : “FR”, “fr”, “France”.
  - correction des fautes : “Eletronique” → “Électronique”.
- **Typage correct des colonnes :**
  - dates, booléens, numériques, catégories.
- **Mise à l'échelle des variables numériques** (si nécessaire) :
  - normalisation (0–1), standardisation (z-score).
- Objectif : obtenir des données homogènes, comparables et exploitables.

# Normalisation des valeurs

- **Objectif** : rendre les variables numériques comparables, réduire l'influence de l'échelle, améliorer les modèles.
- **Normalisation Min–Max** (mise à l'échelle 0–1) :

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Utilisée pour : réseaux neuronaux, k-NN, distances.

- **Standardisation (Z-score)** :

$$z = \frac{x - \mu}{\sigma}$$

où  $\mu$  est la moyenne et  $\sigma$  l'écart-type. Utilisée pour : régression, SVM, PCA.

- **Normalisation L2** (vecteurs) :

$$x_{\text{norm}} = \frac{x}{\|x\|_2}$$

Utilisée pour : traitement du texte, distances cosines.

- **Remarque** : le choix dépend du modèle et des distributions.

- La transformation consiste à modifier, enrichir ou restructurer les données pour les rendre adaptées à l'analyse ou au stockage.
- Elle intervient après le nettoyage / normalisation.
- Objectifs :
  - créer de nouvelles variables (features) ;
  - changer la structure (agrégation, pivot, fusion) ;
  - optimiser les données pour le reporting ou le machine learning.
- Étape centrale dans un pipeline ETL/ELT.

- **Agrégation :**
  - somme, moyenne, maximum, comptage par groupe.
- **Fusion et jointures :**
  - combiner plusieurs tables (JOIN).
- **Restructuration :**
  - pivot, unpivot, réorganisation des colonnes.
- **Feature engineering :**
  - nouvelles variables dérivées (jours, âges, catégories, ratios).
- **Encodage :**
  - transformation des catégories en variables numériques (one-hot, label encoding).
- **Normalisation / standardisation** (si modèle ML).

- **Pandas (Python) :**
  - groupby, merge, pivot, apply, transformations sur colonnes.
- **SQL (dans ETL ou ELT) :**
  - SELECT, GROUP BY, JOIN, CASE WHEN, fonctions de dates.
- **Data warehouses (ELT) :**
  - Snowflake, BigQuery, Redshift : transformations directement dans la base.
- **ETL / Orchestration :**
  - Talend, Airflow, Dataiku, dbt : pipeline de transformations complexes.
- **Spark (big data) :**
  - transformations distribuées sur grands volumes.

- Le stockage consiste à conserver les données de manière durable, organisée et accessible pour l'analyse ou les applications.
- Objectifs :
  - assurer la disponibilité et la sécurité des données ;
  - permettre l'interrogation efficace (SQL, API, BI) ;
  - gérer des volumes croissants (scalabilité).
- Types de stockage : SQL, NoSQL, Data Lake.
- Le choix dépend du type de données et des besoins métier.

- **Bases SQL (relationnelles)**

- Données structurées (tables, schéma strict).
- Langage d'interrogation : SQL.
- Exemples : PostgreSQL, MySQL, Oracle.

- **Bases NoSQL**

- Données semi-structurées ou non structurées.
- Schéma flexible (documents, clés-valeurs, graphes).
- Exemples : MongoDB, Cassandra, Redis.

- **Data Lakes**

- Stockage de données brutes à grande échelle.
- Formats : CSV, JSON, Parquet, images, logs.
- Exemples : S3, Azure Data Lake, HDFS.

- **SQL** si :
  - données structurées ;
  - besoin de relations (JOIN) ;
  - forte cohérence des données.
- **NoSQL** si :
  - données variées, volumineuses ou non structurées ;
  - schéma flexible ;
  - besoin de performance en lecture/écriture.
- **Data Lake** si :
  - besoin de stocker de grandes quantités de données brutes ;
  - usage analytique ou machine learning ;
  - préparation avant stockage dans un data warehouse.
- La bonne solution dépend du volume, du format et des usages.

- **CSV (Comma-Separated Values)**

- Format tabulaire simple, très utilisé.
- Lisible et compatible avec tous les outils.
- Limites : pas de types, pas de schéma, redondant.

- **JSON (JavaScript Object Notation)**

- Format semi-structuré, hiérarchique (clé-valeur).
- Idéal pour APIs, NoSQL, documents flexibles.
- Taille plus grande que CSV.

- **XML**

- Ancien format structuré basé sur des balises.
- Très verbeux, encore utilisé en finance/industries.

- **Parquet**

- Format **colonnaire**, compressé et optimisé pour l'analyse.
- Idéal pour Data Lakes (S3, ADLS, GCS).
- Très efficace pour requêtes analytiques et volumes importants.

- **Avro**

- Format **orienté ligne**, avec schéma intégré.
- Adapté au streaming (Kafka), pipelines ELT.

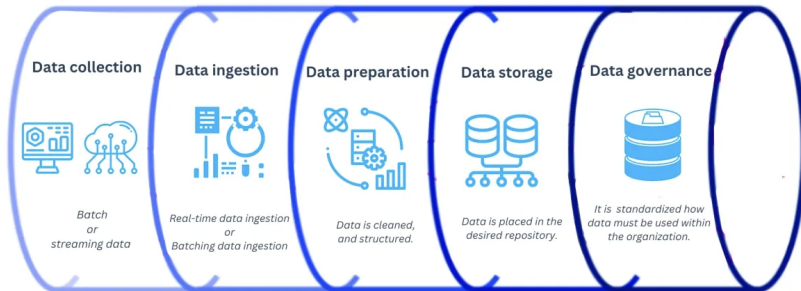
- **ORC**

- Format colonnaire performant, proche de Parquet.
- Spécialement optimisé pour Hadoop/Hive.

- **Résumé :**

- CSV/JSON : lisibles, universels ;
- Parquet/ORC : analytiques, compressés ;
- Avro : idéal pour streaming et schémas évolutifs.

# Pour résumer



# Cas d'étude 1 : Pipeline de données d'un site e-commerce (Amazon)

**Objectif :** comprendre comment un site e-commerce traite les données (logs, commandes, produits, clients) du début à la fin.

- **Sources :**

- Logs de navigation (JSON)
- Base SQL des commandes
- Fichiers produits (CSV)
- Données clients (CRM)

- **Collecte :**

- Batch : export quotidien des commandes
- Streaming : collecte en temps réel des clics via Kafka

- **Ingestion :**

- Données brutes → Data Lake (S3)
- Données structurées → Data Warehouse

# Cas d'étude 1 : Pipeline de données d'un site e-commerce (Amazon)

- **Nettoyage et normalisation :**

- Suppression doublons clients
- Corrections de formats (dates, prix)
- Harmonisation catégories produits

- **Transformation :**

- Jointures : clients + commandes + produits
- Agrégations (ventes / jour, panier moyen)
- Création de variables (jour semaine, type appareil)

- **Stockage final :**

- Données brutes : CSV/JSON
- Données traitées : Parquet dans Data Warehouse

# Travaux Pratiques

# Travail pratique 1 : Description d'un pipeline de données d'un site e-commerce (Amazon)

- Voir notebook1.ipynb

# Travail pratique 2 : Conception d'un pipeline de données Netflix

- Voir notebook2.ipynb

# Résumé du chapitre

- Parcours complet des données : de la collecte à leur stockage.
- Différence entre collecte, ingestion, validation, nettoyage, normalisation et transformation.
- Rôle des pipelines de données, ETL et ELT.
- Types de stockage : SQL, NoSQL, Data Lake et principaux formats (CSV, JSON, Parquet...).
- Enjeux de qualité : schéma, types, indicateurs de qualité, bonnes pratiques.

## Compétences acquises

- Décrire les étapes d'un pipeline de données de bout en bout.
- Identifier les principaux types de stockage et leurs usages.
- Expliquer l'importance de la validation, du nettoyage et de la normalisation.
- Proposer une architecture simple de collecte–ingestion–stockage pour un cas métier.

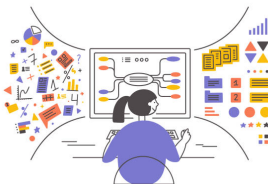
## Livrable

- Un notebook `.ipynb` documenté (code + commentaires).
- Un court paragraphe de conclusion expliquant les choix effectués (validation, nettoyage, format et type de stockage).

- Je peux expliquer, avec mes mots, les grandes étapes d'un pipeline de données.
- Je sais faire la différence entre collecte, ingestion, transformation et stockage.
- Je suis capable de repérer des problèmes de qualité dans un jeu de données (valeurs manquantes, incohérences, doublons).
- Je peux justifier le choix d'un type de stockage (SQL, NoSQL, Data Lake) dans un cas concret.
- Si je devais décrire le pipeline de données d'un service réel (e-commerce, réseau social, etc.), je saurais en tracer les grandes lignes.

# Chapitre 3

## Préparation, exploration et visualisation des données



Comprendre, analyser et visualiser les données.

- ① Préparation avancée des données
- ② Exploration des données (EDA)
- ③ Visualisation des données
- ④ Bonnes pratiques et pièges à éviter
- ⑤ Cas d'étude et travaux pratiques

**À la fin du chapitre, vous serez capables de :**

- Préparer un jeu de données pour l'analyse exploratoire
- Appliquer les méthodes d'analyse univariée, bivariée et multivariée
- Construire et interpréter des visualisations adaptées aux données
- Identifier des tendances, corrélations et anomalies dans un dataset
- Utiliser Python (Pandas, Matplotlib/Seaborn) pour l'EDA et la visualisation

- Étape qui suit la collecte, l'ingestion et le stockage (Chapitre 2).
- Objectif : obtenir un **dataset analysable** : propre, cohérent, typé.
- Lien direct avec la qualité des conclusions et des modèles.
- On se concentre ici sur la préparation pour l'**analyse exploratoire** (EDA).

- Formats courants : CSV, JSON, SQL, Parquet.
- **Fusion de tables :**
  - `merge()`, `join()`, `concat()` en Pandas.
  - Jointures : `inner`, `left`, `right`, `outer`.
- **Restructuration :**
  - `pivot` / `unpivot` ;
  - `melt()` pour passer de large à long.

- Vérifier et corriger les **types** :
  - numériques (int, float), dates, booléens, catégories.
- Conversion :
  - `astype()`, `to_datetime()`, `to_numeric()`.
- Optimisation mémoire :
  - utilisation de `category` pour les colonnes qualitatives ;
  - downcasting des entiers / floats si besoin.

- **Diagnostiquer :**
  - taux de valeurs manquantes par colonne ;
  - patterns (plusieurs colonnes manquantes en même temps).
- **Stratégies :**
  - suppression (lignes/colonnes) ;
  - imputation simple (moyenne, médiane, mode) ;
  - imputation plus avancée (KNN, modèles).
- **Séries temporelles :**
  - interpolation linéaire, forward/backward fill.

- Définition : valeurs extrêmes par rapport au reste des données.
- Méthodes de détection :
  - **IQR** : valeurs en dehors de  $[Q1 - 1.5 \text{ IQR} ; Q3 + 1.5 \text{ IQR}]$ .
  - **Z-score** :  $|z| > 3$ .
- Que faire ?
  - les garder (cas réels importants) ;
  - les corriger ou les tronquer ;
  - les supprimer (si clairement des erreurs).

- **EDA** : Exploratory Data Analysis.
- Objectifs :
  - comprendre la structure des données ;
  - détecter des patterns, tendances, anomalies ;
  - formuler des hypothèses.
- Étape indispensable avant toute modélisation.

- **Variables numériques :**
  - histogrammes, densité (KDE) ;
  - statistiques : moyenne, médiane, variance, quartiles.
- **Variables qualitatives :**
  - diagrammes en barres ;
  - fréquences et modes.
- Questions typiques :
  - la distribution est-elle symétrique, biaisée, multimodale ?
  - y a-t-il des valeurs extrêmes ?

- **Numérique numérique :**
  - scatter plot ;
  - corrélation (Pearson, Spearman).
- **Catégorielle numérique :**
  - boxplots, violin plots ;
  - comparaison de moyennes (ANOVA).
- **Catégorielle catégorielle :**
  - tables de contingence ;
  - heatmaps de fréquences.

- **Matrice de corrélation :**
  - visualisation des corrélations entre nombreuses variables.
- **Scatter matrix / pairplot :**
  - nuages de points pour plusieurs variables simultanément.
- **Analyse par composantes principales :**
  - réduire la dimension ;
  - représenter les données dans un plan 2D/3D.

- Une bonne visualisation doit être :
  - claire ;
  - honnête (pas de manipulation d'échelle) ;
  - adaptée au message.
- Éviter :
  - graphiques 3D inutiles ;
  - couleurs confuses ;
  - légendes manquantes.

# Types de graphiques courants

- **Histogramme** : distribution d'une variable numérique.
- **Boxplot** : quartiles, médiane, outliers.
- **Scatter plot** : relation entre deux variables numériques.
- **Bar chart** : comparaison de catégories.
- **Line plot** : séries temporelles.
- **Heatmap** : matrice de corrélation, tableaux croisés.

- **Matplotlib :**

- bibliothèque de base en Python ;
- très flexible, bas niveau.

- **Seaborn :**

- construit sur Matplotlib ;
- graphiques statistiques, styles préconfigurés.

- **Outils BI :**

- Tableau, Power BI, Metabase (mention).

# Cas d'étude : Dataset Netflix (EDA)

- Exemple de questions :
  - part des films vs séries ;
  - genres les plus fréquents ;
  - évolution du nombre de contenus dans le temps ;
  - distribution de la durée des films.
- Objectif : pratiquer la préparation, l'EDA et la visualisation sur des données réelles.

# Cas d'étude : Données GOZEM (EDA)

- Analyse des trajets :
  - distribution des distances et durées ;
  - détection de trajets aberrants (outliers).
- Analyse temporelle :
  - heures de pointe, jours de semaine vs week-end.
- Visualisations possibles :
  - histogrammes, boxplots, courbes temporelles.

- **TP 1** : EDA sur le dataset Netflix
  - chargement, préparation, nettoyage ;
  - analyse univariée/bivariée ;
  - visualisations (Matplotlib/Seaborn).
- **TP 2** : EDA sur un dataset métier (GOZEM, e-commerce, open data)
  - exploration guidée ;
  - mini-rapport (quelques graphiques + interprétation).

# Résumé du Chapitre 3

- Préparation avancée : typage, fusion, valeurs manquantes, outliers.
- EDA : analyse univariée, bivariée, multivariée.
- Visualisation : choix du graphique adapté, bonnes pratiques.
- Application sur des cas concrets (Netflix, GOZEM).

## Compétences acquises

- Mener une analyse exploratoire complète sur un dataset.
- Mettre en évidence des patterns et des anomalies.
- Communiquer les résultats avec des graphiques clairs.

- Je sais charger et préparer un dataset pour l'analyse.
- Je peux décrire la distribution d'une variable (numérique ou catégorielle).
- Je suis capable de repérer des corrélations et des relations simples.
- Je sais choisir un type de graphique adapté à une question donnée.
- Je peux réaliser une EDA de base sur un nouveau jeu de données.