

Project-1 (Data Engineering Batch-1)

Ingestion and Transformation of Olympics Data



Vuthur Sriganga

Hexaware Data Engineering Batch-1

vuthursriganga@gmail.com

Ingestion and Transformation of Olympics Data

Title:

Data Ingestion: Configure and run the ADF pipelines to ingest the Olympics data into Azure Data Lake Storage Gen2.

Data Transformation: Execute the PySpark notebooks in Azure Databricks to perform data transformation tasks.

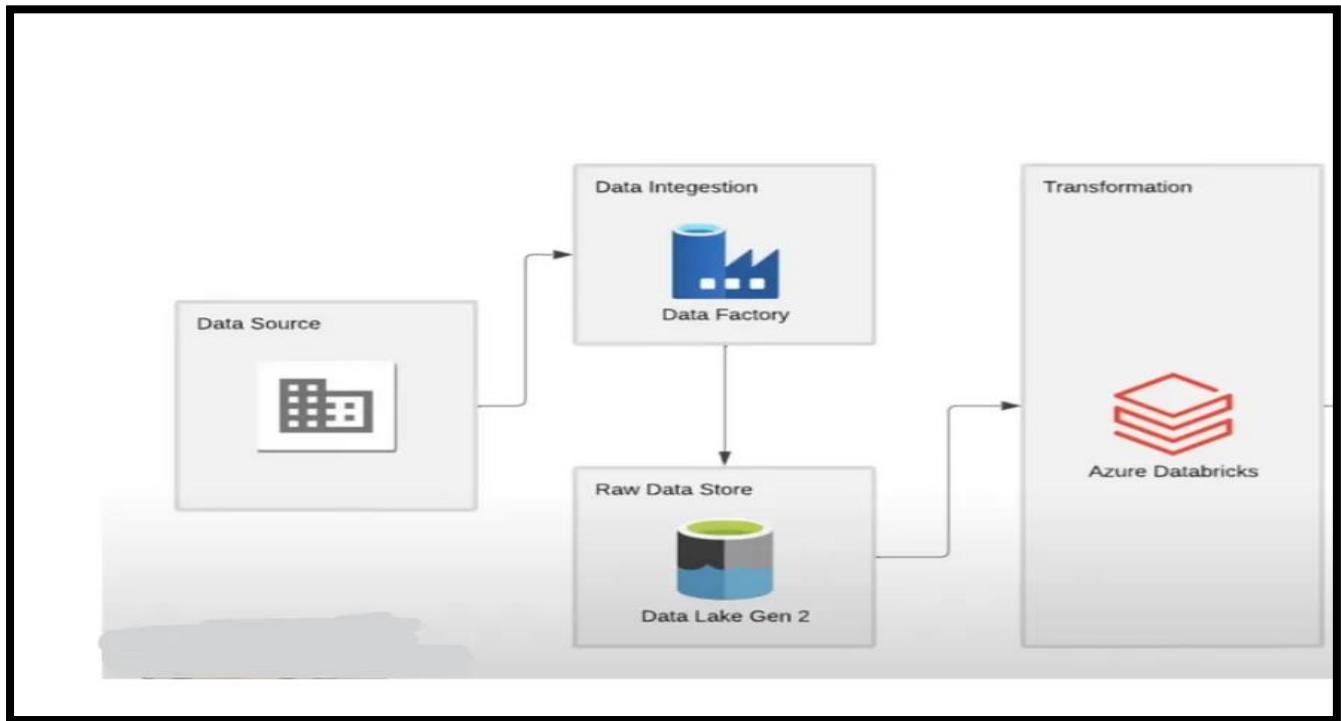
Project Overview:

This project is designed to showcase how various Azure services can be utilized to ingest, store and transform Olympics-related data. This project can configure and run Azure Data Factory pipelines to ingest Olympics data into Azure Data Lake Storage Gen2.

About the Project:

The main objective of this project is to ingest Olympics data into Azure Data Lake Storage Gen2 (ADLS Gen2) using Azure Data Factory (ADF) pipelines and perform data transformation tasks using PySpark notebooks in Azure Databricks.

Architecture Diagram:



This architecture diagram shows about ingesting and transforming the Olympics data using Azure Data Factory , Azure Data Lake Storage Gen2 and Azure Data Bricks.

How it works:

1. Data Ingestion:

1) Source Data Identification:

- Identify the sources of Olympics data.
- These sources could include databases, Azure storage accounts, or any other data repositories where the Olympics data is stored.

2) Azure Data Lake Storage Gen2 Configuration:

- Set up Azure Data Lake Storage Gen2 (ADLS Gen2) as the destination for storing the Olympics data.
- Go to Azure Portal and search for Storage Accounts.
- Click on + to create. Add the Subscription , Resource group name , storage account name
- Enable Hierarchical Namespace to organize data in a hierarchical structure, improving performance and manageability in Advanced setting
- Click on Create
- Deployment is in progress.
- Once the deployment is complete Click on Go to Resource.

3) Azure Data Factory (ADF) Pipeline Configuration:

- Create a Azure data factory by selecting the Azure subscription,Resource group and give a name for data factory and Click on Create.
- Once the deployment is complete, Click on Go to Resource
- Click on Launch Studio.
- Click on Ingest Data
- Give the details of Source and Destination to perform Copy Activity .

4) Run the Pipeline:

- Once the pipeline is configured, trigger the pipeline execution manually or let it run based on the defined schedule.
- Monitor the pipeline execution for any errors or issues and troubleshoot as needed.
- Verify that the data has been successfully ingested into ADLS Gen2.

2. Data Transformation:

1) Azure Databricks Environment Setup:

- Set up an Azure Databricks workspace in the Azure portal.
- Configure clusters within the Databricks workspace with appropriate specifications for running PySpark notebooks.
- Specify the cluster type, instance types, number of instances, and auto-scaling settings based on the workload requirements.
- It is a best practice to use personal compute to reduce the costing.

2) PySpark Notebook Development:

- Create a new notebook within the Azure Databricks workspace.
- Set the configuration to Azure Data Lake Storage gen2 account by providing name of the storage account, name of container and key.
- Develop PySpark notebooks to perform various data transformation tasks on the ingested Olympics data. Tasks may include aggregation, filtering, distinct operations, data normalization, joining datasets, etc.
- Write PySpark code to implement the desired data transformation logic.
- Ensure that the code is well-documented and includes comments for clarity and maintainability.
- Test the PySpark code within the notebook to verify that it produces the expected output.

3) Notebook Execution and Scheduling:

- Execute the developed PySpark notebooks within the Azure Databricks environment to perform data transformation tasks on the ingested Olympics data.
- Schedule the execution of notebooks at specified intervals or triggers to ensure regular updates to the transformed data.
- Configure notebook parameters and input/output paths as necessary for scheduled execution.
- Review the transformed data output to ensure it meets the desired transformation requirements.

Azure Resources / tools used for this Project:

1) Azure Data Lake Storage Gen2 (ADLS Gen2):

- ADLS Gen2 is the storage solution where the Olympics data will be ingested.
- ADLS Gen2 provides a scalable and cost-effective storage solution for big data analytics workloads.

2) Azure Data Factory (ADF):

- Azure Data Factory is a cloud-based data integration service that allows us to create, schedule, and orchestrate data pipelines to move and transform data across various data stores.
- In this project, Azure Data Factory will be used to configure and run pipelines for ingesting Olympics data into ADLS Gen2.

3) Azure Databricks:

- Azure Databricks is a powerful analytics platform that enables organizations to use the capabilities of Apache Spark for processing big data in a collaborative and scalable environment.
- PySpark notebooks are be used in Azure Databricks to perform data transformation tasks like filtering ,group by and aggregations, selecting distinct data, renaming columns etc., on the ingested Olympics data.

4) Python with PySpark:

PySpark is the Python API for Apache Spark, which is a fast and general-purpose cluster computing system. PySpark would be used within Azure Databricks notebooks for developing and executing data transformation tasks.

5) Azure Portal:

The Azure Portal is a web-based interface for managing Azure services. It would be used for provisioning and configuring resources such as Azure Data Lake Storage Gen2, Azure Data Factory, and Azure Databricks.

6) Microsoft Excel :

Microsoft Excel might be used for data preparation or as a data source for creating input datasets for the project.

Project Requirements:

1) Data Ingestion:

- Source Data Identification
- Azure Data Lake Storage Gen2 Configuration
- Azure Data Factory (ADF) Pipeline Configuration for Copy Activity
- Debugging ADF pipeline

2) Data Transformation:

- Azure Databricks Environment Setup
- PySpark Databricks Notebook
- Notebook Execution

Tasks performed:

Data Ingestion with ADF Pipelines:

- Configure Azure Data Factory (ADF) pipelines to ingest Olympics data.
- Define the necessary data sources and sinks, specifying Azure Data Lake Storage Gen2 as the destination.
- Set up activities within the pipeline to orchestrate the data movement.
- Schedule or trigger the pipeline execution as per our requirements.

Data Transformation with PySpark Notebooks in Azure Databricks:

- Utilize Azure Databricks for data transformation using PySpark.
- Create Azure Databricks workspace in Azure portal.
- Develop PySpark notebooks to perform specific transformation tasks like sorting, filtering, aggregations, distinct columns etc., on the ingested Olympics data.
- Leverage the scalability of Databricks for efficient distributed processing.
- Execute the notebooks, ensuring they handle the data according to your transformation logic.

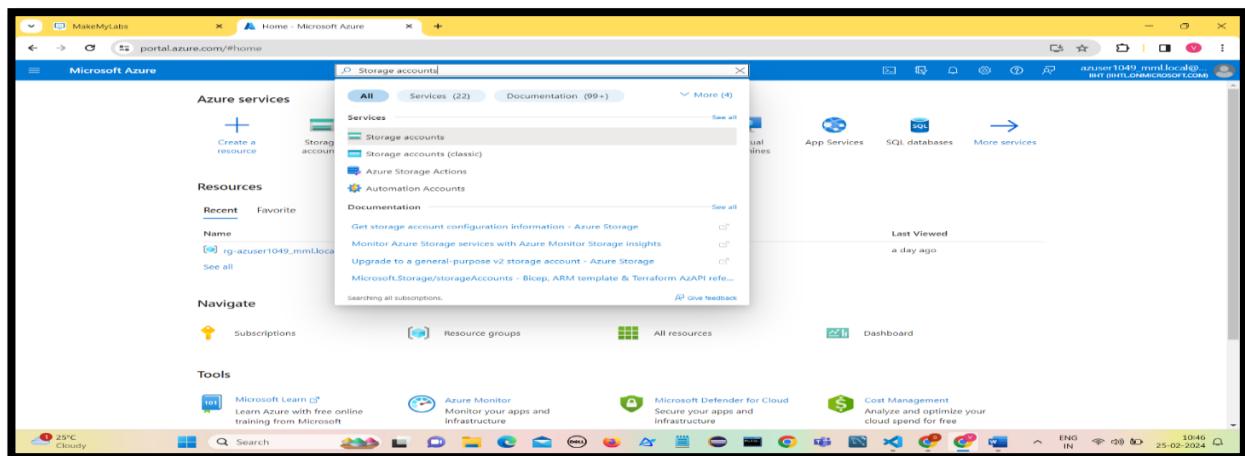
Overall Execution Flow:

- Initiate the Azure Data Factory pipeline to ingest raw Olympics data into Azure Data Lake Storage Gen2.
- Once ingestion is complete, trigger/run the PySpark notebooks in Azure Databricks for data transformation.
- PySpark notebooks read data from Data Lake Storage, apply transformations on the data.
- Monitor pipeline and notebook executions for any errors or performance issues.
- Schedule these processes to run at suitable intervals to keep data up-to-date.

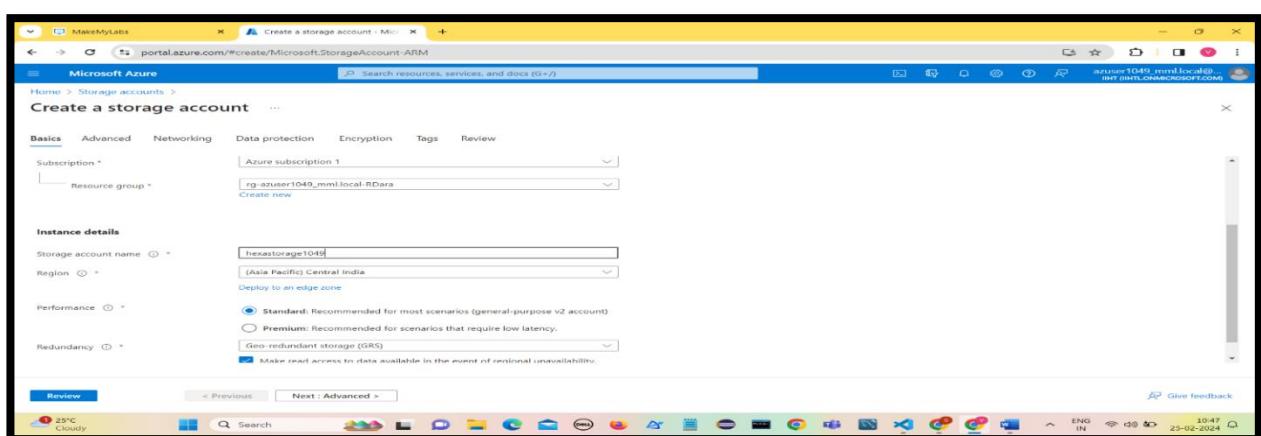
Result :

1) Data Ingestion

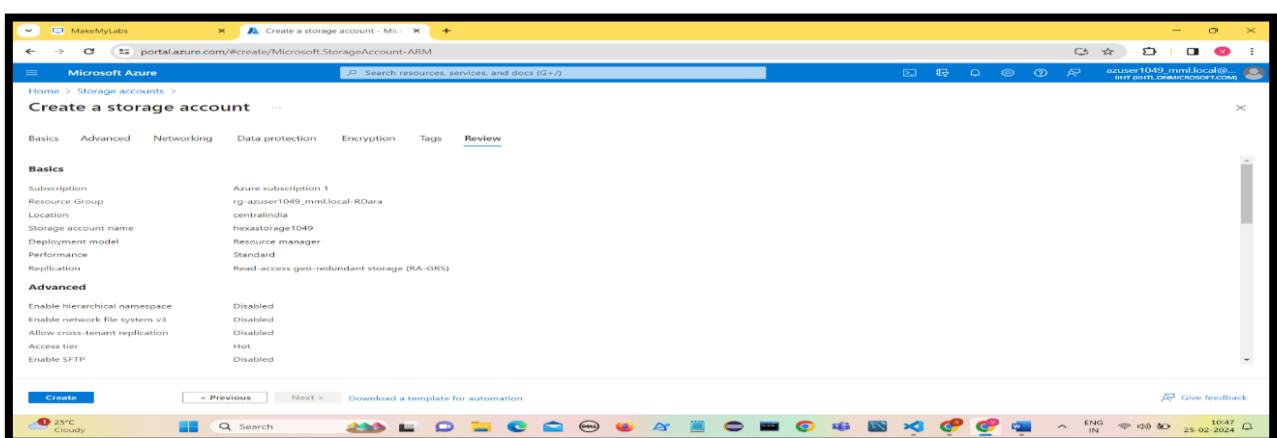
Go to Azure Portal and Search for Storage Accounts.



To create a Storage Account fill in all the details and Click on Review.



After verifying all the details of Storage Account , Click on Create.



Once the deployment is complete, Click on Go to Resource.

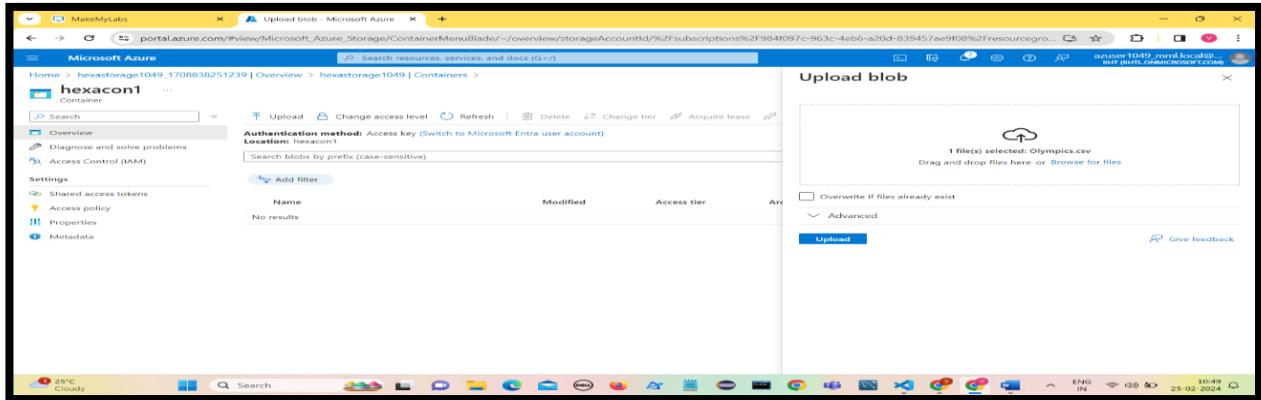
The screenshot shows the Microsoft Azure portal's Deployment Overview page for a deployment named "hexastorage1049_1708838251239". The status is "Deployment succeeded" with a green checkmark icon. The deployment details include the deployment name, subscription, resource group, start time (2/25/2024, 10:47:33 AM), and correlation ID. A "Go to resource" button is visible. The page also features links for Cost Management, Microsoft Defender for Cloud, Free Microsoft tutorials, and Work with an expert.

The screenshot shows the Microsoft Azure portal's Storage Account Overview page for "hexastorage1049". The account type is "Storage account". The "Essentials" section displays various properties such as resource group, location, primary/secondary locations, subscription, disk state, and tags. The "Properties" tab is selected, showing blob service settings like hierarchical namespace, default access tier, and blob soft delete. The "Security" tab shows settings for secure transfer, storage account key access, minimum TLS version, and infrastructure encryption. The "Networking" tab is also visible.

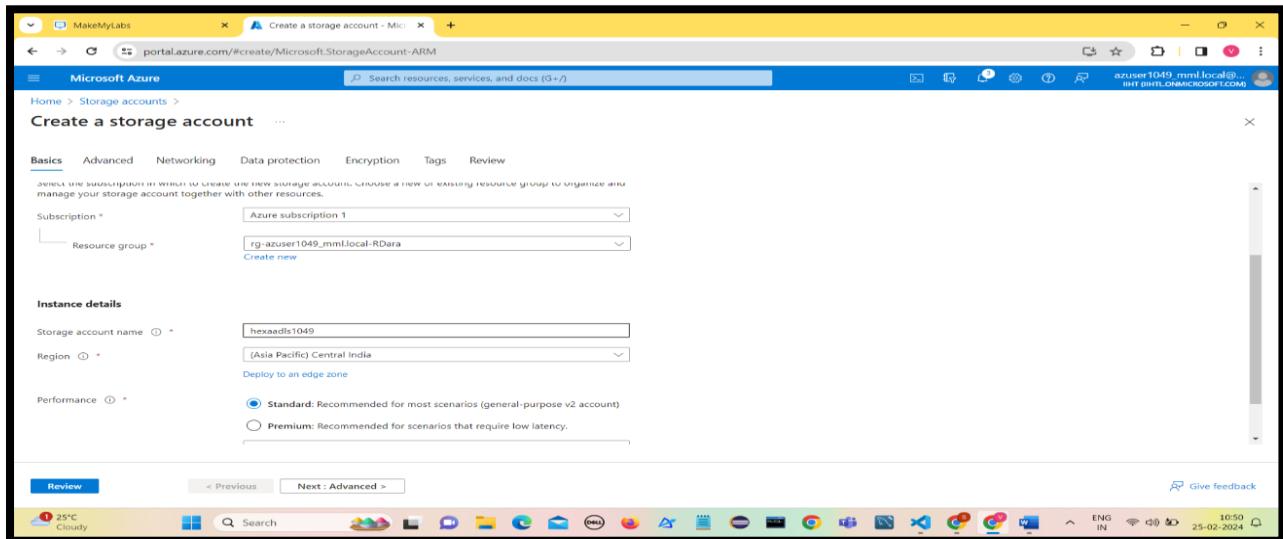
Create a container (hexacon1) in hexastorage1049 storage account

The screenshot shows the Microsoft Azure portal's "New container" dialog box. It is titled "New container" and has a "Name" field set to "hexacon1". The "Anonymous" dropdown is set to "Private (no anonymous access)". A note indicates that the access level is set to private because anonymous access is disabled on this storage account. The "Create" button is at the bottom right.

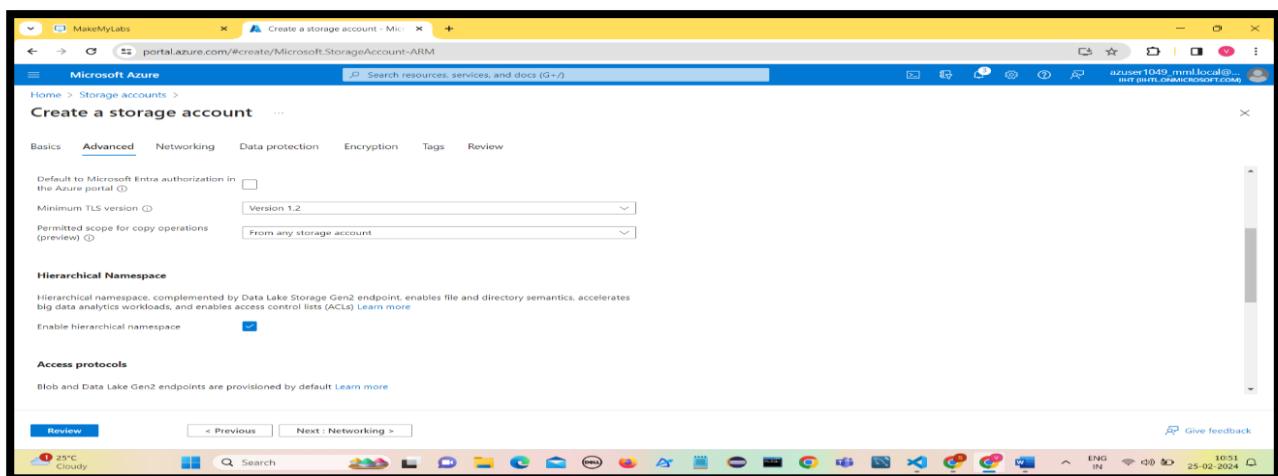
Upload Olympics.csv file (blob) into the container



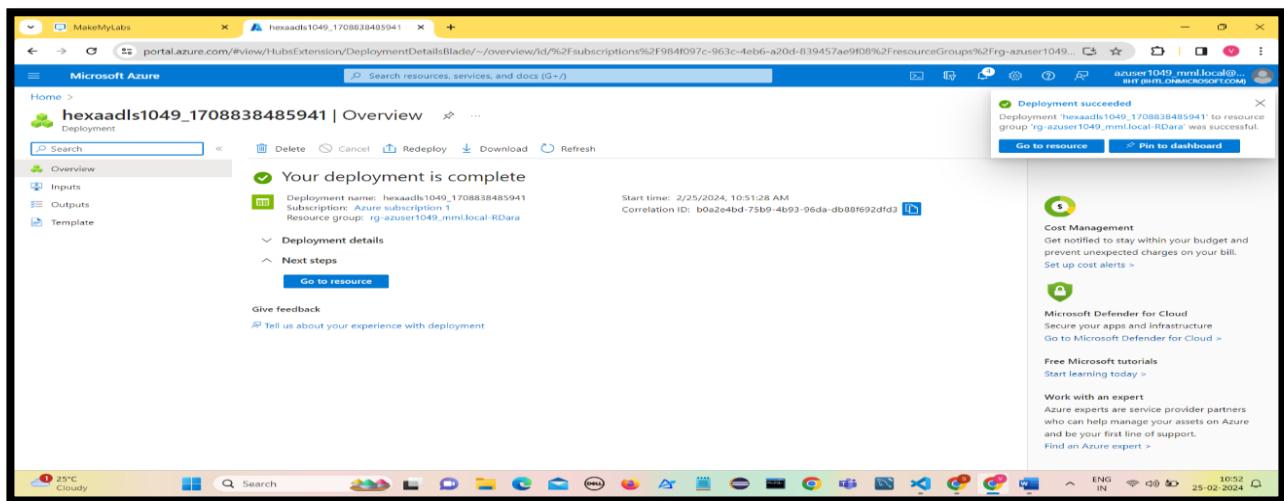
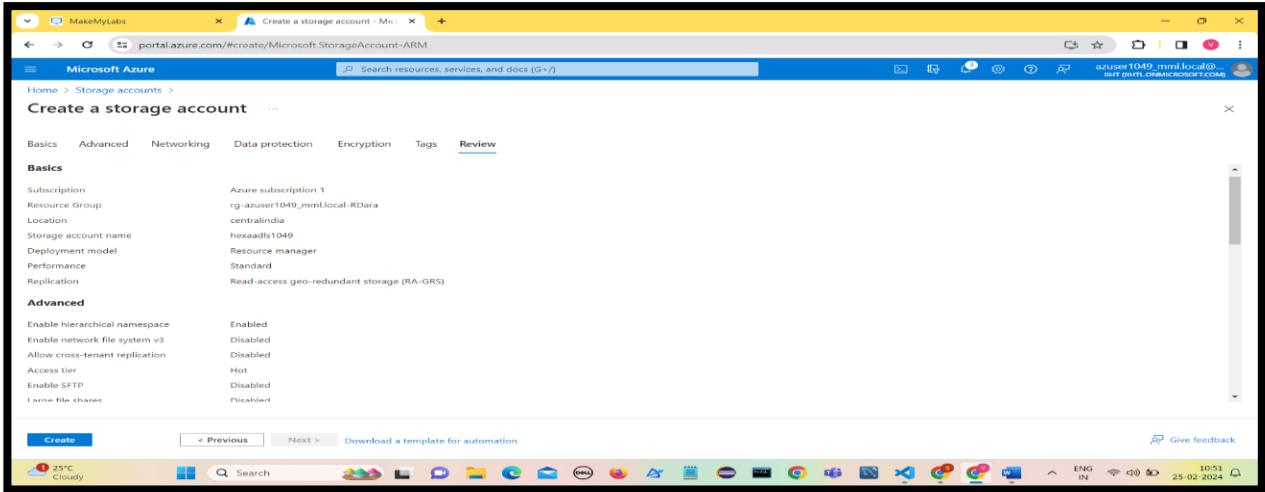
Creating an ADLS gen2 account (hexaadls1049)



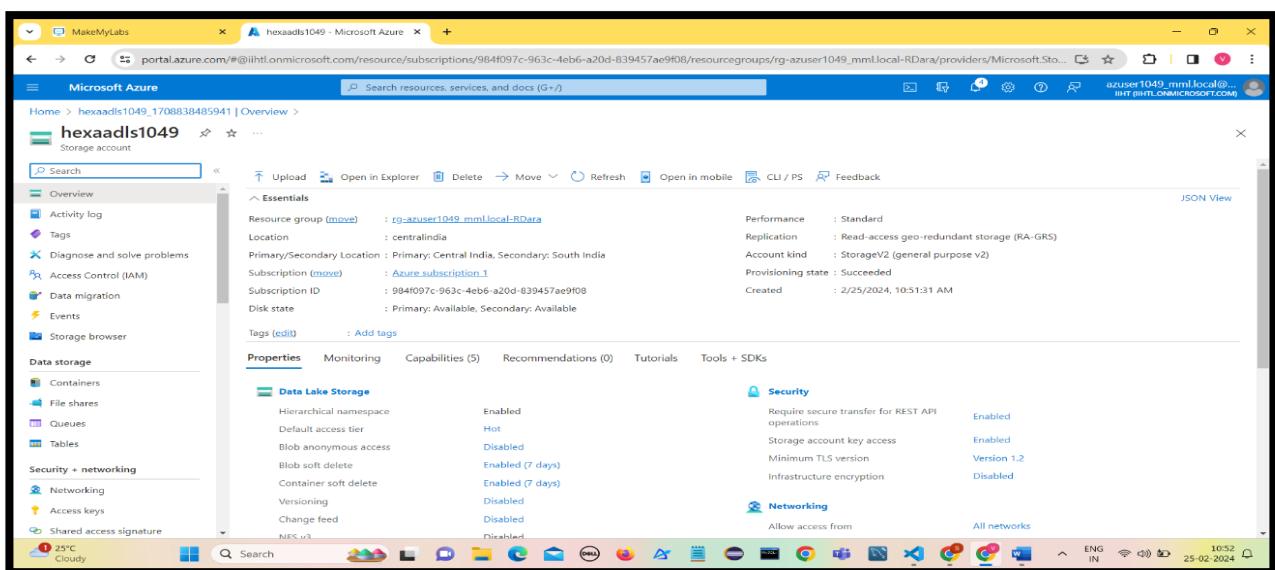
In Advanced , Enable Hierarchical Namespace to make it ADLS gen2 Account



Verify all the details of Storage account and Click on Create



Once the deployment is complete , Click on Go to Resource.



Create a container hexacon2

The screenshot shows the Microsoft Azure portal interface. On the left, the navigation menu is open, showing 'Containers' under 'Data storage'. In the center, a list of existing containers is shown, with one named 'logs'. On the right, a 'New container' dialog box is open, prompting for a name ('hexacon2'), setting the 'Anonymous access level' to 'Private (no anonymous access)', and providing an optional note about anonymous access being disabled. At the bottom right of the dialog is a 'Create' button.

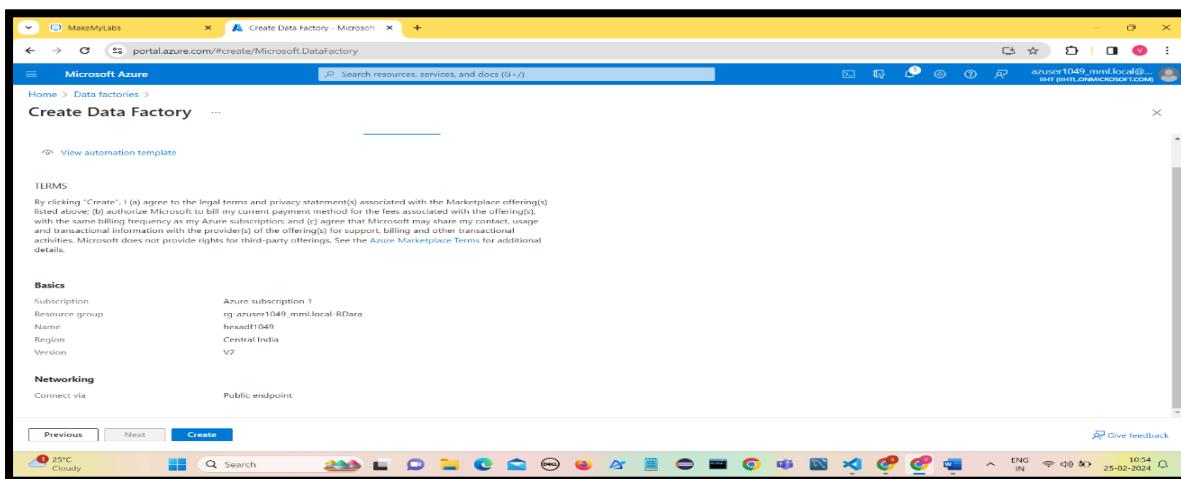
Create a Azure Data Factory (hexadf1049)

The screenshot shows the Microsoft Azure portal interface. The left sidebar is expanded to show 'Data factories'. The main area displays a list of 104 data factories, each with a name, type (e.g., 'Data factory (V2)'), subscription, resource group, location, and a three-dot ellipsis icon for more options. The data is paginated, showing page 2 of 2. At the bottom right of the screen, there is a status bar showing 'Cloudy' weather, the date '25-02-2024', and the time '10:53'.

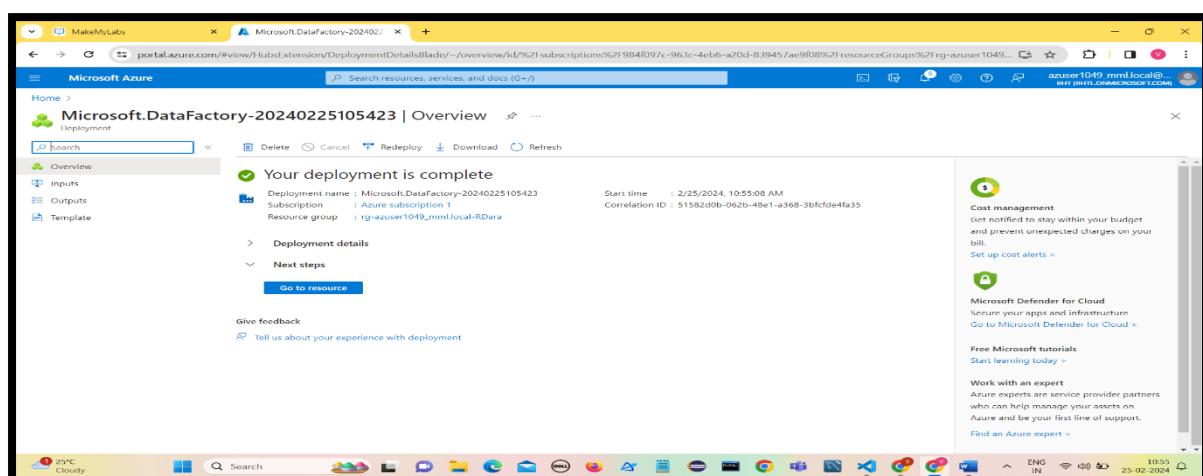
Click on Review+Create

The screenshot shows the Microsoft Azure portal interface for creating a new Data Factory. The top navigation bar indicates the task is 'Create Data Factory - Microsoft'. The main area is titled 'Create Data Factory' and includes tabs for 'Basics', 'Git configuration', 'Networking', 'Advanced', 'Tags', and 'Review + create'. The 'Basics' tab is selected. It contains fields for 'Subscription' (set to 'Azure subscription 1' and 'rg-azuser1049_mml.local-RDara'), 'Resource group' (set to 'rg-azuser1049_mml.local-RDara'), and 'Name' (set to 'hexadf1049'). Below these are 'Region' (set to 'Central India') and 'Version' (set to 'V2'). At the bottom of the form are buttons for 'Previous', 'Next', and 'Review + create'. The status bar at the bottom right shows 'Cloudy' weather, the date '25-02-2024', and the time '10:54'.

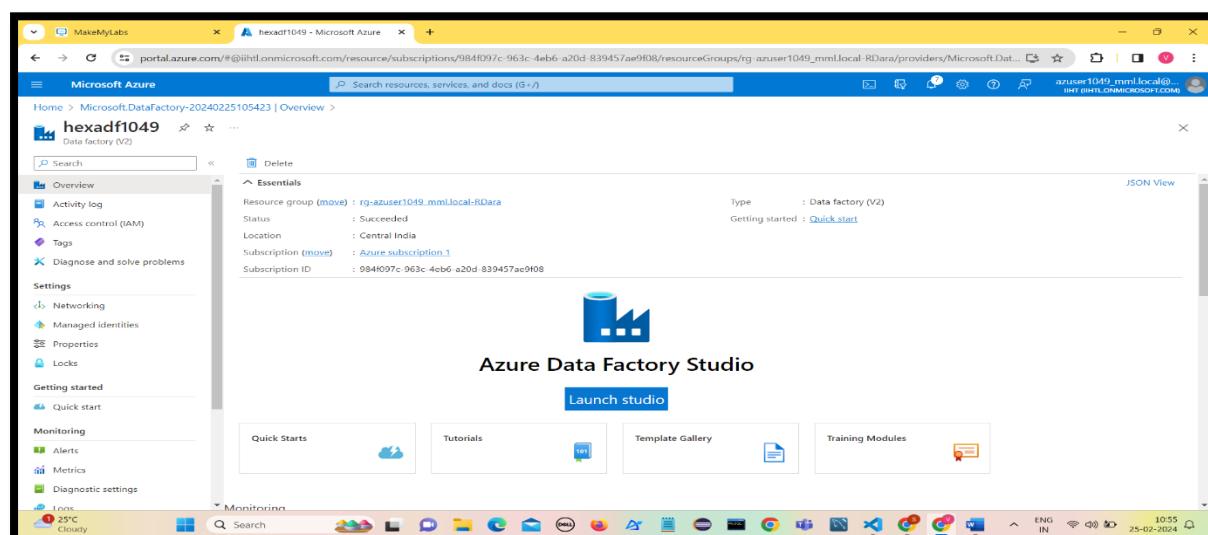
Verify the details and Click on Create



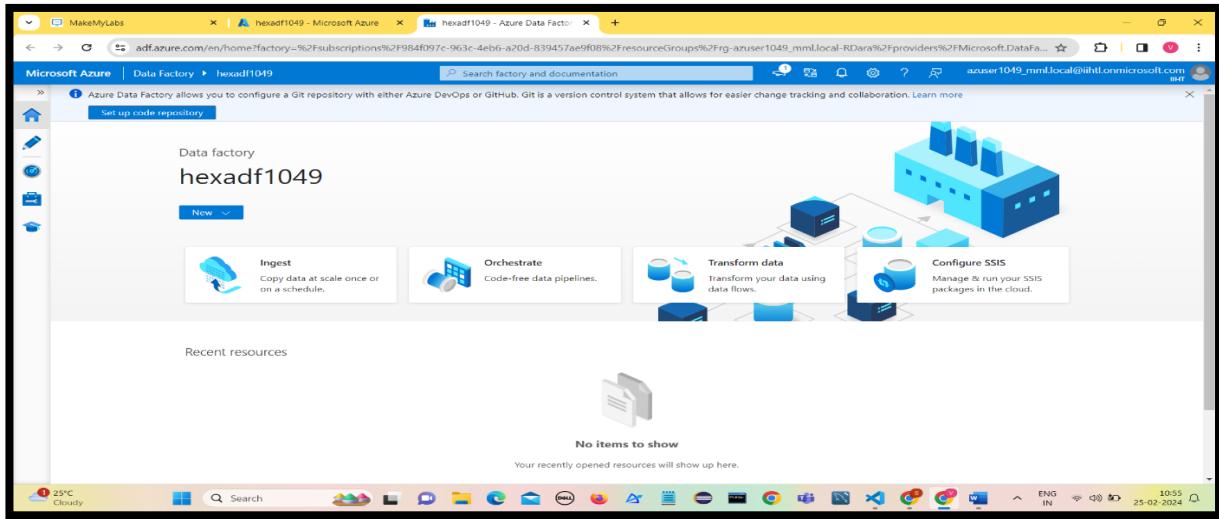
Once after deployment is complete, Click on Go to Resource



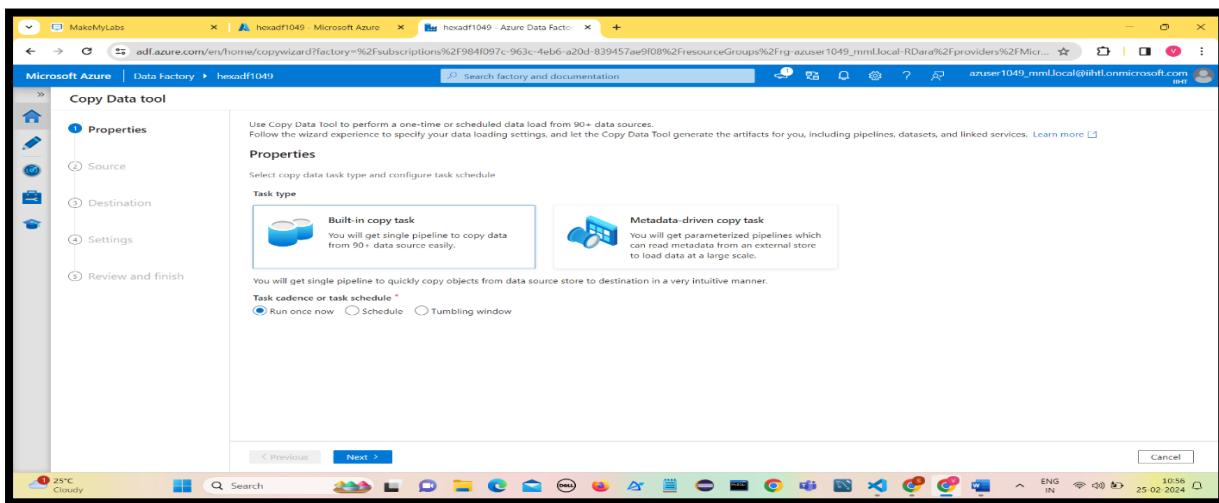
Click on Launch Studio



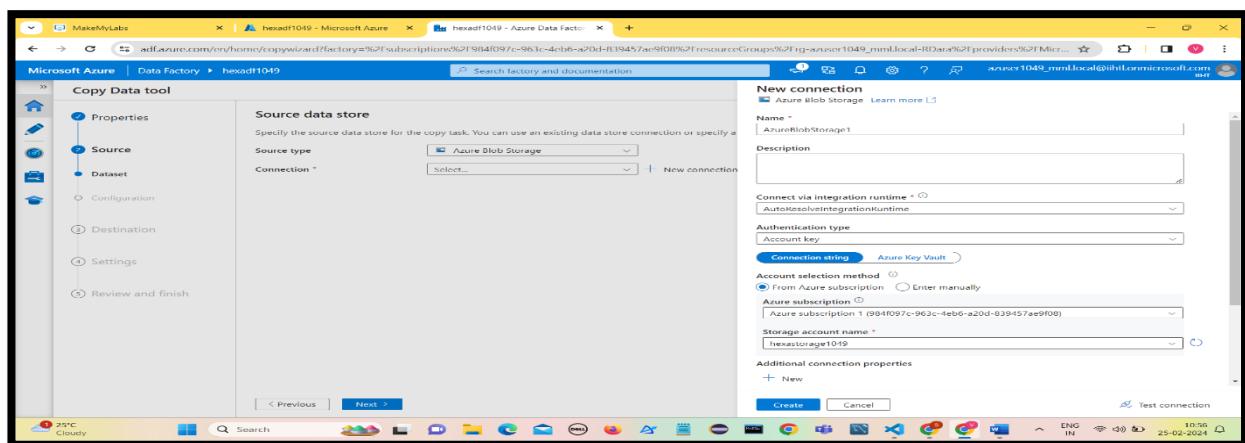
Click on Ingest to create a data factory pipeline to ingest data into ADLS gen2 storage.



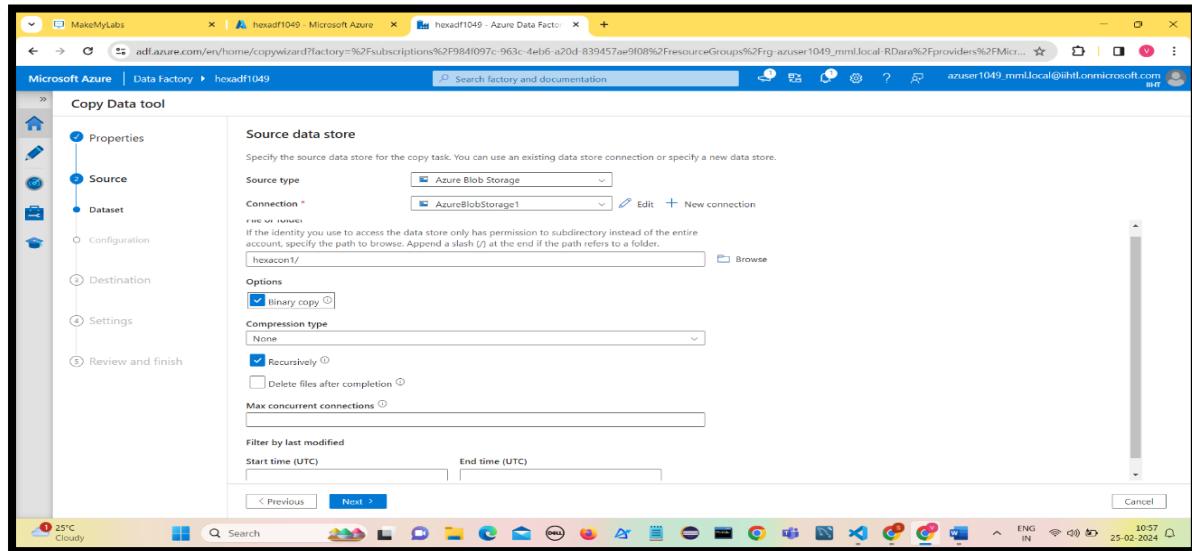
Click on built-in copy task and Enable Run Once now.



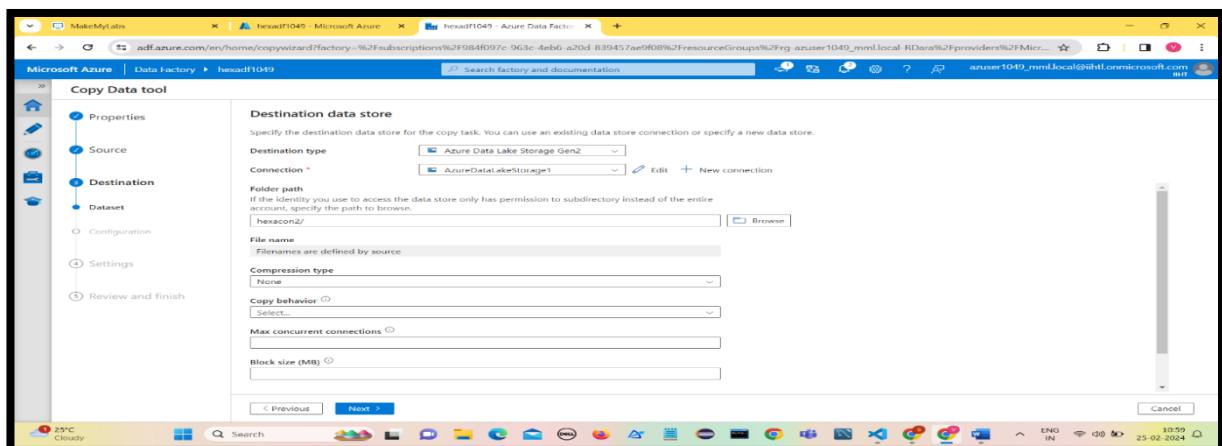
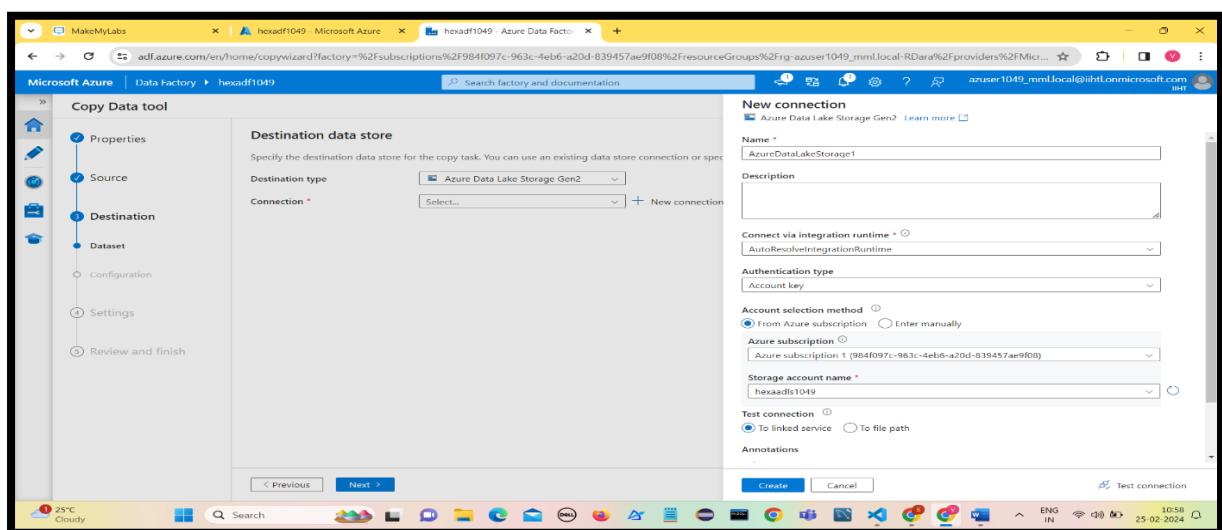
Create Source Connection by specifying the required details.



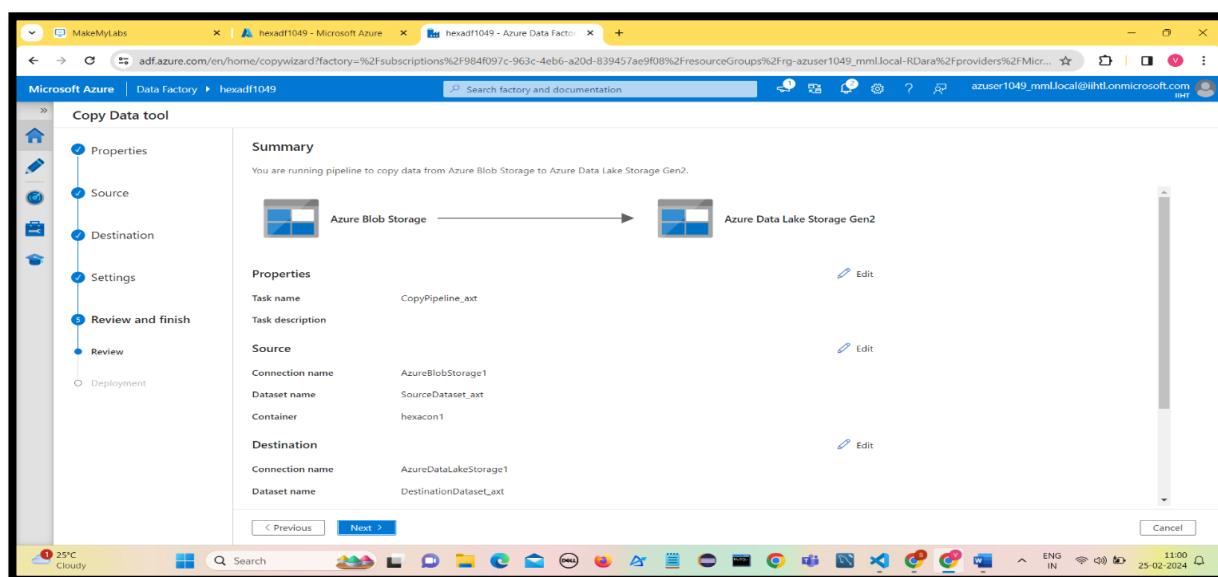
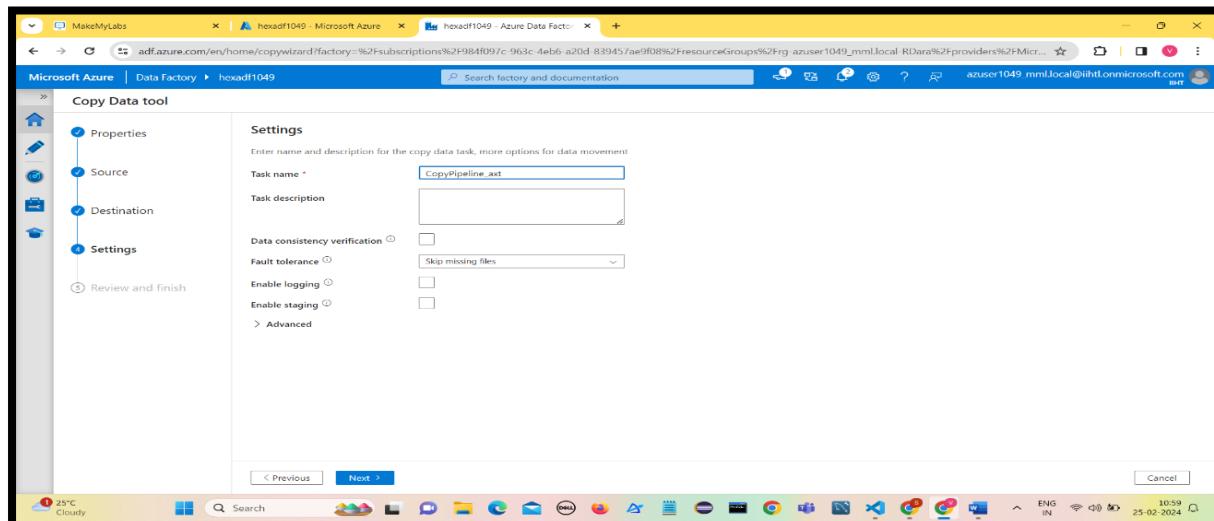
Give the folder details to copy data from storage account



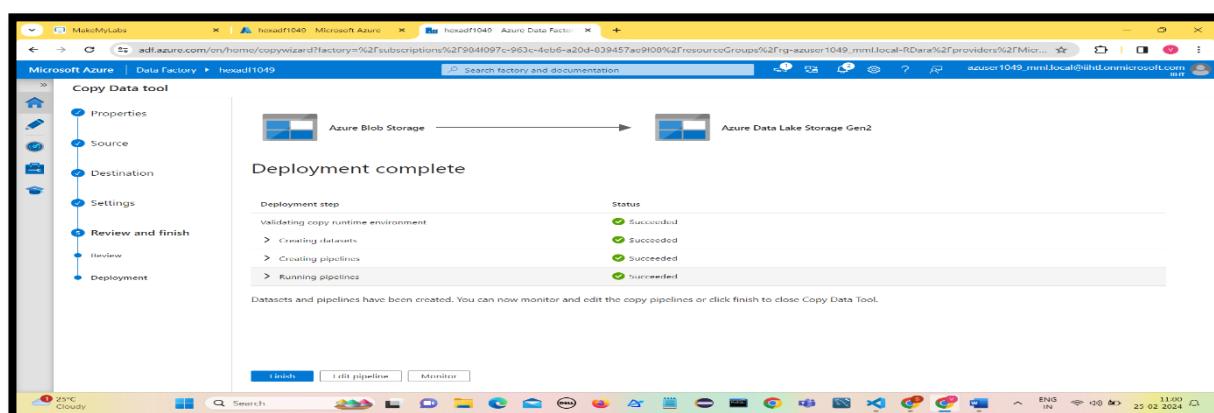
Create Destination Connection by specifying the required details.



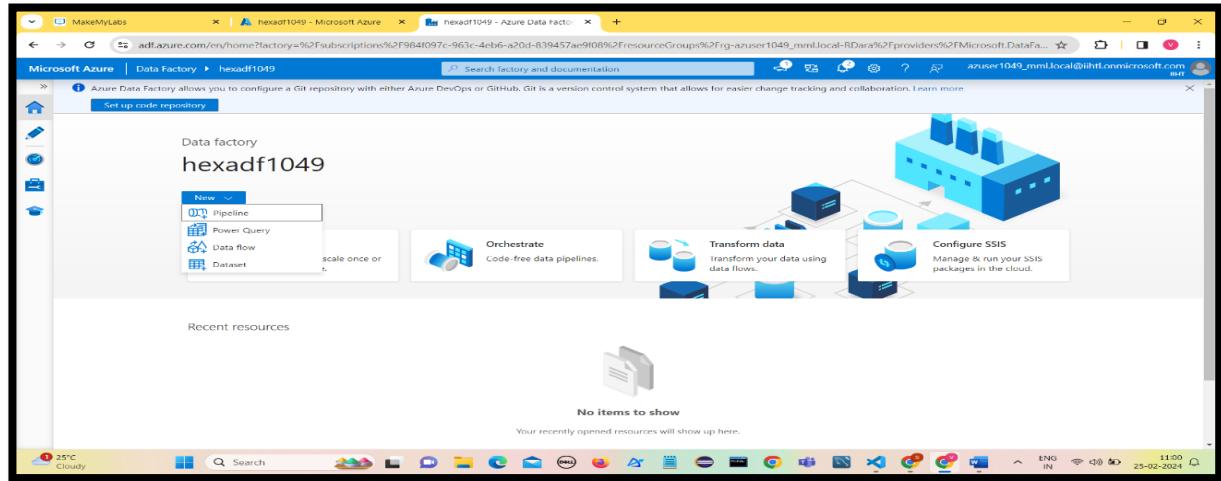
Give the name of pipeline and click on Next



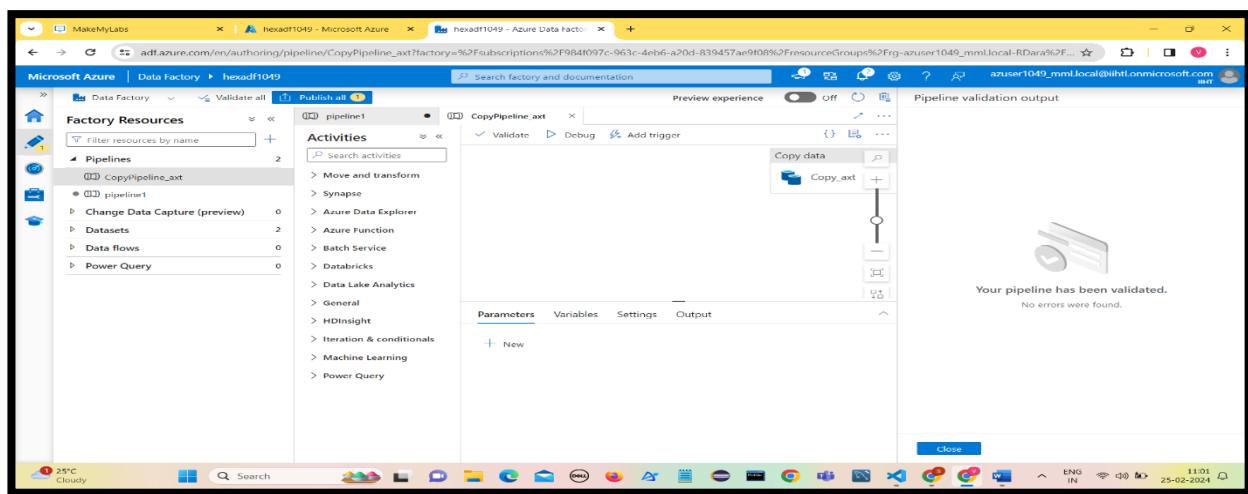
Click on Next



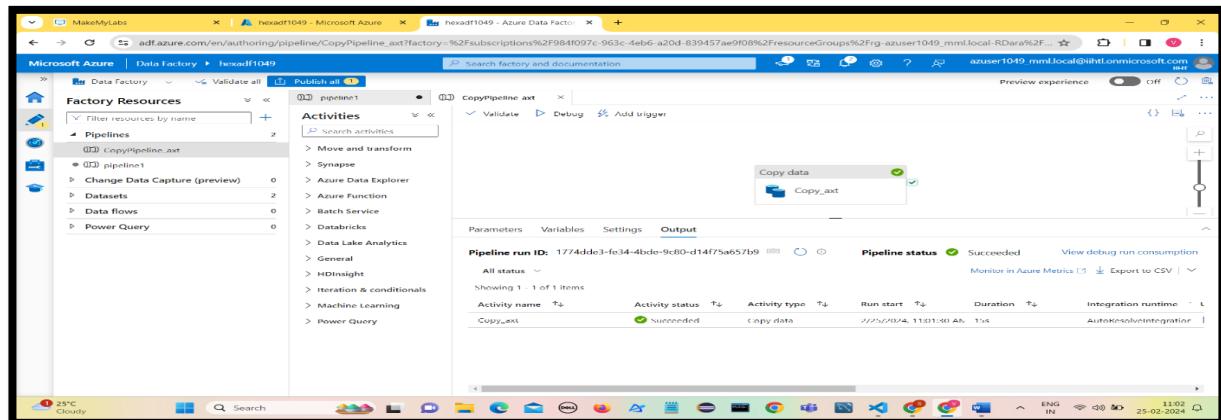
Click on Finish



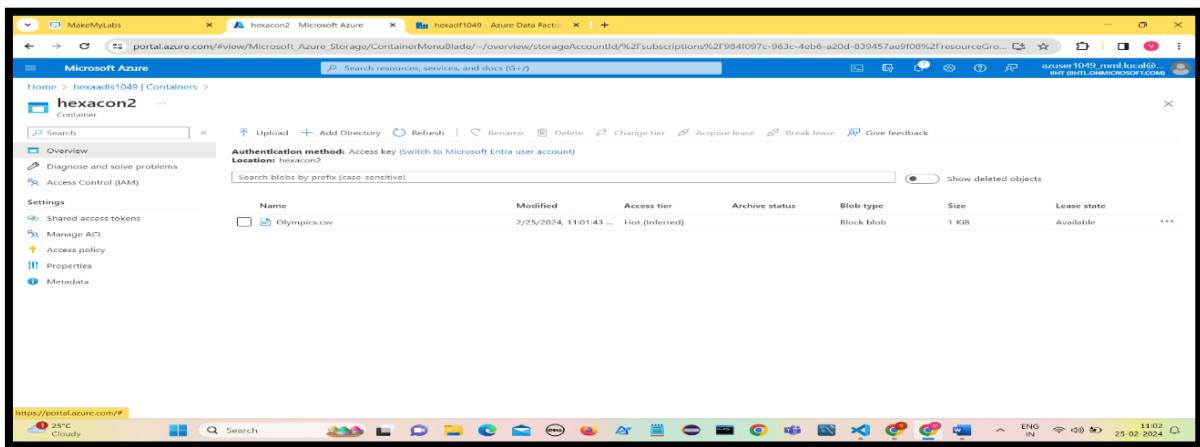
Click on New and then Click on Pipeline
Click on Validate



Click on Debug

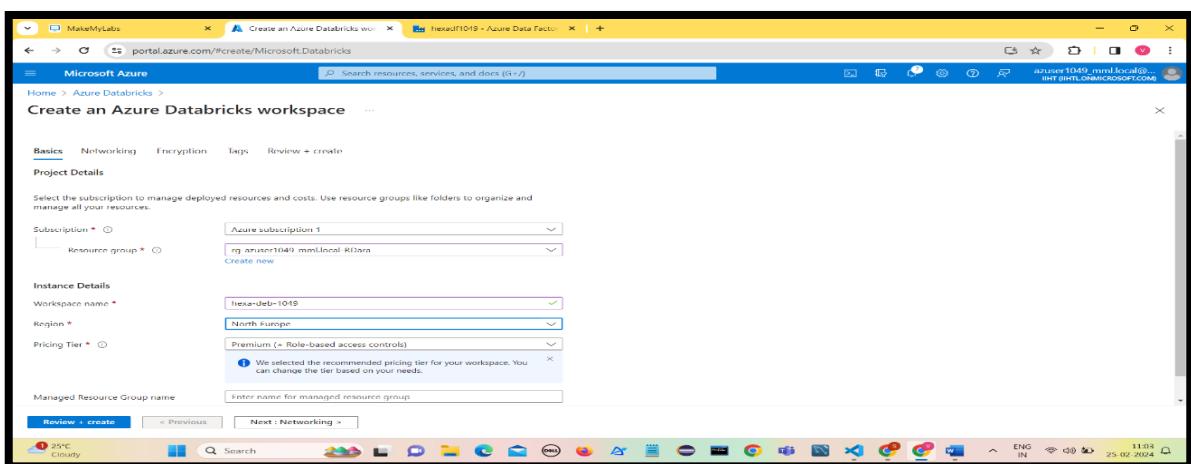


Since pipeline got executed , the file from source is copied into ADLS gen2 container

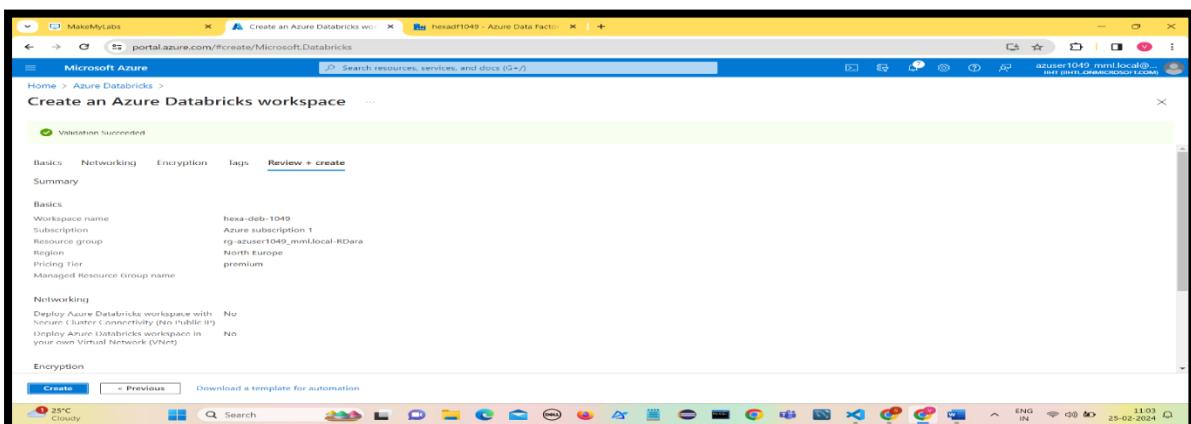


2) Data Transformation

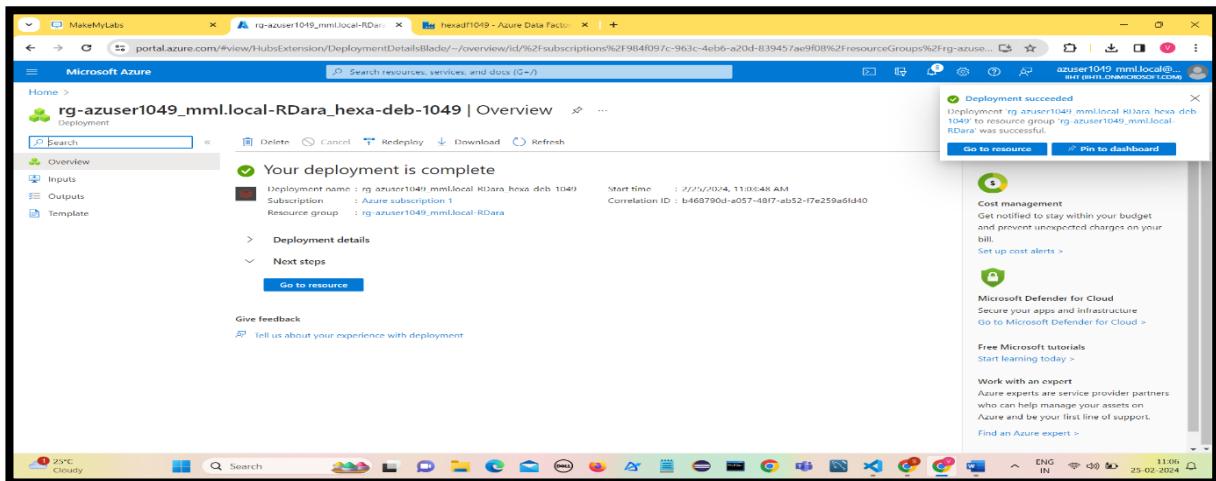
Creating Azure Data Bricks workspace



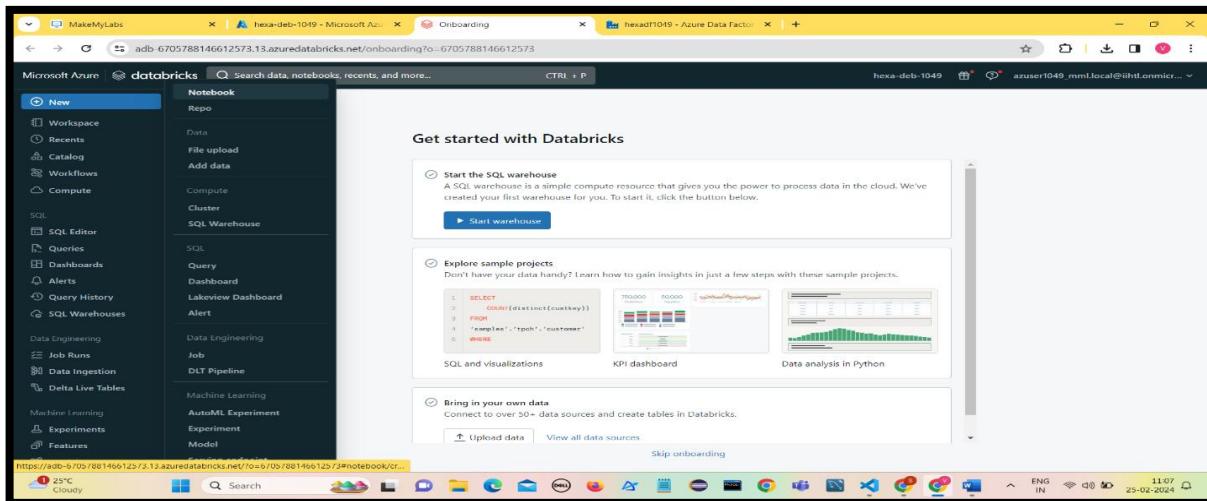
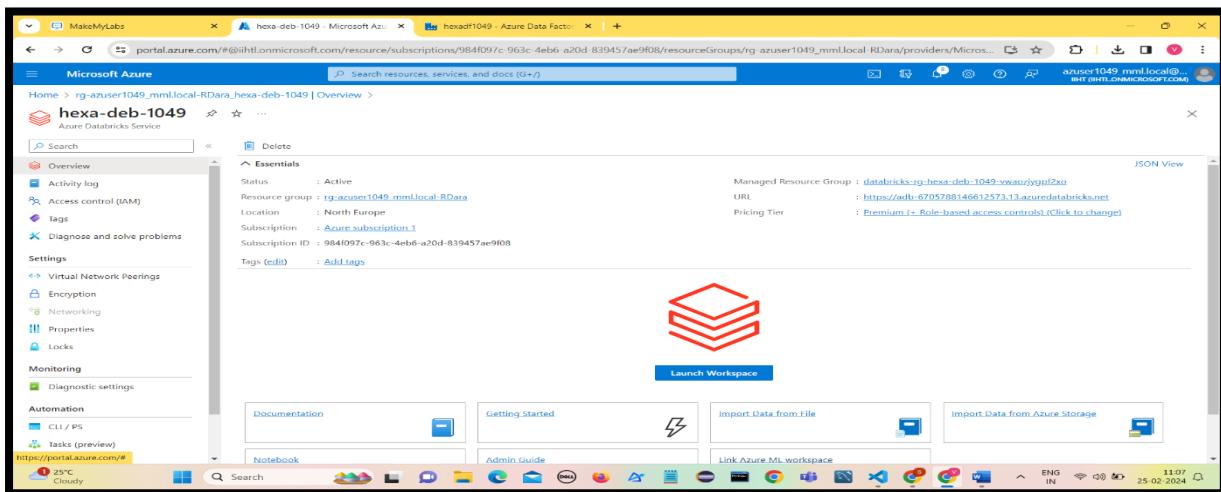
Click on Review+Create



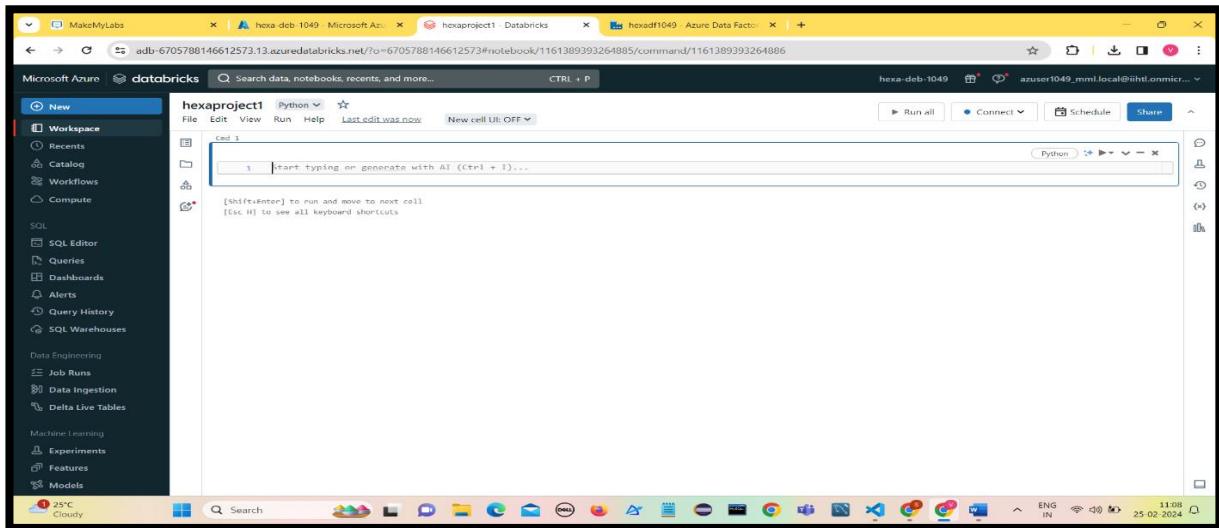
Once the deployment is complete , Click on Go to Resource.



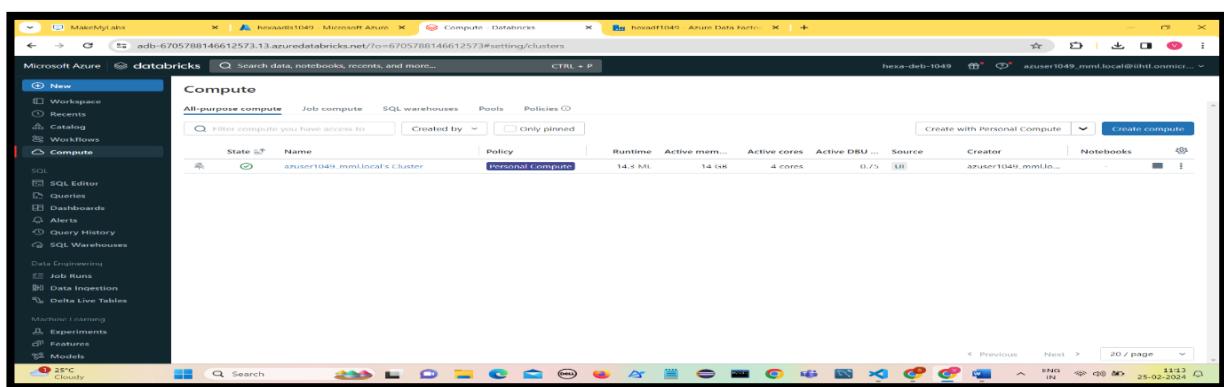
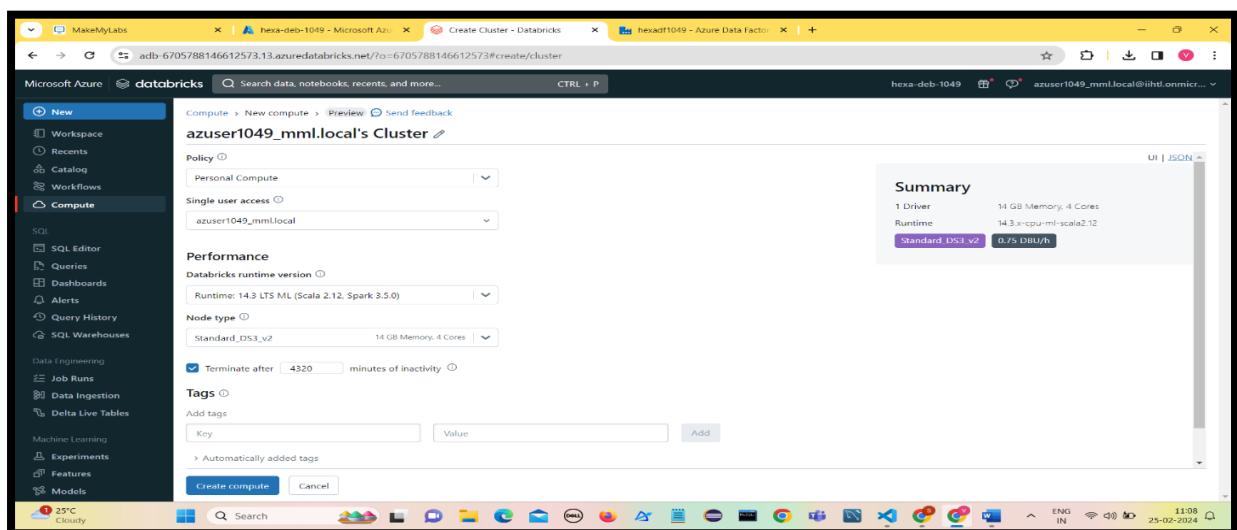
Click on Launch Workspace.



To create a Note Book in Data Bricks, Click on New and then Notebook.



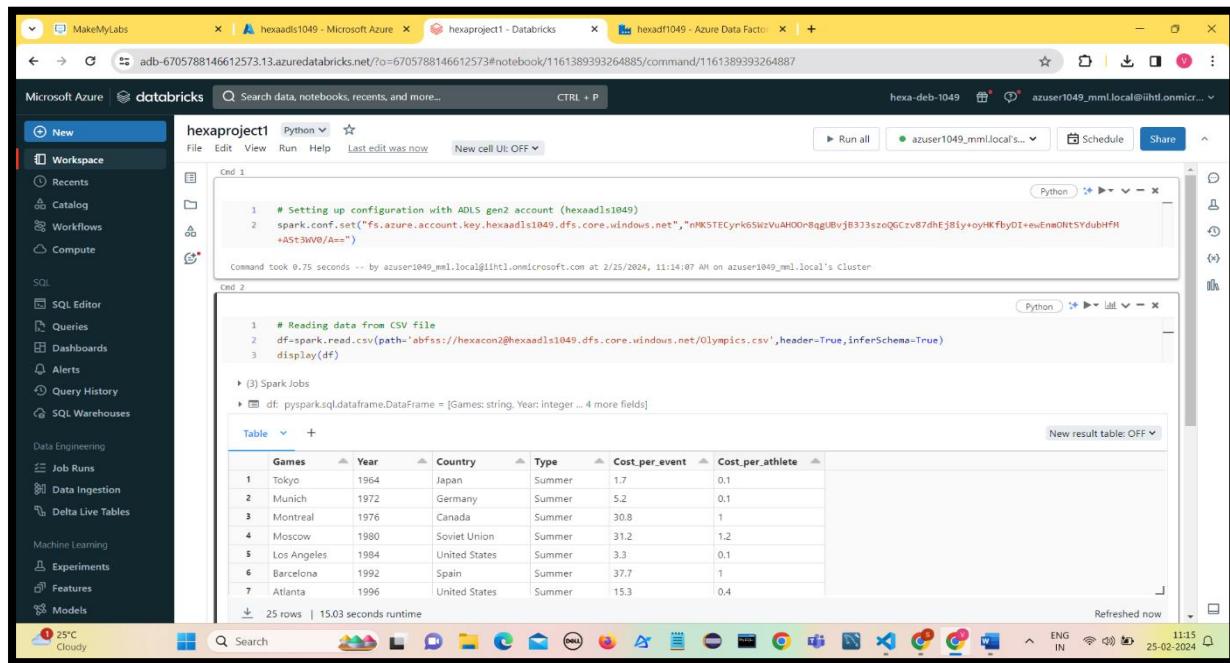
Create a compute (cluster) to run the Data Bricks Notebook .



Now the compute is ready to execute Data Bricks notebook.

Applying transformations on Ingested Data into Azure Data Lake Storage Gen2.

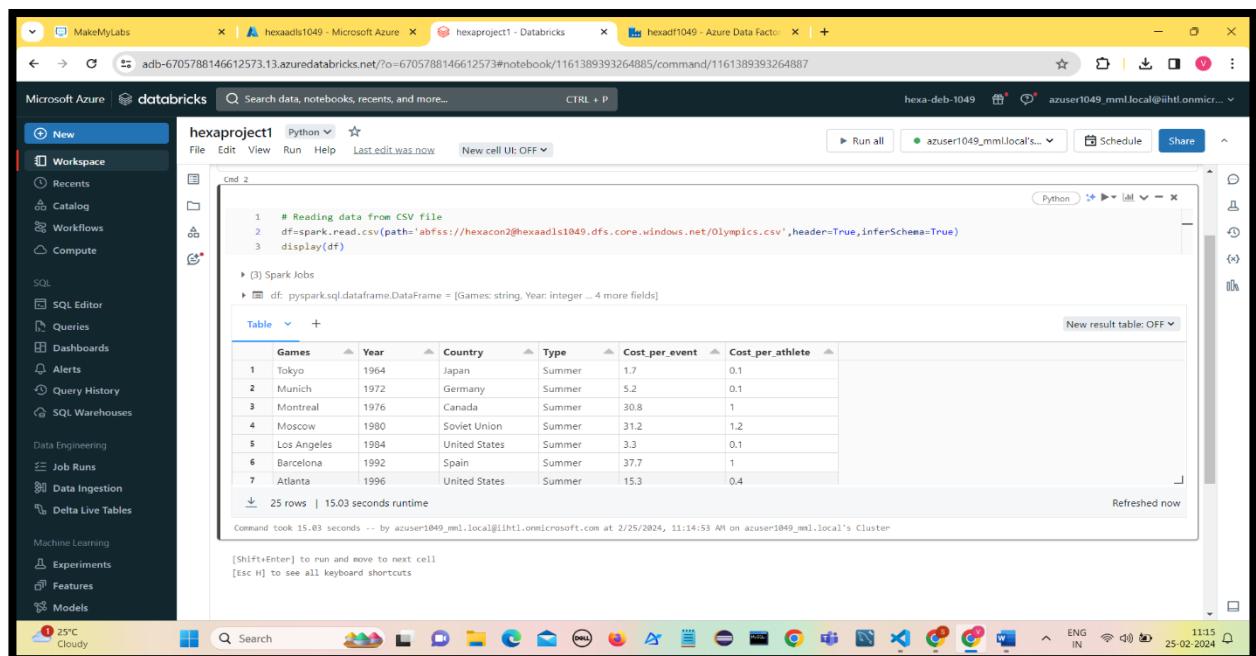
Setting up the configuration with ADLS Gen2 account with path and key as parameters.



The screenshot shows a Microsoft Azure Databricks notebook titled "hexaproject1". The left sidebar contains a navigation menu with sections like Workspace, Catalog, Workflows, Compute, SQL, Data Engineering, Machine Learning, and Data Ingestion. The main area has two code cells labeled Cmd 1 and Cmd 2. Cmd 1 contains Python code for setting up configuration with ADLS Gen2 account parameters. Cmd 2 contains Python code for reading data from a CSV file named "Olympics.csv" located at "abfss://hexacon2@hexaadls1049.dfs.core.windows.net/Olympics.csv". A preview of the data is shown in a table:

Games	Year	Country	Type	Cost_per_event	Cost_per_athlete
Tokyo	1964	Japan	Summer	1.7	0.1
Munich	1972	Germany	Summer	5.2	0.1
Montreal	1976	Canada	Summer	30.8	1
Moscow	1980	Soviet Union	Summer	31.2	1.2
Los Angeles	1984	United States	Summer	3.3	0.1
Barcelona	1992	Spain	Summer	37.7	1
Atlanta	1996	United States	Summer	15.3	0.4

Reading data from Olympics.csv file using `spark.read.csv` where the data of Olympics is present.



This screenshot shows the same Databricks notebook environment. The sidebar and Cmd 1 code are identical to the previous screenshot. Cmd 2 now contains additional Python code for transforming the DataFrame:

```
df = df.withColumn("Year", df["Year"].cast("int"))
df = df.withColumn("Cost_per_event", df["Cost_per_event"].cast("float"))
df = df.withColumn("Cost_per_athlete", df["Cost_per_athlete"].cast("float"))
```

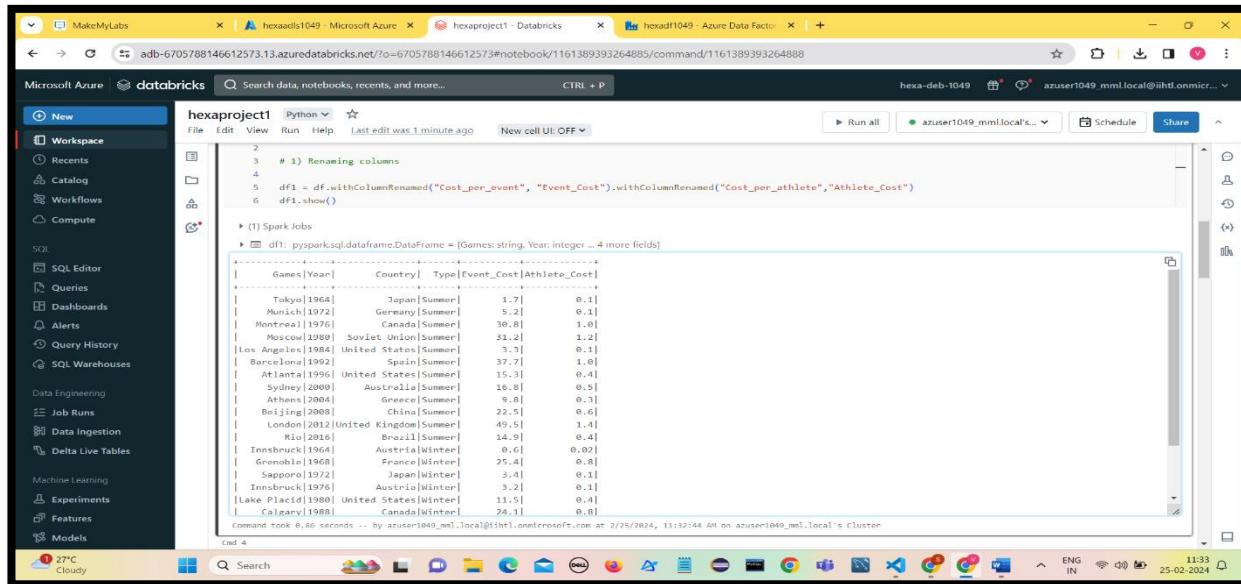
A preview of the transformed data is shown in a table:

Games	Year	Country	Type	Cost_per_event	Cost_per_athlete
Tokyo	1964	Japan	Summer	1.7	0.1
Munich	1972	Germany	Summer	5.2	0.1
Montreal	1976	Canada	Summer	30.8	1
Moscow	1980	Soviet Union	Summer	31.2	1.2
Los Angeles	1984	United States	Summer	3.3	0.1
Barcelona	1992	Spain	Summer	37.7	1
Atlanta	1996	United States	Summer	15.3	0.4

Applying transformations on Ingested Data into Azure Data Lake Storage Gen2.

1) Renaming Columns

Rename the column Cost_per_event with Event_Cost and Cost_per_athlete with Athlete_Cost.



The screenshot shows a Microsoft Azure Databricks workspace. A notebook titled 'hexaproject1' is open in Python mode. The code cell contains:

```
2 # 1) Renaming columns
3
4 df1 = df.withColumnRenamed("Cost_per_event", "Event_Cost").withColumnRenamed("Cost_per_athlete", "Athlete_Cost")
5 df1.show()
```

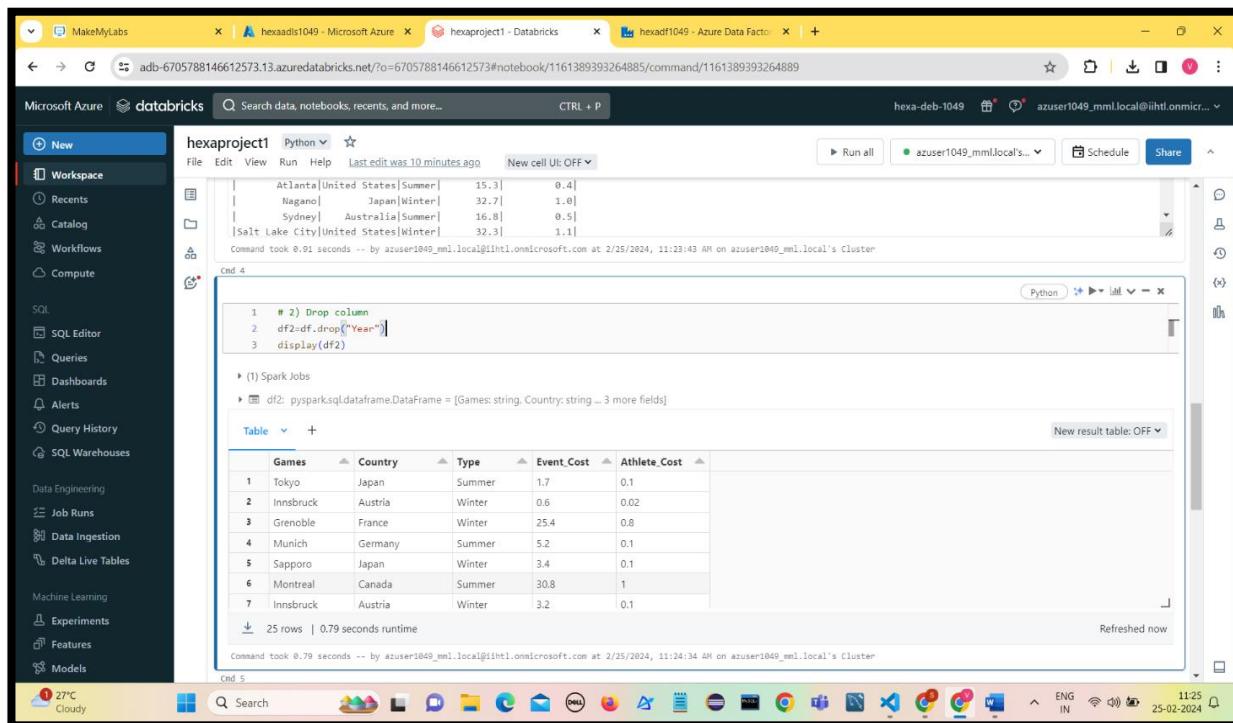
Below the code, a table is displayed with the following data:

Games	Year	Country	Type	Event_Cost	Athlete_Cost
Tokyo 1964		Japan Summer		1.7	0.1
Munich 1972		Germany Summer		5.2	0.1
Montreal 1976		Canada Summer		30.8	1.0
Moscow 1980		Soviet Union Summer		51.2	1.2
Los Angeles 1984		United States Summer		5.3	0.1
Barcelona 1992		Spain Summer		37.7	1.0
Antwerp 1996		Belgium Summer		15.3	0.4
Sydney 2000		Australia Summer		16.8	0.5
Athens 2004		Greece Summer		9.6	0.1
Beijing 2008		China Summer		22.5	0.6
London 2012	United Kingdom	United Kingdom Summer		49.5	1.4
Rio 2016		Brazil Summer		14.9	0.4
Innsbruck 1964		Austria Winter		0.6	0.02
Grenoble 1968		France Winter		25.4	0.8
Sapporo 1972		Japan Winter		3.4	0.1
Innsbruck 1976		Austria Winter		3.2	0.1
Lake Placid 1980	United States	United States Winter		11.5	0.4
Calgary 1988		Canada Winter		24.1	0.8

Command took 0.88 seconds -- by azuser1049_mml.local@iht1.onmicrosoft.com at 2/25/2024, 11:32:44 AM on azuser1049_mml.local's Cluster

2) Dropping a column

Drop Year column from the data and display the result.



The screenshot shows a Microsoft Azure Databricks workspace. A notebook titled 'hexaproject1' is open in Python mode. The code cell contains:

```
1 # 2) Drop column
2 df2=df.drop("Year")
3 display(df2)
```

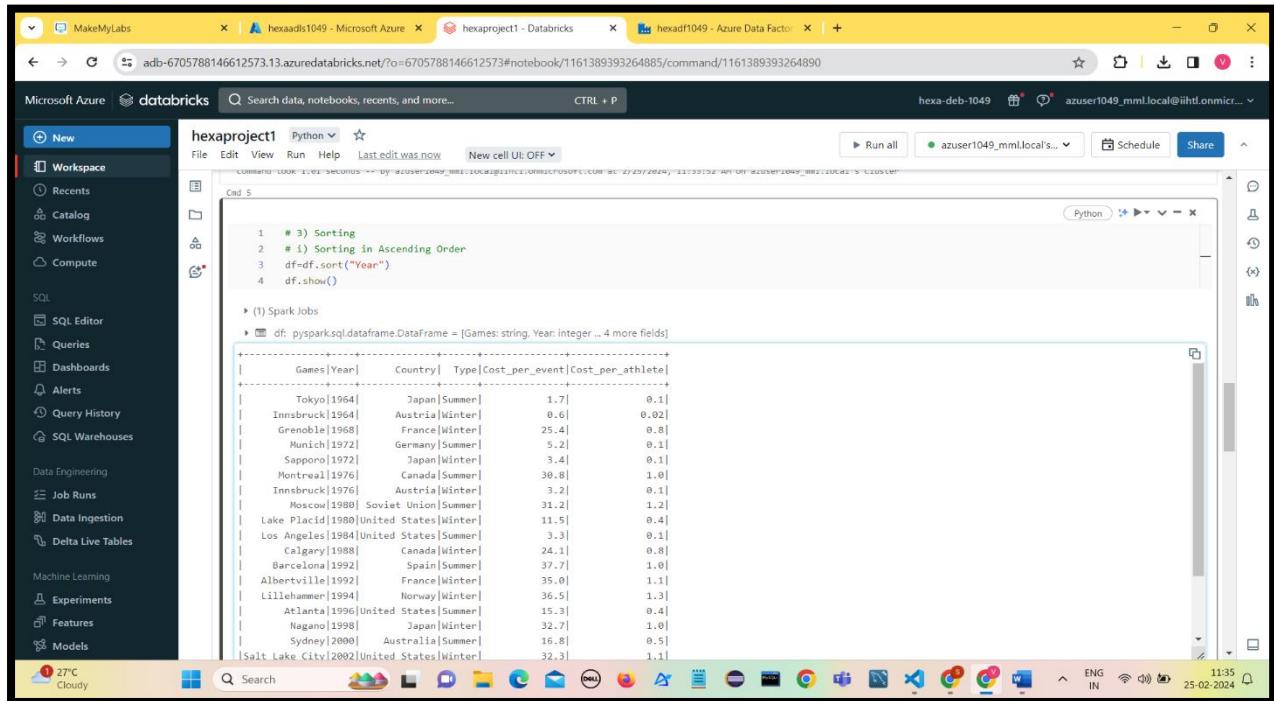
Below the code, a table is displayed with the following data:

Games	Country	Type	Event_Cost	Athlete_Cost
1 Tokyo	Japan	Summer	1.7	0.1
2 Innsbruck	Austria	Winter	0.6	0.02
3 Grenoble	France	Winter	25.4	0.8
4 Munich	Germany	Summer	5.2	0.1
5 Sapporo	Japan	Winter	3.4	0.1
6 Montreal	Canada	Summer	30.8	1
7 Innsbruck	Austria	Winter	3.2	0.1

Command took 0.79 seconds -- by azuser1049_mml.local@iht1.onmicrosoft.com at 2/25/2024, 11:24:34 AM on azuser1049_mml.local's Cluster

3) Sorting

Sort the Year column in Ascending Order.



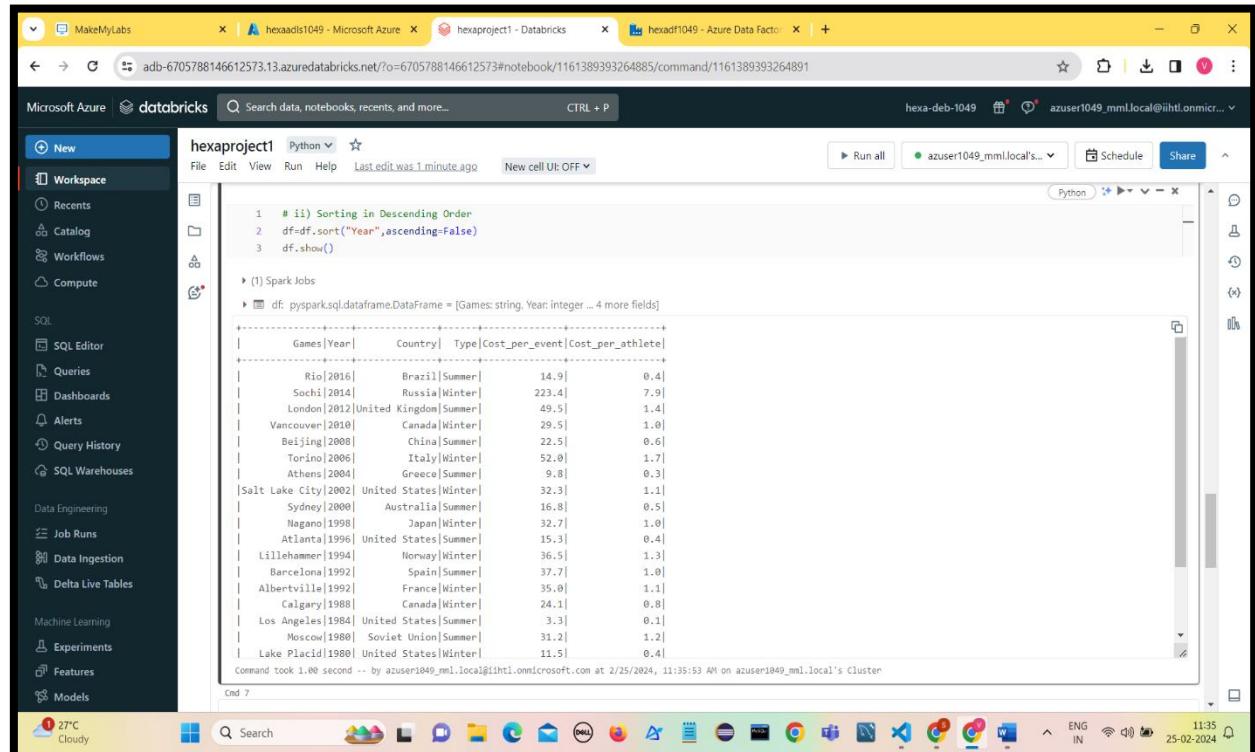
The screenshot shows a Microsoft Azure Databricks workspace titled "hexaproject1". The left sidebar contains various navigation options like Workspace, Catalog, Workflows, Compute, SQL, and Machine Learning. The main notebook cell contains the following Python code:

```
1 # 3) Sorting
2 # i) Sorting in Ascending Order
3 df=df.sort("Year")
4 df.show()
```

The output of the cell shows a table of Olympic games data sorted by year in ascending order:

Games	Year	Country	Type	Cost_per_event	Cost_per_athlete
Tokyo	1964	Japan	Summer	1.7	0.1
Innsbruck	1964	Austria	Winter	0.6	0.02
Grenoble	1968	France	Winter	25.4	0.8
Munich	1972	Germany	Summer	5.2	0.1
Sapporo	1972	Japan	Winter	3.4	0.1
Montreal	1976	Canada	Summer	30.8	1.0
Innsbruck	1976	Austria	Winter	3.2	0.1
Moscow	1980	Soviet Union	Summer	31.2	1.2
Lake Placid	1980	United States	Winter	11.5	0.4
Los Angeles	1984	United States	Summer	3.3	0.1
Calgary	1988	Canada	Winter	24.1	0.8
Barcelona	1992	Spain	Summer	37.7	1.0
Albertville	1992	France	Winter	35.0	1.1
Lillehammer	1994	Norway	Winter	36.5	1.3
Atlanta	1996	United States	Summer	15.3	0.4
Nagano	1998	Japan	Winter	32.7	1.0
Sydney	2000	Australia	Summer	16.8	0.5
Salt Lake City	2002	United States	Winter	32.3	1.1

Sort the year in Descending Order using `ascending=False`



The screenshot shows a Microsoft Azure Databricks workspace titled "hexaproject1". The left sidebar contains various navigation options like Workspace, Catalog, Workflows, Compute, SQL, and Machine Learning. The main notebook cell contains the following Python code:

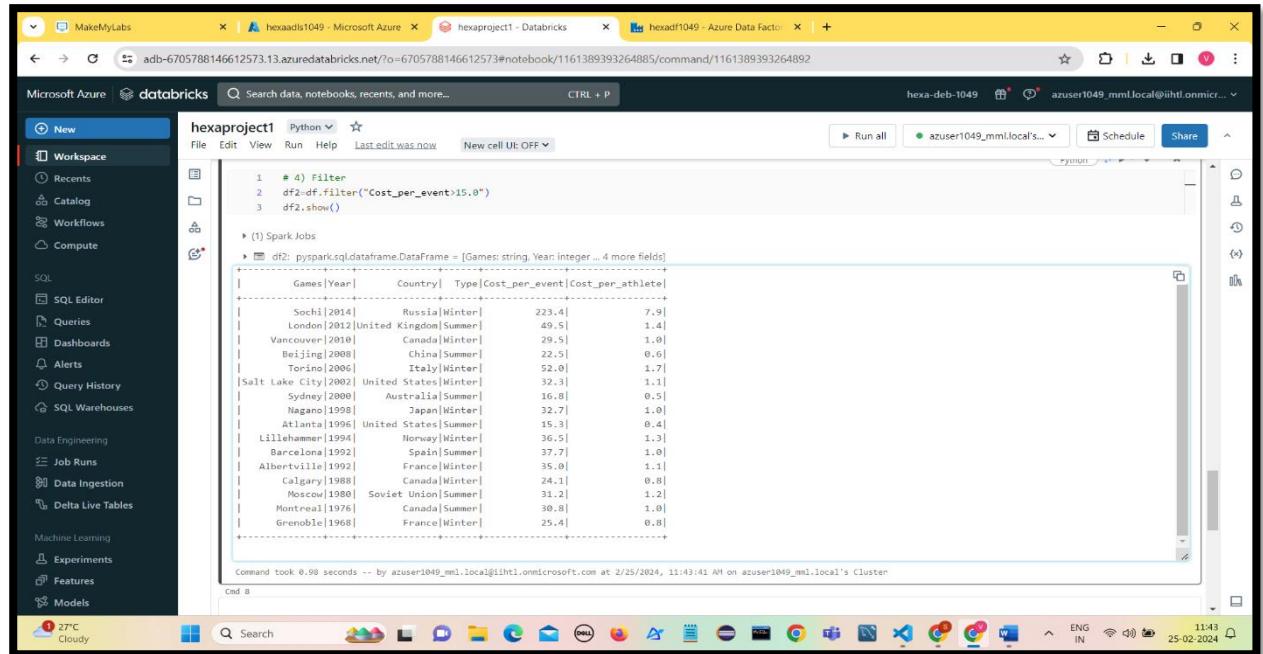
```
1 # ii) Sorting in Descending Order
2 df=df.sort("Year",ascending=False)
3 df.show()
```

The output of the cell shows a table of Olympic games data sorted by year in descending order:

Games	Year	Country	Type	Cost_per_event	Cost_per_athlete
Rio	2016	Brazil	Summer	14.9	0.4
Sochi	2014	Russia	Winter	223.4	7.9
London	2012	United Kingdom	Summer	49.5	1.4
Vancouver	2010	Canada	Winter	29.5	1.0
Beijing	2008	China	Summer	22.5	0.6
Torino	2006	Italy	Winter	52.0	1.7
Athens	2004	Greece	Summer	9.8	0.3
Salt Lake City	2002	United States	Winter	32.3	1.1
Sydney	2000	Australia	Summer	16.8	0.5
Nagano	1998	Japan	Winter	32.7	1.0
Atlanta	1996	United States	Summer	15.3	0.4
Lillehammer	1994	Norway	Winter	36.5	1.3
Barcelona	1992	Spain	Summer	37.7	1.0
Albertville	1992	France	Winter	35.0	1.1
Calgary	1988	Canada	Winter	24.1	0.8
Los Angeles	1984	United States	Summer	3.3	0.1
Moscow	1980	Soviet Union	Summer	31.2	1.2
Lake Placid	1980	United States	Winter	11.5	0.4

4) Filtering

Filter the Olympics Data where `Cost_per_event > 15.0`



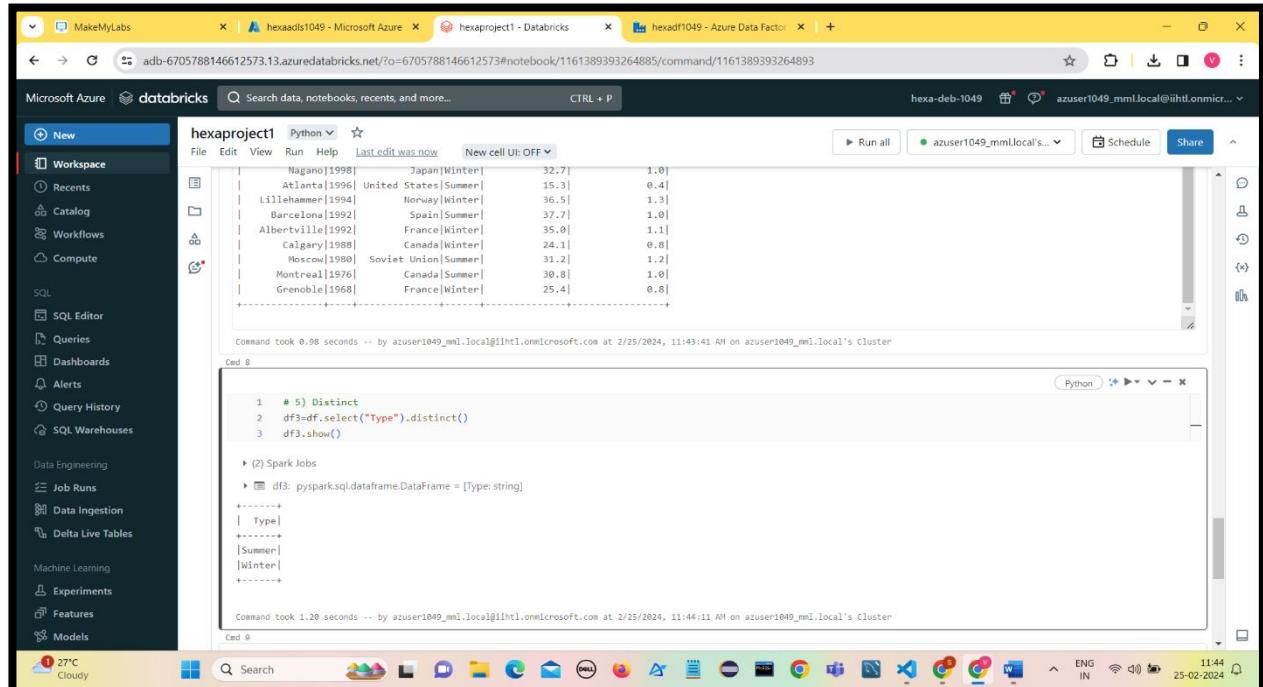
```
1 # 4) Filter
2 df2=df.filter("Cost_per_event>15.0")
3 df2.show()
```

Games	Year	Country	Type	Cost_per_event	Cost_per_athlete
Sochi	2014	Russia	Winter	223.4	7.9
London	2012	United Kingdom	Summer	49.5	1.4
Vancouver	2010	Canada	Winter	29.5	1.0
Beijing	2008	China	Summer	22.5	0.6
Torino	2006	Italy	Winter	52.0	1.7
Salt Lake City	2002	United States	Winter	32.3	1.1
Sydney	2000	Australia	Summer	16.8	0.5
Nagano	1998	Japan	Winter	32.7	1.0
Atlanta	1996	United States	Summer	15.3	0.4
Lillehammer	1994	Norway	Winter	36.5	1.3
Barcelona	1992	Spain	Summer	37.7	1.0
Albertville	1992	France	Winter	35.0	1.1
Calgary	1988	Canada	Winter	24.1	0.8
Moscow	1980	Soviet Union	Summer	31.2	1.2
Montreal	1976	Canada	Summer	30.8	1.0
Grenoble	1968	France	Winter	25.4	0.8

Command took 0.98 seconds -- by azuser1049_mml.local@iht1.onmicrosoft.com at 2/25/2024, 11:43:41 AM on azuser1049_mml.local's Cluster

5) Distinct

Selecting the distinct values of a Type column



```
1 # 5) Distinct
2 df3=df.select("Type").distinct()
3 df3.show()
```

Type
[Summer]
[Winter]

Command took 1.20 seconds -- by azuser1049_mml.local@iht1.onmicrosoft.com at 2/25/2024, 11:44:11 AM on azuser1049_mml.local's Cluster

6) Group By and Aggregations

1) sum()

Find the total amount of Cost_per_Event for each Type.

```
hexaproject1 - Python 3.8
File Edit View Run Help Last edit was 1 minute ago New cell UI: OFF
Run all azuser1049_mml.local@iht1.onmicrosoft.com
hexadbf1049 - Azure Data Factory
hexadbf1049 - Microsoft Azure
hexadbf1049 - Databricks
Search data, notebooks, recents, and more...
New cell UI: OFF
Run all Schedule Share
New Workspace Recents Catalog Workflows Compute SQL Editor Queries Dashboards Alerts Query History SQL Warehouses Job Runs Data Ingestion Delta Live Tables Machine Learning Experiments Features Models
27°C Cloudy
25-02-2024 11:44
238.7
509.5999999999997
```

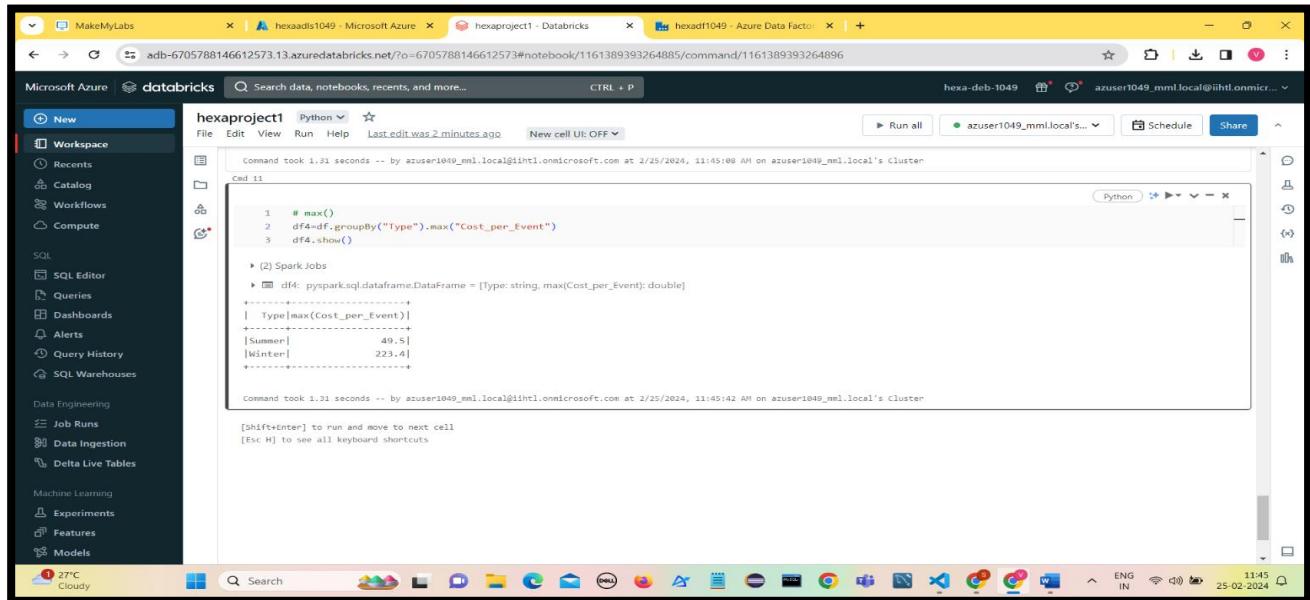
2) min()

Find the minimum of Cost_per_Event for each Type.

```
hexaproject1 - Python 3.8
File Edit View Run Help Last edit was 1 minute ago New cell UI: OFF
Run all azuser1049_mml.local@iht1.onmicrosoft.com
hexadbf1049 - Azure Data Factory
hexadbf1049 - Microsoft Azure
hexadbf1049 - Databricks
Search data, notebooks, recents, and more...
New cell UI: OFF
Run all Schedule Share
New Workspace Recents Catalog Workflows Compute SQL Editor Queries Dashboards Alerts Query History SQL Warehouses Job Runs Data Ingestion Delta Live Tables Machine Learning Experiments Features Models
27°C Cloudy
1.7
0.6
```

3) max()

Find the Maximum of Cost_per_Event for each Type.

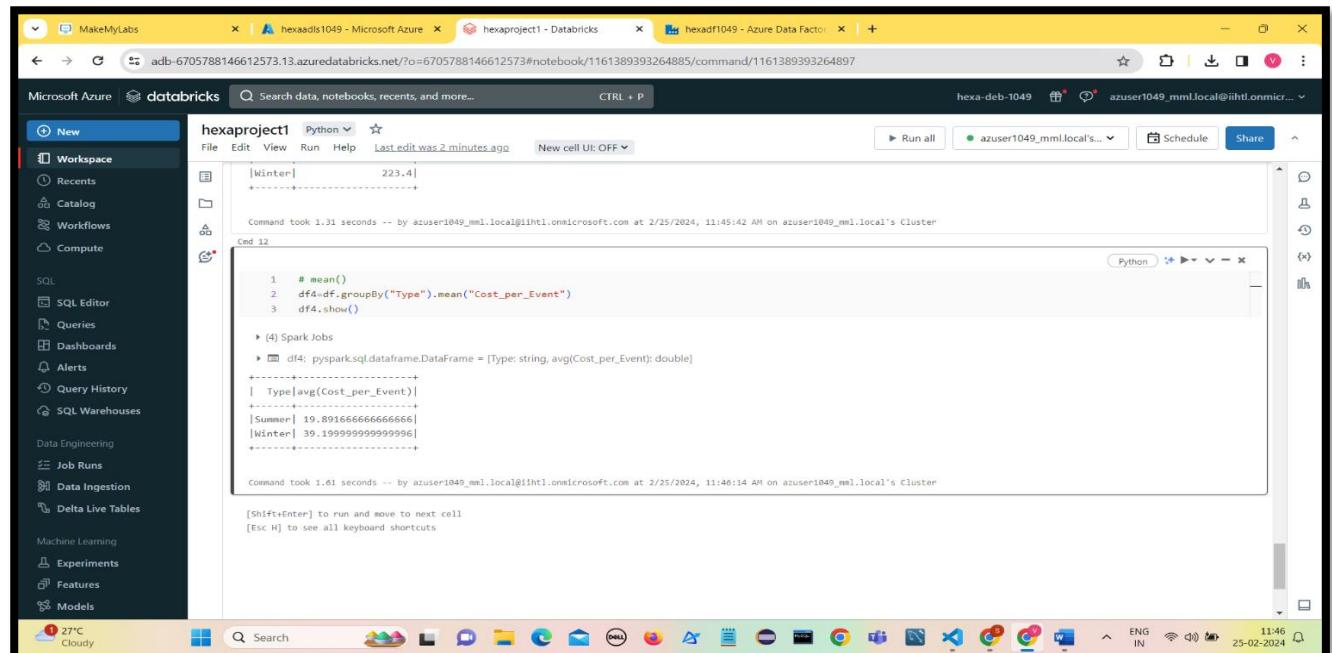


```
1 # max()
2 df4=df.groupby("Type").max("Cost_per_Event")
3 df4.show()

+-----+
| Type|max(Cost_per_Event)|
+-----+
| [Summer]        49.5|
| [Winter]        223.4|
+-----+
```

4) mean()

Find the Mean of Cost_per_Event for each Type.

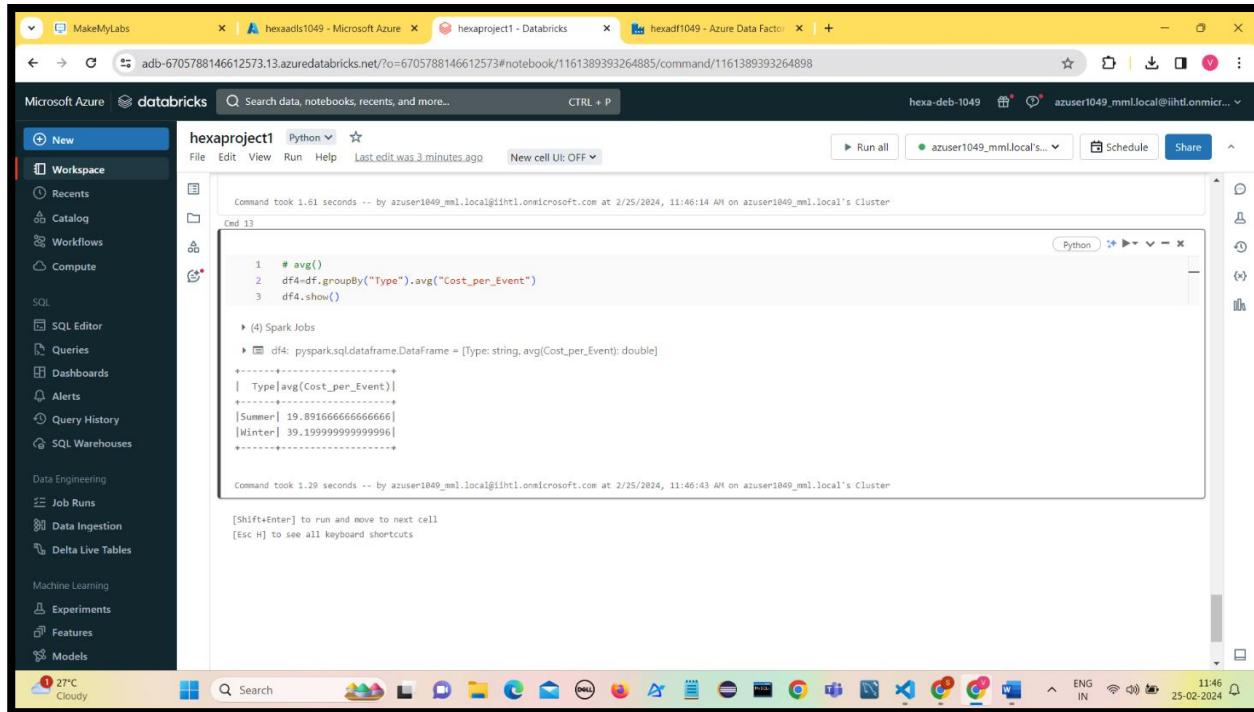


```
1 # mean()
2 df4=df.groupby("Type").mean("Cost_per_Event")
3 df4.show()

+-----+
| Type|avg(Cost_per_Event)|
+-----+
| [Summer]    19.891666666666666|
| [Winter]    39.199999999999996|
+-----+
```

5) avg()

Find the Average of Cost_per_Event for each Type.



The screenshot shows a Microsoft Edge browser window with four tabs open: 'MakeMyLabs', 'hexacls1049 - Microsoft Azure', 'hexaproject1 - Databricks', and 'hexadf1049 - Azure Data Factory'. The 'hexaproject1 - Databricks' tab is active, displaying a Python notebook titled 'hexaproject1'. The notebook contains the following code:

```
1 # avg()
2 df4=df.groupby("Type").avg("Cost_per_Event")
3 df4.show()
```

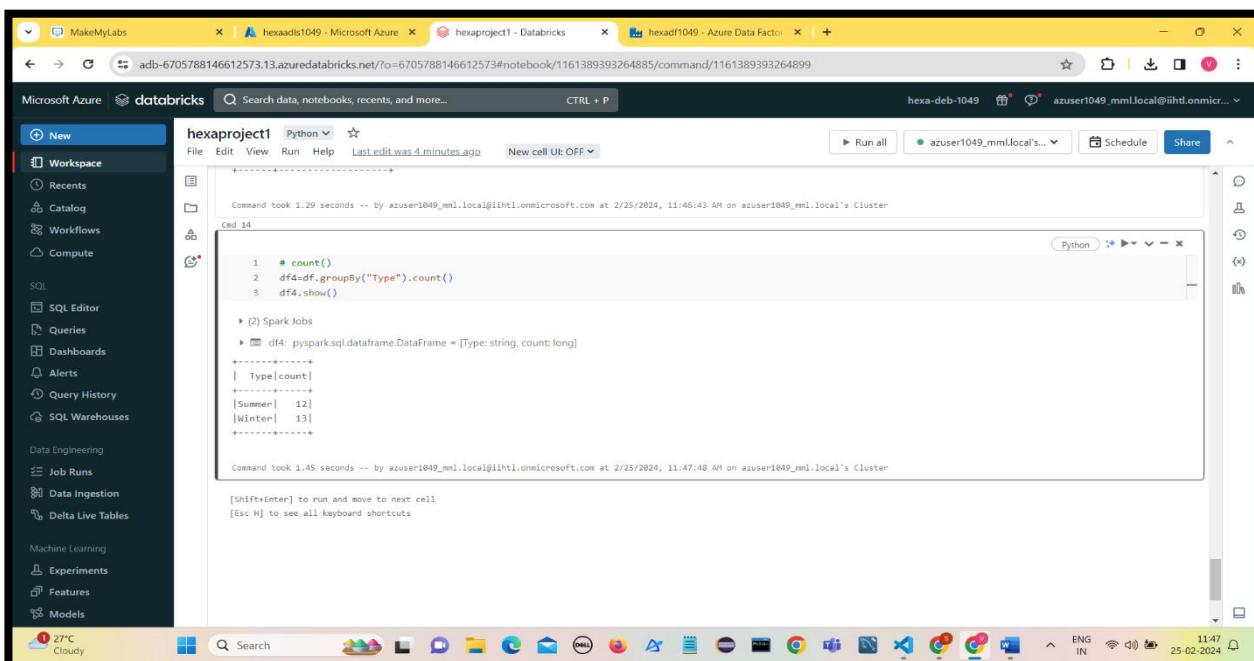
The output of the code is:

```
(4) Spark Jobs
+-----+
| df4: pyspark.sql.dataframe.DataFrame = [Type: string, avg(Cost_per_Event): double]
+-----+
| Type|avg(Cost_per_Event)|
+-----+
|[Summer]| 19.891666666666666 |
|[Winter]| 39.19999999999996 |
+-----+
```

Below the code cell, there is a note: '[Shift+Enter] to run and move to next cell [Esc H] to see all keyboard shortcuts'.

6) count()

Find the count (Number of rows) for each Type.



The screenshot shows a Microsoft Edge browser window with four tabs open: 'MakeMyLabs', 'hexacls1049 - Microsoft Azure', 'hexaproject1 - Databricks', and 'hexadf1049 - Azure Data Factory'. The 'hexaproject1 - Databricks' tab is active, displaying a Python notebook titled 'hexaproject1'. The notebook contains the following code:

```
1 # count()
2 df4=df.groupby("Type").count()
3 df4.show()
```

The output of the code is:

```
(2) Spark Jobs
+-----+
| df4: pyspark.sql.dataframe.DataFrame = [Type: string, count: long]
+-----+
| Type|count|
+-----+
|[Summer]| 12 |
|[Winter]| 13 |
+-----+
```

Below the code cell, there is a note: '[Shift+Enter] to run and move to next cell [Esc H] to see all keyboard shortcuts'.

7) agg0

Using more than one aggregates at a time

Find the sum of Cost_per_Event and minimum of Cost_per_Athlete

```
hexaproject1 Python ★
File Edit View Run Help Last edit was 9 minutes ago New cell UI: OFF ▾

(c) spark jobs
└ df4: pyspark.sql.dataframe.DataFrame = [Type: string, count: long]
    +---+
    | Type|count|
    +---+
    |[Summer]| 12|
    |[Winter]| 13|
    +---+


Command took 1.45 seconds -- by azuser1049_mml.local@iithl.onmicrosoft.com at 2/25/2024, 11:47:48 AM on azuser1049_mml.local's cluster
Cmd 15
Python ▶ ▶ ▶ ✖

1 # agg()
2 from pyspark.sql.functions import sum,min
3 df4=df.groupBy("Type").agg(sum(df["Cost_per_Event"]).alias("Total_Event_Cost"), min(df["Cost_per_Athlete"]).alias("Min_Athlete_Cost"))
4 df4.show()

▶ (4) Spark Jobs
▶ df4: pyspark.sql.dataframe.DataFrame = [Type: string, Total_Event_Cost: double ... 1 more field]
    +---+
    | Type| Total_Event_Cost|Min_Athlete_Cost|
    +---+
    |[Summer]| 238.7| 0.1|
    |[Winter]| 1509.5999999999997| 0.02|
    +---+


Command took 1.24 seconds -- by azuser1049_mml.local@iithl.onmicrosoft.com at 2/25/2024, 11:52:38 AM on azuser1049_mml.local's cluster
[Tshift+Enter] to run and move to next cell
```

Dataset used:

Conclusion:

This project successfully demonstrated the implementation of a robust data pipeline for ingesting Olympics data into Azure Data Lake Storage Gen2 using Azure Data Factory (ADF) pipelines and performing data transformation tasks using PySpark notebooks in Azure Databricks. The project enables stakeholders to gain valuable insights and analysis from the processed data, contributing to data-driven decision-making and business intelligence efforts.

Vuthur Sriganga

Data Engineering Batch-1

vuthursriganga@gmail.com