END TO END E-COMMERCE PROJECT USING SPARK AND DATABRICKS

Problem Statement:

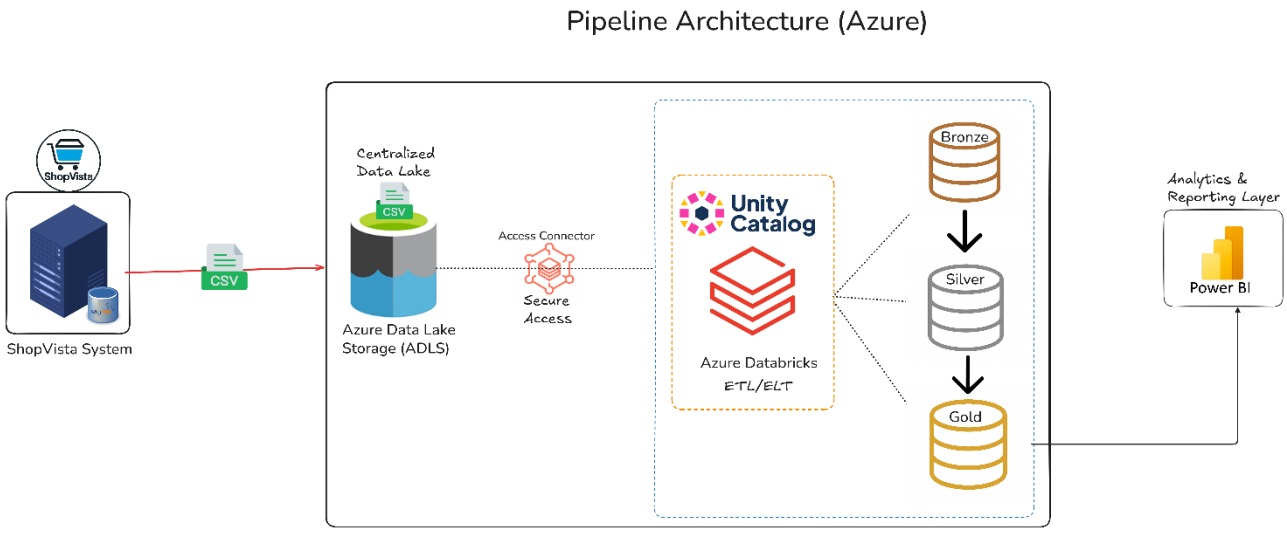
- ShopVista's business data was scattered across multiple systems, requiring manual reconciliation for reporting.
- This led to delayed insights, data inconsistencies, and limited visibility into sales, customer behavior, returns, and shipments.
- The company needed a centralized and automated data platform to ensure a single source of truth and enable scalable analytics.

Project Objective:

- Build a centralized data platform to unify orders, shipments, returns, and master data.
- Automate data ingestion and transformation to reduce manual reporting effort.
- Deliver clean, analytics-ready data for reliable business insights.
- Enable self-service reporting through Power BI dashboards.

Scope:

- Designed and implemented a Bronze → Silver → Gold data architecture on Azure.
- Ingested historical and incremental data using Databricks Auto Loader.
- Performed data cleaning, standardization, and enrichment for analytics.
- Built integrated fact and dimension tables using a star schema.
- Developed Power BI dashboards for business, customer, product, return, and shipment analysis.



Resources

Recent Favorite

Name	Type	Last Viewed
shopvistaecommerce001	Storage account	a few seconds ago
dbw-ecommerce-dev-centralindia-001	Azure Databricks Service	23 minutes ago
rg-ecommerce-dev-centralindia-001	Resource group	2 weeks ago
ac-dbw-ecommerce-dev-centralindia-001	Access Connector for Azure Databricks	3 weeks ago
databricks-rg-dbw-ecommerce-dev-centralindia-001-nxj24j7nhkirq	Resource group	3 weeks ago

See all

The data is ingested to Azure data lake storage (ADLS)

ecom-raw-data

Authentication method: Access key (Switch to Microsoft Entra user account)

Search blobs by prefix (case-sensitive)

Showing all 9 items

Name	Last modified	Access tier
brands	12/7/2025, 12:54:21 PM	
category	12/7/2025, 1:10:24 PM	
checkpoint	12/10/2025, 6:50:29 PM	
customers	12/7/2025, 12:55:03 PM	
date	12/7/2025, 12:56:38 PM	
order_items	12/7/2025, 12:57:45 PM	
order_returns	12/7/2025, 1:03:01 PM	
order_shipments	12/7/2025, 1:03:12 PM	
products	12/7/2025, 12:52:39 PM	

Bronze Layer (Raw data Ingestion):

- Ingested raw e-commerce data from ADLS into the Bronze layer.
- Historical data (January–August) was loaded as a one-time batch load.
- Incremental data (August–December) was ingested for ongoing processing.
- Fact tables (Order Items, Returns, Shipments) were ingested using Databricks Auto Loader to handle incremental data efficiently.
- Dimension tables (Customers, Products, Brands, Categories, Date) were ingested using batch loads without Auto Loader.
- Raw data was stored as-is, preserving the original schema for traceability.
- Checkpoints were enabled for fact tables to ensure fault tolerance and avoid duplicate ingestion.

Dimension Tables

Brands:

```
Dec 08, 2025 (<1s)                                                    13
# Load data using the schema defined
raw_data_path = f"/Volumes/{catalog_name}/raw/raw_landing/products/*.csv"

df = spark.read.option("header", "true")\
                .option("delimiter", ",")\
                .schema(products_schema)\
                .csv(raw_data_path) \
                .withColumn("_source_file", F.col("_metadata.file_path")) \
                .withColumn("ingest_timestamp", F.current_timestamp())
df:  pyspark.sql.connect.dataframe.DataFrame = [product_id: string, sku: string ... 13 more fields]
```

```
Dec 08, 2025 (3s)                                                     14
# Write raw data to the Bronze layer (catalog: ecommerce, schema: bronze, table: brz_products)
df.write.format("delta") \
    .mode("overwrite") \
    .option("mergeSchema", "true") \
    .saveAsTable(f"{catalog_name}.bronze.brz_products")
```

Other dimension tables (Customers, Products, Categories, Date) followed the same ingestion

Fact Tables

Order_items

```
spark.readStream \
 .format("cloudFiles") \
 .option("cloudFiles.format", "csv")  \
 .option("cloudFiles.schemaLocation", bronze_checkpoint_path) \
 .option("cloudFiles.schemaEvolutionMode", "rescue") \
 .option("header", "true") \
 .option("cloudFiles.inferColumnTypes", "true") \
 .option("rescuedDataColumn", "_rescued_data") \
 .option("cloudFiles.includeExistingFiles", "true")  \
 .option("pathGlobFilter", "*.csv") \
 .load(adls_path) \
 .withColumn("ingest_timestamp", F.current_timestamp()) \
 .withColumn("source_file", F.col("_metadata.file_path")) \
 .writeStream \
 .outputMode("append") \
 .option("checkpointLocation", bronze_checkpoint_path) \
 .trigger(availableNow=True) \
 .toTable(f"{catalog_name}.bronze.brz_order_items") \
 .awaitTermination()
```

```
# Checkpoint folders for streaming (bronze, silver, gold)
bronze_checkpoint_path = f"abfss://{container_name}@{storage_account_name}.dfs.core.windows.net/checkpoint/bronze/fact_order_items/"
```

Other
fact tables order returns and shipments also followed the same ingestion

Silver Layer (Data Cleaning):

- The Silver layer focuses on cleaning, standardizing, and validating raw data ingested from the Bronze layer.
- This layer ensures high data quality, schema consistency, and reliable incremental processing, making the data suitable for business enrichment and analytics in the Gold layer.

Tables Processed in Silver Layer

- Fact Order Items (Daily)

- Fact Order Returns (Monthly)
- Fact Order Shipments (Monthly)
- Dimension Tables (Brand, Product, Category, Customer, Date)

Fact Order Items(daily):

Fact Order returns (Monthly):

```
df = df.dropDuplicates(["order_id", "order_dt", "return_ts"])

#Convert `reason` column to uppercase and trim whitespace.
df = df.withColumn('reason', F.upper(F.col('reason'))).withColumn('reason', F.trim(F.col('reason')))

#Add `processed_time` column with the current timestamp
df = df.withColumn(
    "processed_time", F.current_timestamp()
)
```

Fact Order Shipments (Monthly):

```
df = df.dropDuplicates(["order_id", "order_dt", "shipment_id"])


# convert carrier column to uppercase and trim whitespace
df = df.withColumn('carrier', F.upper(F.col('carrier'))).withColumn('carrier', F.trim(F.col('carrier')))

#Add `processed_time` column with the current timestamp
df = df.withColumn(
    "processed_time", F.current_timestamp()
)
```

Incremental Processing Logic

- Silver layer uses UPSERT (MERGE) to avoid duplicate records
- Processes only new daily and monthly data
- Ensures idempotent and reliable pipeline execution

Fact order items (daily)

```
Dec 10, 2025 (<1s)                                                    13

def upsert_to_silver(microBatchDF, batchId):
    table_name = f"{catalog_name}.silver.slv_order_items"
    if not spark.catalog.tableExists(table_name):
        print("creating new table")
        microBatchDF.write.format("delta").mode("overwrite").saveAsTable(table_name)
        spark.sql(
            f"ALTER TABLE {table_name} SET TBLPROPERTIES (delta.enableChangeDataFeed = true)"
        )
    else:
        deltaTable = DeltaTable.forName(spark, table_name)
        deltaTable.alias("silver_table").merge(
            microBatchDF.alias("batch_table"),
            "silver_table.order_id = batch_table.order_id AND silver_table.item_seq = batch_table.item_seq",
        ).whenMatchedUpdateAll().whenNotMatchedInsertAll().execute()
```

```
Dec 10, 2025 (53s)                                                    14

df.writeStream.trigger(availableNow=True).foreachBatch(
    upsert_to_silver
).format("delta").option("checkpointLocation", silver_checkpoint_path).option(
    "mergeSchema", "true"
).outputMode(
    "update"
).trigger(
    once=True
```

Fact order returns (monthly)

```
Dec 12, 2025 (<1s)                                                    12

def upsert_to_silver(microBatchDF, batchId):
    table_name = f"{catalog_name}.silver.slv_order_returns"
    if not spark.catalog.tableExists(table_name):
        print("creating new table")
        microBatchDF.write.format("delta").mode("overwrite").saveAsTable(table_name)
        spark.sql(
            f"ALTER TABLE {table_name} SET TBLPROPERTIES (delta.enableChangeDataFeed = true)"
        )
    else:
        deltaTable = DeltaTable.forName(spark, table_name)
        deltaTable.alias("silver_table").merge(
            microBatchDF.alias("batch_table"),
            "silver_table.order_id = batch_table.order_id AND silver_table.order_dt = batch_table.order_dt AND silver_table.return_ts = batch_table.
            return_ts",
        ).whenMatchedUpdateAll().whenNotMatchedInsertAll().execute()
```

```
Dec 12, 2025 (6s)                                    13                Python

df.writeStream.trigger(availableNow=True).foreachBatch(
    upsert_to_silver
).format("delta").option("checkpointLocation", silver_checkpoint_path).option(
    "mergeSchema", "true"
).outputMode(
    "update"
).trigger(
```

Fact order shipments also follows same upsert strategy

Dimension Table:

- Performed basic cleansing and validation to ensure referential integrity
- Ensured schema consistency across all dimension tables
- Converted data into Delta format for optimized joins and BI consumption
- Applied a uniform transformation approach across all dimension tables

Brands:

```
# triming the extra spaces from brand_name
df_silver = df_bronze.withColumn('brand_name', F.trim(F.col('brand_name')))
df_silver.show(10)
```

```
df_silver = df_silver.withColumn('brand_code', F.regexp_replace(F.col('brand_code'), '[^a-zA-Z0-9]', ''))
df_silver.show(10)
```

The same cleaning and standardization logic was applied to other dimension tables such as Product, Customers, Category, and Date.

Gold Layer (BI Ready Tables)

- In the Gold layer, cleaned and standardized Silver data was transformed into business-ready dimension and fact tables optimized for analytics and Power BI reporting.
- This layer follows a star schema design, enabling fast queries, simplified reporting, and a single source of truth for business users.

Product Dimension (Brand + Category Combined):

- The Product dimension was created by combining product, brand, and category data into a single denormalized table.
- Unnecessary technical columns were removed, and only analytics-relevant attributes such as category, brand, color, size, and material were retained to support product-level insights.

```
WITH brands_categories AS (
  SELECT
    b.brand_name,
    b.brand_code,
    c.category_name,
    c.category_code
  FROM v_brands b
  INNER JOIN v_category c
  ON
    b.category_code = c.category_code
)
SELECT
  p.product_id,
  p.sku,
  p.category_code,
  COALESCE(bc.category_name, 'Not Available') AS category_name,
  p.brand_code,
  COALESCE(bc.brand_name, 'Not Available')   AS brand_name,
  p.color,
  p.size,
  p.material,
  p.weight_grams,
  p.length_cm,
  p.width_cm,
  p.height_cm,
  p.rating_count,
  p._source_file,
  p.ingest_timestamp
FROM v_products p
LEFT JOIN brands_categories bc
  ON p.brand_code = bc.brand_code;
```

Customer Dimension:

- The Customer dimension contains cleaned and enriched customer attributes including country, state, region, and contact details.
- This table enables geographic and customer segmentation analysis across sales, returns, and shipments.

```
    rows = []
    for country, states in country_state_map.items():
        for state_code, region in states.items():
            rows.append(Row(country=country, state=state_code, region=region))
    rows[:10]
df_gold = df_silver.join(df_region_mapping, on=['country', 'state'], how='left')
df_gold = df_gold.fillna({'region': 'Other'})
display(df_gold.limit(5))
```

Date Dimension:

- The Date dimension was created to support time-based analysis and includes attributes such as year, month, quarter, week, and weekend flag.
- A surrogate date_id was used to ensure consistent joins with all fact tables.

```
df_gold = df_silver.withColumn("date_id", F.date_format(F.col("date"), "yyyyMMdd").cast("int"))

# Add month name (e.g., 'January', 'February', etc.)
df_gold = df_gold.withColumn("month_name", F.date_format(F.col("date"), "MMMM"))

# Add is_weekend column
df_gold = df_gold.withColumn(
    "is_weekend",
    F.when(F.col("day_name").isin("Saturday", "Sunday"), 1).otherwise(0)
)

display(df_gold.limit(5))
```

```
desired_columns_order = ["date_id", "date", "year", "month_name", "day_name", "is_weekend", "quarter", "week", "_ingested_at", "_source_file"]
df_gold = df_gold.select(desired_columns_order)
display(df_gold.limit(5))
```

Fact Order Items (Incremental Upsert)

- Fact Order Items was implemented as an incremental fact table using Delta Lake upsert logic and checkpoints.
- This ensures daily data is processed without duplication while supporting historical and incremental loads efficiently.

```
def upsert_to_gold(microBatchDF, batchId):
    table_name = f"{catalog_name}.gold.gld_fact_order_items"
    if not spark.catalog.tableExists(table_name):
        print("creating new table")
        microBatchDF.write.format("delta").mode("overwrite").saveAsTable(table_name)
        spark.sql(
            f"ALTER TABLE {table_name} SET TBLPROPERTIES (delta.enableChangeDataFeed = true)"
        )
    else:
        deltaTable = DeltaTable.forName(spark, table_name)
        deltaTable.alias("gold_table").merge(
            microBatchDF.alias("batch_table"),
            "gold_table.transaction_id = batch_table.transaction_id AND gold_table.seq_no = batch_table.seq_no",
        ).whenMatchedUpdateAll().whenNotMatchedInsertAll().execute()

orders_gold_df.writeStream.trigger(availableNow=True).foreachBatch(
    upsert_to_gold
).format("delta").option("checkpointLocation", gold_checkpoint_path).option(
    "mergeSchema", "true"
).outputMode(
    "update"
).trigger(
    once=True
).start().awaitTermination()
```

Fact Returns & Shipments

Fact Returns and Fact Shipments were processed using the same incremental upsert and checkpointing strategy as Order Items, ensuring monthly data is ingested safely without reprocessing previous months.

The final Gold layer follows a **star schema** with three-dimension tables (Customer, Product, Date) and three fact tables (Order Items, Returns, Shipments), optimized for Power BI consumption and business reporting.

The Gold layer provides a scalable, analytics-ready data model that enables accurate reporting, faster insights, and supports future business growth.

Orchestration & Job Scheduling

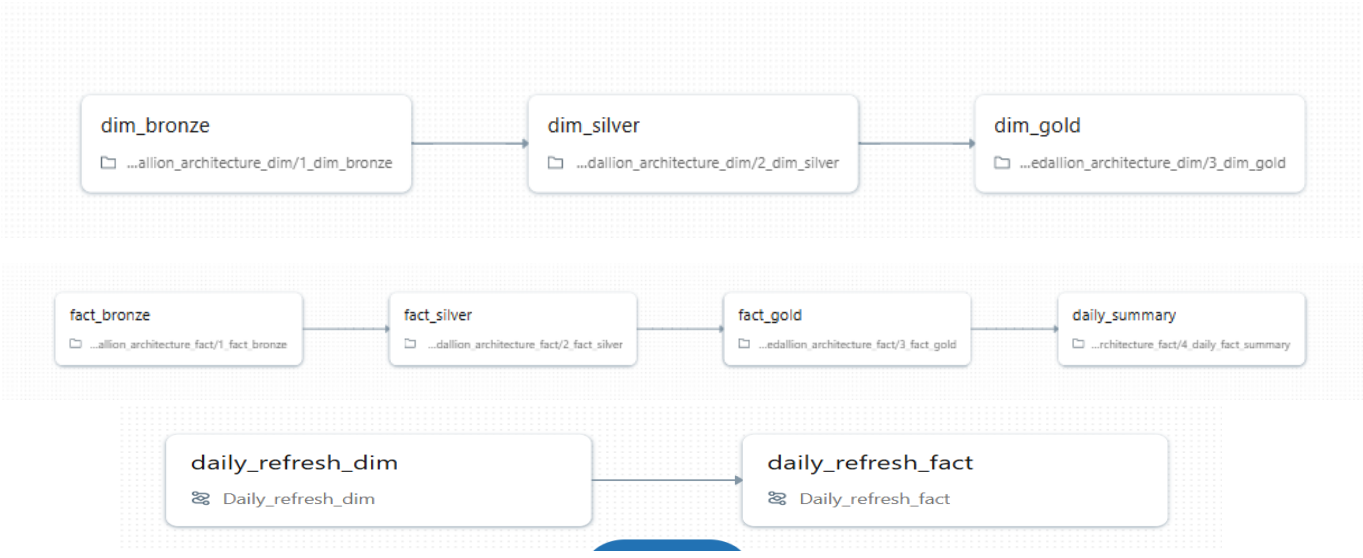
To ensure reliable and automated data processing, Databricks Jobs were used to orchestrate the end-to-end data pipelines across Bronze, Silver, and Gold layers.

Daily Job – Order Items Pipeline

A daily scheduled job was created for processing Order Items data, as this dataset is received every day.

Key details:

- Runs once per day
- Executes Bronze → Silver → Gold in sequence
- Uses Autoloader for ingestion
- Implements Delta upsert logic with checkpoints
- Ensures incremental daily data is processed without duplication



Monthly Job – Returns & Shipments Pipeline

A separate monthly scheduled job was created for processing Returns and Shipments data, as these datasets arrive on a monthly basis.

Key details:

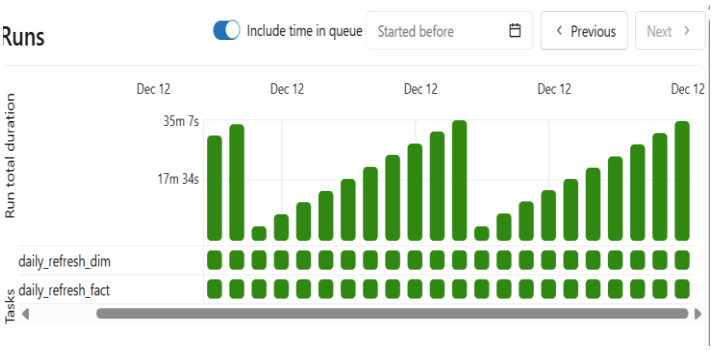
- Runs once every month (e.g., 1st day of the month)
- Processes only new monthly data
- Executes Bronze → Silver → Gold in the correct dependency order
- Uses incremental upsert logic to avoid reprocessing historical

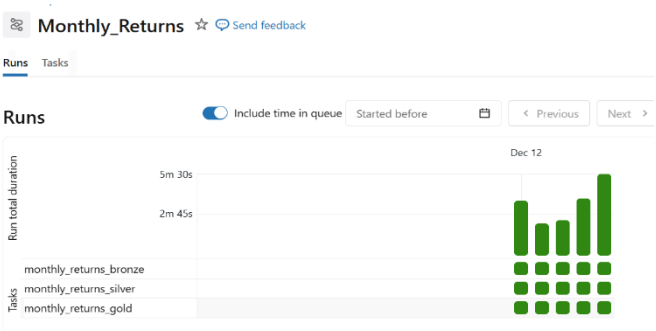
Dependency Management

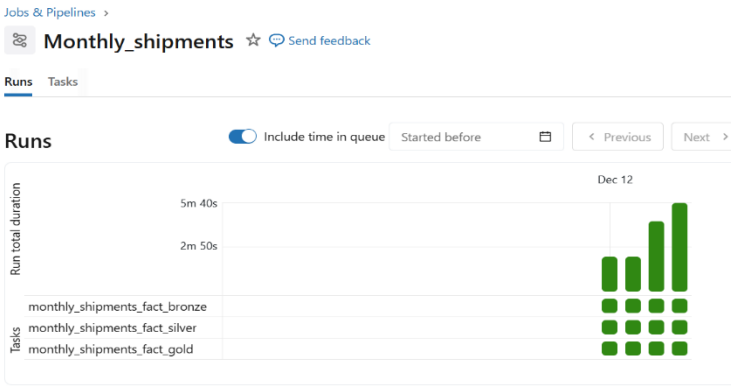
Task dependencies were configured to ensure:

- Bronze ingestion completes before Silver transformations
- Silver data is finalized before Gold enrichment
- Fact tables are updated only after relevant dimension tables are refreshed

This guarantees data consistency and reliability across all reporting layers.

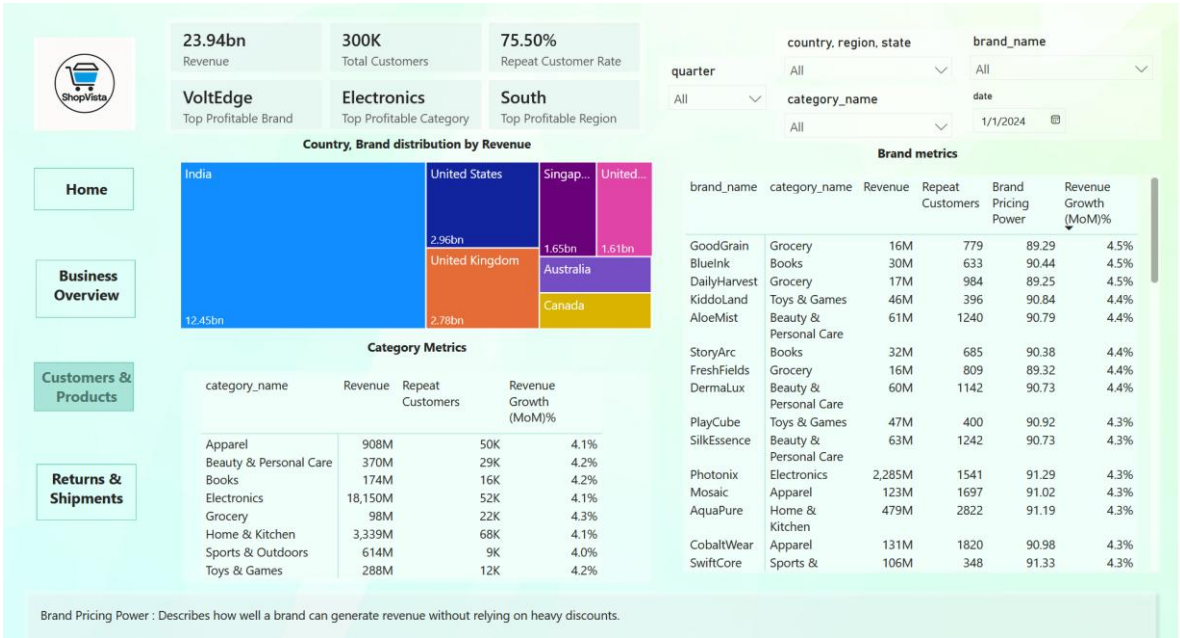






Power BI Integration & Reporting

- The Gold layer tables were connected to Power BI to enable business reporting and analytics using a star schema model.
- A star schema was implemented in Power BI with Customer, Product, and Date as dimension tables and Order Items, Returns, and Shipments as fact tables.
- Business-focused KPIs and visuals were created to analyze revenue, customer behavior, product performance, returns, and shipment efficiency.
- Business-focused KPIs and visuals were created to analyze revenue, customer behavior, product performance, returns, and shipment efficiency.
- The report provides actionable insights and supports data-driven decision-making.



Final Outcome

- Built a centralized and scalable data platform using Azure Data Lake and Databricks.
- Implemented automated daily and monthly data pipelines with incremental processing.
- Established a single source of truth through clean, business-ready Gold tables.
- Enabled fast and reliable business reporting through Power BI dashboards.
- Improved visibility into sales, customers, products, returns, and shipment performance.
- Created a future-ready data foundation that can easily scale with business growth.