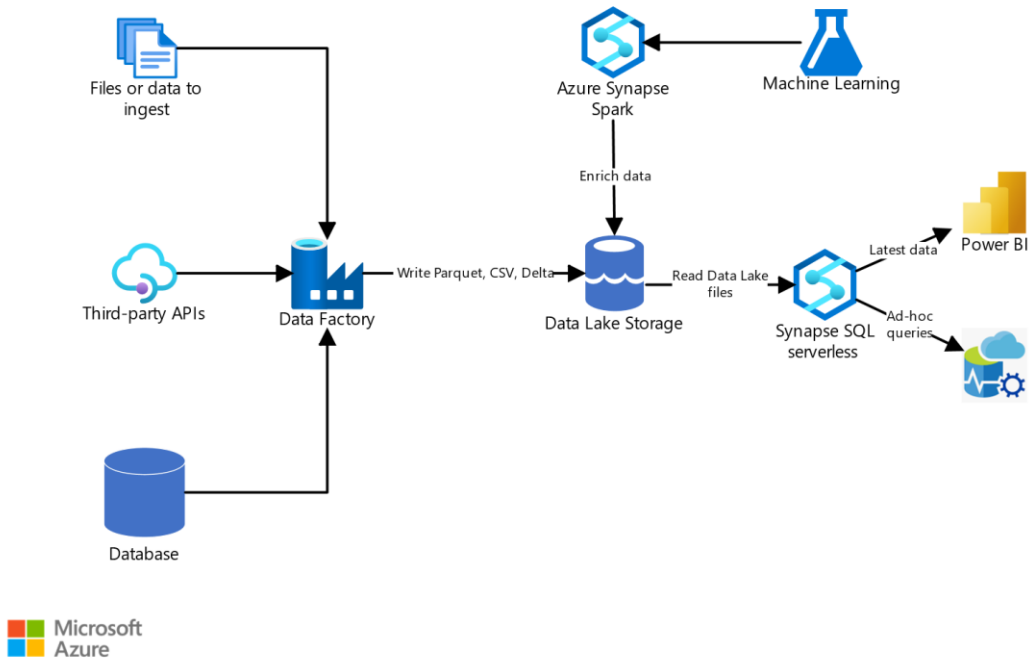
Data-Engineering-Project-Using-Azure-Databricks

Data Lake Exploration and Optimization

Project Overview:

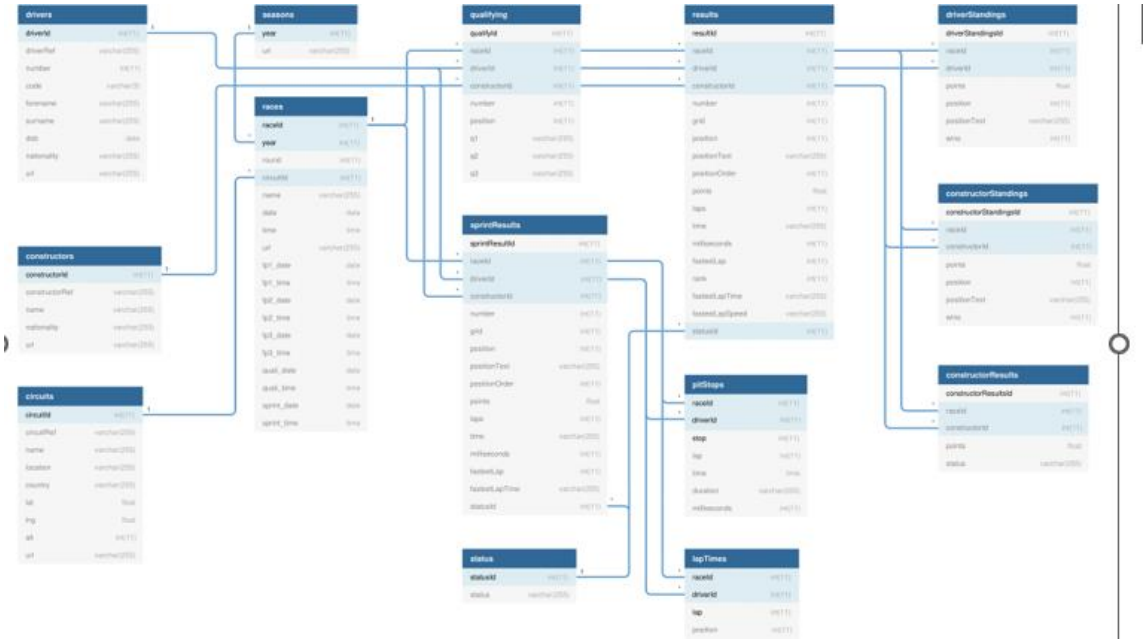
Objective: This project aims to explore and optimize data stored in an Azure Data Lake using PySparkSQL on the Azure Databricks platform. By leveraging PySparkSQL's functionalities, you'll gain insights into the data and improve its efficiency for further analysis or downstream applications.

Architecture diagram:



ER Diagram:

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



Components:

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

Data Flow:

1. **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:

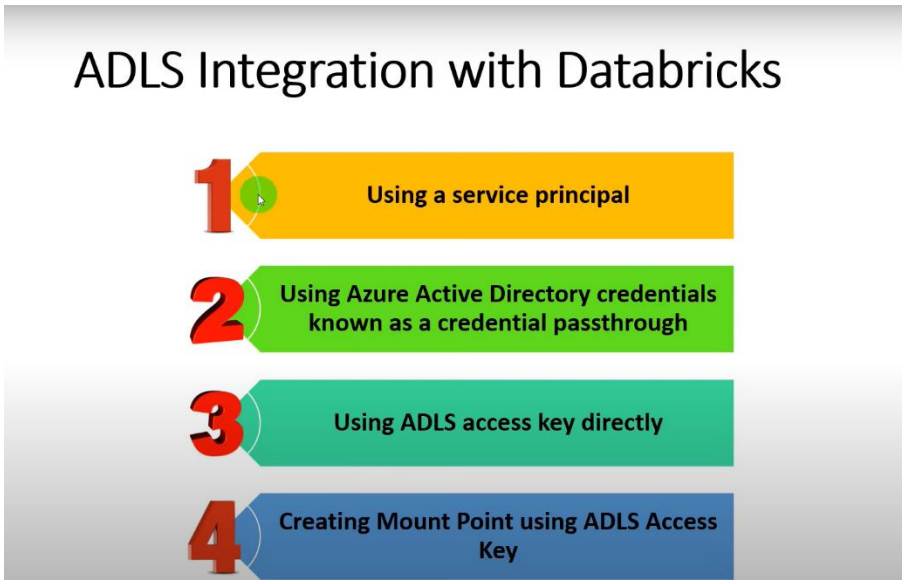
- **Insights:** Define the specific insights you want to extract from the data. This might involve analyzing trends, relationships between variables, or identifying patterns within the data.
- **Reporting:** Specify how you want to report the findings. This could include generating visualizations using 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:

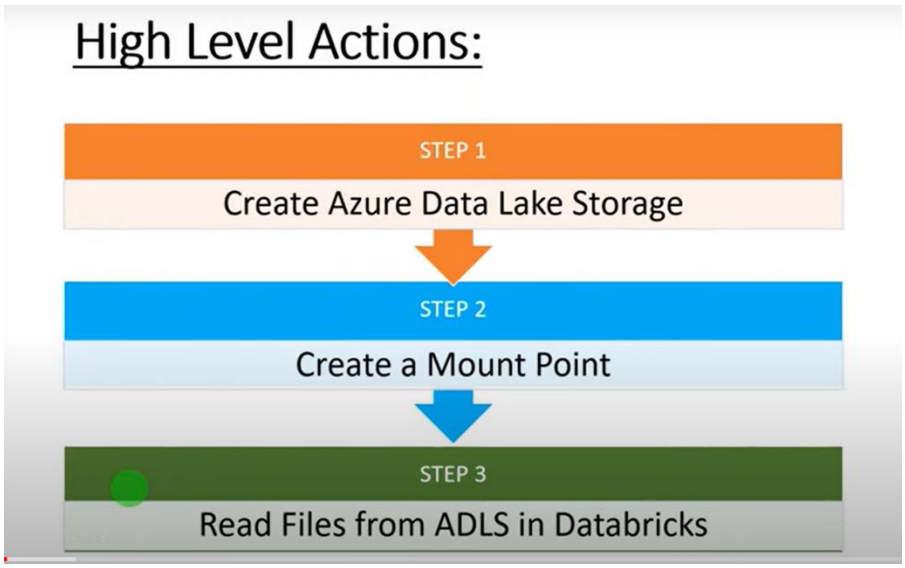
- **Performance:** Specify any performance expectations for data processing and query execution times.
- **Scalability:** Define if the project needs to handle growing data volumes in the future, requiring adjustments to cluster size or optimization techniques.

- **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.

Four Methods to Integrate ADLS with Databricks



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

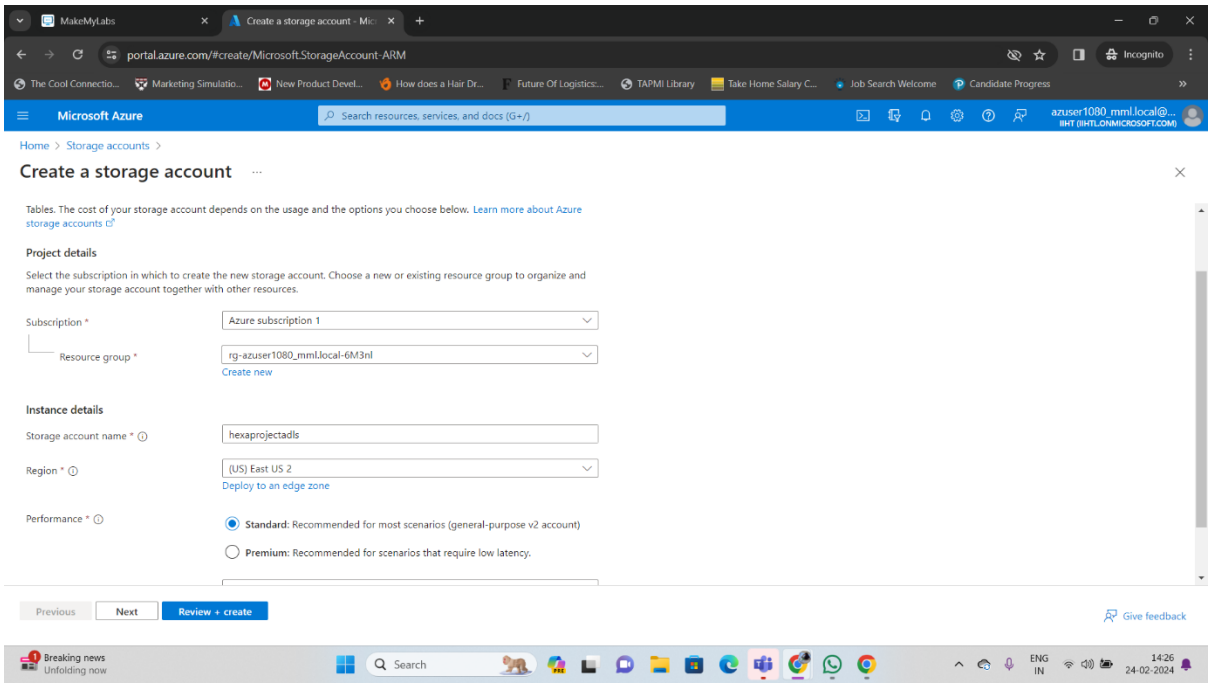


Tasks performed:

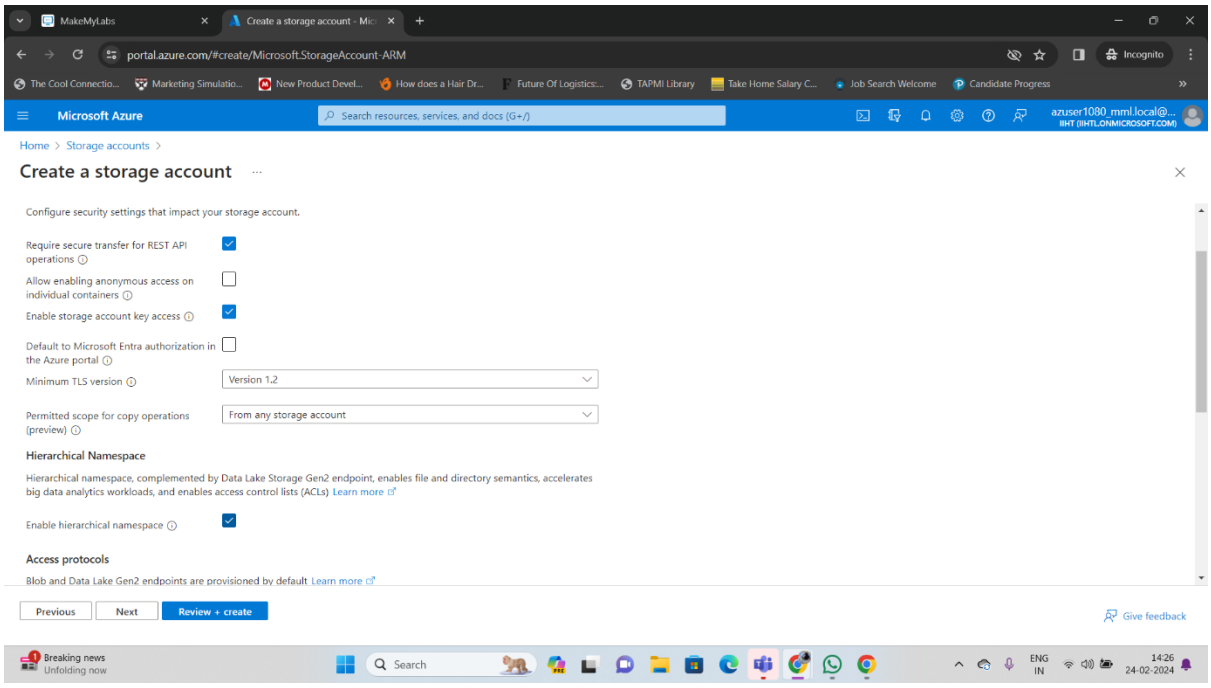
1. Data Exploration:

Login in to Azure Portal and create a Storage Account

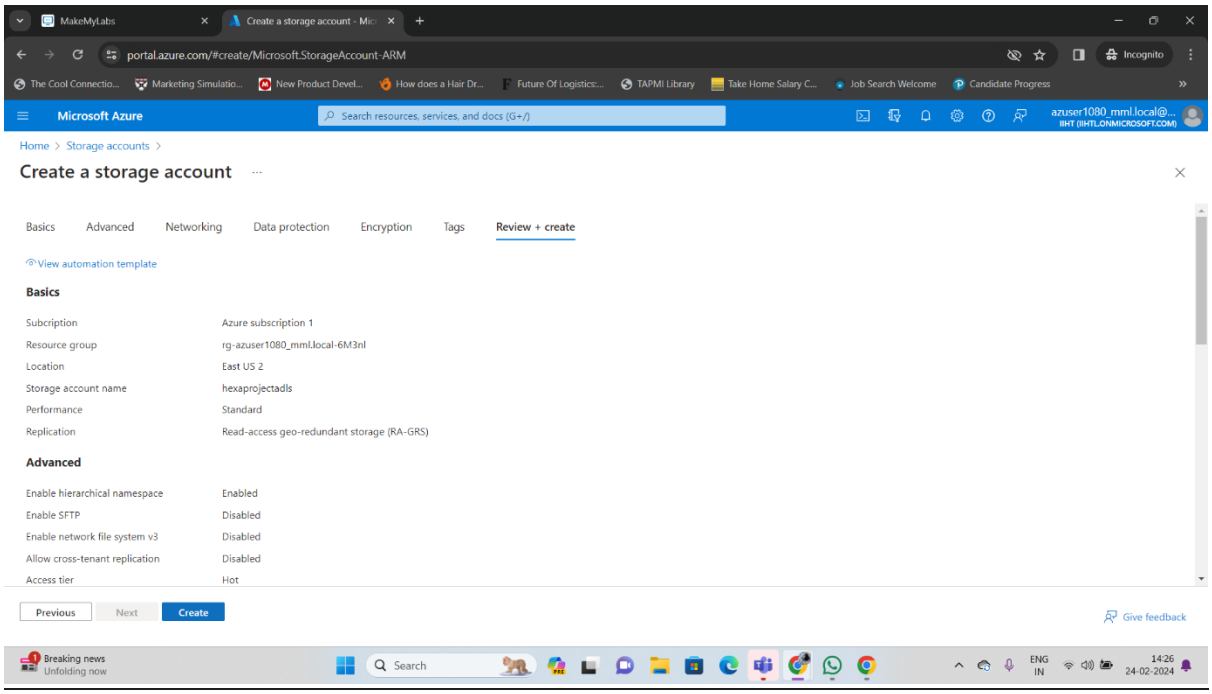
Name “hexaprojectadls”



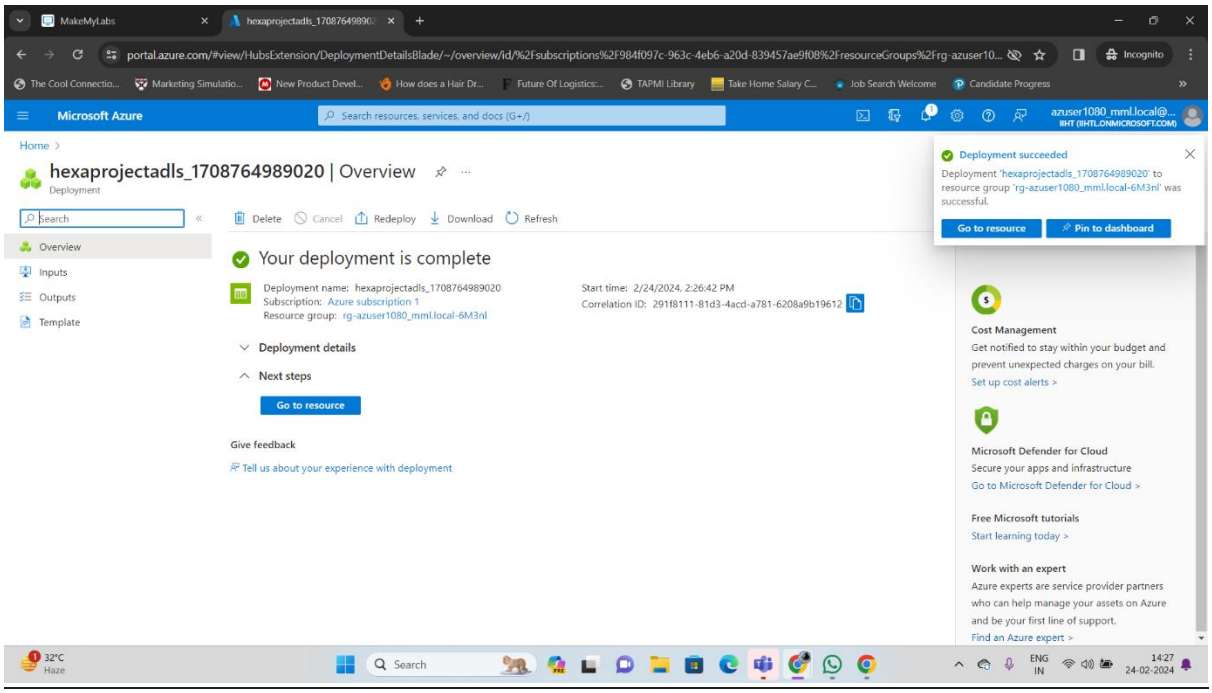
Enabling Hierarchical Namespace



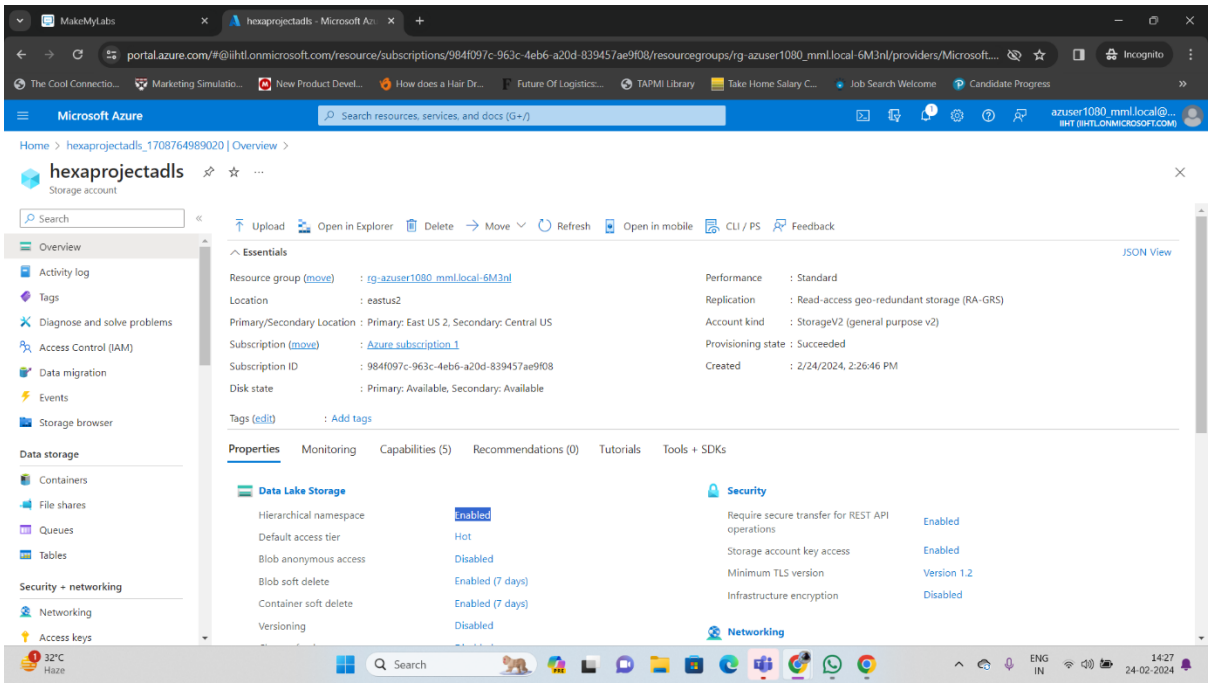
Review & and Create Storage Account



Storage Account Created

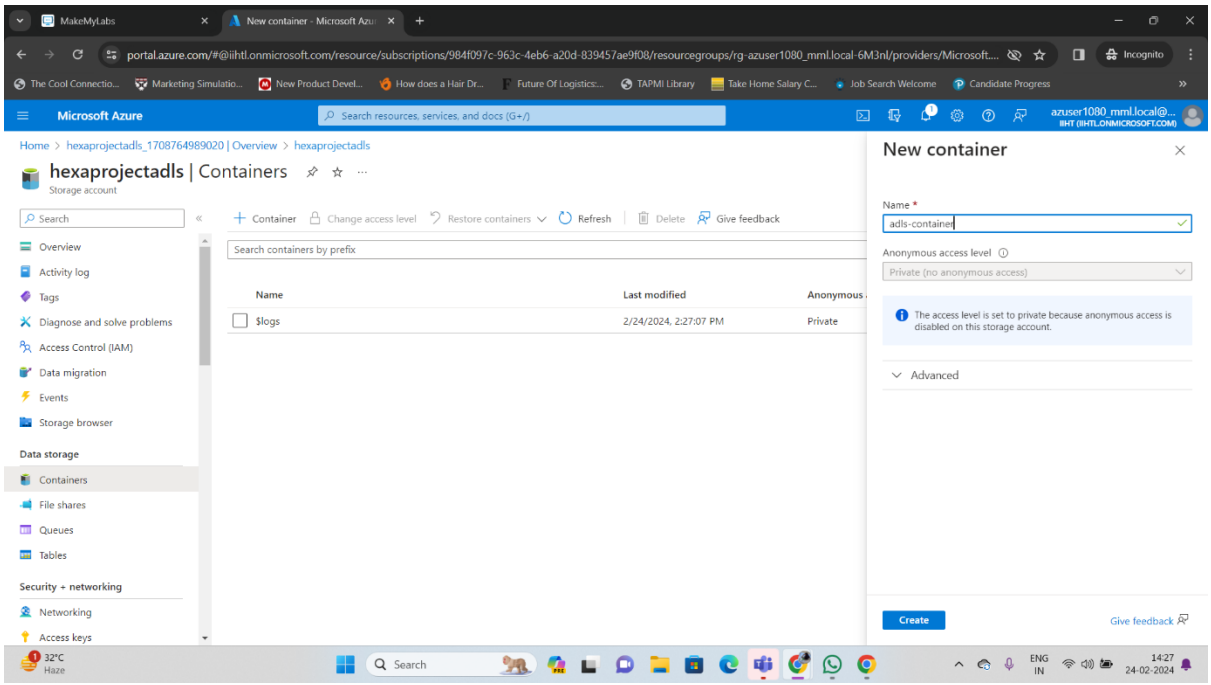


Click Go to Resource

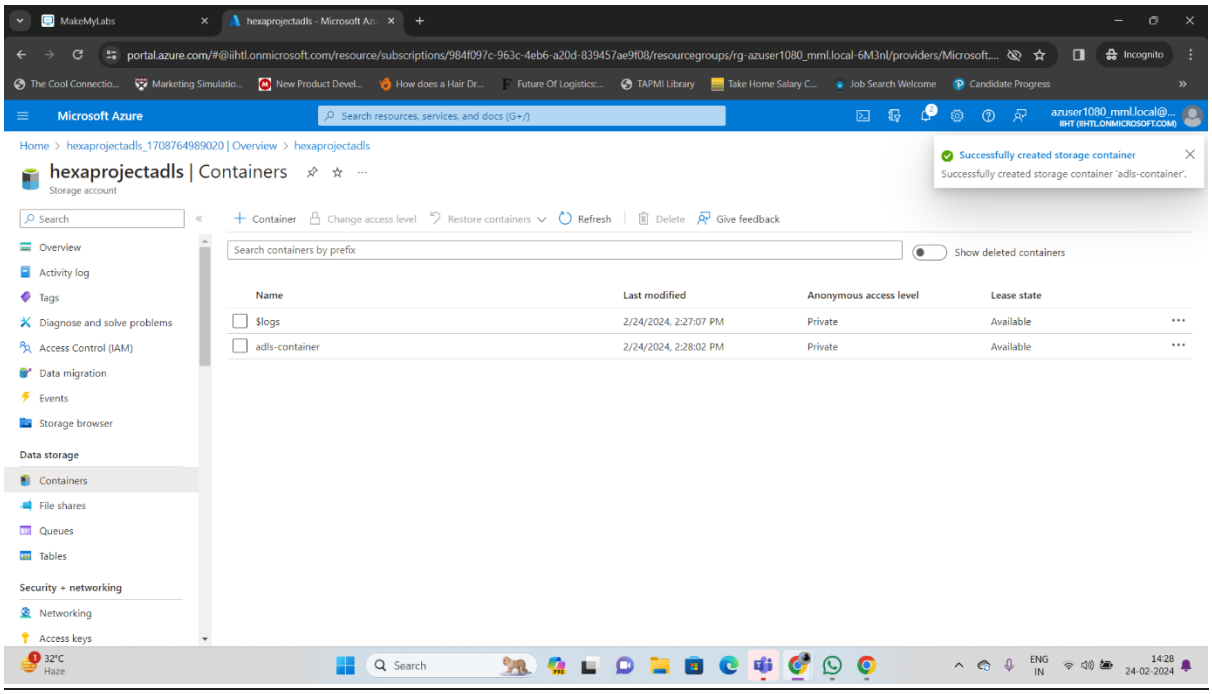


Create Container

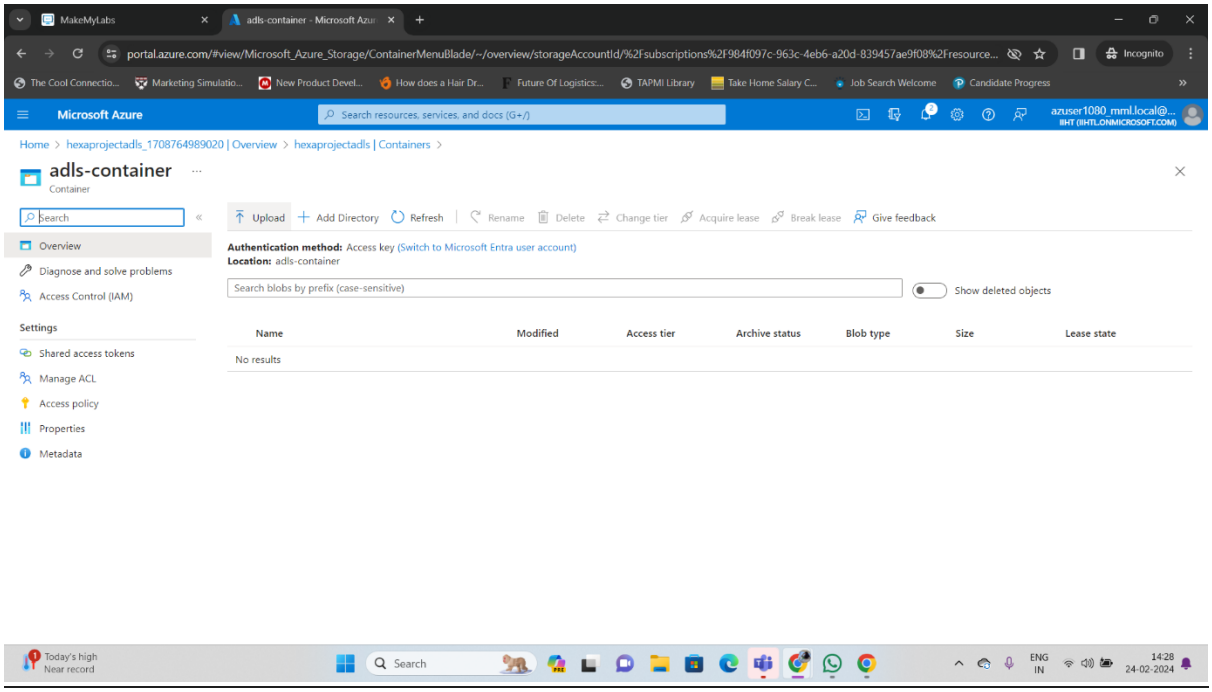
Name adls-container

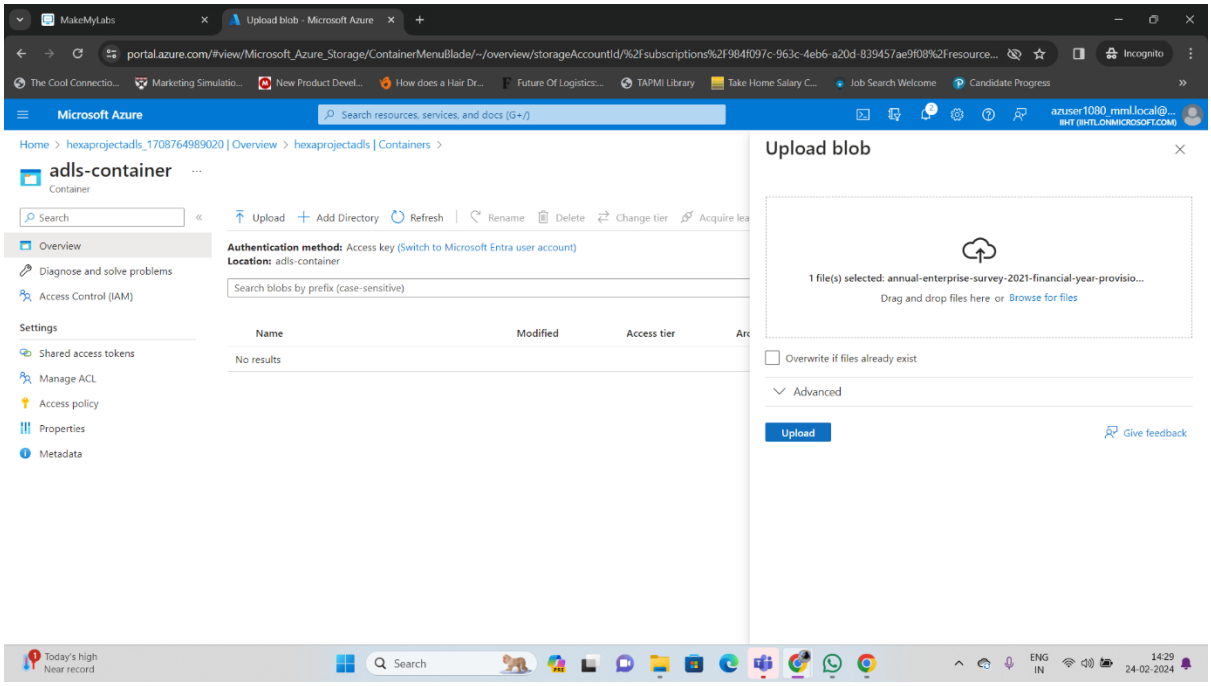


Container Created

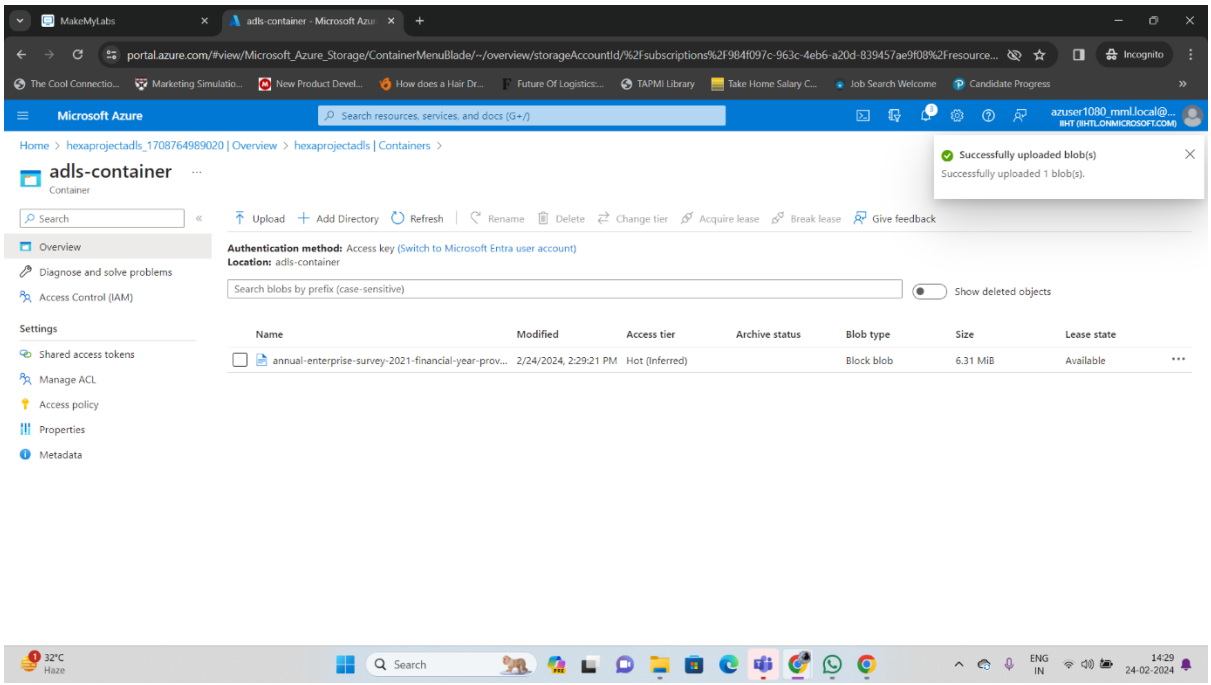


Upload a CSV file in the Container

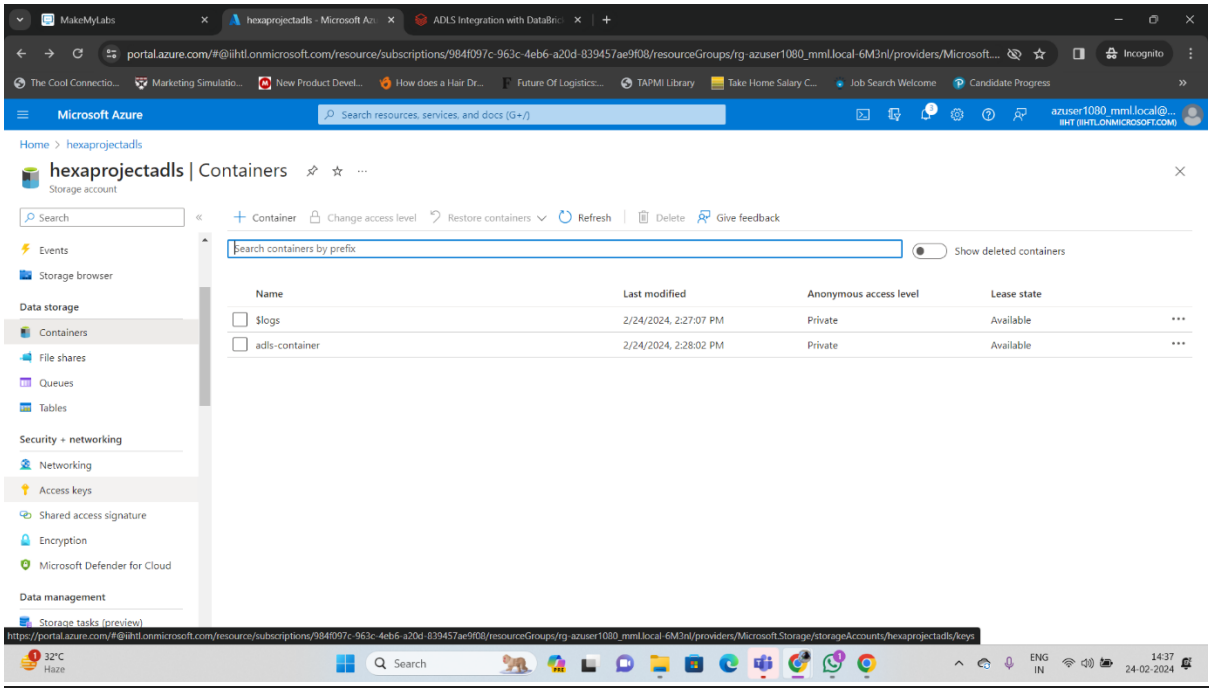




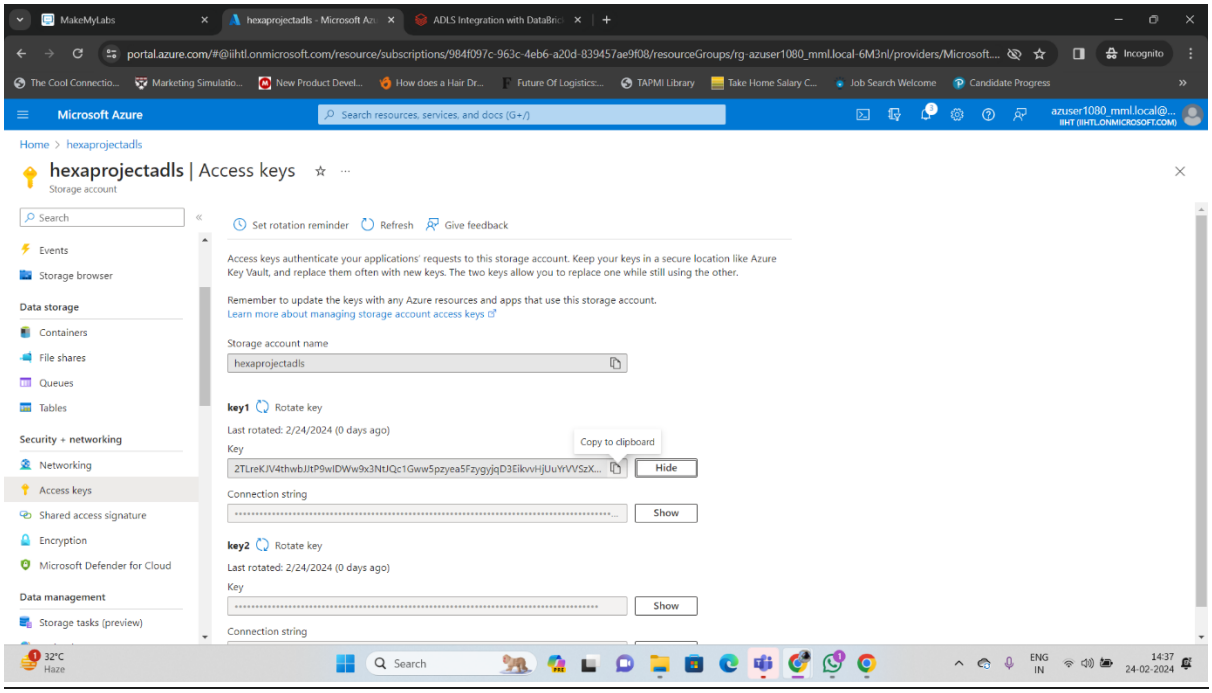
File Uploaded



Now go to Storage Account get Access key

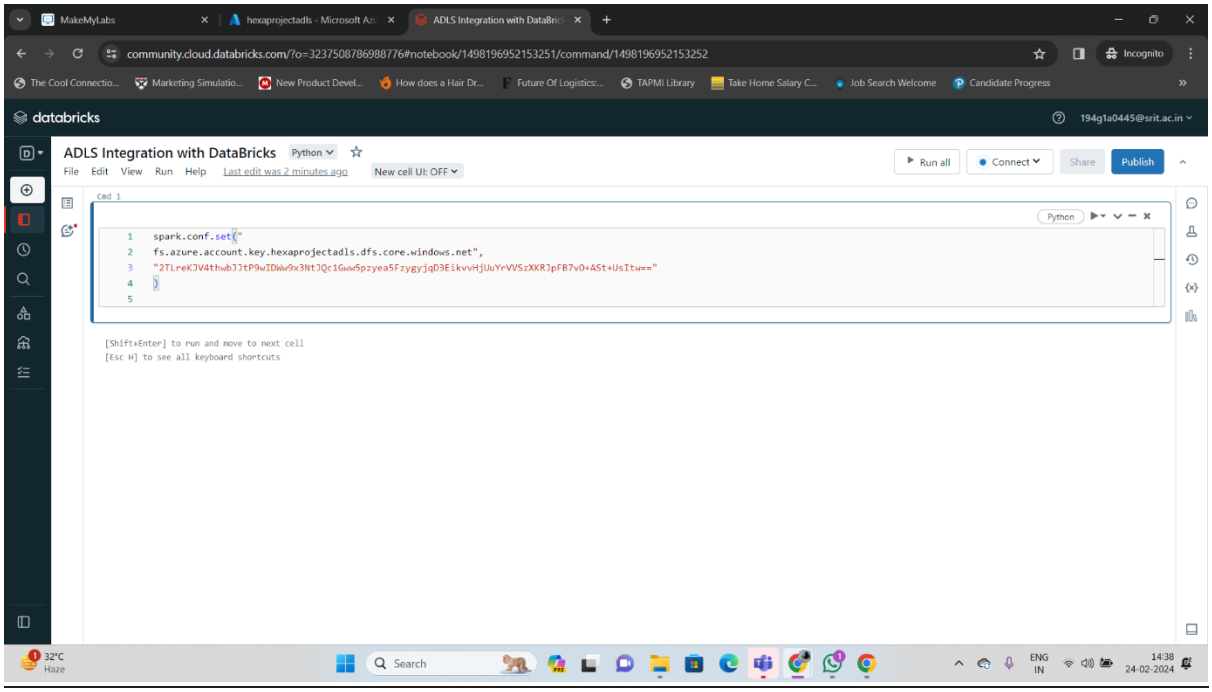


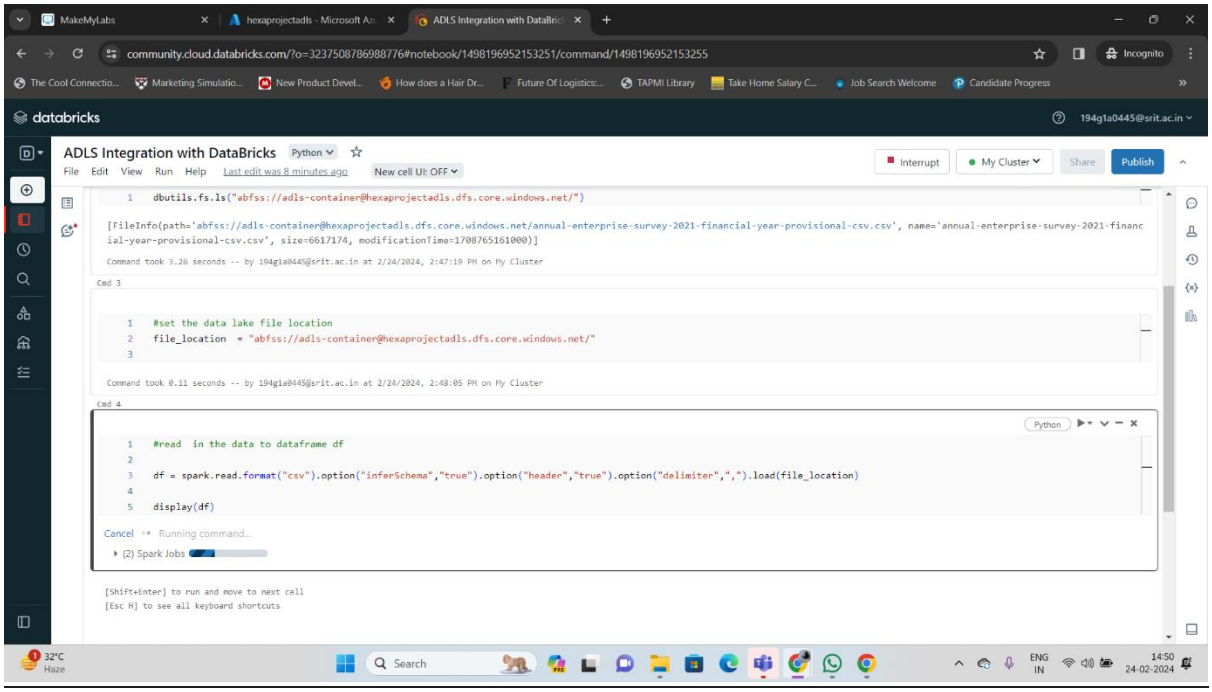
Copy Access Key



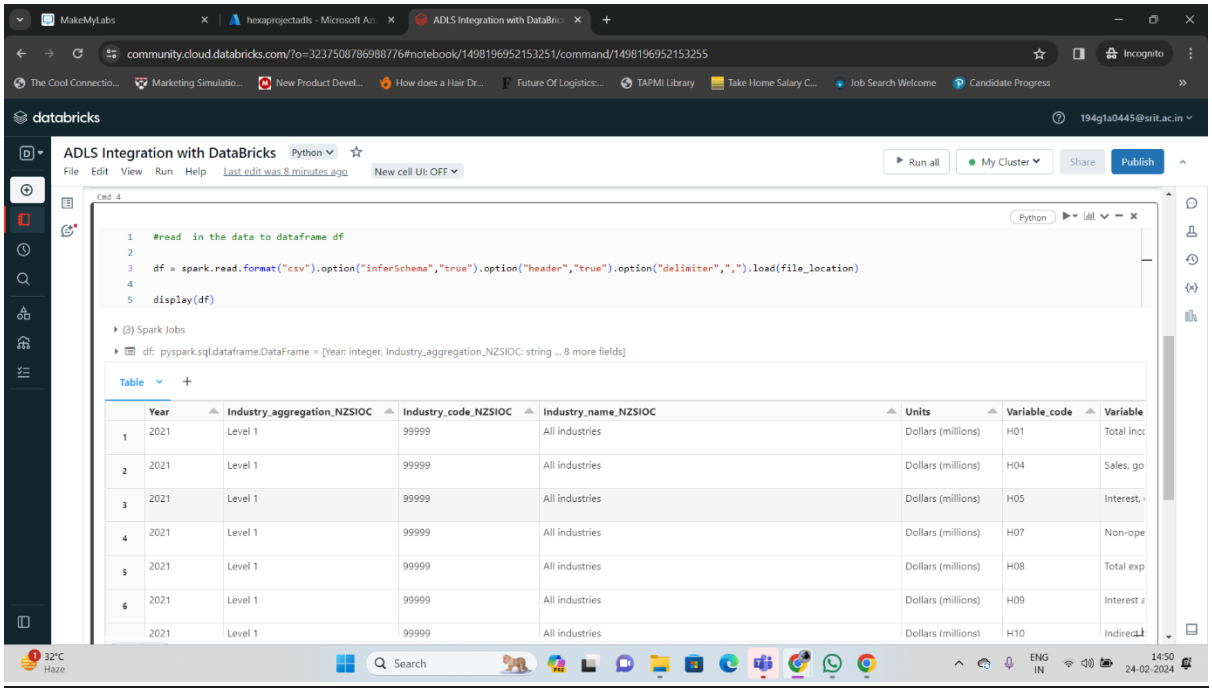
Go to Databricks Portal

Run PySpark command by giving Storage account name and access key

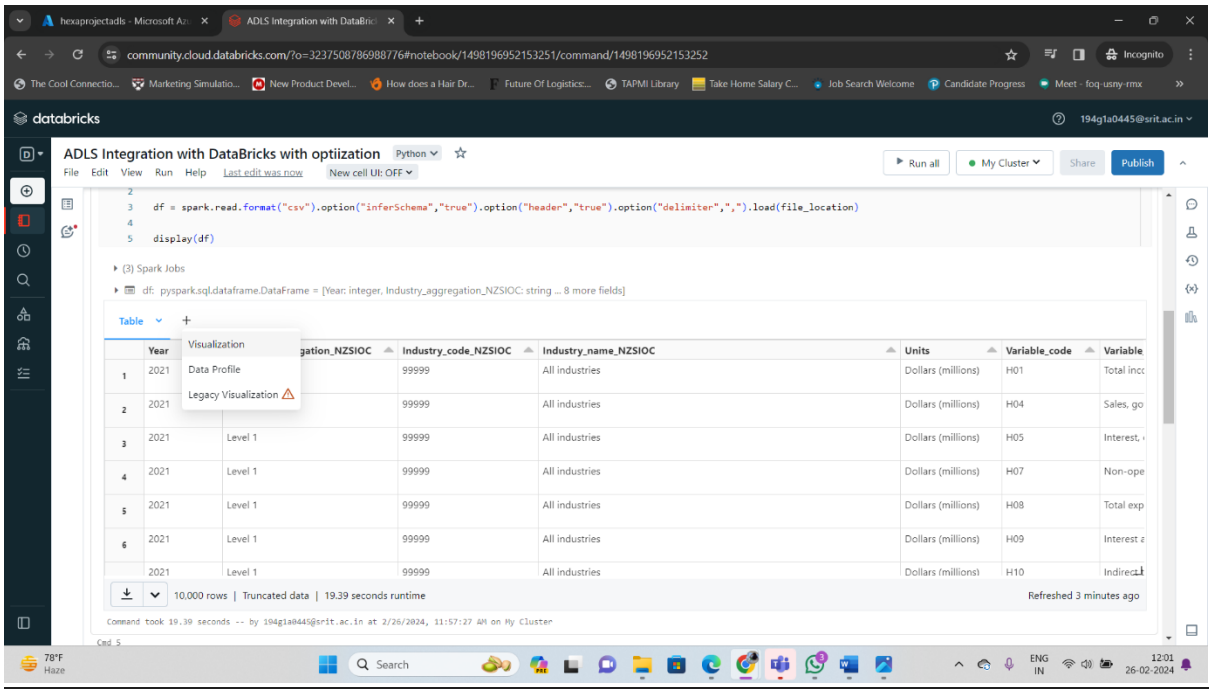


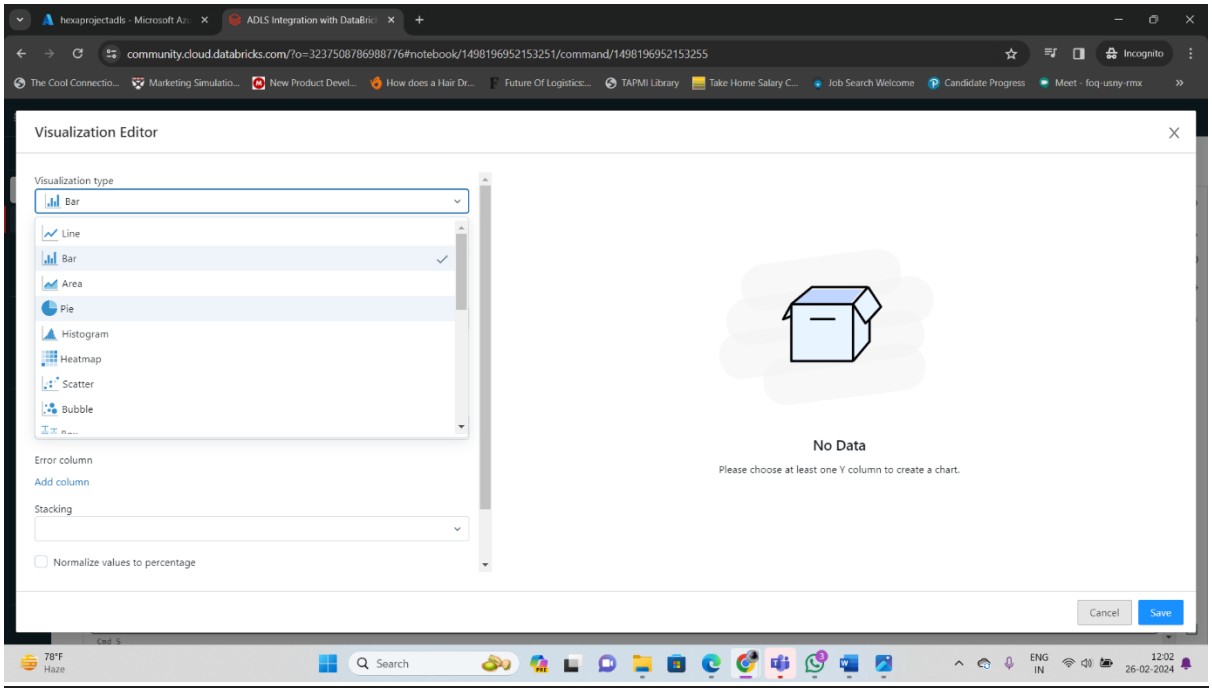


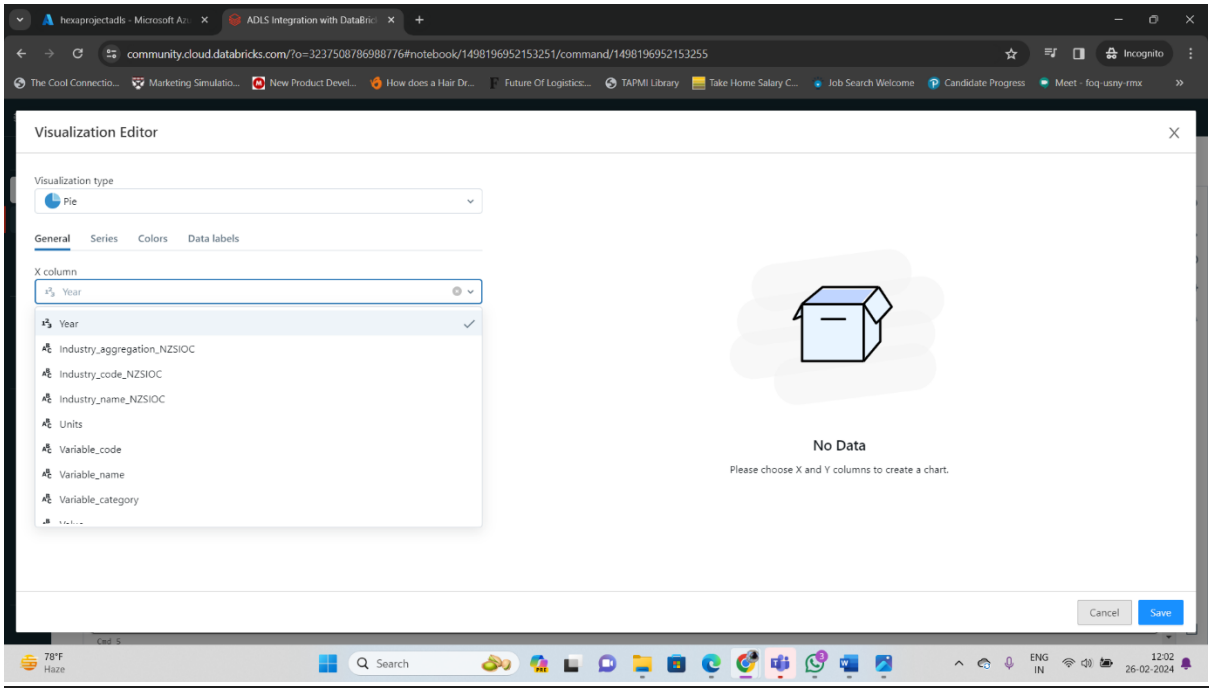
To read data to Dataframe [Df] & Display

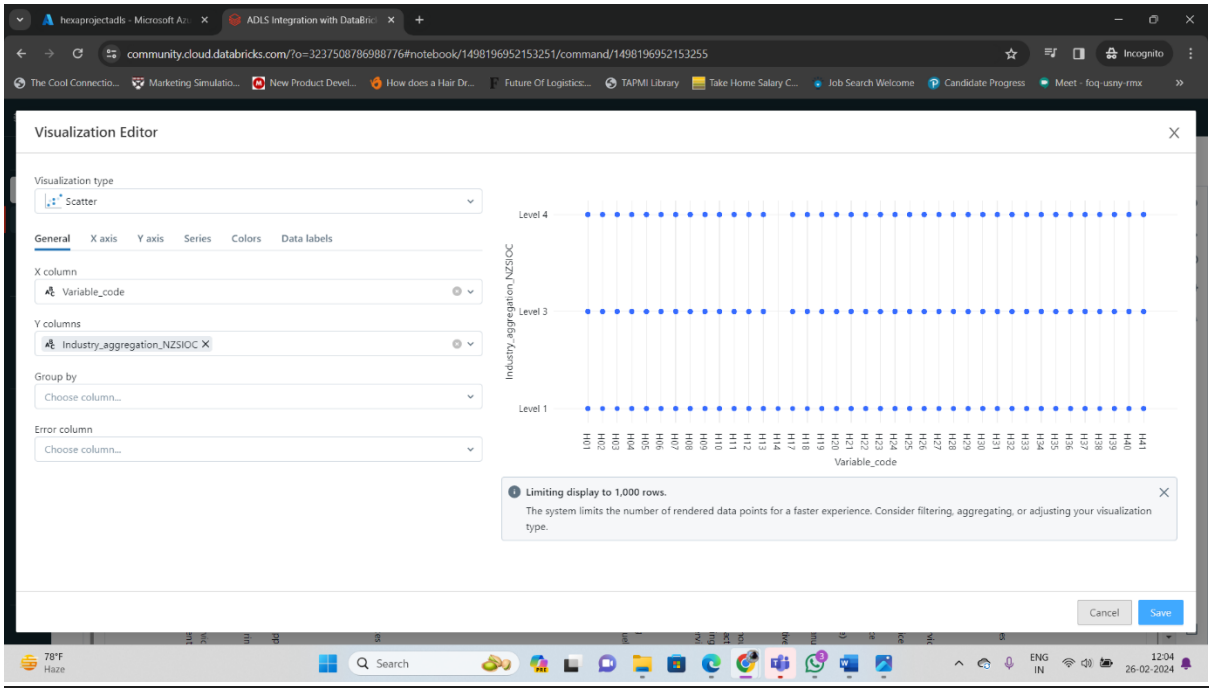


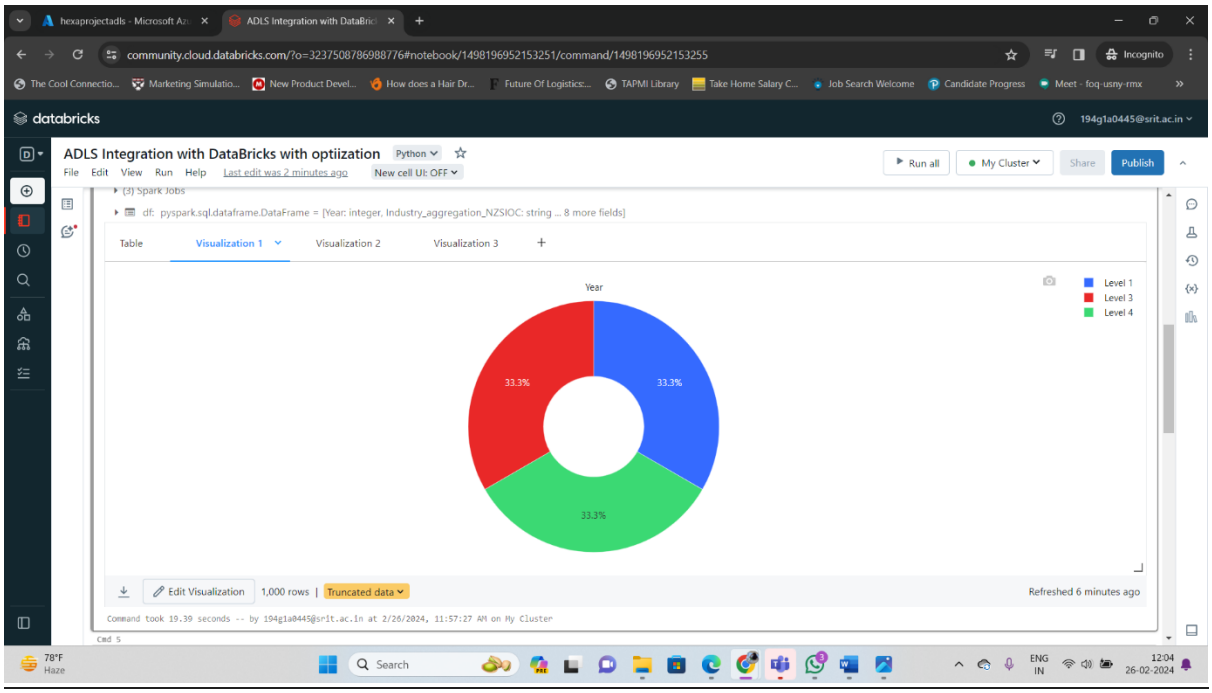
Perform Visualization in order to Explore Data

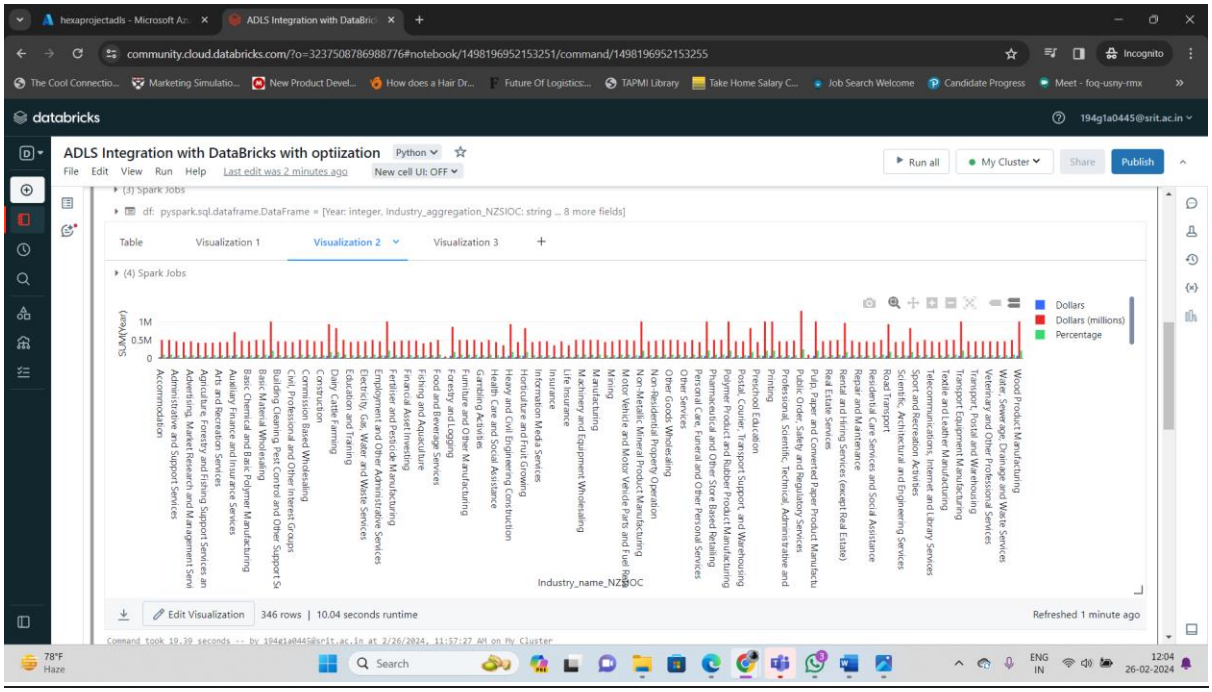


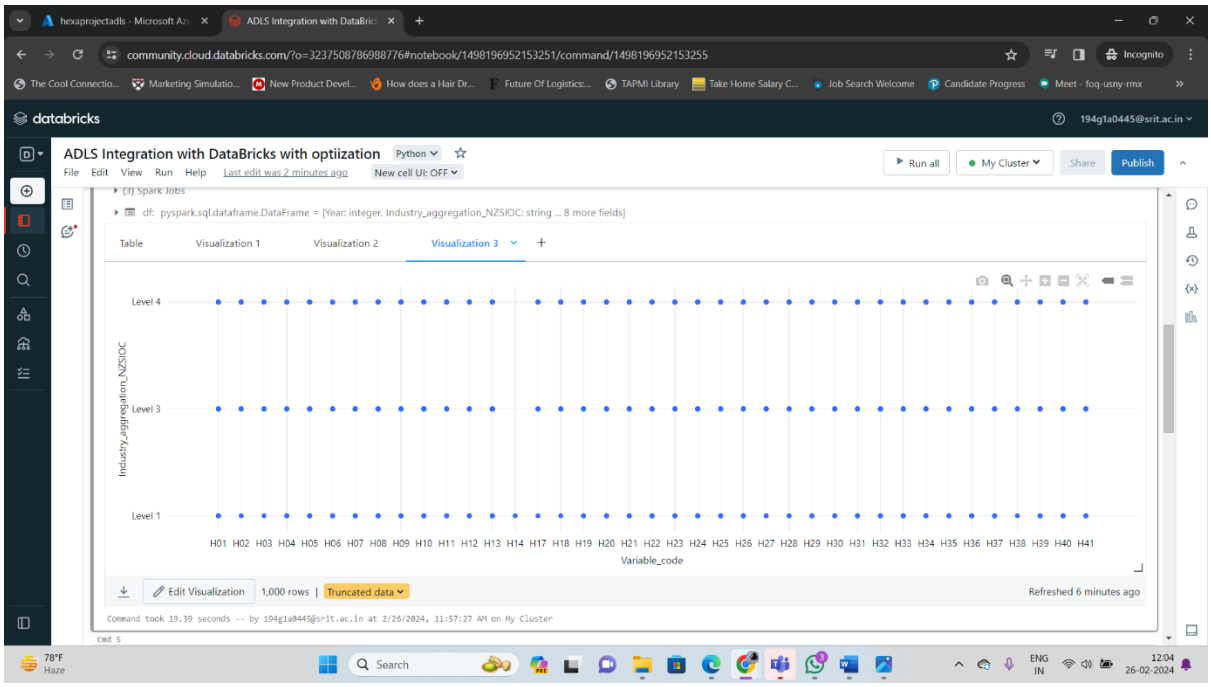

Visualization Editor
Visualization type
Bar
Line
Bar
Area
Pie
Histogram
Heatmap
Scatter
Bubble
Error column
Add column
Stacking
Normalize values to percentage
No Data
Please choose at least one Y column to create a chart.
Cancel
Save


Visualization Editor
Visualization type
Pie
General
Series
Colors
Data labels
X column
Year
Industry_aggregation_NZSIOC
Industry_code_NZSIOC
Industry_name_NZSIOC
Units
Variable_code
Variable_name
Variable_category
No Data
Please choose X and Y columns to create a chart.
Cancel
Save


Visualization Editor
Visualization type
Scatter
General
X axis
Y axis
Series
Colors
Data labels
X column
Variable_code
Y columns
Industry_aggregation_NZSIOC
Group by
Choose column...
Error column
Choose column...
Level 4
Level 3
Level 1
Industry_aggregation_NZSIOC
Variable_code
Limiting display to 1,000 rows.
The system limits the number of rendered data points for a faster experience. Consider filtering, aggregating, or adjusting your visualization type.
Cancel
Save


ADLS Integration with DataBricks with optiization
Python
File Edit View Run Help
Run all
My Cluster
Share
Publish
Table
Visualization 1
Visualization 2
Visualization 3
Year
Level 1
Level 3
Level 4
33.3%
33.3%
33.3%
Edit Visualization
1,000 rows | Truncated data
Refreshed 6 minutes ago


ADLS Integration with DataBricks with optiization
Python
File Edit View Run Help Last edit was 2 minutes ago
New cell UI: OFF
Run all
My Cluster
Share
Publish
(3) Spark Jobs
df: pyspark.sql.dataframe.DataFrame = [Year: integer, Industry_aggregation_NZSIOC: string ... 8 more fields]
Table
Visualization 1
Visualization 2
Visualization 3
(4) Spark Jobs
SUM(Year)
1M
0.5M
0
Dollars
Dollars (millions)
Percentage
Industry_name_NZSIOC
Edit Visualization
346 rows | 10.04 seconds runtime
Refreshed 1 minute ago
Command took 19.39 seconds -- by 194g1a0445@srit.ac.in at 2/26/2024, 11:57:27 AM on My Cluster


ADLS Integration with DataBricks with optiization
Python
File Edit View Run Help Last edit was 2 minutes ago
New cell UI: OFF
Run all
My Cluster
Share
Publish
(3) Spark Jobs
df: pyspark.sql.dataframe.DataFrame = [Year: integer, Industry_aggregation_NZSIOC: string ... 8 more fields]
Table
Visualization 1
Visualization 2
Visualization 3
Industry_aggregation_NZSIOC
Level 4
Level 3
Level 1
H01 H02 H03 H04 H05 H06 H07 H08 H09 H10 H11 H12 H13 H14 H17 H18 H19 H20 H21 H22 H23 H24 H25 H26 H27 H28 H29 H30 H31 H32 H33 H34 H35 H36 H37 H38 H39 H40 H41
Variable_code
Edit Visualization
1,000 rows | Truncated data
Refreshed 6 minutes ago
Command took 19.39 seconds -- by 194g1a0445@srit.ac.in at 2/26/2024, 11:57:27 AM on My Cluster

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.

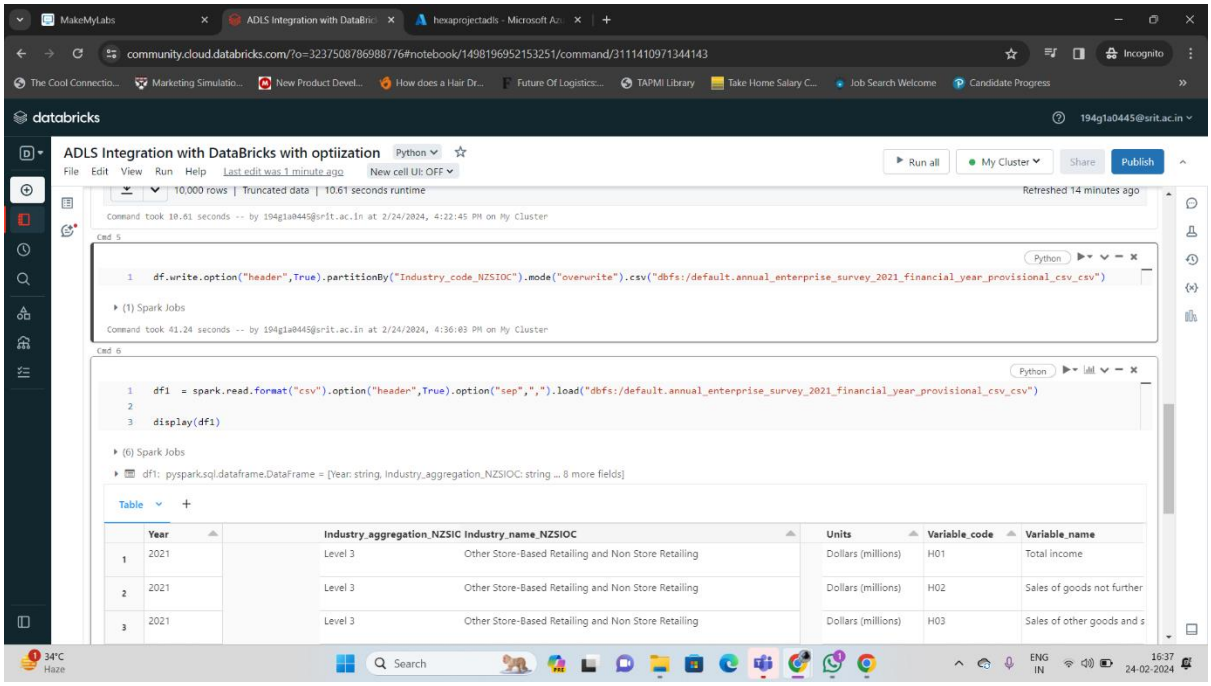
  These techniques, along with partitioning, work together to create a well-optimized data lake that facilitates efficient data management and faster analytics.

Here's a summary table for easier reference:

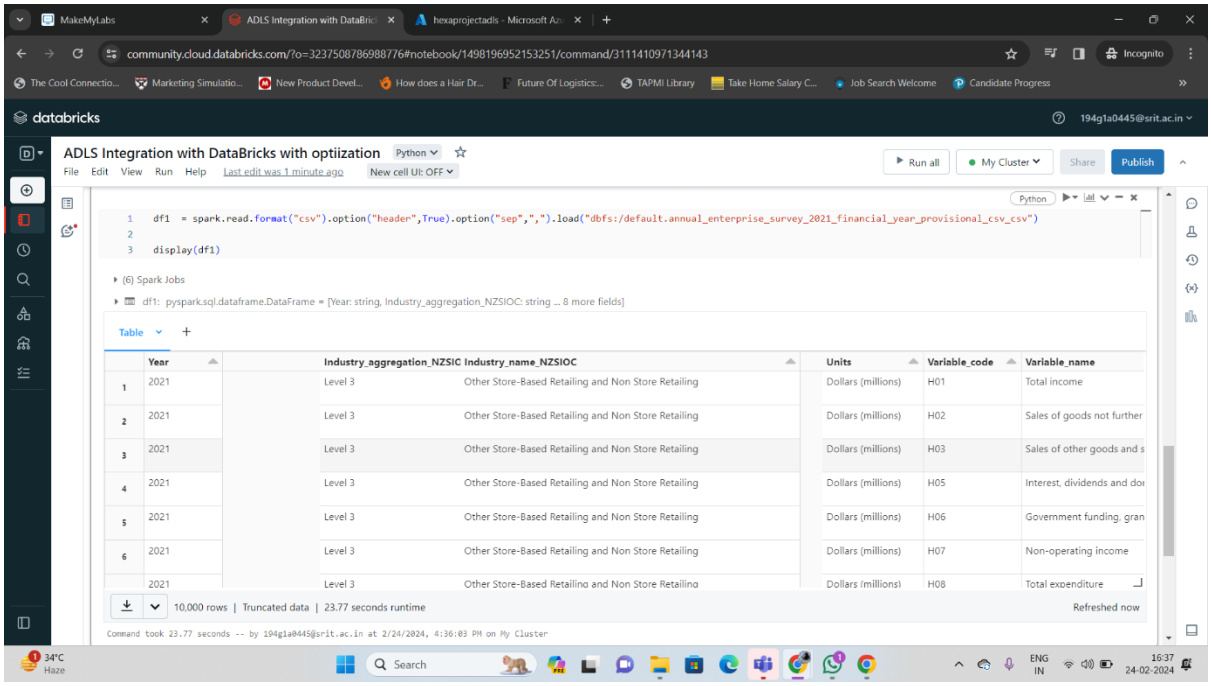
Technique	Description	Benefits
Partitioning	Dividing data based on specific criteria	Faster queries, improved scalability, simplified data management
Compacting	Combining smaller files into larger ones	Reduced overhead, faster sequential reads
Z-ordering	Physically re-ordering files based on a chosen attribute	Faster queries accessing specific values of the Z-key
Data Skipping	Filtering data at the source based on query filters	Reduced network transfer and processing overhead
Caching	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 Column has been removed and we got Optimize Data



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)

REFERCENCE 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