Retail Sales Data Pipeline - Task Summary

1. Uploaded Raw Data to GCS (Bronze Layer)

What you did:

Uploaded `customers_sample.csv`, `products_sample.csv`, and `sales_sample.json` to GCS under:

gs://retail-data-pipeline-bucket/bronze/products/

Why it's needed:

Acts as the raw data lake storage for staging raw files from various sources before any transformation. Ensures traceability and rollback.

When it is needed:

First step in the pipeline - known as the Bronze Layer in the data lake architecture.

How it was done:

Used `gsutil cp` or Cloud Console to upload local files into GCS.

2. Wrote Dataflow Python Pipelines

What you did:

Created 3 Apache Beam (Dataflow) pipelines for:

- customers_transform_pipeline.py
- products_transform_pipeline.py
- sales_transform_pipeline.py

Why it's needed:

To clean and normalize the raw data before storing in the Silver layer.

When it is needed:

After ingestion, just before loading into analytics or reporting tools.

How it was done:

Used Apache Beam with Python, defined custom DoFn classes to transform each row, saved `.py` files using nano editor in Cloud Shell.

3. Created Metadata Files

What you did:

Created `metadata.json` files for each pipeline (e.g., sales_metadata.json).

Why it's needed:

Defines runtime parameters like input/output paths for Flex Template execution.

When it is needed:

At the time of building Flex Templates and launching Dataflow jobs.

How it was done:

Used nano to create JSON files with name, description, and parameter keys.

4. Built Dataflow Flex Templates

What you did:

Built templates using gcloud CLI for:

- customers_template.json
- products_template.json
- sales_template.json

Why it's needed:

So that the pipeline can be triggered on-demand or on-schedule without having to rebuild each time.

When it is needed:

Before setting up automation using Pub/Sub or Scheduler.

How it was done:

Used `gcloud dataflow flex-template build` commands with proper flags, linking metadata and Python files.