

End-to-End Data Ingestion and Transformation through Azure Data Engineering

A Real-World Data Pipeline Project

Presented by:

Arpan Ray, Mainak Mukherjee, Trishita Mukherjee

TABLE OF CONTENT

- Introduction & Objectives
- Architecture & Workflow
- Resource Setup
- Data Ingestion (ADF)
- Transformations in Databricks
- Results, Future Scope
- Conclusion



INTRODUCTION

This project showcases an end-to-end data engineering pipeline using Microsoft Azure, Databricks, and PySpark. Data from multiple sources is ingested through Azure Data Factory, then cleaned, transformed, and processed in Azure Databricks using PySpark. The processed data is stored in Delta Lake, ensuring optimized performance and reliability. By leveraging the orchestration and automation features of ADF, the pipeline achieves scalability, security, and efficiency, making it a robust solution for modern data analytics and business intelligence needs.

PROJECT OVERVIEW

- Build an automated data pipeline using Azure Data Factory and Databricks
- Transform data across Bronze, Silver, and Gold layers

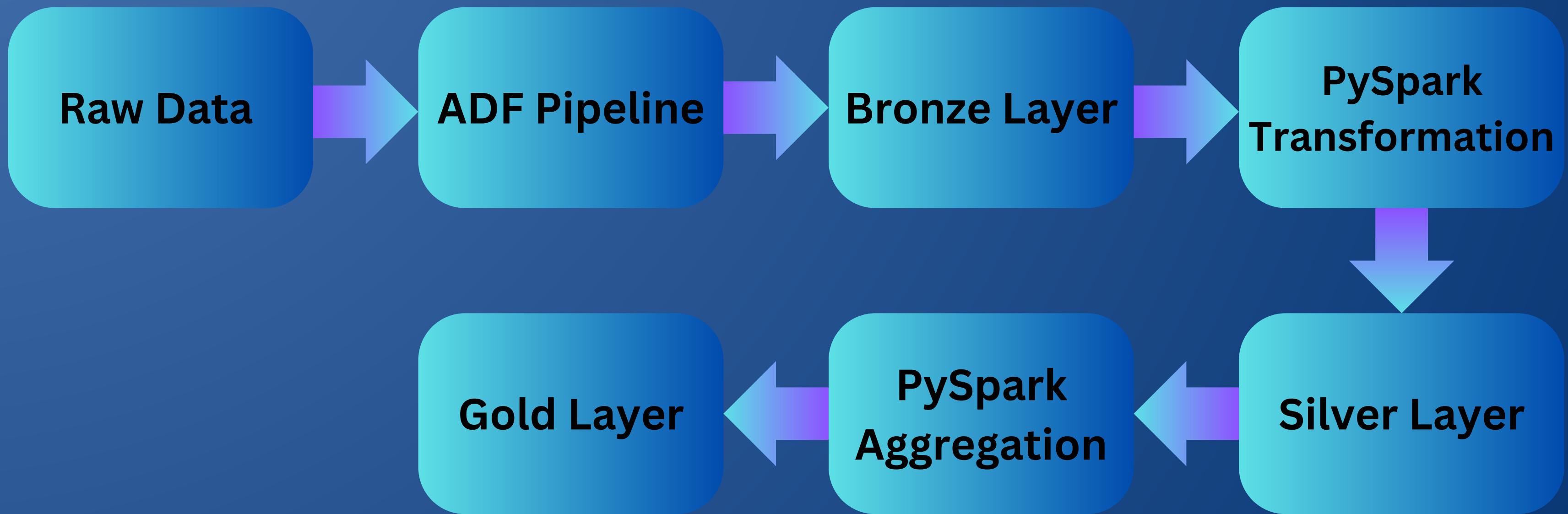
OBJECTIVES

- Build a scalable data pipeline to ingest raw data from diverse sources using Azure Data Factory.
- Transform and clean data efficiently with PySpark in Azure Databricks.
- Optimize data storage and retrieval by leveraging Delta Lake on Azure.
- Automate and orchestrate workflows with ADF for seamless data movement and processing.
- Ensure security, scalability, and reliability across the entire data engineering lifecycle.
- Enable advanced analytics and reporting through well-structured, high-quality data.

TECHNOLOGIES & TOOLS USED

- Azure Data Factory (ADF) - Data ingestion & orchestration
- Azure Databricks - Transformation with PySpark
- Delta Lake - Optimized storage
- Azure Storage - Raw, Bronze, Silver, Gold containers
- Azure AD - Security and Role Assignment

PROJECT WORKFLOW



KEY FEATURES

- Scalability: Handles large data volumes
- Automation: End-to-end pipeline automation
- Reliability: Delta Lake ensures ACID transactions
- Analytics-ready: High quality data for BI tools

RESOURCE GROUP CREATION

Home >

Create a resource group

Basics Tags Review + create

Resource group - A container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group. You decide how you want to allocate resources to resource groups based on what makes the most sense for your organization. [Learn more](#)

Subscription * ⓘ Azure subscription 1

Resource group name * ⓘ FinalProj0

Region * ⓘ (Asia Pacific) Central India

Previous Next Review + create

STORAGE ACCOUNT CREATION

Home > Resource groups > FinalProj0 > Marketplace > Storage account >

Create a storage account ...

Select the subscription in which to create the new storage account. Choose a new or existing resource group to organize and manage your storage account together with other resources.

Subscription *: Azure subscription 1

Resource group *: FinalProj0
Create new

Instance details

Storage account name *: strgaccf0

Region *: (Asia Pacific) Central India
Deploy to an Azure Extended Zone

Preferred storage type: Choose preferred storage type

This helps us provide relevant guidance. It doesn't restrict your storage to this resource type. [Learn more](#)

Performance *: Standard: Recommended for most scenarios (general-purpose v2 account)
Premium: Recommended for scenarios that require low latency.

Redundancy *: Locally-redundant storage (LRS)

Previous Next Review + create

Fig1: Storage Account Creation

Home > Resource groups > FinalProj0 > Marketplace > Storage account >

Create a storage account ...

Basics Advanced Networking Data protection Encryption Tags Review + create

Security

Configure security settings that impact your storage account.

Require secure transfer for REST API operations:

Allow enabling anonymous access on individual containers:

Enable storage account key access:

Default to Microsoft Entra authorization in the Azure portal:

Minimum TLS version: Version 1.2

Permitted scope for copy operations (preview): From any storage account

Hierarchical Namespace

Hierarchical namespace, complemented by Data Lake Storage Gen2 endpoint, enables file and directory semantics, accelerates big data analytics workloads, and enables access control lists (ACLs). [Learn more](#)

Enable hierarchical namespace:

Previous Next Review + create

Fig2 : Storage Account Creation

Home > Resource groups > FinalProj0 > Marketplace > Storage account >

Create a storage account ...

big data analytics workloads, and enables access control lists (ACLs). [Learn more](#)

Enable hierarchical namespace:

Access protocols

Blob and Data Lake Gen2 endpoints are provisioned by default. [Learn more](#)

Enable SFTP:

Enable network file system v3:

Blob storage

Allow cross-tenant replication:

Cross-tenant replication and hierarchical namespace cannot be enabled simultaneously.

Access tier: Hot: Optimized for frequently accessed data and everyday usage scenarios
Cool: Optimized for infrequently accessed data and backup scenarios
Cold: Optimized for rarely accessed data and backup scenarios

Azure Files

Enable large file shares:

Previous Next Review + create

Fig3: Storage Account Creation

CREATION OF REQUIRED CONTAINERS

The screenshot shows the 'Containers' blade for the storage account 'strgaccf0'. The left sidebar includes links for Overview, Activity log, Tags, Diagnose and solve problems, Access Control (IAM), Data migration, Events, Storage browser, Partner solutions, Resource visualizer, Data storage (with Containers selected), File shares, Queues, Tables, Security + networking, and Data management. The main area displays a table of containers:

Name	Last modified	Anonymous access level	Lease state
\$logs	8/29/2025, 12:41:28 AM	Private	Available
bronze	8/29/2025, 12:43:01 AM	Private	Available
gold	8/29/2025, 12:43:14 AM	Private	Available
raw	8/29/2025, 12:42:52 AM	Private	Available
silver	8/29/2025, 12:43:09 AM	Private	Available

UPLOADING PARAMETER CONTAINED JSON FILE

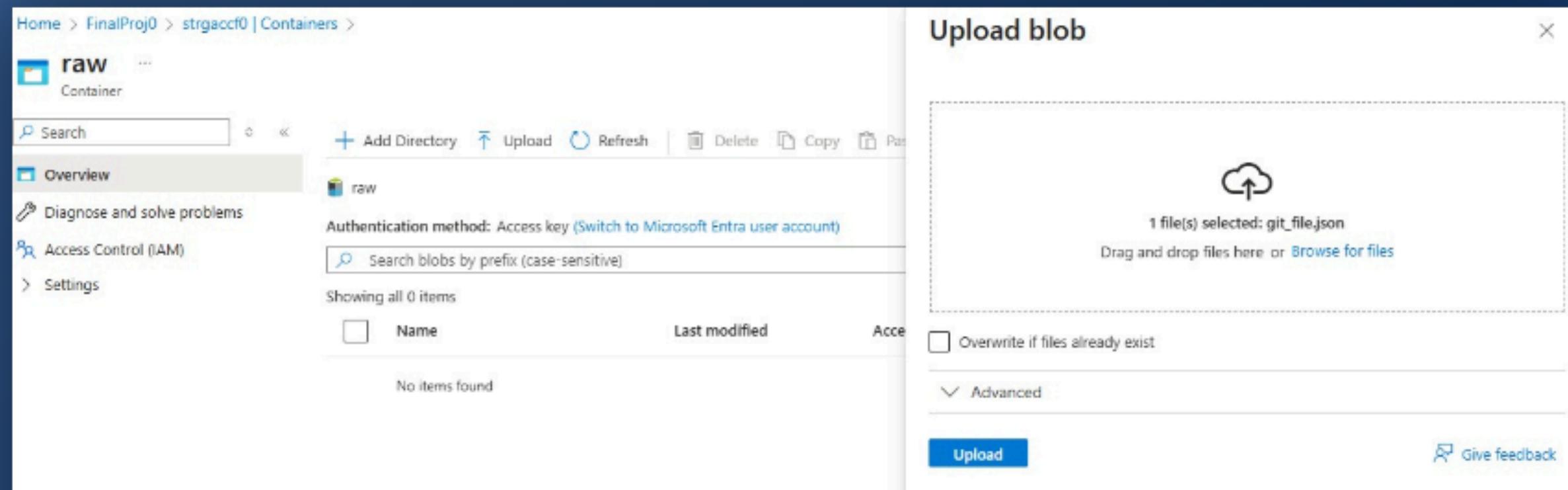


Fig1: Uploading Parameter Contained JSON File

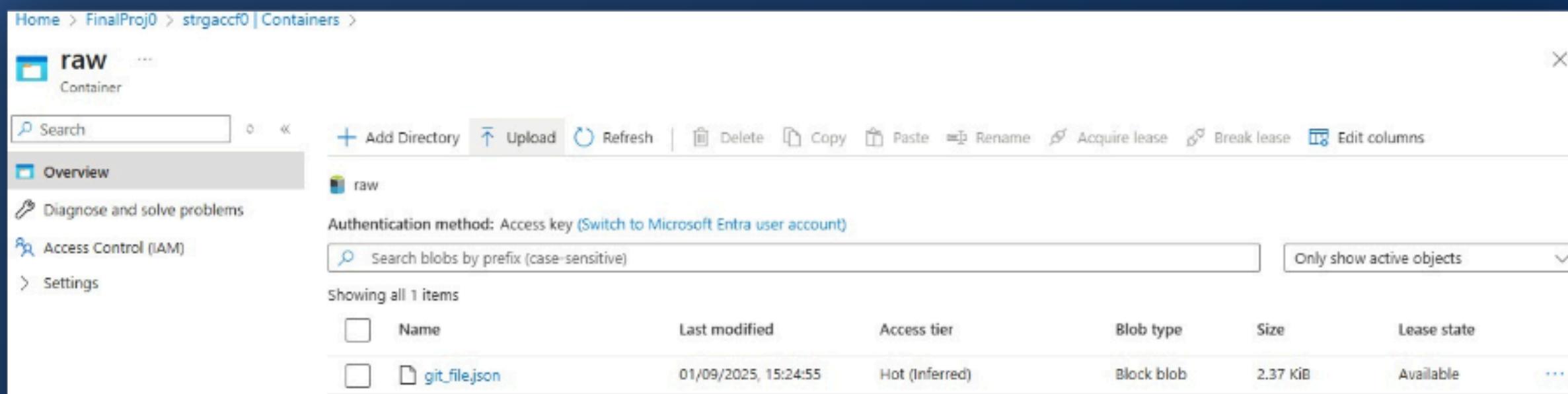


Fig2: Successfully Uploaded

CREATION OF AZURE DATA FACTORY

Home > FinalProj0 > Marketplace > Data Factory >

Create Data Factory

Basics Git configuration Networking Advanced Tags Review + create

One-click to create data factory with sample pipeline and datasets. [Try it](#)

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ Azure subscription 1
Resource group * ⓘ FinalProj0
Create new

Instance details

Name * ⓘ finalADFO
Region * ⓘ Central India
Version * ⓘ V2

Previous Next Review + create

Fig: Creation of Azure Data Factory

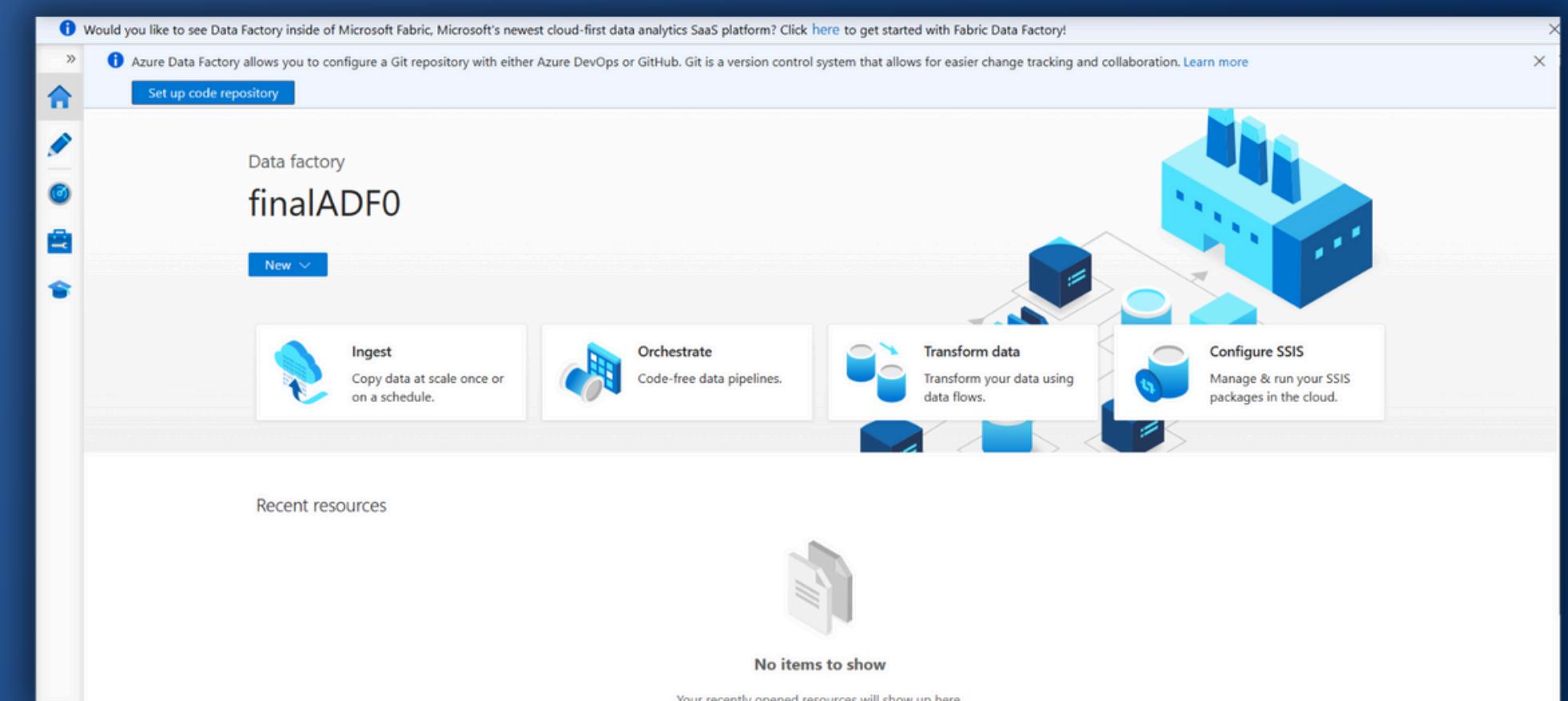


Fig: Successful launching of the ADF

DATA INGESTION PIPELINE

Lookup Activity and Output :

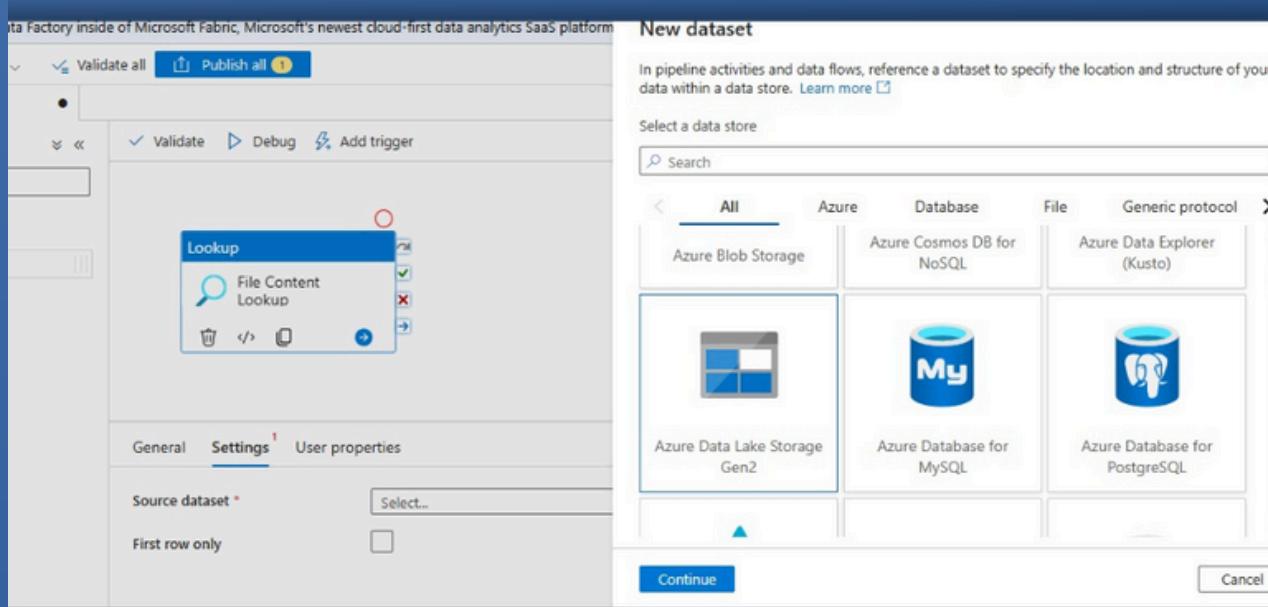


Fig: Dataset Creation for Lookup

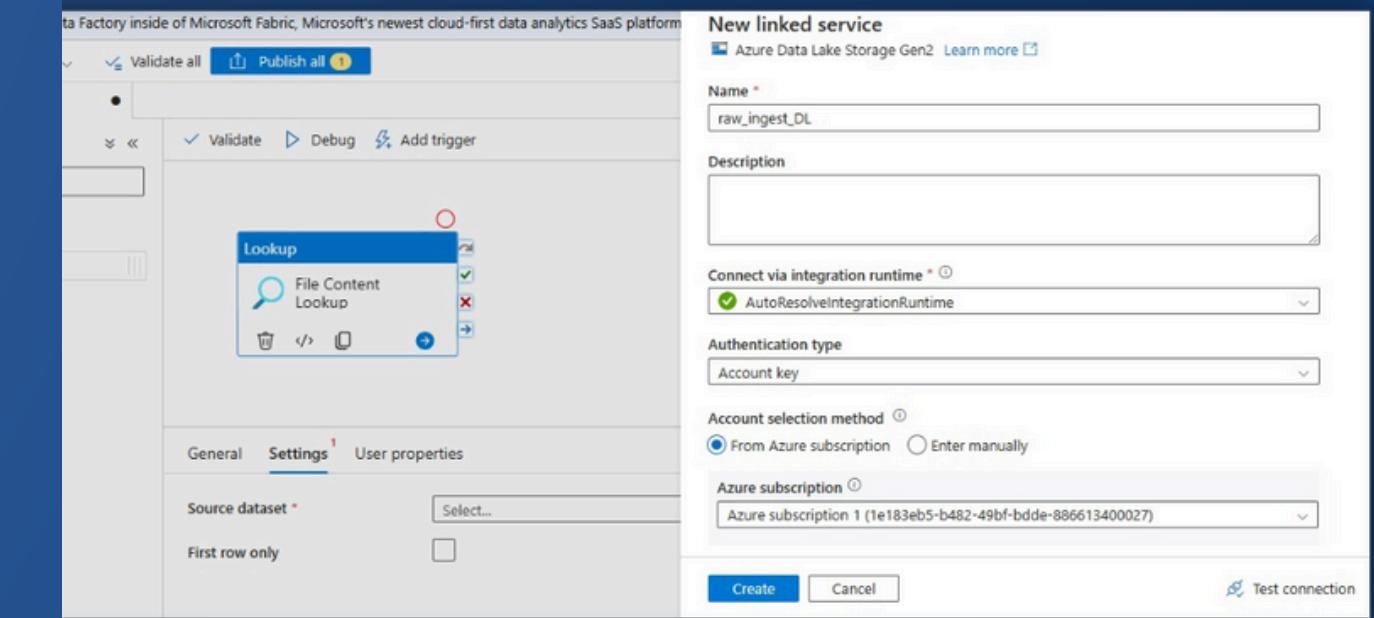


Fig: Linked Service Creation

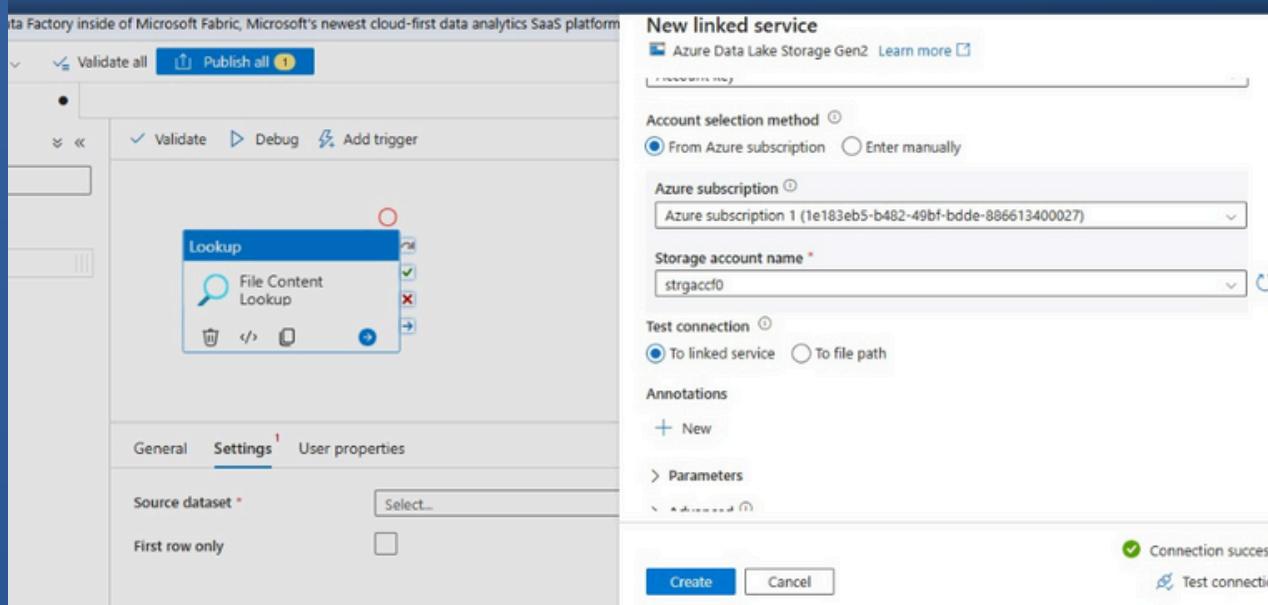


Fig: Linked Service Creation

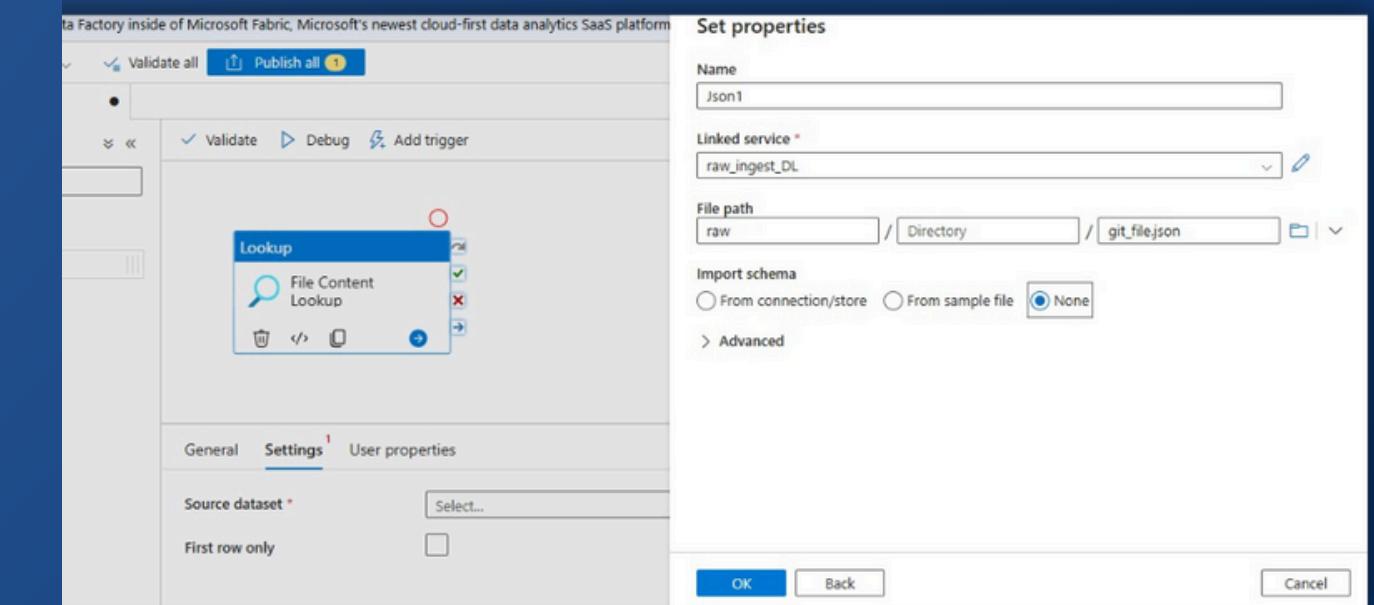


Fig: Selecting GitHub file in Lookup (raw container)

DATA INGESTION PIPELINE CONTINUED...

Lookup Activity and Output:

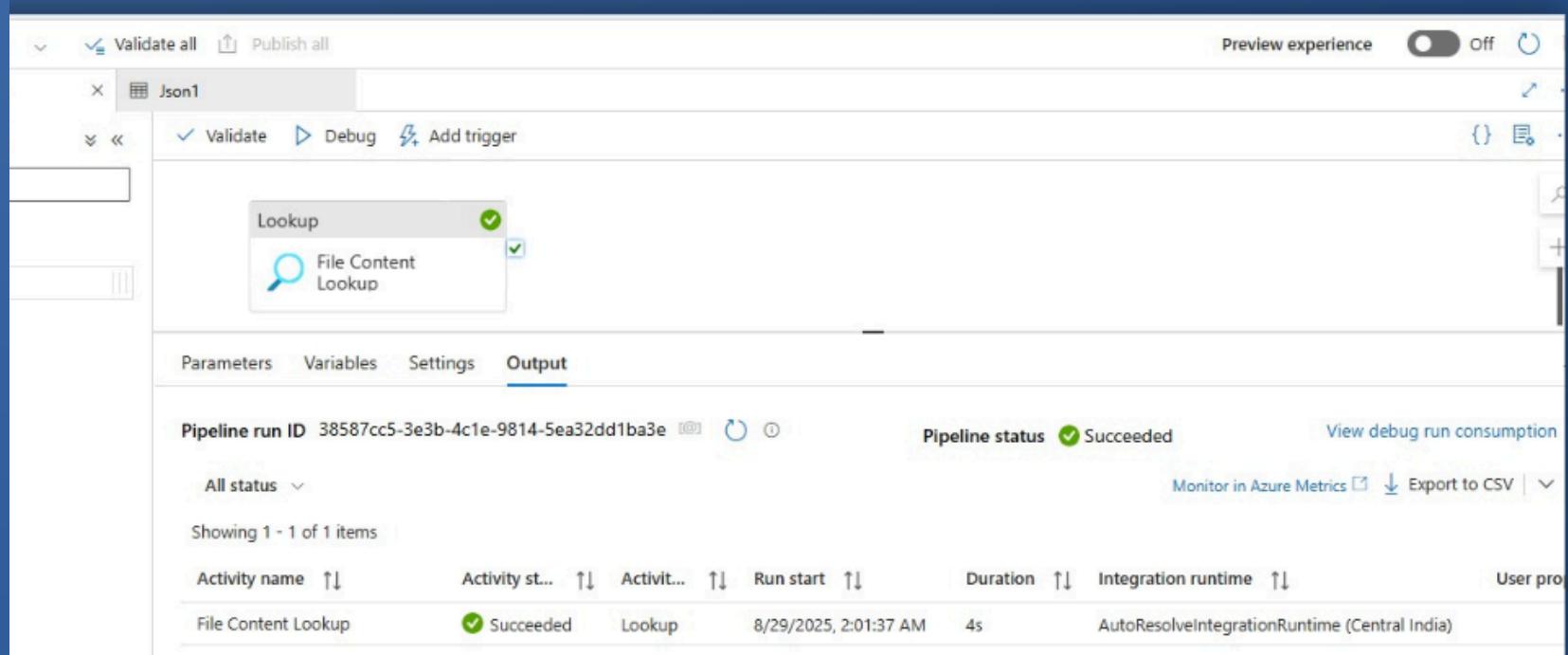


Fig: Lookup Activity run successfully

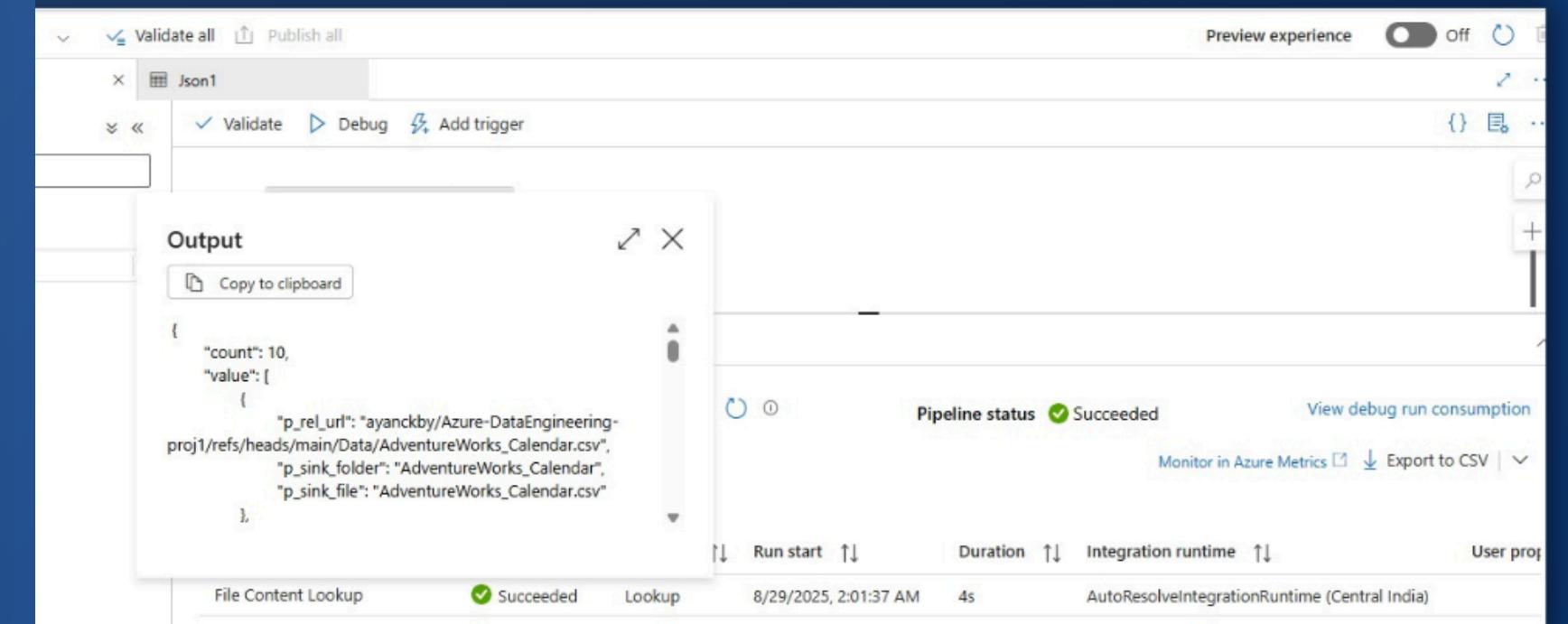


Fig: Lookup Output

DATA INGESTION PIPELINE CONTINUED...

ForEach Activity :

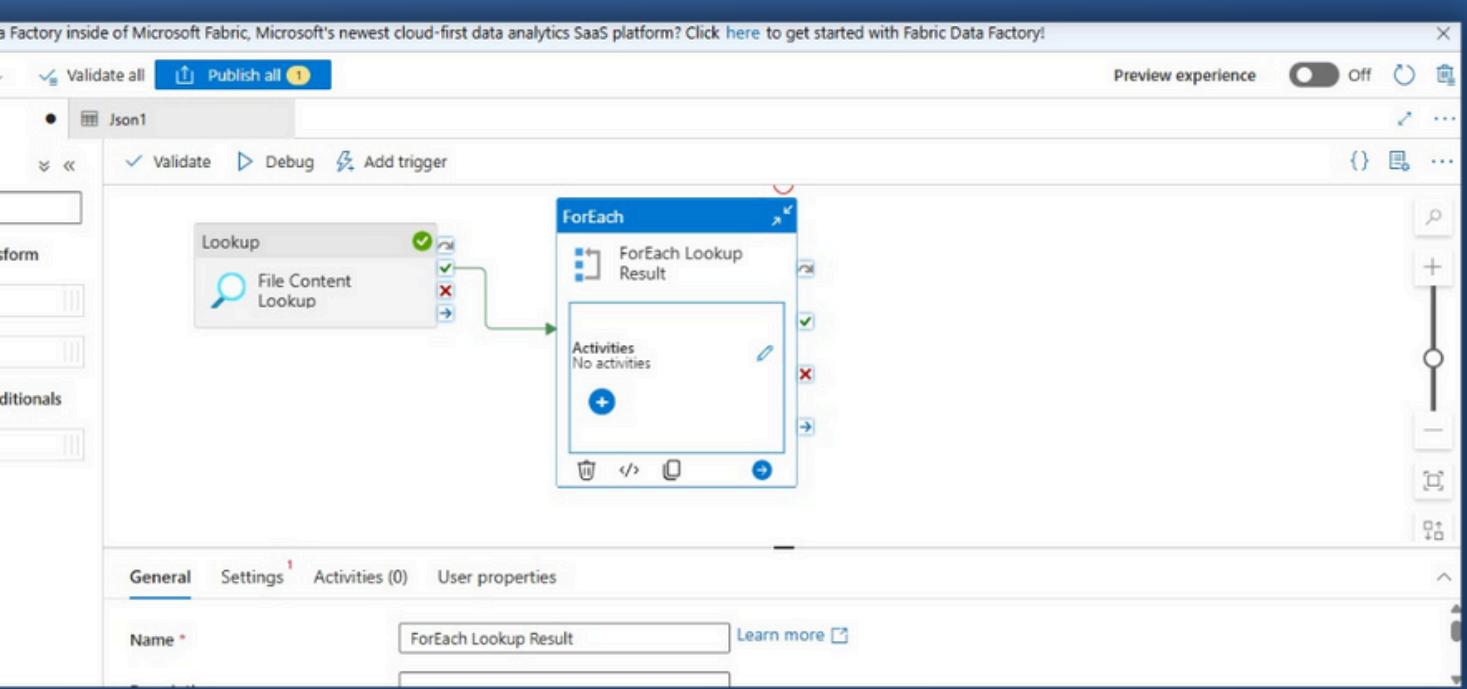


Fig: Connecting ForEach Activity with Lookup

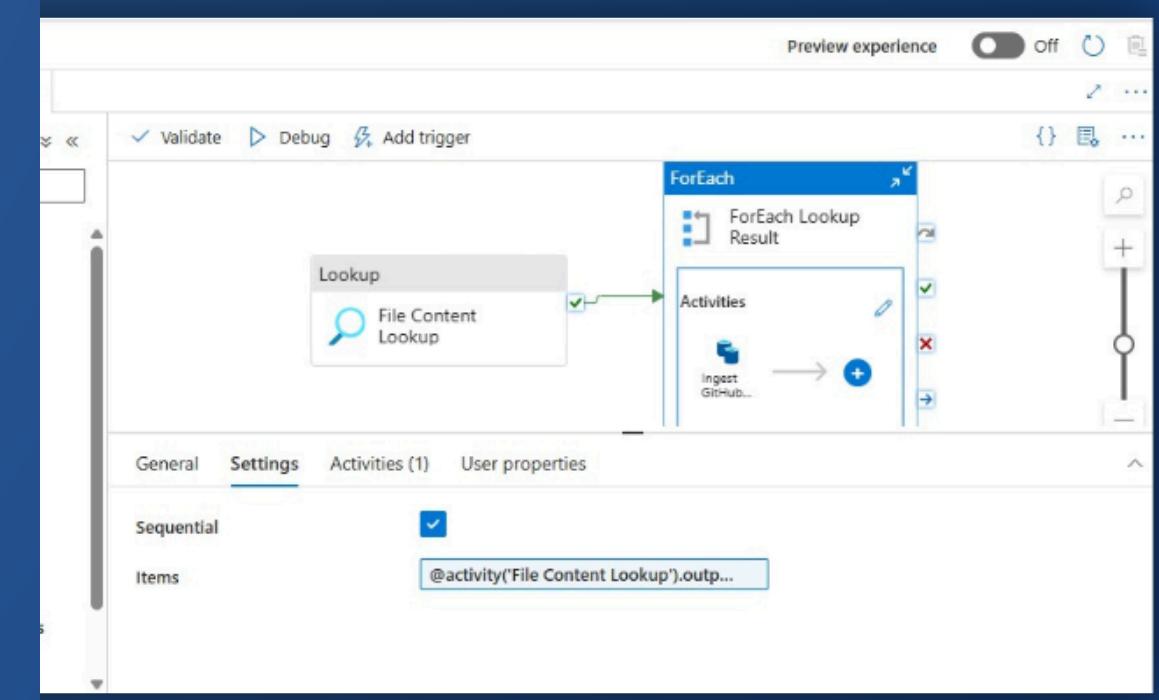


fig: Iterating over Lookup Output

DATA INGESTION PIPELINE CONTINUED...

Copy Data Activity:

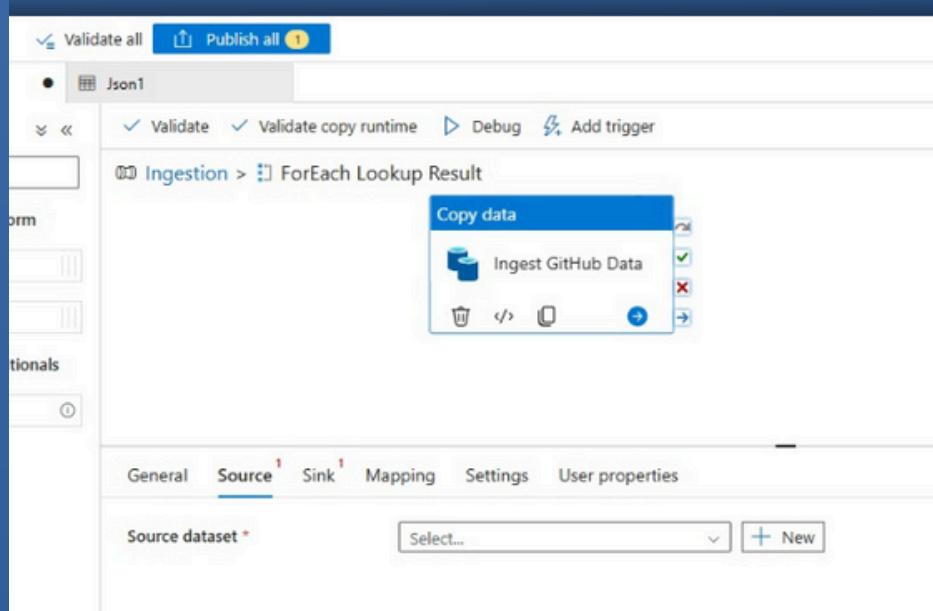


Fig: Selecting Copy data Activity and rename with 'Ingest GitHub Data'

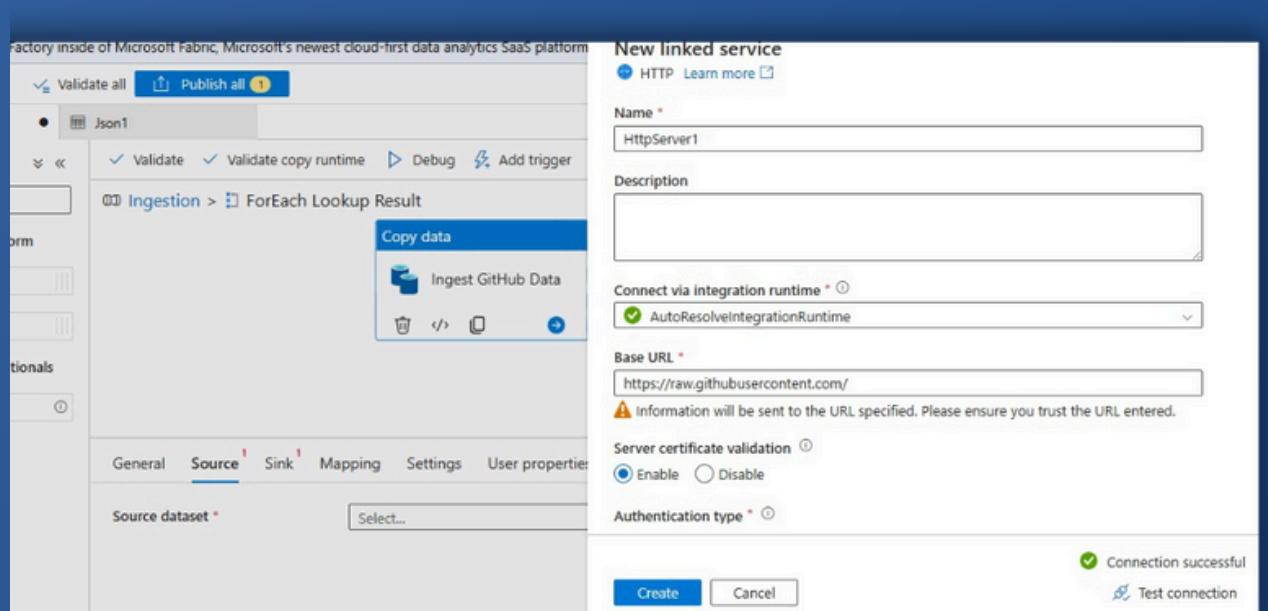


Fig: Linked Service Creation - GitHub as Source

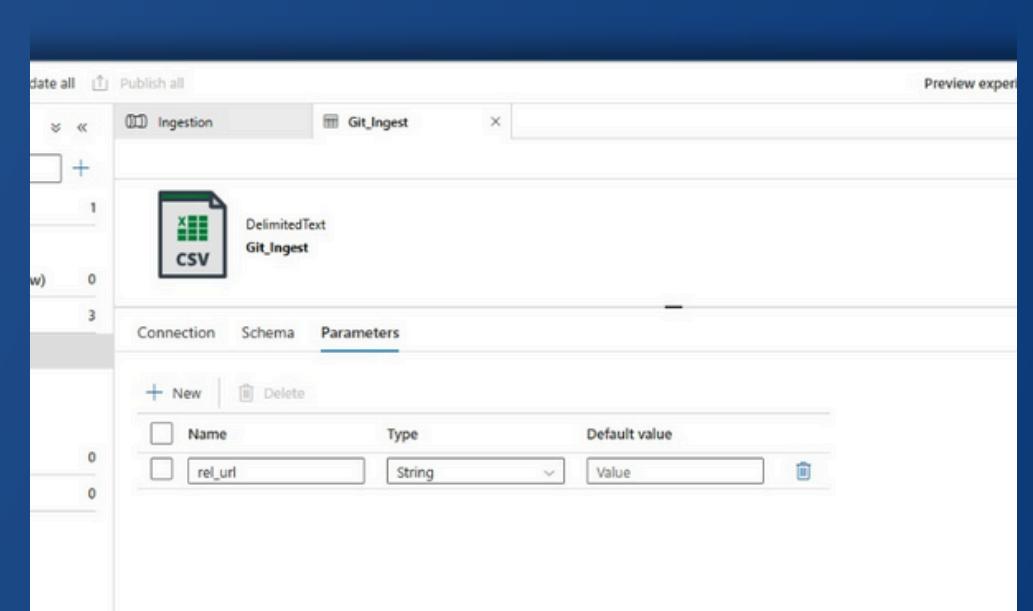


Fig: Parameter Creation for Source

DATA INGESTION PIPELINE CONTINUED...

Copy Data Activity Continued :

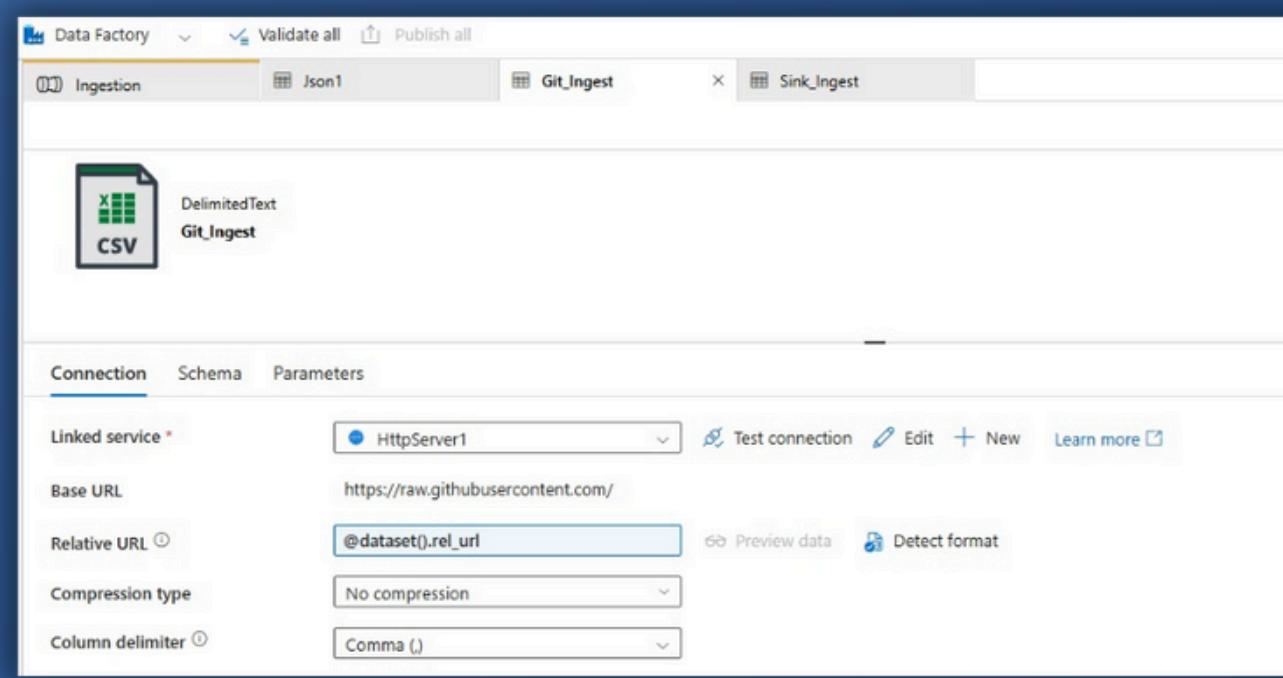


Fig: Relative URL Setup

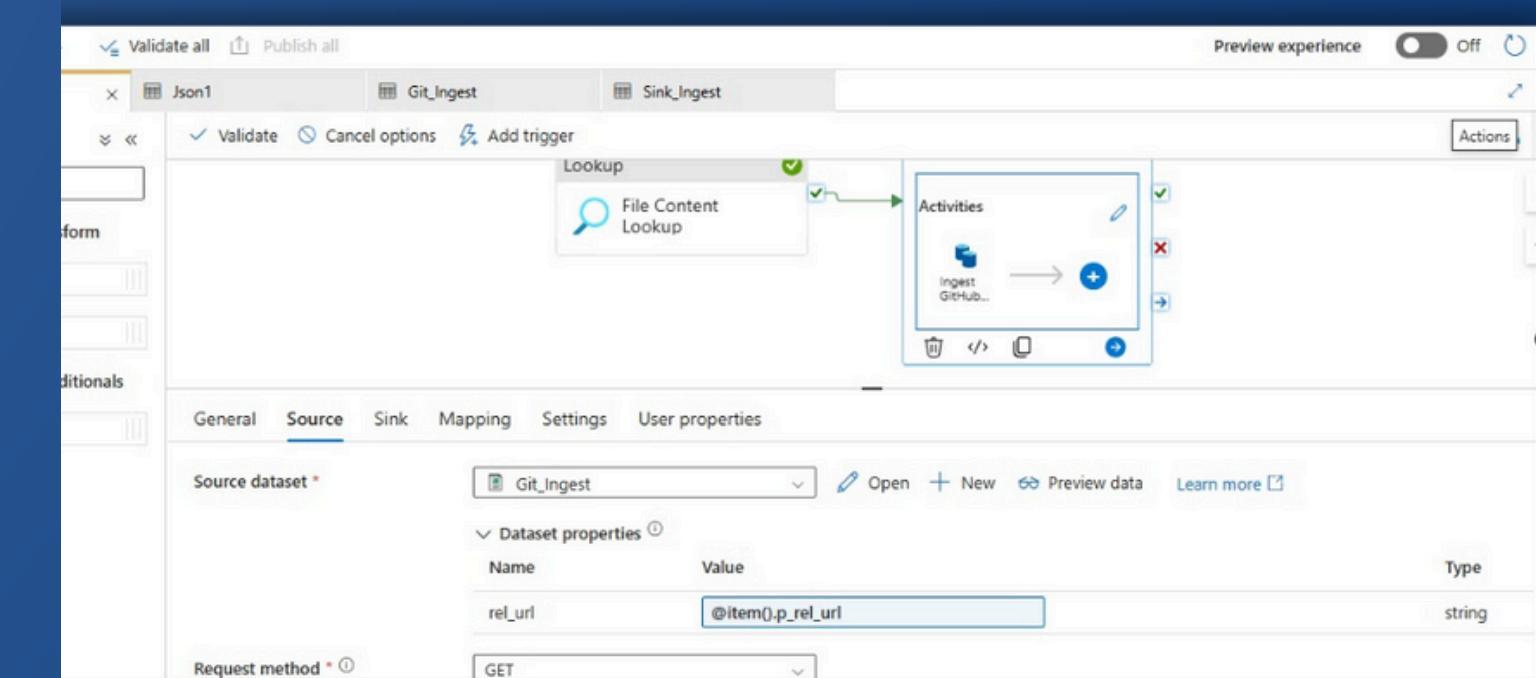


Fig: Assigning value to Relative URL Parameter

DATA INGESTION PIPELINE CONTINUED...

Copy Data Activity Continued :

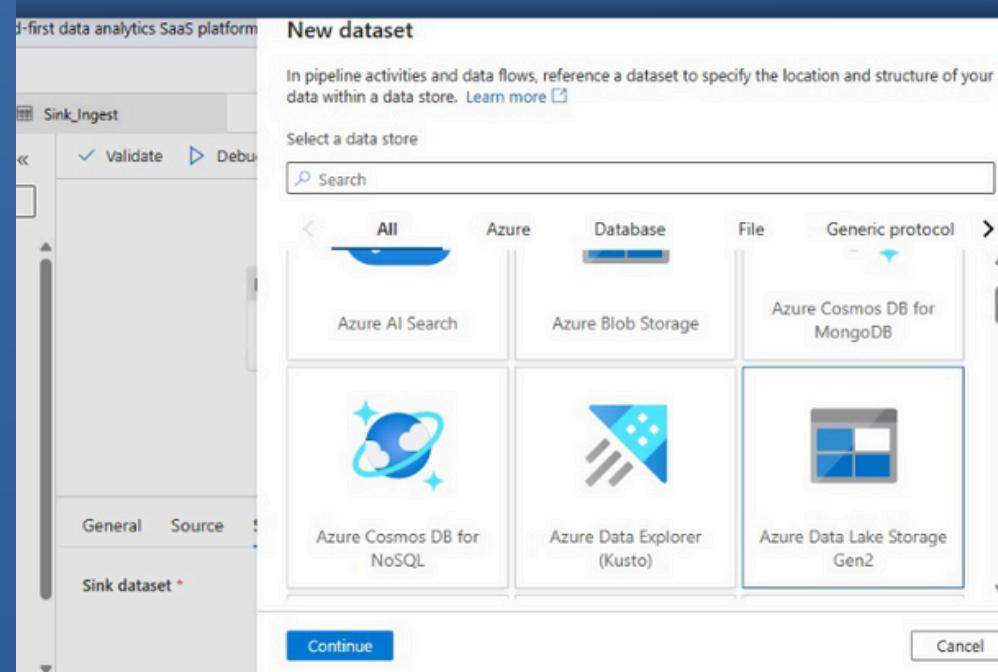


Fig: Dataset Creation of Sink

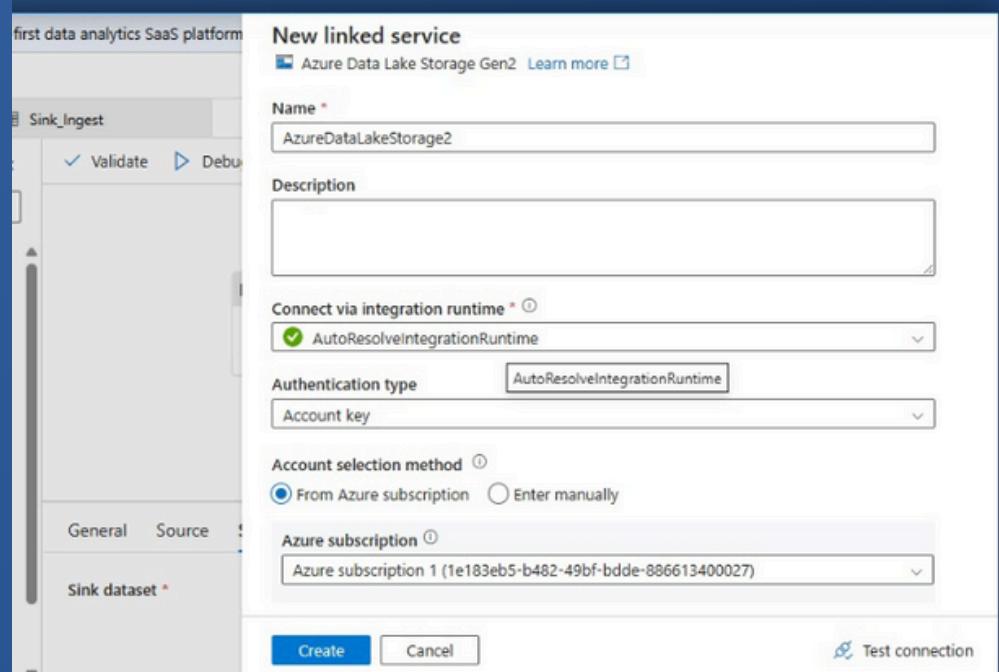


Fig: Linked Service Creation of Sink

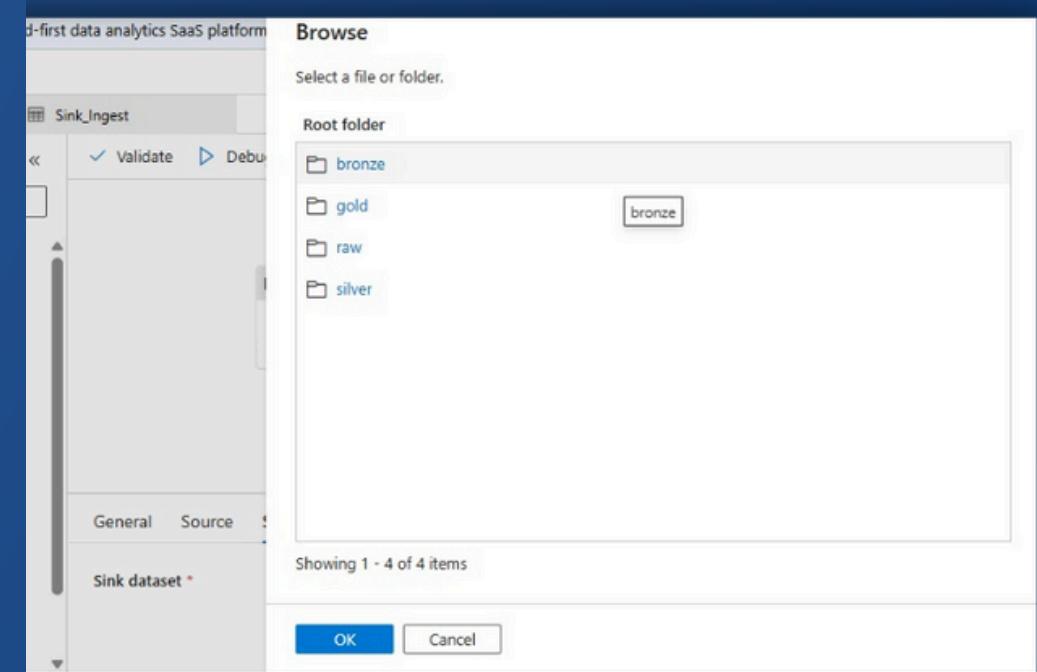


Fig: Selecting File Path

DATA INGESTION PIPELINE CONTINUED...

Copy Data Activity Continued :

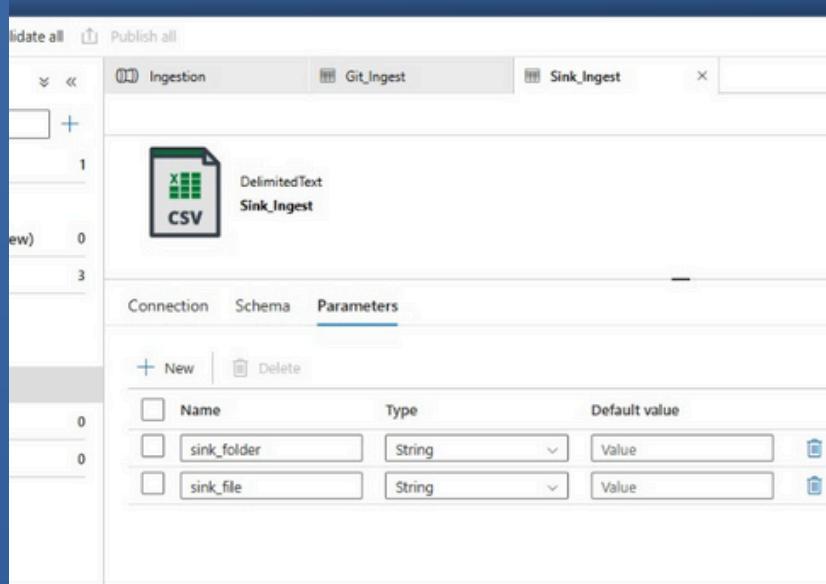


Fig: Creation of Parameters for Sink

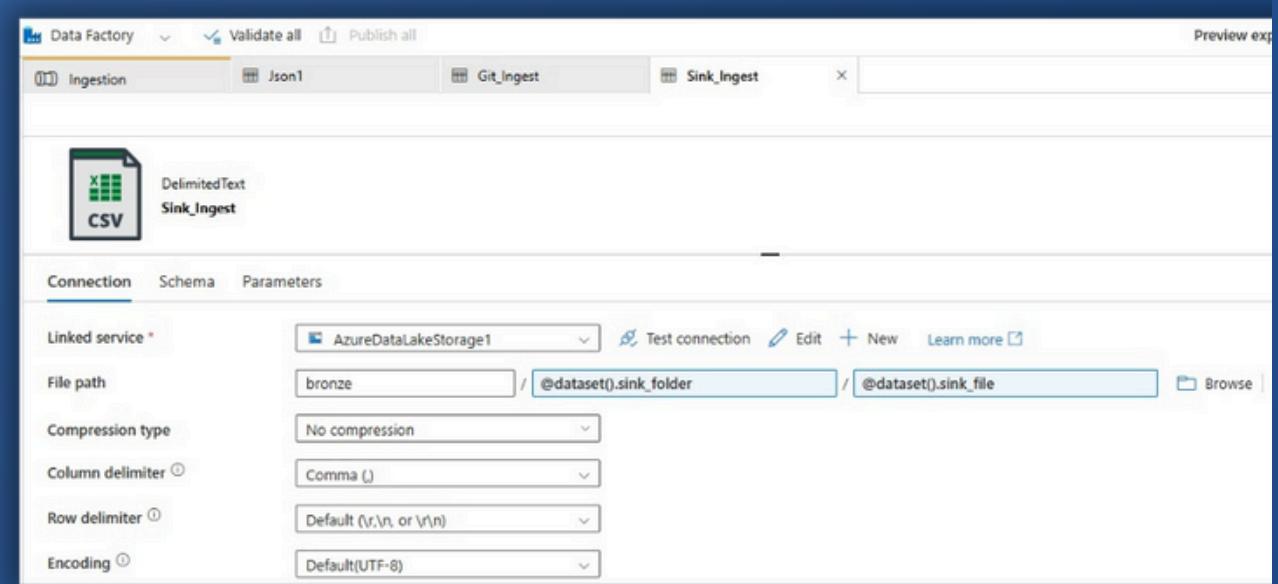


Fig: File path

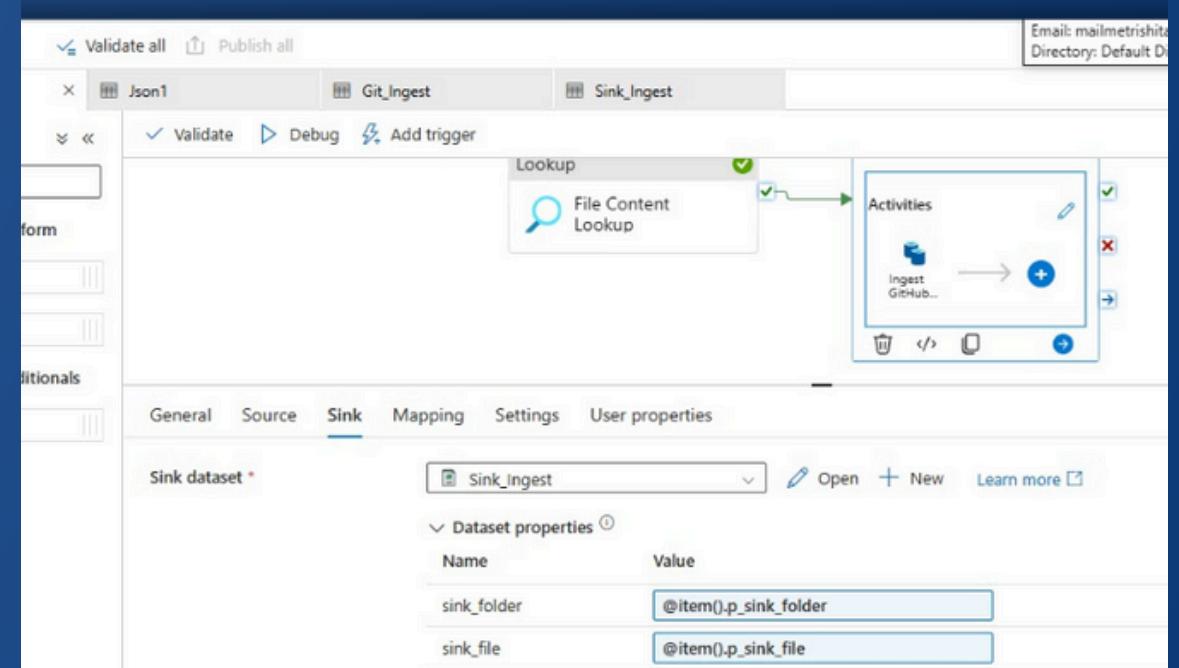


Fig: Assigning Value

DATA INGESTION PIPELINE CONTINUED...

Pipeline Output

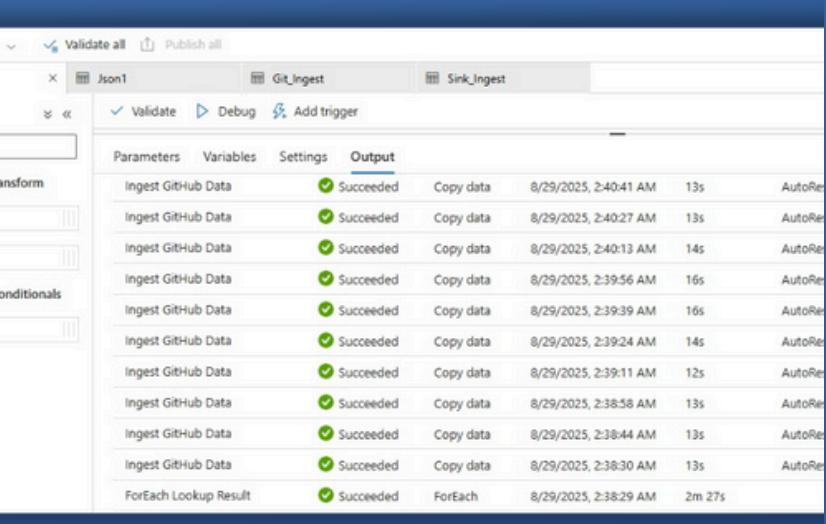


Fig: Pipeline Execution Successful

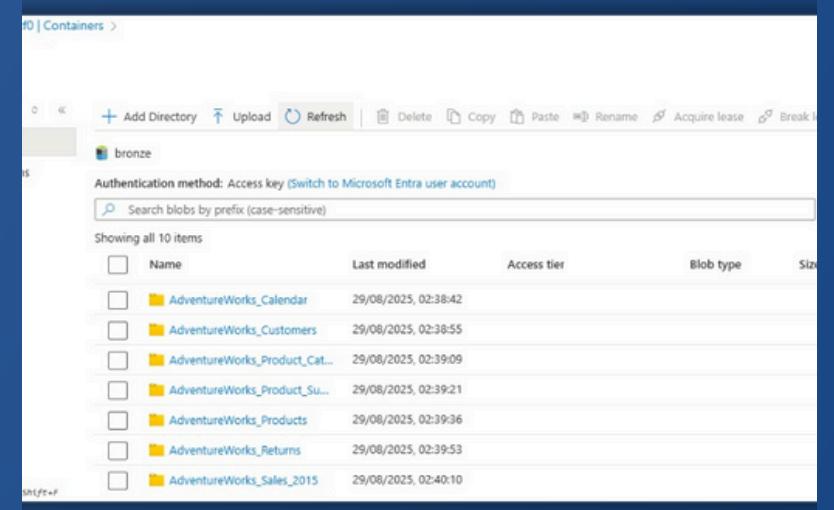


Fig: Successful Data Ingestion to Bronze Container

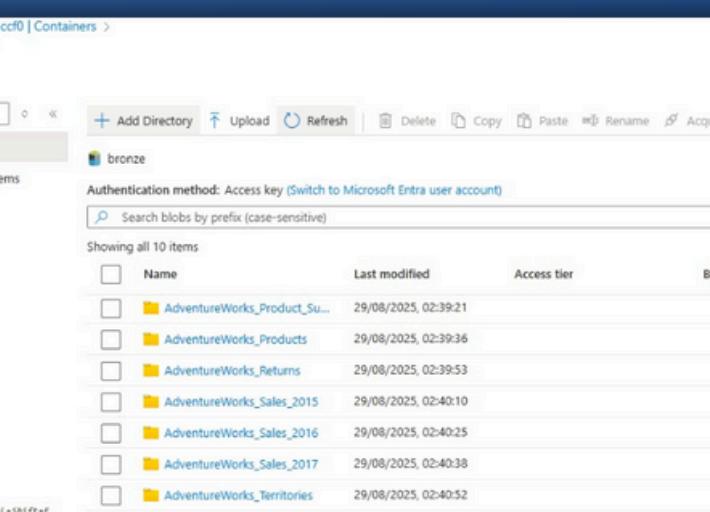


Fig: Successful Data Ingestion to Bronze Container

CREATION OF AZURE DATABRICKS

Home > FinalProj0 > Marketplace > Azure Databricks >

Create an Azure Databricks workspace

[Basics](#) [Networking](#) [Encryption](#) [Security & Compliance](#) [Tags](#) [Review + Create](#) [...](#) [X](#)

Project Details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ

Resource group * ⓘ [Create new](#)

Instance Details

Workspace name * ✓

Region *

Pricing Tier * ⓘ ✗

We selected the recommended pricing tier for your workspace. You can change the tier based on your needs.

Managed Resource Group name

[Review + create](#) [< Previous](#) [Next : Networking >](#)

APP REGISTRATION AND ROLE ASSIGNMENT

The screenshot shows the 'Overview' tab of an app registration named 'FinalProjApp'. Key details visible include:

- Display name: FinalProjApp
- Client credentials: o_certificate_1_secret
- Redirect URIs: Add a Redirect URI
- Application ID: f107c110-05bb-44d8-803a-d5a225b00f69
- Object ID: e81ad002-2e96-488a-9b22-f3bdbba61d14
- Directory (tenant) ID: 22ff6450-d6d9-410b-9522-500c523bfad8
- Managed application in L...: FinalProjApp
- Supported account types: My organization only

A note at the bottom states: "Starting June 30th, 2020 we will no longer add any new features to Azure Active Directory Authentication Library (ADAL) and Azure Active Directory Graph. We will continue to provide technical support and security updates but we will no longer provide feature updates. Applications will need to be upgraded to Microsoft Authentication Library (MSAL) and Microsoft Graph." [Learn more](#)

Fig: Creating App

The screenshot shows the 'Certificates & secrets' tab for the same app registration. It displays a single client secret entry:

Description	Expires	Value	Secret ID
FinalProjApp Client Secret	2/26/2026	vj*****	12345678901234567890

Fig: Providing Client Secret

The screenshot shows the 'Add role assignment' dialog for the 'storage blob data' role. The 'Role' tab is selected, showing the following configuration:

- Selected role: storage blob data
- Assign access to: User, group, or service principal (radio button selected)
- Members: FinalProjApp
- Description: Optional

Fig: Role Assignment

The screenshot shows the 'Add role assignment' dialog for the 'Storage Blob Data Contributor' role. The 'Members' tab is selected, showing the following configuration:

- Selected role: Storage Blob Data Contributor
- Assign access to: User, group, or service principal (radio button selected)
- Members: FinalProjApp
- Description: Optional

Fig: Adding the App as Member

TRANSFORMATION USING DATABRICKS

Creating new Cluster:

The screenshot shows the 'Create new compute' interface in the Databricks UI. The 'General' tab is selected. The 'Compute name' field contains 'FinalProj Cluster 2025-08-30 01:08:15'. The 'Policy' dropdown is set to 'Unrestricted'. Under 'Performance', the 'Machine learning' radio button is selected. In the 'Databricks runtime' section, '16.4 LTS (Scala 2.12)' is selected, and 'Scala 2.12, Spark 3.5.2' is chosen from the dropdown. The 'Photon acceleration' checkbox is checked. The 'Worker type' section shows 'Standard_D4ds_v5' with '16 GB Memory, 4 Cores'. Below this are 'Min' and 'Max' input fields with values '2' and '8' respectively, and a 'Single node' checkbox. At the bottom are 'Create' and 'Cancel' buttons.

Fig: Creating New Cluster

The screenshot shows the 'Create new compute' interface in the Databricks UI. The 'Performance' tab is selected. The 'Compute name' field contains 'FinalProj Cluster 2025-08-30 01:08:15'. The 'Policy' dropdown is set to 'Unrestricted'. Under 'Performance', the 'Machine learning' radio button is selected. In the 'Databricks runtime' section, '16.4 LTS (Scala 2.12)' is selected, and 'Scala 2.12, Spark 3.5.2' is chosen from the dropdown. The 'Photon acceleration' checkbox is checked. The 'Node type' section shows 'Standard_D4ds_v5' with '16 GB Memory, 4 Cores'. Below this are 'Min' and 'Max' input fields with values '2' and '8' respectively, and a 'Single node' checkbox. A 'Terminate after' dropdown is set to '120 minutes of inactivity'. At the bottom are 'Create' and 'Cancel' buttons.

Fig: Creating New Cluster

The screenshot shows the 'Create new compute' interface in the Databricks UI. The configuration has been completed. The 'Compute name' field contains 'FinalProj Cluster 2025-08-30 01:08:15'. The 'Policy' dropdown is set to 'Unrestricted'. Under 'Performance', the 'Machine learning' radio button is selected. In the 'Databricks runtime' section, '16.4 LTS (Scala 2.12)' is selected, and 'Scala 2.12, Spark 3.5.2' is chosen from the dropdown. The 'Photon acceleration' checkbox is checked. The 'Node type' section shows 'Standard_D2ads_v6' with '8 GB Memory, 2 Cores'. Below this is a checked 'Single node' checkbox. A 'Terminate after' dropdown is set to '60 minutes of inactivity'. At the bottom are 'Create' and 'Cancel' buttons.

Fig: Creating New Cluster

TRANSFORMATION USING DATABRICKS CONTINUED...

Data Transformation and Load into Silver Layer

Data Transformation: Bronze to Silver using PySpark

Granting Databricks access to ADLS via Service Principal

```
sec_id= ""

app_id= "f107c110-05bb-44d8-803a-d5a225b00f69"
ten_id= "22ff6450-d6d9-410b-9522-500c523bfad8"
strgacc = "strgaccf0"
```

FinalProj Bronze to Silver x

```
spark.conf.set("fs.azure.account.auth.type.{strgacc}.dfs.core.windows.net", "OAuth")
spark.conf.set("fs.azure.account.oauth.provider.type.{strgacc}.dfs.core.windows.net", "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider")
spark.conf.set("fs.azure.account.oauth2.client.id.{strgacc}.dfs.core.windows.net", app_id)
spark.conf.set("fs.azure.account.oauth2.client.secret.{strgacc}.dfs.core.windows.net", sec_id)
spark.conf.set("fs.azure.account.oauth2.client.endpoint.{strgacc}.dfs.core.windows.net", "https://login.microsoftonline.com/22ff6450-d6d9-410b-9522-500c523bfad8/oauth2/token")

dbutils.fs.ls("abfss://bronze@strgaccf0.dfs.core.windows.net/")

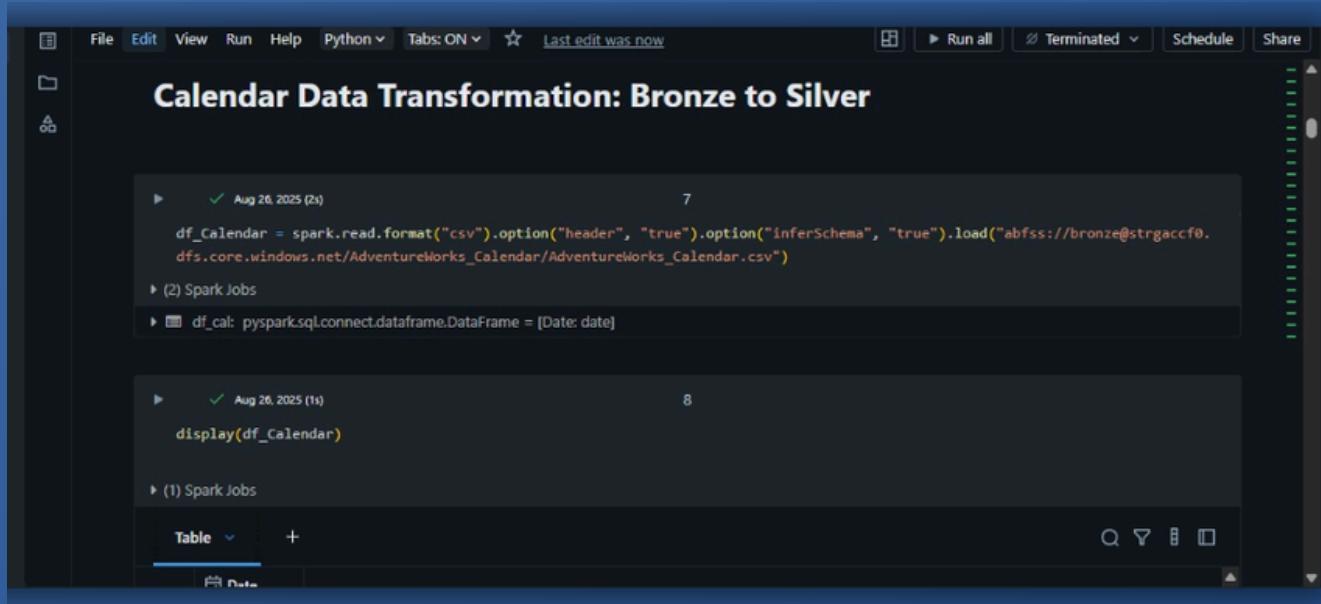
[FileInfo(path='abfss://bronze@strgaccf0.dfs.core.windows.net/AdventureWorks_Calendar/', name='AdventureWorks_Calendar/', size=0, modificationTime=1756415322000),
 FileInfo(path='abfss://bronze@strgaccf0.dfs.core.windows.net/AdventureWorks_Customers/', name='AdventureWorks_Customers/', size=0,
```

Fig:Granting DataBricks Access to ADLS via Service Principal

Fig:Granting DataBricks Access to ADLS via Service Principal

TRANSFORMATION USING DATABRICKS CONTINUED...

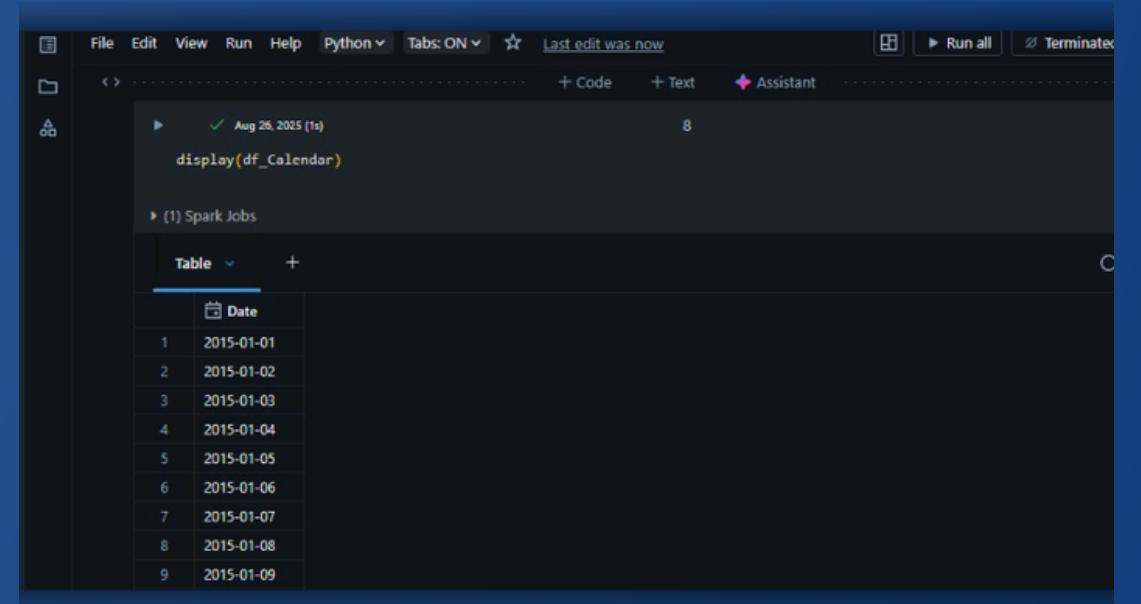
Data Transformation and Load into Silver Layer Continued...



Calendar Data Transformation: Bronze to Silver

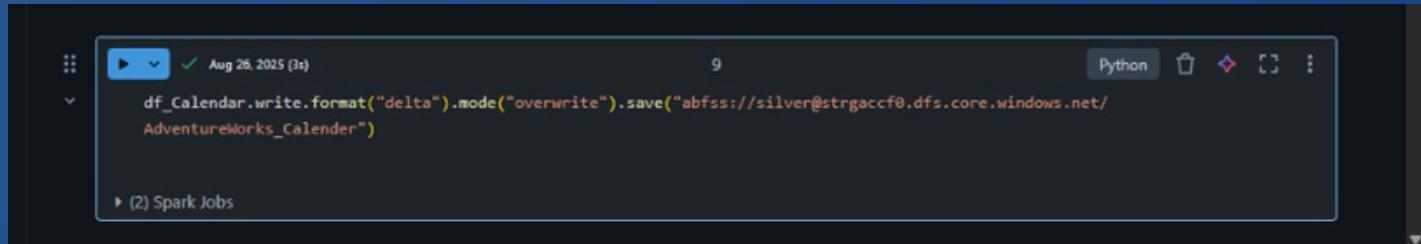
```
Aug 26, 2025 (2s)
df_Calendar = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("abfss://bronze@strgaccf0.dfs.core.windows.net/AdventureWorks_Calendar/Adventureworks_Calendar.csv")  
(2) Spark Jobs  
df_cal: pyspark.sql.connect.DataFrame = [Date: date]  
  
Aug 26, 2025 (1s)
display(df_Calendar)  
(1) Spark Jobs  
Table +  
Data
```

Fig: Calender Data Load Into DataBricks



```
Aug 26, 2025 (1s)
display(df_Calendar)  
(1) Spark Jobs  
Table +  
Date  
1 2015-01-01  
2 2015-01-02  
3 2015-01-03  
4 2015-01-04  
5 2015-01-05  
6 2015-01-06  
7 2015-01-07  
8 2015-01-08  
9 2015-01-09
```

Fig: Calender Data Display



```
Aug 26, 2025 (3s)
df_Calendar.write.format("delta").mode("overwrite").save("abfss://silver@strgaccf0.dfs.core.windows.net/Adventureworks_Calendar")  
(2) Spark Jobs
```

Fig: Calender Data Save Into Silver Layer

TRANSFORMATION USING DATABRICKS CONTINUED...

Data Transformation and Load into Silver Layer Continued...

The screenshot shows a Databricks notebook titled "Customer Data Transformation: Bronze to Silver". The code cell contains Python code to read a CSV file from a bronze storage location and display the resulting DataFrame. The DataFrame table view shows columns: CustomerKey, Prefix, FirstName, LastName, BirthDate, MaritalStatus, Gender, and EmailAddress. The first few rows of data are displayed.

CustomerKey	Prefix	FirstName	LastName	BirthDate	MaritalStatus	Gender	EmailAddress
11000	MR.	JON	YANG	1966-04-08	M	M	jyang@adventure-works.com
11001	MR.	EUGENE	HUANG	1965-05-14	S	M	ehuang@adventure-works.com
11002	MR.	RUBEN	TORRES	1965-08-12	M	M	rtorres@adventure-works.com
11003	MS.	CHRISTY	ZHU	1968-02-15	S	F	czhu@adventure-works.com

Fig: Customer Data Load Into DataBricks

The screenshot shows a continuation of the Databricks notebook. The code cell contains Python code for transforming the customer data. It includes importing functions, adding a new column "Name" by concatenating "Prefix", "FirstName", and "LastName", dropping unnecessary columns, and arranging the columns in a specific order. The final DataFrame table view shows columns: CustomerKey, Name, BirthDate, Gender, EmailAddress, AnnualIncome, Occupation, and HomeOwner.

CustomerKey	Name	BirthDate	Gender	EmailAddress	AnnualIncome	Occupation	HomeOwner
11025	ALEJANDRO BECK	1945-12-23	NA	alejandro45@adventure-works.com	\$10,000	Analyst	Yes
11026	MR. HAROLD SAI	1946-04-03	M	harold3@adventure-works.com	\$30,000	Manager	No
11027	MR. JESSIE ZHAO	1946-12-07	M	jessie16@adventure-works.com	\$30,000	Manager	No
11028	MRS. JILL JIMENEZ	1946-04-11	F	jill13@adventure-works.com	\$30,000	Manager	No
11029	MR. JIMMY MORENO	1946-12-21	M	jimmy9@adventure-works.com	\$30,000	Manager	No

Fig: Customer Data Transformation

The screenshot shows a continuation of the Databricks notebook. The code cell contains Python code for transforming the customer data. It includes selecting specific columns and displaying the resulting DataFrame. The final DataFrame table view shows columns: CustomerKey, Name, BirthDate, Gender, EmailAddress, and AnnualIncome.

CustomerKey	Name	BirthDate	Gender	EmailAddress	AnnualIncome
11025	ALEJANDRO BECK	1945-12-23	NA	alejandro45@adventure-works.com	\$10,000
11026	MR. HAROLD SAI	1946-04-03	M	harold3@adventure-works.com	\$30,000
11027	MR. JESSIE ZHAO	1946-12-07	M	jessie16@adventure-works.com	\$30,000
11028	MRS. JILL JIMENEZ	1946-04-11	F	jill13@adventure-works.com	\$30,000
11029	MR. JIMMY MORENO	1946-12-21	M	jimmy9@adventure-works.com	\$30,000

Fig: Customer Data Transformation

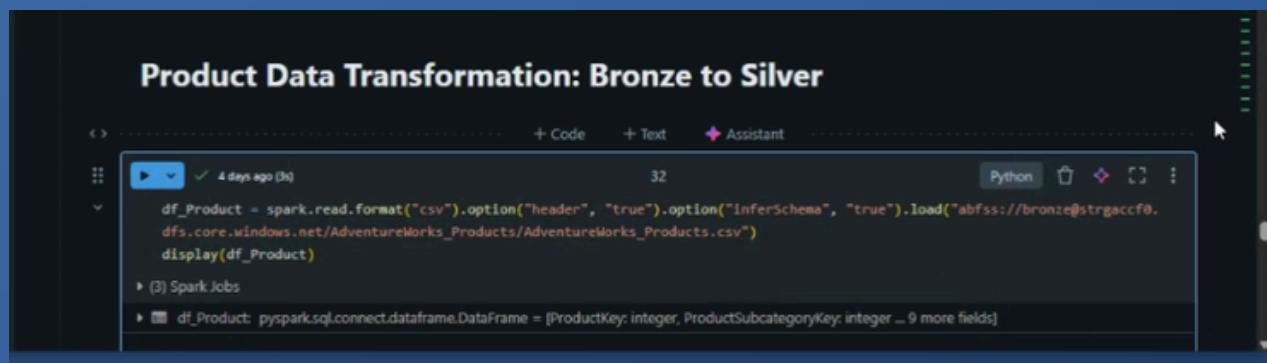
The screenshot shows a continuation of the Databricks notebook. The code cell contains Python code for saving the transformed customer data into a silver storage layer. The code uses the Delta format and overwrites the existing data in the "AdventureWorks_Customers" table.

```
df_Customer.write.format("delta").mode("overwrite").save("abfss://silver@trishstorageacc.dfs.core.windows.net/AdventureWorks_Customers")
```

Fig: Customer Data Save Into Silver Layer

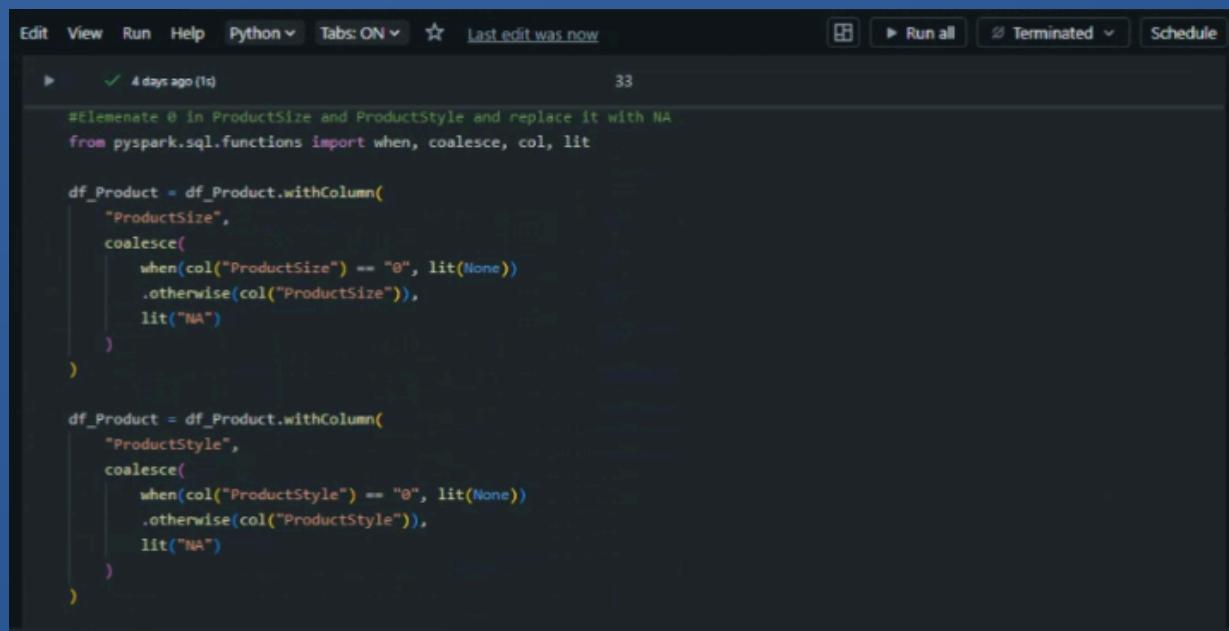
TRANSFORMATION USING DATABRICKS CONTINUED...

Data Transformation and Load into Silver Layer Continued...



```
Product Data Transformation: Bronze to Silver
+ Code + Text ♡ Assistant
4 days ago (3s) 32 Python
df_Product = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("abfss://bronze@strgaccf0.dfs.core.windows.net/AdventureWorks_Products/AdventureWorks_Products.csv")
display(df_Product)
(3) Spark jobs
df_Product: pyspark.sql.connect.DataFrame = [ProductKey: integer, ProductSubcategoryKey: integer ... 9 more fields]
```

Fig: Product Data Load Into DataBricks

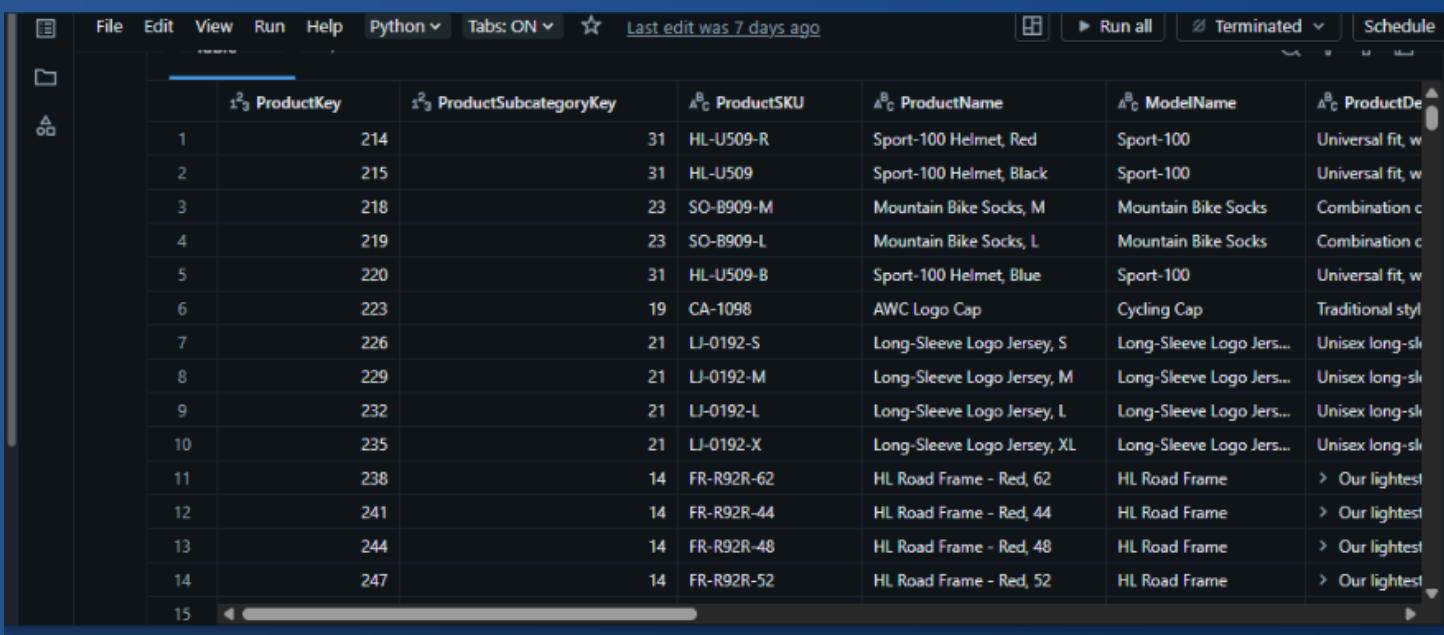


```
#Elenimate @ in ProductSize and ProductStyle and replace it with NA
from pyspark.sql.functions import when, coalesce, col, lit

df_Product = df_Product.withColumn(
    "ProductSize",
    coalesce(
        when(col("ProductSize") == "@", lit(None)),
        otherwise(col("ProductSize")),
        lit("NA")
    )
)

df_Product = df_Product.withColumn(
    "ProductStyle",
    coalesce(
        when(col("ProductStyle") == "@", lit(None)),
        otherwise(col("ProductStyle")),
        lit("NA")
    )
)
```

Fig: Product Data Transformation

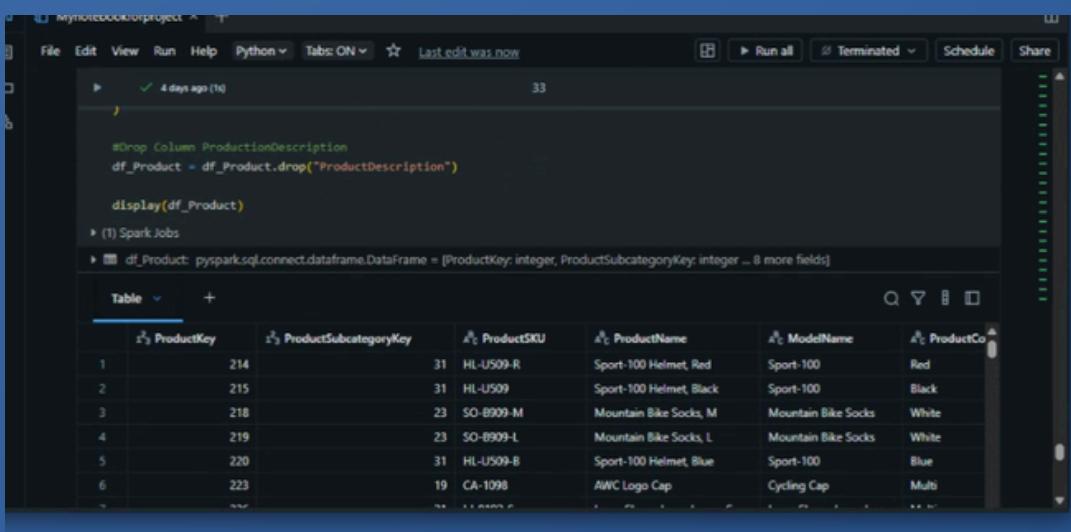


ProductKey	ProductSubcategoryKey	ProductSKU	ProductName	ModelName	ProductDe
1	214	31	HL-U509-R	Sport-100 Helmet, Red	Sport-100
2	215	31	HL-U509	Sport-100 Helmet, Black	Sport-100
3	218	23	SO-B909-M	Mountain Bike Socks, M	Mountain Bike Socks
4	219	23	SO-B909-L	Mountain Bike Socks, L	Mountain Bike Socks
5	220	31	HL-U509-B	Sport-100 Helmet, Blue	Universal fit, w
6	223	19	CA-1098	AWC Logo Cap	Cycling Cap
7	226	21	LI-0192-S	Long-Sleeve Logo Jersey, S	Long-Sleeve Logo Jers...
8	229	21	LI-0192-M	Long-Sleeve Logo Jersey, M	Long-Sleeve Logo Jers...
9	232	21	LI-0192-L	Long-Sleeve Logo Jersey, L	Long-Sleeve Logo Jers...
10	235	21	LI-0192-X	Long-Sleeve Logo Jersey, XL	Unisex long-sleeve
11	238	14	FR-R92R-62	HL Road Frame - Red, 62	Our lightest
12	241	14	FR-R92R-44	HL Road Frame - Red, 44	Our lightest
13	244	14	FR-R92R-48	HL Road Frame - Red, 48	Our lightest
14	247	14	FR-R92R-52	HL Road Frame - Red, 52	Our lightest

Fig: Product Data Display Before Transformation

TRANSFORMATION USING DATABRICKS CONTINUED...

Data Transformation and Load into Silver Layer Continued...



A screenshot of a Databricks notebook titled "mynotebookproject". The notebook has two runs: run 33 and run 34. Run 33 shows Python code for dropping a column and displaying the transformed DataFrame. Run 34 shows the code for saving the DataFrame to a Delta table in the Silver layer. Below the notebook, a preview of the DataFrame shows columns: ProductKey, ProductSubcategoryKey, ProductSKU, ProductName, ModelName, and ProductCo. The data includes items like Sport-100 Helmet, Red and Mountain Bike Socks, M.

	ProductKey	ProductSubcategoryKey	ProductSKU	ProductName	ModelName	ProductCo
1	214	31	HL-US09-R	Sport-100 Helmet, Red	Sport-100	Red
2	215	31	HL-US09	Sport-100 Helmet, Black	Sport-100	Black
3	218	23	SO-8909-M	Mountain Bike Socks, M	Mountain Bike Socks	White
4	219	23	SO-8909-L	Mountain Bike Socks, L	Mountain Bike Socks	White
5	220	31	HL-US09-B	Sport-100 Helmet, Blue	Sport-100	Blue
6	223	19	CA-1098	AWC Logo Cap	Cycling Cap	Multi



A screenshot of a Databricks notebook titled "mynotebookproject". The notebook has two runs: run 33 and run 34. Run 34 shows the code for saving the DataFrame to a Delta table in the Silver layer. The code is: df_Product.write.format("delta").mode("overwrite").save("abfss://silrgaccf0.dfs.core.windows.net/AdventureWorks_Products"). Below the notebook, a preview of the DataFrame shows the same columns and data as in the previous screenshot.

Fig: Product Data Save Into Silver Layer

Fig: Product Data Display After Transformation

TRANSFORMATION USING DATABRICKS CONTINUED...

Data Transformation and Load into Silver Layer Continued...

Product Category Data Transformation: Bronze to Silver

```
16 df_Product_Category = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("abfss://bronze@strgaccf0.dfs.core.windows.net/AdventureWorks_Product_Categories/AdventureWorks_Product_Categories.csv")
display(df_Product_Category)

▶ (3) Spark Jobs
▶ df_Product_Category: pyspark.sql.connect.dataframe.DataFrame = [ProductCategoryKey: integer, CategoryName: string]

Table +  

  ProductCategoryKey CategoryName
  1 Bikes
  2 Components
  3 Clothing
```

```
File Edit View Run Help Python Tabs: ON Last edit was now
Run all Terminated Schedule Share
17 df_Product_Category.write.format("delta").mode("overwrite").save("abfss://silver@strgaccf0.dfs.core.windows.net/AdventureWorks_Product_Categories")

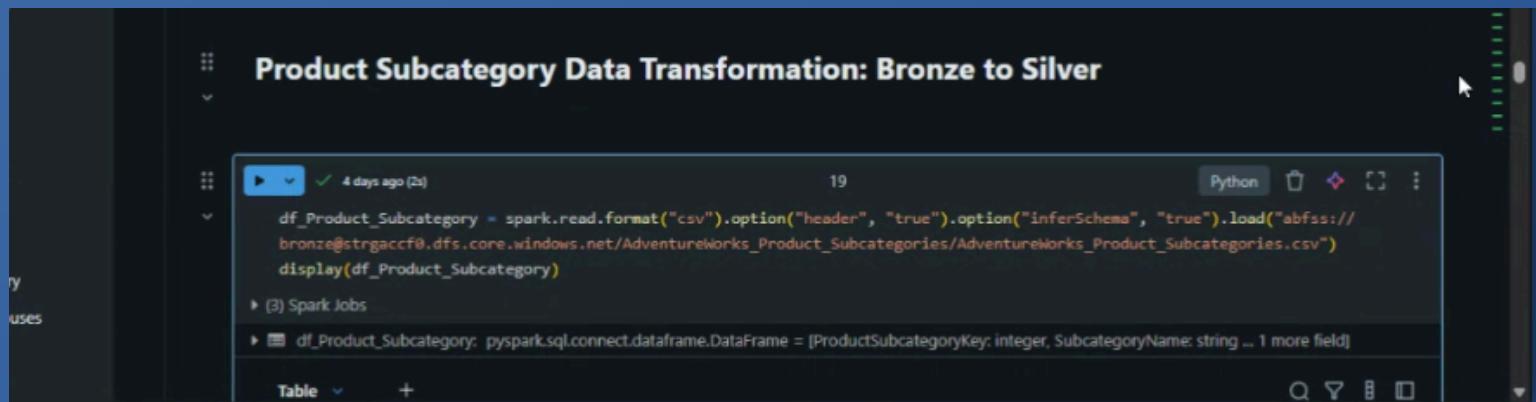
▶ (2) Spark Jobs
```

Fig: Product Catagory Data Save Into Silver Layer

Fig: Product Catagory Data Load Into DataBricks and Display

TRANSFORMATION USING DATABRICKS CONTINUED...

Data Transformation and Load into Silver Layer Continued...



```
Product Subcategory Data Transformation: Bronze to Silver

19
df_Product_Subcategory = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("abfss://bronze@strgaccf0.dfs.core.windows.net/AdventureWorks_Product_Subcategories/AdventureWorks_Product_Subcategories.csv")
display(df_Product_Subcategory)

(3) Spark Jobs
df_Product_Subcategory: pyspark.sql.connect.DataFrame = [ProductSubcategoryKey: integer, SubcategoryName: string ... 1 more field]

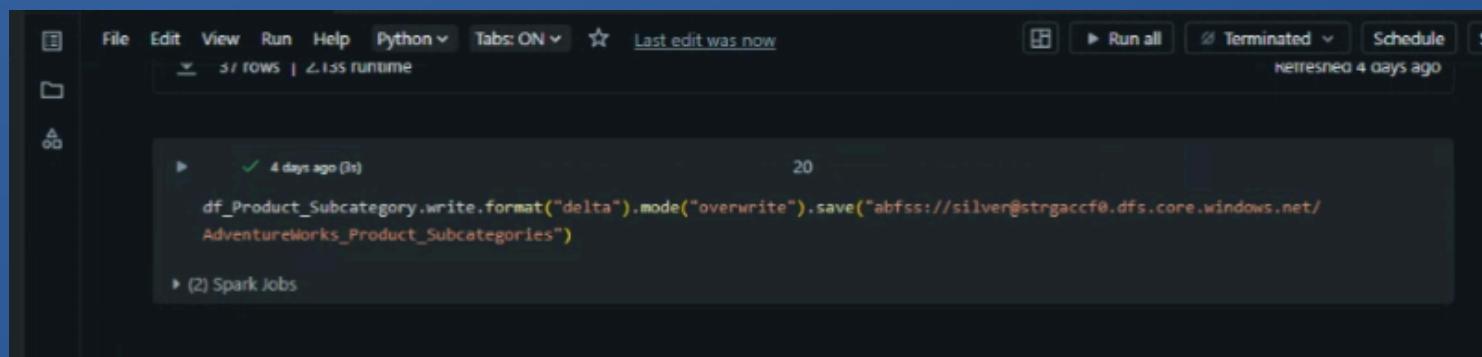
Table +
```

Fig: Product SubCatagory Data Load Into DataBricks



ProductSubcategoryKey	SubcategoryName	ProductCategoryKey
1	Mountain Bikes	1
2	Road Bikes	1
3	Touring Bikes	1
4	Handlebars	2
5	Bottom Brackets	2
6	Brakes	2
7	Chains	2
8	Cranksets	2
9	Derailleurs	2
10	Forks	2

Fig: Product SubCatagory Data Display



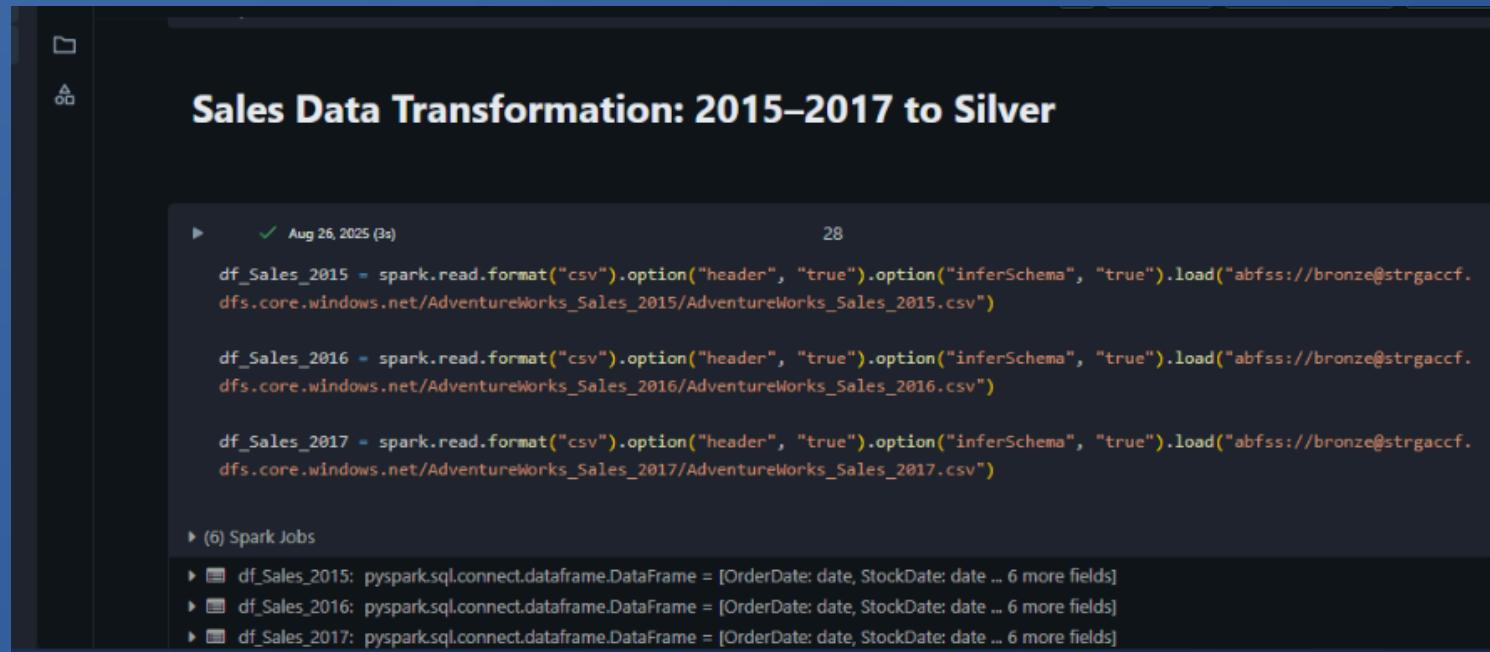
```
File Edit View Run Help Python Tabs: ON Last edit was now
Run all Terminated Schedule
51 ROWS | 155 runtime
4 days ago (1r)
df_Product_Subcategory.write.format("delta").mode("overwrite").save("abfss://silver@strgaccf0.dfs.core.windows.net/AdventureWorks_Product_Subcategories")

(2) Spark Jobs
```

Fig: Product SubCatagory Data Save Into Silver Layer

TRANSFORMATION USING DATABRICKS CONTINUED...

Data Transformation and Load into Silver Layer Continued...



Sales Data Transformation: 2015–2017 to Silver

```
Aug 26, 2025 (2s) 28
df_Sales_2015 = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("abfss://bronze@strgaccf.dfs.core.windows.net/AdventureWorks_Sales_2015/AdventureWorks_Sales_2015.csv")

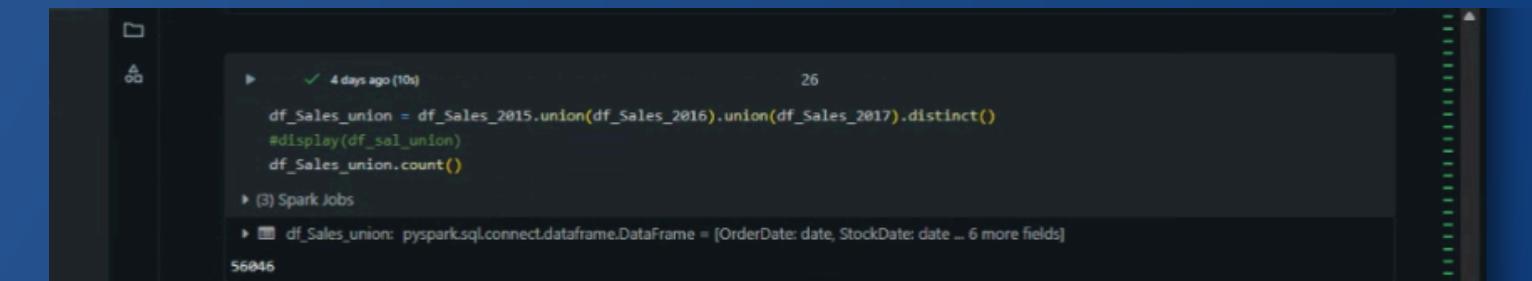
df_Sales_2016 = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("abfss://bronze@strgaccf.dfs.core.windows.net/AdventureWorks_Sales_2016/AdventureWorks_Sales_2016.csv")

df_Sales_2017 = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("abfss://bronze@strgaccf.dfs.core.windows.net/AdventureWorks_Sales_2017/AdventureWorks_Sales_2017.csv")
```

(6) Spark Jobs

- df_Sales_2015: pyspark.sql.connect.DataFrame = [OrderDate: date, StockDate: date ... 6 more fields]
- df_Sales_2016: pyspark.sql.connect.DataFrame = [OrderDate: date, StockDate: date ... 6 more fields]
- df_Sales_2017: pyspark.sql.connect.DataFrame = [OrderDate: date, StockDate: date ... 6 more fields]

Fig: Sales Data(2015-17) Load Into DataBricks

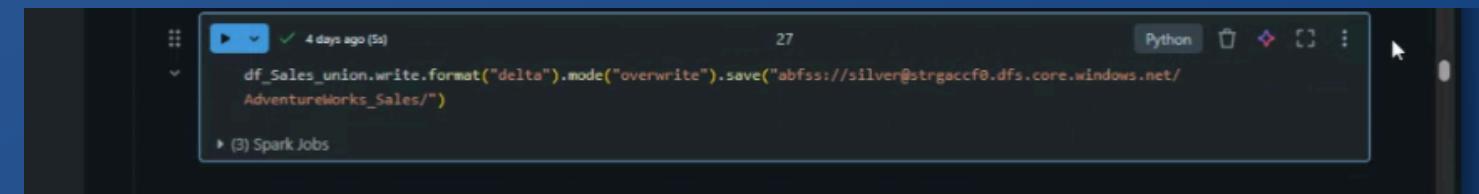


```
4 days ago (10s) 26
df_Sales_union = df_Sales_2015.union(df_Sales_2016).union(df_Sales_2017).distinct()
#display(df_Sales_union)
df_Sales_union.count()
```

(3) Spark Jobs

- df_Sales_union: pyspark.sql.connect.DataFrame = [OrderDate: date, StockDate: date ... 6 more fields]

Fig: Union Operation of Sales Data 2015,2016 and 2017



```
4 days ago (5s) 27
df_Sales_union.write.format("delta").mode("overwrite").save("abfss://silver@strgaccf0.dfs.core.windows.net/AdventureWorks_Sales/")
```

(3) Spark Jobs

Fig: After Union Operation Save Into Silver Layer

TRANSFORMATION USING DATABRICKS CONTINUED...

Data Transformation and Load into Silver Layer Continued...

The screenshot shows a Databricks notebook titled "Returns Data Transformation: Bronze to Silver". It contains a single code cell with the following Python code:

```
df_Returns = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("abfss://bronze@strgaccf0.dfs.core.windows.net/AdventureWorks_Returns/AdventureWorks_Returns.csv")
display(df_Returns)
```

The cell has a green checkmark icon and the number "22" indicating it has run successfully.

Fig: Returns Data Load Into DataBricks

The screenshot shows the same Databricks notebook with the code cell run. The results pane displays the data as a table:

ReturnDate	TerritoryKey	ProductKey	ReturnQuantity
2015-01-18	9	312	1
2015-01-18	10	310	1
2015-01-21	8	346	1
2015-01-22	4	311	1
2015-02-02	6	312	1
2015-02-15	1	312	1
2015-02-19	9	311	1
2015-02-24	8	314	1
2015-03-08	8	350	1
2015-03-13	9	350	1

Fig: Returns Data Display

The screenshot shows the Databricks notebook with the following code cell:

```
df_Returns.write.format("delta").mode("overwrite").save("abfss://silver@strgaccf0.dfs.core.windows.net/AdventureWorks_Returns")
```

The cell has a green checkmark icon and the number "23" indicating it has run successfully.

Fig: Returns Data Save Into Silver Layer

TRANSFORMATION USING DATABRICKS CONTINUED...

Data Transformation and Load into Silver Layer Continued...

Territories Data Transformations: Bronze to Silver

```
File Edit View Run Help Python Tabs: ON Last edit was now Run all Terminated Schedule
29
df_Territories = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("abfss://bronze@strgaccf0.dfs.core.windows.net/AdventureWorks_Territories/AdventureWorks_Territories.csv")
display(df_Territories)
(3) Spark Jobs
df_Territories: pyspark.sql.connect.DataFrame = [SalesTerritoryKey: integer, Region: string ... 2 more fields]
Table + 
SalesTerritoryKey Region Country Continent
1 Northwest United States North America
2 Northeast United States North America
3 Central United States North America
4 Southwest United States North America
```

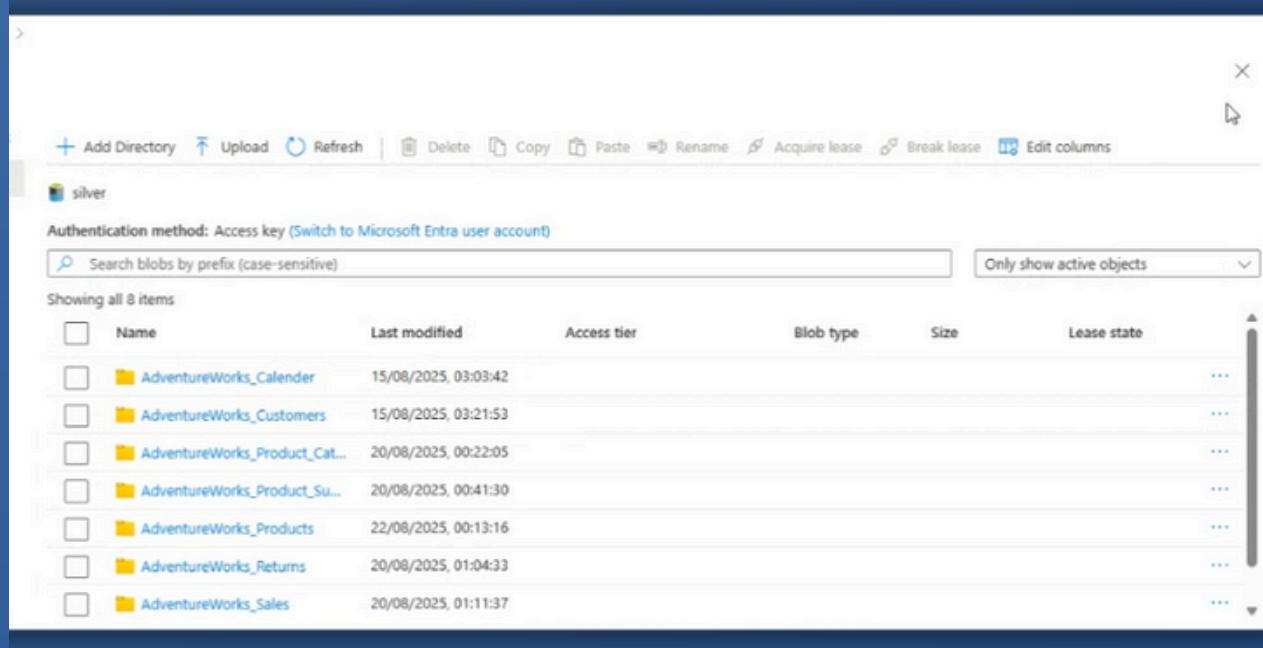
```
File Edit View Run Help Python Tabs: ON Last edit was now Run all Terminated Schedule
30
df_Territories.write.format("delta").mode("overwrite").save("abfss://silver@strgaccf0.dfs.core.windows.net/AdventureWorks_Territories")
(2) Spark Jobs
```

Fig: Territories Data Save Into Silver Layer

Fig: Territories Data Load Into DataBricks

TRANSFORMATION USING DATABRICKS CONTINUED...

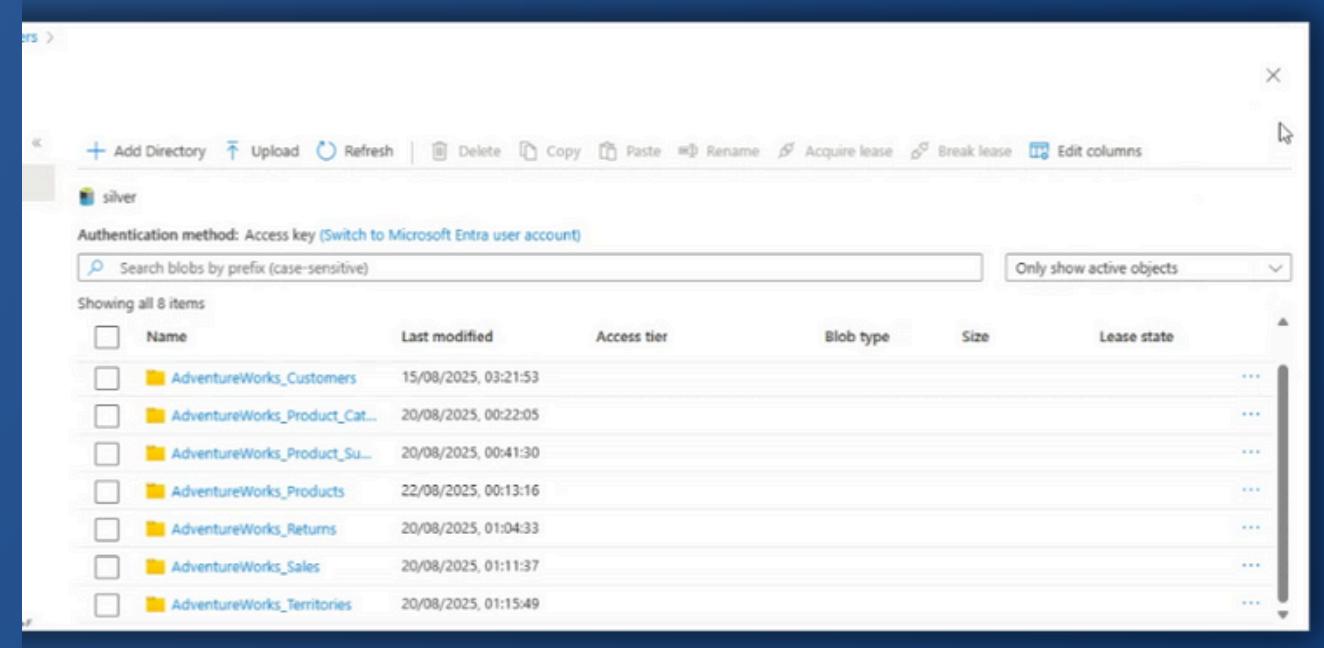
Successful Data Ingestion: Bronze to Silver



A screenshot of the Azure Storage Explorer interface. The left sidebar shows a single folder named "silver". The main pane displays a list of 8 items under the heading "Showing all 8 items". The columns are: Name, Last modified, Access tier, Blob type, Size, and Lease state. All items are of type "Block blob" and have a yellow icon next to their names. The items listed are: AdventureWorks_Calendar, AdventureWorks_Customers, AdventureWorks_Product_Cat..., AdventureWorks_Product_Su..., AdventureWorks_Products, AdventureWorks_Returns, AdventureWorks_Sales, and AdventureWorks_Territories. The last modified dates range from 15/08/2025 to 20/08/2025.

Name	Last modified	Access tier	Blob type	Size	Lease state
AdventureWorks_Calendar	15/08/2025, 03:03:42		Block blob		
AdventureWorks_Customers	15/08/2025, 03:21:53		Block blob		
AdventureWorks_Product_Cat...	20/08/2025, 00:22:05		Block blob		
AdventureWorks_Product_Su...	20/08/2025, 00:41:30		Block blob		
AdventureWorks_Products	22/08/2025, 00:13:16		Block blob		
AdventureWorks_Returns	20/08/2025, 01:04:33		Block blob		
AdventureWorks_Sales	20/08/2025, 01:11:37		Block blob		

Fig: Transformed Data in Silver Layer



A screenshot of the Azure Storage Explorer interface. The left sidebar shows a single folder named "silver". The main pane displays a list of 8 items under the heading "Showing all 8 items". The columns are: Name, Last modified, Access tier, Blob type, Size, and Lease state. All items are of type "Block blob" and have a yellow icon next to their names. The items listed are: AdventureWorks_Customers, AdventureWorks_Product_Cat..., AdventureWorks_Product_Su..., AdventureWorks_Products, AdventureWorks_Returns, AdventureWorks_Sales, and AdventureWorks_Territories. The last modified dates range from 15/08/2025 to 20/08/2025.

Name	Last modified	Access tier	Blob type	Size	Lease state
AdventureWorks_Customers	15/08/2025, 03:21:53		Block blob		
AdventureWorks_Product_Cat...	20/08/2025, 00:22:05		Block blob		
AdventureWorks_Product_Su...	20/08/2025, 00:41:30		Block blob		
AdventureWorks_Products	22/08/2025, 00:13:16		Block blob		
AdventureWorks_Returns	20/08/2025, 01:04:33		Block blob		
AdventureWorks_Sales	20/08/2025, 01:11:37		Block blob		
AdventureWorks_Territories	20/08/2025, 01:15:49		Block blob		

Fig: Transformed Data in Silver Layer

TRANSFORMATION USING DATABRICKS CONTINUED...

Data Aggregation and Storing into Gold Layer:

Data Transformation: Silver to Gold using PySpark

```
from pyspark.sql.functions import *
from pyspark.sql.types import *
```

Fig: Importing All Functions

Data Transformation: Bronze to Silver using PySpark

Granting Databricks access to ADLS via Service Principal

```
sec_id= "f107c110-05bb-44d8-803e-d5a225b00f69"
app_id= "22ff6450-d6d9-410b-9522-500c523bfad8"
strgacc = "strgaccf0"
```

Fig: Granting DataBricks Access to ADLS via Service Principal

```
spark.conf.set("fs.azure.account.auth.type", "OAuth")
spark.conf.set("fs.azure.account.oauth.provider.type", "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider")
spark.conf.set("fs.azure.account.oauth2.client.id", strgacc)
spark.conf.set("fs.azure.account.oauth2.client.secret", strgacc)
spark.conf.set("fs.azure.account.oauth2.endpoint", "https://login.microsoftonline.com/22ff6450-d6d9-410b-9522-500c523bfad8/oauth2/token")
```

Fig: Granting DataBricks Access to ADLS via Service Principal

TRANSFORMATION USING DATABRICKS CONTINUED...

Data Aggregation and Storing into Gold Layer Continued...

Monthly Sales & Profit Summary: Silver to Gold

File Edit View Run Help Python Tabs: ON Last edit was now Run all Terminated Schedule Share

```
Yesterday (1s) 8 Python
```

```
Product_df = spark.read.format("delta").load("abfss://silver@strgaccf0.dfs.core.windows.net/AdventureWorks_Products/")
Product_df = Product_df.drop("ProductDescription")
display(Product_df)
```

▶ (1) Spark Jobs

▶ Product_dt: pyspark.sql.connect.DataFrame = [ProductKey: integer, ProductSubcategoryKey: integer ... 8 more fields]

Table	+	🔍	✖	☰	□	
ProductKey	ProductSubcategoryKey	ProductSKU	ProductName	ModelName	ProductCo	
1	214	31	HL-U509-R	Sport-100 Helmet, Red	Sport-100	Red
2	215	31	HL-U509	Sport-100 Helmet, Black	Sport-100	Black
3	218	23	SO-B909-M	Mountain Bike Socks, M	Mountain Bike Socks	White
4	219	23	SO-B909-L	Mountain Bike Socks, L	Mountain Bike Socks	White
5	220	31	HL-U509-B	Sport-100 Helmet, Blue	Sport-100	Blue
6	223	19	CA-1098	AWC Logo Cap	Cycling Cap	Multi
7	226	21	UJ-0192-S	Long-Sleeve Logo Jersey, S	Long-Sleeve Logo Jers...	Multi
8	229	21	I-J-0192-M	Long-Sleeve Logo Jersey, M	Long-Sleeve Logo Jers...	Multi

Fig: Product Data Load Into DataBricks From Silver Layer

```
sales_df = spark.read.format("delta").load("abfss://silver@strgaccf0.dfs.core.windows.net/AdventureWorks_Sales/")
display(sales_df)

▶ (1) Spark Jobs

▶ sales_df: pyspark.sql.connect.DataFrame = [OrderDate: date, StockDate: date ... 6 more fields]

Table +
```

	OrderDate	StockDate	OrderNumber	ProductKey	CustomerKey	TerritoryKey	OrderID
1	2015-01-20	2001-11-02	SO45195	313	18929	9	
2	2015-03-13	2002-02-01	SO45661	342	25827	9	
3	2015-03-27	2002-02-02	SO45754	311	11529	1	
4	2015-03-29	2002-01-20	SO45769	314	11327	4	
5	2015-04-18	2002-03-08	SO45931	310	20249	9	
6	2015-04-25	2002-01-27	SO45978	350	11967	9	
7	2015-06-11	2002-02-18	SO46458	310	12771	4	
8	2015-06-20	2002-05-28	SO46525	314	20816	9	
9	2015-07-26	2002-06-24	SO46991	363	12249	9	

Fig: Sales Data Load Into DataBricks From Silver Layer

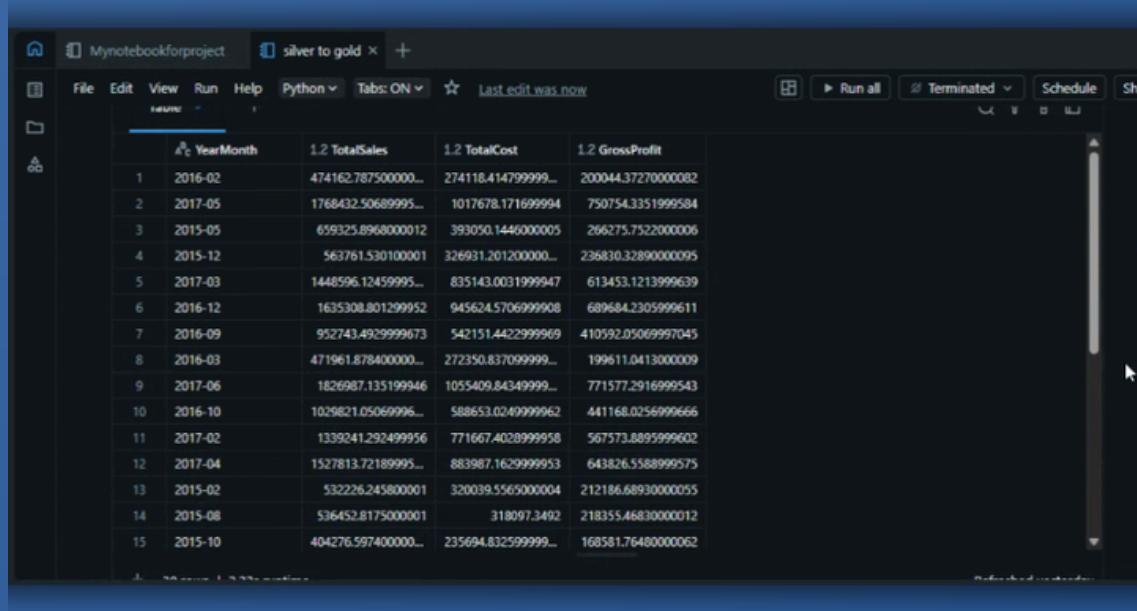
```
▶ [✓] Yesterday (2s) 10
from pyspark.sql.functions import col, date_format, sum as _sum

monthly_sales = (
    sales_df.alias("s")
    .join(Product_df.alias("p"), col("s.ProductKey") == col("p.ProductKey"))
    .withColumn("YearMonth", date_format(col("s.OrderDate"), "yyyy-MM"))
    .groupBy("YearMonth")
    .agg(
        _sum(col("s.OrderQuantity") * col("p.ProductPrice")).alias("TotalSales"),
        _sum(col("s.OrderQuantity") * col("p.ProductCost")).alias("TotalCost"),
        (
            _sum(col("s.OrderQuantity") * col("p.ProductPrice")) -
            _sum(col("s.OrderQuantity") * col("p.ProductCost"))
        ).alias("GrossProfit")
    )
)
display(monthly_sales)
```

Fig: Monthly Sales Aggregation

TRANSFORMATION USING DATABRICKS CONTINUED...

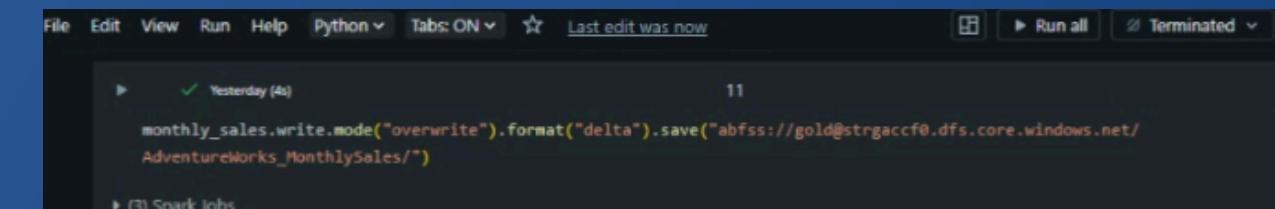
Data Aggregation and Storing into Gold Layer Continued...



A screenshot of a Databricks notebook titled "Mynotebookforproject" with a tab named "silver to gold". The notebook interface includes a top navigation bar with File, Edit, View, Run, Help, Python, Tabs: ON, and a status bar indicating "Last edit was now". Below the navigation is a sidebar with a tree view and a search bar. The main area displays a DataFrame with four columns: "YearMonth", "1.2 TotalSales", "1.2 TotalCost", and "1.2 GrossProfit". The data consists of 15 rows of monthly sales figures from 2015-02 to 2017-05.

	YearMonth	1.2 TotalSales	1.2 TotalCost	1.2 GrossProfit
1	2016-02	474162.787500000...	274118.414799999...	200044.37270000082
2	2017-05	1768432.506899995...	1017678.171699994	750754.3351999584
3	2015-05	659325.8968000012	393050.1446000005	266275.752000006
4	2015-12	563761.530100001	326931.201200000...	236830.32890000095
5	2017-03	1448596.12459995...	835143.0031999947	613453.1213999639
6	2016-12	1635308.801299952	945624.5706999908	689684.2305999611
7	2016-09	952743.4929999673	542151.4422999969	410592.05069997045
8	2016-03	471961.878400000...	272350.83709999...	199611.0413000009
9	2017-06	1826987.135199946	1055409.84349999...	771577.2916999543
10	2016-10	1029821.05069996...	588653.0240999962	441168.0256999666
11	2017-02	1339241.292499956	771667.4020999958	567573.8895999602
12	2017-04	1527813.72189995...	883987.162999953	643826.5588999575
13	2015-02	532226.245800001	320039.5565000004	212186.68930000055
14	2015-08	536452.817500001	318097.3492	218355.46830000012
15	2015-10	404276.597400000...	235694.832599999...	168581.76480000062

Fig: Monthly Sales Data Display



A screenshot of a Databricks notebook titled "Mynotebookforproject" with a tab named "silver to gold". The notebook interface includes a top navigation bar with File, Edit, View, Run, Help, Python, Tabs: ON, and a status bar indicating "Last edit was now". Below the navigation is a sidebar with a tree view and a search bar. The main area shows a PySpark command in a code cell:

```
monthly_sales.write.mode("overwrite").format("delta").save("abfss://gold@strgaccf0.dfs.core.windows.net/AdventureWorks_MonthlySales/")
```

The command is highlighted in yellow. A success message "Yesterday (4s)" is shown above the command, and "11" is displayed below it. A "Spark Jobs" section is visible at the bottom of the notebook.

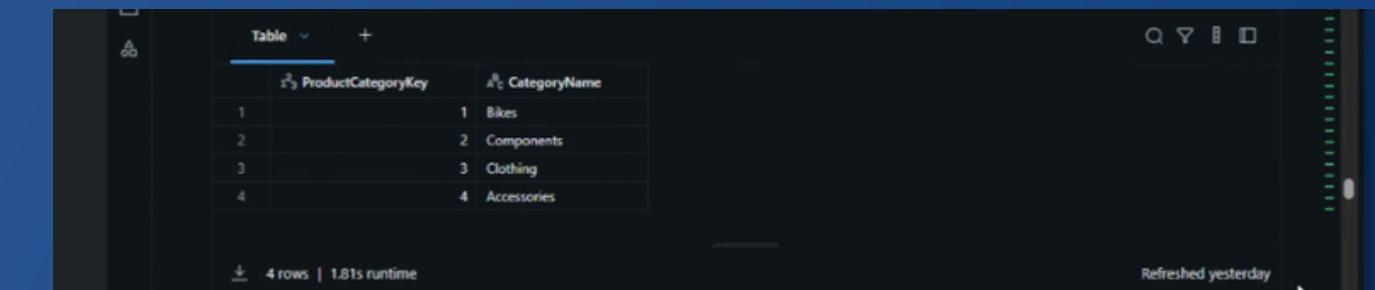
Fig: Monthly Sales Data Save Into Gold Layer

TRANSFORMATION USING DATABRICKS CONTINUED...

Data Aggregation and Storing into Gold Layer Continued...:



```
Sales by Product Category: Silver to Gold
+ Code + Text Assistant
Yesterday (2s) 13 Python
ProductCategory_df = spark.read.format("delta").load("abfss://silver@strgaccf0.dfs.core.windows.net/AdventureWorks_Product_Categories/")
display(ProductCategory_df)
(1) Spark Jobs
ProductCategory_df: pyspark.sql.connect.DataFrame = [ProductCategoryKey: integer, CategoryName: string]
Table +
```



ProductCategoryKey	CategoryName
1	Bikes
2	Components
3	Clothing
4	Accessories

4 rows | 1.81s runtime
Refreshed yesterday

Fig: Product Category Data Display

Fig: Product Category Data Load Into DataBricks From Silver Layer



```
14 Python
ProductSubcategory_df = spark.read.format("delta").load("abfss://silver@strgaccf0.dfs.core.windows.net/AdventureWorks_Product_SubCategories/")
display(ProductSubcategory_df)
(1) Spark Jobs
ProductSubcategory_df: pyspark.sql.connect.DataFrame = [ProductSubcategoryKey: integer, Sub categoryName: string]
```



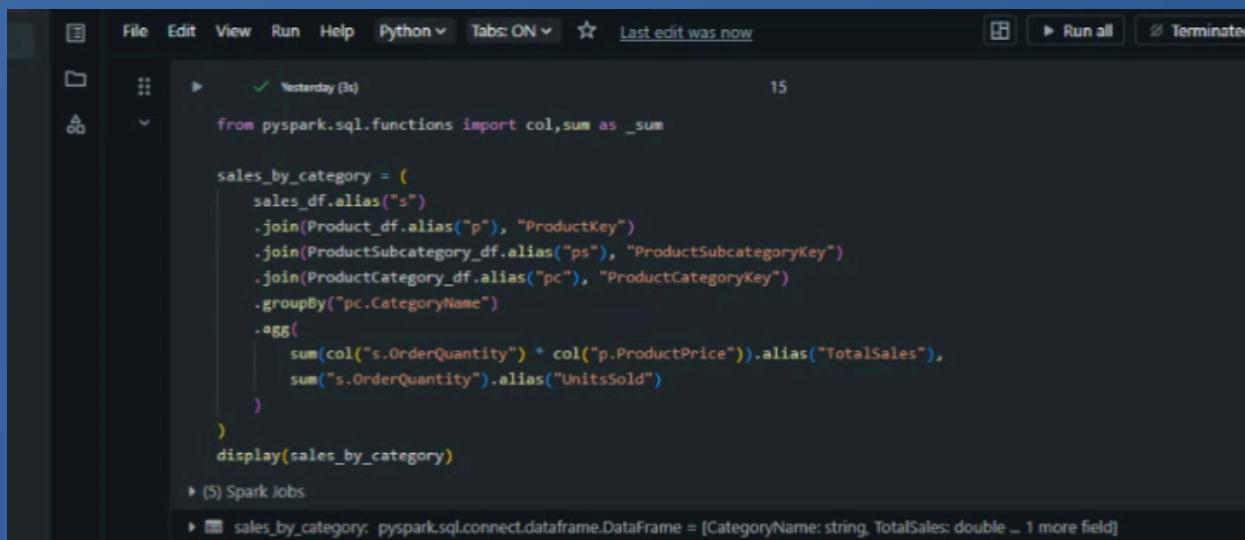
ProductSubcategoryKey	Sub categoryName	ProductCategoryKey
1	Mountain Bikes	1
2	Road Bikes	1
3	Touring Bikes	1
4	Handlebars	2
5	Bottom Brackets	2
6	Brakes	2
7	Chains	2
8	Cranksets	2
9	Deraileurs	2
10	Forks	2
11	Headsets	2
12	Mountain Frames	2

Fig: Product SubCategory Data Load Into DataBricks From Silver Layer

Fig: Product SubCategory Data Display

TRANSFORMATION USING DATABRICKS CONTINUED...

Data Aggregation and Storing into Gold Layer Continued...:



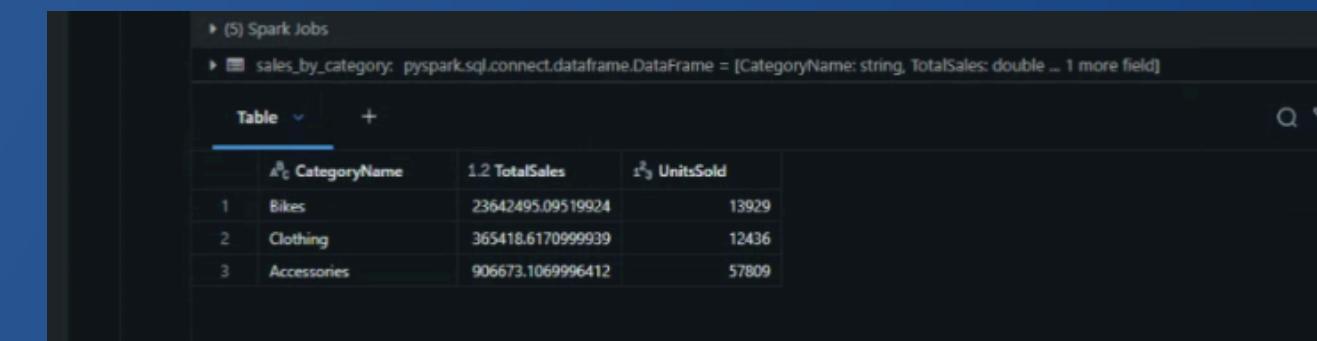
A screenshot of a Databricks notebook titled "Yesterday (3s)". The code cell contains the following Python code:

```
from pyspark.sql.functions import col,sum as _sum

sales_by_category = (
    sales_df.alias("s")
    .join(Product_df.alias("p"), "ProductKey")
    .join(ProductSubcategory_df.alias("ps"), "ProductSubcategoryKey")
    .join(ProductCategory_df.alias("pc"), "ProductCategoryKey")
    .groupBy("pc.CategoryName")
    .agg(
        sum(col("s.OrderQuantity") * col("p.ProductPrice")).alias("TotalSales"),
        sum("s.OrderQuantity").alias("UnitsSold")
    )
)
display(sales_by_category)
```

The code uses PySpark's DataFrame API to perform a multi-table join and group by operation to calculate total sales and units sold by product category. The resulting DataFrame is then displayed.

Fig: Sales By Product Category Aggregation



A screenshot of a Databricks notebook showing the results of the aggregation query. The results are displayed in a table:

CategoryName	TotalSales	UnitsSold
Bikes	2364295.09519924	13929
Clothing	365418.6170999939	12436
Accessories	906673.1069996412	57809

Fig: Sales By Product Category Data Display



A screenshot of a Databricks notebook titled "Aug 22, 2025 (6s)". The code cell contains the following Python code:

```
sales_by_category.write.mode("overwrite").format("delta").save("abfss://gold@strgaccf0.dfs.core.windows.net/AdventureWorks_SalesByCategory/")
```

The code uses PySpark's DataFrame API to save the aggregated sales data into a Delta Lake table named "AdventureWorks_SalesByCategory" in the "gold" container of a Azure Blob File System (ABFS).

Fig: Sales By Product Category Data Save Into Gold Layer

TRANSFORMATION USING DATABRICKS CONTINUED...

Data Aggregation and Storing into Gold Layer Continued...

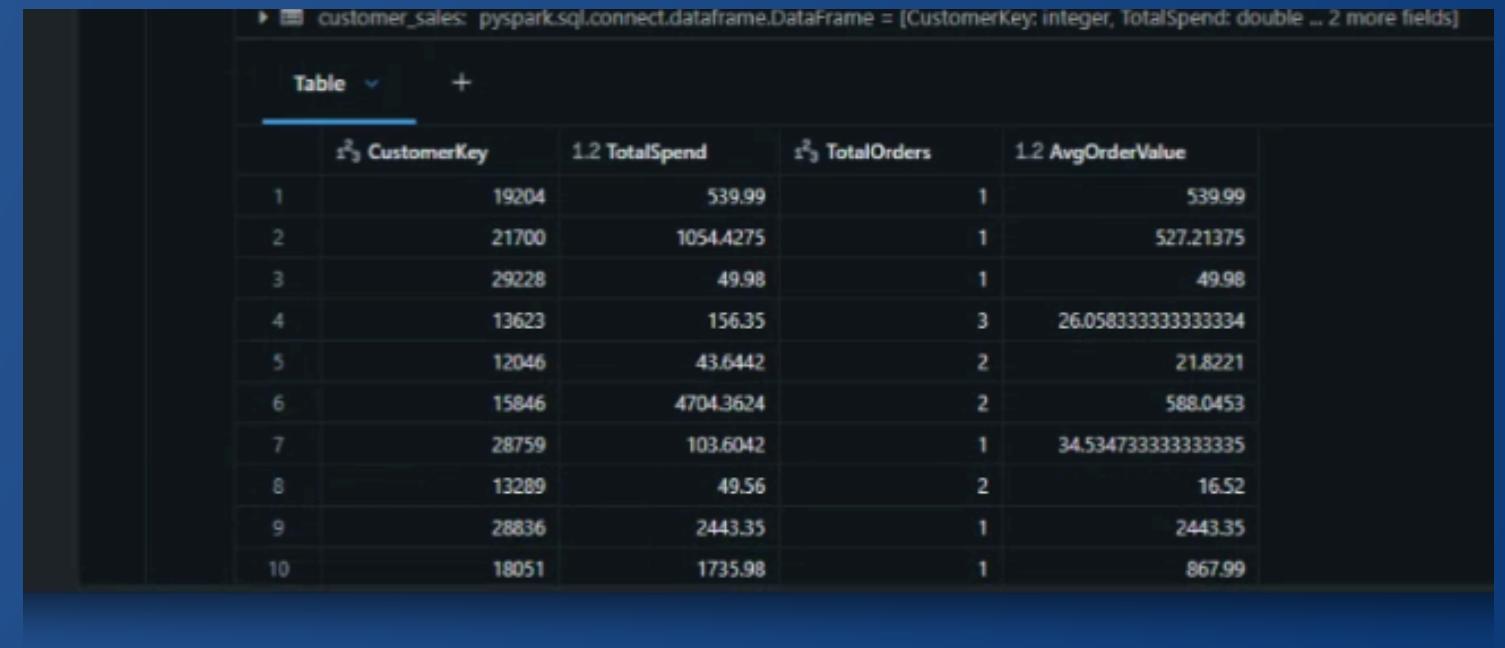


The screenshot shows a Databricks notebook titled "Customer Insights (Lifetime Value): Silver to Gold". The code cell contains the following Python code:

```
from pyspark.sql import functions as F

customer_sales = (
    sales_df.alias("s")
    .join(Product_df.alias("p"), "ProductKey")
    .groupBy("s.CustomerKey")
    .agg(
        F.sum(F.col("s.OrderQuantity") * F.col("p.ProductPrice")).alias("TotalSpend"),
        F.countDistinct("s.OrderNumber").alias("TotalOrders"),
        F.avg(F.col("s.OrderQuantity") * F.col("p.ProductPrice")).alias("AvgOrderValue")
    )
)
display(customer_sales)
```

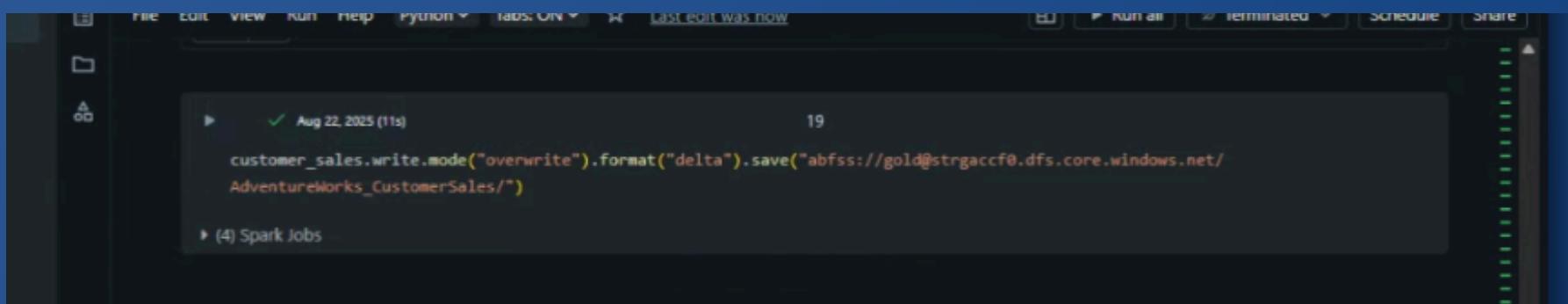
Fig: Customer Insights(Lifetime Value) Aggregation



The screenshot shows a Databricks notebook displaying the resulting DataFrame named "customer_sales". The table has four columns: CustomerKey, TotalSpend, TotalOrders, and AvgOrderValue. The data is as follows:

CustomerKey	TotalSpend	TotalOrders	AvgOrderValue
1	539.99	1	539.99
2	1054.4275	1	527.21375
3	49.98	1	49.98
4	156.35	3	26.05833333333334
5	43.6442	2	21.8221
6	4704.3624	2	588.0453
7	103.6042	1	34.53473333333335
8	49.56	2	16.52
9	2443.35	1	2443.35
10	1735.98	1	867.99

Fig: Customer Insights(Lifetime Value) Data Display



The screenshot shows a Databricks notebook with the following code in a cell:

```
customer_sales.write.mode("overwrite").format("delta").save("abfss://gold@strgaccf0.dfs.core.windows.net/AdventureWorks_CustomerSales/")
```

Fig: Customer Insights(Lifetime Value) Data Save Into Gold Layer

TRANSFORMATION USING DATABRICKS CONTINUED...

Data Aggregation and Storing into Gold Layer Continued...

The screenshot shows a Databricks notebook cell titled "Returns Rate by Product: Silver to Gold". The code in the cell is:

```
Return_df = spark.read.format("delta").load("abfss://silver@strgaccf0.dfs.core.windows.net/AdventureWorks_Returns/")  
display(Return_df)
```

Below the cell, it shows "(1) Spark Jobs" and "Return_df: pyspark.sql.connect.DataFrame = [ReturnDate: string, TerritoryKey: integer ... 2 more fields]".

Fig: Returns Data Load Into DataBricks From Silver

The screenshot shows a Databricks notebook cell titled "Yesterday (2s)". The code in the cell is:

```
import pyspark.sql.functions as F  
returns_summary = (  
    Return_df.alias("r")  
    .join(Product_df.alias("p"), "ProductKey")  
    .groupBy("r.ProductKey")  
    .agg(  
        F.sum("r.ReturnQuantity").alias("TotalReturnedUnits"),  
        F.sum(F.col("r.ReturnQuantity") * F.col("p.ProductPrice")).alias("ReturnAmount")  
    )  
)  
display(returns_summary)
```

Below the cell, it shows "(3) Spark Jobs" and "returns_summary: pyspark.sql.connect.DataFrame = [ProductKey: integer, TotalReturnedUnits: long ... 1 more field]".

Fig: Returns Rate By Product Aggregation

The screenshot shows a Databricks notebook cell titled "Table". It displays a table with the following data:

	ReturnDate	TerritoryKey	ProductKey	ReturnQuantity
1	01/18/2015	9	312	1
2	01/18/2015	10	310	1
3	01/21/2015	8	346	1
4	01/22/2015	4	311	1
5	02/02/2015	6	312	1
6	02/15/2015	1	312	1
7	02/18/2015	0	311	1

Fig: Returns Data Display

The screenshot shows a Databricks notebook cell titled "Table". It displays a table with the following data:

	ProductKey	TotalReturnedUnits	ReturnAmount
1	471	8	508
2	540	28	912.800000000004
3	580	7	11906.93
4	481	11	98.8899999999999
5	588	5	3847.45
6	472	7	444.5
7	322	2	1398.1964
8	362	18	36883.7676
9	375	8	17452.5
10	501	2	1129.08

Fig: Returns Rate By Product Data Display

The screenshot shows a Databricks notebook cell titled "Yesterday (4s)". The code in the cell is:

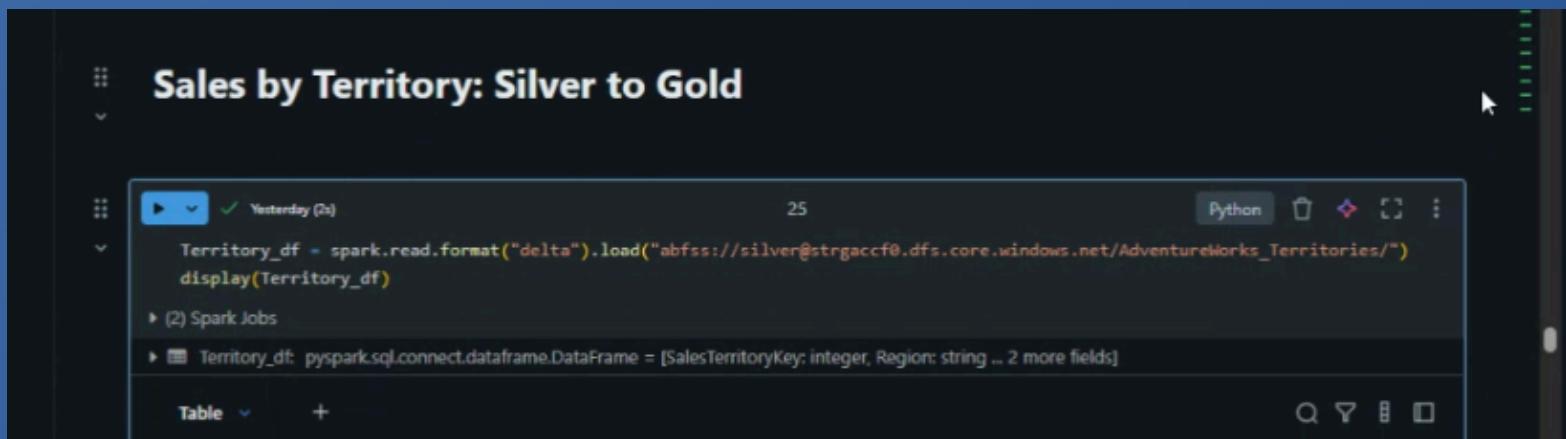
```
returns_summary.write.mode("overwrite").format("delta").save("abfss://gold@strgaccf0.dfs.core.windows.net/AdventureWorks_Returns/")
```

Below the cell, it shows "(3) Spark Jobs".

Fig: Returns Rate By Product Data Save Into Gold Layer

TRANSFORMATION USING DATABRICKS CONTINUED...

Data Aggregation and Storing into Gold Layer Continued...



```
Sales by Territory: Silver to Gold

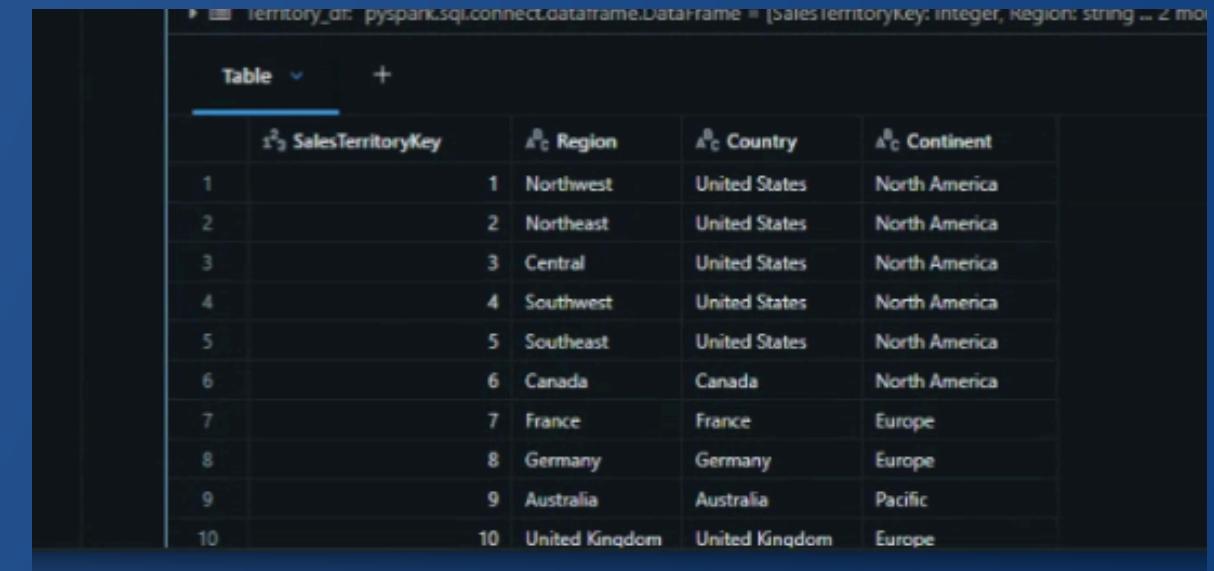
25
Python

Territory_df = spark.read.format("delta").load("abfss://silver@strgaccf0.dfs.core.windows.net/AdventureWorks_Territories/")
display(Territory_df)

(2) Spark Jobs
 Territory_dt: pyspark.sql.connect.DataFrame = [SalesTerritoryKey: integer, Region: string ... 2 more fields]

Table +
```

Fig: Territory Data Load Into DataBricks From Silver

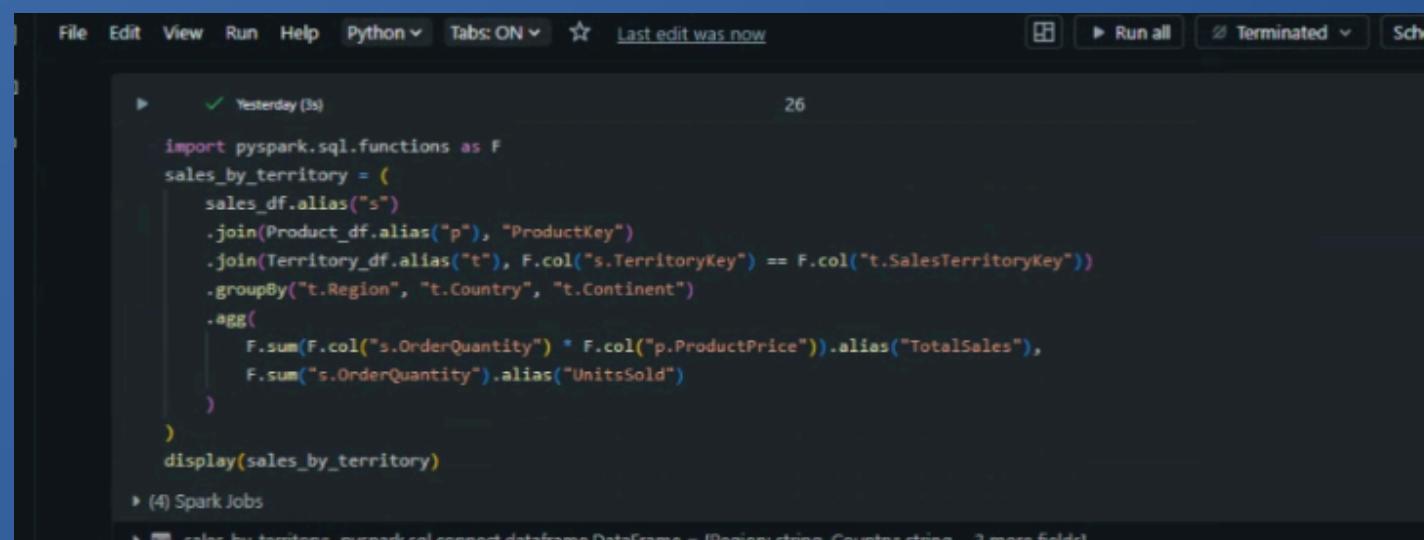


	SalesTerritoryKey	Region	Country	Continent
1	1	Northwest	United States	North America
2	2	Northeast	United States	North America
3	3	Central	United States	North America
4	4	Southwest	United States	North America
5	5	Southeast	United States	North America
6	6	Canada	Canada	North America
7	7	France	France	Europe
8	8	Germany	Germany	Europe
9	9	Australia	Australia	Pacific
10	10	United Kingdom	United Kingdom	Europe

Fig: Territory Data Display

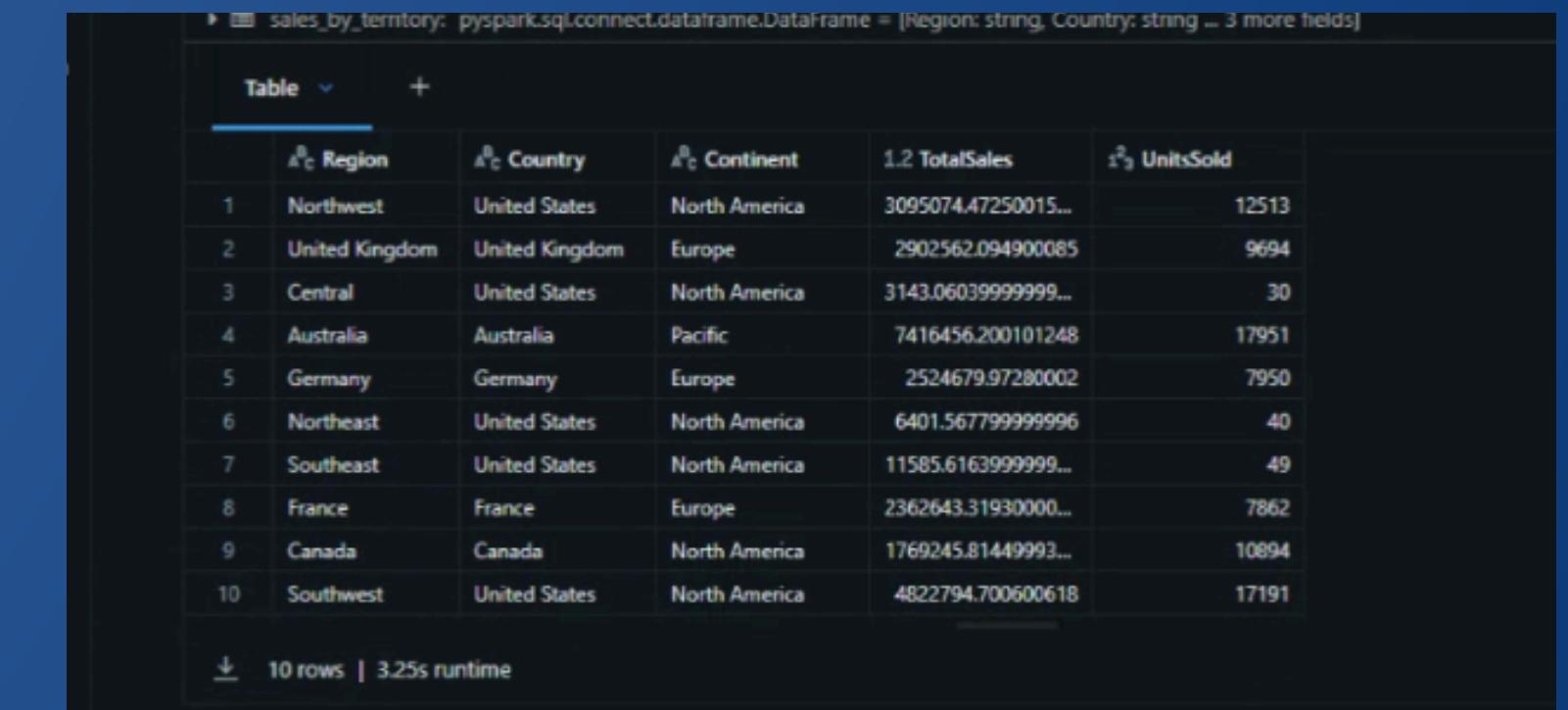
TRANSFORMATION USING DATABRICKS CONTINUED...

Data Aggregation and Storing into Gold Layer Continued...



```
File Edit View Run Help Python ▾ Tabs: ON ▾ Last edit was now 26  
Yesterday (3s)  
import pyspark.sql.functions as F  
sales_by_territory = (  
    sales_df.alias("s")  
    .join(Product_df.alias("p"), "ProductKey")  
    .join(Territory_df.alias("t"), F.col("s.TerritoryKey") == F.col("t.SalesTerritoryKey"))  
    .groupBy("t.Region", "t.Country", "t.Continent")  
    .agg(  
        F.sum(F.col("s.OrderQuantity") * F.col("p.ProductPrice")).alias("TotalSales"),  
        F.sum("s.OrderQuantity").alias("UnitsSold")  
    )  
display(sales_by_territory)  
(4) Spark Jobs  
sales_by_territory: pyspark.sql.connect.DataFrame = [Region: string, Country: string ... 3 more fields]
```

Fig: Sales By Territory Aggregation

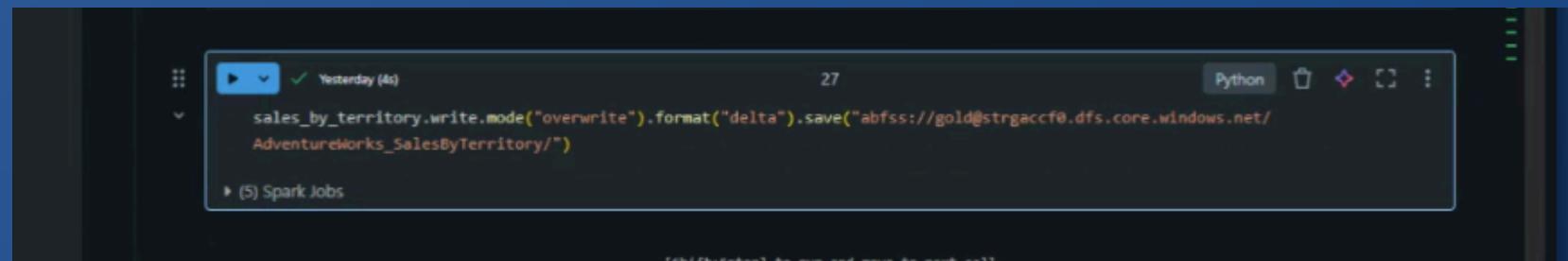


sales_by_territory: pyspark.sql.connect.DataFrame = [Region: string, Country: string ... 3 more fields]

	Region	Country	Continent	TotalSales	UnitsSold
1	Northwest	United States	North America	3095074.47250015...	12513
2	United Kingdom	United Kingdom	Europe	2902562.094900085	9694
3	Central	United States	North America	3143.06039999999...	30
4	Australia	Australia	Pacific	7416456.200101248	17951
5	Germany	Germany	Europe	2524679.97280002	7950
6	Northeast	United States	North America	6401.56779999996	40
7	Southeast	United States	North America	11585.6163999999...	49
8	France	France	Europe	2362643.3193000...	7862
9	Canada	Canada	North America	1769245.81449993...	10894
10	Southwest	United States	North America	4822794.700600618	17191

10 rows | 3.25s runtime

Fig: Sales By Territory Data Display

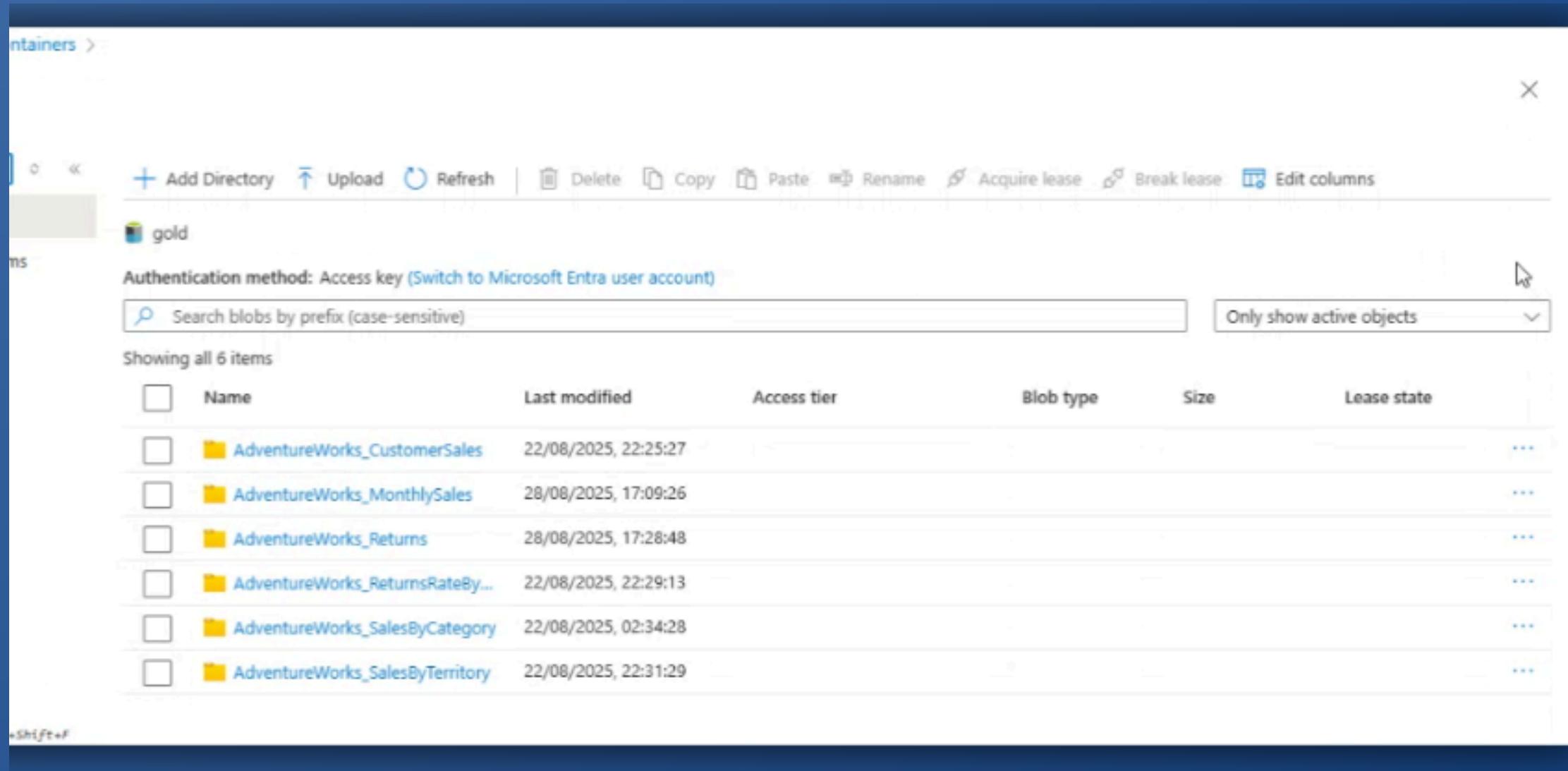


```
Yesterday (4s)  
27  
sales_by_territory.write.mode("overwrite").format("delta").save("abfss://gold@strgaccf0.dfs.core.windows.net/  
AdventureWorks_SalesByTerritory/")  
(5) Spark Jobs
```

Fig: Sales By Territory Data Save Into Gold Layer

TRANSFORMATION USING DATABRICKS CONTINUED...

Successful Data Store : Silver to Gold



Successful Data Store Into Gold Layer

FUTURE SCOPE

As a future scope of this project, I plan to connect the Gold layer with Power BI to generate meaningful business insights and interactive reports. This will help in visualizing transformed data and supporting data-driven decision-making.

CONCLUSION

This project successfully demonstrates the power of Azure Data Engineering technologies in building a scalable, secure, and automated data pipeline. By integrating Azure Data Factory for ingestion and orchestration, Databricks with PySpark for transformation, and Delta Lake for optimized storage, the solution ensures efficient handling of raw data from diverse sources. The architecture not only improves data quality and performance, but also enables seamless analytics and business intelligence, making it a robust foundation for real-world data-driven decision-making.

The background features a large, abstract graphic element composed of several overlapping blue arrows pointing towards the center. Below this, a dark blue silhouette of a city skyline is visible, featuring recognizable buildings like the Eiffel Tower and the Louvre Pyramid.

THANK YOU