

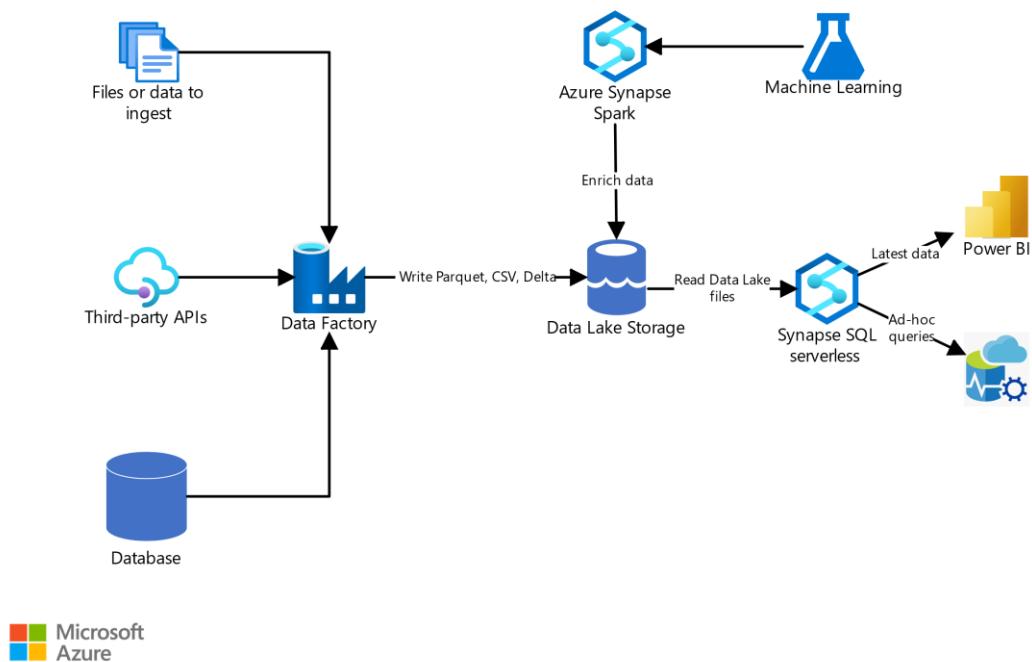
# Data-Engineering-Project-Using-Azure-Databricks

## Data Lake Exploration and Optimization

### Project Overview:

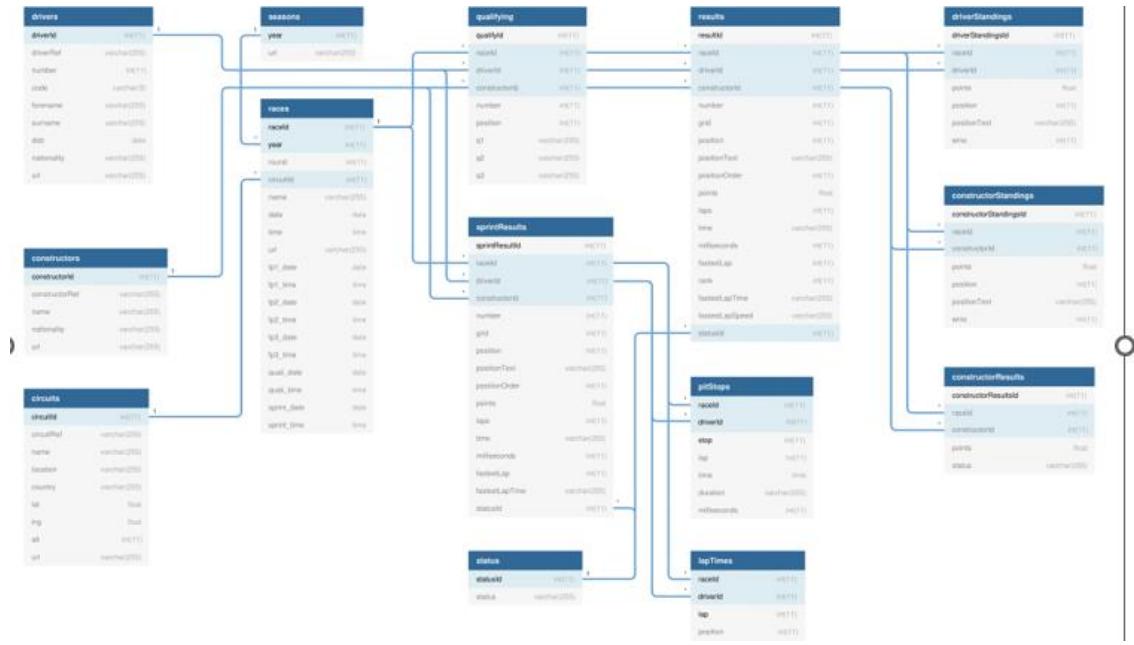
**Objective:** This project aims to explore and optimize data stored in an Azure Data Lake using PySparkSQL on the Azure Databricks platform. By leveraging PySparkSQL's functionalities, you'll gain insights into the data and improve its efficiency for further analysis or downstream applications.

### Architecture diagram:



## ER Diagram:

The structure of the database is shown in the following ER Diagram and explained in the Database File.



## Components:

- Azure Data Lake Storage (ADLS):** Stores the raw data in various formats (parquet, CSV, etc.).
- Azure Databricks Cluster:** Provides the processing environment with PySpark installed.
- PySpark Notebook:** Contains Python code using PySparkSQL functions for data exploration, transformation, and optimization.
- Data Visualization Tool (Optional):** Used to create visualizations like charts or graphs from the processed data (e.g., Matplotlib, Plotly).
- Reporting Tool (Optional):** Used to generate reports based on the analyzed data (e.g., Power BI, Tableau).

## Data Flow:

- Data Ingestion:** Raw data is ingested from various sources (e.g., sensors, databases) into ADLS.

2. **Data Access:** The PySpark notebook in the Azure Databricks cluster reads data from ADLS using PySparkSQL functions.
3. **Data Exploration and Transformation:** The notebook performs various operations on the data:
  - **EDA:** Analyze data characteristics using PySparkSQL functions (e.g., describe, groupBy).
  - **Cleaning:** Address missing values, outliers, and inconsistencies.
  - **Transformation:** Filter, aggregate, join datasets, and perform other manipulations.
  - **Optimization:** Partition data, cache frequently used datasets, define UDFs (if applicable).
4. **Analysis and Visualization:** The processed data is analyzed within the notebook. Optionally, data visualization tools can be used to create charts and graphs.
5. **Reporting (Optional):** The insights and visualizations can be exported to reporting tools for further analysis and communication.

## How it works:

Code snippet

```
graph LR
A[Azure Data Lake Storage (ADLS)] --> B{Raw Data}
B --> C{Azure Databricks Cluster}
C --> D{PySpark Notebook}
D --> E{Data Exploration & Transformation}
D --> F{Data Visualization (Optional)}
E --> G{Analysis & Insights}
G --> H{Reporting (Optional)}
```

## Data Flow:

1. The user submits a request to explore and optimize data.
2. The Azure Databricks Workspace manages and allocates resources (cluster) for the project.
3. The PySpark notebook in the cluster reads data from ADLS.
4. The notebook performs data exploration, transformation, and optimization.
5. Analysis and insights are generated from the processed data.
6. Optionally, the data can be exported for visualization.
7. Optionally, the insights can be exported to a reporting tool for further analysis and communication.

This block diagram focuses on the high-level interaction between components and the data flow. It provides a clearer picture of the user interaction and the overall processing pipeline.

## Execution Overview:

### **1. Setup:**

- **Provision Azure Databricks resources:**
  - Create an Azure Databricks workspace.
  - Launch a cluster with appropriate configurations for your data size and processing needs.
- **Mount ADLS:**
  - Use dbutils library to mount your ADLS account to the cluster.
- **Prepare PySpark notebook:**
  - Create a new notebook in your workspace.
  - Import necessary libraries (e.g., pyspark.sql).
  - Define connection details for ADLS.

### **2. Data Exploration:**

- **Read data from ADLS:**

- Use functions like spark.read.parquet or spark.read.csv based on data format.
- **Analyze schema:**
  - Utilize df.printSchema() to understand data structure, including column names and data types.
- **Perform exploratory data analysis (EDA):**
  - Utilize functions like df.describe(), df.groupBy(...).count(), etc., to gain insights into data distribution, missing values, and summary statistics.

### **3. Data Transformation and Optimization:**

- **Data cleaning:**
  - Address missing values with df.fillna(value) or impute them using appropriate methods.
  - Handle outliers or inconsistencies as needed.
- **Data transformation:**
  - Filter specific data subsets using df.filter(condition).
  - Aggregate data with functions like df.groupBy(...).agg(avg("numeric\_column")).
  - Join datasets from different files using df.join(df2, on="column\_name", how="inner").
- **Data optimization:**
  - Partition data based on specific columns using df.repartition(n, col1, col2).
  - Cache frequently accessed data frames using df.cache().
  - Consider cost-based optimization using EXPLAIN and indexing techniques for large datasets.

### **4. Analysis and Reporting:**

- **Analyze data:**
  - Utilize the processed data frame for further analysis and visualization.
- **Visualize insights (Optional):**
  - Integrate libraries like Matplotlib or Plotly to create visualizations.

- **Generate reports (Optional):**
  - Export data or visualizations to formats like CSV, parquet, or notebooks for sharing or further analysis.

## 5. Cleanup:

- **Terminate Azure Databricks cluster:**
  - Release resources when processing is complete.
- **Clear notebook variables (Optional):**
  - Ensure proper memory management within the notebook.

## Azure Resources Used for this Project:

Here are the Azure resources used for the data exploration and optimization project in Azure Databricks:

### Essential Resources:

- **Azure Databricks Workspace:** Serves as the central platform for creating and managing your Databricks projects, including notebooks, clusters, and data access.
- **Azure Databricks Cluster:** Provides a scalable compute environment with Apache Spark installed, where your PySpark notebooks execute the data exploration and optimization operations.
- **Azure Data Lake Storage (ADLS):** Serves as the data source, storing your raw data in various formats like parquet, CSV, etc.

### Optional Resources:

- **Azure Synapse Analytics:** Can be used for data warehousing and potentially integrated with Azure Databricks for advanced analytics and data management. (Optional for this specific project)
- **Azure Data Factory:** Can be used for data orchestration and automating data pipelines, potentially moving data from various sources to ADLS before exploration. (Optional for this specific project)
- **Azure Key Vault:** Can be used to securely manage secrets and credentials used within your PySpark notebook for accessing ADLS or other resources. (Optional for this specific project)

## Project Requirements:

The requirements for this project are broken down into six different parts which are

### **1. Data Access:**

- **Format:** Specify the format of the data stored in ADLS (e.g., parquet, CSV, JSON). Knowing the format helps determine the appropriate reading method in PySparkSQL.
- **Location:** Provide the path to the data location within your ADLS Gen2 storage account. This includes the container name and folder path.
- **Credentials:** Outline the access method for ADLS. You might need to configure service principals or use managed identities to grant your Azure Databricks cluster access to your ADLS data.

### **2. Data Exploration:**

- **Specific Tasks:** List specific exploratory analysis tasks you want to achieve. This may include:
  - **Descriptive statistics:** Analyzing data distribution like mean, standard deviation, and count for numerical columns.
  - **Data quality checks:** Identifying missing values, outliers, and inconsistencies within the data.
  - **Data understanding:** Analyzing data schema, column names, and data types to understand the data structure.
- **Visualization:** Define if you need to create visualizations (e.g., histograms, scatter plots) during exploration to gain visual insights.

### **3. Data Transformation:**

- **Filtering:** Define specific conditions for filtering data subsets based on specific column values or conditions.
- **Aggregation:** Specify the desired aggregation operations like calculating averages, sums, or counts across groups of data.

- **Joining:** If applicable, define the join conditions and types (e.g., inner join, left join) for combining data from multiple datasets or tables within ADLS.
- **Other Transformations:** Specify any additional transformations needed, such as renaming columns, handling null values, or converting data types.

## 4. Data Optimization:

- **Partitioning:** Outline the columns for partitioning the data. This improves query performance by allowing Spark to efficiently access relevant data subsets.
- **Caching:** Identify frequently used data frames or transformations that can benefit from caching in memory for faster access during subsequent operations.
- **Cost-based Optimization:** If dealing with large datasets, consider using EXPLAIN to analyze query execution plans and identify potential optimizations like indexing relevant columns.

## 5. Analysis and Reporting:

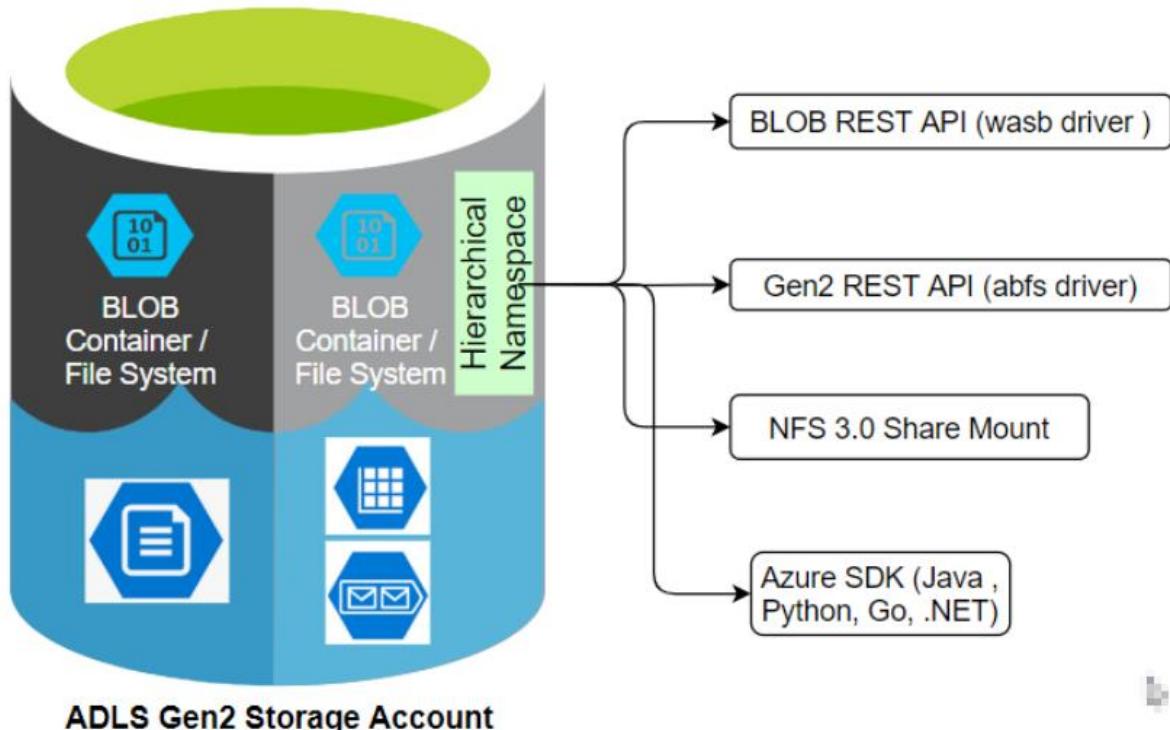
- **Insights:** Define the specific insights you want to extract from the data. This might involve analyzing trends, relationships between variables, or identifying patterns within the data.
- **Reporting:** Specify how you want to report the findings. This could include generating visualizations using libraries like Matplotlib or Plotly, creating reports in formats like CSV or PDF, or integrating the results with other reporting tools.

## 6. Non-Functional Requirements:

- **Performance:** Specify any performance expectations for data processing and query execution times.
- **Scalability:** Define if the project needs to handle growing data volumes in the future, requiring adjustments to cluster size or optimization techniques.

- **Security:** Outline any security requirements related to access controls, data encryption, or audit logging for both your data in ADLS and your processing environment in Azure Databricks.

## ADLS(Azure Data Lake Storage)



Azure Data Lake Storage (ADLS) is a specific component within the broader Azure Data Lake offering. It focuses on the **storage aspect** of managing large-scale data. Here's a deeper dive into ADLS:

### What is ADLS?

ADLS is a **cloud-based storage solution** designed to handle **massive amounts of data in any format**, including structured (databases, spreadsheets), semi-structured (JSON, XML), and unstructured (text, images, videos). It serves as the foundation for building **enterprise data lakes** on Microsoft Azure.

## Key Features:

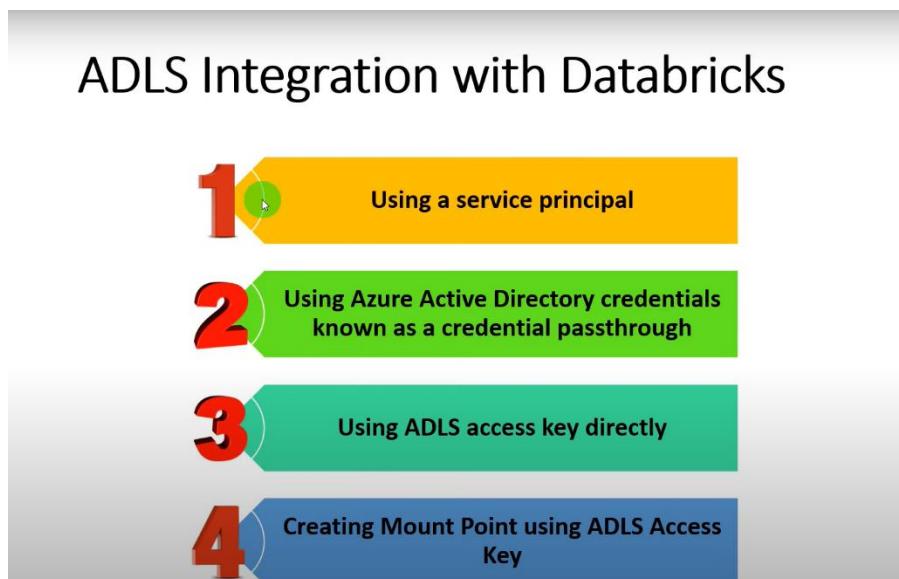
- **Scalability:** ADLS can scale **elastically** to accommodate ever-increasing data volumes, allowing you to store petabytes of data efficiently.
- **Cost-effectiveness:** It offers **tiered storage** options, letting you optimize costs based on your data access frequency. You pay only for the storage used and the retrieval operations performed.
- **Data Security:** ADLS prioritizes data security through features like **access control lists (ACLs)**, **encryption at rest and in transit (TLS)**, and **auditing capabilities**. This ensures your sensitive data is protected.
- **File System Semantics:** ADLS provides **POSIX-compliant file system semantics**. This means you can access and manage your data using familiar file system commands and tools, making it easier to work with for users accustomed to traditional file systems.

## Benefits of using ADLS:

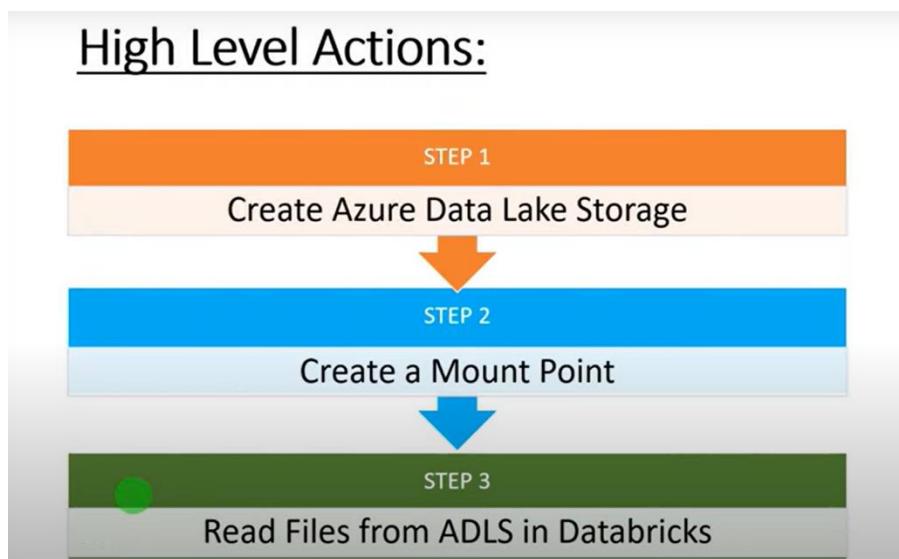
- **Centralized Data Repository:** Store all your data, regardless of format or size, in a single location for easy access and management.
- **Simplified Analytics:** Facilitate big data analytics by storing data in its native format, eliminating the need for pre-processing before analysis.
- **Flexibility:** Integrate with various analytics tools and frameworks like Azure Databricks, Azure HDInsight, and Power BI, allowing you to choose the best fit for your specific needs.

**Overall, ADLS is a powerful tool for storing and managing large datasets in the cloud. Its scalability, cost-effectiveness, and security features make it a valuable asset for organizations with big data needs.**

## Four Methods to Integrate ADLS with Databricks



In this Project we are using ADLS Access Key directly to connect with Azure Databricks

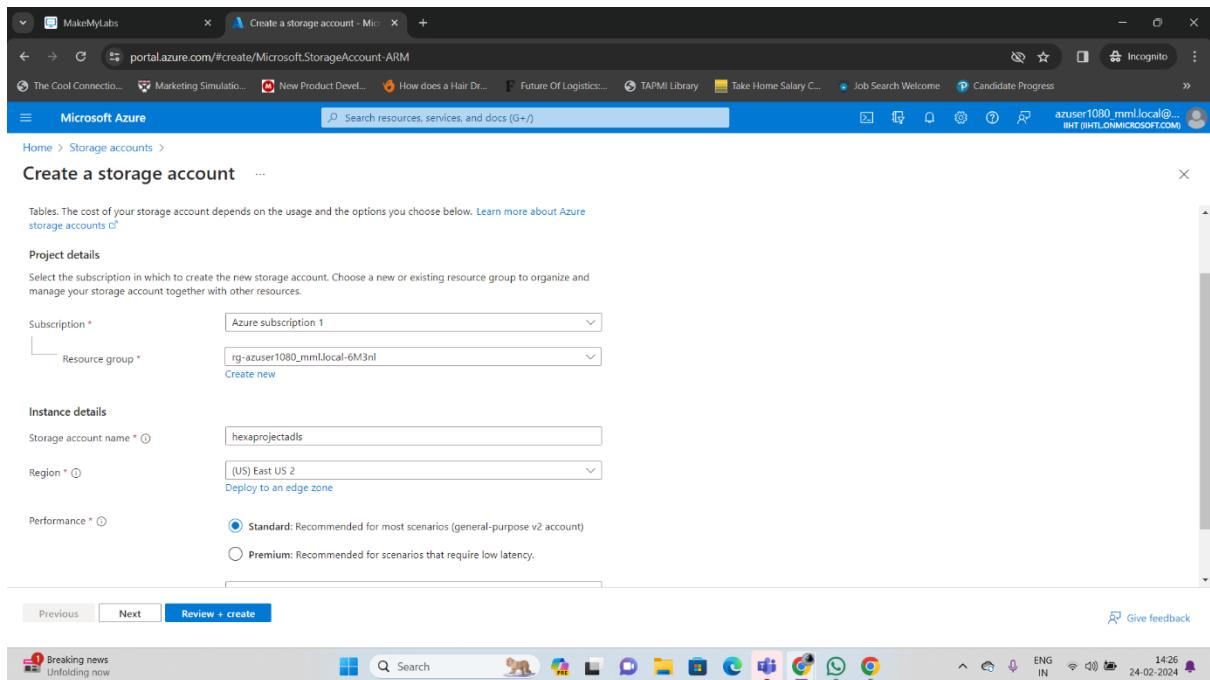


## Tasks performed:

### 1. Data Exploration:

Login in to Azure Portal and create a Storage Account

Name “hexaprojectadls”



## Enabling Hierarchical Namespace

The screenshot shows the 'Create a storage account' wizard on the Microsoft Azure portal. The current step is 'Configure security settings that impact your storage account'. Under 'Hierarchical Namespace', it is described as enabling file and directory semantics, accelerating big data analytics workloads, and enabling access control lists (ACLs). A checkbox for 'Enable hierarchical namespace' is checked. Below this, the 'Access protocols' section indicates that Blob and Data Lake Gen2 endpoints are provisioned by default. At the bottom, there are 'Previous', 'Next', and 'Review + create' buttons, along with a 'Give feedback' link.

## Review & and Create Storage Account

The screenshot shows the 'Create a storage account' wizard on the Microsoft Azure portal, specifically the 'Review + create' step. It displays the configuration details for the storage account:

Category	Setting	
Basics	Subscription	Azure subscription 1
	Resource group	rg-azuse1080_mmllocal-6M3n1
	Location	East US 2
	Storage account name	hexaprojectadls
	Performance	Standard
	Replication	Read-access geo-redundant storage (RA-GRS)
<b>Advanced</b>		
Enable hierarchical namespace	Enabled	
Enable SFTP	Disabled	
Enable network file system v3	Disabled	
Allow cross-tenant replication	Disabled	
Access tier	Hot	

At the bottom, there are 'Previous', 'Next', and 'Create' buttons, along with a 'Give feedback' link.

## Storage Account Created

The screenshot shows the Microsoft Azure portal's Deployment blade for a deployment named "hexaprojectadls\_1708764989020". The status is "Deployment succeeded". The deployment was started on 2/24/2024 at 2:26:42 PM with Correlation ID: 291b111-81d3-4acd-a781-6208a9b19612. The resource group is "rg-azuser1080\_mmilocal-6M3nl". On the right side, there are promotional cards for Cost Management, Microsoft Defender for Cloud, Free Microsoft tutorials, and Work with an expert.

## Click Go to Resource

The screenshot shows the Microsoft Azure portal's Storage account blade for the "hexaprojectadls" storage account. It displays deployment details: Resource group (move) is "rg-azuser1080\_mmilocal-6M3nl", Location is "eastus2", Primary/Secondary Location is "Primary: East US 2, Secondary: Central US", Subscription (move) is "Azure subscription 1", Subscription ID is "984f097c-963c-4eb6-a20d-839457ae9f08", Disk state is "Primary: Available, Secondary: Available", and Tags (edit) are "Add tags". The Properties tab is selected, showing Data Lake Storage settings like Hierarchical namespace (Enabled), Default access tier (Hot), Blob anonymous access (Disabled), Blob soft delete (Enabled (7 days)), Container soft delete (Enabled (7 days)), and Versioning (Disabled). The Security tab shows settings for REST API operations (Enabled), Storage account key access (Enabled), Minimum TLS version (Version 1.2), and Infrastructure encryption (Disabled). The Networking tab is partially visible.

## Create Container

### Name adls-container

The screenshot shows the Microsoft Azure portal interface. The user is navigating through the 'hexaprojectadls' storage account under the 'Containers' section. A modal window titled 'New container' is open, prompting the user to enter a name ('adls-container') and set the anonymous access level to 'Private (no anonymous access)'. The main pane displays a list of existing containers, including 'Slogs'. The status bar at the bottom right indicates the user's location as 'ENG IN' and the date as '24-02-2024'.

## Container Created

The screenshot shows the Microsoft Azure portal interface after the container creation. A success message box is visible in the top right corner, stating 'Successfully created storage container' and 'Successfully created storage container 'adls-container''. The main pane now lists two containers: 'Slogs' and 'adls-container'. The status bar at the bottom right indicates the user's location as 'ENG IN' and the date as '24-02-2024'.

# Upload a CSV file in the Container

The screenshot shows the Microsoft Azure Storage Container Overview page for the 'adls-container'. The container name is highlighted in blue. The page includes a toolbar with actions like Upload, Add Directory, Refresh, Rename, Delete, Change tier, Acquire lease, Break lease, and Give feedback. A search bar and a 'Show deleted objects' checkbox are also present. The main area displays a table with columns: Name, Modified, Access tier, Archive status, Blob type, Size, and Lease state. A message indicates 'No results'.

This screenshot is identical to the one above, showing the Microsoft Azure Storage Container Overview page for the 'adls-container'. The interface, toolbar, and table structure are the same, indicating no files have been uploaded yet.



The screenshot shows the 'Upload blob' dialog box overlaid on the Microsoft Azure Storage Container Overview page. The dialog has a cloud icon and a message stating '1 file(s) selected: annual-enterprise-survey-2021-financial-year-provisio...'. It includes a 'Browse for files' link and a checkbox for 'Overwrite if files already exist'. At the bottom are 'Advanced' and 'Upload' buttons, along with a 'Give feedback' link.



## File Uploaded

The screenshot shows the Microsoft Azure Storage Container Overview page. The container name is 'adls-container'. A success message at the top right says 'Successfully uploaded blob(s)' and 'Successfully uploaded 1 blob(s)'. The main table lists one blob: 'annual-enterprise-survey-2021-financial-year-prov...' with a size of 6.31 MB and an 'Available' lease state. The left sidebar includes sections for Overview, Diagnose and solve problems, Access Control (IAM), Settings (Shared access tokens, Manage ACL, Access policy, Properties, Metadata), and Data storage (Containers, File shares, Queues, Tables). The Windows taskbar at the bottom shows various pinned icons and the date/time as 24-02-2024.

## Now go to Storage Account get Access key

The screenshot shows the Microsoft Azure Storage Account Overview page for 'hexaprojectadls'. The left sidebar has sections for Events, Storage browser, Data storage (Containers, File shares, Queues, Tables), Security + networking (Networking, Access keys, Shared access signature, Encryption, Microsoft Defender for Cloud), and Data management (Storage tasks). The main area shows a table of containers: '\$logs' and 'adls-container'. Below the table, a link leads to 'https://portal.azure.com/#@ihtlonmicrosoft.com/resource/subscriptions/984f097c-963c-4eb6-a20d-839457ae9f08/resourceGroups/rg-azuser1080\_mmilocal-6M3n/providers/Microsoft.Storage/storageAccounts/hexaprojectadls/keys'. The Windows taskbar at the bottom shows various pinned icons and the date/time as 24-02-2024.

## Copy Access Key

The screenshot shows the Microsoft Azure portal interface. The left sidebar is collapsed. The main content area is titled "hexaprojectadls | Access keys". It displays two sets of access keys: "key1" and "key2". Each key has a "Rotate key" button, a "Last rotated" timestamp (2/24/2024), and a "Copy to clipboard" button. Below each key is a "Connection string" field. The "key1" connection string starts with "2TlrsKJ...". The "key2" connection string starts with "2TlreKJ...". The "key1" key value starts with "2TlrsKJ...". The "key2" key value starts with "2TlreKJ...".

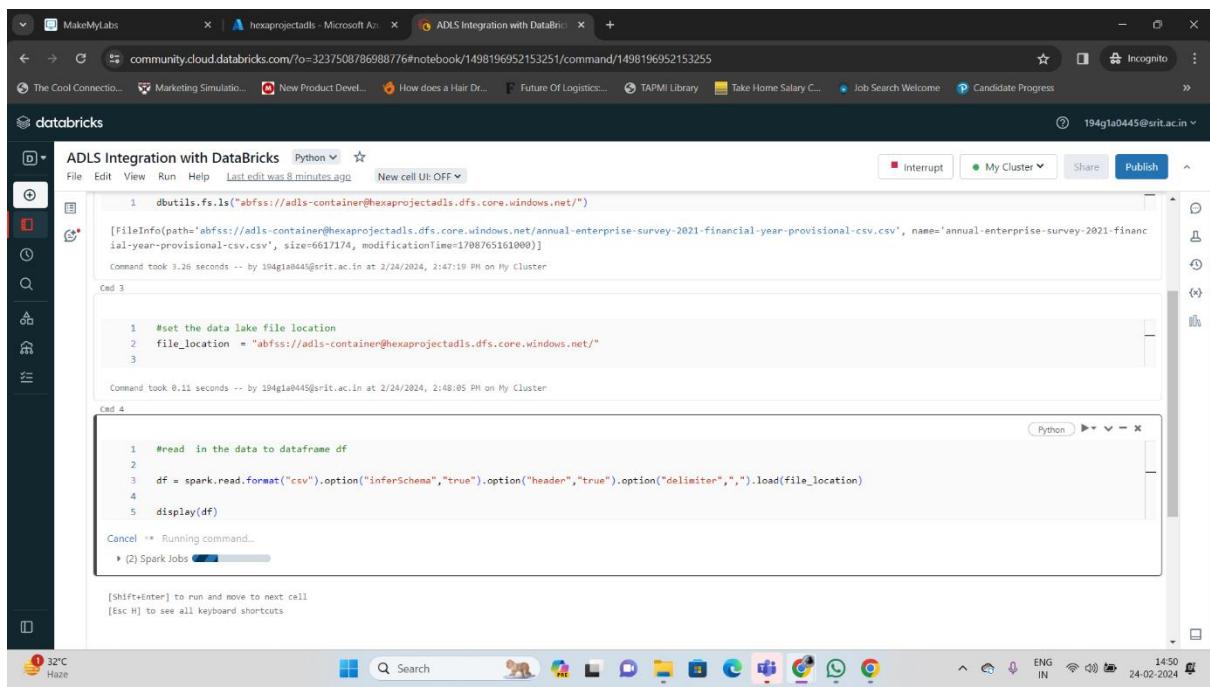
## Go to Databricks Portal

Run PySpark command by giving Storage account name and access key

The screenshot shows a Databricks notebook interface. The title bar says "ADLS Integration with DataBrics". The notebook is in Python mode. Cell 1 contains the following PySpark command:

```
1 spark.conf.set("fs.azure.account.key.hexaprojectadls.dfs.core.windows.net",
2 "2TlreKJ4thwbtP9wiDWw9x3NtQc1Gww5pzyea5FzygyjqD3ElkvHijUuYVVSzX...
3 ...
4 ...
5 )
```

The command uses the `spark.conf.set` method to set the Azure account key for the specified storage account.



```
1 dbutils.fs.ls("abfss://adls-container@hexaprojectadls.dfs.core.windows.net/")

[FileInfo(path='abfss://adls-container@hexaprojectadls.dfs.core.windows.net/annual-enterprise-survey-2021-financial-year-provisional-csv.csv', name='annual-enterprise-survey-2021-financial-year-provisional-csv.csv', size=6617174, modificationTime='198765161000')]

Command took 3.26 seconds -- by 194gia8445@srit.ac.in at 2/24/2024, 2:47:19 PM on My Cluster

Cell 3

1 #set the data lake file location
2 file_location = "abfss://adls-container@hexaprojectadls.dfs.core.windows.net/"
3

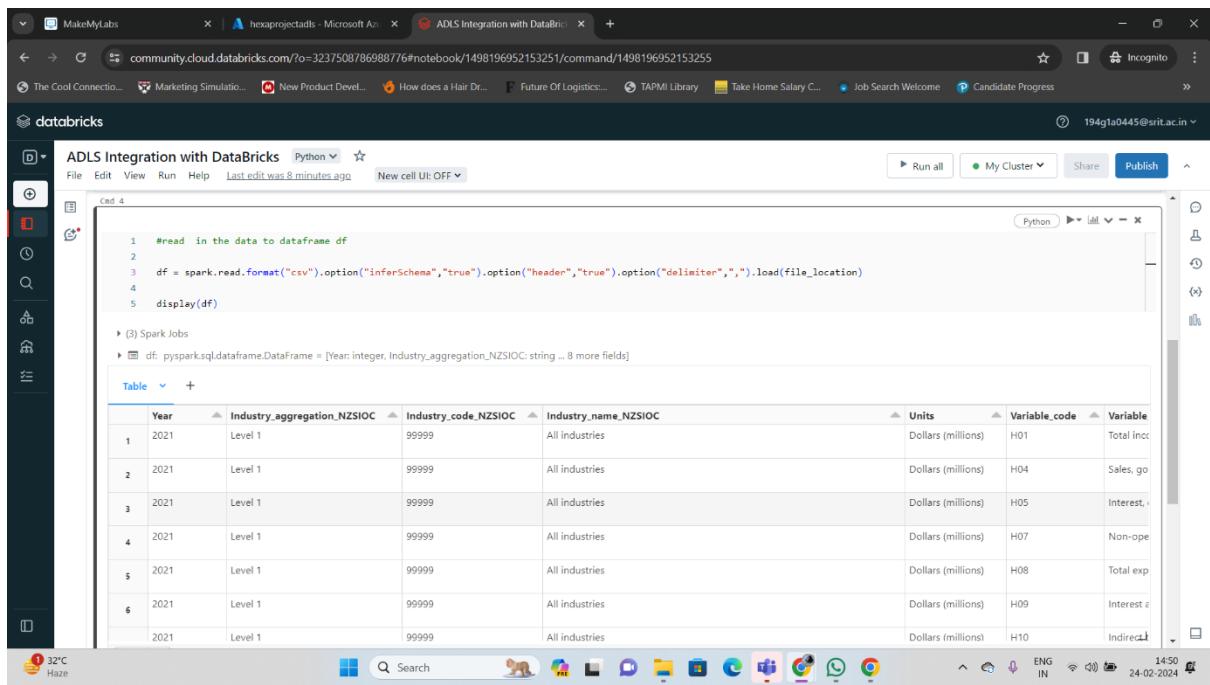
Command took 0.11 seconds -- by 194gia8445@srit.ac.in at 2/24/2024, 2:48:05 PM on My Cluster

Cell 4

1 #read in the data to dataframe df
2
3 df = spark.read.format("csv").option("inferSchema","true").option("header","true").option("delimiter",",").load(file_location)
4
5 display(df)

Cancel ** Running command...
▶ (2) Spark Jobs [2/24/2024 2:48:05 PM] [32C Haze]
```

## To read data to Dataframe [Df] & Display



Year	Industry_aggregation_NZSIOC	Industry_code_NZSIOC	Industry_name_NZSIOC	Units	Variable_code	Variable	
1	2021	Level 1	99999	All industries	Dollars (millions)	H01	Total incr.
2	2021	Level 1	99999	All industries	Dollars (millions)	H04	Sales, go
3	2021	Level 1	99999	All industries	Dollars (millions)	H05	Interest, r
4	2021	Level 1	99999	All industries	Dollars (millions)	H07	Non-ope
5	2021	Level 1	99999	All industries	Dollars (millions)	H08	Total exp
6	2021	Level 1	99999	All industries	Dollars (millions)	H09	Interest r
	2021	Level 1	99999	All industries	Dollars (millions)	H10	Indirect

## Perform Visualization in order to Explore Data

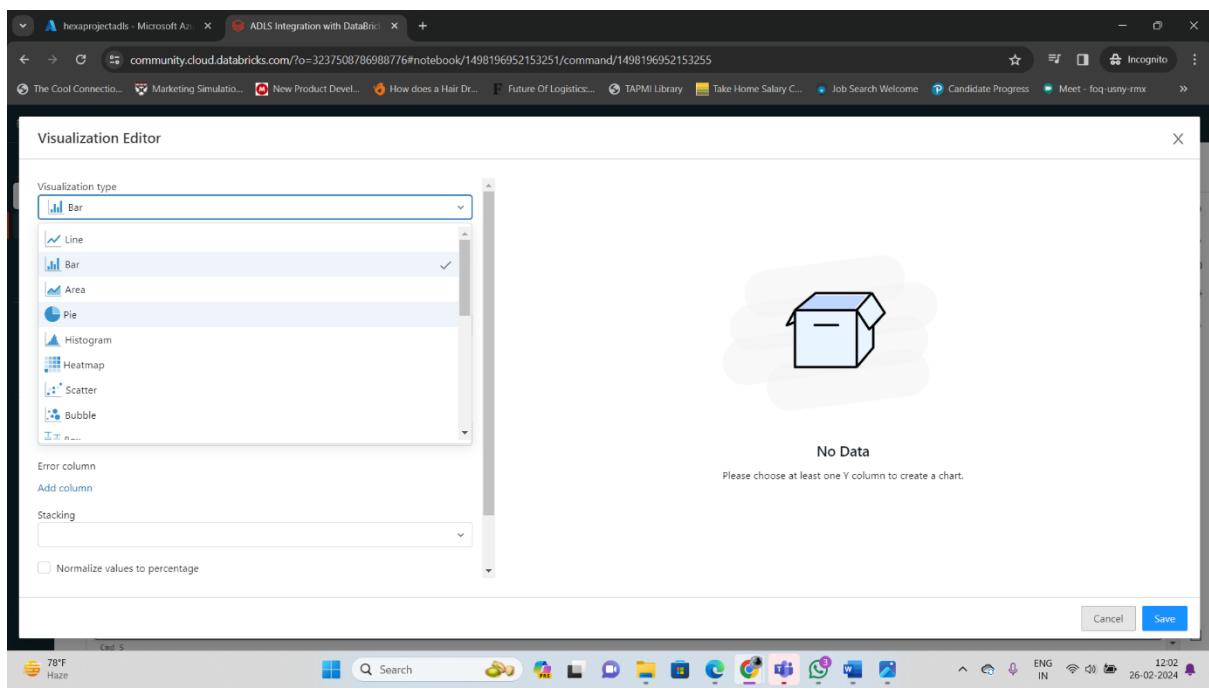
```
2 df = spark.read.format("csv").option("inferSchema","true").option("header","true").option("delimiter",",") .load(file_location)
3
4 display(df)
5
6
7 (3) Spark Jobs
8 df: pyspark.sql.dataframe.DataFrame = [Year: integer, Industry_aggregation_NZSIOC: string ... 8 more fields]
9
10 Table +
```

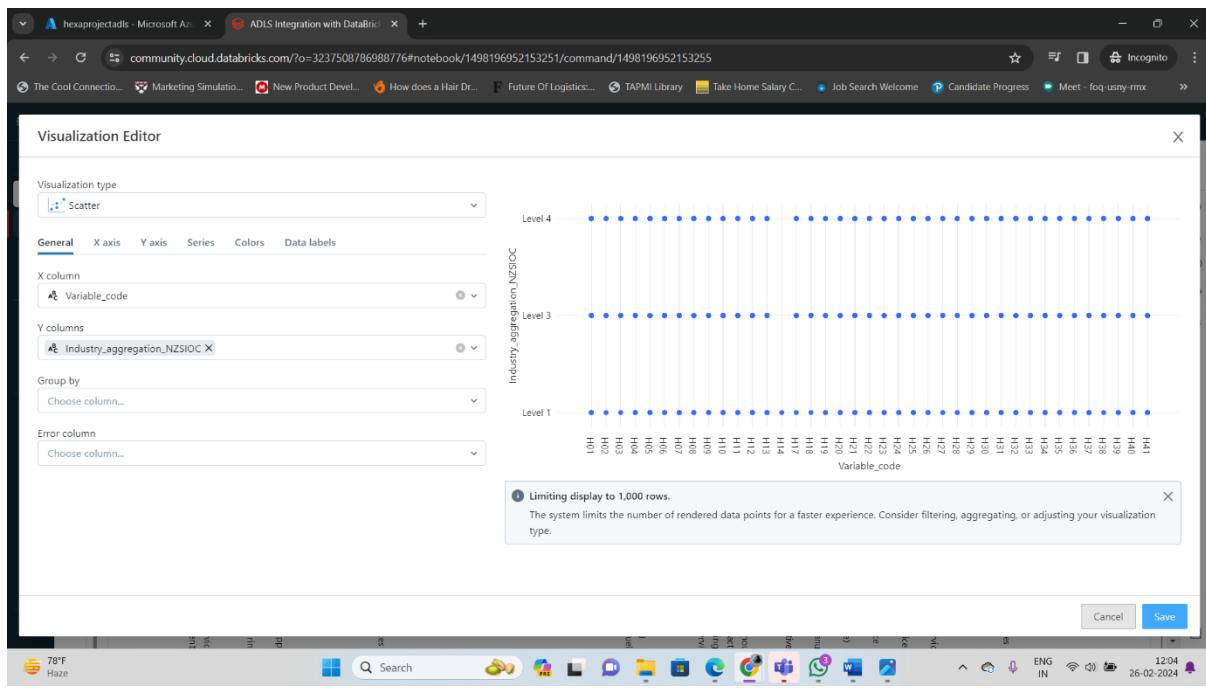
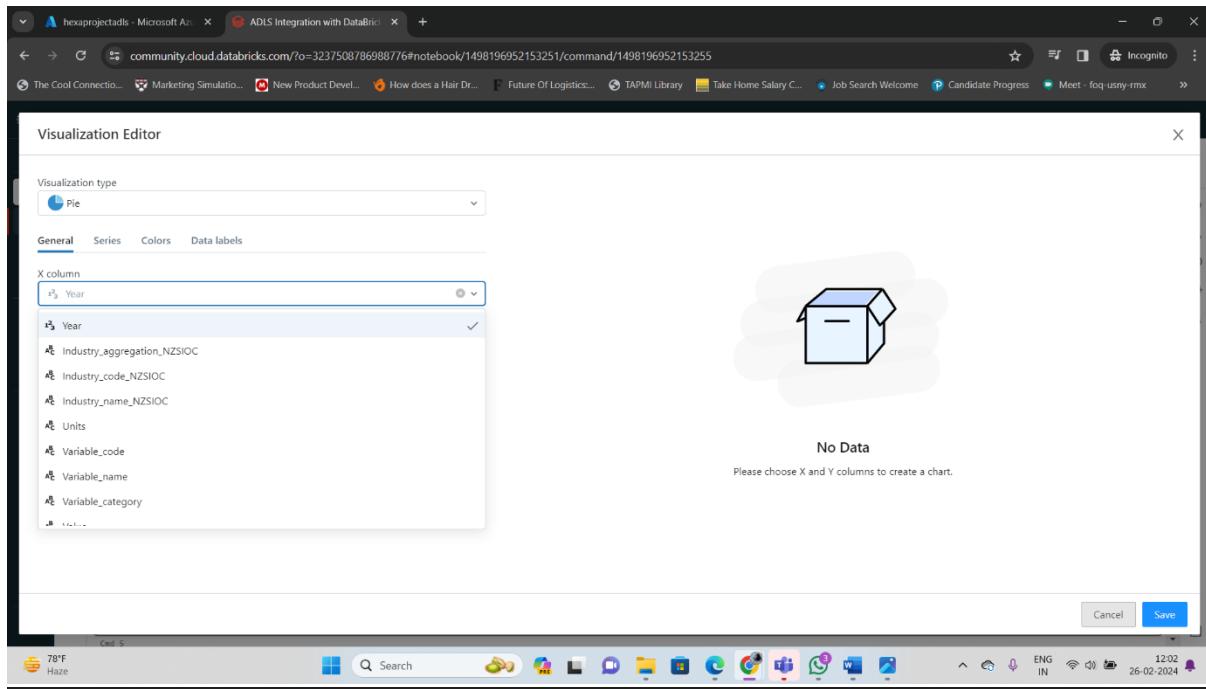
Year	Industry_aggregation_NZSIOC	Industry_code_NZSIOC	Industry_name_NZSIOC	Units	Variable_code	Variable
2021	Data Profile	99999	All industries	Dollars (millions)	H01	Total inc...
2021	Legacy Visualization	99999	All industries	Dollars (millions)	H04	Sales, go...
2021	Level 1	99999	All industries	Dollars (millions)	H05	Interest, i...
2021	Level 1	99999	All industries	Dollars (millions)	H07	Non-ope...
2021	Level 1	99999	All industries	Dollars (millions)	H08	Total exp...
2021	Level 1	99999	All industries	Dollars (millions)	H09	Interest z...
2021	Level 1	99999	All industries	Dollars (millions)	H10	Indirect...

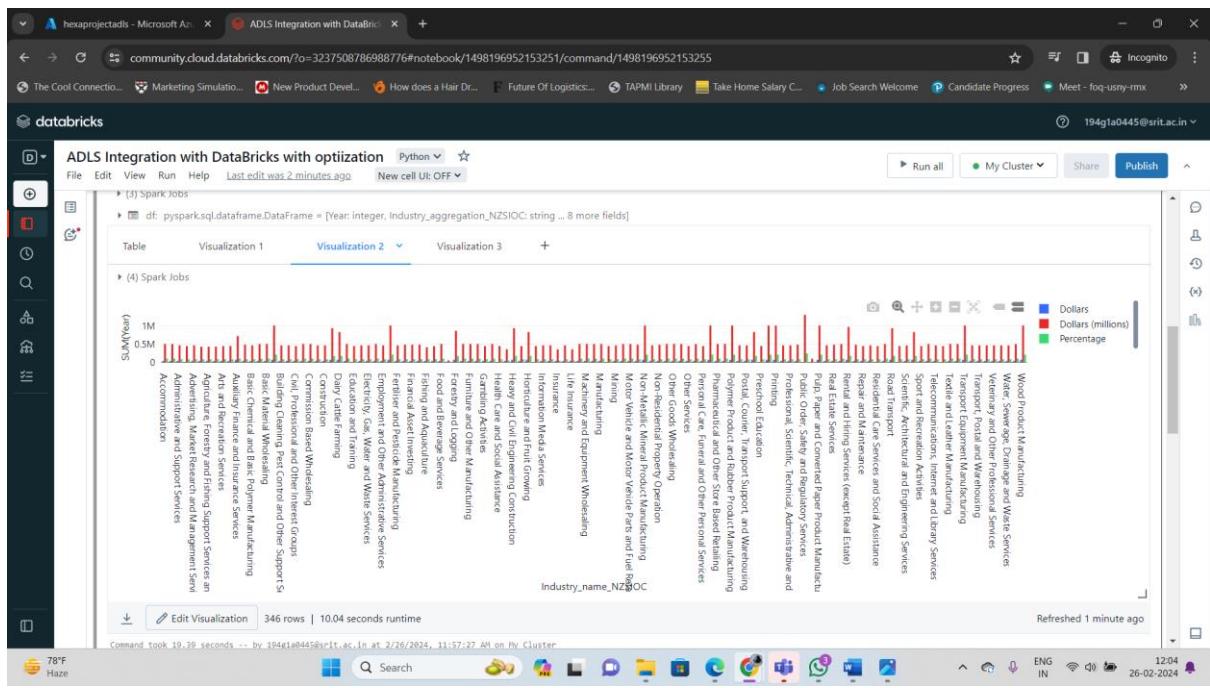
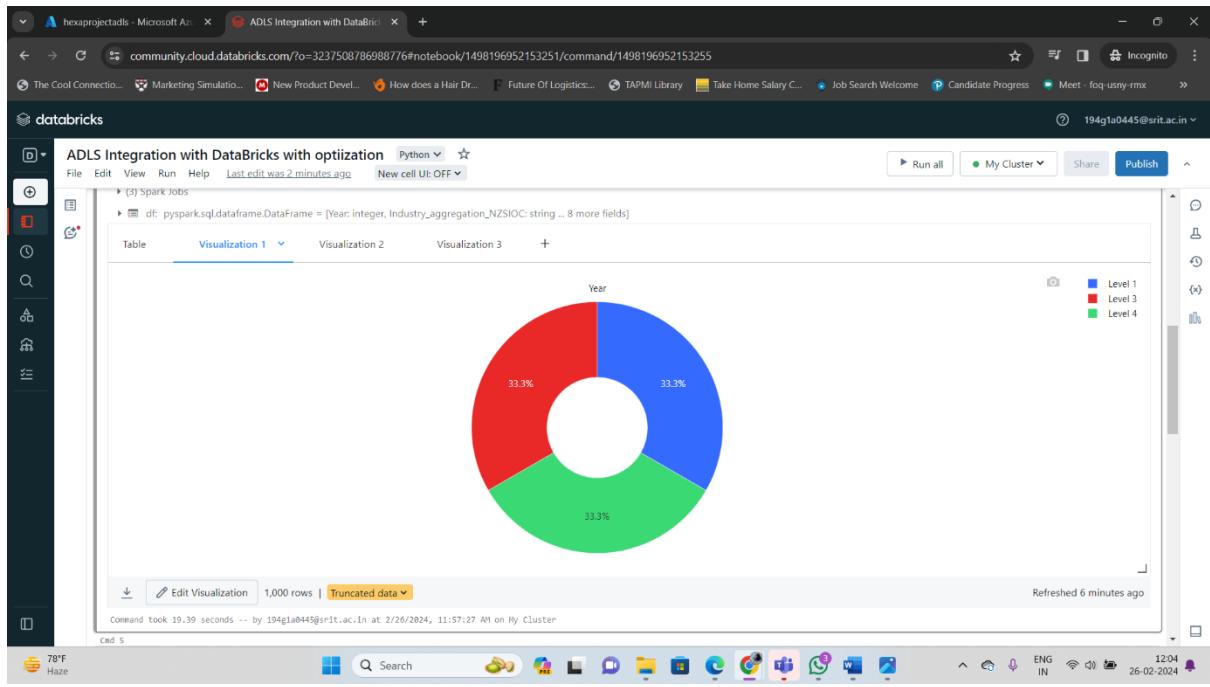
10,000 rows | Truncated data | 19.39 seconds runtime  
Refreshed 3 minutes ago

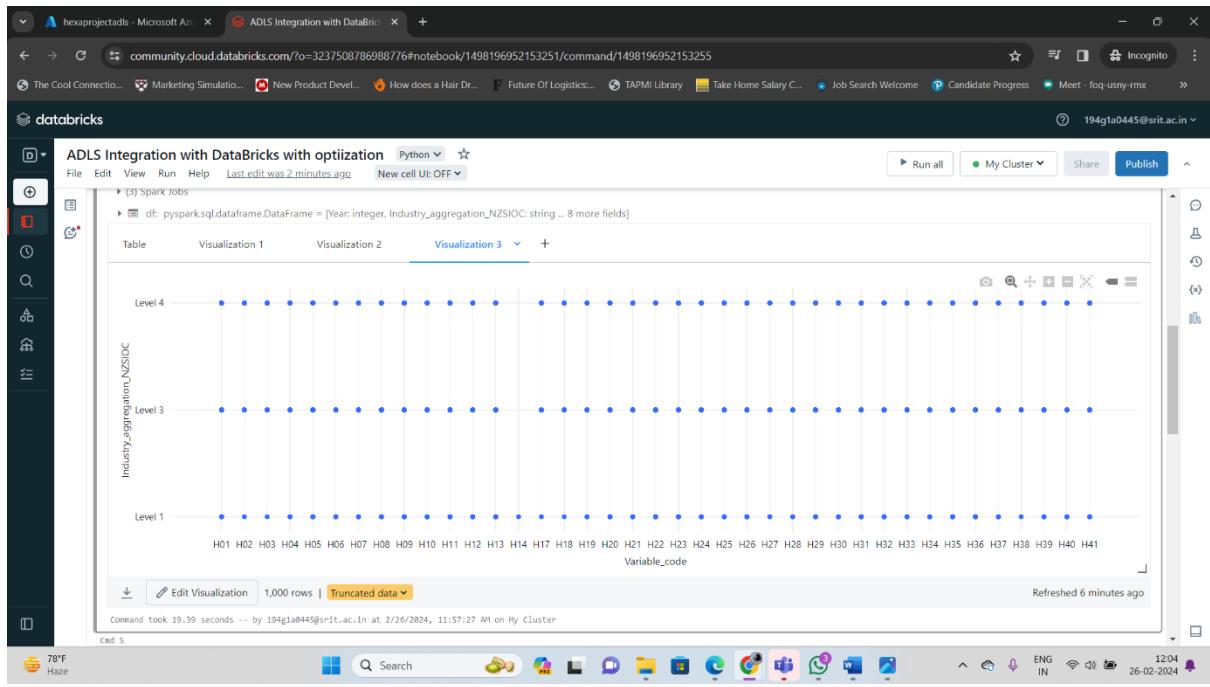
Command took 19.39 seconds -- by 194g1a0445@srit.ac.in at 2/26/2024, 11:57:27 AM on My Cluster

Cod 5 78°F Haze 12:01 ENG IN 26-02-2024









## 2. Data Optimization:

For the same set of we perform Optimization

### Key Highlights:

- 1) Delta table Optimization
  - Partitioning
- 2) File Management Optimization
  - Compacting
  - Z-ordering
- 3) Data Skipping
- 4) Caching

### ➤ Partitioning

Partitioning is a crucial technique for optimizing data lake performance and managing large datasets effectively. Here's a breakdown of its role and considerations:

## What is data lake partitioning?

Data lake partitioning involves dividing your data into smaller, manageable segments based on specific criteria called **partition keys**. These keys could be any relevant attribute like:

- **Time:** Year, month, day, hour, etc. (e.g., data partitioned by year)
- **Location:** Country, state, city, etc. (e.g., data partitioned by country)
- **Category:** Product type, user type, etc. (e.g., data partitioned by product type)

## Benefits of partitioning:

- **Faster query performance:** When querying the data lake, the query engine only needs to scan the relevant partitions based on the query filters. This significantly reduces the amount of data scanned, leading to faster query execution times.
- **Improved scalability:** As your data volume grows, partitioning allows you to add new data efficiently by creating new partitions. This makes scaling your data lake more manageable.
- **Simplified data management:** Partitioning simplifies tasks like data deletion, archiving, and backup, as you can focus on specific partitions based on their keys.

## When is partitioning not recommended?

- **Small data lakes:** For datasets smaller than a terabyte, the overhead of managing partitions might outweigh the performance benefits.
- **Unpredictable access patterns:** If your access patterns are unpredictable and don't follow a specific key, partitioning might not be as effective.

## ➤ File Management Optimization:

- **Compacting:**

This process combines smaller data files into larger ones, which can improve performance by:

- **Reducing the number of files:** Fewer files mean fewer directory entries to scan and less metadata overhead.
- **Improving sequential reads:** Larger files enable faster data reads by minimizing seek times on storage devices.

- **Z-ordering:**

This technique physically re-orders data files based on a chosen attribute (Z-key), ensuring data related to specific queries becomes physically closer on storage. This can significantly speed up queries that frequently access specific values of the Z-key.

## ➤ Data Skipping:

This optimization helps avoid processing irrelevant data during analytics. It involves techniques like predicate pushdown, which allows the query engine to filter data on the source side (e.g., within the storage layer) before transferring it for processing. This reduces network transfer and processing overhead.

## ➤ Caching:

Caching stores frequently accessed data in a readily available location, like memory or a dedicated caching layer. This

significantly reduces the time to retrieve data for subsequent queries that access the same information, leading to faster query responses.

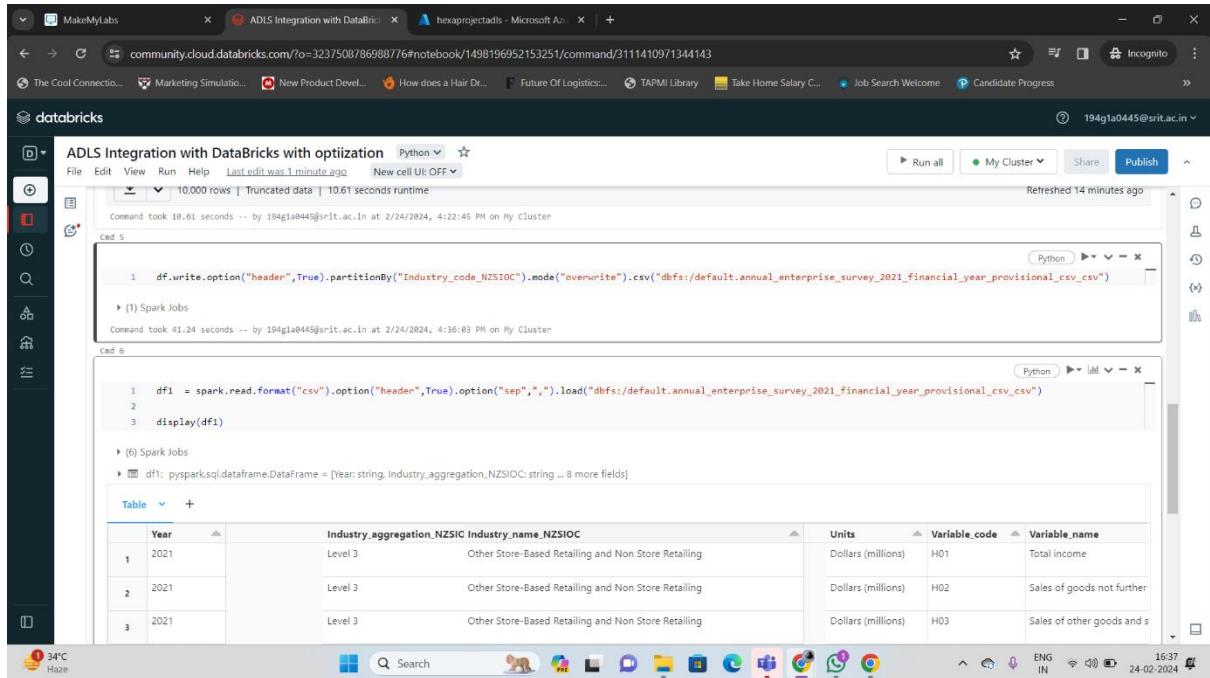
These techniques, along with partitioning, work together to create a well-optimized data lake that facilitates efficient data management and faster analytics.

Here's a summary table for easier reference:

Technique	Description	Benefits
<b>Partitioning</b>	Dividing data based on specific criteria	Faster queries, improved scalability, simplified data management
<b>Compacting</b>	Combining smaller files into larger ones	Reduced overhead, faster sequential reads
<b>Z-ordering</b>	Physically re-ordering files based on a chosen attribute	Faster queries accessing specific values of the Z-key
<b>Data Skipping</b>	Filtering data at the source based on query filters	Reduced network transfer and processing overhead
<b>Caching</b>	Storing frequently accessed data in a readily available location	Faster query response times

In this project we have used Partitioning

We are doing Partitioning by column with name  
“Industry\_code\_NZSIOC”



The screenshot shows a Databricks notebook titled "ADLS Integration with DataBrics with optimization". The notebook contains the following Python code:

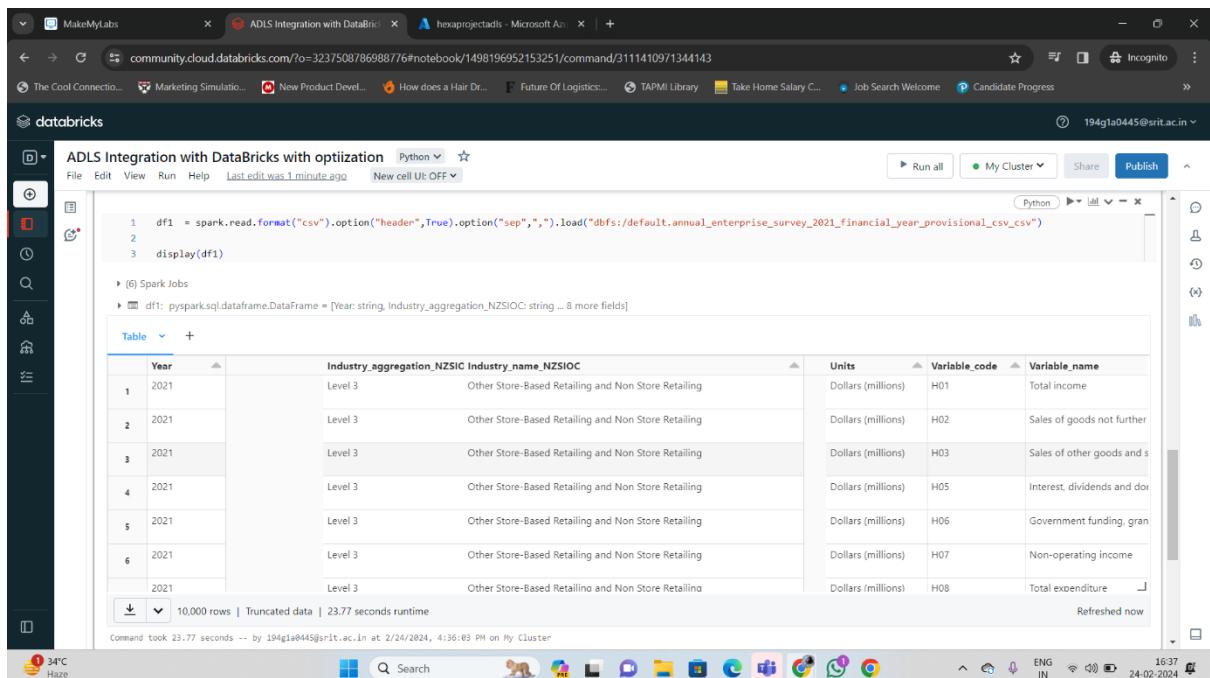
```
1 df.write.option("header",True).partitionBy("Industry_code_NZSIOC").mode("overwrite").csv("dbfs:/default.annual_enterprise_survey_2021_financial_year_provisional_csv_csv")
```

Output from the first cell shows the command took 10.61 seconds. The second cell contains:

```
1 df1 = spark.read.format("csv").option("header",True).option("sep",",").load("dbfs:/default.annual_enterprise_survey_2021_financial_year_provisional_csv_csv")
```

Output from the second cell shows the DataFrame df1 with 10,000 rows. The table has columns: Year, Industry\_aggregation\_NZSIC, Industry\_name\_NZSIOC, Units, Variable\_code, and Variable\_name. The data includes rows for 2021 with various industry categories and their corresponding units and codes.

The Column has been removed and we got Optimize Data



The screenshot shows a Databricks notebook titled "ADLS Integration with DataBrics with optimization". The notebook contains the same Python code as the previous screenshot:

```
1 df1 = spark.read.format("csv").option("header",True).option("sep",",").load("dbfs:/default.annual_enterprise_survey_2021_financial_year_provisional_csv_csv")
```

Output from the first cell shows the command took 23.77 seconds. The second cell contains:

```
1 df1 = spark.read.format("csv").option("header",True).option("sep",",").load("dbfs:/default.annual_enterprise_survey_2021_financial_year_provisional_csv_csv")
```

Output from the second cell shows the DataFrame df1 with 10,000 rows. The table has columns: Year, Industry\_aggregation\_NZSIC, Industry\_name\_NZSIOC, Units, Variable\_code, and Variable\_name. The data includes rows for 2021 with various industry categories and their corresponding units and codes, similar to the previous screenshot.

## Spark (Only PySpark and SQL)

- Spark architecture, Data Sources API, and Dataframe API.
- PySpark - Ingested CSV, simple, and complex JSON files into the data lake as parquet files/ tables.
- PySpark - Transformations such as Filter, Join, Simple Aggregations, GroupBy, Window functions etc.
- PySpark - Created global and temporary views.
- Spark SQL - Created databases, tables, and views.
- Spark SQL - Transformations such as Filter, Join, Simple Aggregations, GroupBy, Window functions etc.
- Spark SQL - Created local and temporary views.
- Implemented full refresh and incremental load patterns using partitions.

## Delta Lake

- Performed Read, Write, Update, Delete, and Merge to delta lake using both PySpark as well as SQL.
- History, Time Travel, and Vacuum.
- Converted Parquet files to Delta files.
- Implemented incremental load pattern using delta lake.

## Azure Data Factory

- Created pipelines to execute Databricks notebooks.
- Designed robust pipelines to deal with unexpected scenarios such as missing files.
- Created dependencies between activities as well as pipelines.
- Scheduled the pipelines using data factory triggers to execute at regular intervals.
- Monitored the triggers/ pipelines to check for errors/ outputs.

## Technologies/Tools Used:

- Pyspark
- Spark SQL
- Delta Lake
- Azure Databricks
- Azure Data Factory
- Azure Date Lake Storage Gen2
- Azure Key vault
- Power BI(Optional)

## **REFERENCE LINKS:**

<https://learn.microsoft.com/en-us/azure/architecture/example-scenario/data/synapse-exploratory-data-analytics>

<https://learn.microsoft.com/en-us/azure/databricks/delta/optimize>