l'école d'ingénierie informatique

Rapport

TP Conception-Pipeline

Étudiants:

Deboub Maher

maher.deboub@ecoles.epsi.net

Naghmouchi Med Aziz

m.naghmouchi@ecoles-epsi.net

Intervenant

Marie Amina

23/05/2025

Explication du code ETL

Ce script met en place un **pipeline ETL (Extract - Transform - Load)** complet pour nettoyer et enrichir un fichier CSV de ventes.

Importations

```
import pandas as pd
import numpy as np
import logging
from sklearn.preprocessing import MinMaxScaler
```

- `pandas` : gestion des données tabulaires.
- `numpy` : opérations numériques.
- `logging` : journalisation des événements dans un fichier log.
- `MinMaxScaler` : normalisation des valeurs numériques entre 0 et 1.

Configuration des logs

```
logging.basicConfig(filename='etl.log', level=logging.INFO,
format='%(asctime)s - %(levelname)s - %(message)s')
```

- Configure un fichier de log nommé `etl.log` pour suivre les étapes de traitement.

Étape 1 : Chargement des données

```
def charger_donnees(fichier_csv):
```

- Charge le fichier CSV.
- Vérifie la présence des colonnes obligatoires.
- Gère les erreurs : fichier manquant, format invalide, parsing…

Erreurs gérées :

- `FileNotFoundError`
- `pd.errors.ParserError`
- Colonnes manquantes
- Toute autre exception

Étape 2 : Analyse exploratoire (EDA)

```
def analyse_donnees(df):
```

- Affiche un aperçu des données :
    - `head()`, `info()`, `isnull()`, `describe()` pour comprendre la structure.

Utilisé pour valider manuellement les données avant traitement.

Étape 3 : Traitement des valeurs manquantes

```
def traitement_valeurs_manquantes(df):
```

- Supprime les lignes sans valeurs dans les colonnes critiques (`dropna(subset=...)`).
- Remplir les quantités manquantes avec `0`.

- Remplir les prix manquants avec la **médiane**.

But : éviter les erreurs dans les calculs ultérieurs.

Étape 4 : Gestion des valeurs aberrantes

```
def gestion_valeurs_aberrantes(df):
```

- Supprime les lignes où `Quantite_vendue <= 0` ou `Prix_unitaire <= 0`
- Convertit `Date_vente` en format `datetime`, supprime les dates invalides (`NaT`)

Étape 5 : Suppression des doublons

```
def suppression_doublons(df):
```

- Supprime les lignes identiques pour éviter de fausser les statistiques.

Étape 6 : Transformations

```
def transformations(df):
```

1. Montant total de la vente

```
df['Montant_total'] = df['Quantite_vendue'] * df['Prix_unitaire']
```

2. Normalisation du prix unitaire

```
 scaler = MinMaxScaler()
    df['Prix_unitaire_normalisé'] =
scaler.fit_transform(df[['Prix_unitaire']]).round(3)
```

- Ramène toutes les valeurs entre 0 et 1 (arrondi à 3 décimales).

3. Extraction mois/année

```
df['Mois_vente'] = df['Date_vente'].dt.month
df['Annee_vente'] = df['Date_vente'].dt.year
```

- Pour analyse temporelle.

4. Nettoyage du nom produit

```
df['Nom_produit'] = df['Nom_produit'].str.strip().str.lower()
```

- Nettoie les espaces, harmonise la casse.

5. Catégorisation des montants

```
df['Categorie_montant'] = pd.cut(
        df['Montant_total'],
        bins=[-1, 50, 200, np.inf],
        labels=['faible', 'moyen', 'élevé']
    )
```

- Classe les montants en 3 niveaux.

6. Catégorisation des produits

```
def categorie_produit(nom):
    ...
df['Categorie_produit'] = df['Nom_produit'].apply(categorie_produit)
```

- Classe les produits en `habillement`, `électronique`, `autre` selon des mots-clés.

Étape 7 : Validation croisée

```
def validation_croisée(df):
```

- Vérifie que :
    - Aucune valeur manquante
    - Quantité > 0
    - `Date_vente` est bien de type `datetime`

Étape 8 : Sauvegarde des données

```
def sauvegarde(df, chemin_sortie):
```

- Sauvegarde au format CSV avec l'encodage `utf-8-sig` (utile pour bien gérer les accents dans Excel).

Fonction principale : `pipeline_etl`

```
def pipeline_etl(fichier_entree, fichier_sortie):
```

- Exécute toutes les étapes ci-dessus dans l'ordre logique.
- Journalise chaque étape.
- Gère les erreurs globales avec un bloc `try/except`.

Lancement du script

```
if __name__ == "__main__":
    pipeline_etl("ventes.csv", "vente_clean.csv")
```

- Point d'entrée du programme.

Conclusion

Ce pipeline :

- Nettoie les données
- Ajoute des colonnes utiles
- Catégorise et normalise
- Valide la qualité des données
- Gère toutes les erreurs critiques
- Documente tout dans un fichier `etl.log`

Explication du code PySpark Streaming CSV

Contexte

L'objectif est de configurer un pipeline de traitement **en streaming** de fichiers CSV, qui :

- Lit en continu les fichiers déposés dans un dossier
- Définit un schéma clair des données
- Nettoie et transforme les données
- Réalise une analyse temps réel (exemple : calcul du CA cumulé)
- Sauvegarde les résultats dans des fichiers partitionnés par date
- Gère les erreurs lors du traitement

---

1. Création de la session Spark

```
spark = SparkSession.builder \

    .appName("ETL Streaming CSV") \

    .getOrCreate()



# Pour afficher les logs Spark moins verbeux

spark.sparkContext.setLogLevel("WARN")
```

- **SparkSession** : point d'entrée principal pour utiliser Spark (mode batch ou streaming).
- `appName` donne un nom à l'application.

- On réduit le niveau des logs pour éviter de polluer la console avec des messages inutiles.

---

2. Définition du schéma

```
schema = StructType([
    StructField("ID_produit", IntegerType(), True),
    StructField("Nom_produit", StringType(), True),
    StructField("Quantite_vendue", IntegerType(), True),
    StructField("Prix_unitaire", DoubleType(), True),
    StructField("Date_vente", StringType(), True)  # On va la convertir en timestamp plus tard
])
```

- On crée un **schéma explicite** qui décrit la structure des données attendues.
- Cela évite les erreurs de type et accélère la lecture.
- Ici, les colonnes correspondent au contenu du CSV fourni.

---

3. Lecture en streaming des fichiers CSV

```
df_stream = spark.readStream \
    .option("header", True) \
    .schema(schema) \
    .csv(input_path)
```

- `readStream` indique une lecture en streaming (flux continu).
- Spark surveille le dossier `input_path` et lit tout nouveau fichier CSV.

- `option("header", True)` signifie que la première ligne du CSV contient les noms des colonnes.
- Le schéma défini est appliqué à la lecture.

---

4. Nettoyage et transformation des données

```
from pyspark.sql.functions import to_timestamp

# Conversion de la date en timestamp
df_clean = df_stream.withColumn("Date_vente",
to_timestamp(col("Date_vente"), "yyyy-MM-dd HH:mm:ss"))

# Suppression des valeurs aberrantes
df_clean = df_clean.filter(
    (col("Quantite_vendue") > 0) &
    (col("Prix_unitaire") > 0) &
    (col("Date_vente").isNotNull())
)

# Nettoyage du nom du produit : minuscules et trim
df_clean = df_clean.withColumn("Nom_produit",
trim(lower(col("Nom_produit"))))

# Calcul du montant total
df_clean = df_clean.withColumn("Montant_total",
col("Quantite_vendue") * col("Prix_unitaire"))

# Ajout des colonnes mois et année
df_clean = df_clean.withColumn("Mois_vente",
month(col("Date_vente")))
df_clean = df_clean.withColumn("Annee_vente",
year(col("Date_vente")))

# Catégorisation du montant total
df_clean = df_clean.withColumn(
    "Categorie_montant",
```

```
    when(col("Montant_total") <= 50, "faible")
    .when((col("Montant_total") > 50) & (col("Montant_total") <= 200), "moyen")
    .otherwise("élevé")
)

# Catégorisation du type de produit selon le nom
df_clean = df_clean.withColumn(
    "Categorie_produit",
    when(col("Nom_produit").rlike("chemise|pantalon|veste"), "habillement")
    .when(col("Nom_produit").rlike("ordinateur|écran"), "électronique")
    .otherwise("autre")
)
```

- **Conversion de la date** : transformation de la colonne `Date_vente` en type timestamp.
- **Filtrage des lignes invalides** : on enlève les lignes avec quantité ou prix négatifs, ou date nulle.
- **Nettoyage du texte** : mise en minuscules et suppression des espaces inutiles sur le nom du produit.
- **Création de nouvelles colonnes** :
    - `Montant_total` = Quantité × Prix unitaire
    - `Mois_vente` et `Annee_vente` extraits de la date
    - Catégories pour le montant total (faible, moyen, élevé)
    - Catégories pour le type de produit (habillement, électronique, autre)

---

5. Analyse temps réel

```
current_year =
df_clean.select(year(current_date())).collect()[0][0]

df_ca_annee = df_clean.filter(col("Annee_vente") == current_year)
\
    .groupBy("Annee_vente") \
    .agg(_sum("Montant_total").alias("CA_annee_cumule"))

```

- On récupère l'année courante.
- On filtre les ventes de cette année.
- On calcule le chiffre d'affaires cumulé (`Montant_total` agrégé par somme).
- Cette étape permet un suivi en temps réel du CA annuel.

---

6. Écriture des données transformées

```
query = df_clean.writeStream \
    .outputMode("append") \
    .format("parquet") \
    .option("checkpointLocation", "chemin/vers/checkpoint") \
    .option("path", output_path) \
    .partitionBy("Annee_vente", "Mois_vente") \
    .start()
```

- On écrit en mode streaming les données nettoyées dans un dossier de sortie.
- Le format utilisé est **csv**
- On partitionne les fichiers par année et mois, ce qui facilite l'archivage et l'accès ciblé.

- Le `checkpointLocation` est obligatoire en streaming pour gérer la tolérance aux pannes (enregistrer l'état du streaming).

---

7. Affichage du CA cumulé en console

```
query_ca = df_ca_annee.writeStream \
    .outputMode("complete") \
    .format("console") \
    .start()

```

- Affiche en continu dans la console le CA cumulé.
- Utile pour monitorer en temps réel directement dans le terminal.

---

8. Gestion des erreurs

```
try:
    query.awaitTermination()   # Attend la fin du streaming (CTRL+C pour interrompre)
    query_ca.awaitTermination()
except Exception as e:
    print("Erreur dans le streaming :", e)
    spark.stop()
```

- La fonction `awaitTermination()` permet d'attendre la fin du flux (qui est en principe infini).
- En cas d'erreur, on la capture et on stoppe proprement la session Spark.

Conclusion

Ce pipeline PySpark en streaming permet de traiter **en temps réel** des fichiers CSV déposés dans un dossier.

Il réalise :

- Une lecture et un schéma précis,
- Un nettoyage rigoureux,
- Des transformations métiers (calculs, catégorisation),
- Une analyse du CA en temps réel,
- Une sauvegarde optimisée des données transformées,
- Une gestion simple des erreurs.
