CASE STUDY 1
Title: Order Processing and Analytics Pipeline using PySpark

A retail company operates in multiple cities and receives daily order data from its transactional systems.
The data is exported every day as a CSV file named:

```
orders.csv
```

This file is the only input to the system.
There are no assumptions of any other datasets, sources, or prior processing.

Your responsibility is to build a complete PySpark pipeline that makes this data analytics-ready and business-usable.

The file contains the following columns:

```
order_id
customer_id
city
category
product
amount
order_date
status
```

The data looks normal at first glance, but when inspected carefully it has real-world quality problems such as:

- Extra spaces in city and category names
- Mixed casing (mumbai, Mumbai, MUMBAI)
- Amount stored as text instead of number
- Invalid amount values like `invalid`, `12,000`, empty strings
- Multiple date formats
- Duplicate order IDs
- Cancelled and completed orders mixed together

This is not artificial data. This is exactly how production CSV exports usually look.

Your task is to make this dataset:

- Clean
- Consistent

- Optimized
- Analytics ready

Using only PySpark.

No shortcuts.
No assumptions.
No external tools.

PHASE 1 – Data Ingestion

Read the file:

```
orders.csv
```

Tasks:

1. Load the CSV file without schema inference.
2. Print the schema.
3. Count total records.
4. Show sample rows.
5. Explain why all columns must be treated as StringType initially.

PHASE 2 – Data Cleaning

The dataset must be cleaned in the following way:

1. Remove leading and trailing spaces from:
    - city
    - category
    - product
2. Standardize text:
    - Convert city, category, and product to proper case.
3. Clean the amount column:
    - Remove commas.
    - Replace empty strings and invalid values with null.
    - Convert amount into IntegerType.
    - Rows with invalid amounts must not crash the pipeline.
4. Clean the order_date column:

- Support the following formats:
  - yyyy-MM-dd
  - dd/MM/yyyy
  - yyyy/MM/dd
- Create a new column:

```
order_date_clean
```

  with DateType.

5. The original columns must remain for auditing.

PHASE 3 – Data Validation

1. Count how many records had invalid amounts.
2. Count how many records had invalid dates.
3. Identify duplicate order_id values.
4. Remove duplicates using order_id.
5. Filter only records with:

```
status = "Completed"
```

6. Record row counts at every stage.

PHASE 4 – Performance Engineering

1. Check the number of partitions.
2. Run a groupBy on city and calculate total revenue.
3. Use:

```
explain(True)
```

to analyze execution.
4. Identify where shuffle happens.
5. Repartition the dataset by city.
6. Compare execution plans before and after repartition.

This phase exists to demonstrate understanding of Spark internals, not just outputs.

PHASE 5 – Analytics

Using the cleaned dataset:

1. Total revenue per city.
2. Total revenue per category.
3. Average order value per city.
4. Top 10 products by revenue.
5. Cities sorted by revenue descending.

PHASE 6 – Window Functions

1. Rank cities by revenue.
2. Rank products inside each category by revenue.
3. Find the top product for every category.
4. Identify the top 3 performing cities.

PHASE 7 – Broadcast Join

A small lookup table is provided:

```
city,region
Delhi,North
Mumbai,West
Bangalore,South
Hyderabad,South
Pune,West
Chennai,South
Kolkata,East
```

Tasks:

1. Join the orders data with this city-region dataset.
2. Apply broadcast join explicitly.
3. Verify using the physical plan that:

```
BroadcastHashJoin
```

is used.
4. Explain why broadcast join is efficient in this case.

PHASE 8 – UDF

Create a classification based on amount:

```
amount >= 80000  → High
amount >= 40000  → Medium
else             → Low
```

Add a new column:

```
order_value_category
```

Analyze distribution.

PHASE 9 – RDD

1. Convert the cleaned DataFrame to RDD.
2. Compute:
    - Total revenue using reduce.
    - Orders per city using map and reduce.
3. Explain why DataFrames are preferred over RDDs for analytics.

PHASE 10 – Caching

1. Identify datasets reused in multiple queries.
2. Apply cache().
3. Execute multiple aggregations.
4. Compare performance.
5. Unpersist after use.

Explain why unnecessary caching is dangerous.

PHASE 11 – Storage Formats

1. Write cleaned dataset to:

```
Parquet
```

Partitioned by:

```
city
```

2. Write aggregated datasets to:

```
ORC
```

3. Read both formats back and validate:
    - Schema
    - Row counts
4. Compare size and performance against CSV.

PHASE 12 – Debugging

Explain why this breaks:

```
df = df.filter(df.amount > 50000).show()
```

And why after this line `df` is no longer a DataFrame.

PHASE 13 – Final Validation

1. Confirm:
    - amount is IntegerType
    - order_date_clean is DateType
    - No nulls in critical business fields.
2. Document:
    - Cleaning strategy
    - Performance strategy
    - Debugging learnings
