□ Low-Level Design (LLD) – Retail Sales Analysis (PySpark Version)

Difficulty Level: Easy | **Total Marks**: 20
Standards Followed: 4 Functions | 4 Visible Test Cases

□ Summary of Requirements (Converted to PySpark)

- Use `spark.read.csv()` to load sales data
- Use `groupBy()` and `agg()` to calculate revenue per category/product
- Use `.orderBy()` to sort by revenue or quantity
- Use `.limit()` and `.collect()` for Top-N logic
- Return `float`, `tuple`, or `list` as needed

□ Concepts Tested

- Reading CSV with PySpark
- Grouping and aggregation using `.groupBy().agg()`
- Adding derived columns using `withColumn()`
- Sorting using `.orderBy(desc)`
- Extracting top records using `.limit().collect()`
- Type-safe output handling

□ Problem Statement

You're given a `sales.csv` file that tracks retail transactions.
Each record contains product name, category, quantity, and unit price.

Perform the following analyses using **PySpark**:

- Load the sales data
- Calculate total revenue
- Identify top product category
- List the top 3 products by quantity sold

□ Operations

□ 1. Load Transactions

□ Load the CSV file into a PySpark DataFrame.

□ Function Prototype:

```
def load_transactions(self, path: str) -> DataFrame:
```

□ Input: `"sales.csv"`
□ Output: Spark DataFrame

□ Implementation Flow:

- Use `spark.read.csv(path, header=True, inferSchema=True)`
- Return full DataFrame

□ 2. Total Purchase Value

□ Calculates total revenue across all transactions.

□ Function Prototype:

```
def total_purchase_value(self, df: DataFrame) -> float:
```

□ Input: DataFrame with `quantity`, `unit_price`
□ Output: `float`

□ Implementation Flow:

- Use `withColumn("revenue", quantity * unit_price)`
- Use `.agg({"revenue": "sum"})`
- Return the result as `float`

□ 3. Find Top Product Category

□ Find the product category with highest total sales.

□ Function Prototype:

```
dit
```

```
def top_product_category(self, df: DataFrame) -> tuple:
```

 Input: DataFrame
 Output: Tuple – `(category_name, total_revenue)`

 Implementation Flow:

- Add `revenue = quantity * unit_price`
- Group by `product_category`, sum revenue
- Order by total revenue descending
- Return top 1 row as a tuple

---

4. Top 3 Products by Units Sold

 Return the top 3 best-selling products by quantity.

 Function Prototype:

```
def top_n_products(self, df: DataFrame) -> list:
```

 Input: DataFrame
 Output: List of tuples – `[(product_name, total_quantity), ...]`

 Implementation Flow:

- Group by `product_name`, sum `quantity`
- Order by quantity descending
- Use `.limit(3).collect()` to get top 3
- Return list of tuples

Test Cases & Marks Allocation

Test Case ID	Description	Function	Marks
TC1	Load sales.csv into DataFrame	`load_transactions()`	5
TC2	Calculate total purchase value	`total_purchase_value()`	5
TC3	Top product category by revenue	`top_product_category()`	5
TC4	Top 3 best-selling products	`top_n_products()`	5

 Total Marks: 20

□ Visible Test Cases (4)

□ TC1: Load CSV

- Input: `"sales.csv"`
- Expected: Non-empty DataFrame with required columns

□ TC2: Total Purchase Value

- Input: DataFrame
- Expected Output: `128500.75` (example)

□ TC3: Best Category

- Input: DataFrame
- Expected Output: `("Grocery", 42000.0)`

□ TC4: Top 3 Products

- Input: DataFrame
- Expected Output: `[("Rice", 430), ("Notebook", 420), ("Soap", 390)]`