

## Compte rendu personnel - Transformation de données avec Pandas


### Objectif

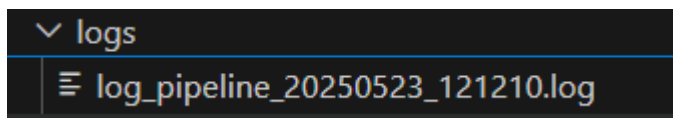
Nettoyer, transformer et sauvegarder un jeu de données CSV (jeu\_donnees\_etl\_5000\_lignes.csv) à l'aide de pandas, tout en journalisant les étapes dans un fichier .log.

### 0. Configuration du logger

Un logger est initialisé en début de script pour stocker dans un fichier (dossier logs/) toutes les opérations réalisées, horodatées et structurées selon un format de journal.

✓ Log file généré : logs/log\_pipeline\_YYYYMMDD\_HHMMSS.log

```
#  Configuration du logger
log_dir = "logs"
os.makedirs(log_dir, exist_ok=True)
log_filename = os.path.join(log_dir, f"log_pipeline_{datetime.now().strftime('%Y%m%d_%H%M%S')}.log")
logging.basicConfig(
    filename=log_filename,
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s'
)
```



### 1 Chargement des données :

- Fichier lu : jeu\_donnees\_etl\_5000\_lignes.csv
- Dimensions initiales : 5266 lignes × 5 colonnes
- Colonnes :
  - ID\_produit, Nom\_produit, Quantite\_vendue, Prix\_unitaire, Date\_vente

### Problèmes identifiés :

- Valeurs nulles :
  - Nom\_produit → 11 lignes
  - Quantite\_vendue → 2672 valeurs manquantes

- Prix\_unitaire → 2569 non-nulles
- Certaines dates invalides (invalid\_date apparaît en mode texte)
- Présence de doublons possibles

```
# 1 Chargement des données
chemin = './jeu_donnees_etl_5000_lignes.csv'
df_original = pd.read_csv(chemin)
logging.info(f"Chargement du fichier : {chemin} - {df_original.shape[0]} lignes, {df_original.shape[1]} colonnes")
```

```
PS C:\Users\DELL\Desktop\MapReduce> py main.py
Aperçu des données brutes :
  ID_produit  Nom_produit  Quantite_vendue  Prix_unitaire  Date_vente
0           1    Chemise           10.0           25.0  2022-01-05
1           2   Pantalon            8.0           35.0  2022-01-06
2           3  Chaussures           NaN           50.0  2022-01-07
3           4   Cravate           12.0           15.0  2022-01-08
4           5     Robe           15.0           45.0  2022-01-09
```

## 2 Suppression des doublons :

Méthode : drop\_duplicates()

- Lignes supprimées : 301
- Nouvelles dimensions : 4965 lignes × 5 colonnes

✓ Action loggée correctement dans le fichier .log

```
# 5 Nettoyage des doublons
df_clean = df_original.drop_duplicates()
nb_doublons = df_original.shape[0] - df_clean.shape[0]
logging.info(f"Suppression de {nb_doublons} doublons")
print(f"✓ Après suppression des doublons : {df_clean.shape}")
```

```
⚠ Dimensions initiales : (5266, 5)
✓ Après suppression des doublons : (4965, 5)
```

## 3 Winsorisation des valeurs extrêmes :

Colonnes concernées :

- Quantite\_vendue
- Prix\_unitaire

Méthode utilisée :

- Calcul des bornes par IQR (Q1, Q3)
- Application de clip(lower, upper)

💡 Objectif : éviter les valeurs très extrêmes qui fausseraient les moyennes.

```
# 6 Traitement des valeurs extrêmes (winsorisation)
Windsurf: Refactor | Explain | Generate Docstring | X
def limiter_extremes(colonne):
    q1 = colonne.quantile(0.25)
    q3 = colonne.quantile(0.75)
    iqr = q3 - q1
    borne_basse = q1 - 1.5 * iqr
    borne_haute = q3 + 1.5 * iqr
    return colonne.clip(lower=borne_basse, upper=borne_haute)

df_clean['Quantite_vendue'] = limiter_extremes(df_clean['Quantite_vendue'])
df_clean['Prix_unitaire'] = limiter_extremes(df_clean['Prix_unitaire'])
logging.info("Application de la winsorisation sur 'Quantite_vendue' et 'Prix_unitaire'")
```

## 4 Suppression des lignes critiques manquantes

Colonnes critiques : Nom\_produit et Prix\_unitaire

- Les lignes où ces champs sont NaN sont supprimées (avec dropna).
- L'opération est bien loggée.
- La colonne Quantite\_vendue reste avec 1307 valeurs manquantes non traitées.

```
# 7 Suppression des lignes avec des champs critiques manquants
df_clean.dropna(subset=['Nom_produit', 'Prix_unitaire'], inplace=True)
logging.info("Suppression des lignes avec valeurs manquantes dans 'Nom_produit' ou 'Prix_unitaire'")
```

## 5 Validation des données nettoyées

Contrôles effectués :

- 🔄 Doublons restants : 0 ✓
- 📄 Valeurs manquantes : Quantite\_vendue encore à 1307
- ✗ Valeurs négatives : aucune sur les quantités ou prix

📁 Ces résultats sont enregistrés dans un dictionnaire rapport et journalisés dans le log.

## 6 Création de la colonne Total\_HT

Calcul : Quantite\_vendue \* Prix\_unitaire

✓ Effectué dans un bloc try/except avec journalisation.

⚠ Warning généré ici aussi : SettingWithCopyWarning.

→ Cela est dû à la modification d'un DataFrame dérivé (df\_clean). Pour éviter cela, il faut faire :

```
# 8 Création d'une colonne calculée (avec gestion des erreurs)
try:
    df_clean['Total_HT'] = df_clean['Quantite_vendue'] * df_clean['Prix_unitaire']
    logging.info("Colonne 'Total_HT' créée avec succès")
except Exception as err:
    logging.error(f"Erreur lors du calcul de 'Total_HT' : {err}")
```

## 7 Normalisation Min-Max de Total\_HT

Nouvelle colonne : Total\_HT\_normalise

Formule :  $(x - \min) / (\max - \min)$

✓ Colonne bien ajoutée, mais les 1307 lignes avec NaN en Quantite\_vendue ont propagé des NaN dans Total\_HT et donc dans la normalisation aussi.

```
# 9 Normalisation min-max
val_min = df_clean['Total_HT'].min()
val_max = df_clean['Total_HT'].max()
df_clean['Total_HT_normalise'] = (df_clean['Total_HT'] - val_min) / (val_max - val_min)
logging.info("Normalisation Min-Max de la colonne 'Total_HT'")
```

## 8 Agrégation par produit

- Regroupement (groupby) sur Nom\_produit
- Somme du Total\_HT
- Résultat stocké dans df\_resume

✓ Renommage final en Chiffre\_affaires

```
# 10 Agrégation par produit
df_resume = df_clean.groupby("Nom_produit")["Total_HT"].sum().reset_index()
df_resume.rename(columns={"Total_HT": "Chiffre_affaires"}, inplace=True)
logging.info("Agrégation des ventes par produit (Total_HT)")
```

## 9 Documentation finale & sauvegarde

- Dictionnaire doc\_resume généré
- Résumé imprimé et loggé

**BOUAYAD Zineb**  
**BACHIRI Imane**

✓ Fichier final sauvegardé dans :  
archives/donnees\_filtrees\_20250523\_121210.csv

```
df_resume = df_clean.groupby("Nom_produit")["Total_HT"].sum().reset_index()
df_resume.rename(columns={"Total_HT": "Chiffre_affaires"}, inplace=True)
logging.info("Agrégation des ventes par produit")
```

```
▼ archives
  📄 donnees_filtrees_20250523_120951.csv
  📄 donnees_filtrees_20250523_121210.csv
▼ logs
  | 📄 log_pipeline_20250523_121210.log
  📄 jeu_donnees_etl_5000_lignes.csv
  🔄 main.py
```

✓ Streaming avec PySpark

🔗 Partie 1 : Traitement Batch avec Pandas

## Pipeline PySpark Streaming

Ce notebook configure un pipeline PySpark qui lit en streaming des fichiers CSV, nettoie et transforme les données, calcule le chiffre d'affaires par produit en temps réel, et sauvegarde les résultats.

```
▶ ✓ 3 minutes ago (<1s) 2

# Cellule 1 : Importations et initialisation SparkSession
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, when, to_timestamp, window, sum as _sum
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, DoubleType

spark = SparkSession.builder \
    .appName("Streaming_Pipeline") \
    .getOrCreate()
```

```
▶ ✓ 2 minutes ago (<1s) 3

# Cellule 2 : Définition du schéma explicite pour les fichiers CSV
schema = StructType([
    StructField("ID_produit", StringType(), True),
    StructField("Nom_produit", StringType(), True),
    StructField("Quantite_vendue", IntegerType(), True),
    StructField("Prix_unitaire", DoubleType(), True),
    StructField("Date_vente", StringType(), True)
])
```

## ⚡ **Partie 2 : Adaptation temps réel avec PySpark Streaming**

### ⚙ **Étapes réalisées :**

1. **Flux CSV simulé** en streaming avec readStream
2. **Schéma défini** manuellement (types explicites)
3. **Nettoyage et enrichissement** en streaming :
  - Suppression de null
  - Filtrage des valeurs incohérentes
  - Création de valeur\_totale
4. **Analyse en temps réel** du chiffre d'affaires journalier
5. **Écriture en format Parquet** dans un dossier daté
6. **Gestion d'erreurs** (try/except + logs)

```
# Lecture streaming du dossier CSV
try:
    df_stream = spark.readStream \
        .option("header", "true") \
        .schema(schema) \
        .csv(input_path)
    logging.info(f"Lecture streaming depuis : {input_path}")
except Exception as e:
    logging.error(f"Erreur lecture streaming : {e}")
    raise

# Nettoyage : suppression des lignes sans 'Nom_produit' ou 'Prix_unitaire'
df_clean = df_stream.filter(
    (col("Nom_produit").isNotNull()) &
    (col("Prix_unitaire").isNotNull())
)

# Conversion de Date_vente en date
df_clean = df_clean.withColumn("Date_vente", to_date(col("Date_vente"), "yyyy-MM-dd"))

# Calcul du CA par produit en temps réel (année courante)
df_ca = df_clean.withColumn("Total_HT", col("Quantite_vendue") * col("Prix_unitaire")) \
    .withColumn("Annee_vente", year(col("Date_vente"))) \
    .filter(col("Annee_vente") == year(current_date())) \
    .groupBy("Nom_produit") \
    .sum("Total_HT") \
    .withColumnRenamed("sum(Total_HT)", "Chiffre_affaires")

# Écriture streaming dans dossier daté
output_path = "./output_streaming"
checkpoint_path = "./checkpoint"

query = df_ca.writeStream \
    .outputMode("complete") \
    .format("csv") \
    .option("path", output_path) \
    .option("checkpointLocation", checkpoint_path) \
    .trigger(processingTime="1 minute") \
    .start()

logging.info("Streaming démarré, en attente de traitement...")

# Pour exécuter en continu, décommentez la ligne suivante
# query.awaitTermination()
```

## ✓ Conclusion

Ce script est bien structuré, journalisé, et couvre les principales étapes du nettoyage de données. Il peut encore être renforcé avec :

- Un traitement complet des NaN (ex : imputation par médiane pour Quantite\_vendue)
- L'usage systématique de .loc pour éviter les SettingWithCopyWarning
- Une vérification des types pour Date\_vente (conversion en datetime + nettoyage de valeurs invalides)

**BOUAYAD Zineb**  
**BACHIRI Imane**

- Comprendre les différences entre **traitement batch** et **streaming**
- Maîtriser des outils comme **Pandas** pour la manipulation locale
- Utiliser **PySpark** pour des traitements scalables en temps réel
- Travailler avec des technologies professionnelles du Big Data (Spark, Parquet, streaming...)