Chapitre 1:

Introduction et fondations des pipelines de données

4DATA

Benjamin Jurczak

SUPINFO

Sommaire

1. Les pipelines de données : présentation et tour d’horizon
2. Modèles ETL et ELT
3. Outils et technologies associés
4. Cas pratiques

1. Les pipelines de données : présentation et tour d’horizon

1.1 Présentation et tour d’horizon

Définition :

Un pipeline de données est une chaîne **automatisée** de processus permettant de **collecter**, **transformer**, et charger des données d'une ou plusieurs sources vers une destination.

Fonctionnement global:

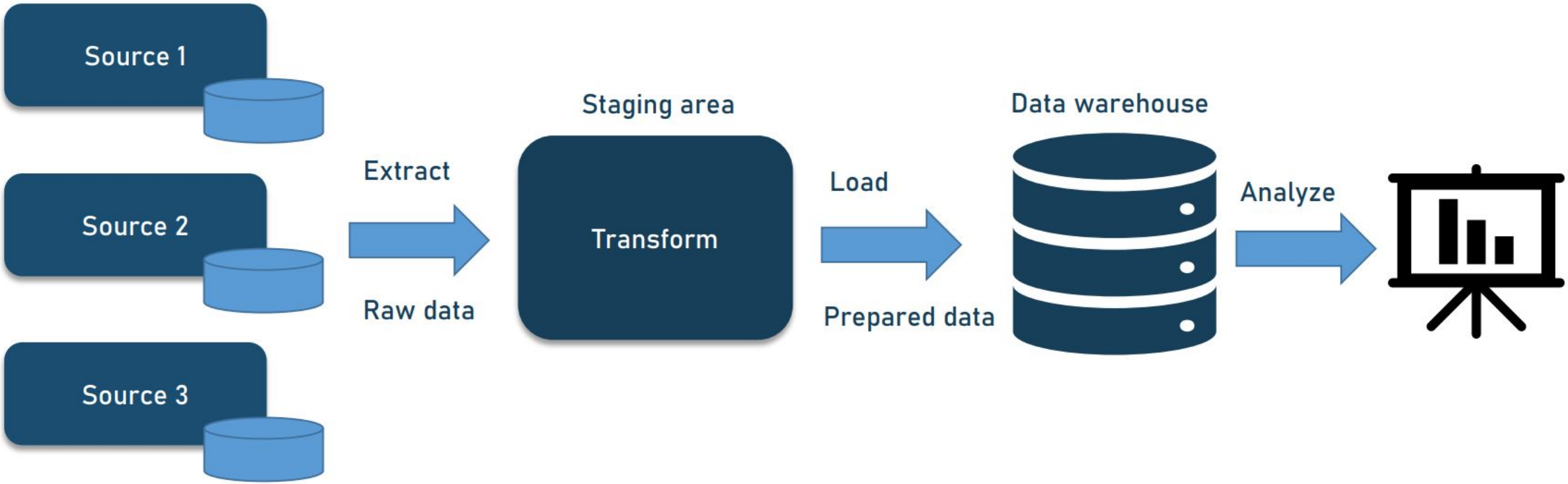
- **Source(s)** : APIs, bases de données, fichiers ...
- **Traitement(s)** : Nettoyage, enrichissement, transformation.
- **Destination** : Entrepôts de données, bases analytiques, outils de visualisation.

Objectif principal:

Garantir un flux continu et fiable de données pour répondre à des besoins métier ou analytiques.

1.1 Présentation et tour d’horizon

Architecture de base



1.1 Présentation et tour d‘horizon

Pourquoi des pipelines de données

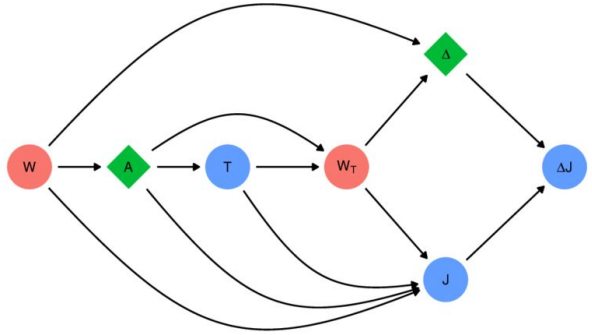
- **Automatisation** :
  Réduire les tâches manuelles et répétitives pour un traitement plus rapide et fiable.

- **Efficacité** :
  Traiter des volumes massifs de données provenant de sources multiples sans intervention humaine.

- **Qualité et cohérence** :
  Standardiser, nettoyer, et enrichir les données pour garantir leur utilisabilité.

- **Flexibilité** :
  S'adapter à des besoins variés : intégration de données, analyses avancées, machine learning.

1.2 Concept de DAG (Direct Acyclic Graph)



Définition :

Un DAG est une structure graphique où les **nœuds** représentent des tâches et les **arcs dirigés** montrent les dépendances entre ces tâches.

Caractéristiques :

- **Dirigé** : Chaque arc indique un sens d'exécution.
- **Acyclique** : Aucun cycle, ce qui garantit qu'une tâche ne peut pas se dépendre elle-même.

Utilité :

Organiser et exécuter des workflows complexes de manière ordonnée et sans ambiguïté.

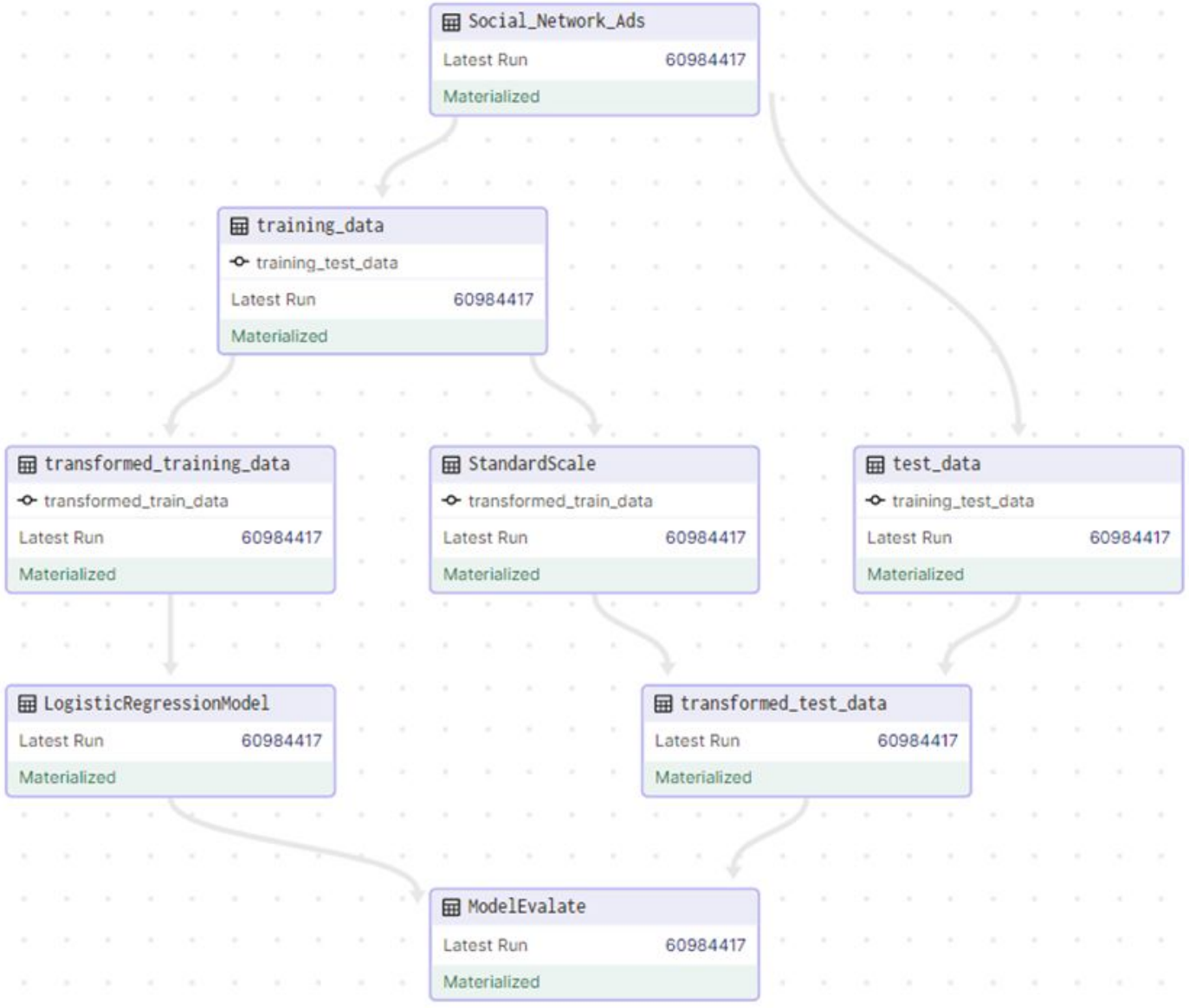
1.3 Lien entre DAG et pipeline de données

Un DAG modélise un pipeline comme un ensemble de tâches interconnectées avec des dépendances clairement définies.

Pourquoi utiliser un DAG ?

- **Orchestration** : Gérer l’ordre d’exécution des tâches (ex. : extraction avant transformation).
- **Fiabilité** : Assurer qu'une tâche ne démarre que si ses dépendances sont terminées avec succès.
- **Optimisation** : Permet l’exécution parallèle des tâches indépendantes, réduisant le temps total.

1.4 Exemple de workflow dagster



1. Les pipelines de données: présentation et tour d’horizon

2. Modèles ETL et ELT

2.1 Introduction aux modèles ETL et ELT

- Les entreprises collectent des données à partir de plusieurs sources (applications, capteurs IoT, partenaires tiers, etc.).

- Ces données doivent être transformées et nettoyées pour devenir exploitables.

- Deux approches principales : **ETL** (Extraction, Transformation, Chargement) et **ELT** (Extraction, Chargement, Transformation).

2.1 Introduction aux modèles ETL et ELT

ETL : Les données sont transformées avant d'être chargées dans le système cible.

ELT : Les données sont chargées telles quelles dans le système cible, puis transformées selon les besoins.

Processus en 3 étapes communs aux deux modèles :

Collecte des données brutes.

Modification des données pour correspondre aux exigences analytiques.

Stockage des données dans la base cible.

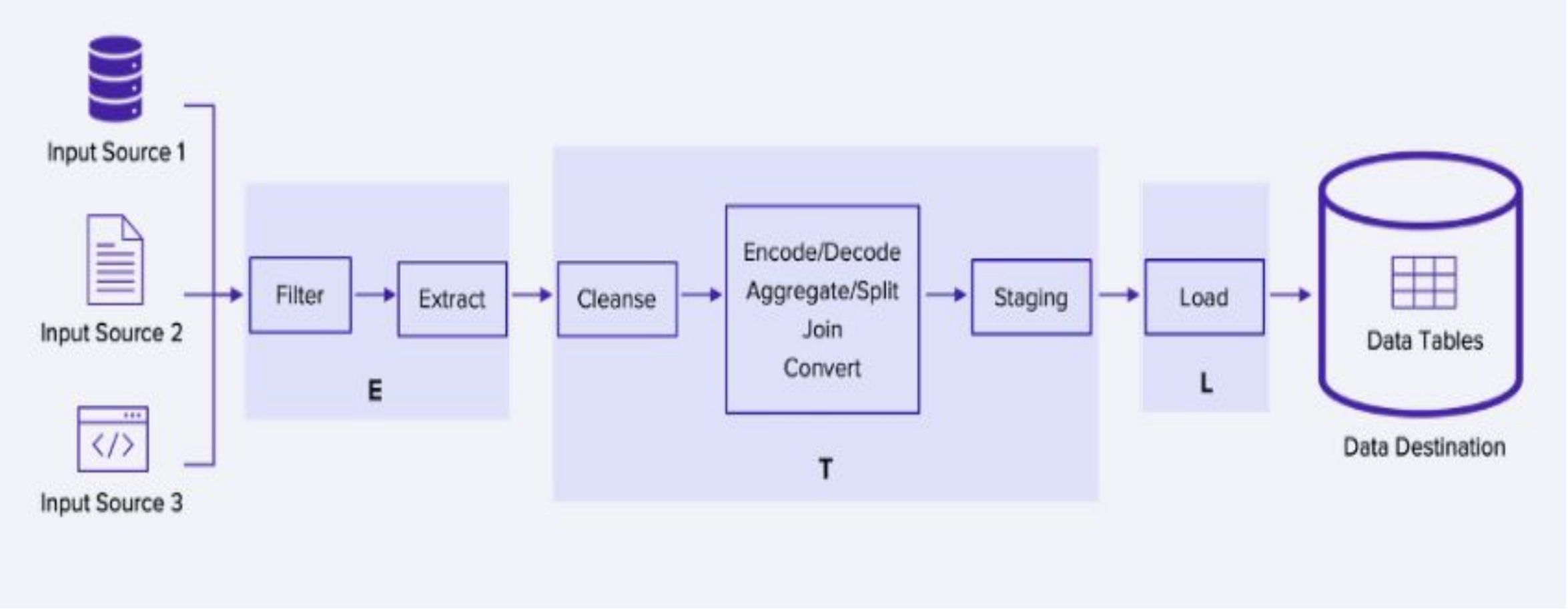
2.2 Modèle ETL

- Processus structuré qui permet de **collecter**, **transformer**, et **charger** des données depuis plusieurs sources vers un système cible, tel qu'un entrepôt de données.
- Utilisé depuis **les années 1970** mais reste une méthode clé pour le traitement des données dans les entreprises.

Contexte historique :

Le modèle ETL est né à une époque où les systèmes de stockage de données étaient limités. Il nécessitait une préparation en amont pour optimiser l'utilisation des ressources (stockage et calcul).

2.2 Modèle ETL



2.2 Modèle ETL

2.2.1 Extraction (E)

- Collecte des données brutes à partir de sources diverses (bases de données relationnelles, fichiers CSV, API, capteurs IoT, etc.).
- L'objectif est de capturer des données structurées (bases de données), semi-structurées (fichiers JSON) et non structurées (documents, images).
- La qualité de l'extraction est essentielle pour garantir l'intégrité des données en aval.

2.2 Modèle ETL

2.2.2 Transformation (T)

- Conversion des données brutes pour répondre aux besoins analytiques ou aux contraintes du système cible.
- Exemples de transformations :
  - Nettoyage des données (suppression des doublons, correction des erreurs).
  - Agrégation (calcul de moyennes, sommes).
  - Changement de format (exemple : conversion de formats de date).
- Cette étape permet également d'appliquer des **règles métier**, comme la segmentation des clients ou la standardisation des unités de mesure.

2.2 Modèle ETL

2.2.3 Chargement (L)

- Importation des données transformées dans un entrepôt de données ou une base cible.
- Peut être réalisé de manière incrémentale (mise à jour partielle) ou en lot (rafraîchissement complet).
- Une fois les données chargées, elles deviennent disponibles pour l’analytique, la création de rapports, et l’aide à la décision.

2.2 Modèle ETL

2.2.4 Avantages

Qualité et cohérence des données : L'étape de transformation garantit que seules des données précises et nettoyées sont chargées dans le système cible.

Meilleure gestion des dépendances : Les processus ETL peuvent inclure des dépendances strictes entre étapes, ce qui évite les erreurs de synchronisation.

Personnalisation avancée : Possibilité d'appliquer des transformations complexes en fonction des besoins métiers spécifiques.

Sécurité accrue : Les données sensibles peuvent être transformées (anonymisation, masquage) avant d'être chargées, réduisant ainsi les risques.

2.2 Modèle ETL

2.2.5 Limites

Temps et complexité : La transformation avant le chargement ralentit le processus global, surtout avec des volumes de données croissants.

Scalabilité limitée : Nécessite des serveurs de traitement intermédiaires, ce qui peut devenir coûteux et difficile à dimensionner.

Rigidité : L'architecture ETL est souvent peu flexible face à des modifications fréquentes des données ou des sources.

2.2 Modèle ETL

2.2.6 Cas d’usage en pratique

Migration de données : Migrer des bases de données existantes vers un nouvel entrepôt de données.

Nettoyage et standardisation : Parfait pour transformer des données provenant de systèmes disparates dans un format homogène.

Préparation pour les analyses récurrentes : Permet de structurer des données pour des tableaux de bord ou des rapports standardisés.

Applications spécifiques (IoT, finance) : Nettoyage et pré-agrégation de données sensibles ou à fréquence élevée avant stockage.

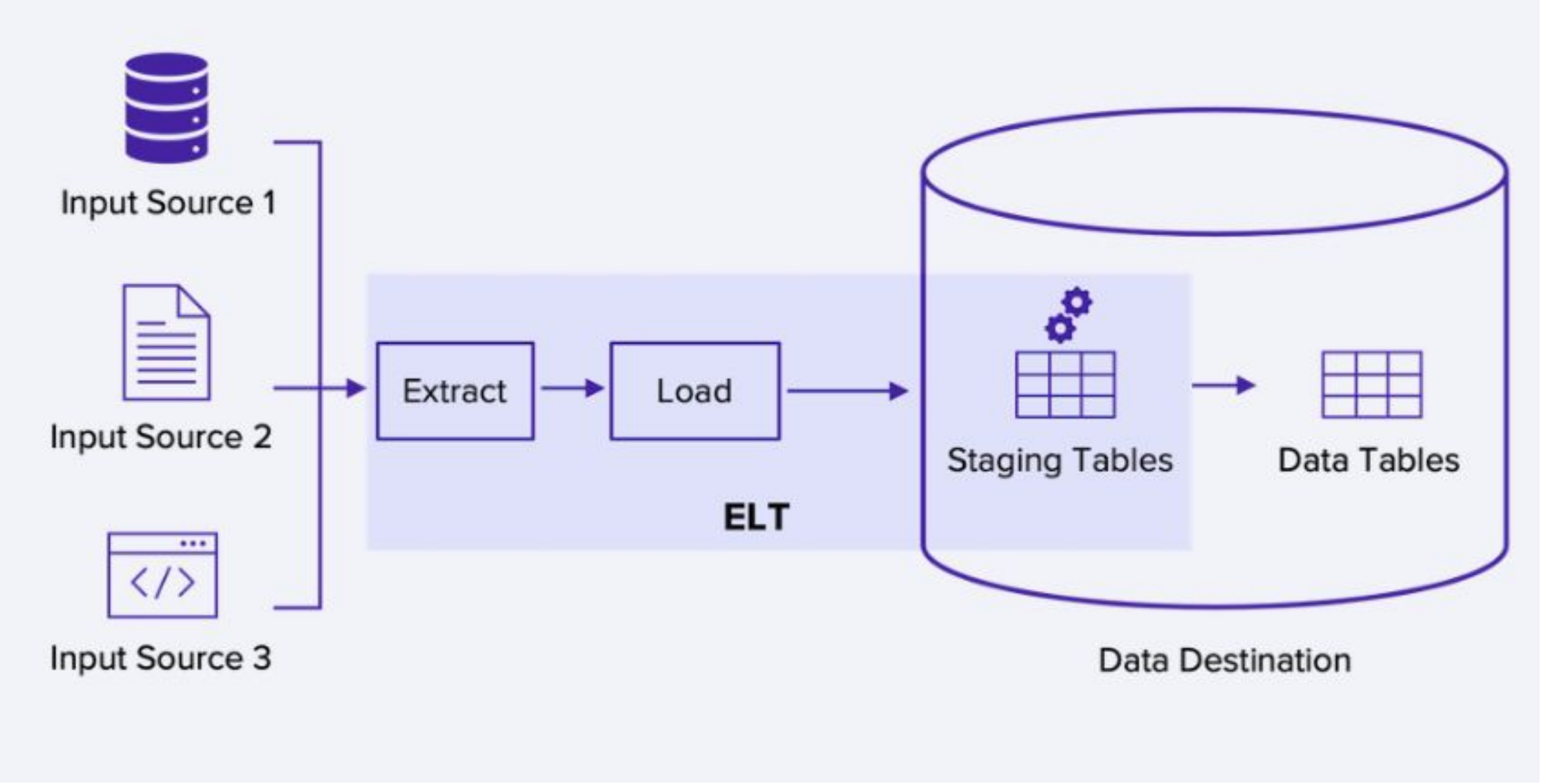
2.3 Modèle ELT

- Approche moderne pour traiter les données, dans laquelle les données brutes sont d'abord chargées dans un système cible (Data Warehouse, Data Lake) puis transformées directement dans cet environnement.
- ELT s'appuie sur les **puissantes capacités de traitement** des systèmes modernes, notamment ceux basés sur le cloud, pour exécuter les transformations.

Contexte historique :

Le modèle ELT est né avec l'émergence des systèmes de stockage massifs dans le cloud (comme Amazon Redshift, Google BigQuery ou Snowflake). Contrairement à ETL, il exploite la puissance de calcul des systèmes cibles pour gérer des volumes massifs de données brutes.

2.3 Modèle ELT



2.3 Modèle ELT

2.3.1 Extraction (E)

- Comme dans ETL, l’extraction consiste à collecter des données à partir de sources diverses (bases de données, API, fichiers, flux IoT).
- L'accent est mis sur la collecte rapide de grandes quantités de données dans leur état brut, sans traitement préalable.

2.3 Modèle ELT

2.3.2 Chargement (L)

- Les données brutes sont directement chargées dans un **lac de données** (data lake) ou un **entrepôt de données** (data warehouse).
- Ce processus peut être réalisé rapidement grâce aux systèmes cloud qui permettent une ingestion massive de données.

2.3 Modèle ELT

2.3.3 Transformation (T)

- Les transformations sont effectuées **dans le système cible** à l'aide de requêtes SQL ou d'outils spécialisés.
- Les données peuvent être transformées plusieurs fois selon les besoins analytiques, permettant une grande flexibilité.
- Exemples de transformation : agrégation, nettoyage, enrichissement, ou application de modèles analytiques avancés.

2.3 Modèle ELT

2.3.4 Avantages

Rapidité : Les données sont chargées immédiatement dans le système cible, sans transformation préalable, ce qui accélère l'ingestion.

Scalabilité : Les systèmes modernes dans le cloud permettent de traiter des volumes massifs de données de manière distribuée et parallèle.

Flexibilité :Les données brutes étant disponibles dans leur intégralité, il est possible de réaliser plusieurs transformations pour répondre à des besoins analytiques divers.
Support des données non structurées : Idéal pour ingérer des types de données variés : images, vidéos, fichiers JSON, logs, etc.

Coûts réduits : Réduction des besoins en infrastructure intermédiaire, puisque les transformations s’appuient sur les capacités internes de l’entrepôt de données.

2.3 Modèle ELT

2.3.5 Limites

Dépendance au cloud : Le modèle ELT repose principalement sur des infrastructures cloud, ce qui peut poser problème pour les entreprises ayant des restrictions de conformité ou des limites budgétaires.

Complexité des requêtes : Les transformations peuvent nécessiter des compétences avancées en SQL ou l'utilisation d'outils spécifiques pour gérer des scénarios complexes.

Gestion des données brutes : L'ingestion directe des données brutes peut augmenter les risques d'erreurs si elles ne sont pas correctement cataloguées ou suivies.

2.3 Modèle ELT

2.3.6 Cas d’usage en pratique

Analytique moderne : Analyse de grandes quantités de données en quasi-temps réel, comme les dashboards de suivi des ventes ou des flux financiers.

Machine Learning : Extraction de données non structurées (images, logs) pour les transformer et les utiliser dans des modèles d’apprentissage automatique.

Applications data-driven : Exploitation des données brutes pour des projets nécessitant plusieurs transformations à la demande (par exemple, détection d'anomalies ou prévisions).

2.4 Comparaison des modèles ETL et ELT

	ETL	**ELT**
Lieu de transformation	Serveur intermédiaire	Entrepôt ou lac de données cible
Compatibilité des données	Données structurées	structuré, semi-structuré, non-structuré
Rapidité	Plus lent car T avant L	Plus rapide
Scalabilité	Limitée, dépend des serveurs intermédiaires	Très scalable grâce aux systèmes distribués dans le cloud
Infrastructure requise	Serveurs dédiés pour l'extraction et la transformation	Infrastructure cloud avancée pour le stockage et le traitement
Flexibilité	Transformation définie avant le chargement, moins flexible	Grande flexibilité, transformation à la demande
Sécurité	Nécessite des mécanismes personnalisés (ex. masquage des données)	Intègre des fonctions de sécurité natives dans l'entrepôt de données
Cas d'usage typiques	Migration de données, analytique complexe, transformation en périphérie	Analytique moderne, machine learning, gestion massive de données brutes

2. Deuxième partie

3. Outils et technologies pour les pipelines de données

3.1 Catégories d’outils pour les pipelines

5 catégories principales associées aux différents usages:

- Outils d’intégration de données (Ingestion)
- Outils de traitement des données (Batch/Streaming).
- Entrepôts et lacs de données (Stockage).
- Orchestration et automatisation.
- Outils de surveillance et qualité des données.

3.1 Catégories d’outils pour les pipelines

3.1.1 Outils d’intégration des données

- **Objectif** : Collecter et consolider les données provenant de multiples sources vers un système cible (entrepôt ou lac de données).
- **Types de sources** : Bases de données relationnelles, fichiers, flux IoT, API SaaS.
- **Exemple de scénario** : Une entreprise collecte des données de ventes issues d'un ERP, d'un CRM, et de capteurs en magasin.

3.1 Catégories d’outils pour les pipelines

3.1.1 Outils d’intégration des données (suite)

- **Technologies principales** :
  - **Apache Kafka** :
    - Système de messagerie distribué pour ingérer des flux de données en temps réel.
    - Cas d’usage : Transmission de logs applicatifs ou gestion de pipelines IoT.
  - **Fivetran** :
    - Automatisation de l'extraction des données depuis des sources prédéfinies vers des entrepôts comme Snowflake.
    - Avantages : Configuration simple, faible maintenance.

3.1 Catégories d’outils pour les pipelines

3.1.2 Outils de traitement des données : traitement par lots

- **Objectif** : Traiter de grandes quantités de données de manière périodique ou planifiée.
- **Exemple de scénario** : Analyse mensuelle des ventes ou préparation quotidienne des données pour un tableau de bord.

3.1 Catégories d’outils pour les pipelines

3.1.2 Outils de traitement des données: traitement par lots (suite)

- **Technologies principales** :
  - **Apache Spark** :
    - Moteur de traitement distribué à hautes performances pour les pipelines ETL.
    - Fonctionnalités : Support des grandes bases de données, scalabilité sur des clusters.
  - **Hadoop** :
    - Historique pour le traitement des données par lots.
    - Cas d’usage : Analyse de gros volumes sur des systèmes distribués.
-

3.1 Catégories d’outils pour les pipelines

3.1.2 Outils de traitement des données: traitement en temps réel

- **Objectif** : Traiter des données à mesure qu'elles arrivent, avec des délais très courts (quasi-instantanés).
- **Exemple de scénario** : Surveillance des fraudes bancaires ou analyse des flux de clics sur un site web.

3.1 Catégories d’outils pour les pipelines

3.1.2 Outils de traitement des données: traitement en temps réel (suite)

- **Technologies principales** :
  - **Apache Flink** :
    - Traitement à faible latence et gestion avancée des états pour des cas complexes.
    - Cas d’usage : Streaming IoT, calcul d’indicateurs en temps réel.
  - **Apache Storm** :
    - Système robuste et simple pour l'analyse des flux de données.
    - Cas d’usage : Surveillance réseaux, flux marketing.

3.1 Catégories d’outils pour les pipelines

3.1.3 Outils traitement des données: dbt

Objectif : Automatiser et organiser les transformations SQL dans les entrepôts de données.
Caractéristiques principales : Versionnement, tests, documentation des modèles.
Cas d’usage : Simplifier et standardiser les transformations SQL dans Snowflake ou BigQuery.

3.1 Catégories d’outils pour les pipelines

Data Warehouse

- **Objectif** : Centraliser et structurer des données pour permettre des analyses complexes et des rapports rapides.
- **Caractéristiques** : Optimisé pour les requêtes SQL analytiques, gestion des données relationnelles.

- **Technologies principales** :
  - **Snowflake** :
    - Solution cloud-native qui s'adapte dynamiquement aux besoins de stockage et de calcul.
  - **Google BigQuery** :
    - Capacités de calcul à grande échelle et prise en charge de requêtes massives.

3.1 Catégories d’outils pour les pipelines

Data lake

- **En quoi consiste un lac de données ?** :
  - **Objectif** : Stocker des données brutes ou semi-structurées en vue de traitements ultérieurs.
  - **Caractéristiques** : Pas de schéma imposé à l’ingestion, idéal pour le Machine Learning ou l’analytique exploratoire.
- **Technologies principales** :
  - **Amazon S3** :
    - Solution de stockage cloud pour des volumes massifs.
  - **Delta Lake** :
    - Couche supérieure pour ajouter des garanties ACID et une meilleure gestion des transactions.

3.1 Catégories d’outils pour les pipelines

Orchestration et automatisation

- **Objectif** : Automatiser et planifier les différentes étapes d’un pipeline de données.
- **Exemple de scénario** : Orchestrer l’extraction quotidienne des données suivie de leur transformation.

- **Technologies principales** :
  - **Dagster** (chapitre 3)
  - **Airflow**
  - **Luigi**

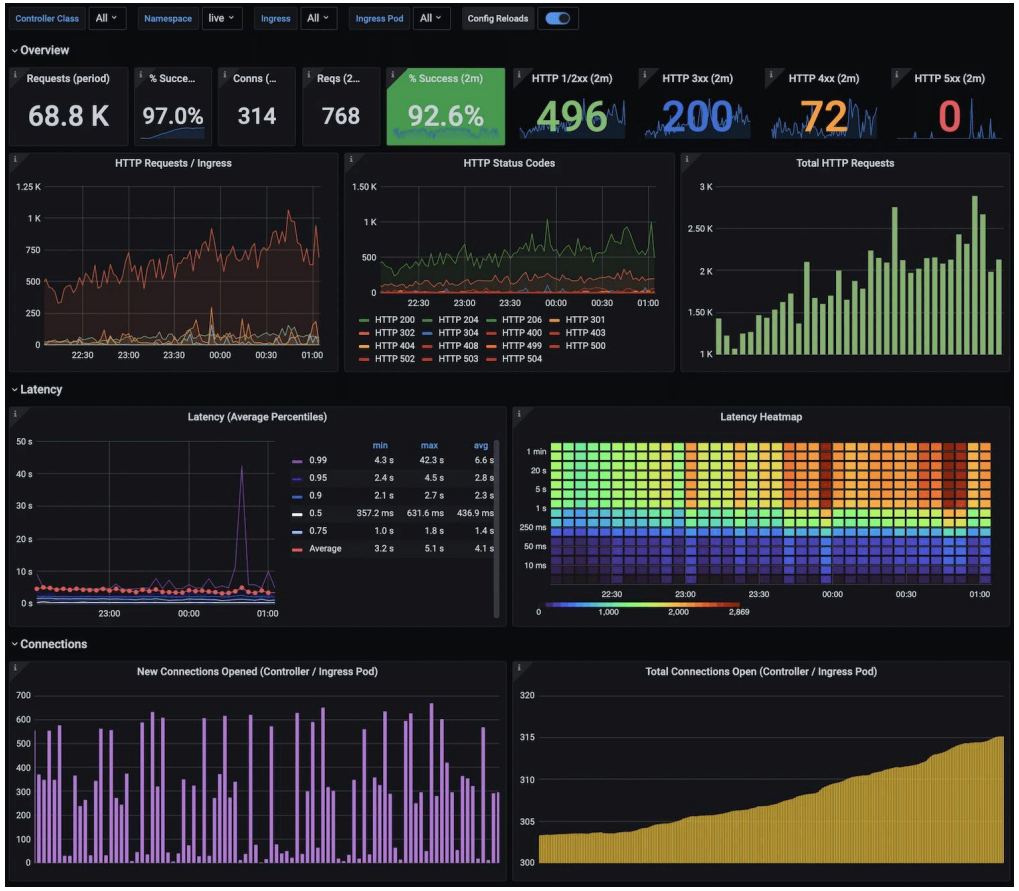
3.1 Catégories d’outils pour les pipelines

Surveillance et qualité des données

- **En quoi consiste cette étape ?** :
  - **Objectif** : Garantir que les pipelines fonctionnent correctement et que les données sont fiables.
  - **Exemple de scénario** : Identifier rapidement un problème de latence dans un pipeline de streaming.
- **Technologies principales** :
  - **Great Expectations** :
    - Tests automatisés pour vérifier la conformité et la qualité des données.
  - **Prometheus + Grafana** :
    - Monitoring et visualisation des performances des pipelines.

3.1 Catégories d’outils pour les pipelines

Surveillance et qualité des données



3.1 Catégorie d’outil pour les pipelines

Comment choisir ses outils

- **Analyse des besoins** :
  - Nature des données : structurées (bases SQL), semi-structurées (JSON, XML), ou non structurées (images, vidéos).
  - Volume des données : petit, moyen ou massivement scalable.
  - Fréquence : traitement par lots ou en temps réel ?
- **Facilité d’intégration** :
  - Les outils doivent bien s’intégrer dans l’infrastructure existante (bases de données, cloud, applications internes).
  - Compatibilité avec les systèmes cloud (AWS, GCP, Azure).

3.1 Catégorie d’outil pour les pipelines

Comment choisir ses outils (suite)

- **Budget et coût total** :
  - Évaluer les coûts initiaux (licences, matériel) et récurrents (hébergement cloud, maintenance).
  - Solutions open-source (Apache Kafka, dbt) vs outils commerciaux (Snowflake)
- **Évolutivité et longévité** :
  - L’outil doit pouvoir évoluer avec la croissance des données et des utilisateurs.
  - Est-ce une technologie bien supportée et maintenue ?
- **Accessibilité et compétences nécessaires** :
  - Évaluer les compétences de l’équipe : maîtrise de Python, SQL, ou besoin de formation ?
  - Préférer des outils avec une bonne documentation et une large communauté.

3.2 Langages essentiels

- **Python** :
  - Langage polyvalent pour automatiser et étendre les pipelines.
  - Bibliothèques clés : pandas (traitement des données), PySpark (big data), requests (API).
  - Cas d’usage : Transformation de données, extraction via API.
- **SQL** :
  - Langage standard pour manipuler les bases de données relationnelles.
  - Cas d’usage : Filtrage, agrégation, jointures sur des données pré-stockées.
- **Complémentarité** : Python pour la flexibilité, SQL pour l’efficacité dans les entrepôts de données.

3.3 Bases de données relationnelles

En quoi consistent-elles ? :

Objectif : Organiser les données sous forme de tables avec des relations définies entre elles.

Caractéristiques principales : ACID, SQL natif pour les manipulations.

Exemples courants :

- **PostgreSQL** : Base open-source puissante et polyvalente.
- **MySQL** : Idéal pour les applications web et les projets de petite à moyenne échelle.

Cas d'usage : Stockage temporaire, préparation des données avant traitement.

3.3 Bases de données NON relationnelles

En quoi consistent-elles ?

Objectif : Stocker des données de manière flexible, sans contrainte de schéma.
Caractéristiques principales : Scalabilité horizontale, absence de schéma fixe, stockage optimisé selon le modèle de données.

Exemples courants :

- **MongoDB** : Base NoSQL orientée documents, idéale pour les applications flexibles et évolutives.
- **Redis** : Base clé-valeur en mémoire, optimisée pour la rapidité et utilisée pour le caching.

3.3 Bases de données NON relationnelles

Cas d’usage :

✅ Stockage de **gros volumes de données non structurées** (logs, réseaux sociaux, IoT).
✅ Applications nécessitant **une lecture/écriture rapide** et évolutive.
✅ **Cache en temps réel** pour améliorer les performances des applications.

3.3 Bases de données: comparatif

Difference between Relational and Non-Relational databases

Relational

student

id	name	surname	age
1	John	Brown	19
2	Emma	Carly	23

Non-Relational

student.json file body:

```
[
  {
    "id": 1,
    "name": "John",
    "surname": "Brown",
    "age": 19
  },
  {
    "id": 2,
    "name": "Emma",
    "surname": "Carly",
    "age": 23
  }
]
```

3. Outils et technologies pour les pipelines de données

4. Cas pratique

4. Cas pratique

Pipeline de données pour un tableau de bord des ventes

Contexte : Une petite entreprise souhaite automatiser la mise à jour quotidienne d'un tableau de bord affichant les chiffres de ventes par région et par produit. Les données proviennent de fichiers CSV exportés depuis un logiciel de gestion commerciale.

Enjeux :

1. Automatiser le processus pour gagner du temps et réduire les erreurs manuelles.
2. Nettoyer les données pour garantir leur qualité et leur cohérence.
3. Mettre en place un pipeline robuste et reproductible.
4. Choisir des outils simples, accessibles au maximum .

4. Cas pratique

Pipeline de données pour un tableau de bord des ventes

- **Choix ETL ou ELT** :

  Nous choisissons ici un modèle **ETL** car :

  - Les fichiers CSV doivent être nettoyés et agrégés avant d'être chargés dans la base de données.
  - Les données sont de petite taille et ne nécessitent pas un entrepôt avancé ni un traitement complexe post-chargement.

4. Cas pratique

Pipeline de données pour un tableau de bord des ventes

Structure du pipeline :

- Extraction : Collecte des fichiers CSV.
- Transformation : Nettoyage, consolidation et agrégation des données.
- Chargement : Insertion des données dans une base de données.
- Visualisation : Création de graphiques et de rapports interactifs.

Organisation du code :

Un fichier principal pipeline.py regroupant toutes les étapes (extraction, transformation, chargement).

Chaque étape divisée en fonctions (extract, transform, load) pour favoriser la clarté, la réutilisabilité et les tests individuels.

4. Cas pratique

Pipeline de données pour un tableau de bord des ventes

Étape 1 : Extraction

- **Objectif** : Collecter les fichiers CSV contenant les données de ventes.
- **Tâches à effectuer** :
  1. Identifier l'emplacement des fichiers (ex. un dossier local ou un espace cloud comme Google Drive).
  2. Lire ces fichiers en utilisant Python.

4. Cas pratique

Pipeline de données pour un tableau de bord des ventes

Étape 1 : Extraction

Pourquoi Python et pandas ? Pandas offre une solution simple et performante pour manipuler des fichiers tabulaires.

Gestion des fichiers : Le code gère plusieurs fichiers automatiquement, évitant les tâches manuelles.

Automatisation :

Configurer une tâche planifiée (ex. avec cron sous Linux ou le Planificateur de tâches sous Windows) pour exécuter ce script quotidiennement.

4. Cas pratique

Pipeline de données pour un tableau de bord des ventes

Étape 1 : Extraction

```python
def extract(csv_folder):
    """Extraction des fichiers CSV."""
    data_frames = []
    for file in os.listdir(csv_folder):
        if file.endswith(".csv"):
            file_path = os.path.join(csv_folder, file)
            df = pd.read_csv(file_path)
            data_frames.append(df)
    return pd.concat(data_frames, ignore_index=True)
```

4. Cas pratique

Pipeline de données pour un tableau de bord des ventes

Étape 2 : Transformation

- **Objectif** : Nettoyer et structurer les données avant leur stockage.
- **Tâches à effectuer** :
  1. Supprimer les doublons.
  2. Gérer les erreurs de format (dates, valeurs manquantes).
  3. Agréger les données par région et par produit.

4. Cas pratique

Pipeline de données pour un tableau de bord des ventes

Étape 2 : Transformation

Pourquoi nettoyer les données ? Assurer leur cohérence et éviter les erreurs dans les analyses futures.

Astuces pour l’automatisation : Le nettoyage doit être générique pour s’adapter à tous les fichiers futurs.

4. Cas pratique

Pipeline de données pour un tableau de bord des ventes

Étape 2 : Transformation

Bonnes pratiques :

i. Valider les données à chaque étape (ex. vérifier les colonnes après transformation).
ii. Documenter les transformations appliquées.

4. Cas pratique

Pipeline de données pour un tableau de bord des ventes

Étape 2 : Transformation

```python
def transform(sales_data):
    """Nettoyage et agrégation des données."""
    sales_data.drop_duplicates(inplace=True)
    sales_data['date'] = pd.to_datetime(sales_data['date'], errors='coerce')
    sales_data = sales_data.dropna(subset=['date'])
    sales_data['sales'] = sales_data['sales'].fillna(0)
    return sales_data.groupby(['region', 'product']).agg({'sales': 'sum'}).reset_index()
```

4. Cas pratique

Pipeline de données pour un tableau de bord des ventes

Étape 3 : Chargement

- **Objectif** : Stocker les données consolidées dans une base de données relationnelle.
- **Tâches à effectuer** :
  1. Créer ou configurer une base de données (ex. SQLite pour ce cas simple).
  2. Charger les données transformées dans une table SQL.

4. Cas pratique

Pipeline de données pour un tableau de bord des ventes

Étape 3 : Chargement

```
def load(aggregated_data, db_path):
    """Chargement des données dans la base SQLite."""
    engine = create_engine(f'sqlite:///{db_path}')
    aggregated_data.to_sql('sales', engine, if_exists='replace', index=False)
```

4. Cas pratique

Pipeline de données pour un tableau de bord des ventes

Étape 3 : Chargement

Pourquoi SQLite ? Une base légère et facile à configurer la phase de tests.

Étendre à d'autres bases : PostgreSQL ou MySQL peuvent remplacer SQLite pour des projets plus complexes (et passage en production)

4. Cas pratique

Pipeline de données pour un tableau de bord des ventes

Étape 4 : Visualisation

- **Objectif** : Générer un tableau de bord interactif pour analyser les données.
- **Tâches à effectuer** :
  - Configurer un outil de visualisation (Google Data Studio, Tableau, ou Power BI).
  - Connecter l'outil à la base SQLite.
  - Créer des graphiques clairs (ex. histogramme des ventes par région).

4. Cas pratique

Pipeline de données pour un tableau de bord des ventes

Étape 4 : Visualisation

- **Pourquoi une visualisation ?** Rendre les données accessibles et exploitables pour les décideurs.
- Pourquoi connecter directement à la base ? Pour permettre une mise à jour automatique du tableau de bord dès que les données changent.
- **Conseils pratiques** : Préférer des graphiques simples et pertinents (ex. barres, camemberts).

- **Automatisation** :
  - Configurer une actualisation automatique des graphiques en connectant directement l'outil à la base.

4. Cas pratique

Pipeline de données pour un tableau de bord des ventes

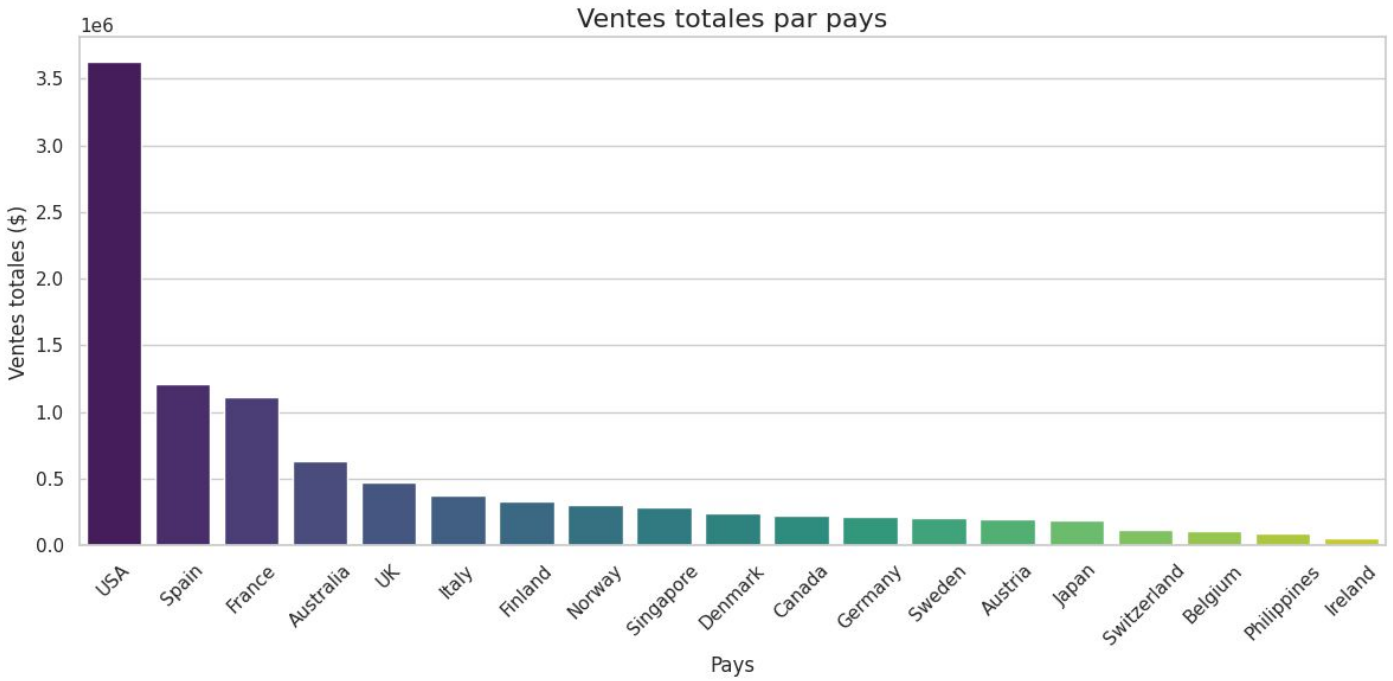
Étape 4 : Visualisation

```
# Graphique 1 : Ventes totales par pays
plt.figure(figsize=(12, 6))
country_sales = sales_data.groupby("COUNTRY")["SALES"].sum().sort_values(ascending=False)
sns.barplot(x=country_sales.index, y=country_sales.values, palette="viridis")
plt.title("Ventes totales par pays", fontsize=16)
plt.xlabel("Pays", fontsize=12)
plt.ylabel("Ventes totales ($)", fontsize=12)
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig("./data/plots/country_sales.png")  # Sauvegarder le graphique
plt.show()
```

4. Cas pratique

Pipeline de données pour un tableau de bord des ventes

Étape 4 : Visualisation



4. Cas pratique

Pipeline de données pour un tableau de bord des ventes

Synthèse

- **Technologies utilisées** : Python, pandas, SQLite, outil de visualisation.
- **Workflow complet** : Extraction → Transformation → Chargement → Visualisation.

Organisation du code :

1. Modulariser chaque étape avec des fonctions (extract, transform, load).
2. ICI, on centralise tout dans un fichier principal (pipeline.py), à éviter par la suite !

4. Cas pratique

Pipeline de données pour un tableau de bord des ventes

Synthèse

- **Bonnes pratiques générales** :
  - Tester chaque fonction indépendamment avant d'automatiser.
  - Documenter chaque décision technique et transformation.
  - Mettre en place une surveillance pour détecter rapidement les erreurs.
- **Conseils pour aller plus loin** :
  - Étendre ce pipeline en utilisant des bases de données plus avancées.
  - Ajouter des alertes en cas de données manquantes ou d'erreurs lors du chargement.

4. Cas pratique

Pipeline de données pour un tableau de bord des ventes

Une démo ?

4. Cas pratique

Pipeline d’analyse multi-sources des données d’un e-commerce

Contexte

Une entreprise de e-commerce souhaite centraliser et analyser ses données provenant de différentes sources pour optimiser ses performances marketing et logistiques. Les sources incluent :

- Les données de commandes provenant d’une base MySQL (système ERP).
- Les données des campagnes publicitaires provenant d’une API (ex. Facebook Ads).
- Les retours clients (avis, commentaires) extraits de fichiers CSV exportés manuellement chaque jour.

4. Cas pratique

Pipeline d’analyse multi-sources des données d’un e-commerce

Objectifs

1. **Centraliser les données** : Intégrer les données hétérogènes dans un **data warehouse** pour une analyse unifiée.
2. **Exploration des données brutes** : Fournir un accès flexible aux données non structurées pour les analystes.
3. **Transformation à la demande** : Appliquer des transformations adaptées aux cas d’usage spécifiques directement dans le data warehouse.
4. **Automatisation** : Mettre en place un pipeline robuste, automatisé et surveillé pour éviter les erreurs manuelles.

4. Cas pratique

Pipeline d’analyse multi-sources des données d’un e-commerce

Choix technologiques et justification

Modèle choisi : ELT

- **Pourquoi ELT ?**
  - Les données brutes doivent être centralisées rapidement dans un **data warehouse** sans transformations complexes initiales.
  - Le volume important de données publicitaires nécessite une capacité d’analyse directe dans le data warehouse.
  - Les transformations sont spécifiques aux cas d’usage, donc elles seront appliquées à la demande par les analystes ou les outils BI.

4. Cas pratique

Pipeline d’analyse multi-sources des données d’un e-commerce

Choix technologiques et justification

Technologies utilisées

Extraction des données :

- **MySQL** pour les commandes du e-commerce (base relationnelle).
- **API REST** (exemple : Facebook Ads) pour les données publicitaires.
- **Fichiers CSV** pour les retours clients.
- Extraction automatisée avec **Python** et **Dagster** pour l’orchestration.

4. Cas pratique

Pipeline d’analyse multi-sources des données d’un e-commerce

Choix technologiques et justification

Technologies utilisées

Chargement des données :

- **Google BigQuery** comme entrepôt de données :
  - Avantages : Scalabilité, flexibilité, et transformation SQL native.
- Chargement via **dbt** (Data Build Tool).

4. Cas pratique

Pipeline d’analyse multi-sources des données d’un e-commerce

Choix technologiques et justification

Technologies utilisées

Transformation :

- **dbt** pour appliquer des transformations SQL dans BigQuery.
- **Jupyter Notebooks** pour les analyses exploratoires des données non structurées.

4. Cas pratique

Pipeline d’analyse multi-sources des données d’un e-commerce

Choix technologiques et justification

Technologies utilisées

Surveillance et monitoring :

- **Apache Airflow** pour orchestrer le pipeline et surveiller les tâches.
- **Prometheus** et **Grafana** pour le monitoring des performances.

4. Cas pratique

Pipeline d’analyse multi-sources des données d’un e-commerce

Enjeux et contraintes

Volume important de données publicitaires :

- Les campagnes publicitaires génèrent des millions d’impressions et de clics, nécessitant un système capable de gérer un grand volume de données.

Hétérogénéité des données :

- Données relationnelles (MySQL), semi-structurées (API) et non structurées (avis clients en CSV).

4. Cas pratique

Pipeline d’analyse multi-sources des données d’un e-commerce

Enjeux et contraintes

Automatisation robuste :

- Le pipeline doit être fiable et surveillé pour garantir la qualité des données.

Flexibilité analytique :

- Permettre aux analystes d’explorer et de transformer les données sans dépendre des ingénieurs.

4. Cas pratique

Pipeline d’analyse multi-sources des données d’un e-commerce

Chargement dans BigQuery

pourquoi BigQuery :

- Gestion facile de grandes quantités de données.
- Support natif pour les transformations SQL à grande échelle.
- Intégration fluide avec dbt.

4. Cas pratique

Pipeline d’analyse multi-sources des données d’un e-commerce

Transformation dans BigQuery

Exemples de transformations avec dbt :

Nettoyage des données :

- Suppression des doublons.
- Gestion des valeurs manquantes (ex. remplacements ou exclusions).
- Conversion des colonnes de date au format standard ISO.

4. Cas pratique

Pipeline d’analyse multi-sources des données d’un e-commerce

Transformation dans BigQuery

Exemples de transformations avec dbt :

Enrichissement des données :

- Jointure des données publicitaires et des commandes pour calculer le ROI par campagne.
- Extraction des sentiments à partir des commentaires clients (analyse NLP via BigQuery ML).

2. **Agrégation** :
   - Calcul des revenus par région, produit et canal marketing.
   - Groupement par semaines pour les analyses temporelles.

4. Cas pratique

Pipeline d’analyse multi-sources des données d’un e-commerce

Transformation dans BigQuery

Exemples de transformations avec dbt :

Agrégation :

- Calcul des revenus par région, produit et canal marketing.
- Groupement par semaines pour les analyses temporelles.

4. Cas pratique

Pipeline d’analyse multi-sources des données d’un e-commerce

Visualisation et exploration des données

Outils BI :

- Tableau ou Google Data Studio pour des tableaux de bord interactifs.

Analyse exploratoire :

- Utilisation de Jupyter Notebooks connectés à BigQuery pour explorer les données brutes et transformées.

4. Cas pratique

Pipeline d’analyse multi-sources des données d’un e-commerce

Pipeline Dagster / Airflow

DAG principal :

- Tâches : Extraction (x3), chargement, transformation.
- Monitoring des erreurs et des latences.

Tâches spécifiques :

- Extraction : Appels API et scripts de connexion MySQL.
- Chargement : Envoi des fichiers vers BigQuery via dbt.
- Transformation : dbt exécute les transformations SQL planifiées.

4. Cas pratique