Assignment#1: Ecommerce Top-Seller items Recommendation System

Use Case Overview

You are tasked with building a **recommendation system for sellers** on an e-commerce platform.

In an e-commerce marketplace:

- There are **multiple sellers**, each with their own catalog of items.
- The platform tracks **sales data for all items** across all sellers and competitors.
- Some items are **top-selling** but not available in all seller catalogs.

Your job is to **analyze internal sales and competitor data** and **recommend top 10 selling items that each seller currently does not have**. This will help sellers **increase revenue and profit** by onboarding these items.

This pipeline should **ingest, clean, and validate data**, handle **schema evolution**, and generate actionable **recommendations**.

Key Objectives

1. **Identify top-selling items within the company**
    - Analyze all sellers' sales data.
    - Determine which items sell the most in each category.
2. **Compare each seller's catalog against top-selling 10 items**
    - Identify items missing in their catalog.
    - Recommend missing items to the seller.
3. **Analyze competitor data**
    - Identify top-selling items in the market (competitor sales).
    - Recommend items missing from the company's catalog but performing well in the market.
4. **Compute business metrics for recommendations**
    - **Current price** (from company or competitor data).
    - **Expected revenue** if the seller adds the item:
      expected_revenue = expected_units_sold * marketplace_price

---

Inputs Datasets:

1. Seller Catalog

Column	Type	Key	Possible Issues	Description
seller_id	STRING	Part of key (seller_id + item_id)	Missing, duplicates	Unique identifier of the seller. Each seller has their own catalog.
item_id	STRING	Part of key (seller_id + item_id)	Missing, duplicates	Unique identifier of the item/product.
item_name	STRING		Null, inconsistent casing	Name of the item as listed in the catalog. May have inconsistent casing or typos.
category	STRING		Null, inconsistent labels	Category to which the item belongs (e.g., Electronics, Apparel). Helps with grouping and analysis.
marketplace_price	DOUBLE		Missing, negative	Price at which the item is being sold on the marketplace by this seller. Used to calculate expected revenue.
stock_qty	INT		Null, negative	Current stock quantity of the item available with the seller. Negative or null values indicate data issues.

---

2. Company Sales Data

Column	Type	Key	Possible Issues	Description
item_id	STRING	Key	Missing, duplicates	Unique identifier of the item sold. Used to join with catalog and competitor data.
units_sold	INT		Negative, missing	Number of units sold for the item in a given period. Negative or missing values indicate incorrect records.
revenue	DOUBLE		Missing, negative	Total revenue generated by the item during the sale period. Negative or missing values indicate data issues.
sale_date	DATE		Malformed dates	Date of sale. Used for time-based aggregations and trend analysis. Malformed or future dates are invalid.

---

3. Competitor Sales Data

Column	Type	Key	Possible Issues	Description
item_id	STRING	Part of key (seller_id + item_id)	Missing, duplicates	Unique identifier of the item sold by the competitor.
units_sold	INT		Negative, missing	Number of units sold by the competitor for this item.
revenue	DOUBLE		Missing, zero, negative	Total revenue generated by the competitor for this item.
marketplace_price	DOUBLE		Missing, zero	Price at which the competitor is selling the item. Used to compare prices and calculate expected revenue.
seller_id	STRING	Part of key (seller_id + item_id)	Null	Identifier for the competitor seller. Null values indicate missing data.
sale_date	DATE		Malformed dates	Date of sale. Used to identify trends and top-selling items.

Notes for Students

- **seller_id** and **item_id** together form a **unique key for the seller catalog** and Competitor sales datasets
- **item_id** is the key for company sales.
- **marketplace_price** is the main column for revenue calculations;
- All datasets may contain **dirty data**, which should be cleaned or moved to the **quarantine zone**.
- Understanding the **purpose of each column** helps determine cleaning, validation, and derivation logic (e.g., revenue, expected revenue).

Key Requirements

1. ETL Ingestion (15 Marks)

- **Read input datasets (CSV/JSON) via a YAML-configurable pipeline. This should support your daily incremental data as well**
- **Use Apache Hudi for:**
    - **Schema evolution (handle new columns or type changes)**
    - **Incremental upserts (idempotent writes)**
- **Data Cleaning and DQ Checks**
- **Quarantine Zone Handling**
- **Use medallion architecture and on top of that have a quarantine zone to move all your bad data, before loading the data into gold layer you need to do the data cleaning and DQ validations**
- **Output cleaned data to Hudi tables for consumption.**
- **3 different pipelines**
- **Your final output should be 3 different Hudi tables with overwrite mode**

Inputs :

Data Cleaning and DQ Checks:

1. Seller Catalog

Columns: seller_id, item_id, item_name, category, marketplace_price, stock_qty

Cleaning Steps:

- **Trim whitespace in all string columns (seller_id, item_id, item_name, category).**
- **Normalize casing:**
  item_name → Title Case
  category → Standardized labels (e.g., "Electronics", "Apparel")
- **Remove duplicates based on key (seller_id + item_id).**
- **Convert marketplace_price → DOUBLE, stock_qty → INT.**
- **Fill missing stock_qty with 0 if applicable.**

DQ Checks:

Rule	Expression	Action
Seller ID exists	seller_id IS NOT NULL	Quarantine
Item ID exists	item_id IS NOT NULL	Quarantine
Price valid	marketplace_price >= 0	Quarantine
Stock valid	stock_qty >= 0	Quarantine
Item name present	item_name IS NOT NULL	Quarantine
Category present	category IS NOT NULL	Quarantine

---

2. Company Sales Data

Columns: item_id, units_sold, revenue, sale_date

Cleaning Steps:

- **Trim strings if needed (item_id)**
- **Convert units_sold → INT, revenue → DOUBLE, sale_date → DATE.**
- **Remove duplicates based on item_id.**
- **Fill missing units_sold or revenue with 0.**
- **Standardize sale_date format.**

DQ Checks:

Rule	Expression	Action
Item ID exists	item_id IS NOT NULL	Quarantine
Units sold valid	units_sold >= 0	Quarantine
Revenue valid	revenue >= 0	Quarantine
Sale date valid	sale_date IS NOT NULL AND sale_date <= current_date()	Quarantine

---

3. Competitor Sales Data

Columns: item_id, units_sold, revenue, marketplace_price, seller_id, sale_date

Cleaning Steps

- **Trim strings (item_id, seller_id) and normalize casing.**
- **Convert numeric columns to proper types (units_sold → INT, revenue → DOUBLE, marketplace_price → DOUBLE).**
- **Convert sale_date → DATE.**
- **Fill missing numeric fields (units_sold, revenue, marketplace_price) with 0.**

DQ Checks

Rule	Expression	Action
Item ID exists	item_id IS NOT NULL	Quarantine
Seller ID exists	seller_id IS NOT NULL	Quarantine
Units sold valid	units_sold >= 0	Quarantine
Revenue valid	revenue >= 0	Quarantine
Marketplace price valid	marketplace_price >= 0	Quarantine
Sale date valid	sale_date IS NOT NULL AND sale_date <= current_date()	Quarantine

Quarantine Zone Handling:

- Records failing **any DQ check** should be **moved to a quarantine path**
- Quarantine path should include:
  - Input datasetname
  - Original record
  - `dq_failure_reason` (e.g., `price_negative`, `missing_item_id`)

---

2. Consumption Layer (5 Marks):

- **Use medallion architecture**
- **Read the above 3 input hudi tables**
- **Data Transformations**
- **Recommendation Calculation**
- **Output gold zone cleaned data to csv files with overwrite mode**

Inputs:

Data Transformation:

- Aggregate **company sales data** to find top-selling items per category.
- Aggregate **competitor sales data** to find top-selling items in the market.
- Compare each seller's catalog with:
    1. **Company top-selling items**
    2. **Competitor top-selling items**
    3. Identify **missing items per seller**.

Recommendation Calculation

- For each missing item, calculate:
    - market_price (from competitor or company sales)
    - expected_units_sold (based on historical top-selling numbers)

      expected_units_sold = total units sold for the item / number of sellers selling this item
    - expected_revenue = expected_units_sold * marketplace_price

- Create a **recommendation file** for sellers:
    - Output Columns:

      seller_id, item_id, item_name, category, market_price, expected_units_sold, expected_revenue

---

Assignment Deliverables:

Instructions:

1. **Project Submission:**

   You must upload your zipped folder following the project structure provided below.

2. **Configuration File (ecomm_prod.yml):**

   Your input file should contain only the variables listed in the sample configuration provided.

3. **Spark Submit Command:**

   Use the similar Spark submit command as shown in the sample (only one config file as an input and try to use the same hudi versions)

4. **Intermediate S3 file access:**

   If you are storing intermediate data in S3 buckets, ensure that your buckets are set to global access, allowing any user to read and write. If you are using the local file system, you can ignore this instruction.

5. **You can use pyspark as a tech stack here.**

Project Structure:

<rollnumber>/ecommerce_seller_recommendation/<s3 or local>/ #if you are using s3 bucket, mention "s3" here otherwise you can use local files and mention "local"

|

├── configs/

| └── ecomm_prod.yml # only input/output paths

├── src/

| ├── etl_seller_catalog.py

| ├── etl_company_sales.py

| ├── etl_competitor_sales.py

| └── consumption_recommendation.py

├── scripts/

| ├── etl_seller_catalog_spark_submit.sh

| ├── etl_company_sales_spark_submit.sh

| ├── etl_competitor_sales_spark_submit.sh

| └── consumption_recommendation_spark_submit.sh

└── README.md # Optional: in case if you want to highlight anything about your assignment here

Sample ecomm_prod.yml

```
# Input and output paths for ETL and consumption
# All output paths are S3 or local paths (Hudi tables for ETL, CSV for consumption)

seller_catalog:
  input_path: "<s3 bucket or local folder>/raw/seller_catalog/seller_catalog_clean.csv"    # raw input files
  hudi_output_path: "<s3 bucket or local folder>/<rollnumber>/processed/seller_catalog_hudi/"  # ETL output (Hudi)

company_sales:
  input_path: "<s3 bucket or local folder>/raw/company_sales/company_sales_clean.csv"     # raw input files
  hudi_output_path: "<s3 bucket or local folder>/<rollnumber>/processed/company_sales_hudi/"  # ETL output (Hudi)

competitor_sales:
```

```
  input_path: <s3 bucket or local folder>/raw/competitor_sales/competitor_sales_clean.sv"  # raw
input files
  hudi_output_path: "<s3 bucket or local
folder>/<rollnumber>/processed/competitor_sales_hudi/" # ETL output (Hudi)

recommendation:
  # Final consumption layer input/output
  seller_catalog_hudi: ""<s3 bucket or local folder>/<rollnumber>/processed/seller_catalog_hudi/"
  company_sales_hudi: "<s3 bucket or local
folder>/<rollnumber>/processed/company_sales_hudi/"
  competitor_sales_hudi: ""<s3 bucket or local
folder>/<rollnumber>/processed/competitor_sales_hudi/"
  output_csv: "s3://your-
bucket/<rollnumber>/processed/recommendations_csv/seller_recommend_data.csv"   # final CSV
```

Example:

```
seller_catalog:
  input_path: "/home/cloud/asgn1/raw/seller_catalog/seller_catalog_clean.csv"
  hudi_output_path: "/home/cloud/asgn1/R1234/processed/seller_catalog_hudi/"

company_sales:
  input_path: "/home/cloud/asgn1/raw/company_sales/company_sales_clean.csv"
  hudi_output_path: "/home/cloud/asgn1/R1234/processed/company_sales_hudi/"

competitor_sales:
  input_path: "/home/cloud/asgn1/raw/competitor_sales/competitor_sales_clean.sv"
  hudi_output_path: "/home/cloud/asgn1/R1234/processed/competitor_sales_hudi/"

recommendation:
  seller_catalog_hudi: "/home/cloud/asgn1/R1234/processed/seller_catalog_hudi/"
  company_sales_hudi: "/home/cloud/asgn1/R1234/processed/company_sales_hudi/"
  competitor_sales_hudi: "/home/cloud/asgn1/R1234/processed/competitor_sales_hudi/"
  output_csv:
"/home/cloud/asgn1/R1234/processed/recommendations_csv/seller_recommend_data.csv"
```

Explanation for Students

- **input_path → location of raw input files (CSV) in S3 or local.**
- **hudi_output_path → Hudi table output path in S3 or local(for ETL pipelines).**
- **seller_catalog_hudi, company_sales_hudi, competitor_sales_hudi → Hudi paths used as inputs for consumption program.**
- **output_csv → CSV path in S3 or local for recommendations.**

Sample Spark Submit Command:

```
spark-submit --packages org.apache.hudi:hudi-spark3.5-
bundle_2.12:0.15.0,org.apache.hadoop:hadoop-
aws:3.3.4,com.amazonaws:aws-java-sdk-bundle:1.12.262 --conf
spark.serializer=org.apache.spark.serializer.KryoSerializer
--conf spark.sql.legacy.timeParserPolicy=LEGACY
<rollnumber>/ecommerce_seller_recommendation/src/<program
name>.py  --config configs/ecomm_prod.yml
```

Sample Spark Session:

```
def get_spark_session(app_name: str) -> SparkSession:
    spark = (
        SparkSession.builder
        .appName(app_name)
        .config("spark.serializer",
"org.apache.spark.serializer.KryoSerializer")
        .config("spark.sql.legacy.timeParserPolicy", "LEGACY")
        .getOrCreate()
    )
    spark.sparkContext.setLogLevel("WARN")
    return spark
```