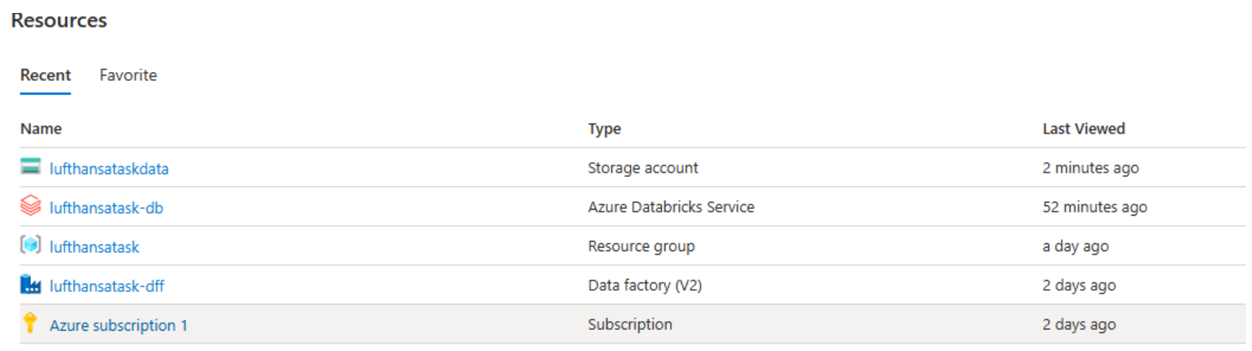
CLOUD SOLUTION

Data Pipeline Task Documentation

Contents

1. Introduction

Data pipeline task was implemented in Azure. The solution involved using Azure services, such as Storage Account, Data Factory, Databricks, to process and manage data, transform and then Azure SQL to validate. The data flow was organized into three layers: Bronze, Silver, and Gold.

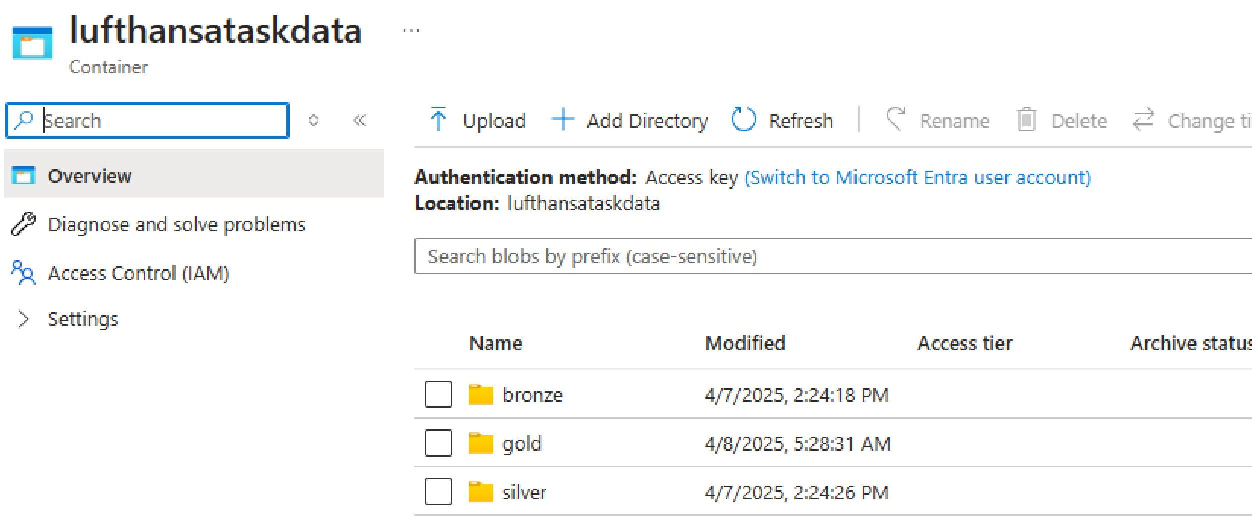


Name	Type	Last Viewed
lufthansataskdata	Storage account	2 minutes ago
lufthansatask-db	Azure Databricks Service	52 minutes ago
lufthansatask	Resource group	a day ago
lufthansatask-dff	Data factory (V2)	2 days ago
Azure subscription 1	Subscription	2 days ago

2. Setting Up Storage

First, I created a storage account in Azure. This account is where all the data for the task will be saved. Inside the storage account, I created a container to keep the data organized. Then, I made three directories inside the container(medallion architecture):

Bronze: For raw data, which is the data straight from the source.
Silver: For transformed data, which is cleaned and processed.
Gold: For the final data, ready for reporting and analysis.



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

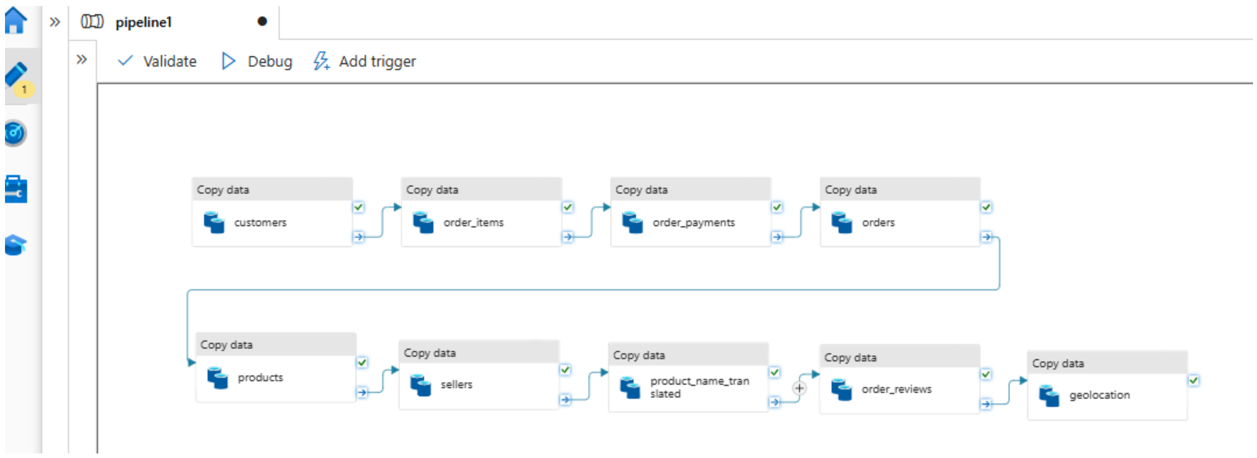
Search blobs by prefix (case-sensitive)

Name	Modified	Access tier	Archive statu
bronze	4/7/2025, 2:24:18 PM		
gold	4/8/2025, 5:28:31 AM		
silver	4/7/2025, 2:24:26 PM		

3. Building a Pipeline

Next, I created a pipeline in Azure Data Factory. The pipeline's job is to move and transform the data.

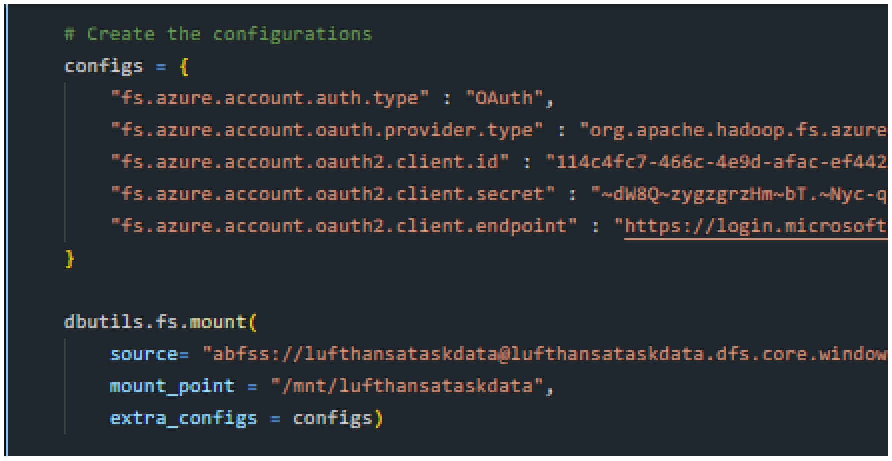
First, I uploaded the raw data to GitHub and got its URL. In the pipeline, I linked the URL from GitHub to fetch the raw data. The pipeline saved the raw data in the Bronze directory of the storage account.



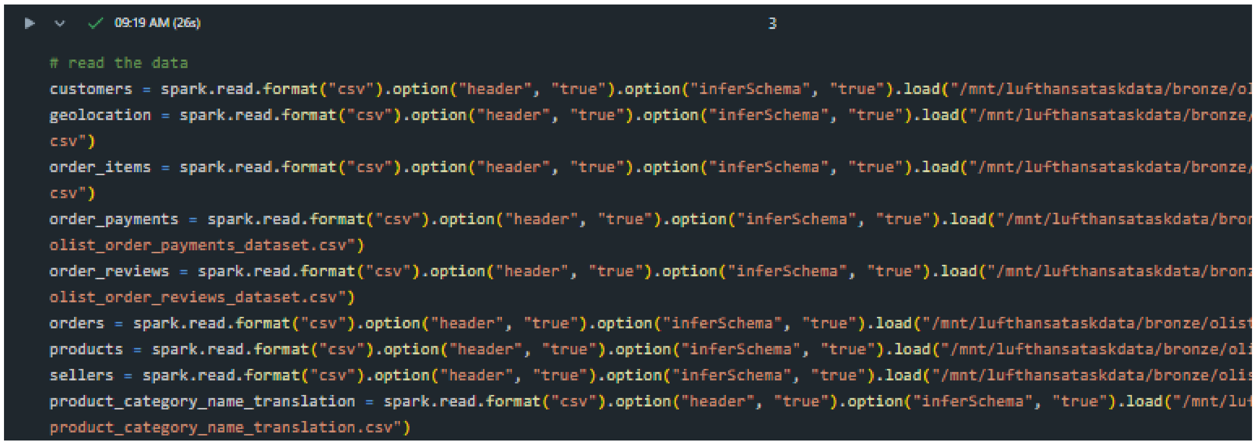
4. Setting Up Databricks for Transformation

After loading the raw data, I set up Databricks for transformations. I created a compute resource in Databricks to run tasks. I made a new app registration in Azure. This step connected the storage account with Databricks, allowing Databricks to access the data. I created a Databricks workspace. This workspace is where I configured everything to work with the data.

Once the workspace was ready, I started working on the data. I read the data stored in the Bronze directory. This is the raw data from the storage account. Next, I checked the data quality for each table in the Bronze folder. I inspected the data for issues like missing values, duplicates, and incorrect types.



```
# Create the configurations
configs = {
    "fs.azure.account.auth.type" : "OAuth",
    "fs.azure.account.oauth.provider.type" : "org.apache.hadoop.fs.azure
    "fs.azure.account.oauth2.client.id" : "114c4fc7-466c-4e9d-afac-ef442
    "fs.azure.account.oauth2.client.secret" : "~dW8Q~zygzgrzHm~bT.~Nyc-q
    "fs.azure.account.oauth2.client.endpoint" : "https://login.microsoft
}

dbutils.fs.mount(
    source= "abfss://lufthansataskdata@lufthansataskdata.dfs.core.window
    mount_point = "/mnt/lufthansataskdata",
    extra_configs = configs)
```



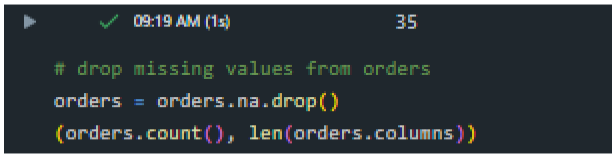
```
# read the data
customers = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("/mnt/lufthansataskdata/bronze/ol
geolocation = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("/mnt/lufthansataskdata/bronze/
csv")
order_items = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("/mnt/lufthansataskdata/bronze/
csv")
order_payments = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("/mnt/lufthansataskdata/bron
olist_order_payments_dataset.csv")
order_reviews = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("/mnt/lufthansataskdata/bron
olist_order_reviews_dataset.csv")
orders = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("/mnt/lufthansataskdata/bronze/olist
products = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("/mnt/lufthansataskdata/bronze/oli
sellers = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("/mnt/lufthansataskdata/bronze/olis
product_category_name_translation = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("/mnt/lu
product_category_name_translation.csv")
```

5. Data processing

I started processing the data by cleaning it and handling missing values:

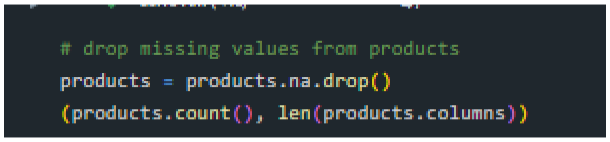
5.1 Drop Missing Values:

In the orders dataset, I removed all rows that had missing values.



```
# drop missing values from orders
orders = orders.na.drop()
(orders.count(), len(orders.columns))
```

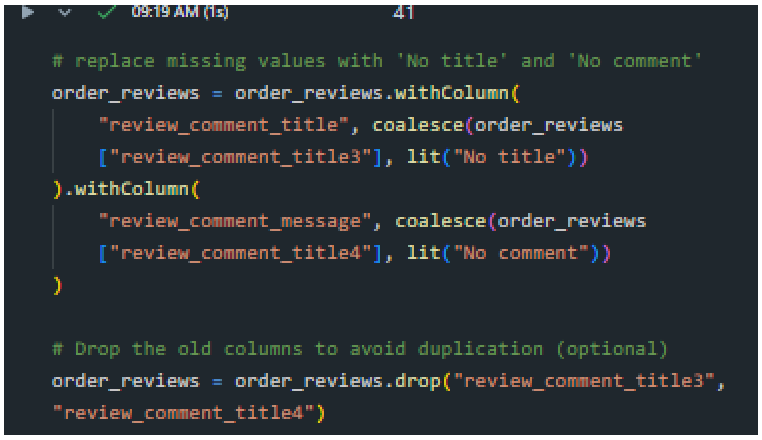
In the products dataset, I also removed rows with missing values.



```
# drop missing values from products
products = products.na.drop()
(products.count(), len(products.columns))
```

5.2 Replace Missing Values in Reviews:

In the order_reviews dataset, I replaced missing values in the columns review_comment_title and review_comment_message with "No title" and "No comment," respectively.



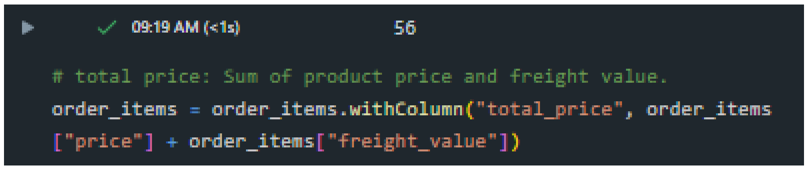
```
09:19 AM (1s)          41

# replace missing values with 'No title' and 'No comment'
order_reviews = order_reviews.withColumn(
    "review_comment_title", coalesce(order_reviews
    ["review_comment_title3"], lit("No title"))
).withColumn(
    "review_comment_message", coalesce(order_reviews
    ["review_comment_title4"], lit("No comment"))
)

# Drop the old columns to avoid duplication (optional)
order_reviews = order_reviews.drop("review_comment_title3",
"review_comment_title4")
```

6. Calculated Columns

After cleaning the data, I created several new columns for analysis:

6.1 Total Price

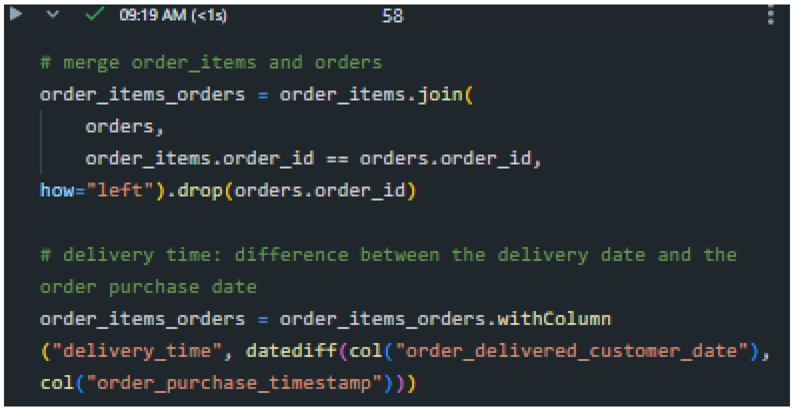
I calculated the total price for each order item by adding the product price and freight value.



```
09:19 AM (<1s)          56

# total price: Sum of product price and freight value.
order_items = order_items.withColumn("total_price", order_items
["price"] + order_items["freight_value"])
```

6.2 Delivery Time

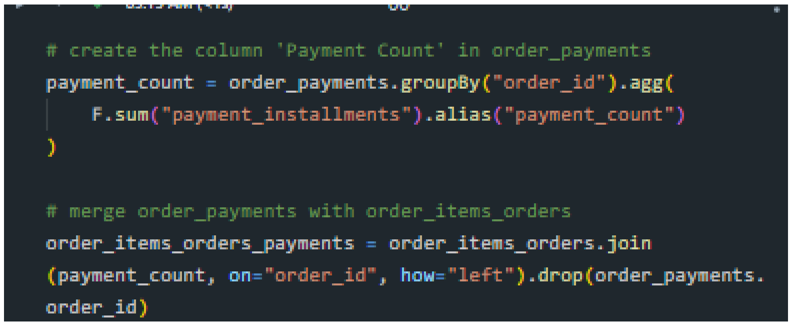
I calculated the number of days it took to deliver each order. To do this, I merged order_items with orders and found the difference between the delivery and purchase dates.


09:19 AM (<1s) 58

```
# merge order_items and orders
order_items_orders = order_items.join(
    orders,
    order_items.order_id == orders.order_id,
how="left").drop(orders.order_id)

# delivery time: difference between the delivery date and the
order purchase date
order_items_orders = order_items_orders.withColumn
("delivery_time", datediff(col("order_delivered_customer_date"),
col("order_purchase_timestamp")))
```

6.3 Payment Count

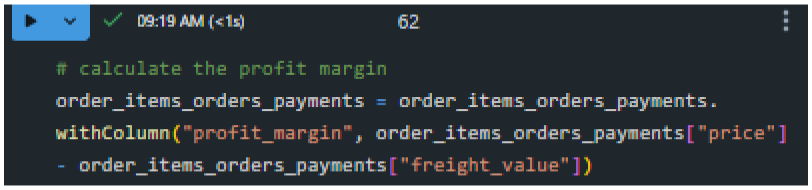
I calculated the total number of payment installments for each order.



```
# create the column 'Payment Count' in order_payments
payment_count = order_payments.groupBy("order_id").agg(
    F.sum("payment_installments").alias("payment_count")
)

# merge order_payments with order_items_orders
order_items_orders_payments = order_items_orders.join
(payment_count, on="order_id", how="left").drop(order_payments.
order_id)
```

6.4 Profit Margin

I calculated the profit margin by subtracting the freight value from the product price.


09:19 AM (<1s) 62

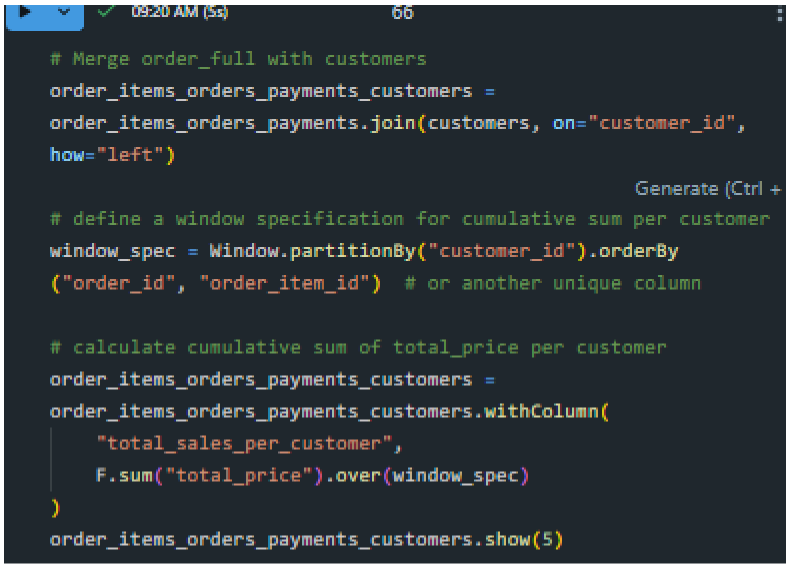
```
# calculate the profit margin
order_items_orders_payments = order_items_orders_payments.
withColumn("profit_margin", order_items_orders_payments["price"]
- order_items_orders_payments["freight_value"])
```

7. Window Functions Over Partitions and Merges

7.1 Total Sales per Customer

I calculated a running total of the product prices for each customer. This helps track how much each customer has spent over their orders.

I merged the order_items_orders_payments dataset with the customers dataset using the customer_id column. This added customer details to the data. I defined a window specification using Window.partitionBy() to group the data by customer_id. Finally, I applied F.sum() with .over() to calculate the cumulative sum of the total_price column for each customer.

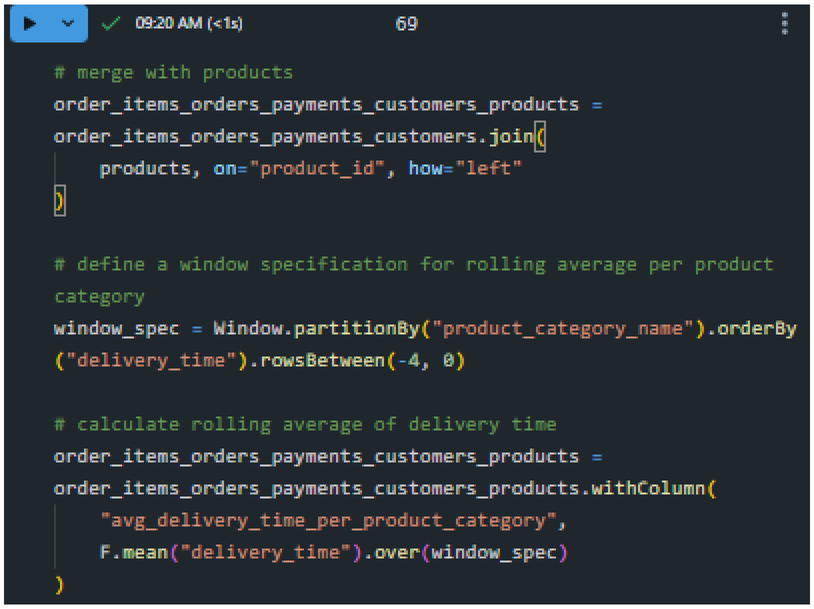


```
# Merge order_full with customers
order_items_orders_payments_customers =
order_items_orders_payments.join(customers, on="customer_id",
how="left")

# define a window specification for cumulative sum per customer
window_spec = Window.partitionBy("customer_id").orderBy
("order_id", "order_item_id")  # or another unique column

# calculate cumulative sum of total_price per customer
order_items_orders_payments_customers =
order_items_orders_payments_customers.withColumn(
    "total_sales_per_customer",
    F.sum("total_price").over(window_spec)
)
order_items_orders_payments_customers.show(5)
```

7.2 Average Delivery Time per Product Category

I calculated a rolling average of delivery time for each product category. This shows the average time it takes to deliver products within each category.

I merged the order_items_orders_payments_customers dataset with the products dataset using the product_id column. This added product details to the data. I defined a window specification using Window.partitionBy() to group the data by product_category_name. I applied F.mean() with .over() and specified a rolling window of 5 rows to calculate the rolling average of delivery times.

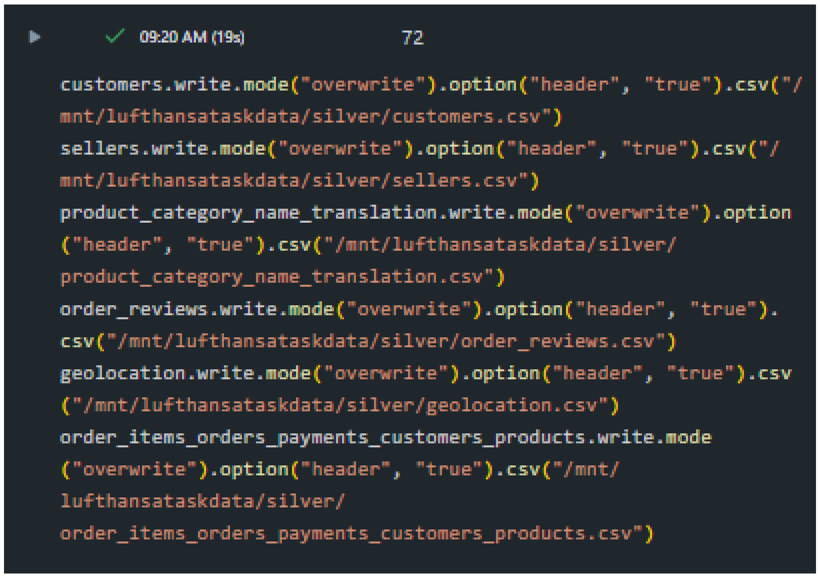


```
09:20 AM (<1s)          69

# merge with products
order_items_orders_payments_customers_products =
order_items_orders_payments_customers.join(
    products, on="product_id", how="left"
)

# define a window specification for rolling average per product
category
window_spec = Window.partitionBy("product_category_name").orderBy
("delivery_time").rowsBetween(-4, 0)

# calculate rolling average of delivery time
order_items_orders_payments_customers_products =
order_items_orders_payments_customers_products.withColumn(
    "avg_delivery_time_per_product_category",
    F.mean("delivery_time").over(window_spec)
)
```

8. Moving Data to Silver Layer. Preparing Data for the Gold Layer

8.1 Moving Transformed Data to the Silver Layer

Once the data transformations were complete, I moved the transformed data into the Silver directory. This layer holds clean and structured data that is ready for further processing and analysis.

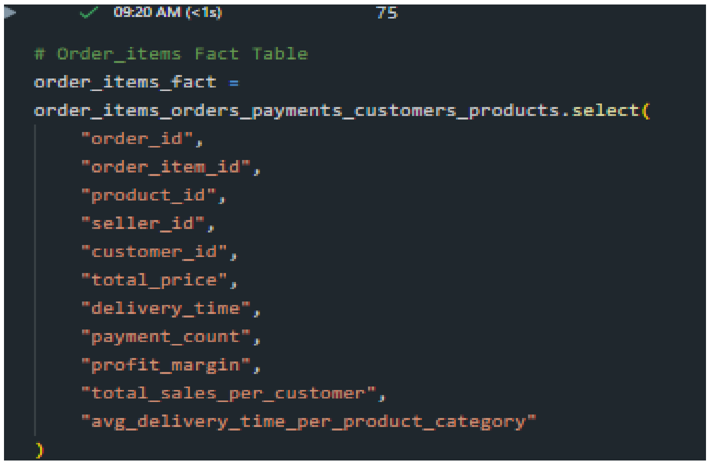


```
09:20 AM (19s)          72

customers.write.mode("overwrite").option("header", "true").csv("/
mnt/lufthansataskdata/silver/customers.csv")
sellers.write.mode("overwrite").option("header", "true").csv("/
mnt/lufthansataskdata/silver/sellers.csv")
product_category_name_translation.write.mode("overwrite").option
("header", "true").csv("/mnt/lufthansataskdata/silver/
product_category_name_translation.csv")
order_reviews.write.mode("overwrite").option("header", "true").
csv("/mnt/lufthansataskdata/silver/order_reviews.csv")
geolocation.write.mode("overwrite").option("header", "true").csv
("/mnt/lufthansataskdata/silver/geolocation.csv")
order_items_orders_payments_customers_products.write.mode
("overwrite").option("header", "true").csv("/mnt/
lufthansataskdata/silver/
order_items_orders_payments_customers_products.csv")
```

8.2 Splitting Data into Fact and Dimension Tables

To prepare the data for the Gold layer, I divided the transformed data into one fact table and several dimension tables. These tables organize the data for efficient reporting and analysis:

Fact Table: The fact table contains detailed information about order items, including calculated columns such as: Total Price, Delivery Time, Payment Count, Profit Margin **etc**



```
09:20 AM (<1s)                75

# Order_items Fact Table
order_items_fact =
order_items_orders_payments_customers_products.select(
    "order_id",
    "order_item_id",
    "product_id",
    "seller_id",
    "customer_id",
    "total_price",
    "delivery_time",
    "payment_count",
    "profit_margin",
    "total_sales_per_customer",
    "avg_delivery_time_per_product_category"
)
```

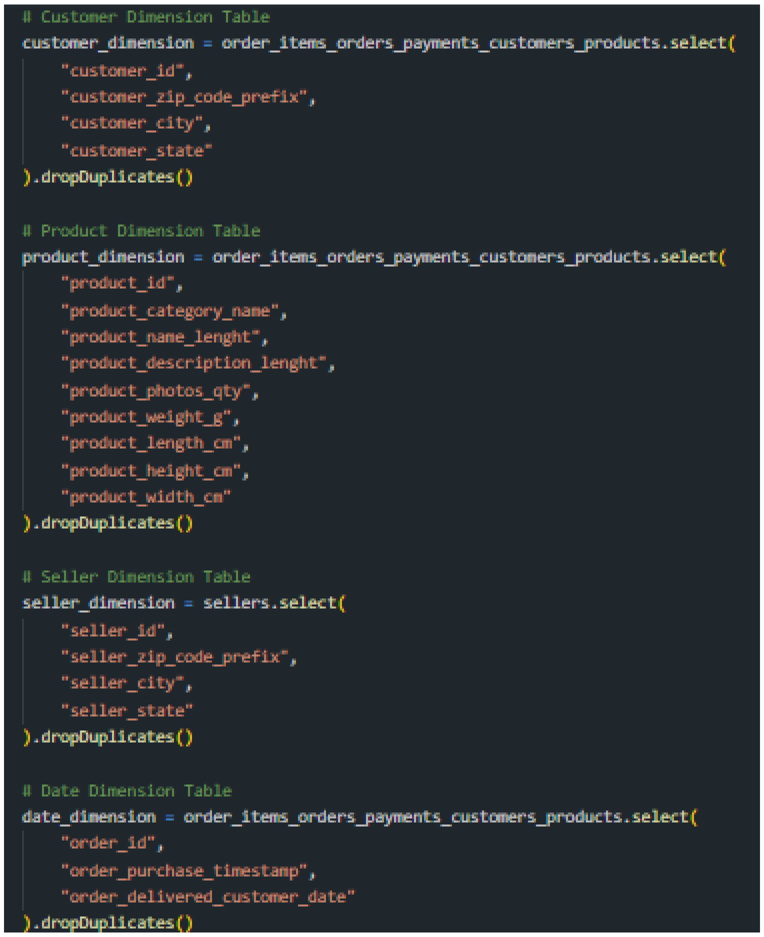
Dimension Tables:

Customers: This table includes customer details like: Customer ID, Zip Code Prefix, City, State

Products: This table contains product-related details, including: Product Name, Product Category, Product Dimensions and Weight

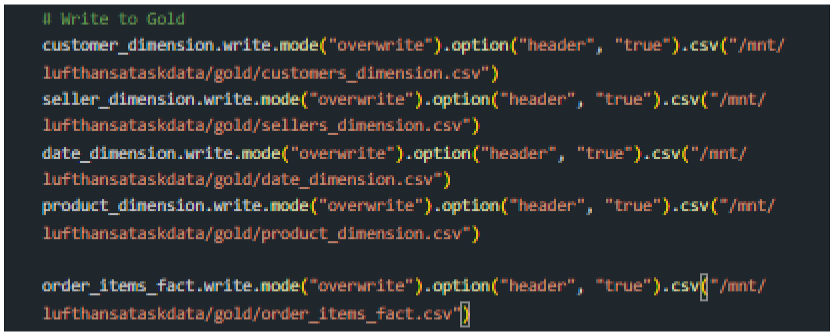
Sellers: This table holds seller details, including: Seller ID, Geographic Information (City, State)

Date: This table represents time-related details, including: Order Purchase Date, Order Delivery Date


```
# Customer Dimension Table
customer_dimension = order_items_orders_payments_customers_products.select(
    "customer_id",
    "customer_zip_code_prefix",
    "customer_city",
    "customer_state"
).dropDuplicates()

# Product Dimension Table
product_dimension = order_items_orders_payments_customers_products.select(
    "product_id",
    "product_category_name",
    "product_name_lenght",
    "product_description_lenght",
    "product_photos_qty",
    "product_weight_g",
    "product_length_cm",
    "product_height_cm",
    "product_width_cm"
).dropDuplicates()

# Seller Dimension Table
seller_dimension = sellers.select(
    "seller_id",
    "seller_zip_code_prefix",
    "seller_city",
    "seller_state"
).dropDuplicates()

# Date Dimension Table
date_dimension = order_items_orders_payments_customers_products.select(
    "order_id",
    "order_purchase_timestamp",
    "order_delivered_customer_date"
).dropDuplicates()
```

8.3 Writing the Fact Table

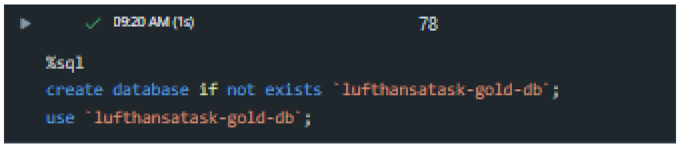
After preparing the fact and dimension tables, I wrote the processed data into the Gold layer. This layer stores the final cleaned, transformed, and structured data for reporting and analysis.


```
# Write to Gold
customer_dimension.write.mode("overwrite").option("header", "true").csv("/mnt/
lufthansataskdata/gold/customers_dimension.csv")
seller_dimension.write.mode("overwrite").option("header", "true").csv("/mnt/
lufthansataskdata/gold/sellers_dimension.csv")
date_dimension.write.mode("overwrite").option("header", "true").csv("/mnt/
lufthansataskdata/gold/date_dimension.csv")
product_dimension.write.mode("overwrite").option("header", "true").csv("/mnt/
lufthansataskdata/gold/product_dimension.csv")

order_items_fact.write.mode("overwrite").option("header", "true").csv("/mnt/
lufthansataskdata/gold/order_items_fact.csv")
```

9. SQL Validation and Reporting

9.1 Creating a Database

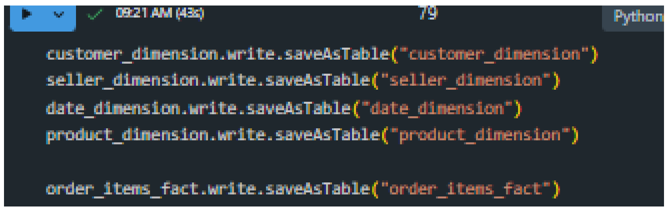
To validate and report on the data stored in the Gold layer, I first created a database in SQL. This database will hold the tables created from the Gold layer data.



```
09:20 AM (1s)                78

%sql
create database if not exists `lufthansatask-gold-db`;
use `lufthansatask-gold-db`;
```

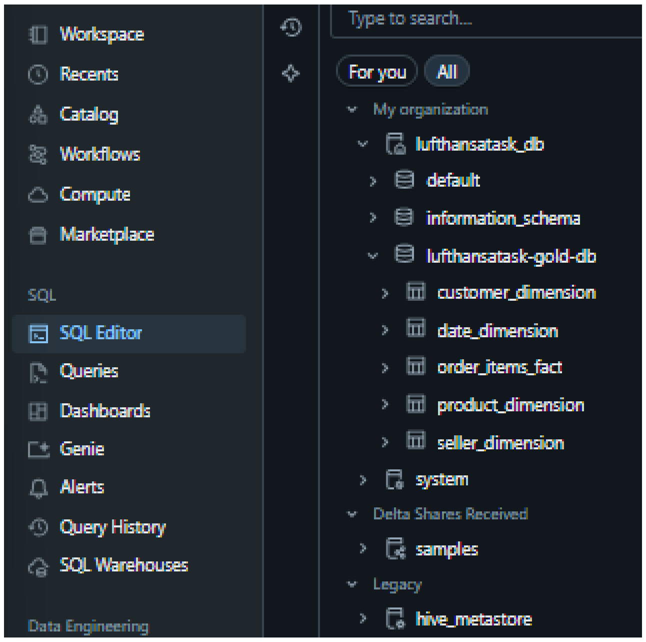
9.2 Loading Gold Layer Data into the Database

Next, I wrote the data from the Gold layer into the newly created database as tables. The data is organized into one fact table and several dimension tables



```
09:21 AM (43s)               79                    Python

customer_dimension.write.saveAsTable("customer_dimension")
seller_dimension.write.saveAsTable("seller_dimension")
date_dimension.write.saveAsTable("date_dimension")
product_dimension.write.saveAsTable("product_dimension")

order_items_fact.write.saveAsTable("order_items_fact")
```

9.3 Validating



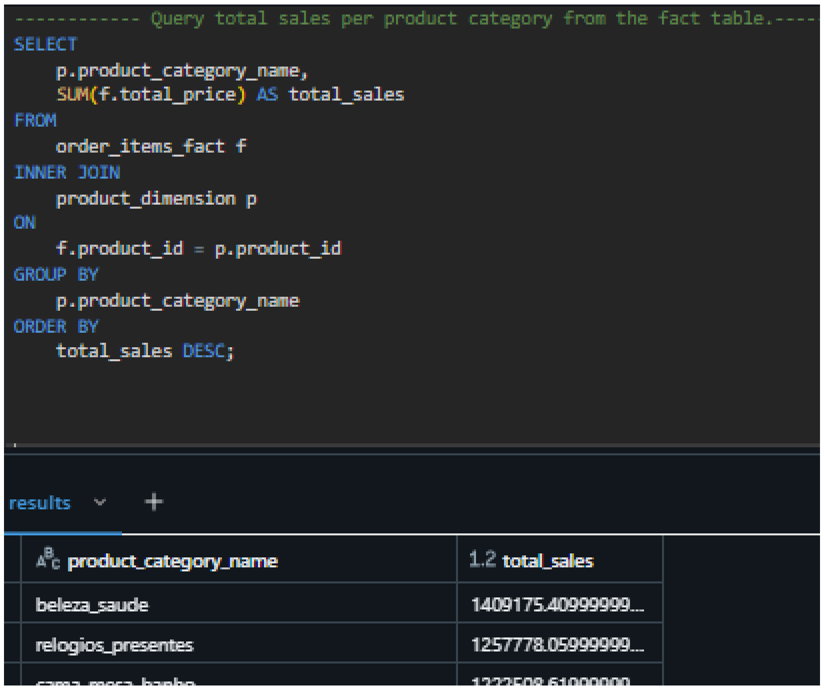
```
My organization
  lufthansatask_db
    default
    information_schema
    lufthansatask-gold-db
      customer_dimension
      date_dimension
      order_items_fact
      product_dimension
      seller_dimension
  system
Delta Shares Received
  samples
Legacy
  hive_metastore
```

9.4 Reporting

I wrote queries to check and validate the data. These queries also provide meaningful insights, such as total sales, average delivery time, and order distribution by states.

9.4.1 Total Sales per Product Category

This query calculates the total sales for each product category by summing up the total_price column. The data comes from the fact table joined with the product dimension table.

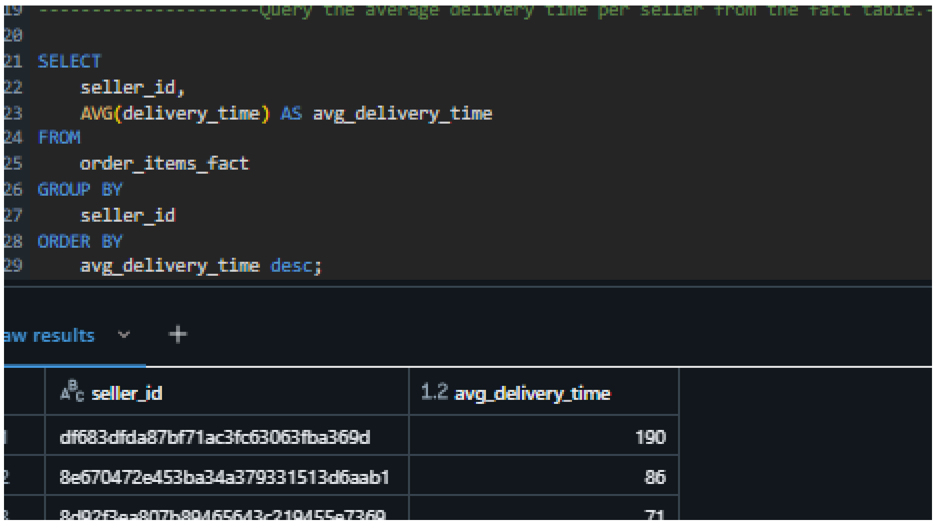


```sql
------------ Query total sales per product category from the fact table.----
SELECT
    p.product_category_name,
    SUM(f.total_price) AS total_sales
FROM
    order_items_fact f
INNER JOIN
    product_dimension p
ON
    f.product_id = p.product_id
GROUP BY
    p.product_category_name
ORDER BY
    total_sales DESC;
```

product_category_name	total_sales
beleza_saude	1409175.40999999...
relogios_presentes	1257778.05999999...
cama_mesa_banho	1222508.61000000

9.4.2 Average Delivery Time per Seller

This query calculates the average delivery time for each seller by grouping data in the fact table.



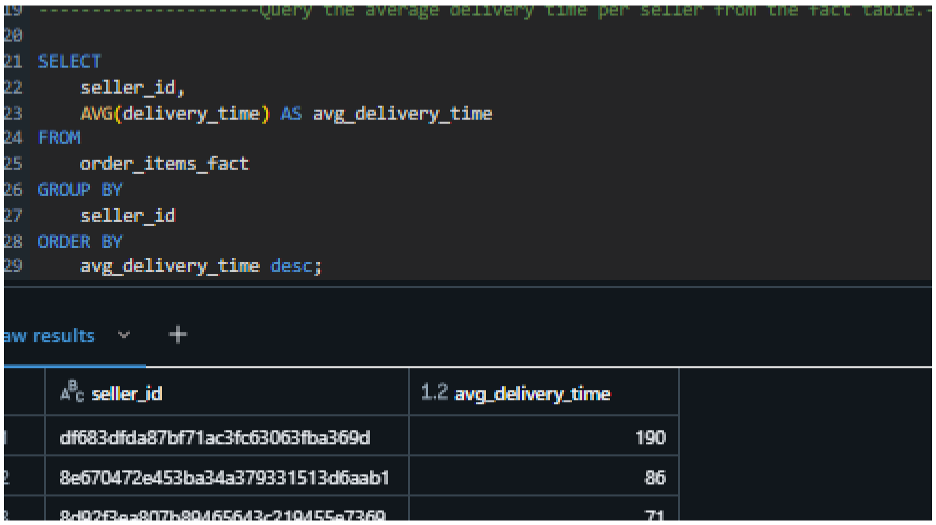
```sql
---------------------Query the average delivery time per seller from the fact table.-

SELECT
    seller_id,
    AVG(delivery_time) AS avg_delivery_time
FROM
    order_items_fact
GROUP BY
    seller_id
ORDER BY
    avg_delivery_time desc;
```

aw results

	seller_id	1.2 avg_delivery_time
1	df683dfda87bf71ac3fc63063fba369d	190
2	8e670472e453ba34a379331513d6aab1	86
3	8d92f3ea807b89465643c219455e7369	71

9.4.3 Number of Orders from Each State

This query counts the number of orders made by customers from each state. The data comes from the fact table joined with the customer dimension table.



```sql
---------------------Query the average delivery time per seller from the fact table.-

SELECT
    seller_id,
    AVG(delivery_time) AS avg_delivery_time
FROM
    order_items_fact
GROUP BY
    seller_id
ORDER BY
    avg_delivery_time desc;
```

aw results

	seller_id	1.2 avg_delivery_time
1	df683dfda87bf71ac3fc63063fba369d	190
2	8e670472e453ba34a379331513d6aab1	86
3	8d92f3ea807b89465643c219455e7369	71