etl_pipeline

August 5, 2025

Setup and imports

[4]:
```
import logging
import requests
from datetime import datetime
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, udf, lit, current_timestamp
from pyspark.sql.types import DoubleType
from pyspark.sql.functions import to_date
import shutil
import os
```

Spark initialization

[5]:
```
spark = SparkSession.builder \
    .appName("AdvancedETL") \
    .config("spark.jars", "/app/jars/mssql-jdbc-12.10.1.jre11.jar") \
    .getOrCreate()
```

Logs handling

[6]:
```
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s [%(levelname)s] %(message)s",
    handlers=[
        logging.FileHandler("/home/jovyan/etl_pipeline_error_log.log"),
        logging.StreamHandler()
    ]
)
```

Load source data

[7]:
```
sales_df = spark.read.option("header", True).csv("data/sales_data_2.csv")
sales_df.show(5)

product_df = spark.read.option("header", True).csv("data/product_reference_2.
 ↪csv")
product_df.show(5)
```

```
+-------+---------+----------+----------+------+----------+--------+--------+
|OrderID|ProductID|SaleAmount| OrderDate|Region|CustomerID|Discount|Currency|
+-------+---------+----------+----------+------+----------+--------+--------+
|   1001|      P50|    299.99|01/05/2023|  East|      C100|     0.1|     USD|
|   1002|      P72|      NULL|01/05/2023|  West|      C101|    NULL|     EUR|
|   1003|      P50|     -10.0|01-06-2023|  East|      C100|    0.05|     GBP|
|   1001|      P50|    299.99|01/05/2023|  East|      C100|     0.1|     USD|
|   1004|      P99|     150.0|      NULL| South|      C102|     0.2|     USD|
+-------+---------+----------+----------+------+----------+--------+--------+
only showing top 5 rows

+---------+-------------------+---------------+
|ProductID|        ProductName|       Category|
+---------+-------------------+---------------+
|      P50|     Wireless Mouse|    Electronics|
|      P72|    Laptop Backpack|    Accessories|
|      P99|            USB Hub|    Electronics|
|      P12|Notebook Stationery|Office Supplies|
|      P88|      Monitor Stand|Office Supplies|
+---------+-------------------+---------------+
only showing top 5 rows
```

Null handling

```
[8]: sales_df.filter(col("SaleAmount").isNull()).show()  # Check rows where␣
      ↪SaleAmount is null

     sales_df.filter(col("OrderDate").isNull()).show()   # Check rows where␣
      ↪OrderDate is null

     sales_df.filter((col("SaleAmount").isNotNull()) & (col("OrderDate").
      ↪isNotNull())).show(5)
```

```
+-------+---------+----------+----------+------+----------+--------+--------+
|OrderID|ProductID|SaleAmount| OrderDate|Region|CustomerID|Discount|Currency|
+-------+---------+----------+----------+------+----------+--------+--------+
|   1002|      P72|      NULL|01/05/2023|  West|      C101|    NULL|     EUR|
+-------+---------+----------+----------+------+----------+--------+--------+

+-------+---------+----------+---------+------+----------+--------+--------+
|OrderID|ProductID|SaleAmount|OrderDate|Region|CustomerID|Discount|Currency|
+-------+---------+----------+---------+------+----------+--------+--------+
|   1004|      P99|     150.0|     NULL| South|      C102|     0.2|     USD|
+-------+---------+----------+---------+------+----------+--------+--------+

+-------+---------+----------+----------+------+----------+--------+--------+
|OrderID|ProductID|SaleAmount| OrderDate|Region|CustomerID|Discount|Currency|
+-------+---------+----------+----------+------+----------+--------+--------+
|   1001|      P50|    299.99|01/05/2023|  East|      C100|     0.1|     USD|
|   1003|      P50|     -10.0|01-06-2023|  East|      C100|    0.05|     GBP|
|   1001|      P50|    299.99|01/05/2023|  East|      C100|     0.1|     USD|
|   1005|      PX1|      89.5|01/07/2023| North|      NULL|     0.0|     USD|
|   1006|      P72|     200.0|2023-13-01|  West|      C101|    0.15|     EUR|
+-------+---------+----------+----------+------+----------+--------+--------+
only showing top 5 rows
```

Duplicate removal

[9]:
```
sales_df_clean = (
    sales_df
    .dropna(subset=["SaleAmount", "OrderDate"])
    .dropDuplicates(["OrderID"])
    .withColumn("OrderDateParsed", to_date("OrderDate", "MM/dd/yyyy"))
    .filter(
        (col("SaleAmount").cast("double").isNotNull()) &
        (col("OrderDateParsed").isNotNull())
    )
)

sales_df_clean.show(5)
```

```
+-------+---------+----------+----------+------+----------+--------+--------+---
------------+
|OrderID|ProductID|SaleAmount|
OrderDate|Region|CustomerID|Discount|Currency|OrderDateParsed|
+-------+---------+----------+----------+------+----------+--------+--------+---
------------+
|   1001|      P50|    299.99|01/05/2023|  East|      C100|     0.1|     USD|
2023-01-05|
|   1005|      PX1|      89.5|01/07/2023| North|      NULL|     0.0|     USD|
2023-01-07|
|   1007|      P12|     120.0|01/05/2023|  East|      C105|     0.1|     GBP|
2023-01-05|
|   1008|      P88|     300.0|02/05/2023| North|      C106|     0.0|     USD|
2023-02-05|
|   1009|      P77|       0.0|03/05/2023| South|      C107|    NULL|     USD|
2023-03-05|
+-------+---------+----------+----------+------+----------+--------+--------+---
------------+
only showing top 5 rows
```

Lookup: Join with product reference

[10]:
```
enriched_df = sales_df_clean.join(product_df, on="ProductID", how="left")
print(f"[INFO] enriched_df row count: {enriched_df.count()}")
enriched_df.show(5)
```

```
[INFO] enriched_df row count: 15
+---------+-------+----------+---------+------+----------+--------+--------+---
------------+------------------+---------------+
|ProductID|OrderID|SaleAmount|
OrderDate|Region|CustomerID|Discount|Currency|OrderDateParsed|
ProductName|       Category|
+---------+-------+----------+---------+------+----------+--------+--------+---
------------+------------------+---------------+
|      P50|   1001|    299.99|01/05/2023|  East|      C100|     0.1|     USD|
2023-01-05|      Wireless Mouse|    Electronics|
|      PX1|   1005|      89.5|01/07/2023| North|      NULL|     0.0|     USD|
2023-01-07|                NULL|           NULL|
|      P12|   1007|     120.0|01/05/2023|  East|      C105|     0.1|     GBP|
2023-01-05|Notebook Stationery|Office Supplies|
|      P88|   1008|     300.0|02/05/2023| North|      C106|     0.0|     USD|
2023-02-05|       Monitor Stand|Office Supplies|
|      P77|   1009|       0.0|03/05/2023| South|      C107|    NULL|     USD|
2023-03-05|    Portable Speaker|    Electronics|
+---------+-------+----------+---------+------+----------+--------+--------+---
------------+------------------+---------------+
only showing top 5 rows
```

Currency conversion via API

[11]:
```
def get_exchange_rates():
    try:
        url = "https://api.exchangerate-api.com/v4/latest/USD"
        response = requests.get(url)
        return response.json().get("rates", {})
    except Exception as e:
        logging.error(f"Exchange rate API failed: {e}")
        return {"EUR": 1.0, "GBP": 1.0}

exchange_rates = get_exchange_rates()
broadcast_rates = spark.sparkContext.broadcast(exchange_rates)


@udf(DoubleType())
def convert_to_usd(amount, currency):
    try:
        rate = broadcast_rates.value.get(currency, 1.0)
        return float(amount) / float(rate)
    except Exception as e:
        logging.error(f"Conversion error: amount={amount}, currency={currency},␣
↪error={e}")
        return None

converted_df = enriched_df.withColumn("SaleAmountUSD",␣
↪convert_to_usd(col("SaleAmount"), col("Currency")))
print(f"[INFO] converted_df row count: {converted_df.count()}")
converted_df.show(5)
```

```
[INFO] converted_df row count: 15
+---------+-------+----------+----------+------+----------+--------+--------+---
------------+-------------------+---------------+-----------------+
|ProductID|OrderID|SaleAmount|
OrderDate|Region|CustomerID|Discount|Currency|OrderDateParsed|
ProductName|       Category|     SaleAmountUSD|
+---------+-------+----------+----------+------+----------+--------+--------+---
------------+-------------------+---------------+-----------------+
|      P50|   1001|    299.99|01/05/2023|  East|      C100|     0.1|     USD|
2023-01-05|     Wireless Mouse|    Electronics|           299.99|
|      PX1|   1005|      89.5|01/07/2023| North|      NULL|     0.0|     USD|
2023-01-07|               NULL|           NULL|             89.5|
|      P12|   1007|     120.0|01/05/2023|  East|      C105|     0.1|     GBP|
2023-01-05|Notebook Stationery|Office Supplies|159.5744680851064|
|      P88|   1008|     300.0|02/05/2023| North|      C106|     0.0|     USD|
2023-02-05|      Monitor Stand|Office Supplies|            300.0|
|      P77|   1009|       0.0|03/05/2023| South|      C107|    NULL|     USD|
2023-03-05|   Portable Speaker|    Electronics|              0.0|
+---------+-------+----------+----------+------+----------+--------+--------+---
------------+-------------------+---------------+-----------------+
only showing top 5 rows
```

Logging conversion info

[12]:
```
conversion_log_df = converted_df.withColumn("ConversionTime",␣
↪current_timestamp()) \
    .select("OrderID", "Currency", "SaleAmount", "SaleAmountUSD",␣
↪"ConversionTime")

log_path = "/app/logs/conversion_log"

# Clean entire log directory if it exists
if os.path.exists(log_path):
    try:
        shutil.rmtree(log_path)  # deletes folder and contents
        print(f"Deleted old log directory at {log_path}")
    except Exception as e:
        print(f"[WARN] Failed to delete log directory: {e}")

# Spark will create this folder fresh
conversion_log_df.coalesce(1).write \
    .mode("overwrite") \
    .option("header", True) \
    .csv(log_path)
```

```
Deleted old log directory at /app/logs/conversion_log
```

Error handling with trashold

[13]:
```
error_df = converted_df.filter(col("SaleAmountUSD").isNull())
error_df = error_df.withColumn("ErrorReason", lit("Invalid currency or␣
 ↪amount")) \
                   .withColumn("RejectedAt", current_timestamp())
error_df.write.mode("overwrite").option("header", True).csv("rejected/
 ↪rejected_records.csv")
error_df.show(5)


error_rate = error_df.count() / converted_df.count()
if error_rate > 0.05:
    raise Exception(f"[ERROR] Rejected records exceed 5% threshold␣
 ↪({error_rate*100:.2f}%)")
```

```
+---------+-------+----------+---------+------+----------+--------+--------+----
-----------+-----------+--------+-------------+-----------+----------+
|ProductID|OrderID|SaleAmount|OrderDate|Region|CustomerID|Discount|Currency|Orde
rDateParsed|ProductName|Category|SaleAmountUSD|ErrorReason|RejectedAt|
+---------+-------+----------+---------+------+----------+--------+--------+----
-----------+-----------+--------+-------------+-----------+----------+
+---------+-------+----------+---------+------+----------+--------+--------+----
-----------+-----------+--------+-------------+-----------+----------+
```

Final clean data

[14]:
```
final_df = converted_df.filter(col("SaleAmountUSD").isNotNull())
```

Write to SQL Database

[15]:
```
jdbc_url = "jdbc:sqlserver://host.docker.internal:1433;databaseName=SalesDB;
 ↪encrypt=true;trustServerCertificate=true"

db_props = {
    "user": "sa",
    "password": "qwe123!@#",
    "driver": "com.microsoft.sqlserver.jdbc.SQLServerDriver"
}

final_df.write.jdbc(url=jdbc_url, table="SalesEnriched", mode="append",␣
 ↪properties=db_props)
```

Wrire rejected records to SQL for tracking

[16]:
```
error_df.write.jdbc(url=jdbc_url, table="RejectedRecords", mode="append",␣
 ↪properties=db_props)
```