

TP Streaming PySpark – Exercice : T pour Transform

1. Introduction


Ce TP a pour objectif de mettre en œuvre un pipeline de traitement de données en streaming à l'aide de PySpark. Il s'inscrit dans le cadre d'un projet pédagogique visant à maîtriser les concepts fondamentaux de l'ingestion, transformation et export de données en temps réel.

2. Objectifs

- Lire un dossier en streaming avec PySpark.
- Appliquer une transformation (ajout du chiffre d'affaire).
- Afficher les résultats en temps réel dans le terminal.
- Exporter les résultats transformés dans des fichiers CSV horodatés.

3. Environnement utilisé

- Système d'exploitation : Windows 11
- Python : 3.10+
- Apache Spark : 3.5.5
- Mode de lancement : PowerShell avec spark-submit

DownloadLibraries ▾Documentation ▾ExamplesCommunity ▾Develop

Download Apache Spark™

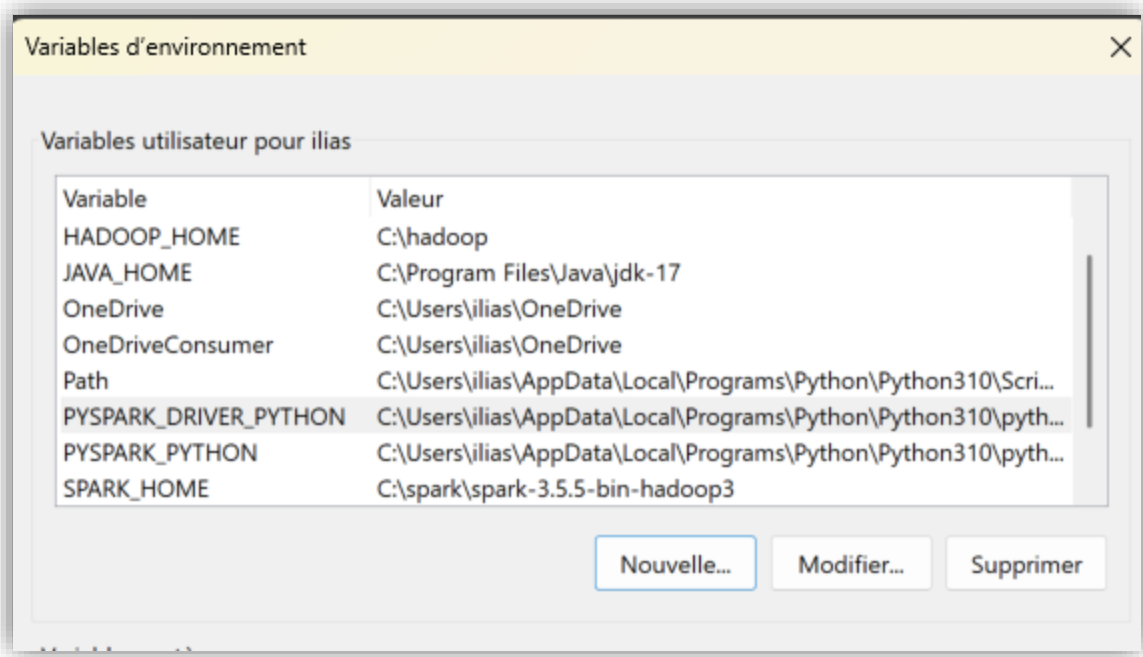
- Choose a Spark release:
- Choose a package type:
- Download Spark: [spark-3.5.5-bin-hadoop3.tgz](#)
- Verify this release using the SHA-256 checksum

Note that Spark 3 is pre-built with Hadoop 3.3. For more information, see the [Hadoop compatibility](#) page.

[Link with Spark](#)

[illegible]

```
$env:PYSARK_PYTHON="C:\Users\ilias\AppData\Local\Programs\Python\Python310\python.exe"
$env:PYSARK_DRIVER_PYTHON="C:\Users\ilias\AppData\Local\Programs\Python\Python310\python.exe"
$env:SPARK_HOME="C:\spark\spark-3.5.5-bin-hadoop3"
$env:PATH="$env:SPARK_HOME\bin;$env:PATH"
```



4. Code utilisé

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
from pyspark.sql.types import StructType, StringType, IntegerType, DoubleType,
TimestampType
import os
from datetime import datetime
```

```
spark = SparkSession.builder.appName("StreamingVentes").getOrCreate()
spark.sparkContext.setLogLevel("WARN")
```

```
schema = StructType() \
    .add("id", IntegerType()) \
    .add("produit", StringType()) \
    .add("quantite", IntegerType()) \
    .add("prix", DoubleType()) \
    .add("date_vente", TimestampType())
```

```
df_stream = spark.readStream.option("sep", ",").option("header",
"true").schema(schema).csv("C:/spark_streaming/stream_input")
```

```
df_transforme = df_stream.withColumn("chiffre_affaire", col("quantite") * col("prix"))
```

```
query_console =
```

```
df_transforme.writeStream.outputMode("append").format("console").start()
```

```
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
```

```
output_dir = f"stream_output/ventes_{timestamp}"
```

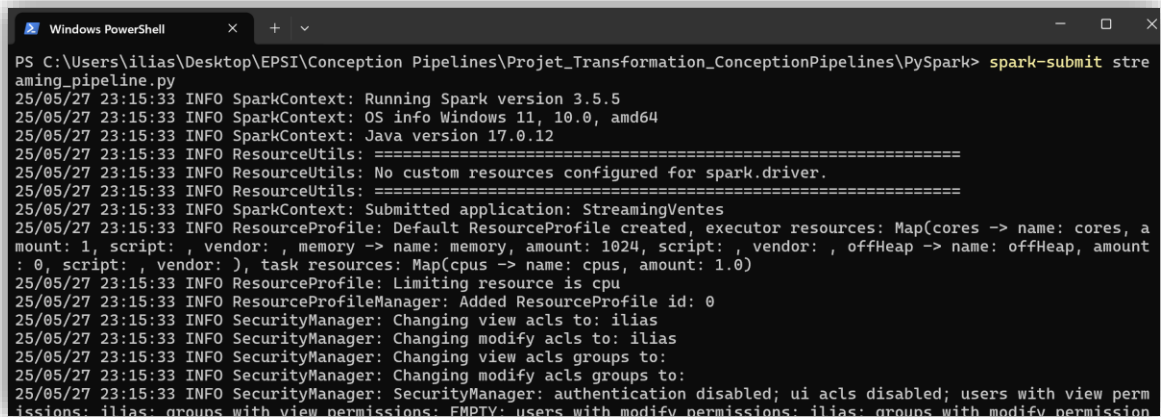
```
query_export = df_transforme.writeStream.outputMode("append") \
    .option("path", output_dir) \
    .option("checkpointLocation", f"{output_dir}/checkpoint") \
    .format("csv") \
    .start()
```

```
query_console.awaitTermination()
```

```
query_export.awaitTermination()
```

5. Résultat obtenu

Après exécution du script avec la commande `spark-submit streaming_pipeline.py`, le traitement en streaming s'est déroulé avec succès. Les données issues des fichiers CSV déposés dans le dossier `stream_input` ont été lues, transformées et affichées dans le terminal. Les fichiers de sortie sont générés dans le dossier `stream_output`, avec un horodatage précis.



```
Windows PowerShell
PS C:\Users\ilias\Desktop\EPSI\Conception Pipelines\Projet_Transformation_ConceptionPipelines\PySpark> spark-submit streaming_pipeline.py
25/05/27 23:15:33 INFO SparkContext: Running Spark version 3.5.5
25/05/27 23:15:33 INFO SparkContext: OS info Windows 11, 10.0, amd64
25/05/27 23:15:33 INFO SparkContext: Java version 17.0.12
25/05/27 23:15:33 INFO ResourceUtils: =====
25/05/27 23:15:33 INFO ResourceUtils: No custom resources configured for spark.driver.
25/05/27 23:15:33 INFO ResourceUtils: =====
25/05/27 23:15:33 INFO SparkContext: Submitted application: StreamingVentes
25/05/27 23:15:33 INFO ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name: cores, amount: 1, script: , vendor: , memory -> name: memory, amount: 1024, script: , vendor: , offHeap -> name: offHeap, amount: 0, script: , vendor: ), task resources: Map(cpus -> name: cpus, amount: 1.0)
25/05/27 23:15:33 INFO ResourceProfile: Limiting resource is cpu
25/05/27 23:15:33 INFO ResourceProfileManager: Added ResourceProfile id: 0
25/05/27 23:15:33 INFO SecurityManager: Changing view acls to: ilias
25/05/27 23:15:33 INFO SecurityManager: Changing modify acls to: ilias
25/05/27 23:15:33 INFO SecurityManager: Changing view acls groups to:
25/05/27 23:15:33 INFO SecurityManager: Changing modify acls groups to:
25/05/27 23:15:33 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: ilias; groups with view permissions: EMPTY; users with modify permissions: ilias; groups with modify permission
```

```
etete it under any circumstances, please set spark.sql.streaming.forceDeleteTempCheckpointLocat
* know deleting temp checkpoint folder is best effort.
25/05/27 23:15:40 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not supported in stre
s and will be disabled.
25/05/27 23:15:41 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not supported in stre
s and will be disabled.
-----
Batch: 0
-----
+-----+-----+-----+-----+-----+-----+
| id|produit|quantite| prix|          date_vente|  chiffre_affaire|
+-----+-----+-----+-----+-----+-----+
| 1|Clavier|      5| 29.99|2025-05-27 14:26:52|        149.95|
| 2| Souris|     10| 19.99|2025-05-27 14:26:52|199.89999999999998|
| 3| Écran|      2|149.99|2025-05-27 14:26:52|        299.98|
+-----+-----+-----+-----+-----+-----+
```

6. Conclusion

Cet exercice a permis de mettre en œuvre un pipeline de transformation de données en streaming avec PySpark. Toutes les étapes demandées ont été réalisées avec succès, du traitement à l'export des données.