ASSIGNMENT — LAB 1.2 : E-Commerce Dataset Exploration (DataFrames)

Learning Outcomes

By the end of this assignment, students should be able to:

- Prepare and mount data files for a Spark cluster running in Docker.
- Generate synthetic CSV data using a Python script.
- Load CSV data into Spark using the DataFrame API.
- Perform basic **data quality checks** (nulls, duplicates).
- Compute **descriptive statistics** and answer business questions using aggregations and groupings.
- Export a small **summary report** from Spark.

---

1. Context & Environment Constraints

You are working with a **simulated e-commerce domain**:

- `customers.csv` — customer information
- `products.csv` — product catalog
- `orders.csv` — order transactions

These files must be accessible to your Spark cluster.

Environment constraints:

- Spark cluster runs in Docker (Spark master reachable at `spark://localhost:7077`).
- Data directory on the host: `spark-data/ecommerce/` inside your project root (e.g. `data-engineering-course/`).
- Data must be available **inside the Spark containers** via one of these approaches (choose one):
  1. **Bind mount** `spark-data` into the Spark containers via `docker-compose.yml`, e.g. mounting host `./spark-data` to `/opt/spark/data` in the container.
  2. **Symlink approach** (host side), for example:

```
sudo mkdir -p /opt/spark
sudo ln -s ~/SANDBOX/PLAYGROUND-DEV/data-engineering-course/spar
```

(Paths may differ on your machine.)

3. **Copy scripts & data into the container** and run them via `docker exec` (less recommended for iterative work).

You must end up with Spark being able to read:

```
spark-data/ecommerce/customers.csv
spark-data/ecommerce/products.csv
spark-data/ecommerce/orders.csv
```

from the **Spark driver context**.

---

2. Assignment Tasks

Part A — Data Preparation

Goal: Generate the CSV dataset.

1. From your project root (e.g. `data-engineering-course`), create the directory structure:

   ```
   mkdir -p spark-data/ecommerce
   cd spark-data/ecommerce
   ```

2. Create a Python script `generate_data.py` that:

   - Generates:
     - **1000 customers** with fields similar to:
       - `customerNumber`, `customerName`, `contactFirstName`, `contactLastName`, `phone`, `addressLine1`, `city`, `state`, `country`, `creditLimit`, `customerSegment`.
     - **100 products** with fields like:

        - `productCode` , `productName` , `productCategory` , `quantityInStock` , `buyPrice` , `MSRP` .
    - **5000 orders** with fields like:
        - `orderNumber` , `orderDate` , `requiredDate` , `status` , `customerNumber` , `totalAmount` , `paymentMethod` .
- Writes three CSV files (with header row):
    - `customers.csv`
    - `products.csv`
    - `orders.csv`

3. Run the generator:

```
python generate_data.py
```

4. Verify that all three CSV files are created and visually inspect one of them (e.g. with `head` or a text editor).

Part B — Load & Explore Data with Spark

Goal: Implement an exploration script using Spark DataFrames.

1. From the **project root**, create a script:

```
lab2_explore_data.py
```

2. In this script, you must:

2.1 Create a SparkSession

- Application name: `"Day1-DataExploration"`
- Master: `spark://localhost:7077`
- Driver memory: at least `2g`

```
spark = (
    SparkSession.builder
    .appName("Day1-DataExploration")
    .master("spark://localhost:7077")
```

```
        .config("spark.driver.memory", "2g")
        .getOrCreate()
    )
```

2.2 Load the three CSV datasets

- Use `option("header", "true")`
- Use `option("inferSchema", "true")`

Required paths (from project root):

- `"spark-data/ecommerce/customers.csv"`
- `"spark-data/ecommerce/products.csv"`
- `"spark-data/ecommerce/orders.csv"`

Store them as:

- `customers`
- `products`
- `orders`

Print a short message confirming load success.

> Optional: cache them with `.cache()` (good habit even if not strictly needed here).

2.3 Inspect schemas

- Print the schema of each DataFrame with `.printSchema()` :
    - CUSTOMERS Schema
    - PRODUCTS Schema
    - ORDERS Schema

2.4 Basic statistics (size)

- Compute and print:
    - Total number of customers
    - Total number of products
    - Total number of orders

Use `.count()` on each DataFrame.

2.5 Data preview

- Show the first 5 rows of each DataFrame with `.show(5, ...)`.

2.6 Data quality checks

- For `customers` and `orders`, compute the number of **nulls per column**:
  - Use `select([...])` with `count(when(col(c).isNull(), c))`.
- Check for **duplicate IDs**:
  - For customers: compare `customers.count()` with `customers.select("customerNumber").distinct().count()`.
  - For orders: same with `orderNumber`.
- Print the number of duplicates for each.

2.7 Exploratory analysis

Implement at least:

- **Customers by segment**
  Group by `customerSegment`, count, order descending.
- **Top 10 countries by customer count**
  Group by `country`, count, order descending, show(10).
- **Orders by status**
  Group by `status`, count.
- **Orders by payment method**
  Group by `paymentMethod`, count.
- **Products by category**
  Group by `productCategory`, count.

2.8 Numerical analysis

Compute and show:

- **Order amount statistics** (from `orders.totalAmount`):
  - count, min, max, average, total sum.
- **Credit limit statistics by segment** (grouped by `customerSegment`):
  - count, average credit, max credit.

- **Product price statistics**:
  - For `buyPrice` and `MSRP` : min, max, average.

2.9 Summary report export

- Compute:
  - total number of customers,
  - total number of products,
  - total number of orders,
  - total revenue (`sum(totalAmount)`),
  - average order value (`avg(totalAmount)`).
- Create a small DataFrame `summary` of shape:

Metric	Value
Total Customers	...
Total Products	...
Total Orders	...
Total Revenue	...
Average Order Value	...

- Write the summary as a **single CSV file** (use `coalesce(1)`) to:

```
spark-data/ecommerce/summary/
```

- Ensure write mode is `"overwrite"` and header is `true` .

3. At the end of the script, print clear end messages and stop Spark with `.stop()` .
4. Run the script:

```
python lab2_explore_data.py
```

5. Verify:
   - All sections print output without error.
   - CSV file is created under `spark-data/ecommerce/summary/` .

Part C — Business Questions with Spark

Extend your existing script (e.g. add a `# SECTION 9: QUESTIONS` before `spark.stop()`), or create a second script reusing the same DataFrames.

For each question, **answer using Spark DataFrame operations only** (no manual counting in Python lists).

> You may reuse `customers` , `orders` , `products` .

Question 1 — Country with highest total credit limit

> Which country has the highest **sum of `creditLimit`** ?

- Group `customers` by `country` .
- Aggregate with `sum("creditLimit")` .
- Order descending and take the top 1.

Question 2 — Most common order status

> What is the most frequent **order `status`** ?

- Group `orders` by `status` .
- Count, order descending, take top 1.

Question 3 — Product category with most stock

> Which `productCategory` has the largest total `quantityInStock` ?

- Group `products` by `productCategory` .
- Aggregate `sum("quantityInStock")` .
- Order descending, take top 1.

Question 4 — Percentage of Enterprise customers

> What percentage of customers belong to the `"Enterprise"` segment?

Hint formula:

[
\text{percentage} = \frac{\text{Enterprise customers}}{\text{Total customers}} \times 100

```
]
```

- Compute:
  - `total_customers`
  - `enterprise_customers` where `customerSegment == "Enterprise"`
- Then compute the percentage using Spark expressions or in Python after collecting a small result.

Question 5 — Distribution of orders by month

> How many orders per **month**?

- Convert `orderDate` to a proper date if needed (e.g. `to_date("orderDate")`).
- Use `month()` function on a date column.
- Group by month and count orders.
- Order by month (1–12).

---

3. Hints

- Use `option("inferSchema", "true")` to let Spark infer numeric types for `creditLimit` , `totalAmount` , etc.
- For **null checks**, the classic pattern is:

```
customers.select([
    count(when(col(c).isNull(), c)).alias(c)
    for c in customers.columns
])
```

- For **duplicate checks**, compare `count()` vs `distinct().count()` on the key column.
- Use `groupBy(...).agg(...)` for numerical aggregations.
- For **month extraction**:
  - Convert string date `"YYYY-MM-DD"` to date type using `to_date(col("orderDate"), "yyyy-MM-dd")` if needed.
  - Then apply `month()` .
- To create a small single CSV file, use `coalesce(1)` before `write` .

---

4. Common Pitfalls

- Spark cannot read files if **paths inside the container** and **paths on the host** don't match your volume mounts/symlinks.
- `month("orderDate")` will fail if `orderDate` stays as a string → cast to date first.
- Forgetting `header=true` leads to the first row being parsed as data.
- `inferSchema=false` (default) will give you everything as `string`, breaking aggregations on numeric fields.
- Writing output to a path that already exists without specifying `mode("overwrite")` will cause errors.

---

5. Deliverables

You should submit:

1. **Generated CSV files:**
   - `customers.csv`
   - `products.csv`
   - `orders.csv`
2. **Console output of `lab2_explore_data.py`**
   - Either as pasted text or as a `.txt` file.
3. **Code answering the 5 questions**
   - Either appended to `lab2_explore_data.py` or in a separate script (e.g. `lab2_questions.py`).
4. **Short data quality notes (3–5 bullet points)**, including:
   - Null values detected (where and how many).
   - Duplicate IDs found (if any).
   - Any skewed distributions (e.g. one segment or one country dominating).
   - Any surprising patterns in orders or products.

---

6. Grading Grid

Criterion	Points
Data generation script + CSV files	4
Correct Spark loading & schema inspection	4
Data quality checks (nulls + duplicates)	4
Exploratory & numerical analysis sections	4
Correct answers to the 5 business questions	4
Quality of notes / interpretation	4
Total	**24**