

Step 1: Azure Environment Setup: Create below Azure services :

- 1) Create a resource group “azure-de-rg”
- 2) Create a Data Factory “azure-de-df” as below:

The screenshot shows the 'Create Data Factory' wizard in the Microsoft Azure portal. In the 'Project details' section, the subscription is set to 'Azure subscription 1' and the resource group is 'azure-de-rg'. Under 'Instance details', the name is 'azure-de-df', region is 'Southeast Asia', and version is 'V2'. At the bottom, there are 'Previous', 'Next', and 'Review + create' buttons, along with a 'Give feedback' link.

The screenshot shows the 'Microsoft.DataFactory-20250514215217 | Overview' page. It displays a summary: 'Your deployment is complete'. Deployment details include: Name: Microsoft.DataFactory-2025051..., Start time: 5/14/2025, 9:53:51 PM; Subscription: Azure subscription 1; Correlation ID: e03a5a68-e20a-4788-a431-bfd...; Resource group: azure-de-rg. There are sections for 'Deployment details', 'Next steps', and 'Give feedback'. On the right, there are promotional cards for 'Cost management', 'Microsoft Defender for Cloud', and 'Free Microsoft tutorials'.

- 3) Create a hierarchical Data lake Gen2 Storage Account , “storageaccazurede” as below:

The screenshot shows the 'Create a storage account' wizard in the Microsoft Azure portal. In the 'Subscription' and 'Resource group' fields, 'Azure subscription 1' and 'azure-de-rg' are selected. Under 'Instance details', the storage account name is 'storageaccazurede', region is '(Asia Pacific) Southeast Asia', and primary service is 'Azure Blob Storage or Azure Data Lake Storage Gen 2'. The 'Performance' section has 'Standard' selected. At the bottom, there are 'Previous', 'Next', and 'Review + create' buttons, along with a 'Give feedback' link.

Create a 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](#)

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

Enable SFTP

Previous **Next** **Review + create** **Give feedback**

ENG IN 21:59 14-05-2025

storageaccazuredede_1747240163637 | Overview

Your deployment is complete

Deployment name: storageaccazuredede_17472... Start time: 5/14/2025, 9:59:33 PM
Subscription: Azure subscription 1 Resource group: azure-de-rg

Deployment details

Next steps

Go to resource

Give feedback

Tell us about your experience with deployment

Cost Management
Get notified to stay within your budget and prevent unexpected charges on your bill. [Set up cost alerts >](#)

Microsoft Defender for Cloud
Secure your apps and infrastructure. [Go to Microsoft Defender for Cloud >](#)

Free Microsoft tutorials

ENG IN 22:00 14-05-2025

4) Create containers bronze/silver/gold inside the storage account → Containers

storageaccazuredede | Containers

Successfully created storage container
Successfully created storage container 'gold'.

| Name | Last modified | Anonymous access level | Lease state |
|------------|------------------------|------------------------|-------------|
| \$logs | 5/14/2025, 10:00:08 PM | Private | Available |
| bronze | 5/15/2025, 3:02:20 PM | Private | Available |
| gold | 5/15/2025, 3:02:42 PM | Private | Available |
| silver | 5/15/2025, 3:02:30 PM | Private | Available |
| synapse-fs | 5/14/2025, 10:09:44 PM | Private | Available |

Add container **Upload** **Refresh** **Delete** **Change access level** **Restore containers** **Edit columns**

Search containers by prefix

Showing all 5 items

Container Actions

Storage browser

Partner solutions

Resource visualizer

Data storage

Containers

File shares

Queues

Tables

Security + networking

Data management

Settings

Top Stories Rajnath Singh C...

ENG IN 15:02 15-05-2025

5) Create Databricks Workspace, “azure-de-db”

The screenshot shows the 'Create an Azure Databricks workspace' wizard. In the 'Subscription' dropdown, 'Azure subscription 1' is selected. Under 'Resource group', 'azure-de-rg' is chosen. The 'Workspace name' field contains 'azure-de-db'. The 'Region' is set to 'Southeast Asia' and the 'Pricing Tier' is 'Premium (+ Role-based access controls)'. A tooltip indicates that the selected tier is the recommended one. At the bottom, there are 'Review + create' and 'Next : Networking >' buttons.

The second part of the screenshot shows the 'Overview' page for the 'azure-de-rg_azure-de-db' resource group. It displays deployment details: Deployment name: 'azure-de-rg_azure-de-db', Subscription: 'Azure subscription 1', Resource group: 'azure-de-rg', Start time: 5/14/2025, 10:03:26 PM, Correlation ID: 'db917e39-dba8-4770-9ad9-9c61959d05f3'. The 'Notifications' sidebar shows three successful deployment logs from the activity log.

6) Create Synapse analytics workspace, “azure-de-sa”

The screenshot shows the 'Create Synapse workspace' wizard. The 'Workspace name' is 'azure-de-sa', 'Region' is 'Southeast Asia', and 'Select Data Lake Storage Gen2' is set to 'From subscription'. An account named 'storageaccazurede' is selected, and a file system named '(New) synapse-fs' is chosen. A checkbox for assigning the 'Storage Blob Data Contributor' role is checked. At the bottom, there are 'Review + create' and 'Next: Security >' buttons.

Choose the authentication method for access to workspace resources such as SQL pools. The authentication method can be changed later on. [Learn more](#)

Authentication method Use both local and Microsoft Entra ID authentication Use only Microsoft Entra ID authentication

SQL Server admin login *

SQL Password

Confirm password

System assigned managed identity permission

Select to grant the workspace network access to the Data Lake Storage Gen2 account using the workspace system identity. [Learn more](#)

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

Deployment

Overview

Your deployment is complete

Deployment name : Microsoft.Azure.SynapseAnalytics-20250514221548 Start time : 5/14/2025, 10:17:15 PM
Subscription : Azure subscription 1 Correlation ID : 05c51d84-4870-4348-8923-2b...
Resource group : azure-de-rg

Deployment details

Next steps

Go to resource group

Give feedback

Tell us about your experience with deployment

Add or remove favorites by pressing **Ctrl + Shift + F**

Air: Severe Saturday

7) Create a Key Vault

Create a key vault

Subscription *

Resource group * [Create new](#)

Instance details

Key vault name *

Region *

Pricing tier *

Recovery options

Soft delete protection will automatically be enabled on this key vault. This feature allows you to recover or permanently delete a key vault and secrets for the duration of the retention period. This protection applies to the key vault and the secrets stored within the key vault.

[Previous](#) [Next](#) [Review + create](#) [Give feedback](#)

Create a key vault

Review + create

Basics

| | |
|--|----------------------|
| Subscription | Azure subscription 1 |
| Resource group | azimuthe-de-rg |
| Key vault name | azimuthe-de-kv |
| Region | Southeast Asia |
| Pricing tier | Standard |
| Soft-delete | Enabled |
| Purge protection during retention period | Disabled |
| Days to retain deleted vaults | 90 days |

Previous **Next** **Create** **Give feedback**

azimuthe-de-kv | Overview

Your deployment is complete

Deployment name : azimuthe-de-kv
Subscription : Azure subscription 1
Resource group : azimuthe-de-rg

Start time : 5/15/2025, 3:08:08 PM
Correlation ID : eb6c0ac7-8518-415b-881d-9e6...

Deployment details

Next steps

Go to resource

Give feedback

Tell us about your experience with deployment

Cost management
Get notified to stay within your budget and prevent unexpected charges on your bill.
Set up cost alerts >

Microsoft Defender for Cloud
Secure your apps and infrastructure
Go to Microsoft Defender for Cloud >

Free Microsoft tutorials
Start learning today >

30°C Mostly sunny **Search** **ENG IN** **15-05-2025**

Resource group elements now are:

Home > azimuthe-de-rg

Overview

Activity log

Access control (IAM)

Tags

Resource visualizer

Events

Settings

Cost Management

Monitoring

Automation

Search **Create** **Manage view** **Delete resource group** **Refresh** **Export to CSV** **Open query** **Assign tags** **Move**

Filter for any field... **Type equals all** **Location equals all** **Add filter**

Showing 1 to 5 of 5 records. **Show hidden types**

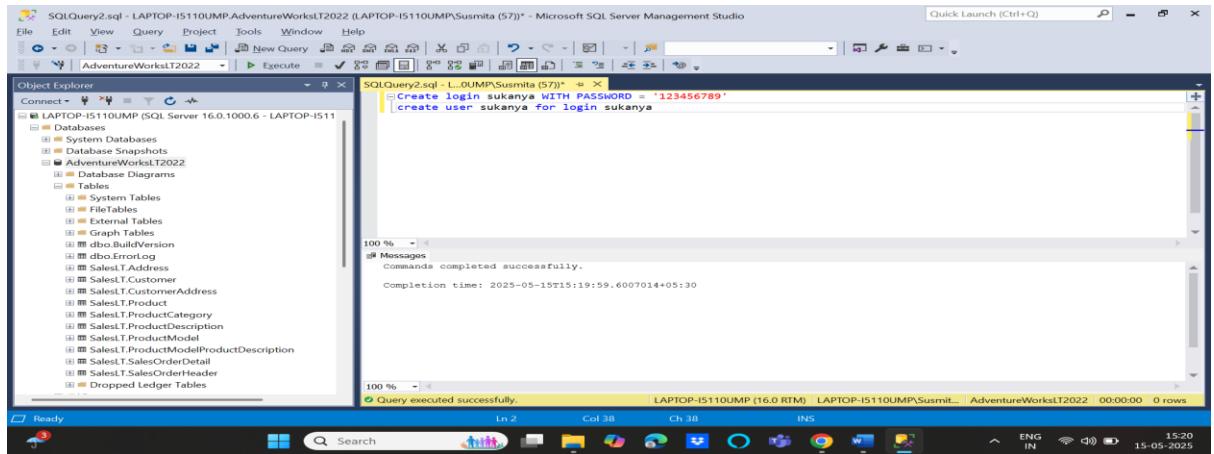
| Name | Type | Location |
|-------------------|--------------------------|----------------|
| azure-de-db | Azure Databricks Service | Southeast Asia |
| azure-de-df | Data factory (V2) | Southeast Asia |
| azure-de-kv | Key vault | Southeast Asia |
| azure-de-sa | Synapse workspace | Southeast Asia |
| storageaccazurede | Storage account | Southeast Asia |

No grouping **List view**

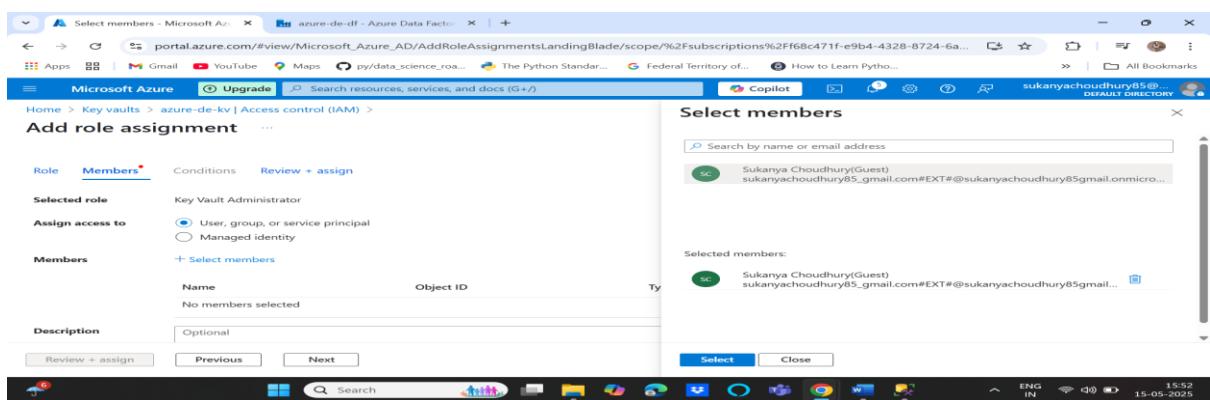
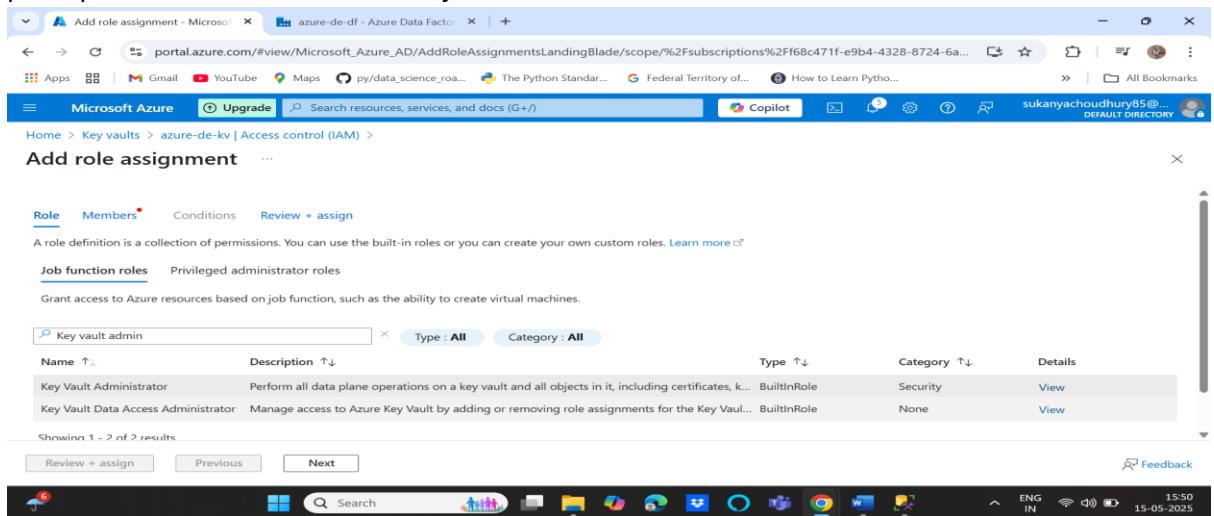
Step 2: Data Ingestion

- Extract customer and sales data from an on-premises SQL database.

- 1) Create an user into the SQL server database AdventureWorks which is downloaded and loaded into the SQL server management studio from Microsoft site.



- 2) Go to Create key vault → Access Control (IAM) Objects → Add → Add role assignment → Key Vault Administrator → Next → Members → user, group or service principal → select members → add yourself as a member



- 3) key vault → objects → Secrets → Generate/Import → add two keys: username(Sukanya) and password (from SSMS)

- 4) Add Key vault secret user to Data factory

Key vault → Access Control (IAM) → Add role assignment—Key Vault Secrets User →

Next → add members → Managed Identity → select members → select your Data factory → Review+assign → Create

- **Load the data into Azure Data Lake Storage (ADLS) using Azure Data Factory (ADF).**

1) Add a pipeline → Activities → Move and Transform → copy activity

2) Configure source and sink by configuring respective linked services and source dataset

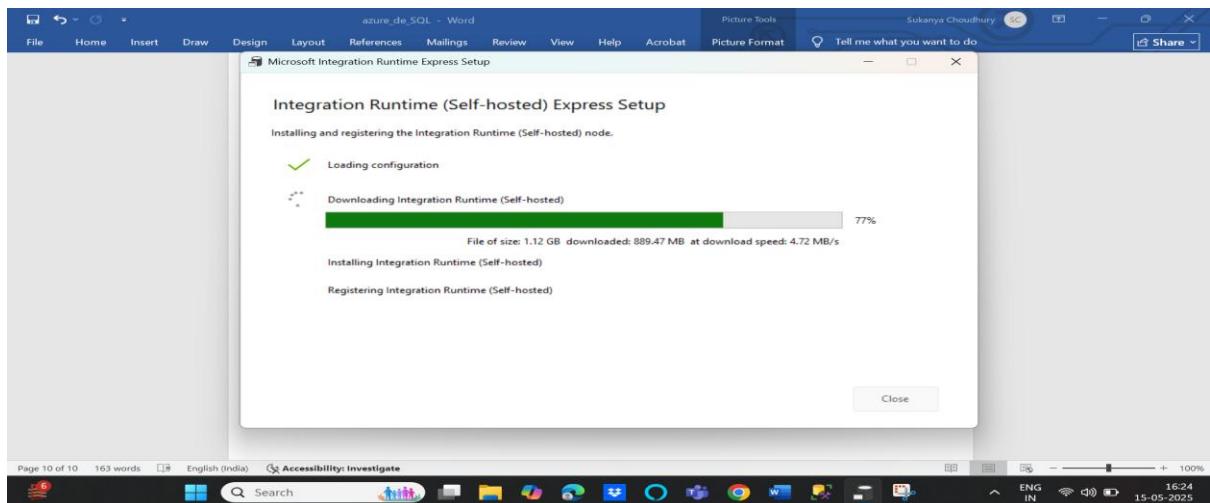
Go to Copydata → source → source dataset → + → SQL server (as our data is in On premise SQL server) → name

3) Then linked service → +New→ Configure linked service→ Connect via integration runtime→ new → self hosted(for onpremise database)

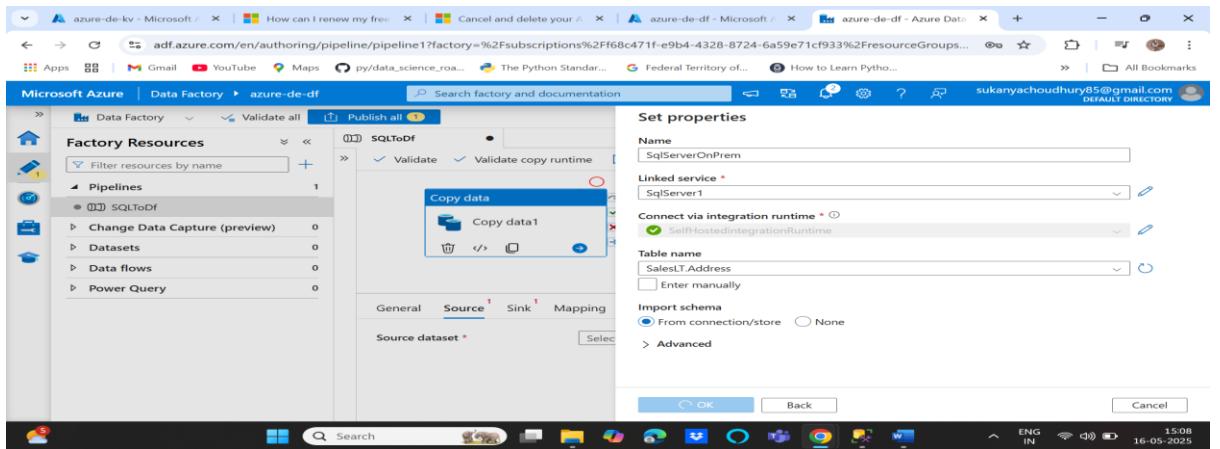
The screenshot shows two side-by-side browser windows. Both windows have the URL adf.azure.com/en/authoring/pipeline/pipeline1?factory=%2Fsubscriptions%2Ff68c471f-e9b4-4328-8724-6a59e71cf933%2FresourceGroups%2Fa.... The top window displays the 'Integration runtime setup' section, which includes options for 'Azure' (running data flows, data movement, external and pipeline activities in a fully managed, serverless compute in Azure), 'Self-Hosted' (running activities in an on-premises / private network), and 'Linked Self-Hosted' (reusing existing infrastructure). The bottom window shows the 'Copy data' activity configuration, where the 'Source dataset' is selected and the 'Type' is set to 'Self-Hosted'. Both windows show the same user interface for managing Azure Data Factory resources.

4) After creating→ Express Setup→ Click here to launch the express setup→ download and run the setup on your local machine—“SelfhostedIntegrateRuntime” is created

This screenshot shows the 'Integration runtime setup' section again. It highlights the 'Self-contained interactive authoring' option, with the 'Enable' radio button selected. Below this, there are two sections: 'Option 1: Express setup' (with a link to 'Click here to launch the express setup for this computer') and 'Option 2: Manual setup' (with steps for 'Download and install integration runtime' and 'Use this key to register your integration runtime'). Two keys are listed: 'Key1' and 'Key2', each associated with a unique identifier. The interface is identical to the one shown in the previous screenshot, with the addition of the runtime configuration options.

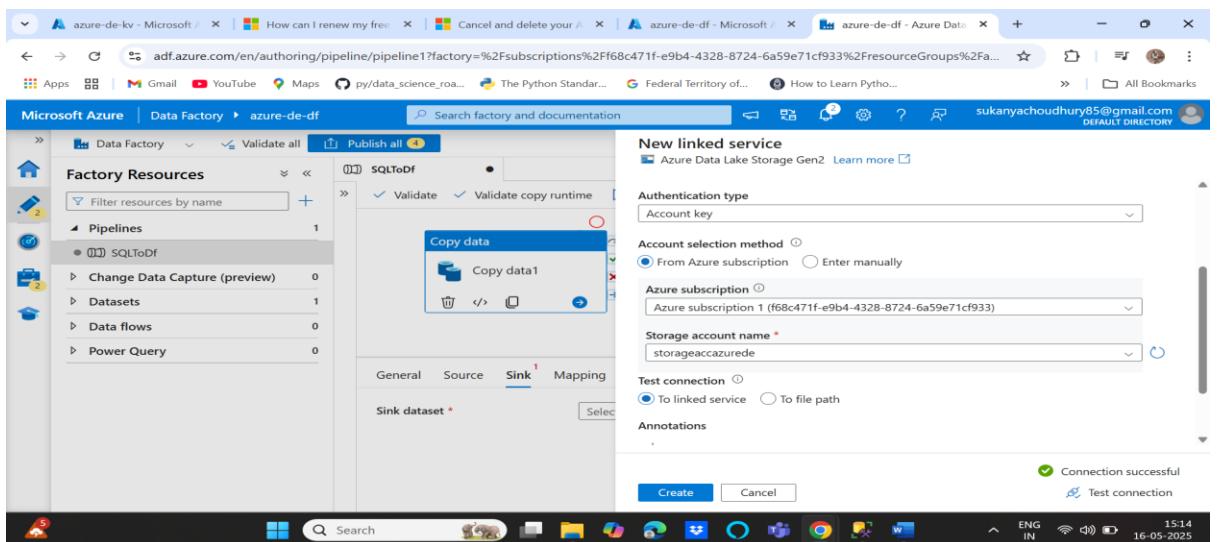


- 5) Use the above created Integrated Runtime for the linked service setup and give the database details and in password → Azure key vault → create a AKV linked service and then use the linked service and the secret key password created before so that the final sql serve linked service looks like below SS2 → create the linked service

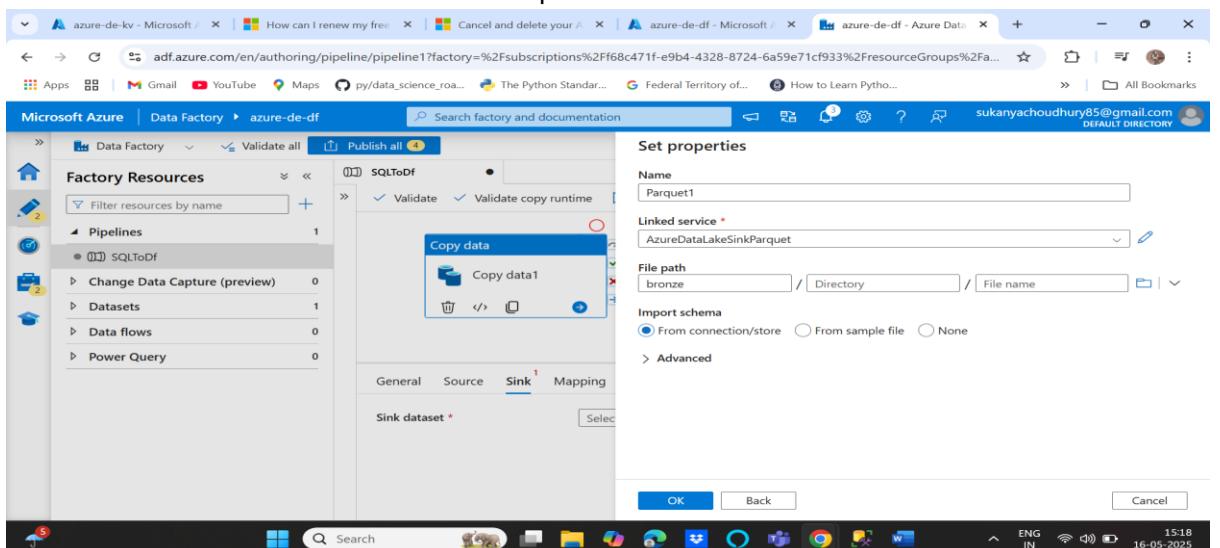


- 6) For sink again set up the data source using azure data lake gen 2 and file format parquet and also create a new linked service for the data source

The integration runtime will be the default “AutoResolveIntegrationRuntime” this time as this is an azure service i.e. azure data lake that we need to store data into.



- 7) Then after selecting the above linked service, we need to give the destination folder in the data lake so select bronze container in the file path



8) Then Debug to run the pipeline once to check any error.

The screenshot shows the Azure Data Factory pipeline editor. A pipeline named "SQLToDF" is open. Inside, there is a single activity named "Copy data1". The status of this activity is "Succeeded". Below the pipeline, a table displays the execution details of the activity:

| Activity name | Activity st... | Activit... | Run start | Duration | Integration runtime |
|---------------|----------------|------------|-----------------------|----------|--------------------------|
| Copy data1 | Succeeded | Copy data | 5/16/2025, 3:19:29 PM | 21s | SelfHostedIntegrationRur |

Confirm after checking storageaccount→containers→bronze whether it's copied:

The screenshot shows the Azure Storage Explorer interface. A container named "bronze" is selected. Inside, there is a single blob named "SalesLT.Address.p...". The blob was last modified on 5/16/2025, 3:19:48 PM, has a size of 35.08 KB, and is in the "Available" state.

9) Above was example to copy a single table, **how to dynamically copy all tables:**

Create another pipeline with a lookup, copy and foreach Activity

Configure lookup activity by setting up the source dataset or using the one created in the last step “ SQLServerOnPrem”. Then open and edit it and remove the selected table as we need to use all the tables. Then in the lookup activity use query as below and uncheck “First row only” then debug to see all tables in output

The screenshot shows the Azure Data Factory pipeline editor. A pipeline named "SQLToDfAllTables" is selected. In the "General" tab of the "Settings" section, the "Source dataset" is set to "SqlServerOnPrem". Under the "Query" section, the following T-SQL query is defined:

```

Select s.name as SchemaName,
       t.name as TableName
  from sys.tables t
  JOIN
       sys.schemas s
  ON t.schema_id=s.schema_id
 WHERE s.name='SalesLT'
    
```

10) On success of this lookup activity, add a foreach activity and inside the foreach activity add a copy activity with below configurations

Source dataset → SQLServerOnPrem(created before) and in query → add dynamic content with below query which runs a sql query by concatenating schema and table names

The screenshot shows the Microsoft Azure Data Factory Pipeline expression builder. A dynamic SQL query is being constructed in the 'Query' field:

```
@{concat('SELECT * from ',item().SchemaName,'.',item().TableName)}
```

The pipeline expression builder interface includes tabs for 'ForEach iterator', 'Activity outputs', 'Parameters', and 'System variables'. The 'ForEach1' iterator is selected.

For sink, use Parquetsink Ds created before and edit it to create parameters to give the destination folders as per schema and table name and then pass values as dynamic content to them in copy data → sink tab

The screenshot shows the 'Sink' tab for a Parquet dataset named 'Parquet1'. The 'Parameters' section contains two parameters:

| Name | Type | Default value |
|------------|--------|---------------|
| schemaname | String | Value |
| tablename | String | Value |

The screenshot shows the 'Copy data' dialog and the 'Sink' tab configuration for the 'Parquet1' dataset. In the 'Sink' tab, 'Dataset properties' are defined with two parameters:

| Name | Value | Type |
|------------|--------------------|--------|
| schemaname | @item().SchemaName | string |
| tablename | @item().TableName | string |

Now, pass these parameters into the sink directory as add dynamic content as below

The screenshot shows the Microsoft Azure Data Factory interface. On the left, there's a navigation pane with icons for Home, New, Pipelines, Schedules, Triggers, and Metrics. The main area shows a pipeline named 'Parquet1'. In the 'Parameters' section of the pipeline editor, there is a 'File path' field containing the expression: `@{concat(dataset().schemaname, '/', dataset().tablename)}`. Below this, there are tabs for 'Parameters' and 'Functions', and a search bar. At the bottom right of the pipeline editor are 'OK' and 'Cancel' buttons.

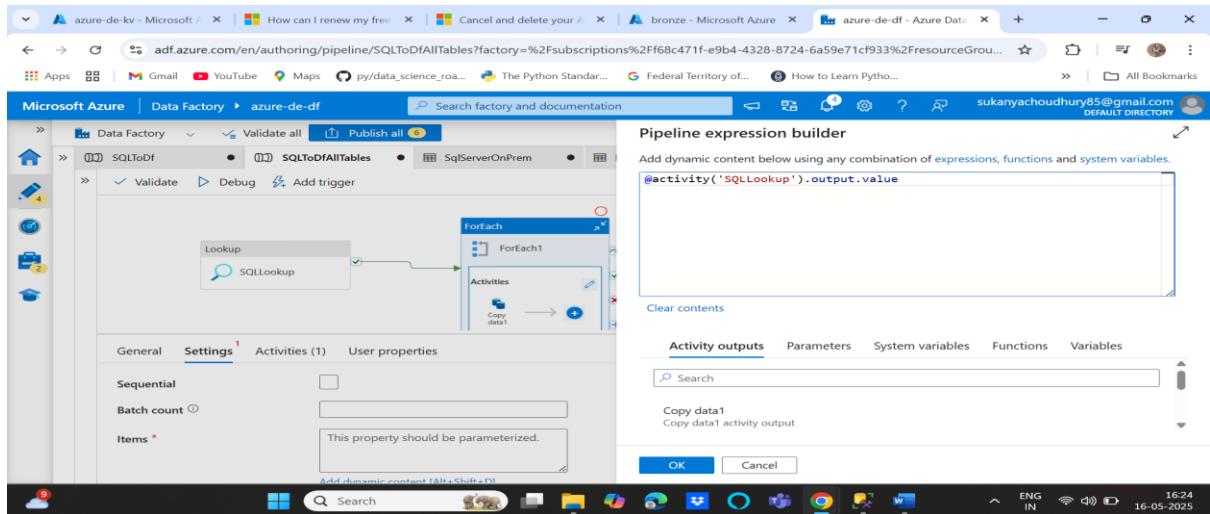
And the filename to save as parquet file as below

This screenshot is similar to the previous one, but the 'File path' field in the 'Parameters' section now contains the expression: `@{concat(dataset().tablename, '.parquet')}`. The rest of the interface is identical, showing the pipeline editor with the 'Parquet1' dataset selected.

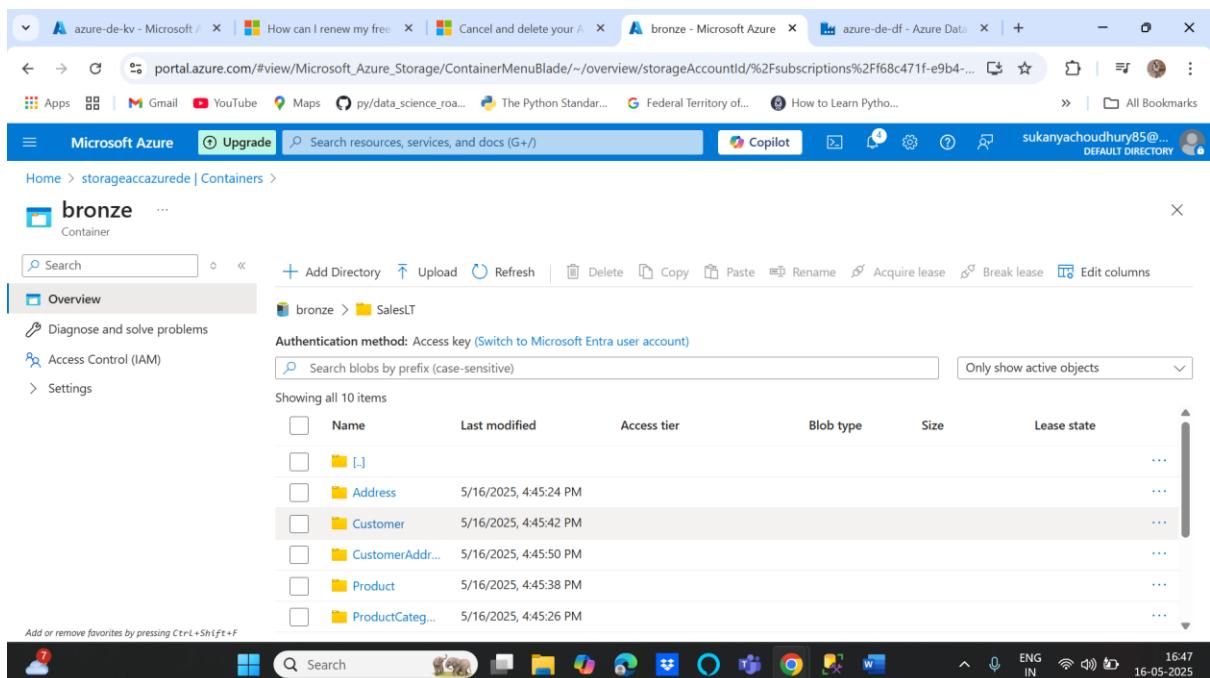
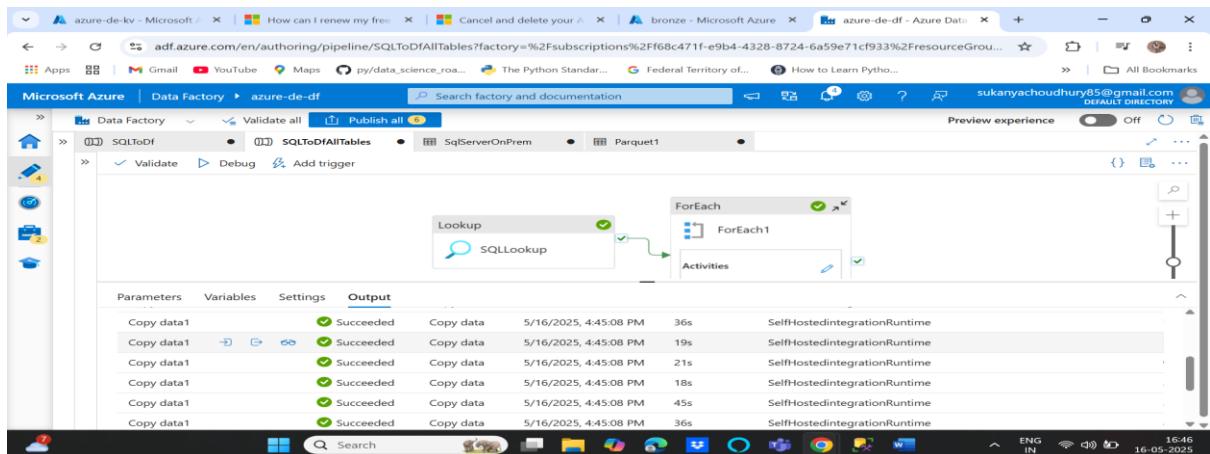
Finally sink data set looks like below

The screenshot shows the Microsoft Azure Data Factory pipeline editor with the 'Parquet1' dataset selected. In the 'Parameters' section, the 'File path' field now contains the full expression: `bronze / @{concat(dataset().schemaname, '/', dataset().tablename)} / @{concat(dataset().tablename, '.parquet')}`. The 'Compression type' is set to 'snappy'. At the top right, there are buttons for 'Preview experience' (set to 'Off'), 'Edit', 'New', 'Learn more', 'Browse', and 'Preview data'. The status bar at the bottom right shows the date and time as 16-05-2025 16:20.

Now setup the foreach activity to get the lookup activity output value:



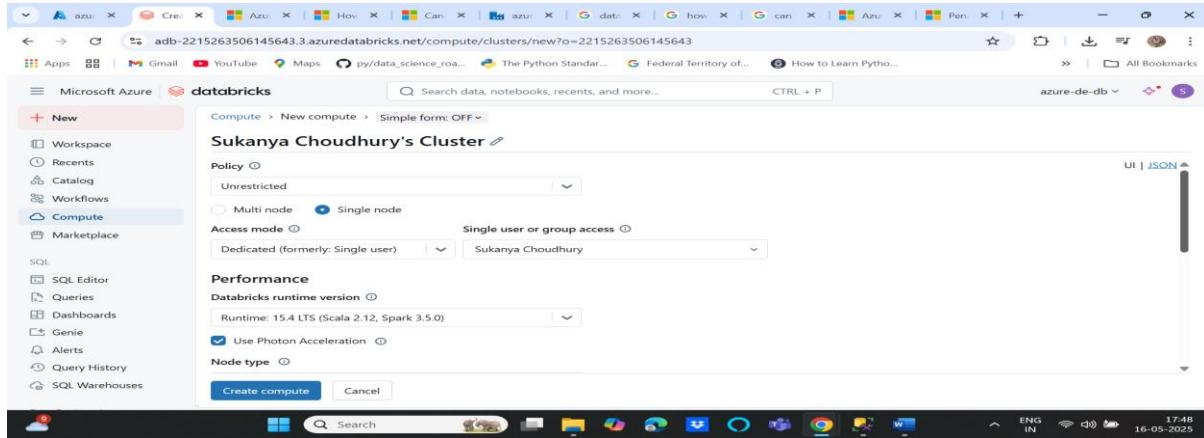
11) Debug lookup activity which runs the whole pipeline and extracts table from sql server into data lake bronze container and then publish all



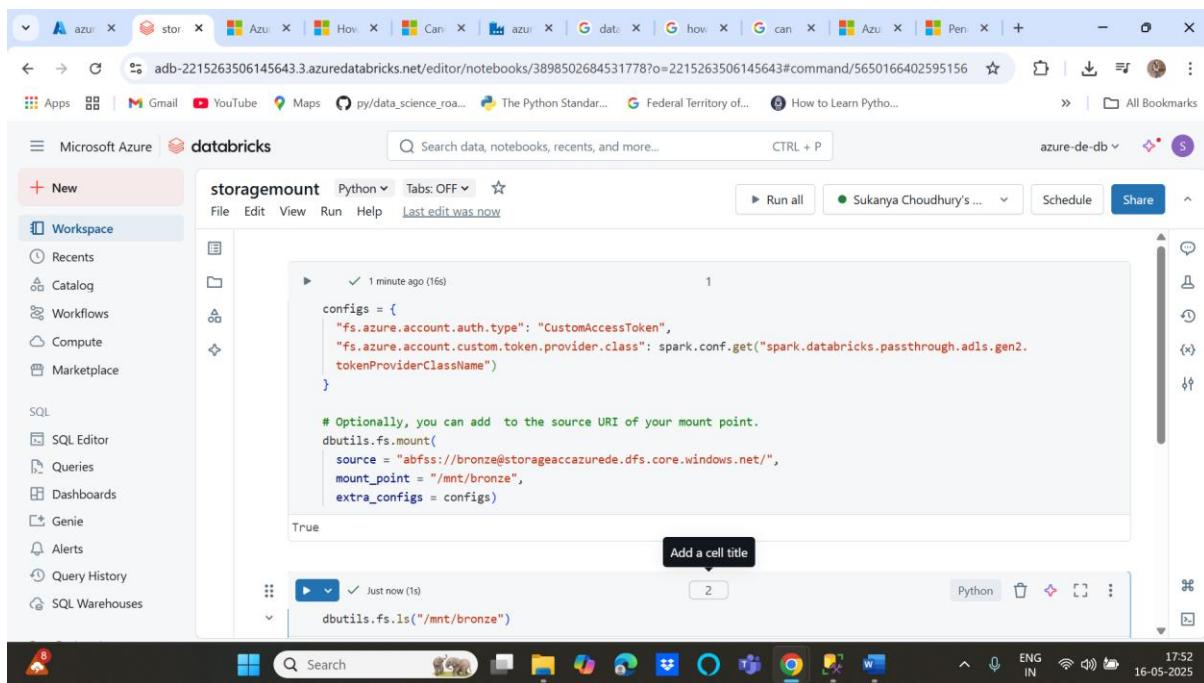
Step 3: Data Transformations using Data Bricks

1) Create a cluster

Launch Databricks Workspace → Compute → Create Compute



2) Mount Data Lake in Databricks: Configure Databricks to access ADLS using below notebook



```

# Optionally, you can add to the silver source URI of your mount point.
dbutils.fs.mount(
    source = "abfss://silver@storageaccazurede.dfs.core.windows.net/",
    mount_point = "/mnt/silver",
    extra_configs = configs)

# Optionally, you can add to the source URI of your mount point.
dbutils.fs.mount(
    source = "abfss://gold@storageaccazurede.dfs.core.windows.net/",
    mount_point = "/mnt/gold",
    extra_configs = configs)

```

True

3) Transform Data: Use Databricks notebooks to clean and aggregate the data, moving it from `bronze` to `silver`

Bronze to silver : list bronze tables

```

dbutils.fs.ls('mnt/bronze/SalesLT')

```

[FileInfo(path='dbfs:/mnt/bronze/SalesLT/Address/', name='Address', size=0, modificationTime=1747394124000),
 FileInfo(path='dbfs:/mnt/bronze/SalesLT/Customer/', name='Customer', size=0, modificationTime=1747394124000),
 FileInfo(path='dbfs:/mnt/bronze/SalesLT/CustomerAddress/', name='CustomerAddress', size=0, modificationTime=174739415000)
 ...],
 FileInfo(path='dbfs:/mnt/bronze/SalesLT/Product/', name='Product', size=0, modificationTime=1747394138000),
 FileInfo(path='dbfs:/mnt/bronze/SalesLT/ProductCategory/', name='ProductCategory', size=0, modificationTime=174739412600
 ...),
 FileInfo(path='dbfs:/mnt/bronze/SalesLT/ProductDescription/', name='ProductDescription', size=0, modificationTime=1747394142000),
 FileInfo(path='dbfs:/mnt/bronze/SalesLT/ProductModel/', name='ProductModel', size=0, modificationTime=1747394123000),
 FileInfo(path='dbfs:/mnt/bronze/SalesLT/ProductModelProductDescription/', name='ProductModelProductDescription', size=0, m
 odificationTime=1747394142000),
 FileInfo(path='dbfs:/mnt/bronze/SalesLT/SalesOrderDetail/', name='SalesOrderDetail', size=0, modificationTime=174739412300
 ...)]

Load the address table from bronze container

```

df = spark.read.format('parquet').load('/mnt/bronze/SalesLT/Address/Address.parquet')

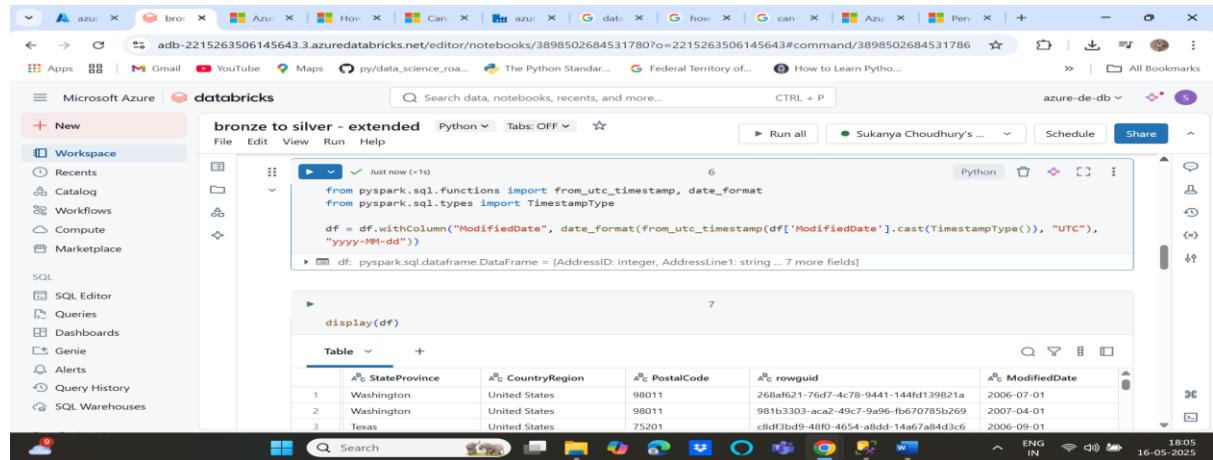
```

(1) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [AddressID: integer, AddressLine1: string ... 7 more fields]

| # | AddressID | AddressLine1 | AddressLine2 | City | StateProvince | CountryRegion |
|---|-----------|---------------------|--------------|----------|---------------|---------------|
| 1 | 9 | 8713 Yosemite Ct. | null | Bothell | Washington | United States |
| 2 | 11 | 1318 Lasalle Street | null | Bothell | Washington | United States |
| 3 | 25 | 9178 Jumping St. | null | Dallas | Texas | United States |
| 4 | 28 | 9228 Via Del Sol | null | Phoenix | Arizona | United States |
| 5 | 32 | 26910 Indela Road | null | Montreal | Quebec | Canada |

Modified date is transformed into UTC date format for Address table



```

bronze to silver - extended Python File Edit View Run Help
Just now (<1s) 6
from pyspark.sql.functions import from_utc_timestamp, date_format
from pyspark.sql.types import TimestampType

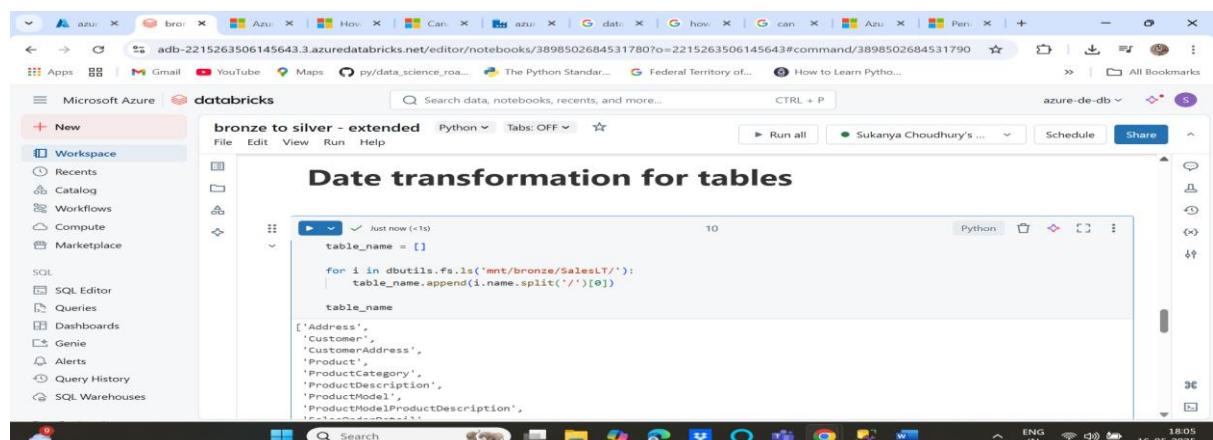
df = df.withColumn("ModifiedDate", date_format(from_utc_timestamp(df["ModifiedDate"].cast(TimestampType()), "UTC"), "yyyy-MM-dd"))

df: pyspark.sql.dataframe.DataFrame = [AddressID: integer, AddressLine1: string ... 7 more fields]

```

The screenshot shows a Databricks notebook titled "bronze to silver - extended". The code cell at the top contains Python code that imports `from_utc_timestamp` and `date_format` from `pyspark.sql.functions`, and `TimestampType` from `pyspark.sql.types`. It then uses the `withColumn` method on a DataFrame `df` to change the `ModifiedDate` column from a timestamp to a UTC date string in the `yyyy-MM-dd` format. Below this, a preview of the DataFrame `df` is shown, displaying three rows of data with columns: StateProvince, CountryRegion, PostalCode, rowguid, and ModifiedDate.

Transformation for all tables: Get the table names in a list and then iterate through each table. Read the data from bronze container by Getting the path by appending tablename to the mount path and the parquet file name and then carry on the data transformation by searching for date columns and then save all tables as delta mode into the silver container



```

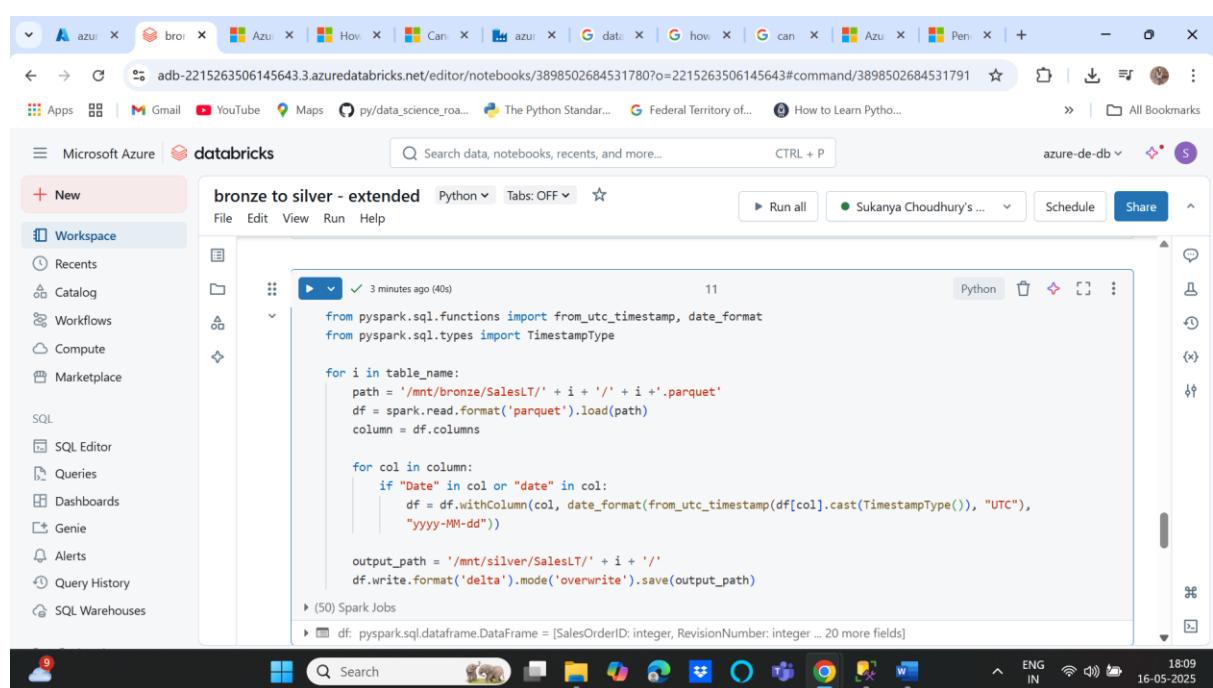
bronze to silver - extended Python File Edit View Run Help
Date transformation for tables
Just now (<1s) 10
table_name = []

for i in dbutils.fs.ls('/mnt/bronze/SalesLT/'):
    table_name.append(i.name.split('/')[0])

table_name

```

This screenshot shows a Databricks notebook with a section titled "Date transformation for tables". The code cell lists all table names in the "SalesLT" folder under the "bronze" mount point. The output of the code is a list of table names: 'Address', 'Customer', 'CustomerAddress', 'Product', 'ProductCategory', 'ProductDescription', 'ProductModel', 'ProductModelProductDescription', and 'SalesOrderDetail'.



```

bronze to silver - extended Python File Edit View Run Help
Just now (40s) 11
from pyspark.sql.functions import from_utc_timestamp, date_format
from pyspark.sql.types import TimestampType

for i in table_name:
    path = '/mnt/bronze/SalesLT/' + i + '/' + i + '.parquet'
    df = spark.read.format('parquet').load(path)
    column = df.columns

    for col in column:
        if "Date" in col or "date" in col:
            df = df.withColumn(col, date_format(from_utc_timestamp(df[col].cast(TimestampType()), "UTC"), "yyyy-MM-dd"))

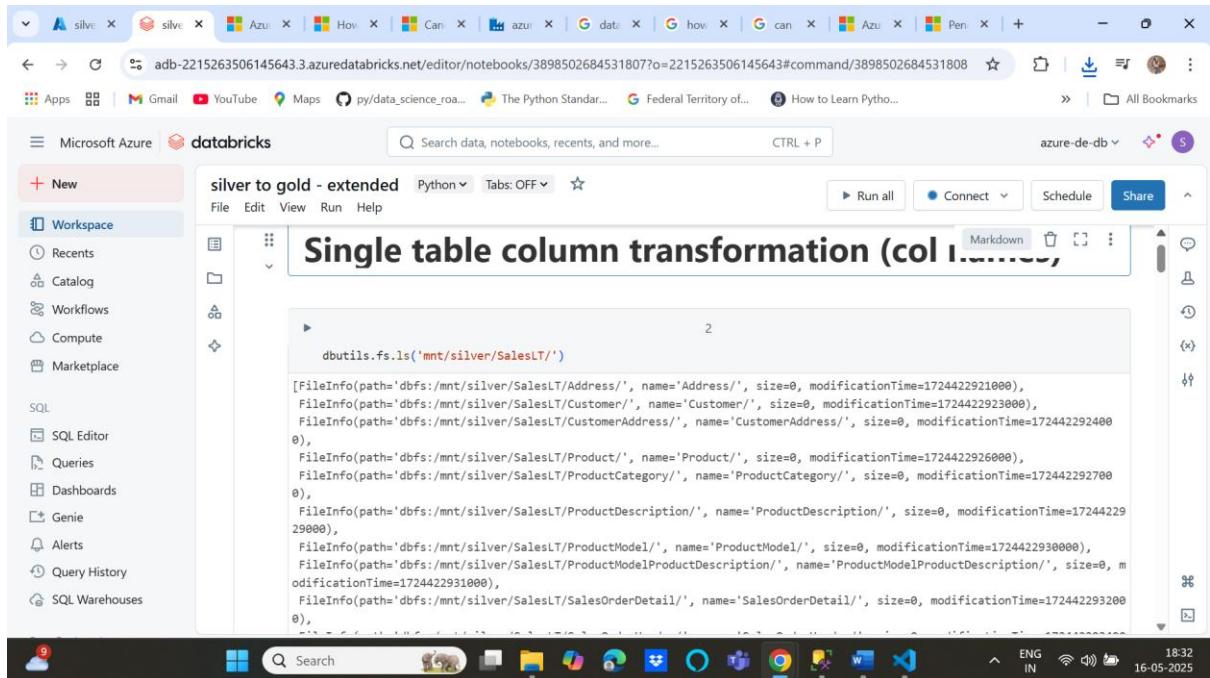
    output_path = '/mnt/silver/SalesLT/' + i + '/'
    df.write.format('delta').mode('overwrite').save(output_path)

(50) Spark Jobs
df: pyspark.sql.dataframe.DataFrame = [SalesOrderID: integer, RevisionNumber: integer ... 20 more fields]

```

This screenshot shows a Databricks notebook continuing the transformation process. The code iterates through the list of table names obtained earlier. For each table, it reads the parquet file, identifies date columns, and applies the same timestamp transformation logic as in the first screenshot. Finally, it saves each transformed DataFrame as a Delta table in the "silver" mount point with the same schema as the original table.

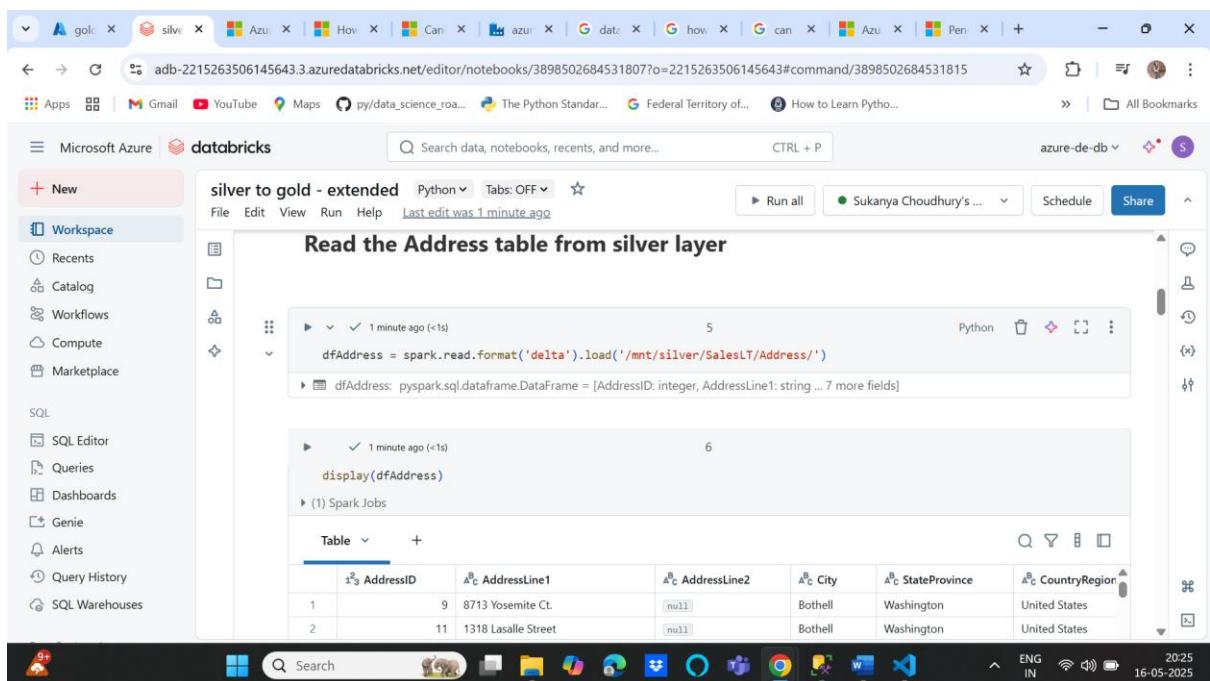
Silver to gold transformations



The screenshot shows a Microsoft Azure Databricks notebook titled "silver to gold - extended" in Python mode. The notebook contains the following code:

```
dbutils.fs.ls('mnt/silver/SalesLT')  
[FileInfo(path='dbfs:/mnt/silver/SalesLT/Address/', name='Address', size=0, modificationTime=1724422921000),  
 FileInfo(path='dbfs:/mnt/silver/SalesLT/Customer/', name='Customer', size=0, modificationTime=1724422923000),  
 FileInfo(path='dbfs:/mnt/silver/SalesLT/CustomerAddress/', name='CustomerAddress', size=0, modificationTime=172442292400  
 0),  
 FileInfo(path='dbfs:/mnt/silver/SalesLT/Product/', name='Product', size=0, modificationTime=1724422926000),  
 FileInfo(path='dbfs:/mnt/silver/SalesLT/ProductCategory/', name='ProductCategory', size=0, modificationTime=172442292700  
 0),  
 FileInfo(path='dbfs:/mnt/silver/SalesLT/ProductDescription/', name='ProductDescription', size=0, modificationTime=17244229  
 29000),  
 FileInfo(path='dbfs:/mnt/silver/SalesLT/ProductModel/', name='ProductModel', size=0, modificationTime=1724422930000),  
 FileInfo(path='dbfs:/mnt/silver/SalesLT/ProductModelProductDescription/', name='ProductModelProductDescription', size=0, m  
 odificationTime=1724422931000),  
 FileInfo(path='dbfs:/mnt/silver/SalesOrderDetail/', name='SalesOrderDetail', size=0, modificationTime=172442293200  
 0),  
 FileInfo(path='dbfs:/mnt/silver/SalesOrderDetailLineItem/', name='SalesOrderDetailLineItem', size=0, modificationTime=1724422933000)
```

Silver to Gold

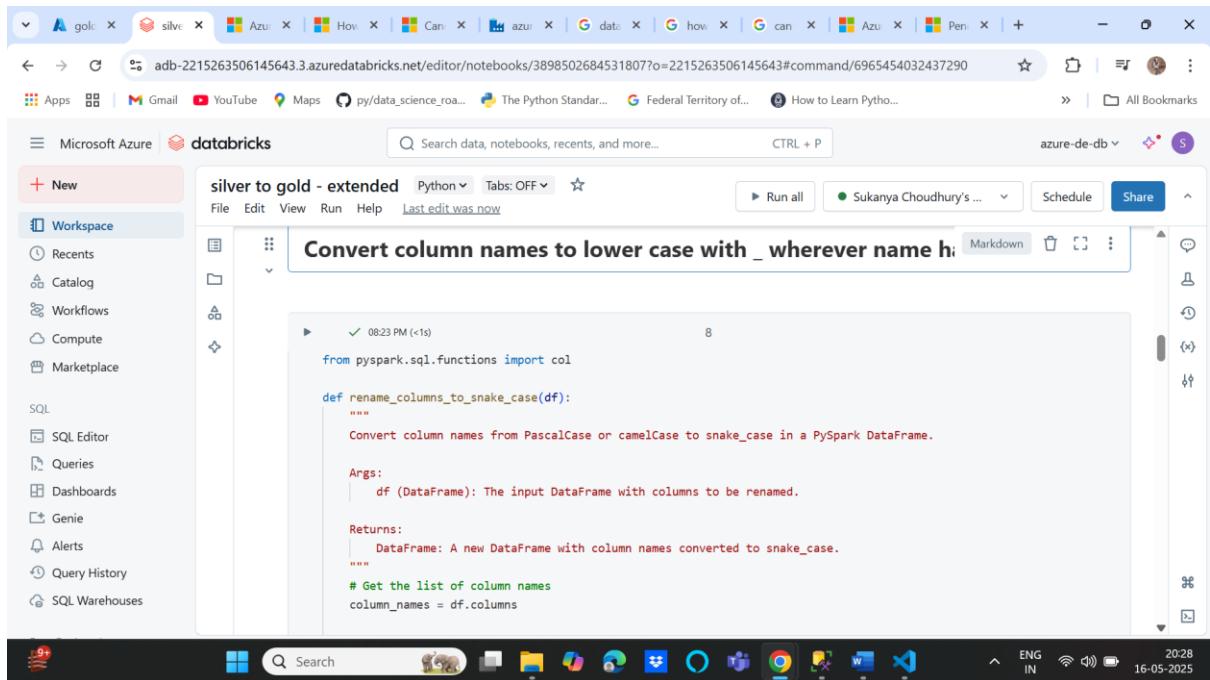


The screenshot shows a Microsoft Azure Databricks notebook titled "silver to gold - extended" in Python mode. The notebook contains the following code and displays a table of address data:

```
dfAddress = spark.read.format('delta').load('/mnt/silver/SalesLT/Address')  
dfAddress: pyspark.sql.dataframe.DataFrame = [AddressID: integer, AddressLine1: string ... 7 more fields]
```

Below the code, the notebook displays the "dfAddress" DataFrame:

| AddressID | AddressLine1 | AddressLine2 | City | StateProvince | CountryRegion |
|-----------|---------------------|--------------|---------|---------------|---------------|
| 1 | 8713 Yosemite Ct. | null | Bothell | Washington | United States |
| 2 | 1318 Lasalle Street | null | Bothell | Washington | United States |

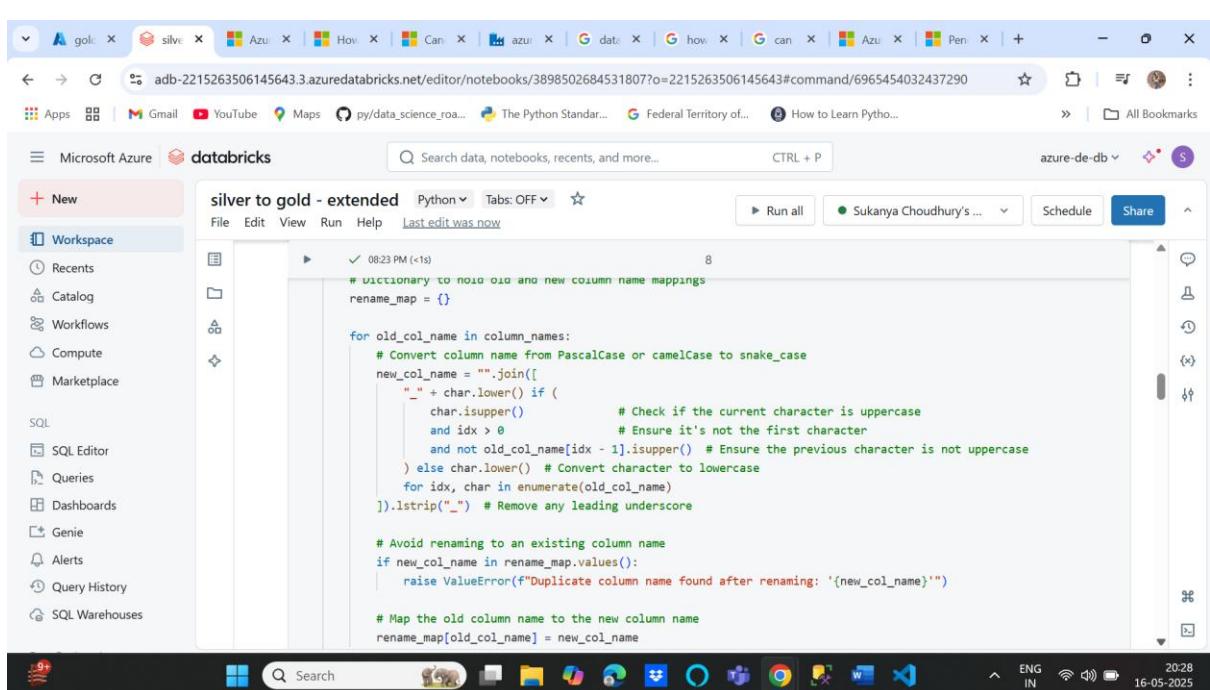


```
silver to gold - extended Python Tabs: OFF
File Edit View Run Help Last edit was now
Convert column names to lower case with _ wherever name has uppercase letters
from pyspark.sql.functions import col

def rename_columns_to_snake_case(df):
    """
    Convert column names from PascalCase or camelCase to snake_case in a PySpark DataFrame.

    Args:
        df (DataFrame): The input DataFrame with columns to be renamed.

    Returns:
        DataFrame: A new DataFrame with column names converted to snake_case.
    """
    # Get the list of column names
    column_names = df.columns
```



```
# dictionary to hold old and new column name mappings
rename_map = {}

for old_col_name in column_names:
    # Convert column name from PascalCase or camelCase to snake_case
    new_col_name = "".join([
        "_" + char.lower() if (
            char.isupper() and idx > 0 and not old_col_name[idx - 1].isupper()) # Check if the current character is uppercase and it's not the first character and the previous character is not uppercase
        else char.lower() # Convert character to lowercase
    ]).lstrip("_") # Remove any leading underscore

    # Avoid renaming to an existing column name
    if new_col_name in rename_map.values():
        raise ValueError(f"Duplicate column name found after renaming: '{new_col_name}'")

    # Map the old column name to the new column name
    rename_map[old_col_name] = new_col_name
```

```
# Rename columns using the mapping
for old_col_name, new_col_name in rename_map.items():
    df = df.withColumnRenamed(old_col_name, new_col_name)

return df

# Example usage
# df = rename_columns_to_snake_case(df)
```

File Edit View Run Help Last edit was now

Run all

Sukanya Choudhury's ... Schedule Share

Search data, notebooks, recents, and more... CTRL + P

08:23 PM (<1s) 8

4 minutes ago (<1s) 9

df: pyspark.sql.dataframe.DataFrame = [address_id: integer, address_line1: string ... 7 more fields]

File Edit View Run Help Last edit was now

Run all

Sukanya Choudhury's ... Schedule Share

Search data, notebooks, recents, and more... CTRL + P

4 minutes ago (<1s) 10

df: pyspark.sql.dataframe.DataFrame = [address_id: integer, address_line1: string ... 7 more fields]

4 minutes ago (<1s) 11

display(df)

(1) Spark Jobs

Table +

| address_id | address_line1 | address_line2 | city | state_province | country_region |
|------------|------------------------|---------------|---------|----------------|----------------|
| 1 | 9 8713 Yosemite Ct. | null | Bothell | Washington | United States |
| 2 | 11 1318 Lasalle Street | null | Bothell | Washington | United States |
| 3 | 25 9178 Jumping St. | null | Dallas | Texas | United States |
| 4 | 28 9228 Via Del Sol | null | Phoenix | Arizona | United States |

All table columns transformation (col names)

```
# To show the basic format of ls
table_name = []

for i in dbutils.fs.ls('mnt/silver/SalesLT'):
    table_name.append(i)

table_name
```

[FileInfo(path='dbfs:/mnt/silver/SalesLT/Address', name='Address', size=0, modificationTime=1747398983000),
 FileInfo(path='dbfs:/mnt/silver/SalesLT/Customer', name='Customer', size=0, modificationTime=1747398999000),
 FileInfo(path='dbfs:/mnt/silver/SalesLT/CustomerAddress', name='CustomerAddress', size=0, modificationTime=1747399002000),
 FileInfo(path='dbfs:/mnt/silver/SalesLT/Product', name='Product', size=0, modificationTime=1747399005000),
 FileInfo(path='dbfs:/mnt/silver/SalesLT/ProductCategory', name='ProductCategory', size=0, modificationTime=1747399007000)]

```
table_name = []

for i in dbutils.fs.ls('mnt/silver/SalesLT'):
    table_name.append(i.name.split('/')[0])
```

```
table_name
['Address',
 'Customer',
 'CustomerAddress',
 'Product',
 'ProductCategory',
 'ProductDescription',
 'ProductModel',
 'ProductModelProductDescription',
 'SalesOrderDetail']
```

```
for name in table_name:
    path = '/mnt/silver/SalesLT/' + name
    print(path)
    df_tables = spark.read.format('delta').load(path)

    df = rename_columns_to_snake_case(df_tables)

    output_path = '/mnt/gold/SalesLT/' + name + '/'
    df.write.format('delta').mode('overwrite').save(output_path)
```

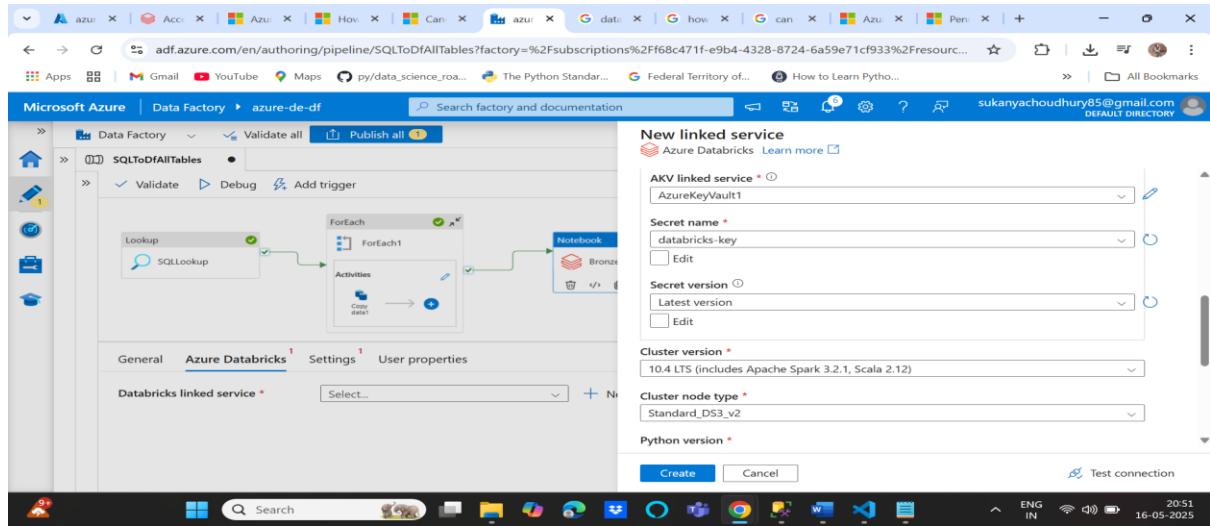
(40) Spark Jobs

- df: pyspark.sql.dataframe.DataFrame = [sales_order_id: integer, revision_number: integer ... 20 more fields]
- df_tables: pyspark.sql.dataframe.DataFrame = [SalesOrderID: integer, RevisionNumber: integer ... 20 more fields]

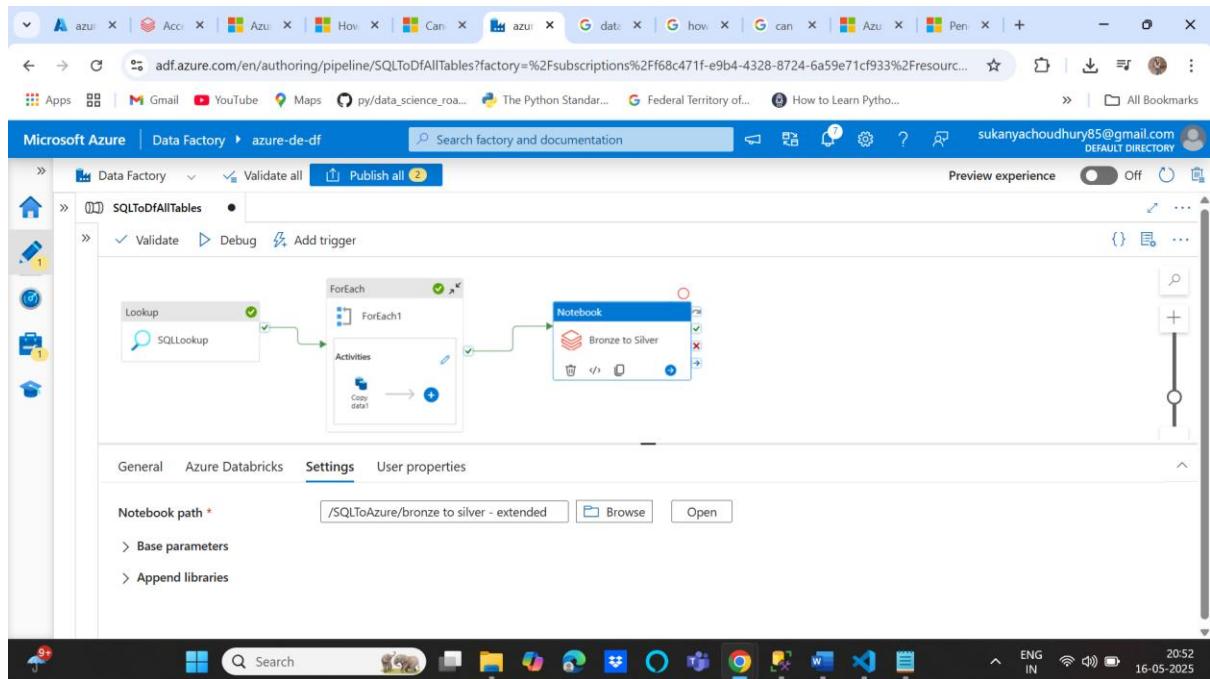
/mnt/silver/SalesLT/Address
/mnt/silver/SalesLT/Customer
/mnt/silver/SalesLT/CustomerAddress
/mnt/silver/SalesLT/Product
/mnt/silver/SalesLT/ProductCategory
/mnt/silver/SalesLT/ProductDescription

- 4) So now we want that these databricks notebooks run automatically from the pipeline so add notebook activity on success to the foreach activity of the pipeline

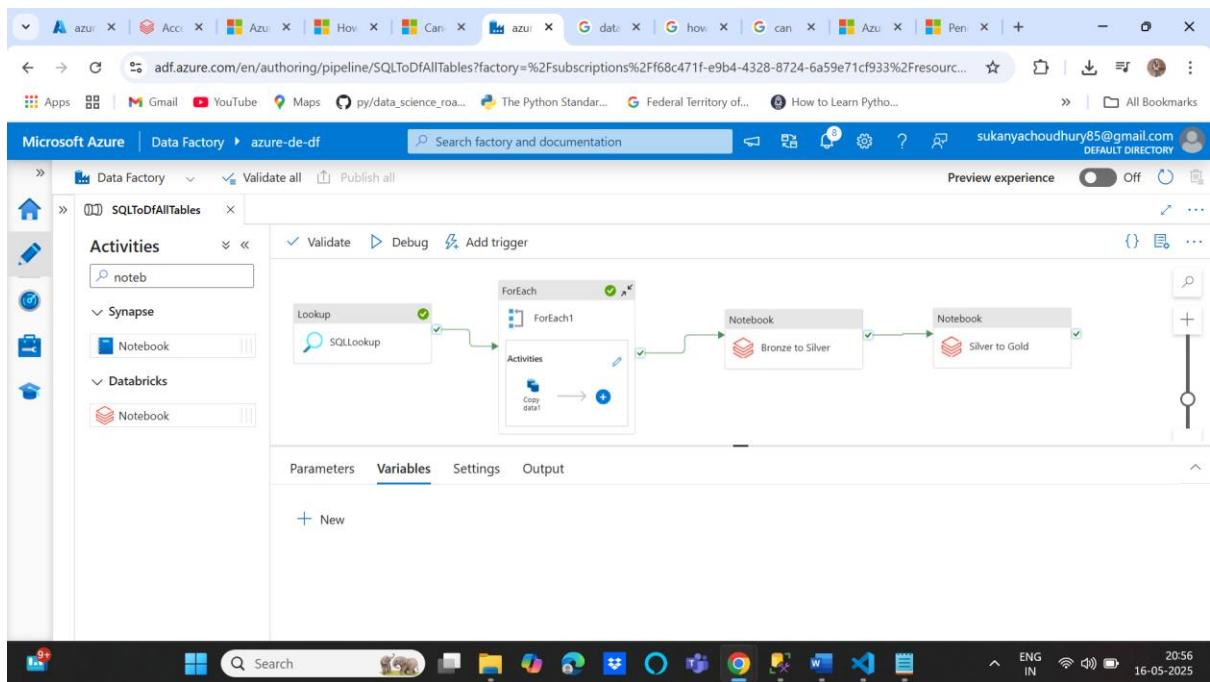
Create a databricks linked service for the notebook activity using access token from databricks settings → developer → access tokens → Generate access token and then add it on the linked service configuration or store it in key vault → secret → generate/import and then use it



Then Give the notebook path



Similarly create another notebook activity for silver to gold and then publish and then add trigger to run the pipeline, monitor on pipeline runs



Step 4: Data Load Into Synapse Analytics

1) Synapse studio—data→add new SQL database

Instead of writing the full query to explore a table, go to linked storage lake → navigate to gold folder and then for each table folder rt click do show 100 rows for delta format → it opens an sql script which shows data for that table

The screenshot shows the Microsoft Azure Synapse Analytics studio. A workspace named "gold" is selected. A SQL script titled "SQL script 1" is open, showing the following auto-generated code:

```

1 -- This is auto-generated code
2 SELECT
3     TOP 100 *
4 FROM
5     OPENROWSET(
6         BULK 'https://storageaccazurede.dfs.core.windows.net/gold/SalesL'
7         FORMAT = 'PARQUET'
8     ) AS [result]
9 
```

The "Properties" panel on the right shows the following details:

- Name: SQL script 1
- Type: .sql script
- Size: 206 bytes

Now we can use the same script to create view by adding a create view statement above as shown below, select appropriate db from use database dropdown

```

CREATE VIEW SalesLT_AddressView_AS
SELECT *
FROM
OPENROWSET(
    BULK 'https://storageaccazurede.dfs.core.windows.net/gold/SalesLT/Address/',
    FORMAT = 'DELTA'
) AS [result]

```

2) Create a sql procedure to load all tables together and create views and publish it

```

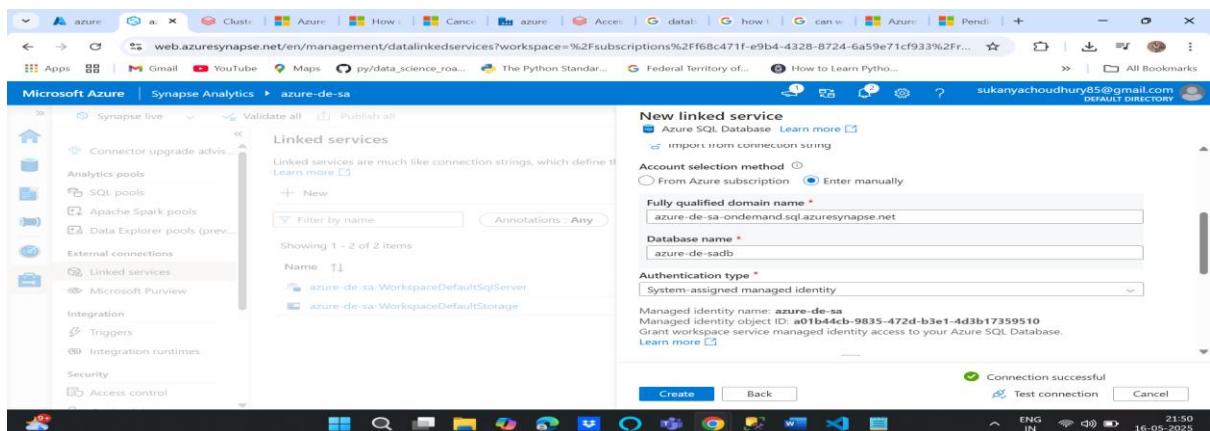
USE azure_de_sadb
GO
CREATE OR ALTER PROC CreateSQLServerlessView_gold @ViewName nvarchar(100)
AS
BEGIN
    DECLARE @statement VARCHAR(MAX)
    SET @statement = N'CREATE OR ALTER VIEW ' + @ViewName + ' AS
        SELECT *
        FROM
        OPENROWSET(
            BULK ''https://storageaccazurede.dfs.core.windows.net/gold/SalesLT/' + @ViewName + '/',
            FORMAT = ''DELTA''
        ) AS [result]'

    EXEC (@statement)
END
GO

```

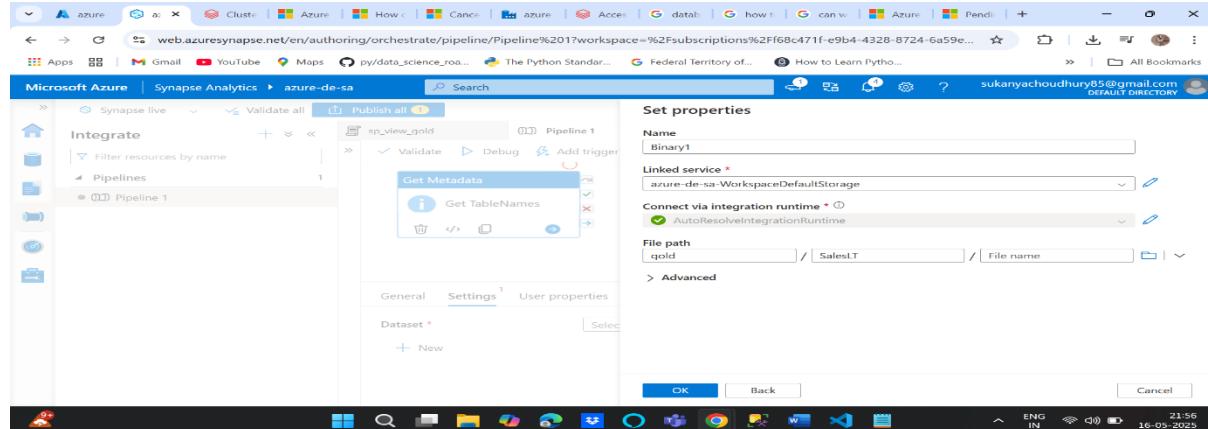
3) Create a pipeline inside synapsis analytics studio which uses this procedure

Go to manage→linkedServices-> +New→ Azure SQL Database→ give the serverless sql endpoint of sa as fully qualified domain name→ create and publish linked service



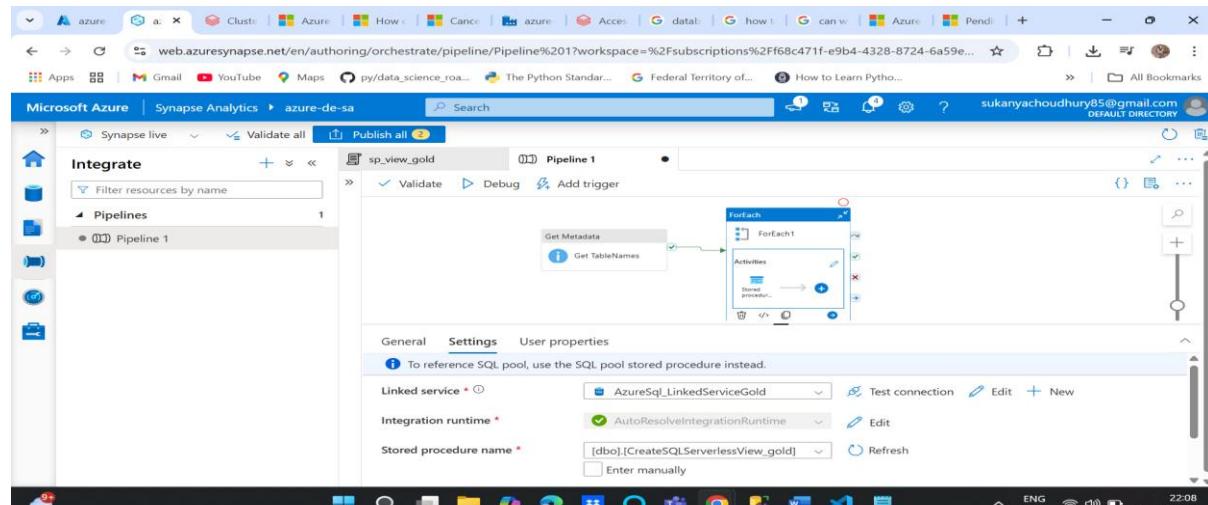
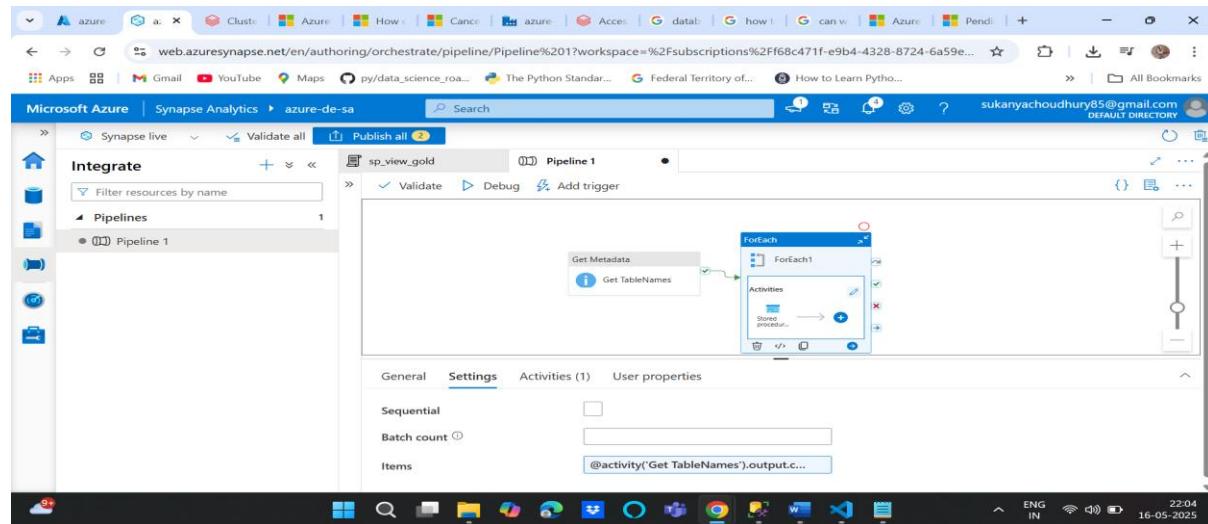
Now go to integrate → + → Pipeline → Add Get Metadata activity → settings—add dataset → azure data lake gen2 → binary format → select the default linked service and the gold directory

And then Getmetadata activity → field list → new → Child items



Now add foreach activity to the pipeline on success of the getmetadata activity

Settings → items → Add dynamic content → GetTable.output.childitems and then add stored procedure to the foreach activity → give the linked service and the procedure to it and parameters value pass ass @item().name i.e view name parameter = tablename



Stored procedure parameters

[Import](#) [New](#) [Delete](#)

| Name | Type * | Value |
|----------|--------|--------------|
| ViewName | String | @item().name |

Run the pipeline

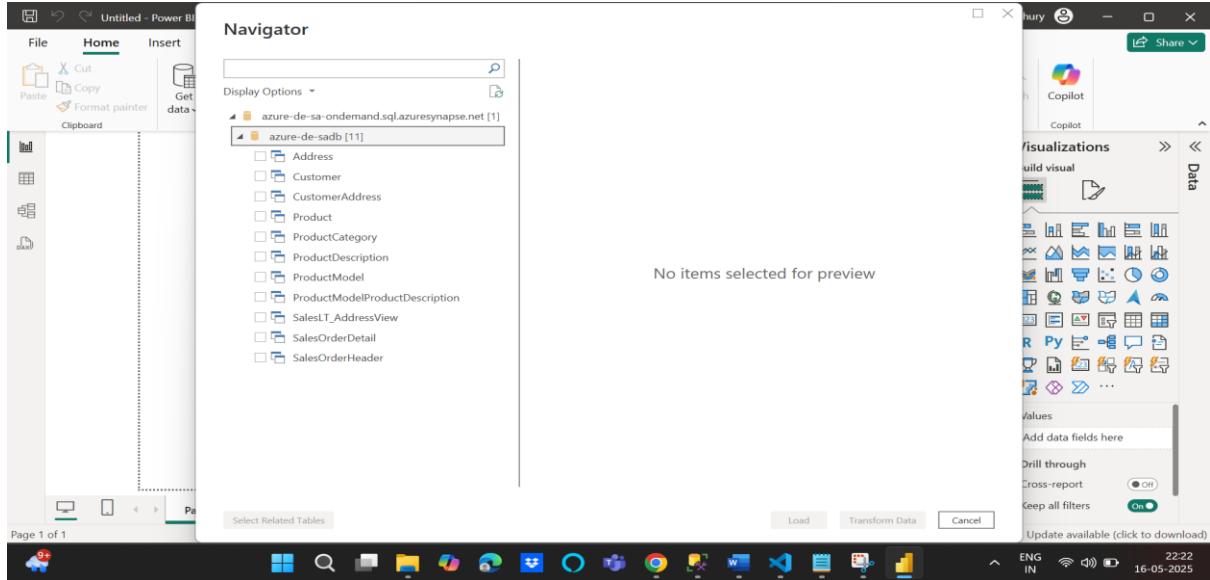
All views are created

4) Trigger pipeline on a schedule:

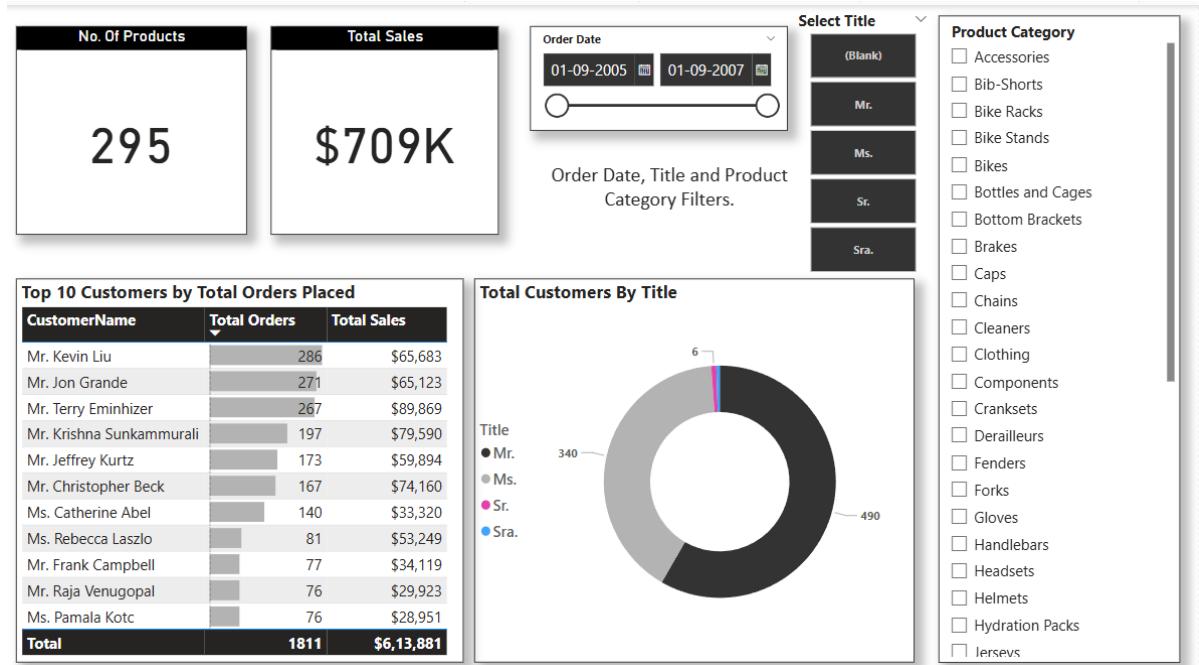
Launch Data Factory → Go to the pipeline → Add Trigger → New →

Step 5: Reporting using Power BI

- 1) Get data → Azure Synapse analytics SQL → give the synapse analytics serverless endpoint → username/password and connect → you can see all the views like below



Create a dashboard like below



Step 6: **Manage Access**: Set up role-based access control (RBAC) using Azure Entra ID (formerly Active Directory).

Go to the Azure resource group → Access control (IAM) → Role Assignments → shows all the users who have access. To have more people access, we should create security groups

The screenshot shows the Microsoft Azure portal interface. The top navigation bar includes links for Apps, Gmail, YouTube, Maps, Python, The Python Standard, Federal Territory of..., How to Learn Python, and All Bookmarks. The user's name, sukanya choudhury85@..., is visible in the top right corner. The main content area is titled 'azur... | Access control (IAM)' and shows the 'Access control (IAM)' tab selected. A summary table indicates 14 role assignments for the subscription, with 2 being 'Privileged'. Below this, a detailed list shows two users with 'Owner' roles: 'Sukanya Choudhury' (User, Owner, Subscription (Inherited), None) and another 'Sukanya Choudhury' (User, Owner, Subscription (Inherited), None). The left sidebar lists various Azure services like Activity log, Resource visualizer, Events, Settings, Cost Management, Monitoring, Automation, Help, and Tags.

To create a security group so that an entire department has access, go to Microsoft Entra ID → Add → Group → give the group name → select the owner (the admin) → select members(users/groups)

The screenshot shows the Microsoft Azure portal interface. The top navigation bar includes links for Apps, Gmail, YouTube, Maps, Python, The Python Standard, Federal Territory of..., How to Learn Python, and All Bookmarks. The user's name, sukanya choudhury85@..., is visible in the top right corner. The main content area is titled 'New Group' and shows the 'data-engineers' group being created. The 'Group name' field is set to 'data-engineers'. The 'Group description' field also contains 'data-engineers'. The 'Membership type' dropdown is set to 'Assigned'. Under the 'Owners' section, it says '1 owner selected'. The 'Members' section is currently empty. At the bottom, there is a 'Create' button. The left sidebar lists various Azure services like Activity log, Resource visualizer, Events, Settings, Cost Management, Monitoring, Automation, Help, and Tags.

Then go to the resource group → Access control(IAM) → Add role assignments → storage blob contributor → add the above security group

The screenshot shows the Azure portal interface for adding a role assignment. The top navigation bar includes links like Apps, Gmail, YouTube, Maps, Python Standard, Federal Territory of..., How to Learn Python, and All Bookmarks. The main title is "Select members - Microsoft Azure". The sub-page title is "Add role assignment". The "Members" tab is active, showing a "Selected role" of "Storage Blob Data Contributor". The "Assign access to" section has "User, group, or service principal" selected. The "Members" table shows one entry: "data-engineers" (Object ID: f25535b3-94d-423...). A "Description" field is present with an optional note. At the bottom are "Review + assign" and "Next" buttons, along with "Select" and "Close" buttons for the modal.

The security group “data-engineers” members have access to all the resources of the resource group “azure-de-rg” i.e. the data factory pipeline, data bricks or the synapsis analytics.

Now, under resource group → IAM → role assignments, the security group shows.

The screenshot shows the Azure portal interface for the "azure-de-rg" resource group. The left sidebar includes "Overview", "Activity log", and "Access control (IAM)". The "Access control (IAM)" section is selected. The main area displays a table of role assignments:

| | Name | Type | Role | Scope | Condition |
|--------------------------|---|-------|-------------------------------|--------------------------|-----------|
| <input type="checkbox"/> | Sukanya Choudhury 1a57bac3-b2d2-403... | User | Owner | Subscription (Inherited) | None |
| <input type="checkbox"/> | Sukanya Choudhury 1a57bac3-b2d2-403... | User | Owner | Subscription (Inherited) | None |
| <input type="checkbox"/> | data-engineers f25535b3-94d-423... | Group | Storage Blob Data Contributor | This resource | Add |

At the bottom, there are "Add or remove favorites by pressing Ctrl+Shift+F" and system status indicators like Rain warning, ENG IN, and 20-05-2025.

Step 7: End-to-End Testing

Trigger and Test Pipelines: Insert new records into the SQL database and verify that the entire pipeline runs successfully, updating the Power BI dashboard.