

Data Lake Exploration & Optimization

Project Documentation

Table of Contents

- Project Statement
- Project Overview
- Prerequisites
- Project Objectives
- Architecture Diagram
- ER Diagram
- Azure Resources Used for the Project
- Tools Used
- How It Works
- Execution Overview
- Practical Implementation on Azure Portal
- Analysis Results
- Implementation-Tasks Performed
- Successful Output Generated
- Strategies that can be used in Optimising the process
- Conclusion

1. Project Statement

The main goal of this project is to **explore and optimize data stored in an Azure Data Lake** using **PySparkSQL** within **Azure Databricks**. By leveraging Databricks, we can perform **seamless data ingestion, transformation, querying, and visualization at scale**.

This project demonstrates how organizations can build an efficient **data analytics pipeline** where raw data is ingested into a Data Lake, processed using PySparkSQL, optimized with Delta format for performance, and then visualized for decision-making.

2. Project Overview

This project walks through the **end-to-end process** of building a data analytics workflow on Azure. The major steps include:

- **Data Ingestion:** Loading CSV data (employees.csv) into **Azure Data Lake Storage (ADLS)**.
- **Storage Mounting:** Connecting ADLS with **Azure Databricks** using secure authentication (SAS token).
- **Exploration with PySparkSQL:** Running SQL-like queries on employee data to understand and analyze the dataset.
- **Data Optimization:** Converting raw data into **Delta Lake format** for improved performance, ACID transactions, and scalability.
- **Analytics & Visualization:** Running advanced queries (e.g., top salaries, department statistics, role-based salary averages) and creating charts using Databricks built-in visualization tools.

By the end of the project, we establish a **scalable and optimized pipeline** for handling structured data in a cloud-native way.

3. Prerequisites

To implement this project, the following requirements are needed:

- Azure Subscription with active credits (student subscription or paid).
- Azure Storage Account with at least one container to hold datasets.
- Azure Databricks Workspace with an active cluster for execution.

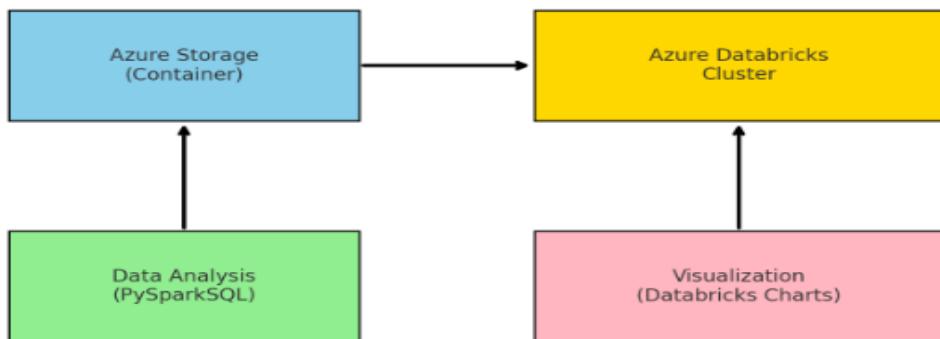
- **Dataset:** employees.csv containing employee records such as ID, name, department, role, salary, etc.
 - **SAS Token** (Secure Access Signature) to mount and connect Databricks with the storage account securely.
-

4. Project Objectives

The objectives of this project are as follows:

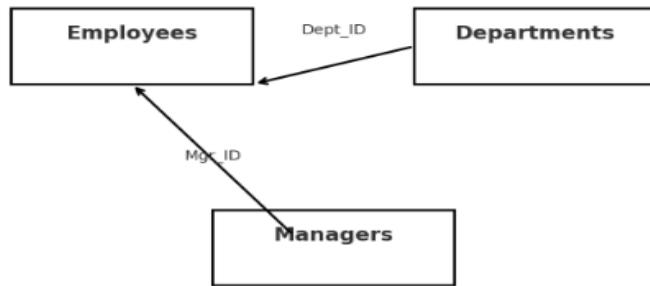
1. **Connect Azure Databricks with Azure Data Lake Storage** using a secure method.
 2. **Perform exploratory data analysis** on employee dataset using PySparkSQL queries.
 3. **Optimize data storage** by converting raw CSV into Delta Lake format (Bronze → Silver layers).
 4. **Generate valuable insights** such as:
 - Top employees with highest salaries.
 - Department-wise employee distribution.
 - Average salaries per role.
 - Trends in employee data for analytics.
 5. **Visualize results** using Databricks charts for better understanding and presentation.
-

5. Architecture Diagram



6. ER Diagram

ER Diagram - Employees Dataset



7. Azure Resources Used for the Project

- **Azure Storage Account:** sivagamistudentstorage
 - **Container Name:** mycontainer (stores employees.csv)
 - **Azure Databricks:** Workspace and Cluster (used for computation and query execution).
 - **Azure Data Lake:** Serves as the central repository for raw and optimized data.
-

8. Tools Used

- **Azure Portal** – For provisioning and managing cloud resources (Storage, Databricks).
 - **Azure Databricks** – For big data processing, analytics, and machine learning.
 - **PySparkSQL** – For running SQL queries on structured data inside Spark.
 - **Azure Storage Explorer (Optional)** – For managing datasets inside containers.
 - **Databricks UI** – For creating notebooks, running queries, and building visualizations.
-

9. How It Works

The workflow of this project can be summarized as follows:

1. **Upload dataset to Data Lake:** The employees.csv file is uploaded to the container (mycontainer) inside the storage account.
 2. **Generate SAS token:** A secure token is created to allow Databricks to access the storage account.
 3. **Mount storage to Databricks:** Using the SAS token, we mount the container into the Databricks file system (DBFS).
 4. **Read and Explore Data:** Load the dataset into a Spark DataFrame and run PySparkSQL queries for analysis.
 5. **Data Optimization:** Store processed data in **Delta Lake format** to support ACID transactions and better query performance.
 6. **Bronze → Silver Architecture:**
 - **Bronze Layer:** Raw ingested CSV data.
 - **Silver Layer:** Cleaned and optimized Delta tables.
 7. **Analytics & Visualization:** Queries are run to derive insights and Databricks' visualization features are used to generate charts.
-

10. Execution Overview

- **Data Source:** Employee dataset (employees.csv) stored in Data Lake.
 - **Storage:** Azure Blob Storage container (mycontainer).
 - **Processing Engine:** Azure Databricks cluster running PySparkSQL.
 - **Optimization:** Data converted to Delta Lake format for performance.
 - **Visualization:** Databricks notebook charts (bar chart, pie chart, etc.) created from SQL query outputs.
-

11. Practical Implementation on Azure Portal

The practical steps carried out on Azure Portal and Databricks are:

1. **Create Storage Account** → Add a container (mycontainer).
2. **Upload Dataset** → Place employees.csv in the container.
3. **Create Databricks Workspace** → Launch Databricks and set up a cluster.
4. **Generate SAS Token** → From the storage account for authentication.
5. **Mount Storage in Databricks** → Use the SAS token to mount the container.
6. **Load Data in Notebook** → Read employees.csv into Spark DataFrame.
7. **Run SQL Queries** → Examples:
 - Top 10 highest salaries.
 - Average salary by department.
 - Role-wise statistics.
8. **Convert to Delta Format** → Save optimized data in Silver layer.
9. **Visualize Data** → Create charts (bar, pie, histogram) within Databricks.

Step 1: Set Up Azure Databricks

- Created **Databricks workspace** and cluster in Azure Portal.
- Started cluster before working.

⌚ Screenshot: Databricks Workspace

This screenshot shows the Microsoft Azure Databricks workspace overview page for the service 'trainingprojectws'. The page includes a search bar, a navigation bar with links like 'Home', 'Copilot', and user information, and a main content area with tabs for 'Overview' and 'Essentials'. The 'Overview' tab is selected, displaying details such as Status (Active), Resource group (hexaware-training), Location (Central India), Subscription (Azure for Students), and a URL (https://adb-3427020724517888.azuredatabricks.net). It also shows a red 3D cube icon representing the cluster, a 'Launch Workspace' button, and an 'Upgrade to Premium' button.

⌚ Screenshot: New Notebook Creation

This screenshot shows the Databricks workspace interface. On the left, there's a sidebar with options like 'New', 'Workspace', 'Recents', 'Catalog', 'Jobs & Pipelines', 'Compute', 'Data Engineering', 'Job Runs', 'AI/ML', 'Playground', 'Experiments', 'Features', 'Models', and 'Serving'. The main area shows a 'Project' notebook with a code editor containing a single cell. The cell has a play button icon and the placeholder text 'Start typing or generate with AI (Ctrl + I)...'. The status bar at the bottom provides keyboard shortcuts for running cells.

⌚ Screenshot: Cluster Running

This screenshot shows the Databricks Compute section. The sidebar on the left is identical to the previous workspace screenshot. The main area displays a table of compute resources. One row is visible, showing a green dot icon for state, the name 'Sivakami G's Cluster 2025-08-23 11:27:11', runtime of 16.4, active memory of 16 GB, 4 cores, 1.5 DBU/h, and the creator 'Sivakami G'. There are also columns for 'Notebooks' and a three-dot menu icon.

Step 2: Create Azure Storage Account & Container

- Created storage account: sivagamistudentstorage.
- Inside it, created container: mycontainer.

⌚ Screenshot: Storage Account Created

Home > sivagamistudentstorage Storage account

Overview

Resource group (move) : myproject-rg

Location : centralindia

Primary/Secondary Location : Primary: Central India, Secondary: South India

Subscription (move) : Azure for Students

Subscription ID : 59718fb6-5385-4894-844f-4ef610d2c5c1

Disk state : Primary: Available, Secondary: Available

Tags (edit) : Add tags

Properties Monitoring Capabilities (5) Recommendations (1) Tutorials Tools + SDKs

Data Lake Storage

Hierarchical namespace	Enabled
Default access tier	Hot
Blob anonymous access	Enabled
Blob soft delete	Enabled (7 days)
Container soft delete	Enabled (7 days)
Versionsing	Disabled
Change feed	Disabled
NFS v3	Disabled
SFTP	Disabled
Storage tasks assignments	None

Security

Require secure transfer for REST API operations	Enabled
Storage account key access	Enabled
Minimum TLS version	Version 1.2
Infrastructure encryption	Disabled

Networking

Public network access	Enabled
Public network access scope	Enable from all networks
Private endpoint connections	0
Network routing	Microsoft network routing

⌚ Screenshot: Container Created

Home > sivagamistudentstorage Storage account

sivagamistudentstorage | Containers

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

Search containers by prefix

Name	Last modified	Anonymous access level	Lease state
\$logs	8/23/2025, 12:29:45 PM	Private	Available
mycontainer	8/23/2025, 12:30:34 PM	Private	Available

Step 3: Upload Dataset

- Uploaded employees.csv file into mycontainer.

⌚ Screenshot: employees.csv Uploaded

The screenshot shows the Azure Storage Explorer interface. On the left, there's a sidebar with navigation links: Home, mycontainer, Overview, Diagnose and solve problems, Access Control (IAM), and Settings. The main area displays a file structure under 'mycontainer': raw/employees.csv. A search bar at the top right says 'Search blobs by prefix (case-sensitive)'. Below the search bar, there's a dropdown menu set to 'Only show active objects'. A table lists the blob 'employees.csv' with columns: Name, Last modified, Access tier, Blob type, Size, and Lease state. The blob was last modified on 8/23/2025, 10:02:19 PM, is in the Hot (Inferred) tier, is a Block blob, and has a size of 3.69 KiB. The lease state is Available.

Step 4: Generate SAS Token

- In storage account → Shared Access Signature.
- Enabled all permissions and generated token.
- Copied SAS token string for later use.

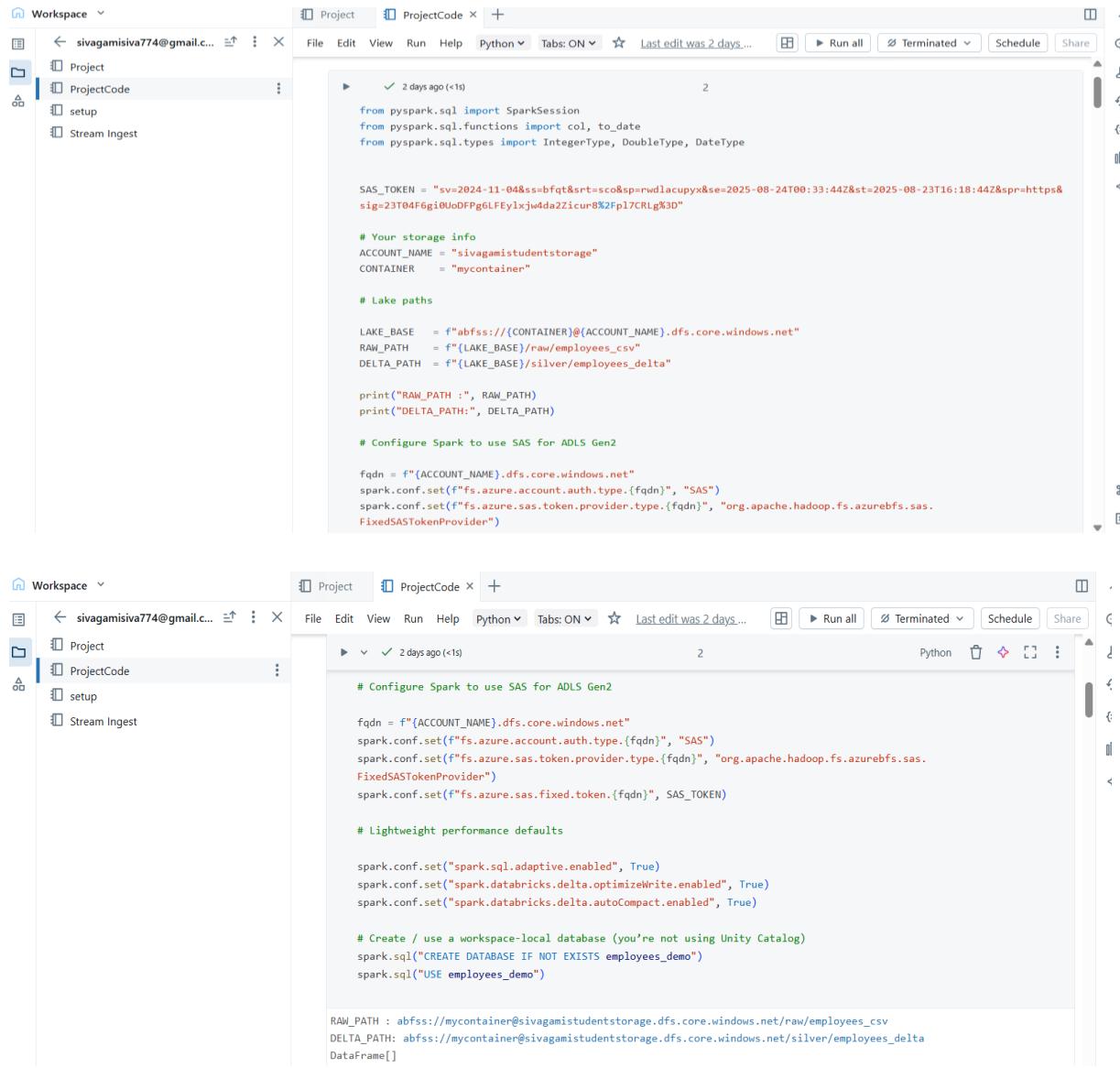
Screenshot: SAS Token Generated

The screenshot shows the 'Shared access signature' configuration page for the 'sivagamistudentstorage' storage account. The left sidebar includes links for Access Control (IAM), Data migration, Events, Storage browser, Partner solutions, Resource visualizer, Data storage (Containers, File shares, Queues, Tables), Security + networking (Networking, Access keys), Shared access signature, Encryption, Microsoft Defender for Cloud, Data management, Settings, Monitoring, Monitoring (classic), Automation, and Help. The main panel is titled 'Shared access signature' and contains sections for Allowed permissions (Read, Write, Delete, List, Add, Create, Update, Process, Immutable storage, Permanent delete), Blob versioning permissions (Enabled deletion of versions), Start and expiry date/time (Start: 25/08/2025, End: 25/08/2025, Duration: 14:18:02), Allowed IP addresses (example: 168.1.5.65 or 168.1.5.65-168.1.5.70), Allowed protocols (HTTPS only selected), Preferred routing tier (Basic (default) selected), Signing key (key1 selected), and a 'Generate SAS and connection string' button. Below the button, two connection strings are shown: 'Connection string' (blobEndpoint=https://sivagamistudentstorage.blob.core.windows.net/QueueEndpoint=https://sivagamistudentstorage.queue.core.windows.net/FileEndpoint=https://sivagamistudentstorage.file.core.windows.net/TableEndpoint=https://sivagamistudentstorage.table.core.windows.net/SharedAccessSignature=somekey) and 'SAS token' (sv=2024-11-04&ss=b&t=0302Z&st=2025-08-25T17:03:02Z&se=2025-08-25T08:48:02Z&spr=https&sig=fAMBHUNv8sdqWk7QoI0DrQc1UHCNrVafFeMieU%3D). There is also a 'Blob service SAS URL' field with the same SAS token value.

Step 5: Mount Storage in Databricks

- Used PySpark to mount container using SAS token.

 *Screenshot: Mounting Code in Databricks*



The image shows two screenshots of the Databricks workspace interface. Both screenshots display a Python notebook titled 'ProjectCode' with a single cell containing code. The code is used to configure a PySpark session to use an Azure Data Lake Storage Gen2 account via SAS tokens.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, to_date
from pyspark.sql.types import IntegerType, DoubleType, DateType

SAS_TOKEN = "sv=2024-11-04&ss=bfqt&srt=sco&sp=rwdrlacupyx&se=2025-08-24T00:33:44Z&st=2025-08-23T16:18:44Z&spr=https&sig=23T04F6g10UoDFPg6LFeylxju4da2Z1cur8%2Fp17CRLgX3D"

# Your storage info
ACCOUNT_NAME = "sivagamistudentstorage"
CONTAINER = "mycontainer"

# Lake paths
LAKE_BASE = f"abfss://{CONTAINER}@{ACCOUNT_NAME}.dfs.core.windows.net"
RAW_PATH = f"{LAKE_BASE}/raw/employees_csv"
DELTA_PATH = f"{LAKE_BASE}/silver/employees_delta"

print("RAW_PATH : ", RAW_PATH)
print("DELTA_PATH: ", DELTA_PATH)

# Configure Spark to use SAS for ADLS Gen2

fqdn = f"{ACCOUNT_NAME}.dfs.core.windows.net"
spark.conf.set("fs.azure.account.auth.type.{fqdn}", "SAS")
spark.conf.set("fs.azure.sas.token.provider.type.{fqdn}", "org.apache.hadoop.fs.azurebfs.sas.FixedSASTokenProvider")

spark.conf.set("fs.azure.sas.fixed.token.{fqdn}", SAS_TOKEN)

# Lightweight performance defaults

spark.conf.set("spark.sql.adaptive.enabled", True)
spark.conf.set("spark.databricks.delta.optimizeWrite.enabled", True)
spark.conf.set("spark.databricks.delta.autoCompact.enabled", True)

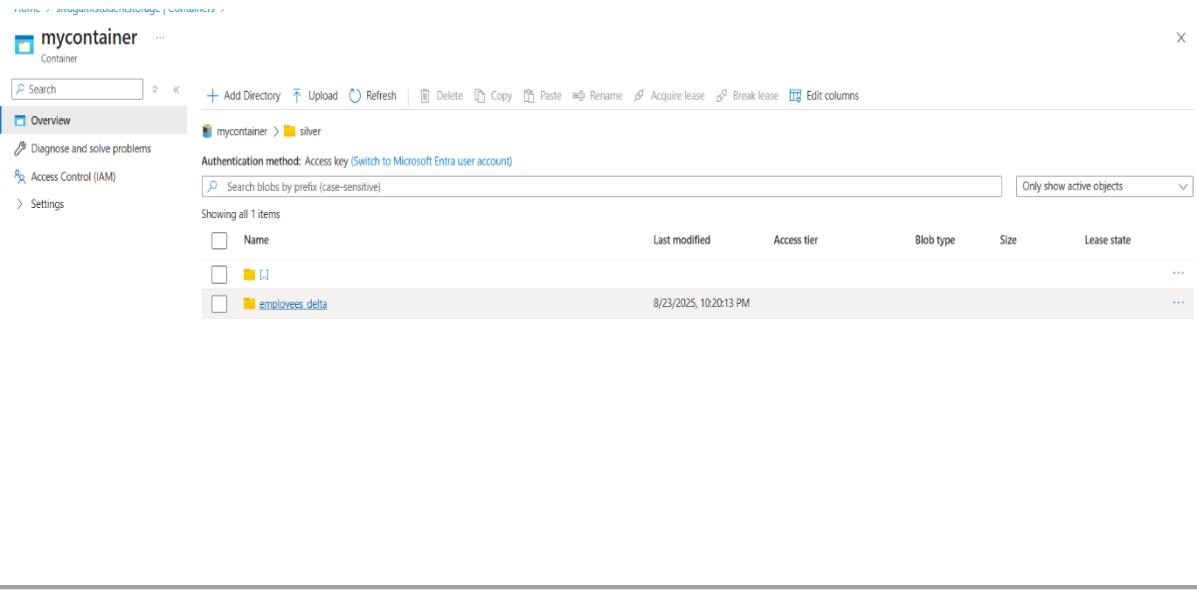
# Create / use a workspace-local database (you're not using Unity Catalog)
spark.sql("CREATE DATABASE IF NOT EXISTS employees_demo")
spark.sql("USE employees_demo")

RAW_PATH : abfss://mycontainer@sivagamistudentstorage.dfs.core.windows.net/raw/employees_csv
DELTA_PATH: abfss://mycontainer@sivagamistudentstorage.dfs.core.windows.net/silver/employees_delta
DataFrame[]
```

Step 6: Create Employees Delta File

- Converted raw CSV into Delta format.
- Saved into mycontainer/silver/employees_delta.

Screenshot: employees_delta File Created



Step 7: Read CSV from Lake

- Used spark.read.csv() to read employees.csv.

Screenshot: Read CSV Code Output

Read CSV from the lake (RAW zone)

```
▶ ▾ ✓ 2 days ago (9s) 4 Python ⚡ ⌂
# Point to the exact file in RAW
raw_csv = f"{RAW_PATH}/employees.csv"

# Read with header + infer schema
df_raw = (spark.read
          .option("header", True)
          .option("inferSchema", True)
          .csv(raw_csv))

df_raw.show(5)
df_raw.printSchema()
print("Rows:", df_raw.count())
```

```

▶ (5) Spark Jobs
df_raw: pyspark.sql.dataframe.DataFrame = [EMPLOYEE_ID: integer, FIRST_NAME: string ... 9 more fields]
| 201| Michael|Hartstein|MHARTSTE|515.123.5555|17-FEB-04| MK_MAN| 13000| - | 100|
0|
| 202| Pat| Fay| PFAY|603.123.6666|17-AUG-05| MK_REP| 6000| - | 201|
0|
+-----+-----+-----+-----+-----+-----+-----+-----+
+
only showing top 5 rows
root
|-- EMPLOYEE_ID: integer (nullable = true)
|-- FIRST_NAME: string (nullable = true)
|-- LAST_NAME: string (nullable = true)
|-- EMAIL: string (nullable = true)
|-- PHONE_NUMBER: string (nullable = true)
|-- HIRE_DATE: string (nullable = true)
|-- JOB_ID: string (nullable = true)
|-- SALARY: integer (nullable = true)
|-- COMMISSION_PCT: string (nullable = true)
|-- MANAGER_ID: string (nullable = true)
|-- DEPARTMENT_ID: integer (nullable = true)

Rows: 50

```

Step 8: Write to File (Delta Format)

- Wrote employees dataset into Delta format for optimization.

Screenshot: Writing to Delta File



The screenshot shows a Databricks notebook interface. The left sidebar displays a project structure with 'Project', 'ProjectCode', 'setup', and 'Stream Ingest'. The main area has a title 'Write Delta (Silver) to the lake + Register table'. A code cell contains the following Python code:

```

# Write to Delta (Silver)
(df.coalesce(1)
 .write
 .format("delta")
 .mode("overwrite")
 .save(DELTA_PATH))

# Register external Delta table
spark.sql("DROP TABLE IF EXISTS employees_delta")
spark.sql(f"""
CREATE TABLE employees_delta
USING DELTA
LOCATION '{DELTA_PATH}'
""")
spark.sql("DESCRIBE DETAIL employees_delta").show(truncate=False)
spark.sql("SELECT COUNT(*) AS total FROM employees_delta").show()

```

12. Analysis Results

◆ Query 1: Top 5 Highest Paid Employees

⌚ SQL Query + Result

The screenshot shows a Jupyter Notebook interface with a sidebar containing project files: Project, ProjectCode, setup, and Stream Ingest. The main area displays a code cell with the following content:

```
# Top 5 highest paid
spark.sql("""
SELECT FIRST_NAME, LAST_NAME, JOB_ID, SALARY
FROM employees_delta
ORDER BY SALARY DESC
LIMIT 5
""").show()
```

Below the code cell, the output is shown under '(1) Spark Jobs':

FIRST_NAME	LAST_NAME	JOB_ID	SALARY
Steven	King	AD_PRES	24000.0
Neena	Kochhar	AD_VP	17000.0
Lex	De Haan	AD_VP	17000.0
Michael	Hartstein	MK_MAN	13000.0
Shelley	Higgins	AC_MGR	12008.0

◆ Query 2: Employees per Department

⌚ SQL Query + Result

The screenshot shows a Jupyter Notebook interface with a sidebar containing project files: Project, ProjectCode, setup, and Stream Ingest. The main area displays a code cell with the following content:

```
# Employees per department
spark.sql("""
SELECT DEPARTMENT_ID, COUNT(*) AS emp_count
FROM employees_delta
GROUP BY DEPARTMENT_ID
ORDER BY emp_count DESC
""").show()
```

Below the code cell, the output is shown under '(2) Spark Jobs':

DEPARTMENT_ID	emp_count
50	23
100	6
30	6
60	5
90	3
20	2
110	2
10	1
40	1
70	1

◆ Query 3: Average Salary by Job Role

⌚ SQL Query + Result

ProjectCode

- setup
- Stream Ingest

```
# Average salary by job role
spark.sql("""
SELECT JOB_ID, ROUND(AVG(SALARY),2) AS avg_salary, COUNT(*) AS emp_count
FROM employees_delta
GROUP BY JOB_ID
ORDER BY avg_salary DESC
""").show()
```

(2) Spark Jobs

JOB_ID	avg_salary	emp_count
AD_PRES	24000.0	1
AD_VP	17000.0	2
MK_MAN	13000.0	1
AC_MGR	12008.0	1
FI_MGR	12008.0	1
PU_MAN	11000.0	1
PR_REP	10000.0	1
AC_ACCOUNT	8300.0	1
FI_ACCOUNT	7920.0	5
ST_MAN	7280.0	5
HR REP	6500.0	1
MK REP	6000.0	1
IT_PROG	5760.0	5
AD_ASST	4400.0	1
PU_CLERK	2780.0	5
ST_CLERK	2750.0	16
SH_CLERK	2600.0	2

◆ Query 4: Manager-Employee Counts

⌚ SQL Query + Result

ProjectCode

- setup
- Stream Ingest

```
# Manager-employee counts
spark.sql("""
SELECT MANAGER_ID, COUNT(*) AS direct_reports
FROM employees_delta
WHERE MANAGER_ID IS NOT NULL
GROUP BY MANAGER_ID
ORDER BY direct_reports DESC
""").show()
```

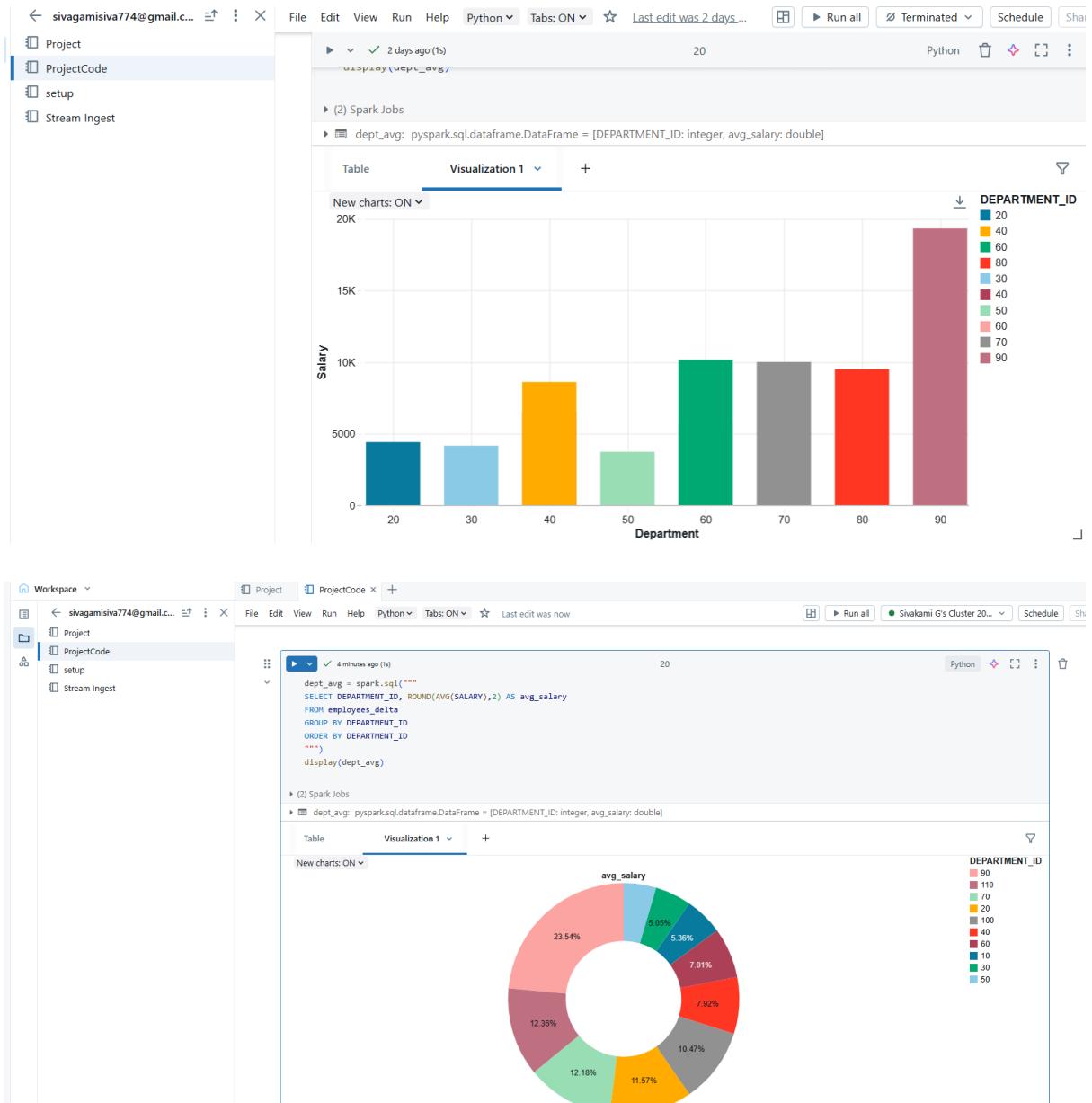
(2) Spark Jobs

MANAGER_ID	direct_reports
100	9
101	5
108	5
114	5
123	4
121	4
103	4
122	4
120	4
124	2
102	1
205	1
201	1

13. Successful Output Generated - Chart Visualization

- Used Databricks chart option to visualize employees per department.

Chart Visualization



14. Implementation Tasks Performed

- Set up Azure Storage Account and container.
- Uploaded employees.csv dataset.
- Created and configured Azure Databricks Workspace.

- Generated SAS Token for secure access.
 - Mounted ADLS to Databricks File System (DBFS).
 - Explored dataset with PySparkSQL queries.
 - Converted raw data to Delta format (Bronze → Silver).
 - Performed analytics and insights extraction.
 - Built visualizations inside Databricks notebooks.
-

15. Strategies that can be used in Optimising the process

- **Use Delta Lake Format** – Enables ACID transactions, schema enforcement, and faster queries compared to raw CSV/Parquet.
 - **Partitioning & Bucketing** – Organize data by columns (e.g., department, date) for efficient filtering and reduced scan times.
 - **Caching & Persisting Data** – Cache frequently used DataFrames or tables in Databricks memory for faster iterative queries.
 - **Optimize File Sizes** – Compact small files into larger ones to reduce overhead and improve Spark performance.
 - **Leverage Z-Ordering & Indexing** – Use Databricks Z-ORDER and indexes for efficient query pruning and faster lookups.
-

16. Conclusion

This project successfully demonstrated how to:

- Connect Azure Databricks with Data Lake.
- Explore and optimize data using PySparkSQL.
- Perform analytics and generate visual insights.
- Optimize data storage with Delta Lake.

The integration of Databricks and ADLS provided a **scalable, efficient, and powerful data analysis environment** for real-world data engineering tasks.