Data quality and ADLS to ADLS pipeline

Introduction:

In today's data-driven world, organisations rely heavily on data to drive strategic decision-making, enhance operational efficiency, and gain competitive advantage. However, the increasing volume, velocity, and variety of data pose significant challenges in ensuring its quality and reliability. Poor data quality can lead to inaccurate insights, faulty analysis, and ultimately, misguided business decisions.

Recognizing the critical importance of data quality, this project focuses on implementing robust data quality assessment techniques and efficient data transfer mechanisms using modern cloud-based technologies. Specifically, the project leverages the capabilities of Databricks, a unified analytics platform powered by Apache Spark, along with Pandas, a versatile data manipulation library in Python, for data quality assessment tasks. Additionally, Azure Data Factory, a cloud-based data integration service, is utilized for seamless data transfer operations between Azure Data Lake Storage (ADLS) instances.

Objectives:

The primary objectives of the project are as follows:

1. Data Quality Assessment: To develop a comprehensive data quality assessment framework utilizing Databricks and Pandas, aimed at ensuring the accuracy, consistency, and completeness of organizational data assets. This involves defining data matrices, filtering out relevant data subsets, identifying and quarantining invalid data, and performing advanced validation techniques.

2. Data Transfer: To establish efficient data transfer pipelines using Azure Data Factory for transferring data between Azure Data Lake Storage (ADLS) instances. This includes configuring pipelines for data replication, backup, or migration while ensuring scalability, reliability, and security of the data transfer process.

Rationale:

- **Data Quality Assurance:** By implementing robust data quality assessment techniques, organizations can enhance the reliability and trustworthiness of their data assets, leading to more accurate insights and informed decision-making processes.
- **Operational Efficiency:** Efficient data transfer mechanisms enable seamless movement of data across different storage environments, facilitating data replication, backup, and migration tasks with minimal manual intervention.
- **Adoption of Cloud Technologies:** Leveraging cloud-based platforms such as Azure provides scalability, flexibility, and cost-effectiveness in managing data quality and transfer operations, thereby empowering organizations to adapt to evolving data challenges.

Scope:

The scope of the project encompasses the following key areas:

- Definition and implementation of data quality assessment processes using Databricks and Pandas, including data filtering, validation, and handling of invalid data.
- Configuration and deployment of data transfer pipelines utilizing Azure Data Factory for transferring data between Azure Data Lake Storage (ADLS) instances.

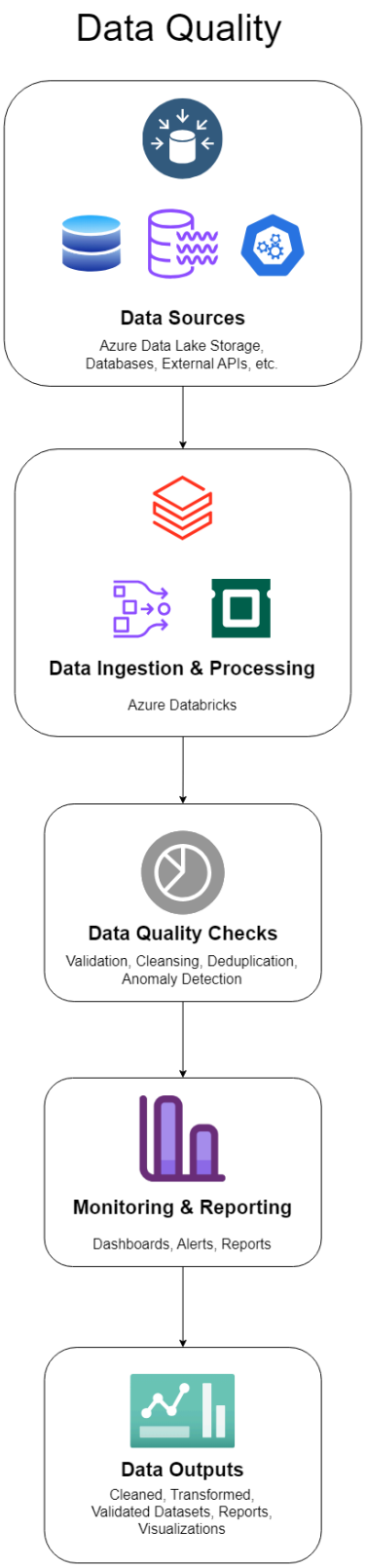
- Evaluation of the effectiveness and efficiency of the implemented solutions in ensuring data quality and facilitating seamless data transfer operations.

Organization of the Report:

The project report is structured as follows:

- **Introduction:** Provides an overview of the project objectives, rationale, and scope.
- **Architectural Diagram:** Represents architecture of data quality.
- **Data Quality Assessment:** Details the methodologies and techniques employed for data quality assessment using Databricks and Pandas.
- **Data Transfer:** Describes the configuration and implementation of data transfer pipelines using Azure Data Factory.
- **Technologies Used:** Technologies that are used to implement data quality and Data Transfer.
- **Results and Evaluation:** Presents the outcomes and assessment of the implemented solutions.
- **Conclusion:** Summarizes the key findings and recommendations for future enhancements.

Architectural Diagram :



How it Works:

1. Data Sources:

- These are the various data repositories or systems from which your project retrieves data.
- Examples include Azure Data Lake Storage, databases, external APIs, or any other data sources relevant to your project.

2. Data Ingestion & Processing (Azure Databricks):

- This component represents the core processing engine for your project, implemented using Azure Databricks.
- Data ingestion involves reading data from the sources into Azure Databricks for further processing.
- Data processing encompasses various transformations, cleansing, enrichment, and aggregation operations performed on the ingested data.

3. Data Quality Checks:

- This component focuses on ensuring the quality and integrity of the processed data.
- It includes validation, cleansing, deduplication, and anomaly detection processes to identify and address any data quality issues.

4. Monitoring & Reporting:

- This component is responsible for monitoring the health, performance, and quality of the data pipeline.
- It includes dashboards, alerts, and reports that provide insights into the data processing workflow and enable stakeholders to track key metrics and performance indicators.

5. Data Outputs:

- This component represents the final output of the data processing pipeline.
- It includes cleaned, transformed, and validated datasets, as well as any reports or visualizations generated from the processed data.

Data Quality

1. Null Counts for Each Column:

- Null counts for each column were computed to identify missing values in the dataset.
- The number of null values for each column provides insights into data completeness and potential data quality issues.
- Null counts help in assessing the extent of missing data and determining the need for data imputation or cleansing.

2. Duplicate Count:

- Duplicate rows were identified and counted to detect and quantify duplicate records within the dataset.
- The presence of duplicate records can impact data integrity and analysis results, leading to erroneous insights and decisions.
- Quantifying duplicate records aids in understanding the scale of duplication and devising strategies to handle or eliminate duplicates.

3. Data Types:

- Data types of columns were inspected to ensure consistency and accuracy in data representation.
- Understanding the data types of columns is crucial for proper data manipulation, transformation, and analysis.
- Inconsistent or incorrect data types may lead to data processing errors and misinterpretation of results.

4. Filtering Data:

- Data was filtered based on a specific condition (e.g., "Year" greater than 2015) to include only relevant records.
- Filtering data helps focus analysis on subsets of data that meet predefined criteria, reducing noise and improving data quality.
- By applying filters, irrelevant or outdated data can be excluded from analysis, ensuring the relevance and accuracy of insights derived.

5. Column Profiling:

- Comprehensive column profiling was performed to analyze various characteristics of each column, including distinct values, null counts, and average value length.
- Distinct values provide insights into data variability and uniqueness, aiding in understanding data distribution and cardinality.
- Null counts highlight data completeness issues, guiding decisions on data imputation or exclusion.
- Average value length helps assess data consistency and identify potential anomalies or outliers.

6. Missing Values:

- Missing values were examined across all columns to identify columns with significant null counts.
- Missing values can indicate data quality issues such as incomplete data collection or data entry errors.
- Identifying columns with high null counts facilitates prioritization of data quality improvement efforts and informs data cleansing strategies.

7. Summary Statistics:

- Summary statistics were generated for numerical columns in the dataset to provide insights into data distribution and variability.
- Summary statistics include count, mean, standard deviation, minimum, and maximum values.
- These statistics offer a comprehensive overview of numerical data characteristics, aiding in data quality assessment and exploratory data analysis.

Conclusion:

- The data quality assessment conducted using the above checks provides valuable insights into the quality, completeness, and integrity of the dataset.
- By identifying missing values, duplicates, and inconsistencies, potential data quality issues were uncovered, paving the way for data cleansing and improvement efforts.
- The findings from the data quality assessment serve as a foundation for ensuring data reliability and trustworthiness, ultimately enhancing the validity and utility of downstream data analysis and decision-making processes.

Data Transfer

Overview

Azure Data Factory (ADF) is a cloud-based data integration service provided by Microsoft that allows users to create, schedule, and manage data pipelines for orchestrating and automating data movement and transformation tasks. In this section, we discuss the configuration and implementation of data transfer pipelines using Azure Data Factory to facilitate seamless movement of data between Azure Data Lake Storage (ADLS) instances.

Configuration

1.Data Factory Creation:

- The first step involves creating an Azure Data Factory instance within the Azure portal.
- Azure Data Factory provides a user-friendly interface for defining and managing data integration pipelines.

2. Linked Services:

- Linked services are used to establish connections to data sources and destinations.
- For our data transfer scenario, linked services for both source and destination ADLS instances need to be configured in Azure Data Factory.
- Azure Data Factory supports various data stores and services, including Azure Blob Storage, Azure SQL Database, and Azure Data Lake Storage.

3.Datasets:

- Datasets represent the data structures and schemas of the source and destination data.
- In Azure Data Factory, datasets define the input and output data formats for data transfer operations.
- Datasets for source and destination ADLS instances are created, specifying the file formats, folder paths, and data schemas.

4.Pipeline Creation:

- Pipelines serve as containers for organizing and orchestrating data movement and transformation activities.
- Within Azure Data Factory, pipelines are created to encapsulate the data transfer process from source to destination.
- Each pipeline consists of one or more activities, such as copy data activity, that define the specific data transfer operations.

Implementation:

1.Copy Data Activity:

- The primary activity used for data transfer within Azure Data Factory is the copy data activity.
- This activity enables copying data from a source dataset to a destination dataset, with support for various data formats and transformation options.
- Configuration settings include specifying source and destination linked services, datasets, mappings, and data movement options.

2.Mapping Data:

- Mapping defines the relationship between the source and destination data structures, including column mappings and data transformations.
- Azure Data Factory provides mapping options for simple copy operations as well as complex data transformations using Azure Data Flow.

3.Scheduling and Monitoring:

- Azure Data Factory allows users to schedule data transfer pipelines to run at specific intervals or triggered by events.
- Monitoring capabilities within Azure Data Factory provide visibility into pipeline execution, including status, duration, and performance metrics.
- Users can monitor pipeline runs, track data lineage, and troubleshoot issues using built-in monitoring and logging features.

Conclusion:

The configuration and implementation of data transfer pipelines using Azure Data Factory enable organizations to efficiently move data between Azure Data Lake Storage instances while maintaining data integrity, reliability, and security. By leveraging Azure Data Factory's intuitive interface and robust features, data engineers can orchestrate complex data movement tasks with ease, facilitating seamless data integration and interoperability across disparate data sources and destinations.

Technologies Used:

- **Azure Data Factory (ADF):** Used for data ingestion, Azure Data Factory allows us to efficiently collect data from various sources and move it to the desired destination. In this project, ADF is responsible for bringing in Olympics data from external sources

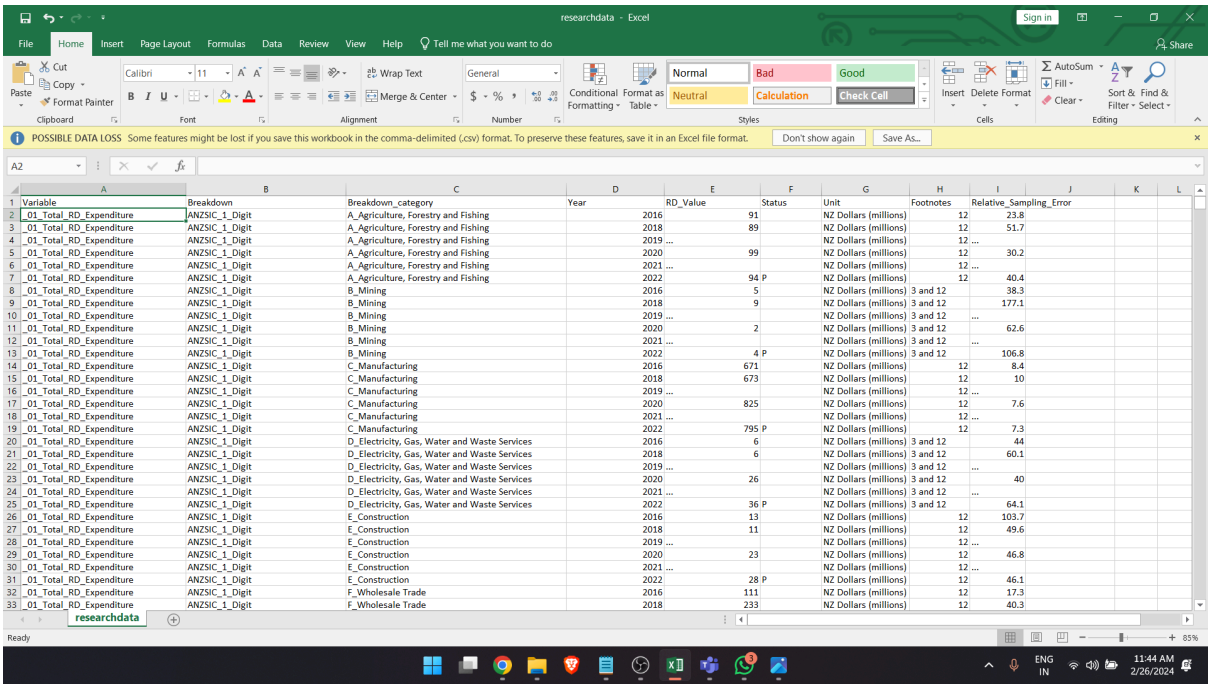
- **Azure Data Lake Storage Gen2:** This is where the ingested data is stored. Azure Data Lake Storage Gen2 provides a scalable and secure platform for storing large volumes of data. It enables us to manage, access, and analyze data effectively.

- **Azure Databricks:** For data transformation, we leverage Azure Databricks with PySpark. Databricks provides a collaborative environment for data engineers and data scientists to work together on big data projects. In our project, we use PySpark to clean, reshape, and process the raw Olympics data into a more usable format.

Results And Evaluation:

Data Quality:

Sample data used:



researchdata - Excel

Variable	Breakdown	Breakdown_category	Year	RD_Value	Status	Unit	Footnotes	Relative_Sampling_Error
_01_Total_RD_Expenditure	ANZSIC_1_Digit	A_Agriculture, Forestry and Fishing	2016	91		NZ Dollars (millions)	12	23.8
_01_Total_RD_Expenditure	ANZSIC_1_Digit	A_Agriculture, Forestry and Fishing	2018	89		NZ Dollars (millions)	12	51.7
_01_Total_RD_Expenditure	ANZSIC_1_Digit	A_Agriculture, Forestry and Fishing	2019	...		NZ Dollars (millions)	12	...
_01_Total_RD_Expenditure	ANZSIC_1_Digit	A_Agriculture, Forestry and Fishing	2020	99		NZ Dollars (millions)	12	30.2
_01_Total_RD_Expenditure	ANZSIC_1_Digit	A_Agriculture, Forestry and Fishing	2021	...		NZ Dollars (millions)	12	...
_01_Total_RD_Expenditure	ANZSIC_1_Digit	A_Agriculture, Forestry and Fishing	2022	94	P	NZ Dollars (millions)	12	40.4
_01_Total_RD_Expenditure	ANZSIC_1_Digit	B_Mining	2016	5		NZ Dollars (millions)	3 and 12	38.3
_01_Total_RD_Expenditure	ANZSIC_1_Digit	B_Mining	2018	9		NZ Dollars (millions)	3 and 12	177.1
_01_Total_RD_Expenditure	ANZSIC_1_Digit	B_Mining	2019	...		NZ Dollars (millions)	3 and 12	...
_01_Total_RD_Expenditure	ANZSIC_1_Digit	B_Mining	2020	2		NZ Dollars (millions)	3 and 12	62.6
_01_Total_RD_Expenditure	ANZSIC_1_Digit	B_Mining	2021	...		NZ Dollars (millions)	3 and 12	...
_01_Total_RD_Expenditure	ANZSIC_1_Digit	B_Mining	2022	4	P	NZ Dollars (millions)	3 and 12	106.8
_01_Total_RD_Expenditure	ANZSIC_1_Digit	C_Manufacturing	2016	671		NZ Dollars (millions)	12	8.4
_01_Total_RD_Expenditure	ANZSIC_1_Digit	C_Manufacturing	2018	673		NZ Dollars (millions)	12	10
_01_Total_RD_Expenditure	ANZSIC_1_Digit	C_Manufacturing	2019	...		NZ Dollars (millions)	12	...
_01_Total_RD_Expenditure	ANZSIC_1_Digit	C_Manufacturing	2020	825		NZ Dollars (millions)	12	7.6
_01_Total_RD_Expenditure	ANZSIC_1_Digit	C_Manufacturing	2021	...		NZ Dollars (millions)	12	...
_01_Total_RD_Expenditure	ANZSIC_1_Digit	C_Manufacturing	2022	795	P	NZ Dollars (millions)	12	7.3
_01_Total_RD_Expenditure	ANZSIC_1_Digit	D_Electricity, Gas, Water and Waste Services	2016	6		NZ Dollars (millions)	3 and 12	44
_01_Total_RD_Expenditure	ANZSIC_1_Digit	D_Electricity, Gas, Water and Waste Services	2018	6		NZ Dollars (millions)	3 and 12	60.1
_01_Total_RD_Expenditure	ANZSIC_1_Digit	D_Electricity, Gas, Water and Waste Services	2019	...		NZ Dollars (millions)	3 and 12	...
_01_Total_RD_Expenditure	ANZSIC_1_Digit	D_Electricity, Gas, Water and Waste Services	2020	26		NZ Dollars (millions)	3 and 12	40
_01_Total_RD_Expenditure	ANZSIC_1_Digit	D_Electricity, Gas, Water and Waste Services	2021	...		NZ Dollars (millions)	3 and 12	...
_01_Total_RD_Expenditure	ANZSIC_1_Digit	D_Electricity, Gas, Water and Waste Services	2022	36	P	NZ Dollars (millions)	3 and 12	64.1
_01_Total_RD_Expenditure	ANZSIC_1_Digit	E_Construction	2016	13		NZ Dollars (millions)	12	103.7
_01_Total_RD_Expenditure	ANZSIC_1_Digit	E_Construction	2018	11		NZ Dollars (millions)	12	49.6
_01_Total_RD_Expenditure	ANZSIC_1_Digit	E_Construction	2019	...		NZ Dollars (millions)	12	...
_01_Total_RD_Expenditure	ANZSIC_1_Digit	E_Construction	2020	23		NZ Dollars (millions)	12	46.8
_01_Total_RD_Expenditure	ANZSIC_1_Digit	E_Construction	2021	...		NZ Dollars (millions)	12	...
_01_Total_RD_Expenditure	ANZSIC_1_Digit	E_Construction	2022	28	P	NZ Dollars (millions)	12	46.1
_01_Total_RD_Expenditure	ANZSIC_1_Digit	F_Wholesale Trade	2016	111		NZ Dollars (millions)	12	17.3
_01_Total_RD_Expenditure	ANZSIC_1_Digit	F_Wholesale Trade	2018	233		NZ Dollars (millions)	12	40.3

Data Explanation for Data Quality Management Project:

The provided dataset contains information related to research and development (R&D) expenditure breakdown across different industries and years. Each row represents a specific breakdown of R&D expenditure for a particular industry category, year, and other relevant attributes. Here's a breakdown of the columns in the dataset:

1. **Variable:** Represents the variable or type of R&D expenditure breakdown.
2. **Breakdown:** Indicates the breakdown category of R&D expenditure, typically based on industry classification (e.g., ANZSIC_1_Digit).
3. **Breakdown Category:** Specifies the category or sector to which the R&D expenditure breakdown belongs (e.g., Agriculture, Forestry and Fishing, Mining, Manufacturing).

4. Year: Represents the year for which the R&D expenditure breakdown is reported.
5. RD Value: Indicates the value of R&D expenditure in NZ Dollars (millions) for the corresponding year and breakdown category.
6. Status: Provides additional information about the status or nature of the R&D expenditure data (e.g., provisional data denoted by "P").
7. Unit: Specifies the unit of measurement for R&D expenditure values (e.g., NZ Dollars (millions)).
8. Footnotes: Includes any footnotes or annotations associated with the R&D expenditure data.
9. Relative Sampling Error: Indicates the relative sampling error associated with the R&D expenditure data, providing insights into data quality and reliability.

Relevance to Data Quality Management Project:

The provided dataset serves as a valuable resource for the data quality management project, as it represents real-world data that requires thorough assessment and validation to ensure its accuracy, completeness, and consistency. Here's how the dataset is relevant to the project:

1. Data Profiling: The dataset can be subjected to data profiling techniques to analyze its structure, identify data types, detect missing values, and assess data distribution across different variables and categories.

2. Data Cleansing: Through data cleansing processes, any inconsistencies, anomalies, or errors in the dataset can be identified and rectified. This may involve handling missing values, correcting erroneous entries, and standardizing data formats.

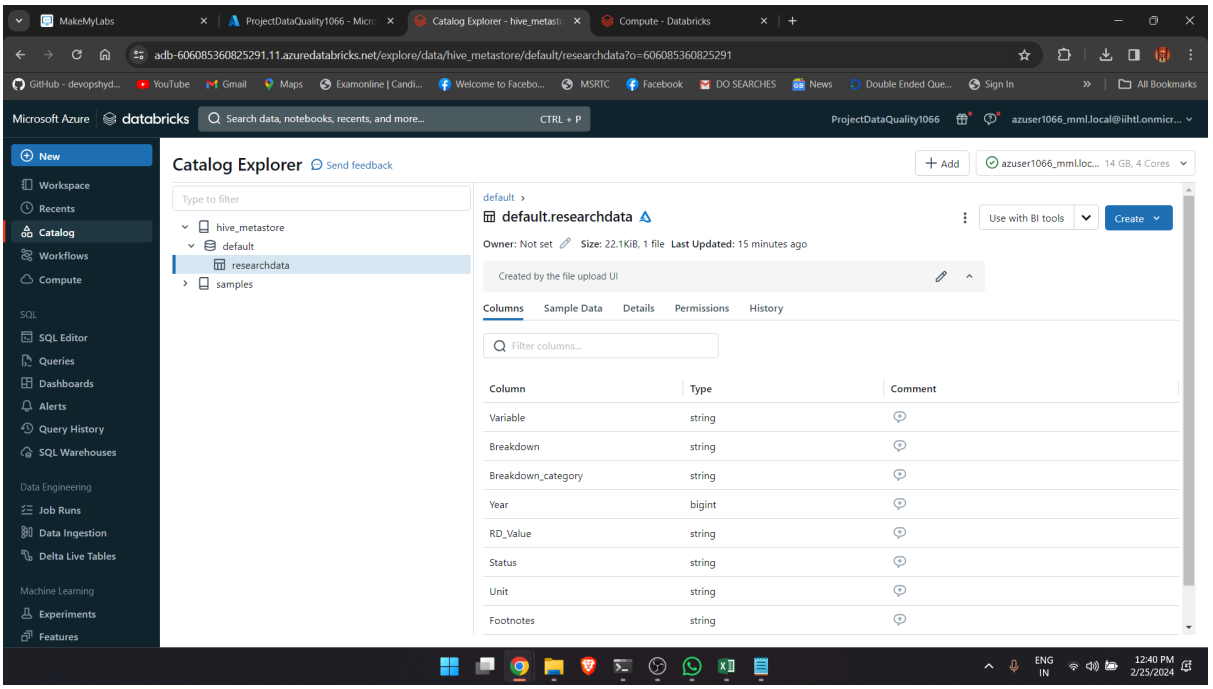
3. Data Validation: The dataset can be validated against predefined business rules and expectations to ensure that it meets quality standards and accurately represents the underlying R&D expenditure breakdown.

4. Quality Assessment: Various data quality metrics, such as completeness, accuracy, consistency, and timeliness, can be evaluated for the dataset to gauge its overall quality and fitness for intended analytical or reporting purposes.

5.Documentation and Metadata Management: Detailed documentation of the dataset's attributes, metadata, and quality assessment findings can be maintained to provide transparency, traceability, and accountability in the data quality management process.

By leveraging the provided dataset as a case study, the data quality management project aims to establish robust data governance practices, enhance data quality assurance processes, and ultimately, foster trust and confidence in the reliability of organizational data assets.

Adding a file using Hive Metastore:



Adding a file using Hive Metastore typically involves a few steps, including preparing the file, registering it with the Metastore, and optionally performing metadata updates if needed. Below is a general process for adding a file using Hive Metastore:

1. Prepare the File:

- Ensure that the file you want to add is available in a location accessible to the Hive Metastore. This could be a local file system or a supported distributed file system such as Hadoop Distributed File System (HDFS) or Azure Data Lake Storage (ADLS).

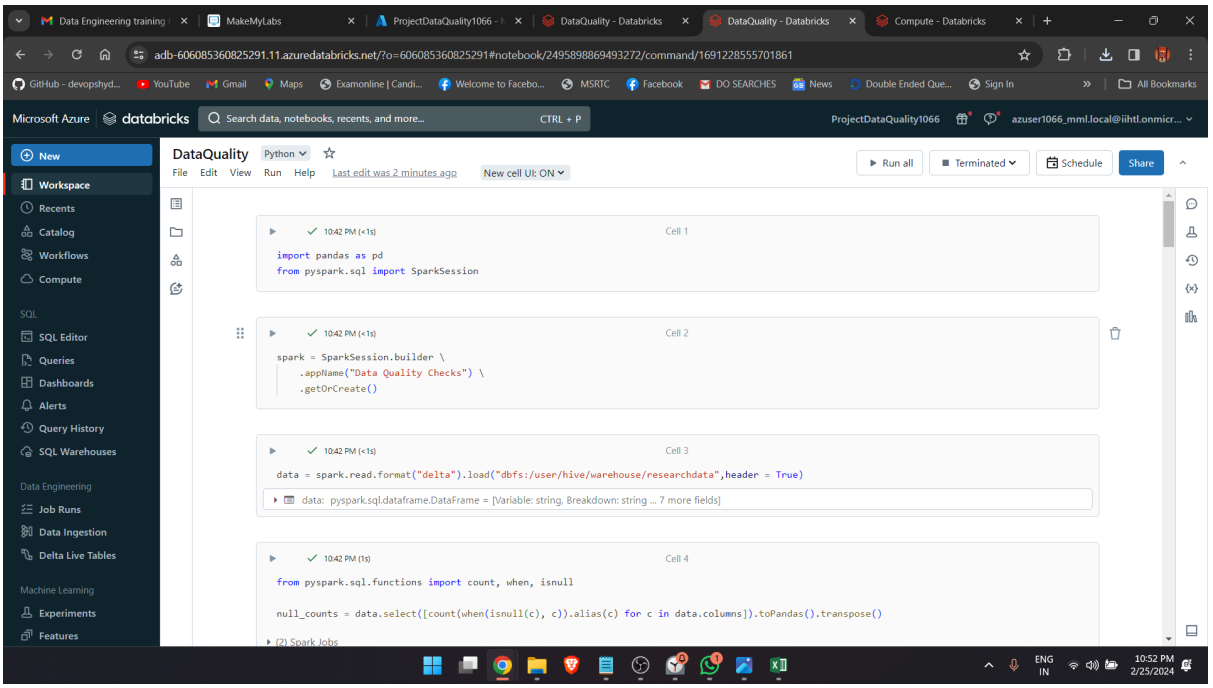
2. Start Hive Shell or Hive CLI:

- Open the Hive shell or Hive CLI on the machine where the Hive Metastore service is running. You can typically access the Hive shell by executing the `hive` command in the terminal.

3. Register the File with the Metastore:

- Use the `CREATE TABLE` or `CREATE EXTERNAL TABLE` statement to register the file with the Hive Metastore. The choice between `CREATE TABLE` and `CREATE EXTERNAL TABLE` depends on whether you want Hive to manage the data files or merely reference them.

Implementation of code :



Certainly! Let's break down the provided code step by step:

1. Import Libraries:

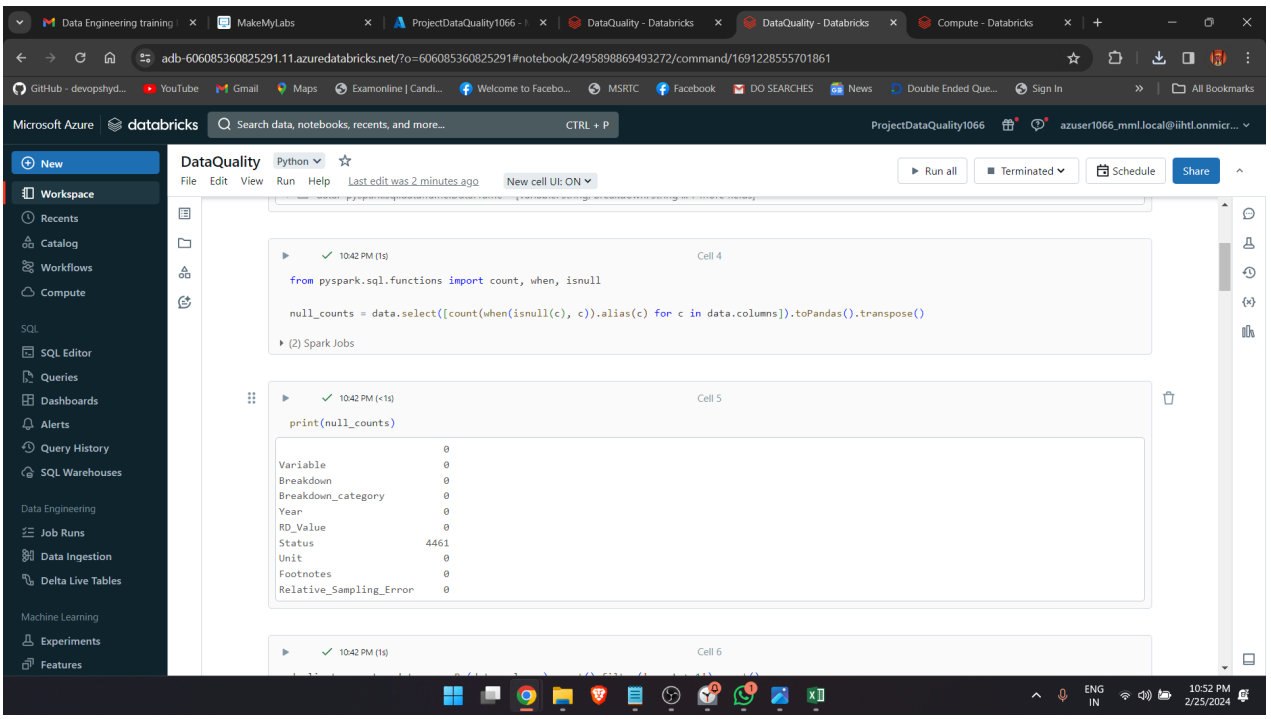
- The first step imports the necessary libraries for data manipulation and Spark session initialization.

- `import pandas as pd`: Imports the Pandas library, a popular data manipulation and analysis library in Python, aliased as `pd`.

- `from pyspark.sql import SparkSession`: Imports the SparkSession class from the `pyspark.sql` module, which is required to create a Spark session.

2. Read Delta Table:

- `data=spark.read.format("delta").load("dbfs:/user/hive/warehouse/researchdata", header=True)`: Reads data from a Delta table located at the specified path into a DataFrame named `data`.
- `spark.read`: Invokes the `read` method of the SparkSession to initiate the DataFrameReader.
- `.format("delta")`: Specifies the format of the data source to be read. In this case, it's "delta", indicating that the data is stored in the Delta format.
- `.load("dbfs:/user/hive/warehouse/researchdata", header=True)`: Loads data from the specified path into the DataFrame.
- `"dbfs:/user/hive/warehouse/researchdata"`: Specifies the path to the Delta table. `dbfs:/` is a Databricks-specific file system protocol, and `user/hive/warehouse/researchdata` is the path to the Delta table.
- `header=True`: Indicates that the first row of the data contains column headers, and Spark should treat it accordingly.



1. Import Libraries:

- `from pyspark.sql.functions import count, when, isnull`: Imports specific functions from the `pyspark.sql.functions` module. These functions are used for data manipulation and analysis in Spark DataFrame operations.

2. Calculate Null Counts:

- `null_counts = data.select([count(when(isnull(c), c)).alias(c) for c in data.columns])`: This line of code calculates the count of null values for each column in the DataFrame `data`.
- `data.columns`: Accesses the list of column names in the DataFrame `data`.
- `[count(when(isnull(c), c)).alias(c) for c in data.columns]`: This is a list comprehension that iterates over each column `c` in the DataFrame's columns.
- `isnull(c)`: Checks if the value in column `c` is null.
- `when(isnull(c), c)`: Uses the `when` function to return the original value `c` if it's not null, and null otherwise.
- `count(when(isnull(c), c))`: Counts the occurrences of null values in column `c`.
- `.alias(c)`: Specifies an alias for the resulting count, using the original column name `c`.

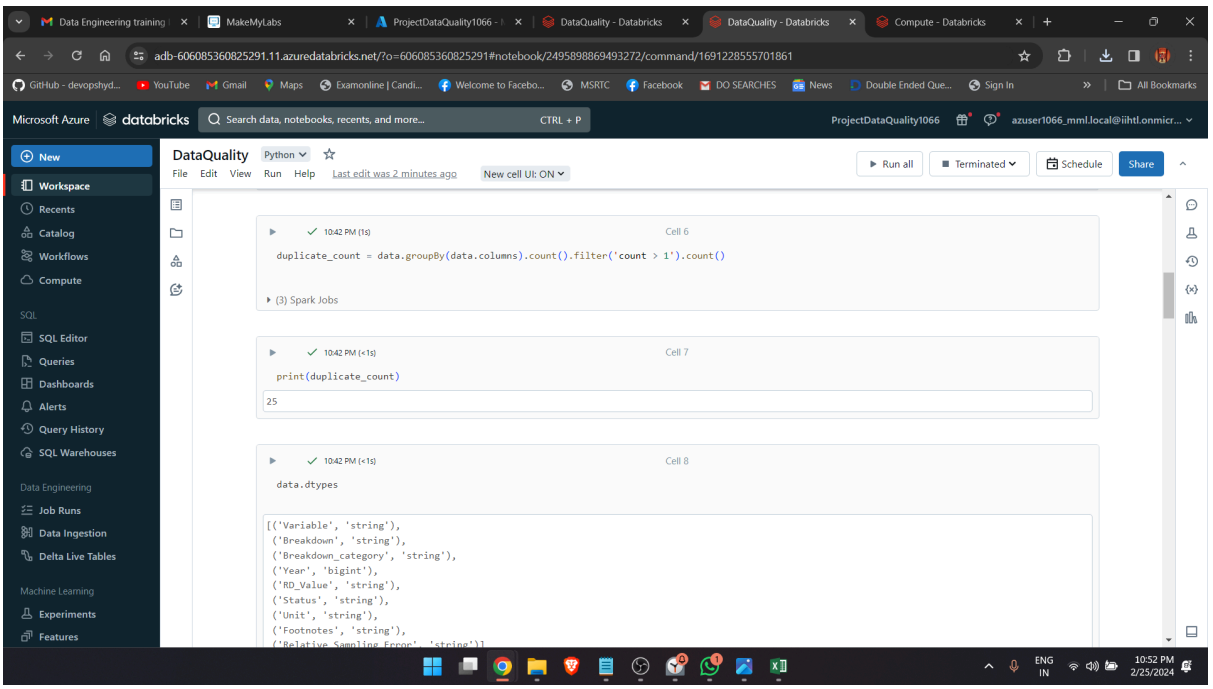
- `data.select(...)`: Selects the calculated counts as a new DataFrame.

3. Convert to Pandas DataFrame:

- `.toPandas().transpose()`: Converts the Spark DataFrame `null_counts` to a Pandas DataFrame and transposes it.
- `.toPandas()`: Converts the Spark DataFrame to a Pandas DataFrame. This action brings the data from distributed Spark memory to the driver node's memory.
- `.transpose()`: Transposes the Pandas DataFrame, swapping rows and columns. This makes the DataFrame more readable, with columns representing the original DataFrame's columns and rows representing the null counts for each column.

4. Print Null Counts:

- `print(null_counts)`: Prints the Pandas DataFrame `null_counts`, which contains the counts of null values for each column in the original DataFrame `data`.



1. Count Duplicate Rows:

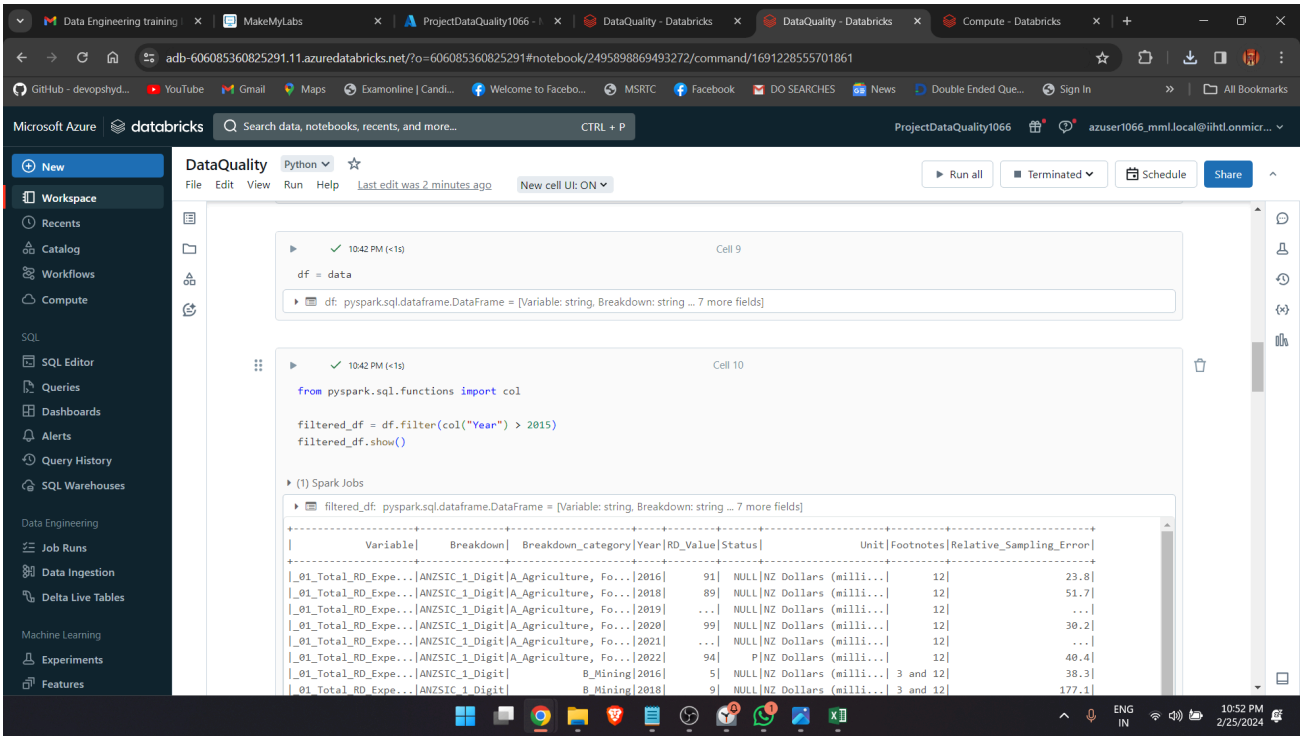
- `duplicate_count = data.groupBy(data.columns).count().filter('count > 1').count()`: This line of code calculates the number of duplicate rows in the DataFrame `data`.
- `data.groupBy(data.columns)`: Groups the DataFrame `data` by all its columns. This means that each unique combination of values across all columns is treated as a group.
- `.count()`: Counts the number of rows in each group.
- `.filter('count > 1')`: Filters the grouped DataFrame to retain only those groups where the count of rows is greater than 1, indicating duplicates.
- `.count()`: Counts the number of remaining groups after filtering, which represents the number of duplicate rows in the original DataFrame `data`.
- `duplicate_count`: Stores the count of duplicate rows.

2. Print Duplicate Count:

- `print(duplicate_count)`: Prints the count of duplicate rows calculated in the previous step.

3. Data Types;

- `data.dtypes`: This line of code retrieves the data types of all columns in the DataFrame `data`.
- `data.dtypes` returns a list of tuples, where each tuple contains the name of a column and its corresponding data type.



1. Assign DataFrame:

- `df = data`: This line assigns the DataFrame `data` to a new variable `df`.
- This step is simply creating an alias `df` for the original DataFrame `data`. Both `df` and `data` now refer to the same DataFrame object.

2. Import Column Function:

- `from pyspark.sql.functions import col`: This line imports the `col` function from the `pyspark.sql.functions` module.

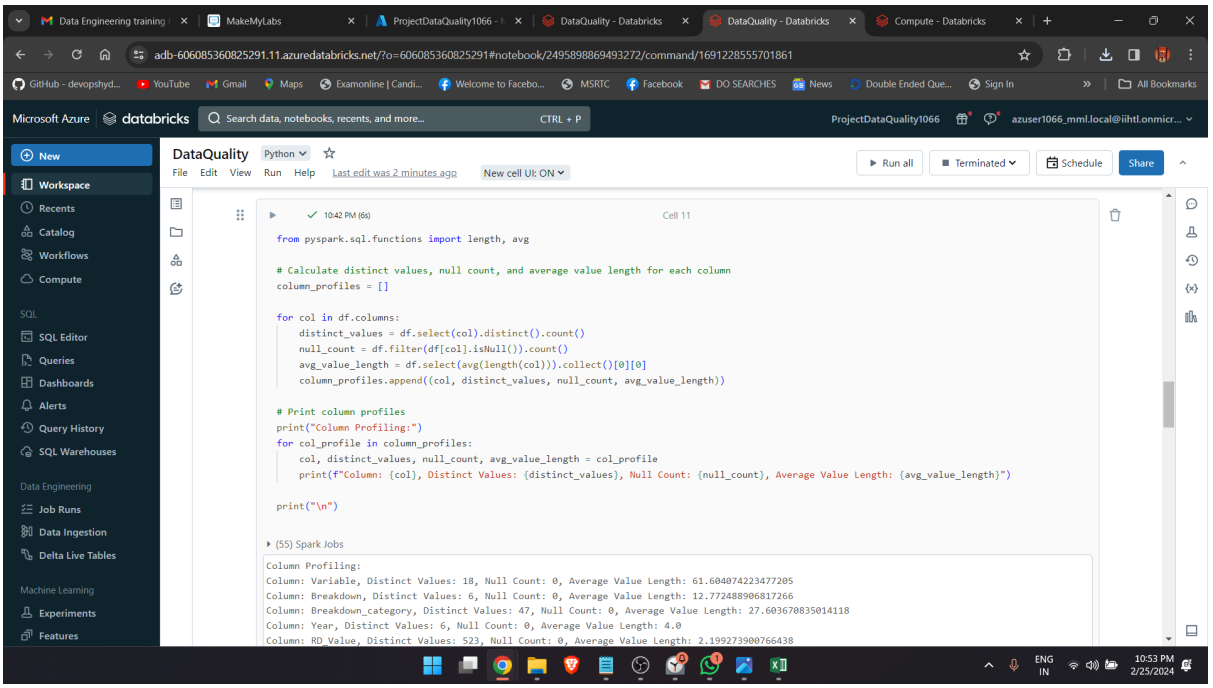
- The `col` function is used to reference DataFrame columns in Spark SQL expressions.

3. Filter DataFrame:

- `filtered_df = df.filter(col("Year") > 2015)`: This line filters the DataFrame `df` to include only rows where the value in the "Year" column is greater than 2015.
- `col("Year")`: References the "Year" column using the `col` function.
- `> 2015`: Specifies the condition for filtering, selecting rows where the value in the "Year" column is greater than 2015.
- The `filter` method is used to apply the condition and retain only the rows that satisfy it.
- The filtered DataFrame is assigned to a new variable `filtered_df`.

4. Show Filtered DataFrame:

- `filtered_df.show()`: This line displays the contents of the filtered DataFrame `filtered_df`.
- The `show` method is used to print the contents of the DataFrame to the console in a tabular format.



1. Import Functions:

- `from pyspark.sql.functions import length, avg`: This line imports the `length` and `avg` functions from the `pyspark.sql.functions` module.
- The `length` function is used to calculate the length of a string.
- The `avg` function is used to calculate the average of values in a column.

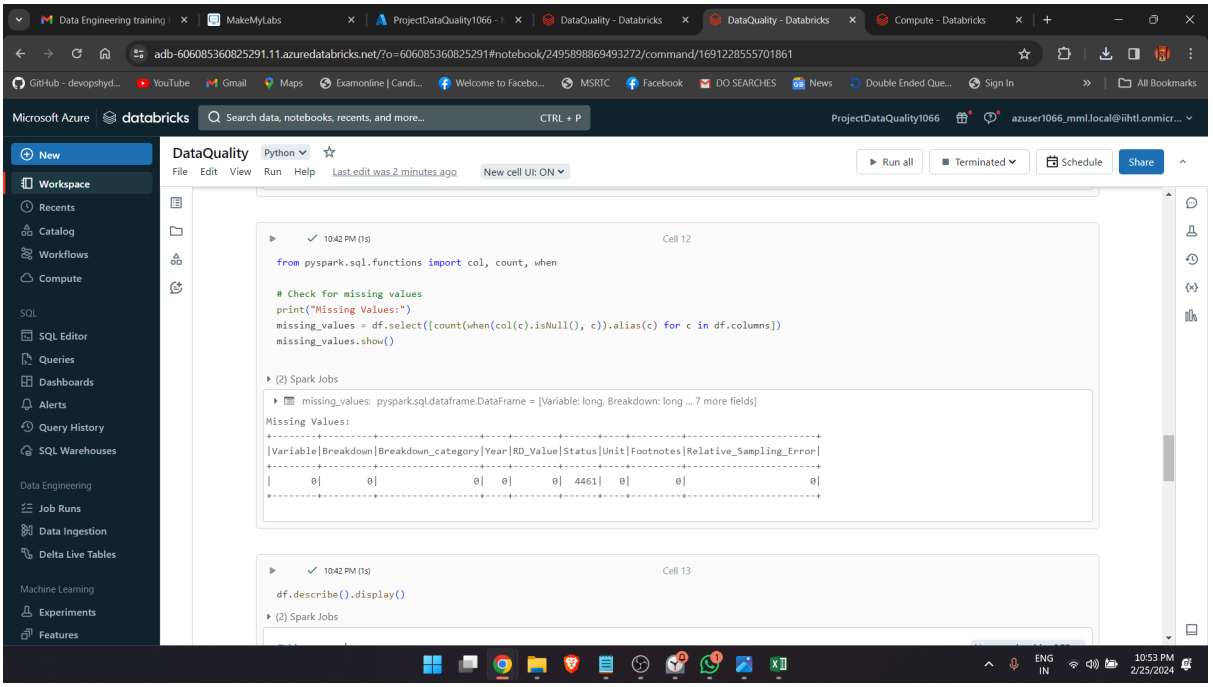
2. Column Profiling Loop:

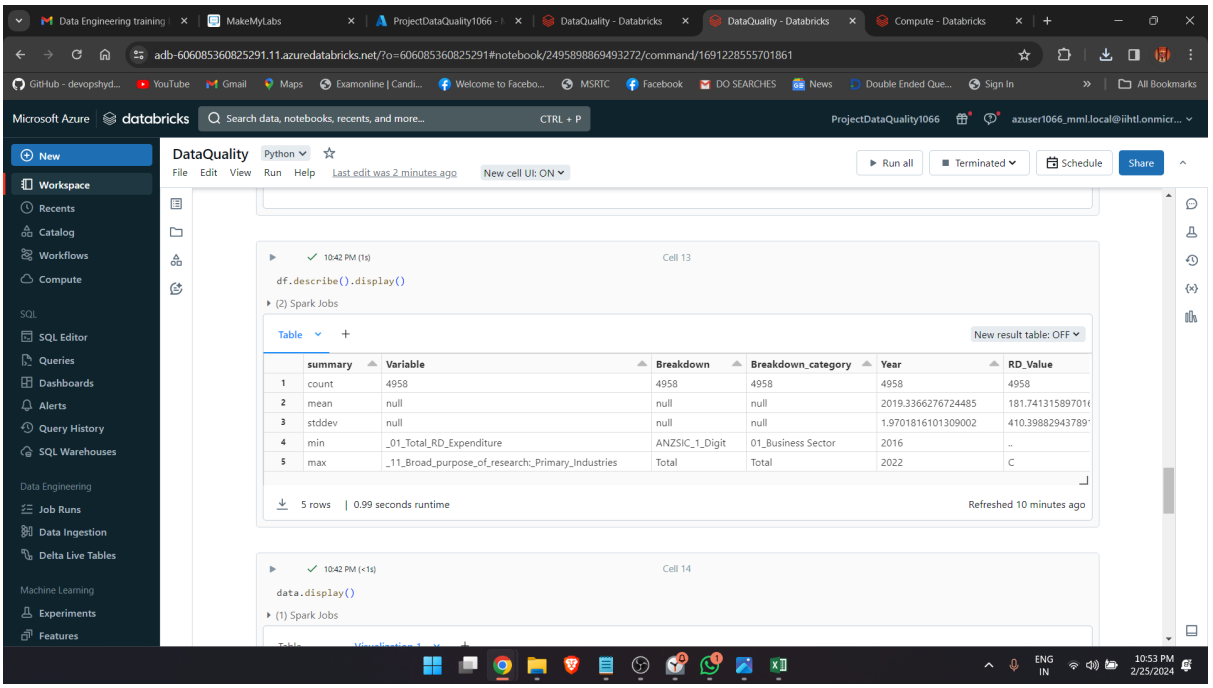
- `column_profiles = []`: Initializes an empty list to store column profiles.
- `for col in df.columns:`: Iterates over each column in the DataFrame `df`.
- `df.columns` returns a list of column names in the DataFrame.
- Inside the loop:
- `distinct_values = df.select(col).distinct().count()`: Calculates the number of distinct values in the current column `col`.
- `null_count = df.filter(df[col].isNull()).count()`: Calculates the count of null values in the current column `col`.
- `avg_value_length = df.select(avg(length(col))).collect()[0][0]`: Calculates the average length of values in the current column `col`.
- `length(col)`: Calculates the length of values in the column using the `length` function.
- `avg(length(col))`: Calculates the average of the lengths using the `avg` function.
- `.collect()[0][0]`: Retrieves the average value from the collected result.
- Appends a tuple `(col, distinct_values, null_count, avg_value_length)` representing the column profile to the `column_profiles` list.

3. Print Column Profiles:

- `print("Column Profiling:")`: Prints a header for the column profiles.
- `for col_profile in column_profiles:`: Iterates over each column profile tuple in the `column_profiles` list.
- Unpacks the tuple into `col`, `distinct_values`, `null_count`, and `avg_value_length`.

- Prints the column name, distinct values count, null count, and average value length using formatted string (f-string).





1. Import Functions:

- `from pyspark.sql.functions import col, count, when`: This line imports the `col`, `count`, and `when` functions from the `pyspark.sql.functions` module.

- The `col` function is used to reference DataFrame columns in Spark SQL expressions.
- The `count` function is used to count the occurrences of non-null values in a column.
- The `when` function is used to conditionally evaluate expressions.

2.Check for Missing Values:

- `print("Missing Values:")`: Prints a header indicating that missing values are being checked.
- `missing_values = df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns])`: This line calculates the count of missing values for each column in the DataFrame `df` and creates a new DataFrame `missing_values`.
- The list comprehension `[count(when(col(c).isNull(), c)).alias(c) for c in df.columns]` iterates over each column `c` in the DataFrame's columns.
- `col(c).isNull()`: Checks if the value in column `c` is null using the `col` and `isNull` functions.
- `when(col(c).isNull(), c)`: Returns the original value `c` if it's null, and null otherwise.
- `count(when(col(c).isNull(), c))`: Counts the occurrences of null values in column `c`.
- .alias(c)`: Specifies an alias for the resulting count, using the original column name `c`.

3. Show Missing Values:

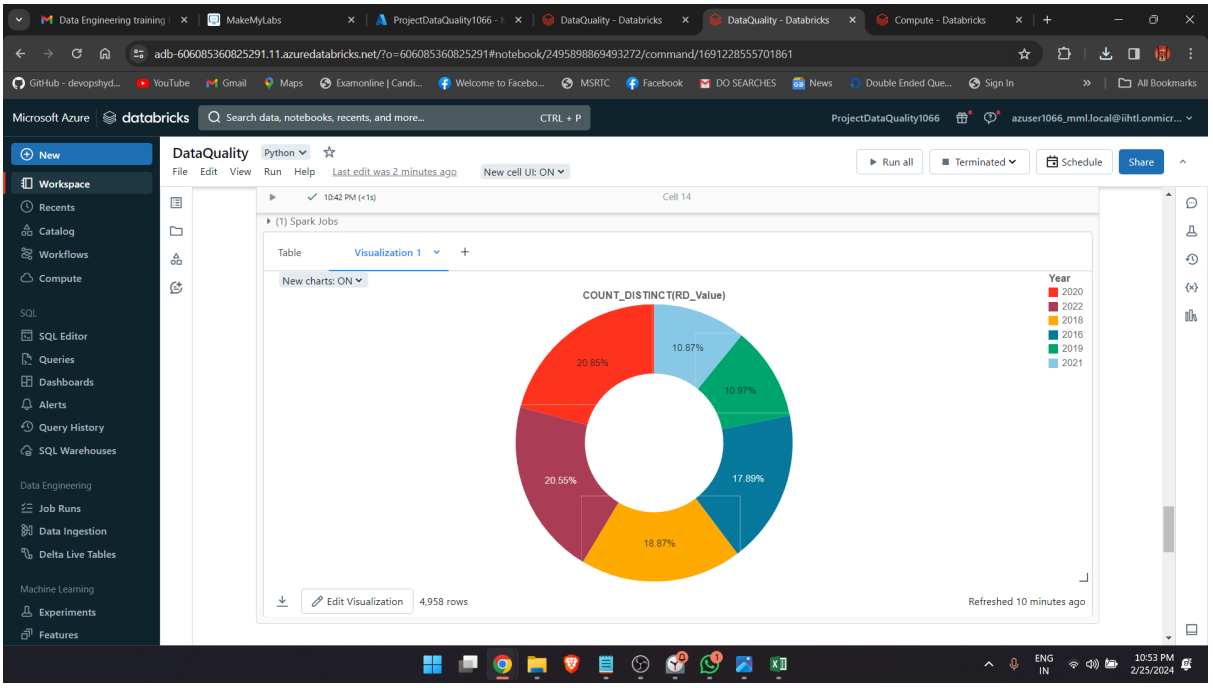
- `missing_values.show()`: Displays the DataFrame `missing_values`, showing the count of missing values for each column in a tabular format.

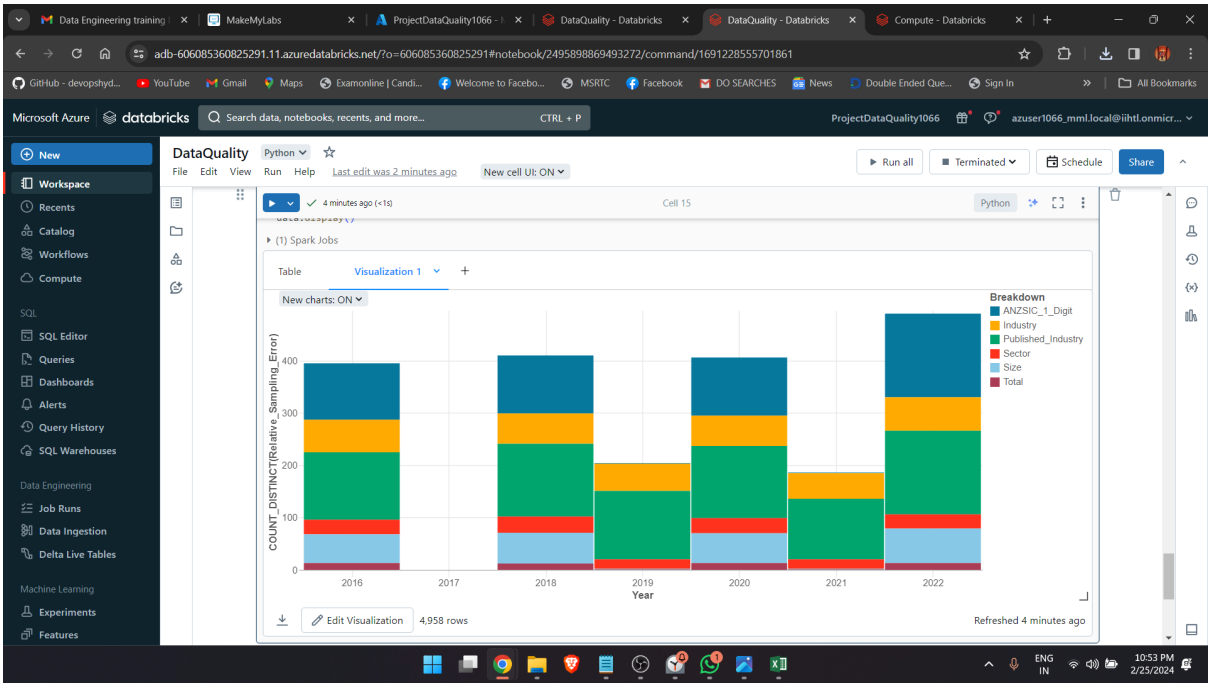
4. Describe DataFrame:

- `df.describe().display()`: This line generates descriptive statistics for the DataFrame `df` and displays them.
- `df.describe()`: Calculates summary statistics (count, mean, stddev, min, max) for numeric columns in the DataFrame.
- `.display()`: Displays the summary statistics in a formatted tabular format

Data visualisation :

Visualisation helps to represent data visually through charts, graphs, or maps, facilitating easier interpretation, analysis, and communication of patterns, trends, and insights within the data.

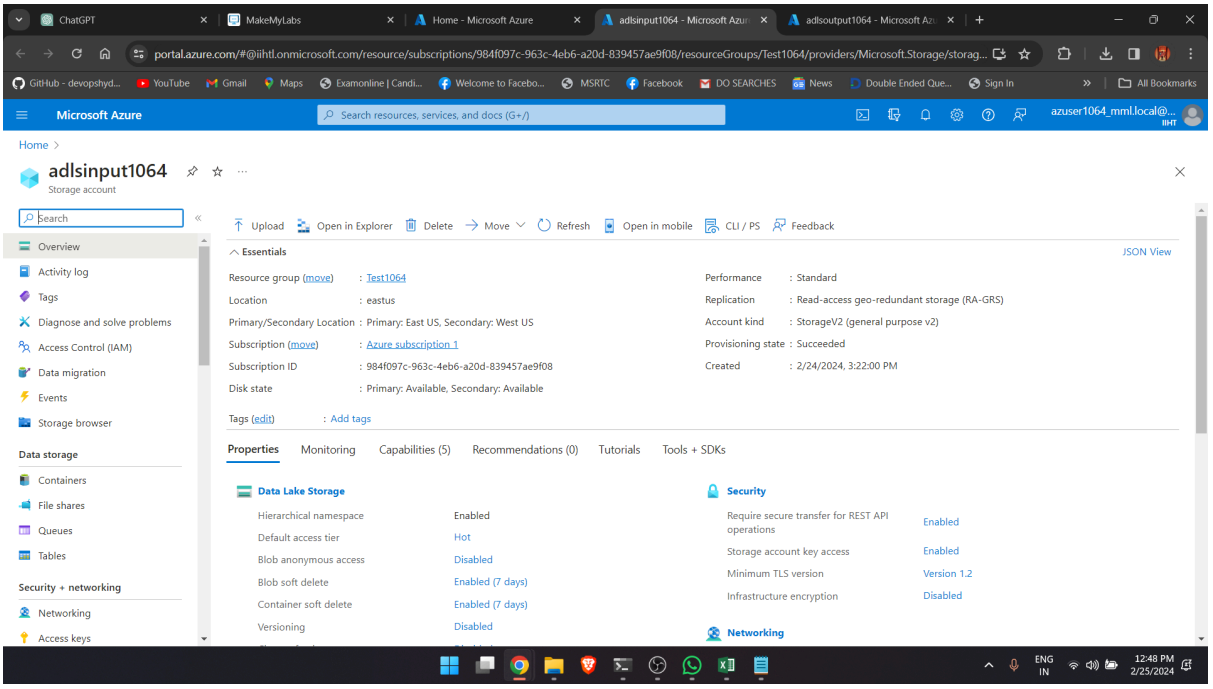


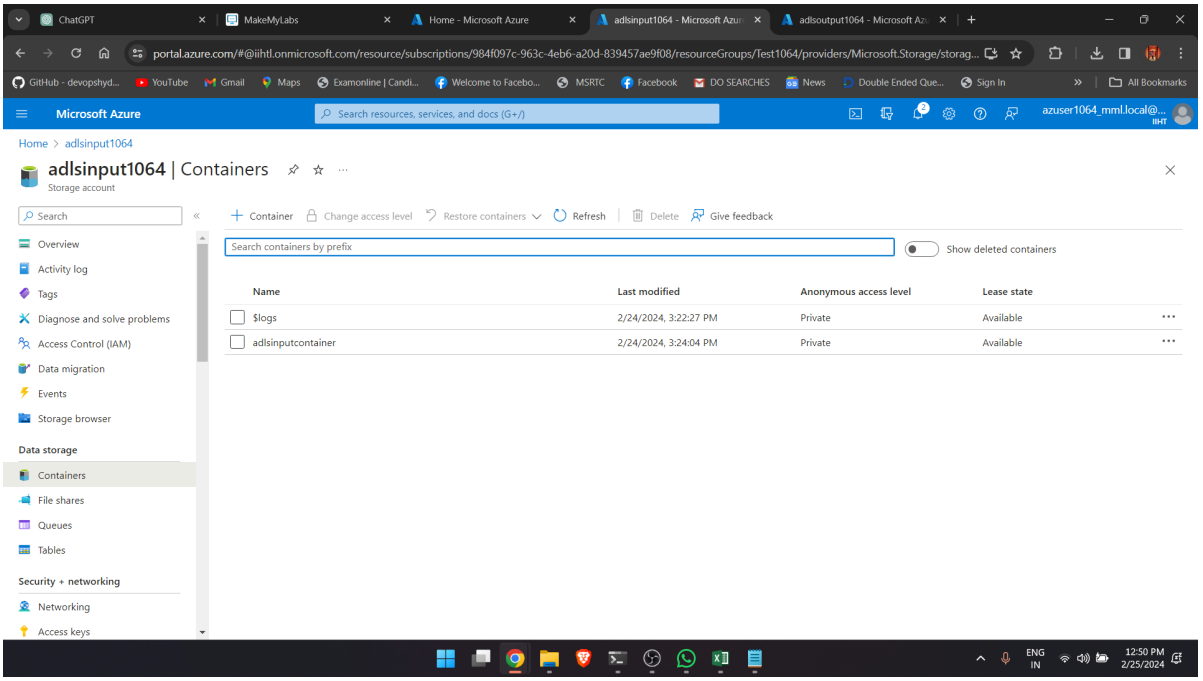


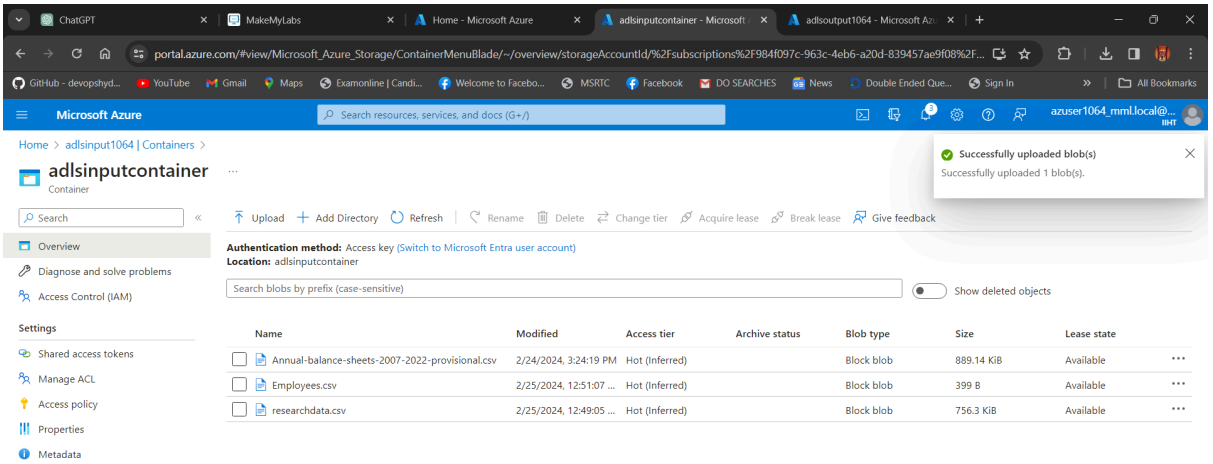
ADLS to ADLS :

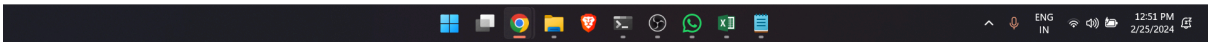
Creating two Azure data lake storage accounts and containers:

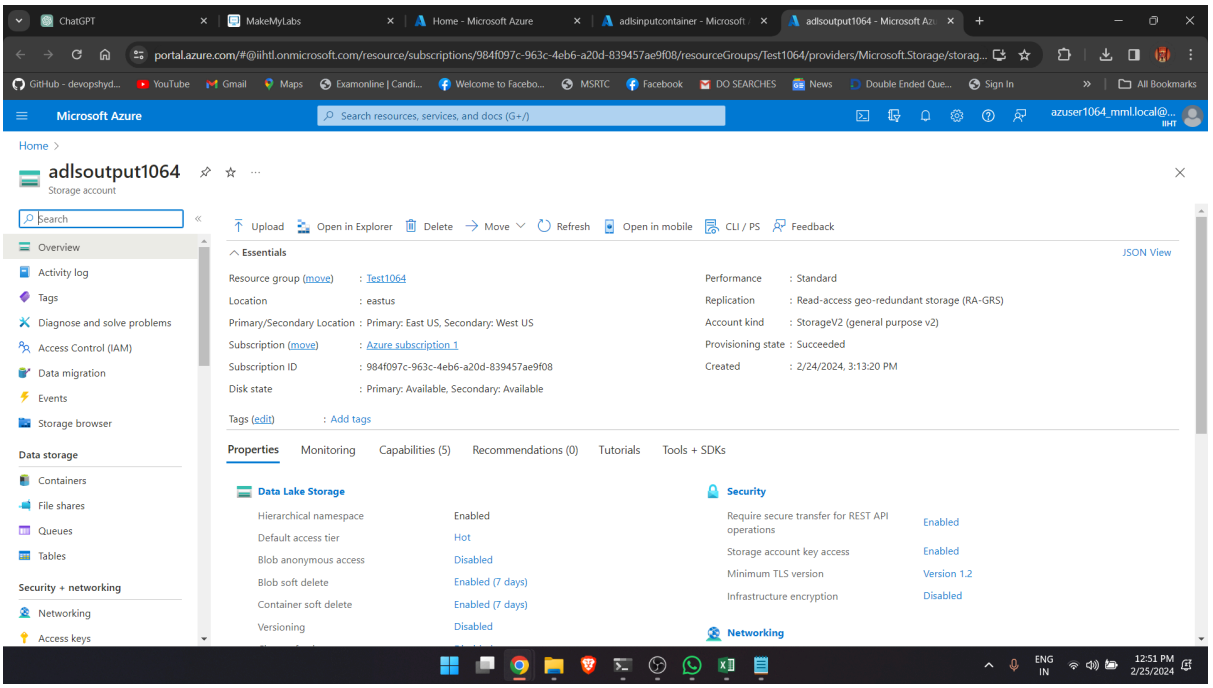
- Created one data storage account having an input container with sample files.
- Created one data storage account having a blank output container for copying the input container files.



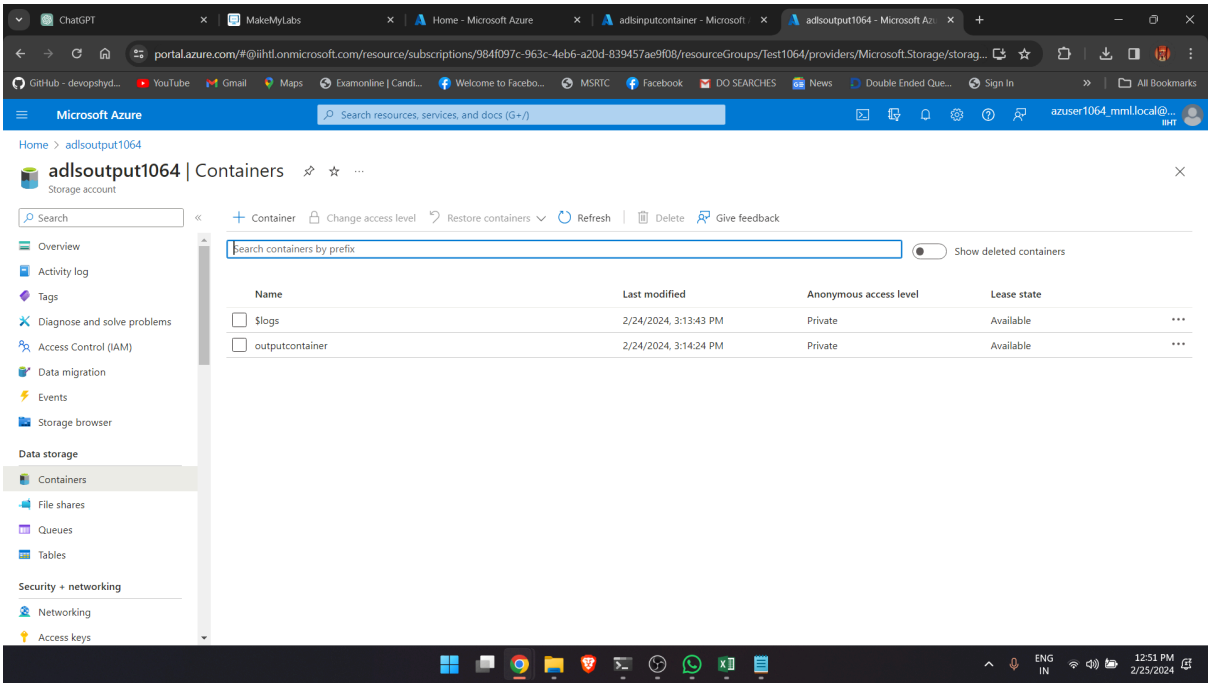


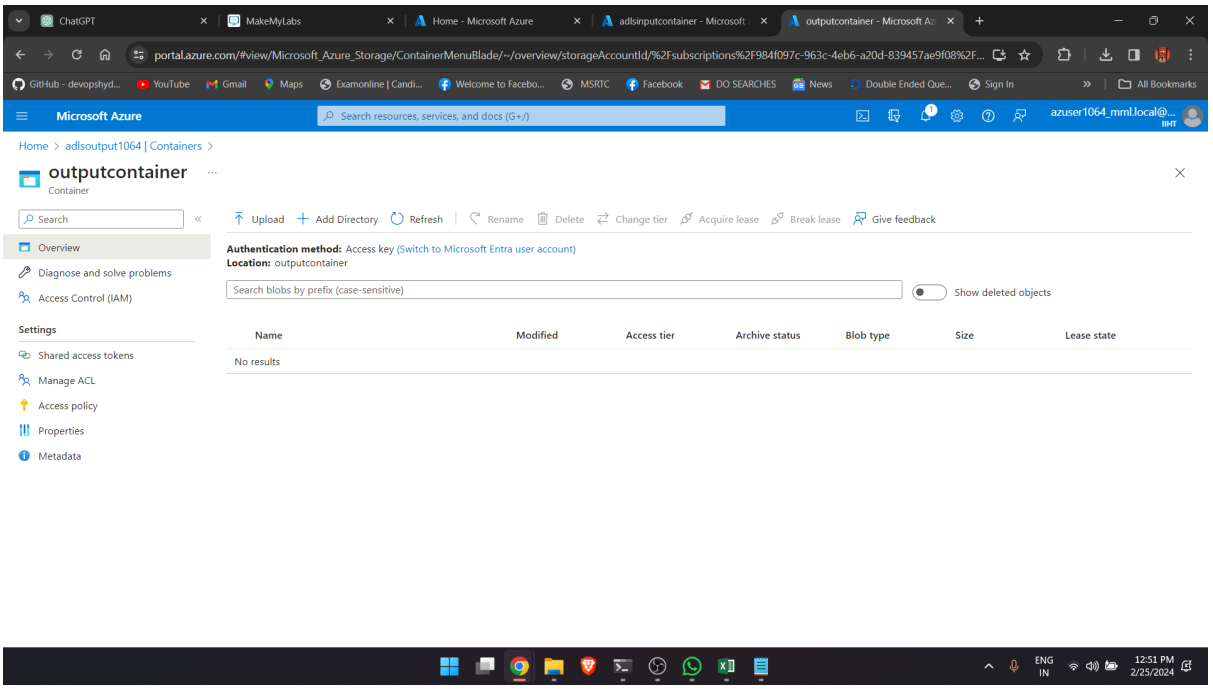

ChatGPT
MakeMyLabs
Home - Microsoft Azure
adlsinputcontainer - Microsoft
adlsoutput1064 - Microsoft Azu
portal.azure.com/#view/Microsoft_Azure_Storage/ContainerMenuBlade/~/overview/storageAccountId/%2Fsubscriptions%2F984f097c-963c-4eb6-a20d-839457ae9f08%2F...
Microsoft Azure
Search resources, services, and docs (G+/)
azuser1064_mml.local@...
IIHT
Home > adlsinput1064 | Containers >
adlsinputcontainer
Container
Successfully uploaded blob(s)
Successfully uploaded 1 blob(s).
Search
Overview
Diagnose and solve problems
Access Control (IAM)
Settings
Shared access tokens
Manage ACL
Access policy
Properties
Metadata
Upload
Add Directory
Refresh
Rename
Delete
Change tier
Acquire lease
Break lease
Give feedback
Authentication method: Access key (Switch to Microsoft Entra user account)
Location: adlsinputcontainer
Search blobs by prefix (case-sensitive)
Show deleted objects
Name
Modified
Access tier
Archive status
Blob type
Size
Lease state
Annual-balance-sheets-2007-2022-provisional.csv
2/24/2024, 3:24:19 PM
Hot (Inferred)
Block blob
889.14 KiB
Available
Employees.csv
2/25/2024, 12:51:07 ...
Hot (Inferred)
Block blob
399 B
Available
researchdata.csv
2/25/2024, 12:49:05 ...
Hot (Inferred)
Block blob
756.3 KiB
Available


ENG
IN
12:51 PM
2/25/2024

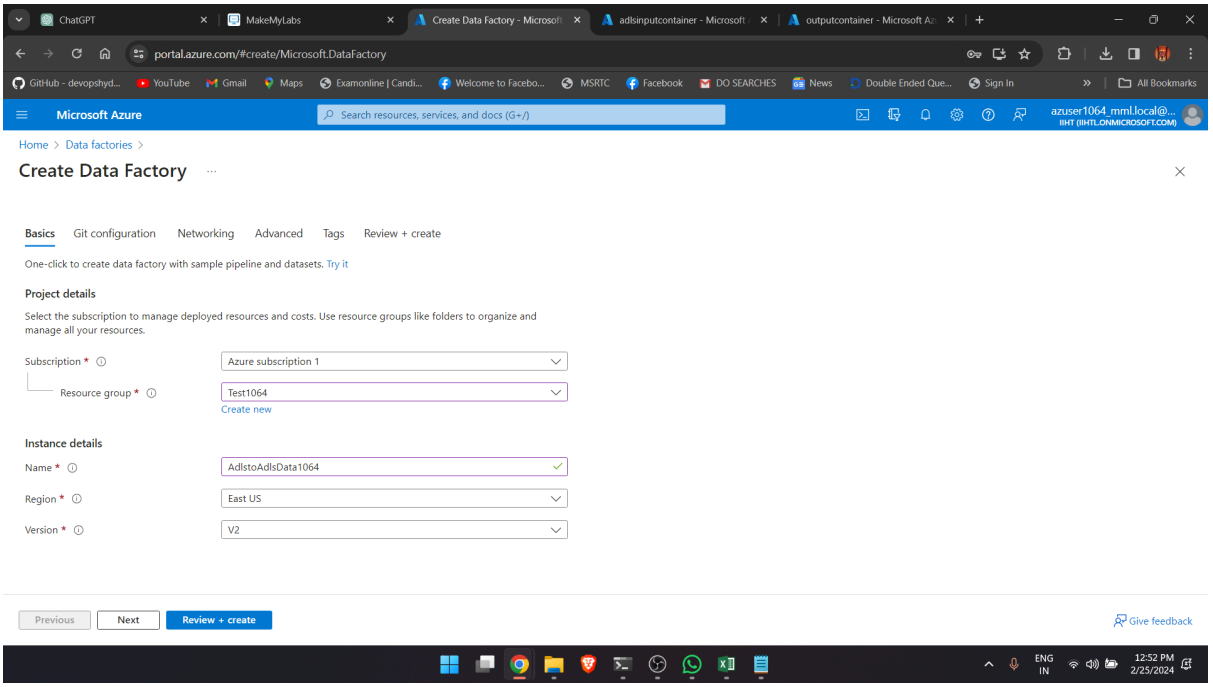

ChatGPT
MakeMyLabs
Home - Microsoft Azure
adlsinputcontainer - Microsoft
adlsoutput1064 - Microsoft Azu
portal.azure.com/#@iihtl.onmicrosoft.com/resource/subscriptions/984f097c-963c-4eb6-a20d-839457ae9f08/resourceGroups/Test1064/providers/Microsoft.Storage/storag...
Microsoft Azure
Search resources, services, and docs (G+/)
azuser1064_mml.local@...
IIHT
Home >
adlsoutput1064
Storage account
Search
Overview
Activity log
Tags
Diagnose and solve problems
Access Control (IAM)
Data migration
Events
Storage browser
Data storage
Containers
File shares
Queues
Tables
Security + networking
Networking
Access keys
Upload
Open in Explorer
Delete
Move
Refresh
Open in mobile
CLI / PS
Feedback
Essentials
JSON View
Resource group (move) : Test1064
Location : eastus
Primary/Secondary Location : Primary: East US, Secondary: West US
Subscription (move) : Azure subscription 1
Subscription ID : 984f097c-963c-4eb6-a20d-839457ae9f08
Disk state : Primary: Available, Secondary: Available
Performance : Standard
Replication : Read-access geo-redundant storage (RA-GRS)
Account kind : StorageV2 (general purpose v2)
Provisioning state : Succeeded
Created : 2/24/2024, 3:13:20 PM
Tags (edit) : Add tags
Properties
Monitoring
Capabilities (5)
Recommendations (0)
Tutorials
Tools + SDKs
Data Lake Storage
Hierarchical namespace Enabled
Default access tier Hot
Blob anonymous access Disabled
Blob soft delete Enabled (7 days)
Container soft delete Enabled (7 days)
Versioning Disabled
Security
Require secure transfer for REST API operations Enabled
Storage account key access Enabled
Minimum TLS version Version 1.2
Infrastructure encryption Disabled
Networking

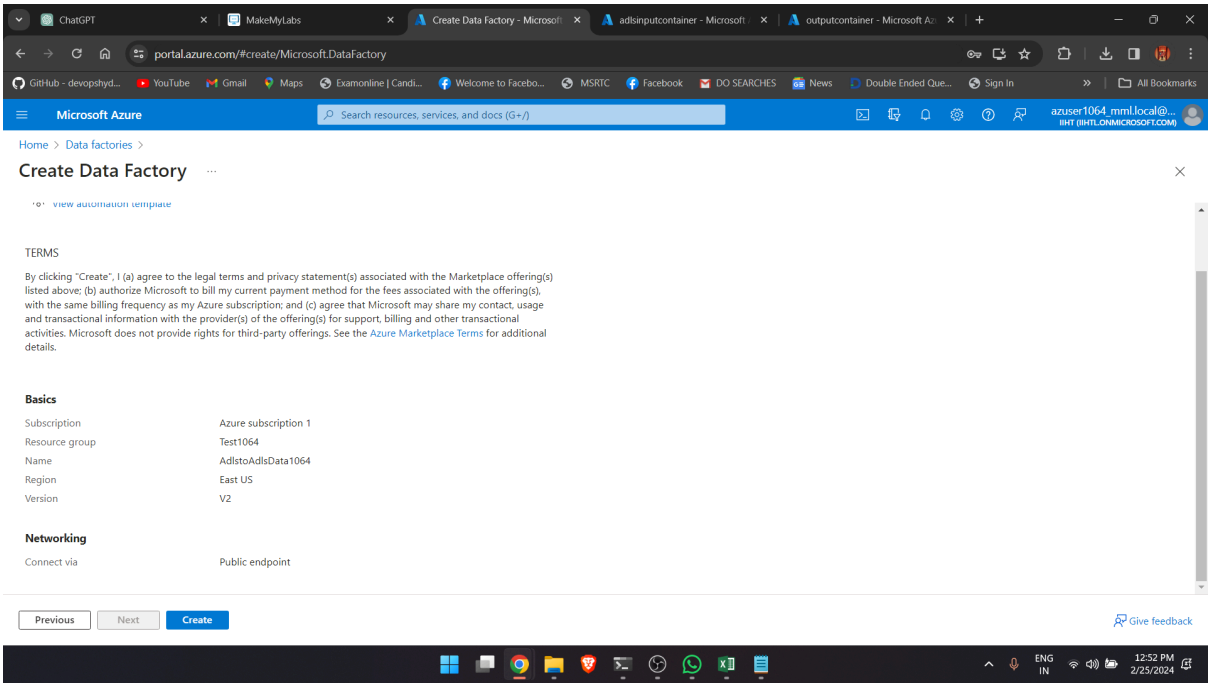
ENG
IN
12:51 PM
2/25/2024

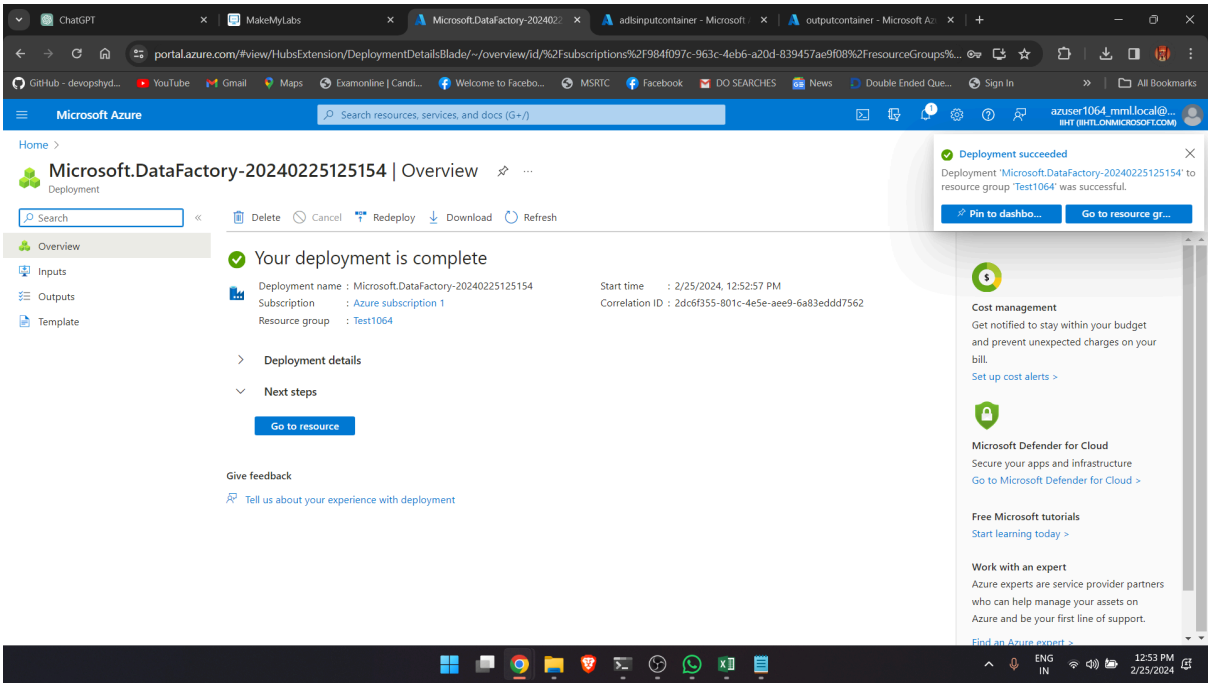

ChatGPT
MakeMyLabs
Home - Microsoft Azure
adlsinputcontainer - Microsoft
adlsoutput1064 - Microsoft Azu
portal.azure.com/#@iihtl.onmicrosoft.com/resource/subscriptions/984f097c-963c-4eb6-a20d-839457ae9f08/resourceGroups/Test1064/providers/Microsoft.Storage/storag...
Microsoft Azure
Search resources, services, and docs (G+/)
azuser1064_mml.local@...
IIHT
Home > adlsoutput1064
adlsoutput1064 | Containers
Storage account
Search
Overview
Activity log
Tags
Diagnose and solve problems
Access Control (IAM)
Data migration
Events
Storage browser
Data storage
Containers
File shares
Queues
Tables
Security + networking
Networking
Access keys
+ Container
Change access level
Restore containers
Refresh
Delete
Give feedback
Search containers by prefix
Show deleted containers
Name
Last modified
Anonymous access level
Lease state
$logs
2/24/2024, 3:13:43 PM
Private
Available
outputcontainer
2/24/2024, 3:14:24 PM
Private
Available
ENG
IN
12:51 PM
2/25/2024

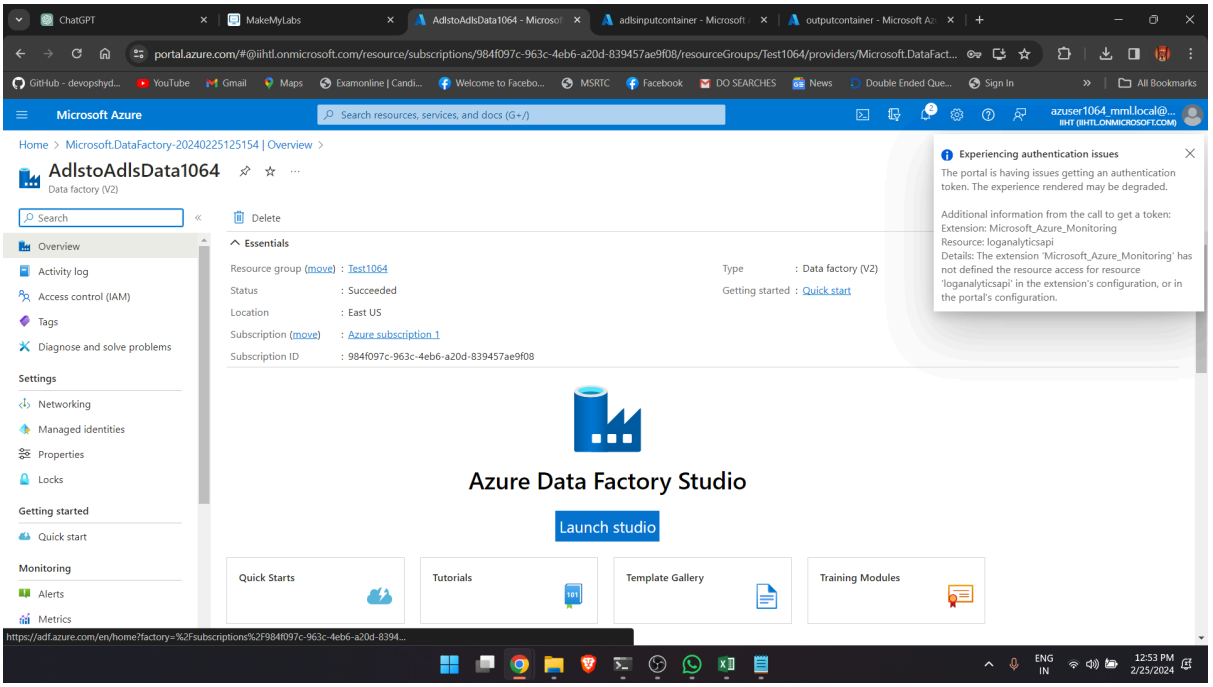

ChatGPT
MakeMyLabs
Home - Microsoft Azure
adlsinputcontainer - Microsoft
outputcontainer - Microsoft Az
portal.azure.com/#view/Microsoft_Azure_Storage/ContainerMenuBlade/~/overview/storageAccountId/%2Fsubscriptions%2F984f097c-963c-4eb6-a20d-839457ae9f08%2F...
Microsoft Azure
Search resources, services, and docs (G+/)
azuser1064_mml.local@...
IIHT
Home > adlsoutput1064 | Containers >
outputcontainer
Container
Search
Overview
Diagnose and solve problems
Access Control (IAM)
Settings
Shared access tokens
Manage ACL
Access policy
Properties
Metadata
Upload
Add Directory
Refresh
Rename
Delete
Change tier
Acquire lease
Break lease
Give feedback
Authentication method: Access key (Switch to Microsoft Entra user account)
Location: outputcontainer
Search blobs by prefix (case-sensitive)
Show deleted objects
Name
Modified
Access tier
Archive status
Blob type
Size
Lease state
No results
ENG
IN
12:51 PM
2/25/2024

Created new Data Factory account for creating pipeline to copy data from input to output container:

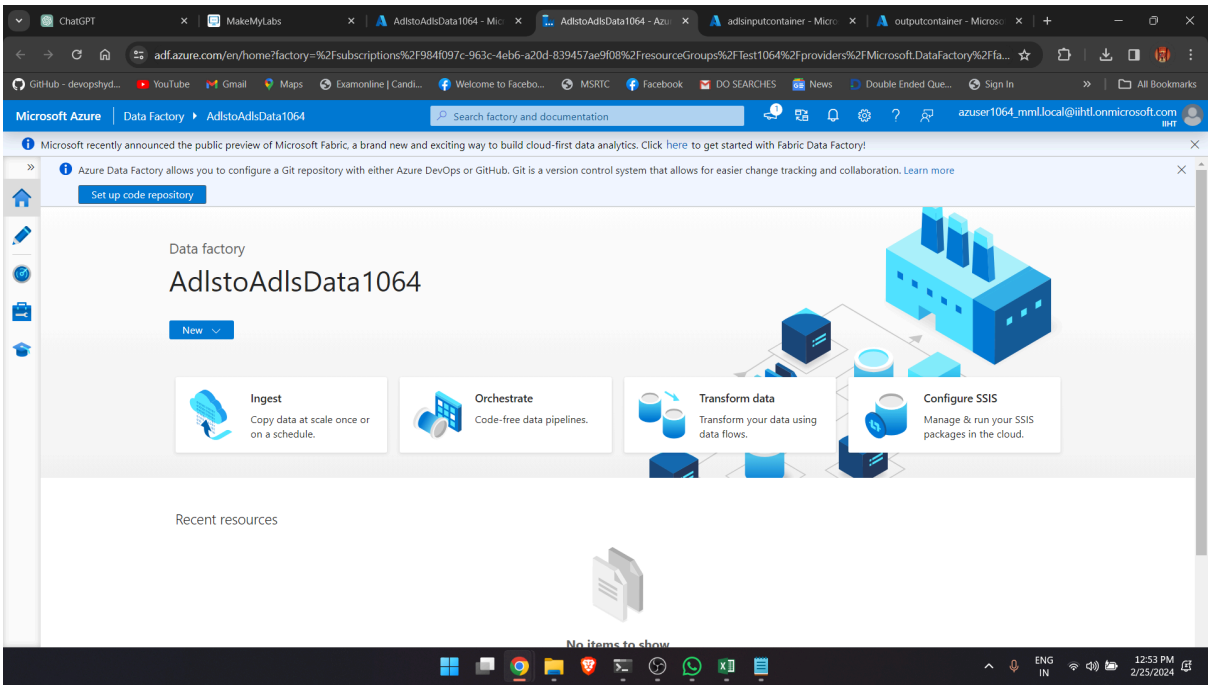


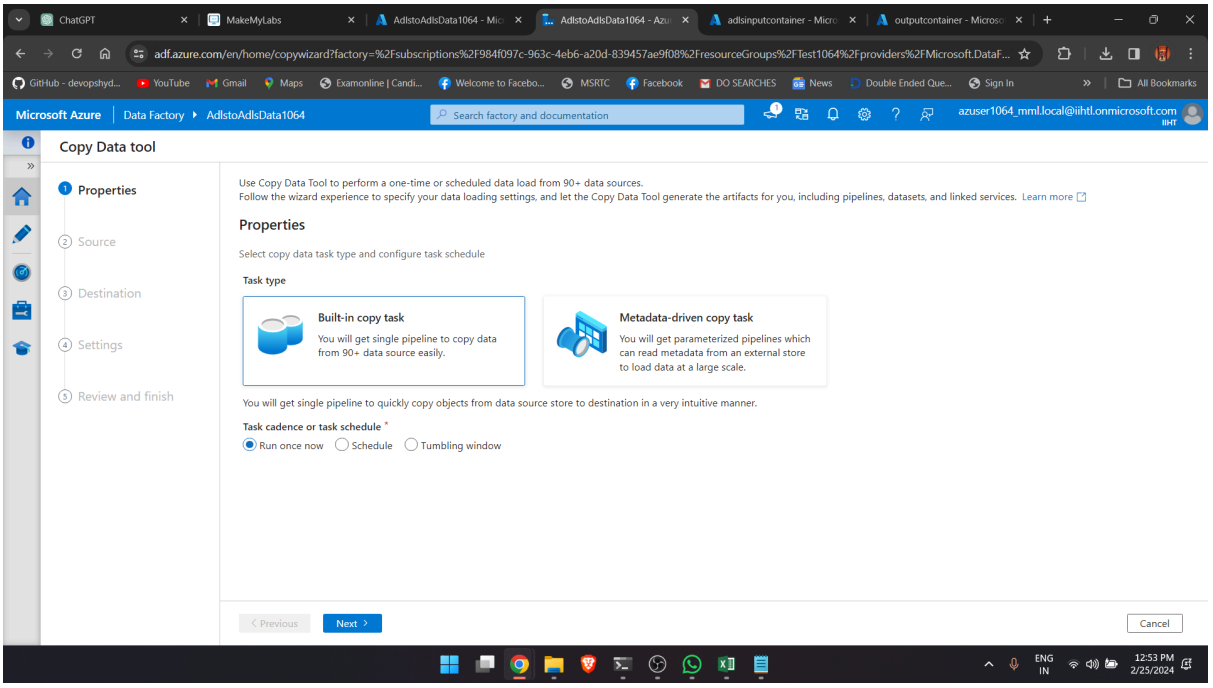



Microsoft Azure
Search resources, services, and docs (G+/)
Home >
Microsoft.DataFactory-20240225125154 | Overview
Deployment
Search
Overview
Inputs
Outputs
Template
Delete
Cancel
Redeploy
Download
Refresh
Your deployment is complete
Deployment name : Microsoft.DataFactory-20240225125154
Subscription : Azure subscription 1
Resource group : Test1064
Start time : 2/25/2024, 12:52:57 PM
Correlation ID : 2dc6f355-801c-4e5e-aee9-6a83eddd7562
Deployment details
Next steps
Go to resource
Give feedback
Tell us about your experience with deployment
Deployment succeeded
Deployment 'Microsoft.DataFactory-20240225125154' to resource group 'Test1064' was successful.
Pin to dashbo...
Go to resource gr...
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 >
Work with an expert
Azure experts are service provider partners who can help manage your assets on Azure and be your first line of support.
Find an Azure expert >

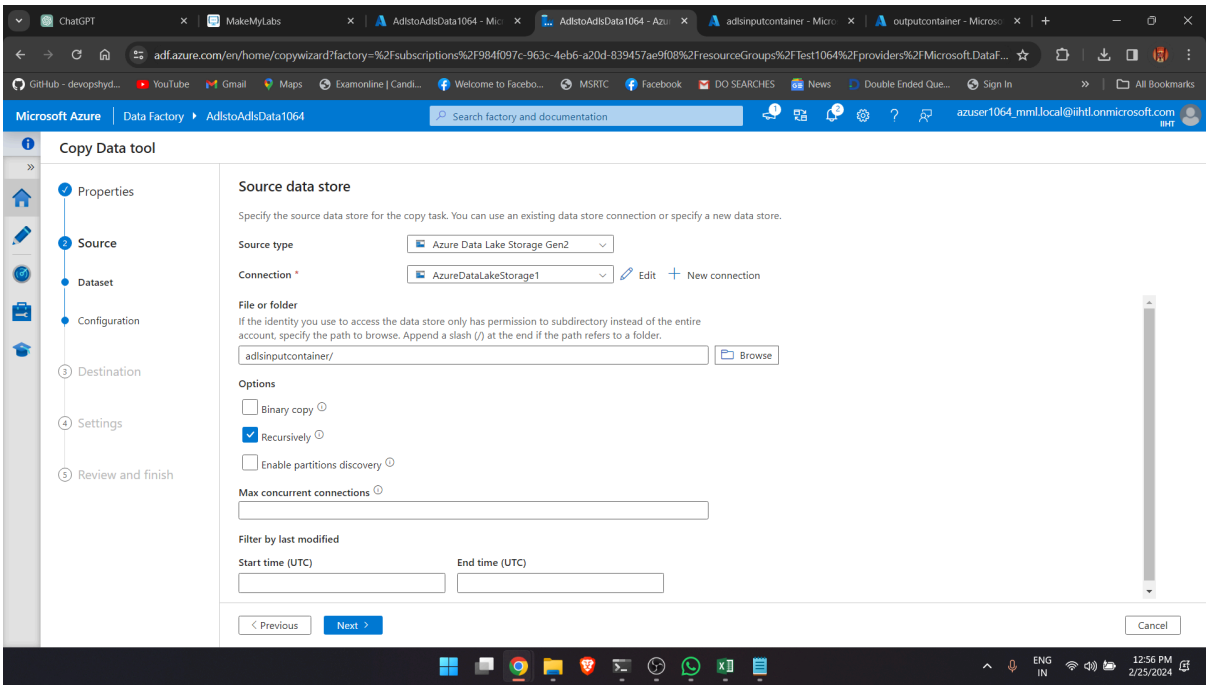

Microsoft Azure
Search resources, services, and docs (G+/)
Home > Microsoft.DataFactory-20240225125154 | Overview >
AdlstoAdlsData1064
Data factory (V2)
Search
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Settings
Networking
Managed identities
Properties
Locks
Getting started
Quick start
Monitoring
Alerts
Metrics
Delete
Essentials
Resource group (move) : Test1064
Status : Succeeded
Location : East US
Subscription (move) : Azure subscription 1
Subscription ID : 984f097c-963c-4eb6-a20d-839457ae9f08
Type : Data factory (V2)
Getting started : Quick start
Azure Data Factory Studio
Launch studio
Quick Starts
Tutorials
Template Gallery
Training Modules
Experiencing authentication issues
The portal is having issues getting an authentication token. The experience rendered may be degraded.
Additional information from the call to get a token:
Extension: Microsoft_Azure_Monitoring
Resource: loganalyticsapi
Details: The extension 'Microsoft_Azure_Monitoring' has not defined the resource access for resource 'loganalyticsapi' in the extension's configuration, or in the portal's configuration.

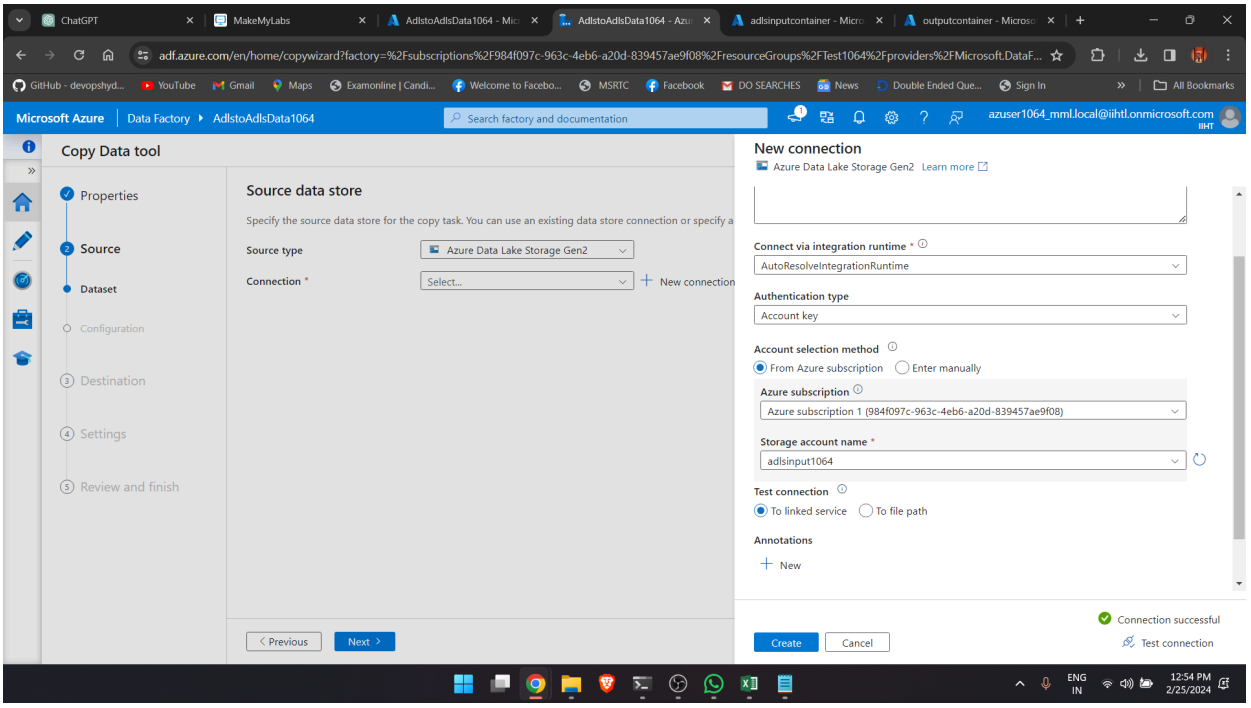
Using Ingest service for using built-in copy method :

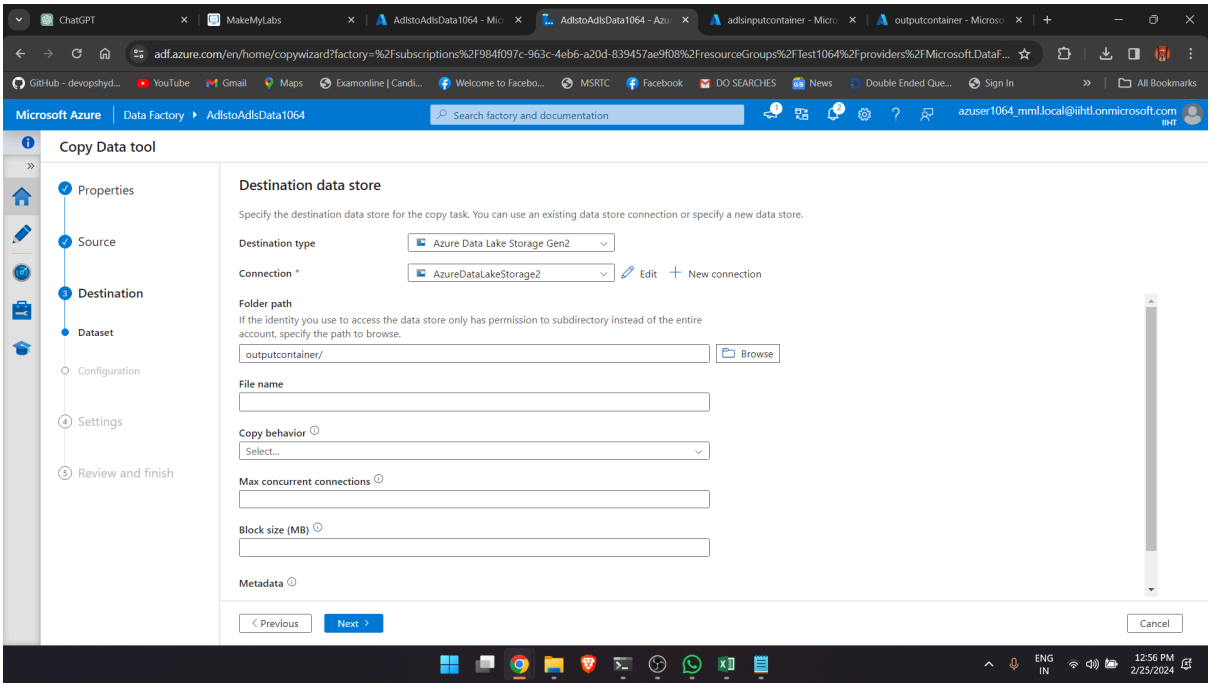




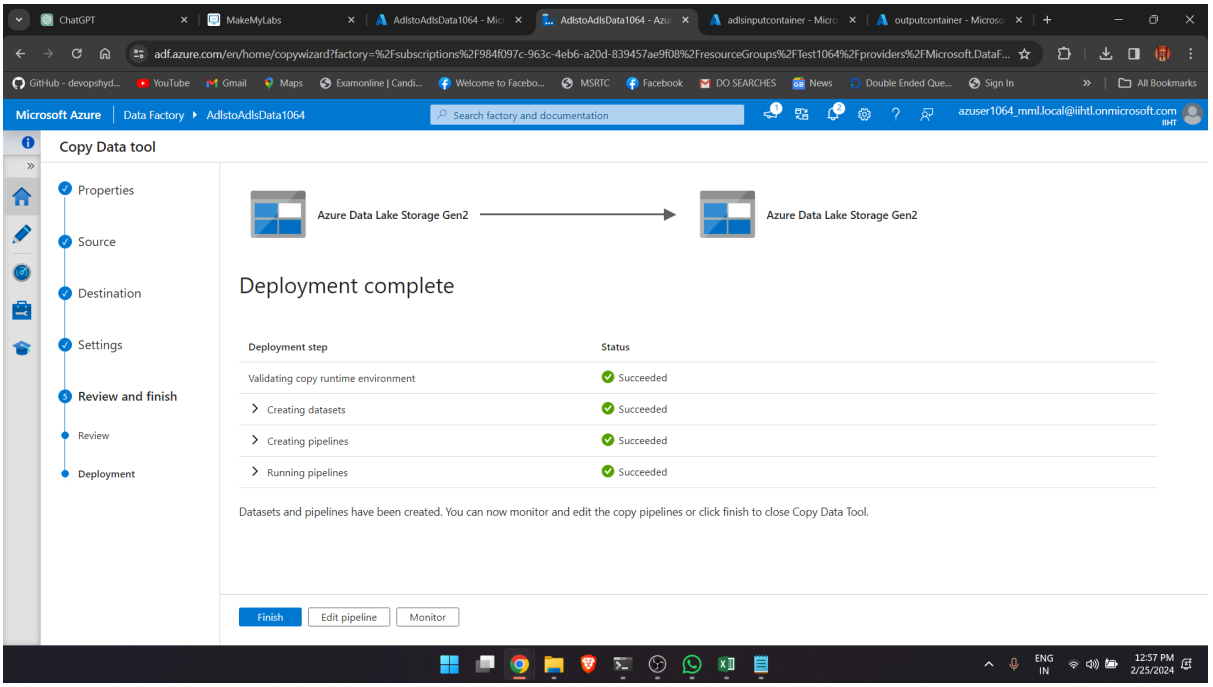
Giving input and output paths for copying data:

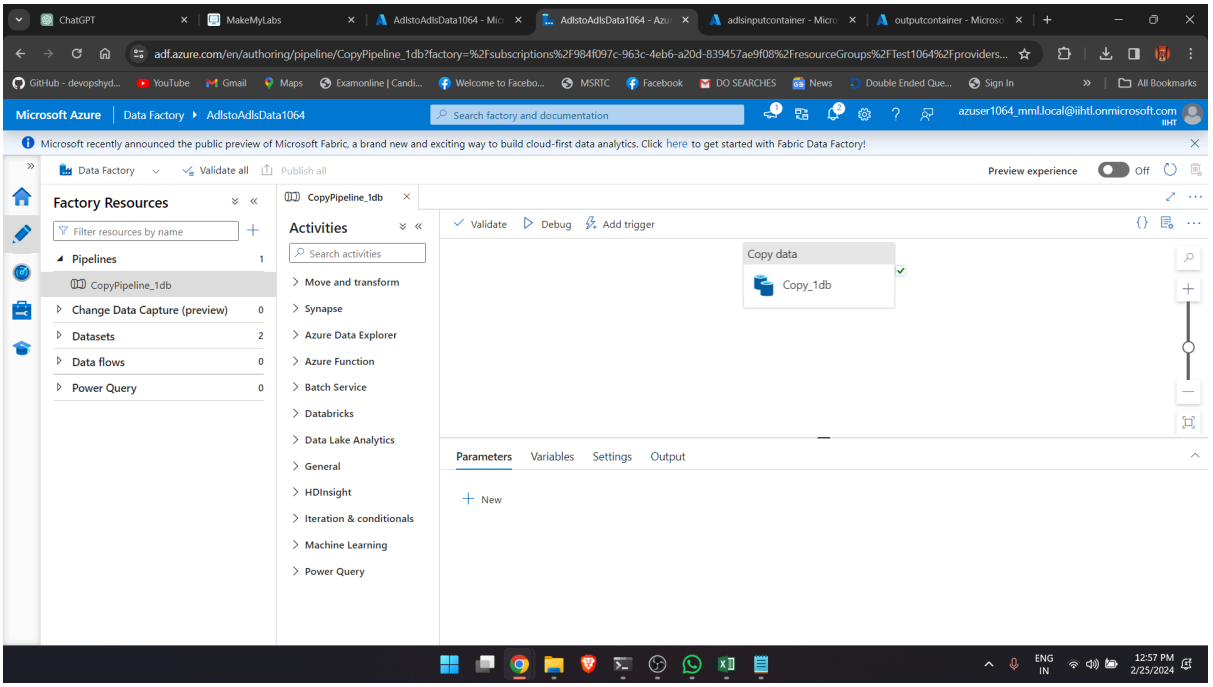






Creating pipeline :





Checking weather data is copied in output container or not :