

## **Project-2 (Data Engineering Batch-1)**

### **Data Analysis using SparkSQL**



**Vuthur Sriganga**

**Hexaware Data Engineering Batch-1**

**vuthursriganga@gmail.com**

## **Project-2 (Data Engineering Batch-1)**

### **Data Analysis using SparkSQL**

#### **Title of Project:**

Implement Data analysis using Spark SQL on Azure Databricks & Process data for errors , Seasonality, and Anomalies.

#### **Project Overview:**

The goal of this project is to implement data analysis using Spark SQL on Azure Databricks and process the data for errors, seasonality, and anomalies. The project involves leveraging Azure Databricks for its Spark-based analytics capabilities and Spark SQL for querying and processing large datasets efficiently.

#### **About the project:**

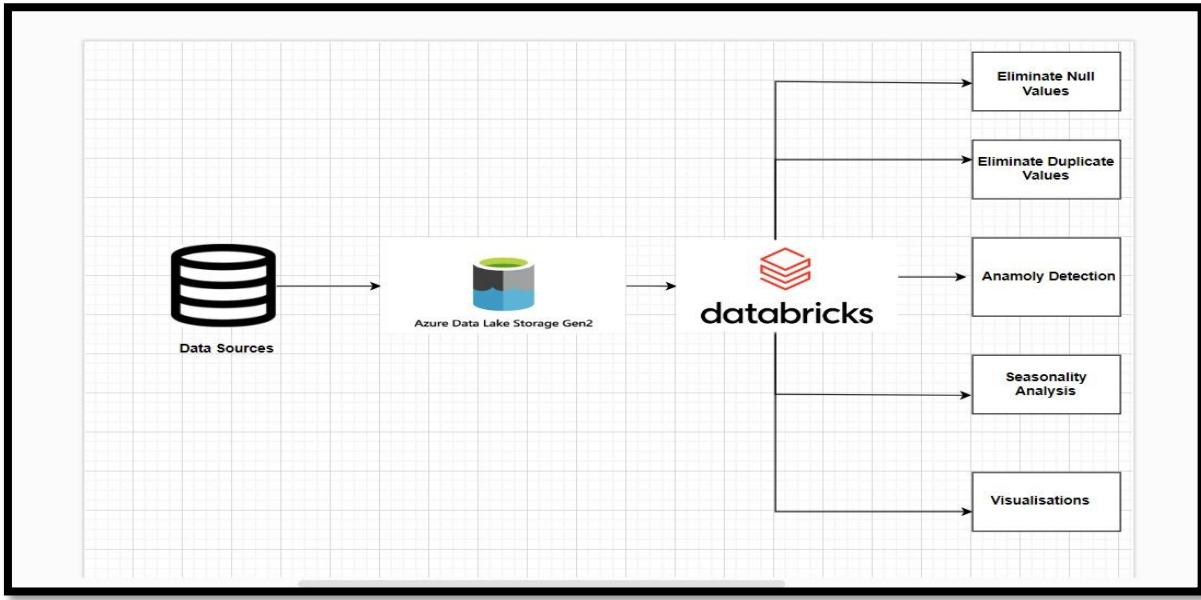
The project involves implementing data analysis using Spark SQL on Azure Databricks and processing the data for errors, seasonality, and anomalies. This initiative aims to leverage the scalability and performance of Spark on the Azure Databricks platform for efficient data processing and analysis.

#### **Key Components of this project:**

- 1) Data Ingestion
- 2) Data Exploration
- 3) Data Cleaning
- 4) Data Transformation
- 5) Data Analysis with Spark SQL
- 6) Error Detection
- 7) Seasonality Analysis
- 8) Anomaly Detection
- 9) Visualization and Reporting

#### **Architecture Diagram:**

The Architecture diagram shows the flow of execution of project and the operations that are performed on the dataset.



## How it works

### 1) Setup Azure Databricks Environment:

- Create an Azure Databricks workspace in the Azure portal.
- Choose the appropriate subscription, resource group, and workspace name.
- Configure the pricing tier based on your requirements. Options include Standard and Premium tiers with different capabilities and performance levels.
- Once the deployment is complete, click on Go to Resource.
- Create a new notebook in the workspace.
- Create a compute by providing the required details and configure it with data bricks notebook.

### 2) Data Ingestion:

- Create an Azure Data Lake Storage Gen2 account in Azure Portal.
- Enable the Hierarchical Namespace in the Advanced Settings of ADLS account.
- Create a container in the ADLS Gen2 account. Upload the source file into container.
- Configure the Azure Data Lake Storage Gen2 account with Data Bricks to perform all the required operations.

### 3) Data Exploration and Cleaning:

- Explore the ingested data using Databricks notebooks, which provide an interactive environment for data exploration and analysis.

- Use Spark DataFrame APIs or Spark SQL within the notebooks to examine the data's schema, sample rows.
- Identify data quality issues such as missing values, duplicate records, inconsistent formats, and outliers.
- Apply data cleaning techniques such as removing null values, deduplication to ensure data quality and consistency.

#### **4) Data Transformation:**

- Transform the cleaned data into a format suitable for analysis using Spark SQL.
- Perform transformations such as filtering, aggregating to prepare the data for analysis.
- Leverage Spark's distributed processing capabilities to handle large-scale data transformations efficiently.

#### **5) Data Analysis with Spark SQL:**

- Write Spark SQL queries within Databricks notebooks to perform advanced analytics on the transformed data.
- Utilize SQL syntax to extract insights, perform aggregations, and apply statistical functions.
- Leverage Spark SQL's built-in functions and windowing functions for complex analytics tasks such as time series analysis, Seasonality detection, and anomaly detection.

#### **6) Error Detection:**

- Implement algorithms or techniques within Spark SQL to detect errors or anomalies in the data.
- Utilize SQL queries and functions to identify inconsistencies or patterns deviating from expected behavior.

#### **7) Seasonality Analysis:**

- Analyze the data for seasonality patterns using Spark SQL.
- Apply time series analysis techniques such as moving averages, or Fourier transforms to identify recurring trends or fluctuations over time.
- Use SQL queries and functions to aggregate data over time intervals and calculate seasonal metrics such as seasonal averages.

#### **Steps:**

- Apply moving average to smooth out seasonal variations.
- Analyze Moving Averages
- Visualize Moving Averages in Azure Data Bricks.

## **8) Anomaly Detection:**

- Develop models or algorithms within Spark SQL to detect anomalies in the data.
- Utilize machine learning algorithms, statistical methods, or rule-based approaches for anomaly detection.

### **Steps:**

- Calculate Z-scores for each value
- Define anomaly threshold (e.g., Z-score greater than 1)
- Analyze the identified anomalies
- Visualization of Anomalies (e.g., scatter plot of ORDERDATE vs. QUANTITYORDERED)

## **9) Visualization and Reporting:**

- Visualize the analyzed data and insights using Databricks notebooks or external visualization tools.
- Use visualization in Databricks' built-in visualization tools to create compelling visualizations that highlight key insights and trends.

### **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 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.

#### **3) Spark SQL:**

- Spark SQL is a component of Apache Spark that introduces a structured data processing model along with a SQL-like query language.
- It allows users to run SQL queries programmatically and seamlessly integrate SQL queries with existing Spark programs.

#### **4) 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.

#### **5) 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:**

Ingest raw data from various sources such as databases, files, or streaming sources into Azure Databricks storage.

#### **2) Data Exploration:**

Explore the ingested data to understand its structure, schema, and quality.

#### **3) Data Cleaning:**

- Clean the data to handle inconsistencies, missing values, and outliers.
- Ensure data consistency and reliability for further analysis.

#### **4) Data Transformation:**

- Transform the cleaned data into a suitable format for analysis.
- Reshape, aggregate, or derive new features as needed for analysis.

#### **5) Data Analysis with Spark SQL:**

- Utilize Spark SQL to perform advanced analytics on the transformed data.
- Write SQL queries to extract insights, perform aggregations, and apply statistical functions.

#### **6) Error Detection:**

- Implement algorithms or techniques to detect errors in the data.
- Identify inconsistencies, anomalies, or patterns deviating from expected behavior.

#### **7) Seasonality Analysis:**

Analyze the data for seasonality patterns to understand recurring trends or fluctuations over time.

## 8) Anomaly Detection:

- Develop models or algorithms to detect anomalies in the data.
- Use outlier detection, anomaly scoring ( using z-score) , or machine learning-based techniques for anomaly detection.

## 9) Visualization and Reporting:

- Visualize analyzed data and insights using interactive visualizations or reports.
- Communicate findings to stakeholders for data-driven decision-making.

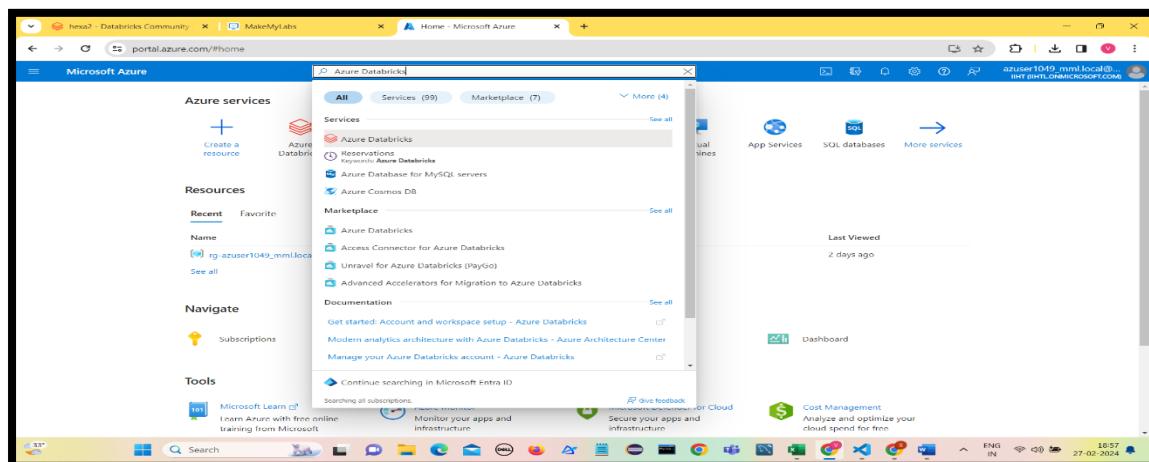
## Overall Execution Flow:

- 1) Provision Azure Databricks workspace in Azure portal.
- 2) Ingest raw data from sources into Azure Databricks.
- 3) Explore data using Databricks notebooks for structure and quality.
- 4) Clean data by handling missing values, duplicates, and inconsistencies.
- 5) Transform cleaned data into suitable format for analysis.
- 6) Apply transformations such as reshaping and aggregating using Spark SQL.
- 7) Write Spark SQL queries in Databricks notebooks for advanced analytics.
- 8) Analyze data for errors, seasonality patterns, and anomalies using SQL queries.
- 9) Implement algorithms in Spark SQL for error detection and anomaly detection.
- 10) Analyze data for seasonality patterns using SQL queries and functions.

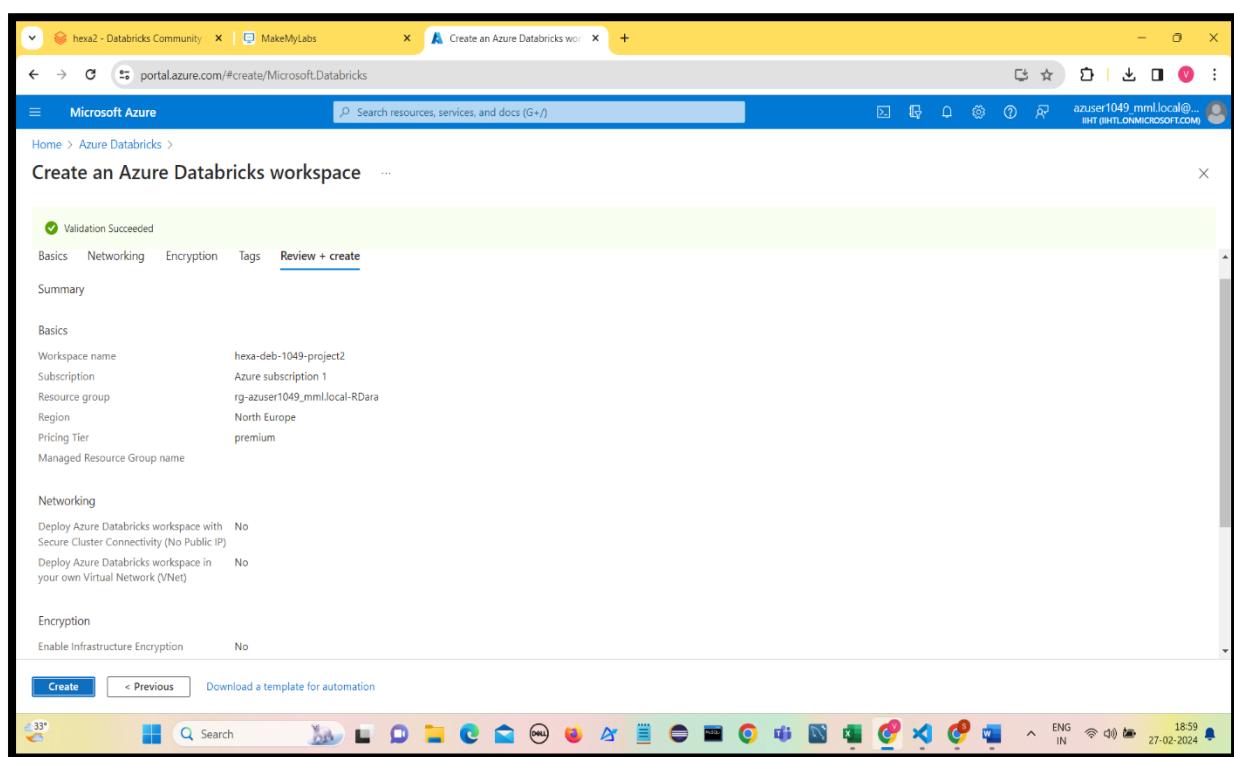
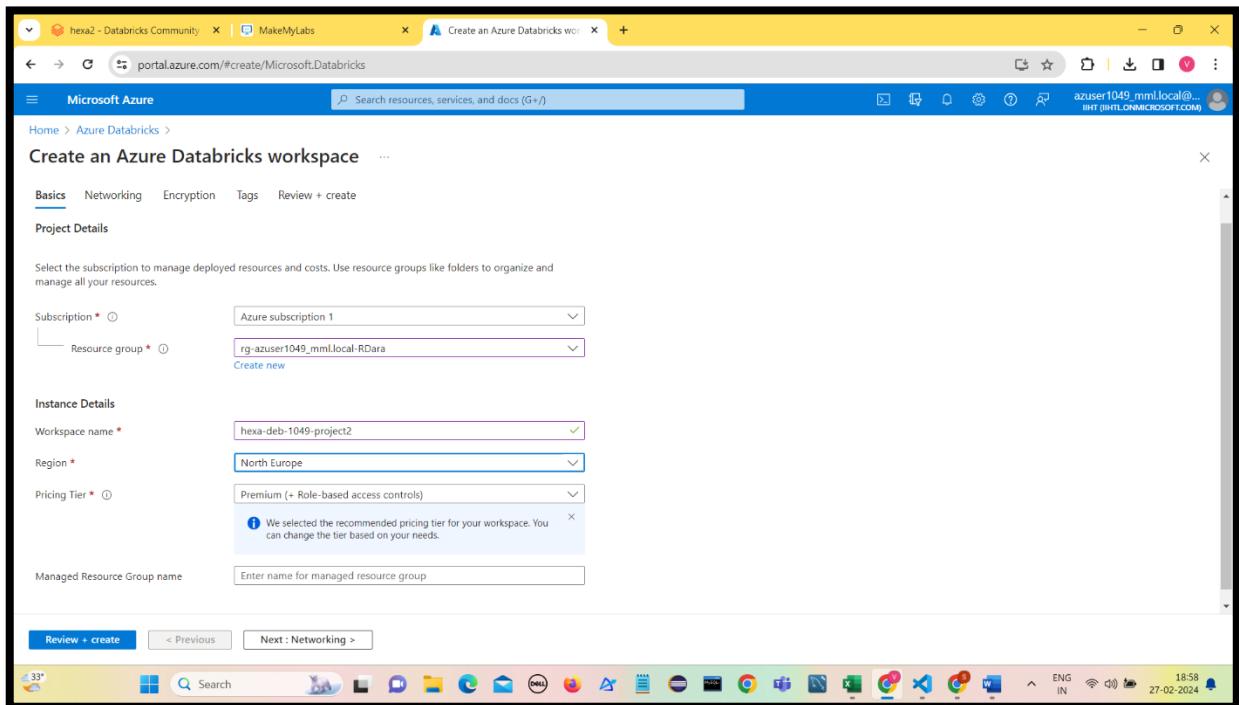
## Execution of Project:

### Creating Azure Databricks Workspace

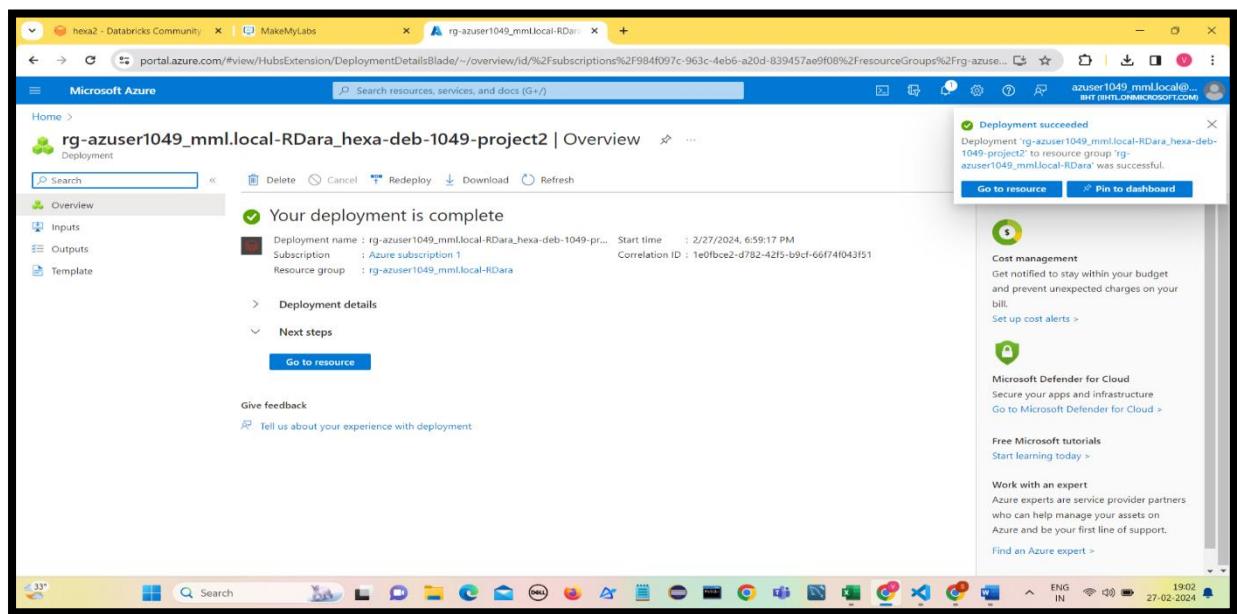
- 1) Log in into the Azure Portal and Search for Azure Databricks



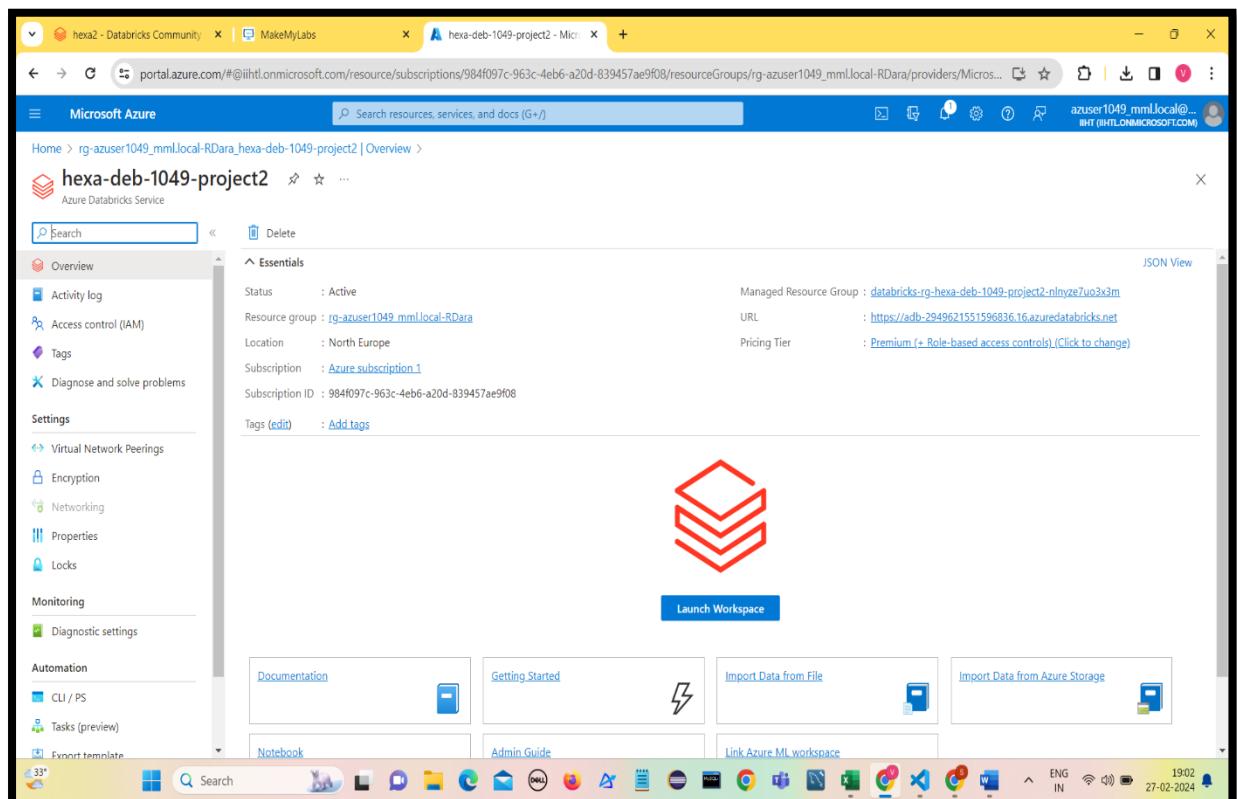
- 2) Create an Azure Databricks workspace by selecting Subscription, Resource Group, Workspace name(hexa-deb-1049-project2),Region,Pricing Tier, Click on **Review+Create**.



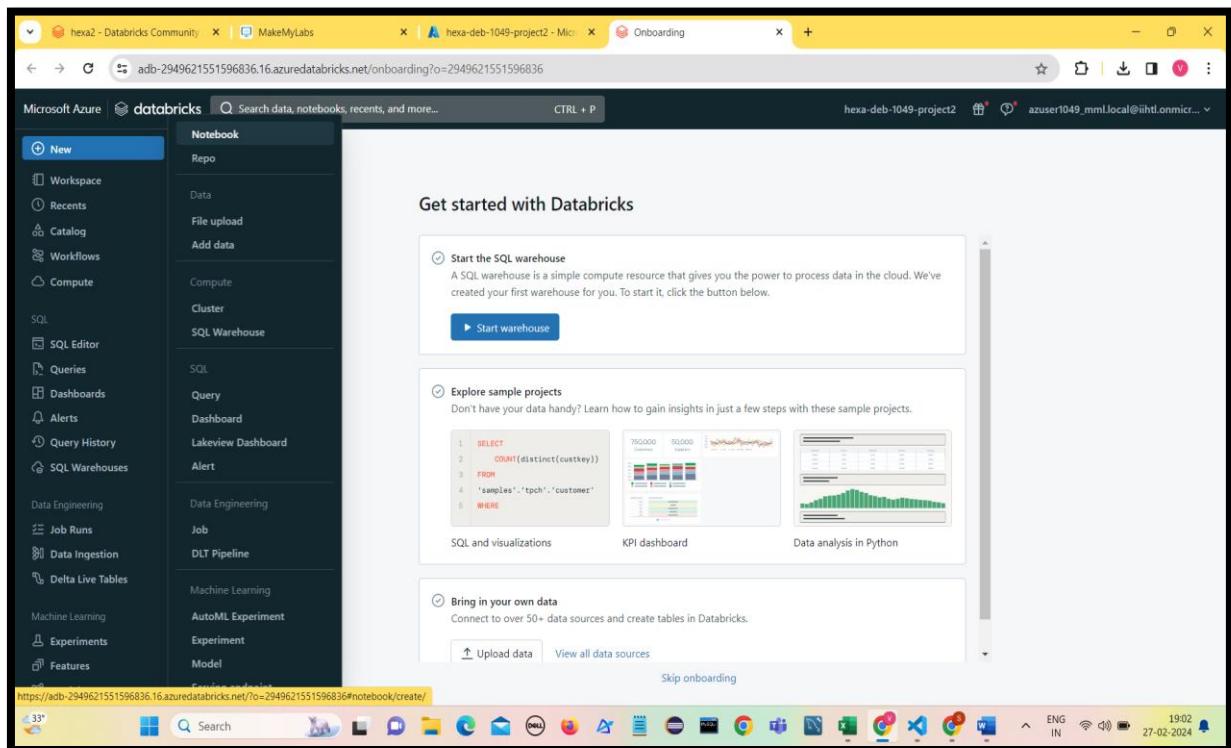
3) After the Validation is Succeeded , Click on Create.



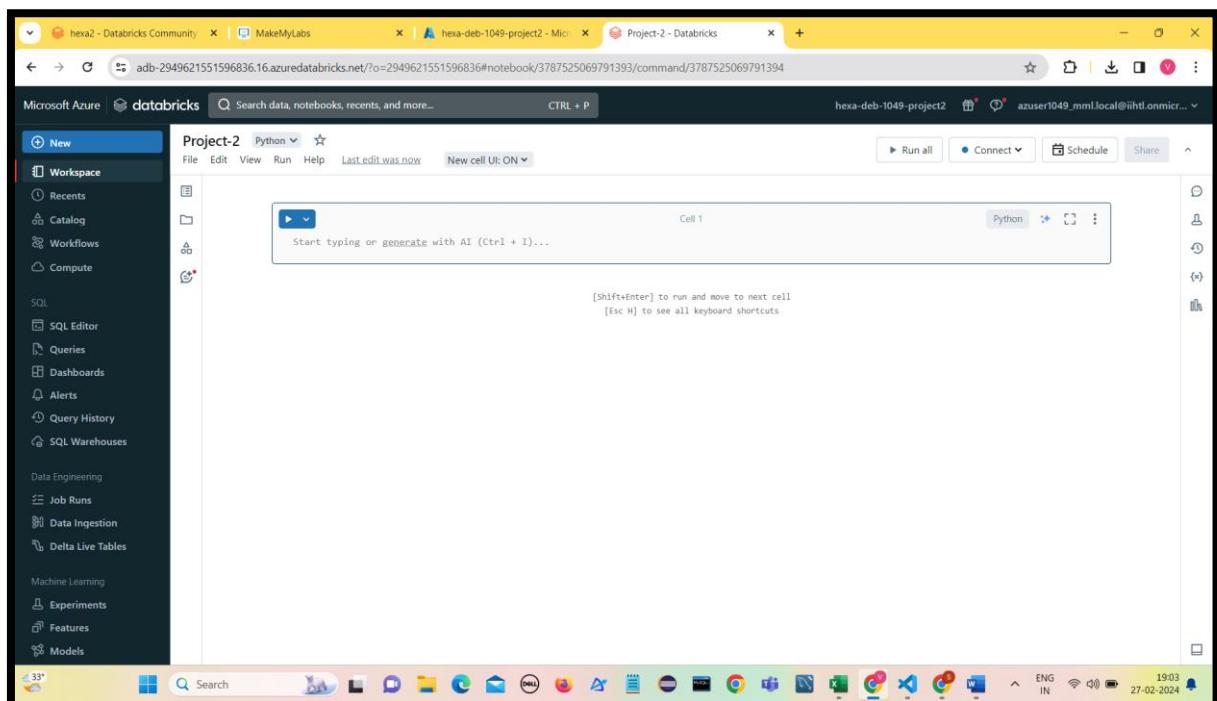
4) Once deployment is complete, Click on Go to resource.



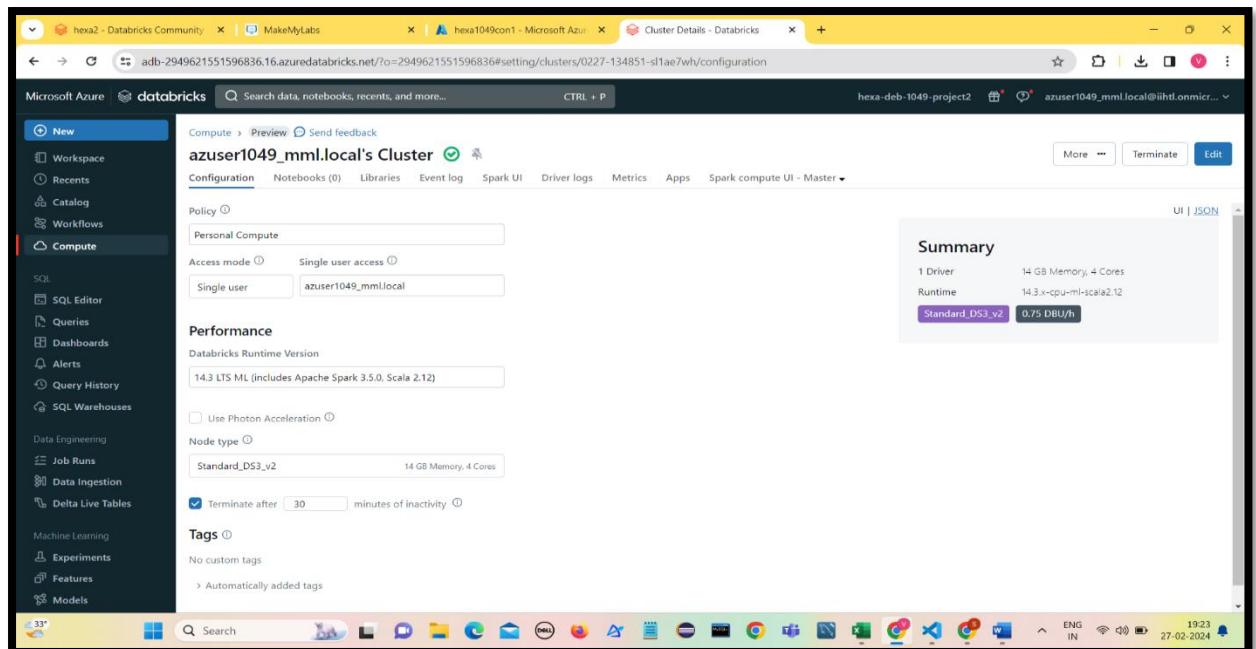
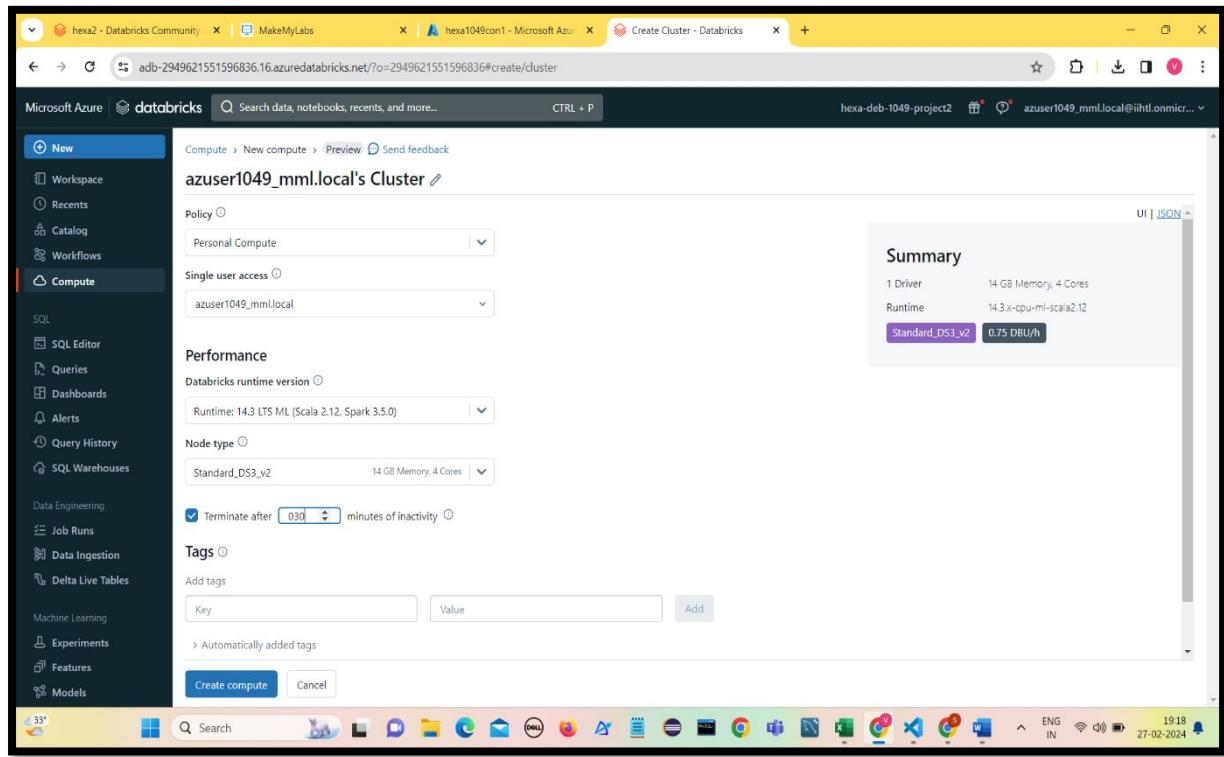
5) Click on Launch Workspace.



6) To create a Databricks notebook , Click New and click Notebook.



7) Create Compute by choosing Policy , Version, Node time etc., and Click on Create Compute,



8) Once the Cluster is created, we can use it to run the PySpark Notebooks.

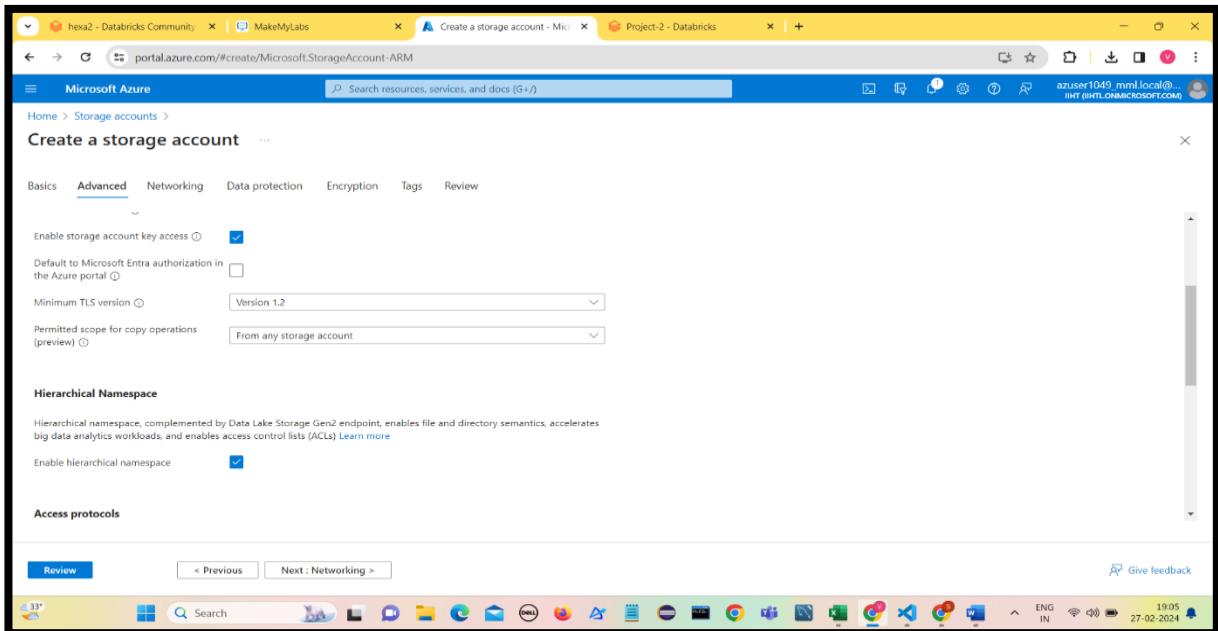
## Creating ADLS Gen2 account (hexaadls1049)

- 9) Search for Storage accounts in Azure portal.

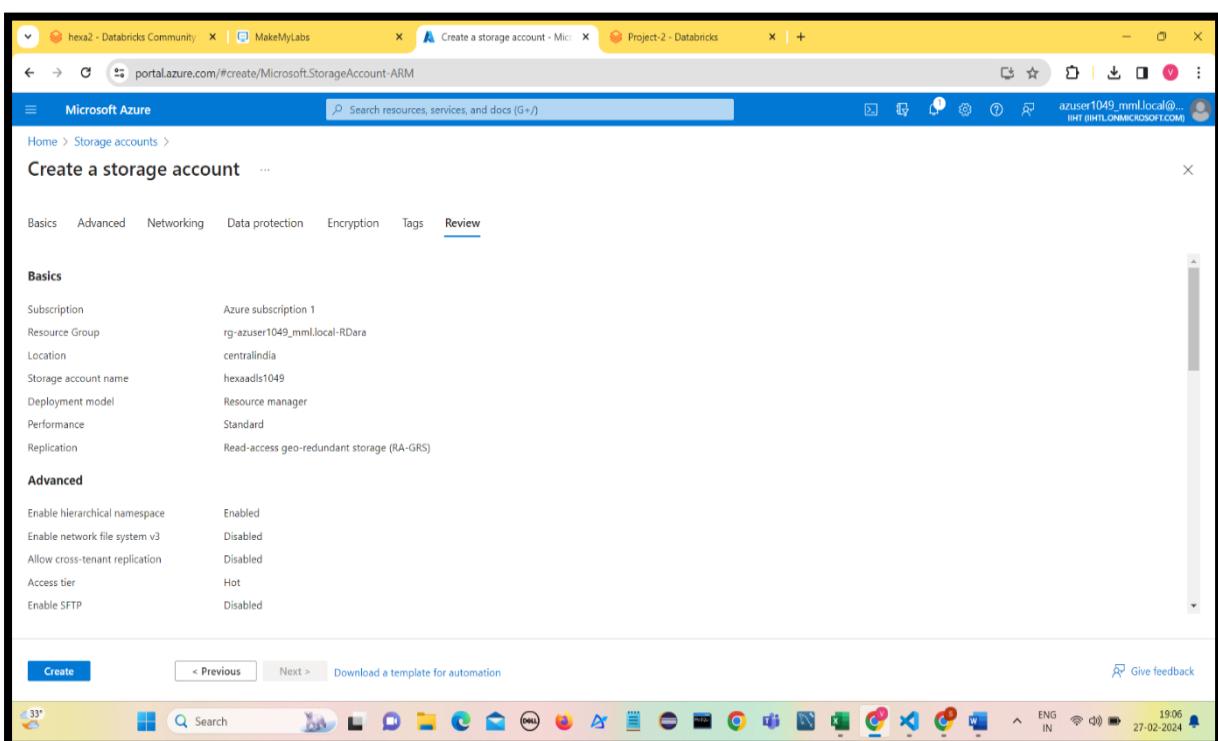
The screenshot shows the Microsoft Azure portal interface. The search bar at the top has 'Storage accounts' typed into it. Below the search bar, the main content area displays a list of services under the 'Services' category. The 'Storage accounts' item is highlighted. To the right of the search results, there are several cards: 'Last Viewed' (Storage account, Azure Storage Mover, Storage task - Azure Storage Actions, Storage Account Using ARM Template), 'Documentation' (Quickstart: Integrate an Azure Storage account with Azure CDN, Get storage account configuration information - Azure Storage, Microsoft.Storage/storageAccounts - Bicep, ARM template & Terraform AzAPI r...), and a 'Dashboard' button. The bottom of the screen shows the Windows taskbar with various pinned icons.

- 10) To create a ADLS account choose the Subscription, Storage account name ( hexaadls1049), Region,Performance etc., and Click on **Review**.

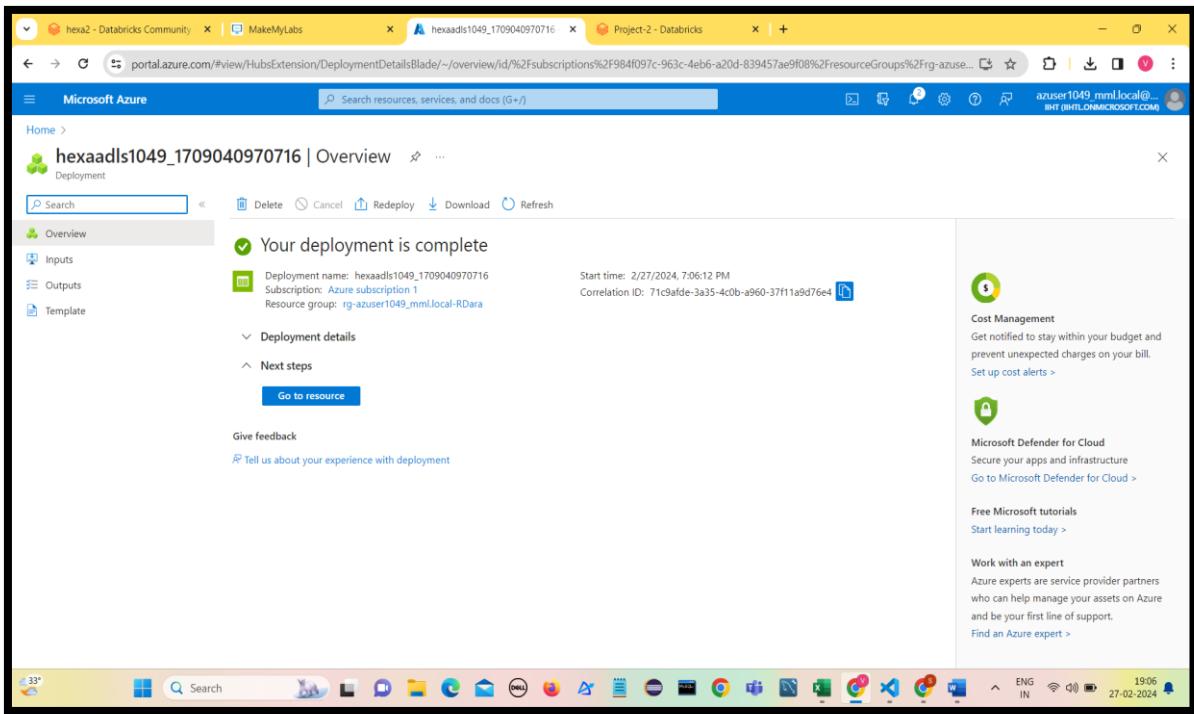
The screenshot shows the 'Create a storage account' wizard in the Microsoft Azure portal. The current step is 'Basics'. The 'Storage account name' field is filled with 'hexaadls1049'. The 'Region' dropdown shows '(Asia Pacific) Central India'. Under 'Performance', the 'Standard' radio button is selected. Under 'Redundancy', 'Geo-redundant storage (GRS)' is chosen. At the bottom, there are 'Review' and 'Next : Advanced >' buttons. The taskbar at the bottom of the screen shows various pinned icons.



11) Enable Hierarchical Namespace in Advanced Settings and Click on Review.

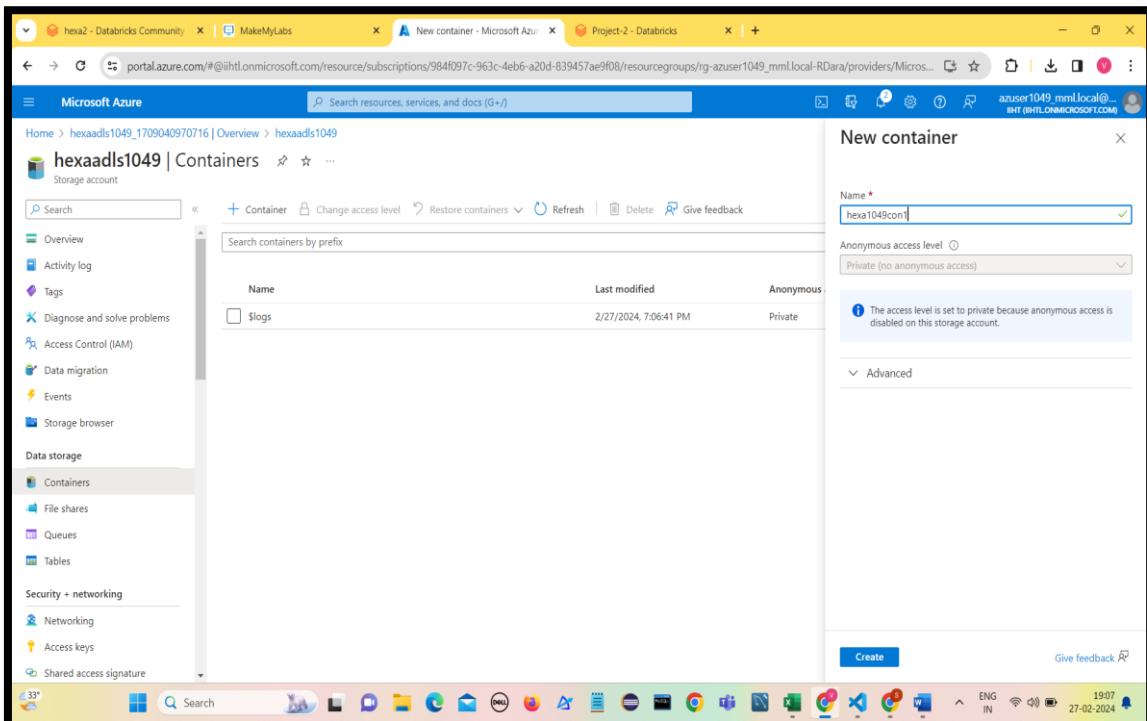


12) Click on Create and Deployment is in process.



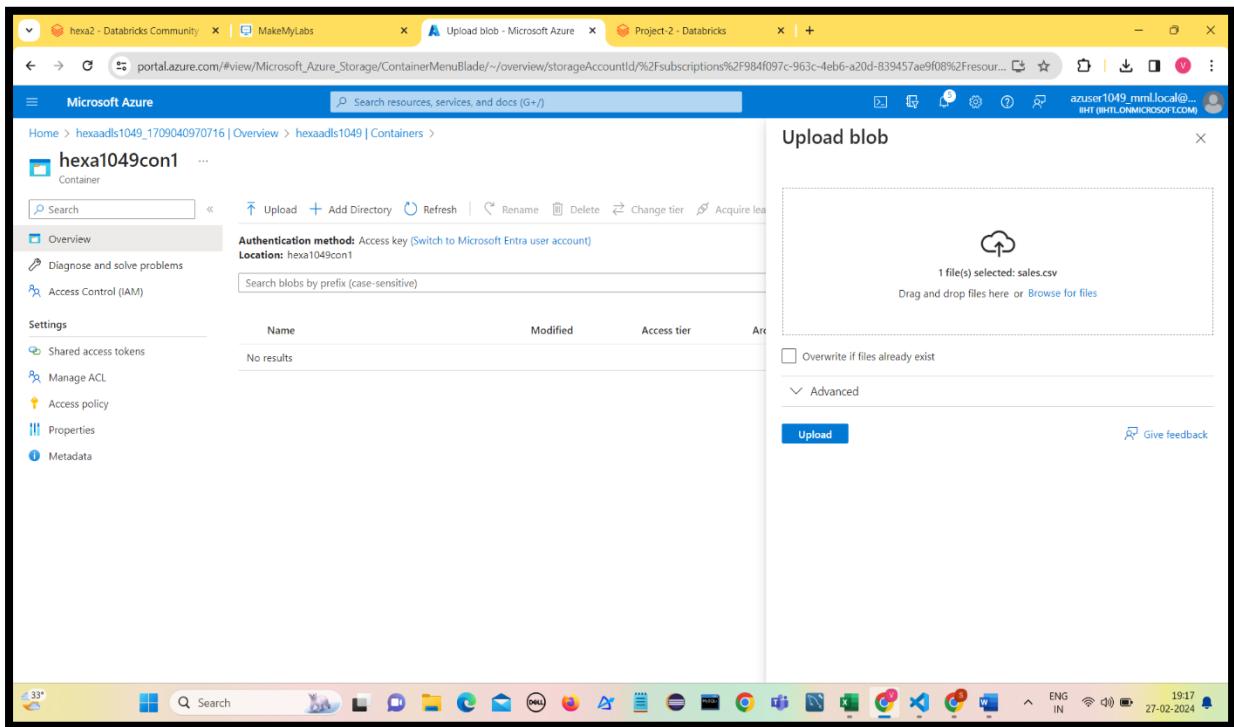
13) Click on resource.

14) Open the hexaadls1049 and Click on Containers and enter the Name of container.

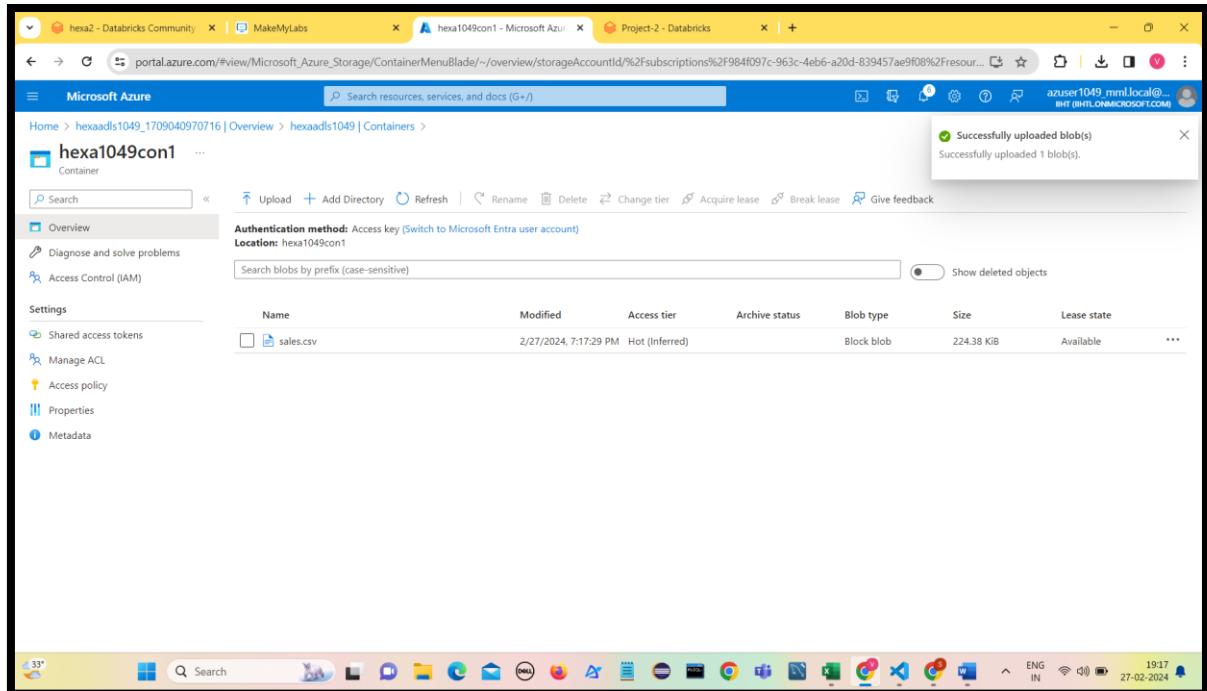


15) Click on Create.

16) Select the file to upload into the container.



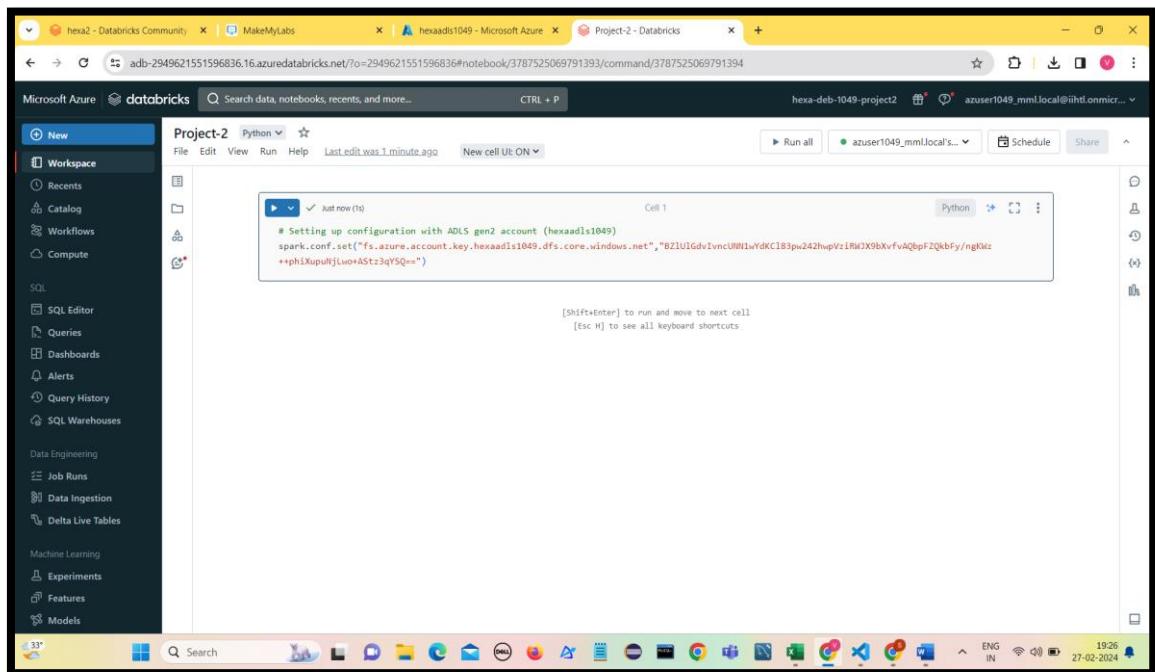
17) Click on Upload



18) sales.csv is added to the container.



## 19) Setting up configuration with ADLS gen2 account (hexaadls1049)

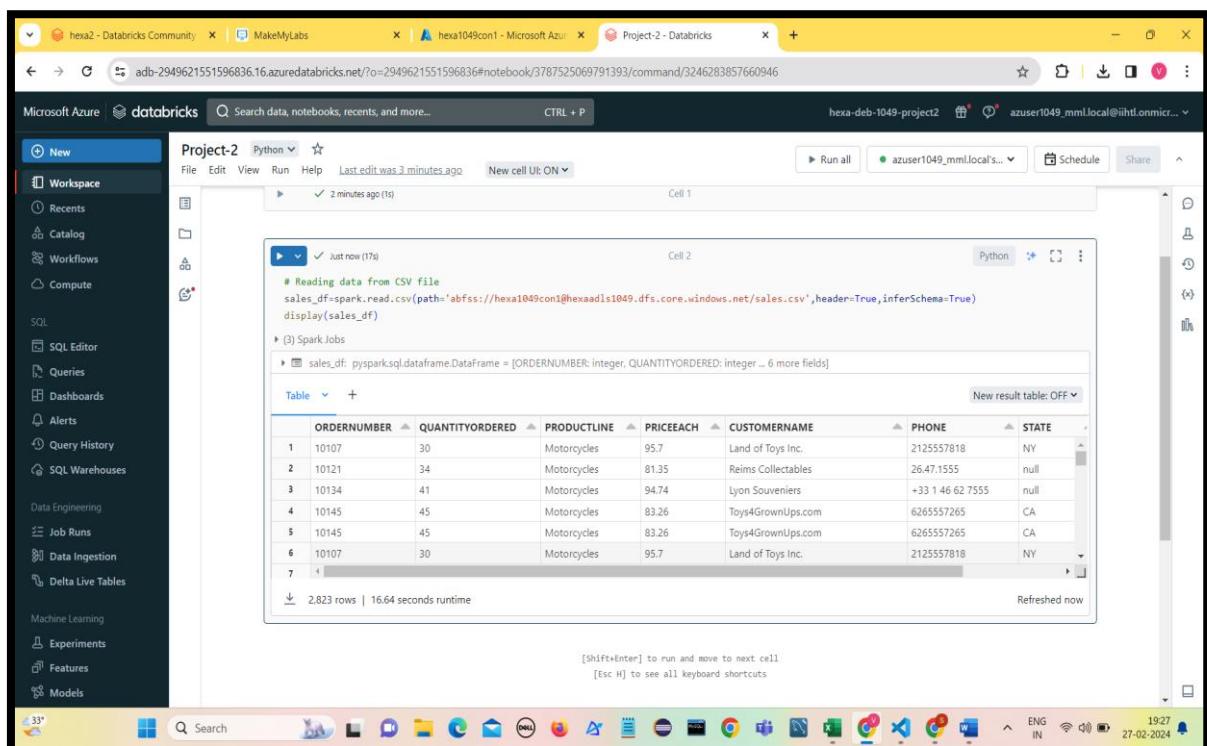


The screenshot shows a Databricks notebook titled "Project-2" in Python. The code cell contains the following configuration snippet:

```
# Setting up configuration with ADLS gen2 account (hexaadls1049)
spark.conf.set("fs.azure.account.key.hexaadls1049.dfs.core.windows.net","B2lU1Gdv1ncURN1wYdKC1B3pu242hwpVz1RnJX9bXfvAQBpFZQkbFy/ngKwz
+phiXupuhjLwu+ASTz3qYSQ==")
```

The notebook interface includes a sidebar with various workspace, catalog, and compute options.

## 20) Reading data from sales.csv file using read.csv



The screenshot shows a Databricks notebook titled "Project-2" in Python. The code cell contains the following code to read data from a CSV file:

```
# Reading data from CSV File
sales_df=spark.read.csv(path='abfss://hexa1049con1@hexaadls1049.dfs.core.windows.net/sales.csv',header=True,inferSchema=True)
display(sales_df)
```

The notebook displays the resulting DataFrame, which contains 2,823 rows. The columns include ORDERNUMBER, QUANTITYORDERED, PRODUCTLINE, PRICEEACH, CUSTOMERNAME, PHONE, and STATE. The data is as follows:

ORDERNUMBER	QUANTITYORDERED	PRODUCTLINE	PRICEEACH	CUSTOMERNAME	PHONE	STATE	
1	10107	30	Motorcycles	95.7	Land of Toys Inc.	2125557818	NY
2	10121	34	Motorcycles	81.35	Reims Collectables	26.47.1555	null
3	10134	41	Motorcycles	94.74	Lyon Souveniers	+33 1 46 62 7555	null
4	10145	45	Motorcycles	83.26	Toys4GrownUps.com	6265557265	CA
5	10145	45	Motorcycles	83.26	Toys4GrownUps.com	6265557265	CA
6	10107	30	Motorcycles	95.7	Land of Toys Inc.	2125557818	NY
7	1						

The screenshot shows a Microsoft Azure Databricks workspace titled "Project-2". On the left is a sidebar with various menu items like Workspace, Recents, Catalog, Workflows, Compute, SQL, and Machine Learning. The main area has a search bar and a "Run all" button. A table output is shown above a code cell. The code cell contains the following Python code:

```
sales_df.show()
```

The table output shows data from a DataFrame named sales\_df. The columns are ORDERNUMBER, QUANTITYORDERED, PRODUCTLINE, PRICEEACH, CUSTOMERNAME, PHONE, STATE, and ORDERDATE. The data includes rows for various customers like Land of Toys Inc., Reims Collectables, Lyon Souveniers, etc., across different states (NY, CA, MA) and dates.

## Data Exploration

### Display the DataFrame Schema.

The screenshot shows a Microsoft Azure Databricks workspace titled "Project-2". The sidebar and interface are identical to the previous screenshot. The main area shows a code cell with the following Python code:

```
# Data Exploration  
# Display the DataFrame schema  
sales_df.printSchema()
```

Below the code, the schema is displayed as:

```
root  
|-- ORDERNUMBER: integer (nullable = true)  
|-- QUANTITYORDERED: integer (nullable = true)  
|-- PRODUCTLINE: string (nullable = true)  
|-- PRICEEACH: double (nullable = true)  
|-- CUSTOMERNAME: string (nullable = true)  
|-- PHONE: string (nullable = true)  
|-- STATE: string (nullable = true)  
|-- ORDERDATE: string (nullable = true)
```

A note at the bottom of the code cell says "[Shift+Enter] to run and move to next cell [Esc H] to see all keyboard shortcuts".

## Create a temporary view for Spark SQL Operations.

A screenshot of a Databricks notebook titled "Project-2". The sidebar on the left shows various Databricks services like Workspace, Catalog, Workflows, Compute, SQL, and Machine Learning. The main area contains two code cells. Cell 4 contains the following Python code:

```
sales_df.printSchema()
```

Cell 5 contains the following Python code:

```
# Create a temporary view for Spark SQL operations
sales_df.createOrReplaceTempView("sales")
```

The status bar at the bottom right shows the date as 27-02-2024 and the time as 19:29.

## Display the details of sales view.

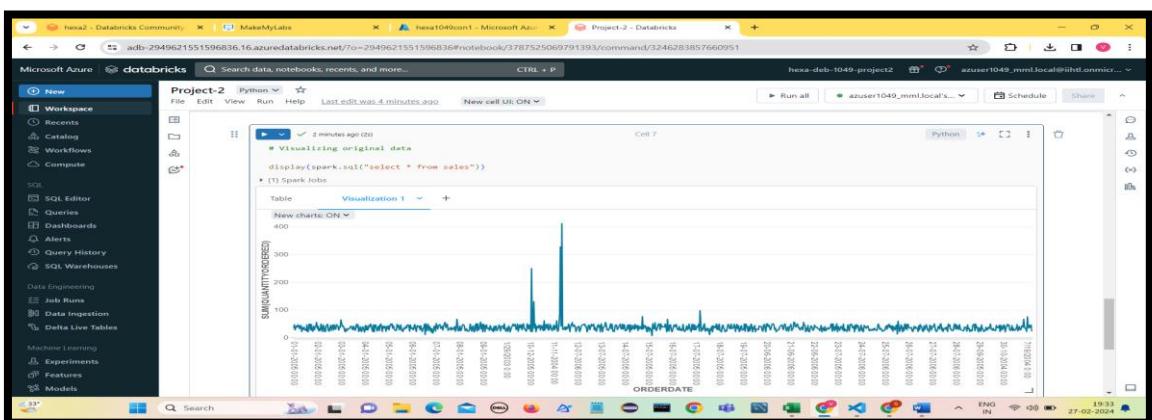
A screenshot of a Databricks notebook titled "Project-2". The sidebar on the left shows various Databricks services like Workspace, Catalog, Workflows, Compute, SQL, and Machine Learning. The main area contains a code cell with the following Python code:

```
spark.sql("select * from sales").show()
```

The resulting table has the following schema:

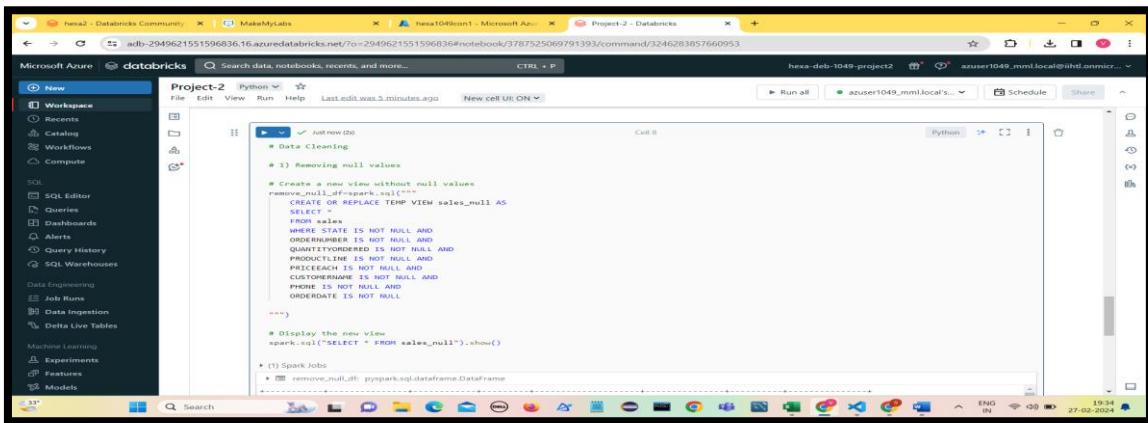
ORDERNUMBER	QUANTITYORDERED	PRODUCTLINE	PRICEEACH	CUSTOMERNAME	PHONE	STATE	ORDERDATE
10107	30	Motorcycles	95.71	Land & Sea Inc.	2125573218	NY	7/23/2004 0:00
10111	34	Motorcycles	98.57	Reich Collectables	2085555555	NULL	11/20/2003 0:00
10134	41	Motorcycles	94.74	Lyon Souvenirs(+3) 1 46 62 7555	NULL	10/28/2003 0:00	
10145	45	Motorcycles	83.26	Toys4Groups.com	6265572651	CA	10/28/2003 0:00
10145	45	Motorcycles	83.26	Toys4Groups.com	6265572651	CA	8/25/2003 0:00
10177	39	Motorcycles	95.71	Land & Sea Inc.	2125573218	NY	7/23/2004 0:00
10180	29	Motorcycles	86.13	Daedalus Designs ....	2016155555	NULL	10/15/2004 0:00
10188	48	Motorcycles	100.01	Herkis Gifts(+47 2267 3235)	NULL	11/18/2003 0:00	
10201	22	Motorcycles	98.57	Mini Wheels Co.	6585557871	CA	10/28/2003 0:00
10211	22	Motorcycles	98.57	Mini Wheels Co.	6585557871	CA	10/28/2003 0:00
10223	37	Motorcycles	100.01	Australium College...	03 9520 4551	Victoria	2/20/2004 0:00
10237	23	Motorcycles	100.01	Vlachrome Inc.	2125551500	NY	10/15/2004 0:00
10237	23	Motorcycles	100.01	Vlachrome Inc.	2125551500	NY	10/15/2004 0:00
10238	34	Motorcycles	100.01	Globe Imports(+20 33 77 00)	2033770000	NY	10/15/2004 0:00
10275	45	Motorcycles	92.83	Le Rochelle Gifts	40.67.8555	NULL	7/23/2004 0:00
10285	36	Motorcycles	100.01	Harto's Replicas Co.	6175585551	PA	8/27/2004 0:00
10309	21	Motorcycles	100.01	Tony of Florida Co.	9047555551	FL	9/10/2004 0:00
10309	41	Motorcycles	100.01	Bonne Mere Import(+87 39 85551)	NULL	10/15/2004 0:00	

## Visualizing original data using Line Chart.



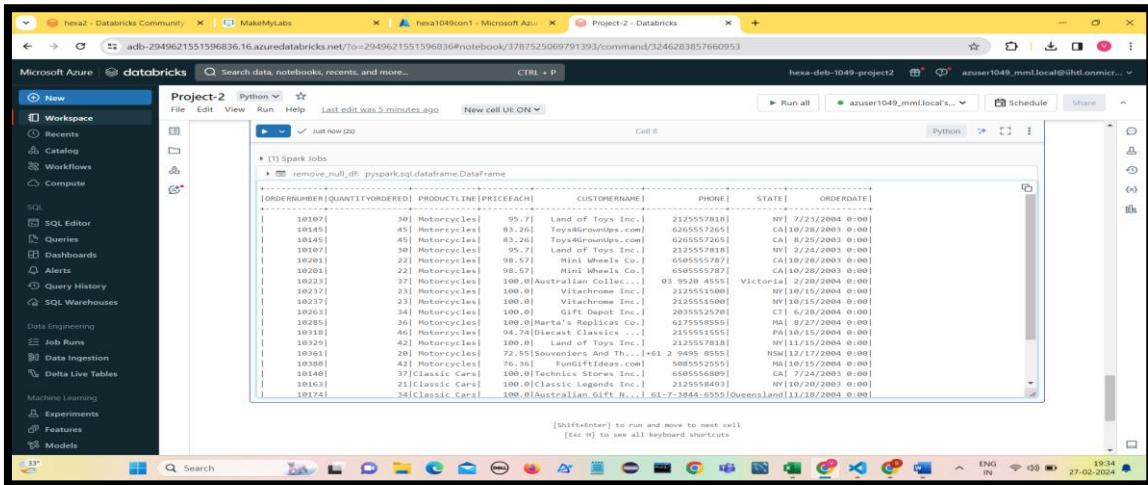
# Data Cleaning

## Removing null values



```
Project-2 Python Just now (2s) Data Cleaning
# 1) Removing null values
# Create a new view without null values
remove_null_df=spark.sql("""
CREATE OR REPLACE TEMP VIEW sales_null AS
SELECT *
FROM sales
WHERE STATE IS NOT NULL AND
ADDRESS IS NOT NULL AND
QUANTITYORDERED IS NOT NULL AND
PRODUCTLINE IS NOT NULL AND
PRICEUNIT IS NOT NULL AND
CUSTOMERNAME IS NOT NULL AND
PHONE IS NOT NULL AND
ORDERDATE IS NOT NULL
""")

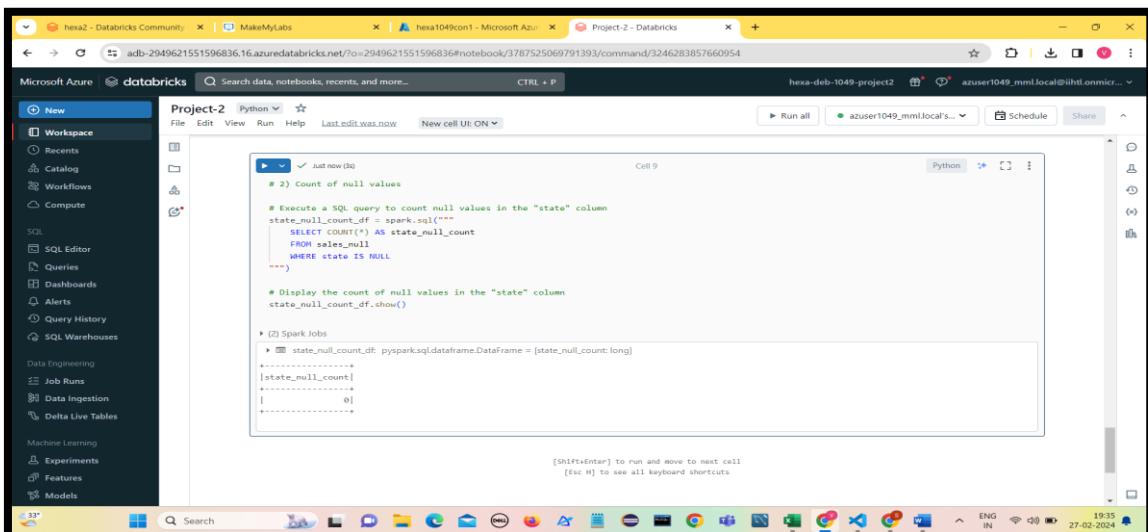
# Display the new view
spark.sql("SELECT * FROM sales_null").show()
```



The screenshot shows the Databricks notebook interface with the results of the previous command. A table named 'sales\_null' is displayed, showing data from the 'sales' table with all null values removed. The columns are: ORDERID, QUANTITYORDERED, PRODUCTLINE, PRICEUNIT, CUSTOMERNAME, PHONE, STATE, and ORDERDATE.

ORDERID	QUANTITYORDERED	PRODUCTLINE	PRICEUNIT	CUSTOMERNAME	PHONE	STATE	ORDERDATE
10107	30	Motorcycles	95.71	Land of Toys Inc.	2125578181	NY	7/23/2003 0:00
10145	40	Motorcycles	83.01	Toys4Groups.com	6265557265	CA	8/25/2003 0:00
10151	45	Motorcycles	83.26	Toyland	6265557265	CA	8/25/2003 0:00
10107	30	Motorcycles	95.71	Land of Toys Inc.	2125578181	NY	2/24/2003 0:00
10201	22	Motorcycles	98.57	Mini Wheels Co.	6505555787	CA	10/28/2003 0:00
10211	24	Motorcycles	93.57	Mini Wheels Co.	6505555787	CA	10/28/2003 0:00
10223	37	Motorcycles	100.01	Australian Collector	03 9528 4555	Victoria	2/28/2004 0:00
10237	23	Motorcycles	100.01	Vitachrome Inc.	2125551500	NY	10/15/2004 0:00
10237	23	Motorcycles	100.01	Vitachrome Inc.	2125551500	NY	10/15/2004 0:00
10241	34	Motorcycles	100.01	Land of Toys Inc.	2125551500	CA	10/15/2004 0:00
10285	36	Motorcycles	100.01	Marta's Replicas Co.	6175559555	MA	8/27/2004 0:00
10318	46	Motorcycles	94.74	[deleat Classic ...]	2155551555	PA	10/15/2004 0:00
10329	42	Motorcycles	100.01	Land of Toys Inc.	2125578181	NY	11/15/2004 0:00
10331	20	Motorcycles	72.55	[deleat Classic ...]	1463 55555555	MA	8/27/2004 0:00
10398	42	Motorcycles	76.36	FunGiftIdeas.com	5085552555	MA	10/15/2004 0:00
10140	37	Classic Cars	100.01	Technics Stores Inc.	6505556809	CA	7/24/2003 0:00
10163	21	Classic Cars	100.01	Classic Legend Inc.	2125558601	CA	10/20/2003 0:00
10174	34	Classic Cars	100.01	Australian Gif...	03 9528 4555	Victoria	11/10/2004 0:00

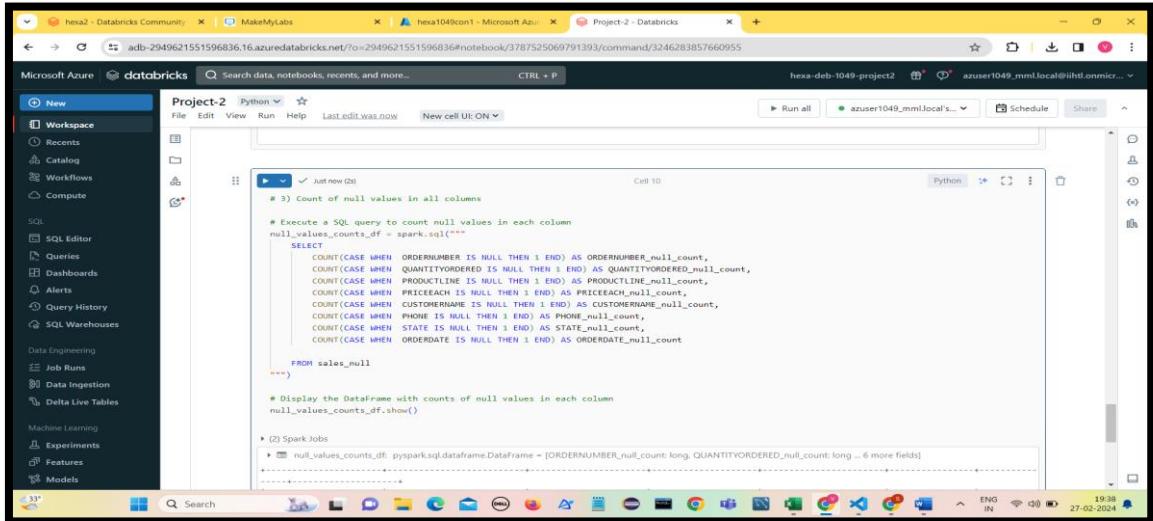
## Count of null values in State column.



```
Project-2 Python Just now (3s) # 2) Count of null values
# Execute a SQL query to count null values in the "state" column
state_null_count_df = spark.sql("""
SELECT COUNT(*) AS state_null_count
FROM sales_null
WHERE state IS NULL
""")

# Display the count of null values in the "state" column
state_null_count_df.show()
```

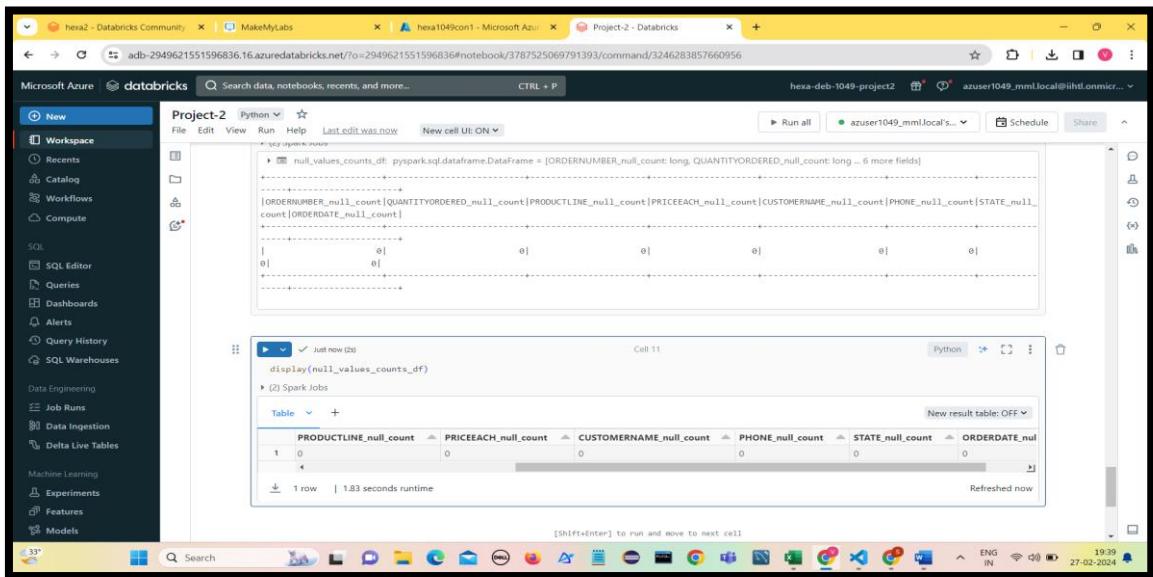
## Count of Null values in all columns.



```
# 3) Count of null values in all columns

# Execute a SQL query to count null values in each column
null_values_counts_df = spark.sql("""
    SELECT
        COUNT(CASE WHEN ORDERNUMBER IS NULL THEN 1 END) AS ORDERNUMBER_null_count,
        COUNT(CASE WHEN QUANTITYORDERED IS NULL THEN 1 END) AS QUANTITYORDERED_null_count,
        COUNT(CASE WHEN PRODUCTLINE IS NULL THEN 1 END) AS PRODUCTLINE_null_count,
        COUNT(CASE WHEN PRICEACH IS NULL THEN 1 END) AS PRICEACH_null_count,
        COUNT(CASE WHEN CUSTOMERNAME IS NULL THEN 1 END) AS CUSTOMERNAME_null_count,
        COUNT(CASE WHEN PHONE IS NULL THEN 1 END) AS PHONE_null_count,
        COUNT(CASE WHEN STATE IS NULL THEN 1 END) AS STATE_null_count,
        COUNT(CASE WHEN ORDERDATE IS NULL THEN 1 END) AS ORDERDATE_null_count
    FROM sales_null
""")

# Display the DataFrame with counts of null values in each column
null_values_counts_df.show()
```



```
+-----+-----+-----+-----+-----+-----+-----+
| null_values_counts_df: pyspark.sql.DataFrame = [ORDERNUMBER_null_count: long, QUANTITYORDERED_null_count: long ... 6 more fields] |
+-----+-----+-----+-----+-----+-----+-----+
|          0|          0|          0|          0|          0|          0|          0|
+-----+-----+-----+-----+-----+-----+-----+
```

Cell 10

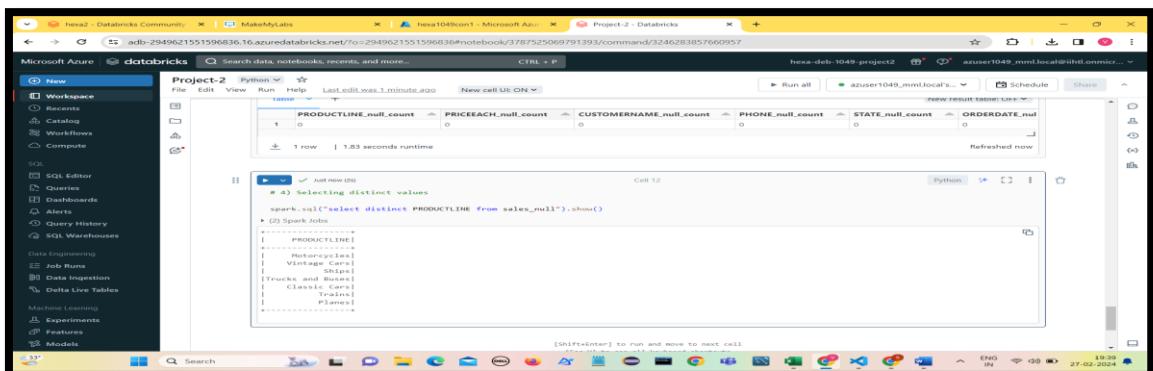
```
display(null_values_counts_df)
```

(2) Spark Jobs

PRODUCTLINE_null_count	PRICEACH_null_count	CUSTOMERNAME_null_count	PHONE_null_count	STATE_null_count	ORDERDATE_null_count
1	0	0	0	0	0

1 row | 1.83 seconds runtime

## Selecting distinct PRODUCTLINE values



```
PRODUCTLINE_null_count  PRICEACH_null_count  CUSTOMERNAME_null_count  PHONE_null_count  STATE_null_count  ORDERDATE_null_count
1 row | 1.83 seconds runtime
```

Cell 11

```
# 4) Selecting distinct values
spark.sql("select distinct PRODUCTLINE from sales_null").show()
```

(2) Spark Jobs

PRODUCTLINE
Motorcycles
Vans
Ships
Trucks and Buses
Classic Cars
Pranks

## Removing duplicates from all columns

The screenshot shows a Databricks workspace titled "Project-2". In Cell 13, Python code is run to create a new view without duplicates:

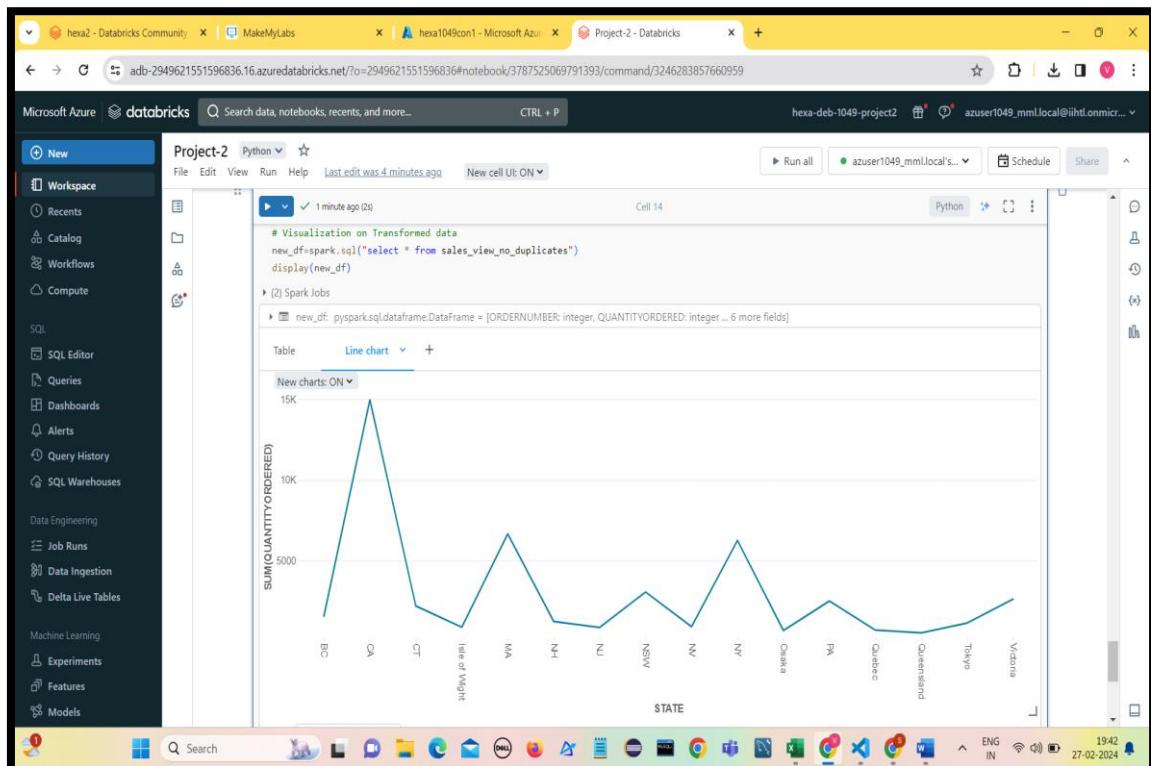
```
# 5) Removing duplicates
distinct_df_spark.sql("""
    CREATE OR REPLACE TEMP VIEW sales_view_no_duplicates AS
    SELECT DISTINCT *
    FROM sales_null
""")

# Display the new view without duplicates
spark.sql("SELECT * FROM sales_view_no_duplicates").show()
```

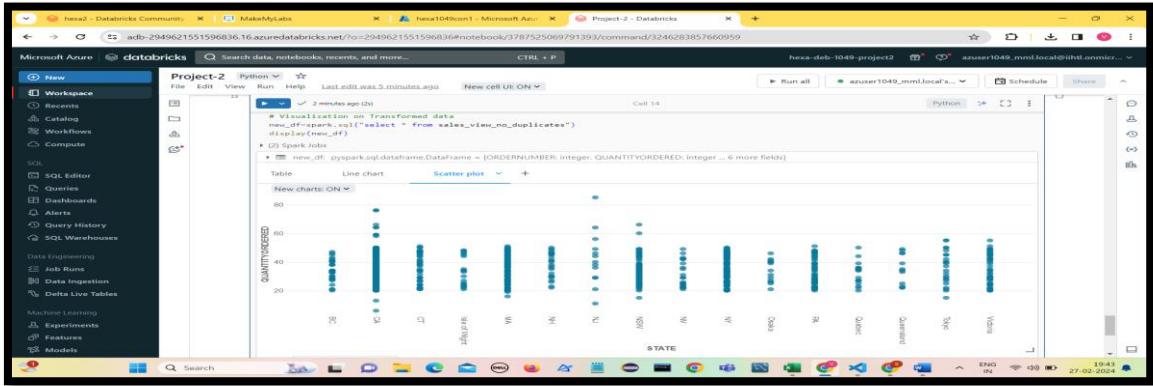
The resulting DataFrame view (Cell 13) displays data with the following schema:

ORDERNUMBER	QUANTITYORDERED	PRODUCTLINE	PRICEEACH	CUSTOMERNAME	PHONE	STATE	ORDERDATE
10163	21	Classic Cars	100.0	Classic Legends Inc.	2125558493	NY	10/20/2003 0:00
10228	29	Classic Cars	100.0	Cambridge Collect...	6175555555	MA	11/18/2004 0:00
10329	42	Motorcycles	100.0	Land of Toys Inc.	212557818	NY	11/15/2003 0:00
10245	34	Classic Cars	100.0	Super Scale Inc.	203559545	CT	11/25/2003 0:00
10223	37	Motorcycles	100.0	Australian Collec...	03 9520 4555	Victoria	2/20/2004 0:00
10215	35	Classic Cars	100.0	West Coast Collect...	310553722	CA	1/29/2004 0:00
10174	34	Classic Cars	100.0	Australian Gift N...	61-7-3844-6555	Queensland	11/18/2004 0:00
10187	30	Motorcycles	95.7	Land of toys Inc.	212557818	NY	2/24/2003 0:00
10285	36	Motorcycles	100.0	Marta's Replicas Co.	6175558555	PA	8/27/2004 0:00
10183	23	Classic Cars	100.0	Classic Gift Idea...	215554695	PA	11/13/2003 0:00
10281	22	Motorcycles	98.57	Mini Wheels Co.	650555787	CA	10/28/2003 0:00

## Visualization on Transformed data ( Line chart)

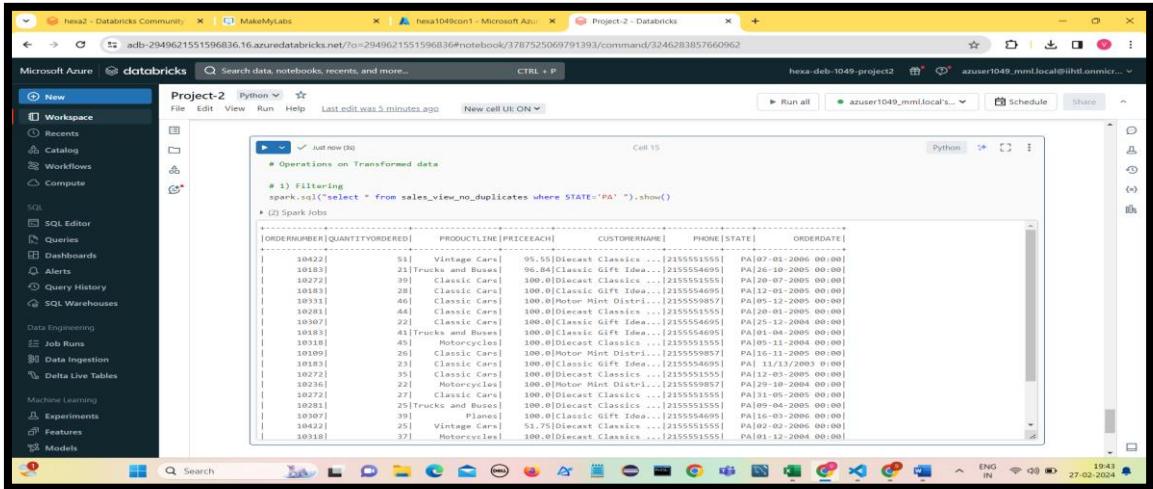


## Scatter plot

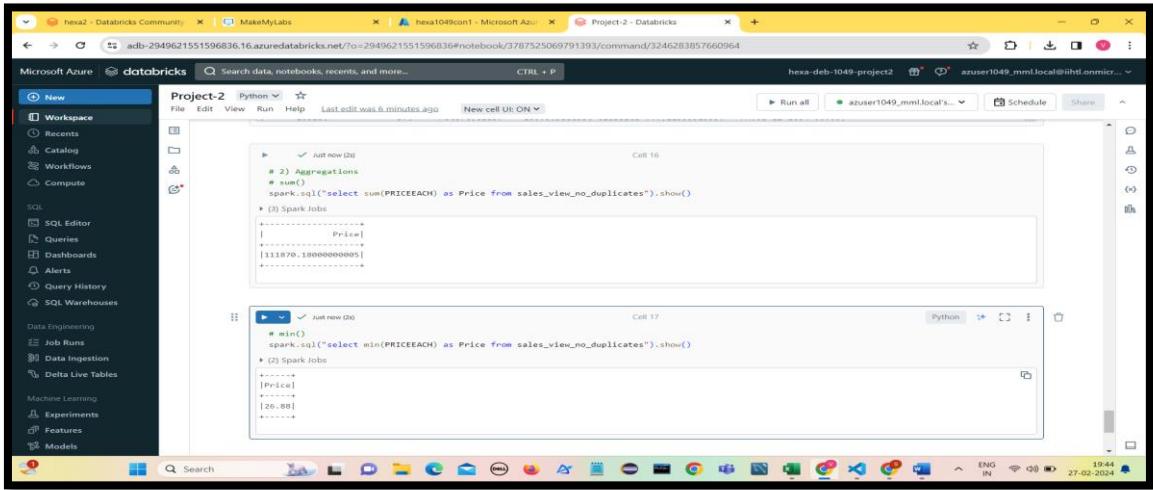


## Operations on Transformed data

### 1) Filtering



### 2) Aggregations



```

# max()
spark.sql("select max(PRICEACH) as Max_Price from sales_view_no_duplicates").show()

+-----+
| Max_Price |
+-----+
| 100.0 |
+-----+

```

```

# avg()
spark.sql("select avg(PRICEACH) as Avg_Price from sales_view_no_duplicates").show()

+-----+
| Avg_Price |
+-----+
| 83.73516467065872 |
+-----+

```

## 1) Anomalies Detection ( Calculating z-scores for each value)

The Z-score is a statistical measure that indicates how many standard deviations a data point is from the mean.

$$Z = (X - \mu) / \sigma$$

where

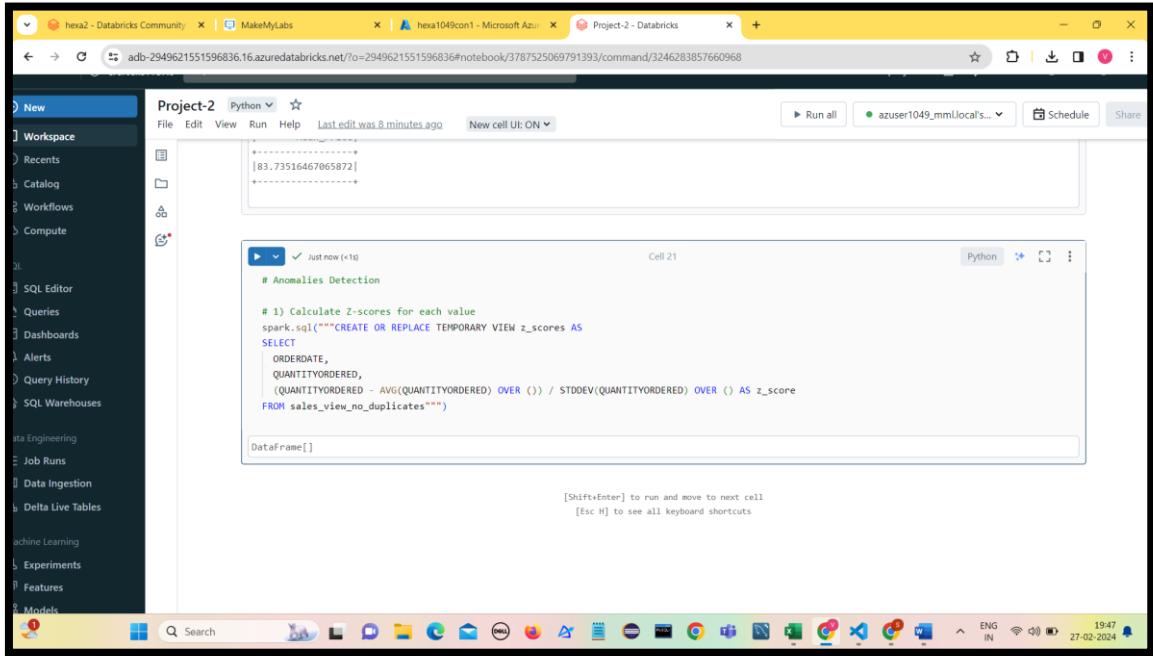
X is the data point,

$\mu$  is the mean, and

$\sigma$  is the standard deviation.

AVG(value) OVER () calculates the mean value of the entire dataset.

STDDEV(value) OVER () calculates the standard deviation of the entire dataset.



The screenshot shows a Databricks notebook interface. On the left is a sidebar with various navigation options like Workspace, Catalog, Workflows, Compute, etc. The main area has a title bar 'Project-2' and a Python tab. Below the title bar is a toolbar with 'Run all', 'Schedule', and 'Share' buttons. The notebook contains two cells:

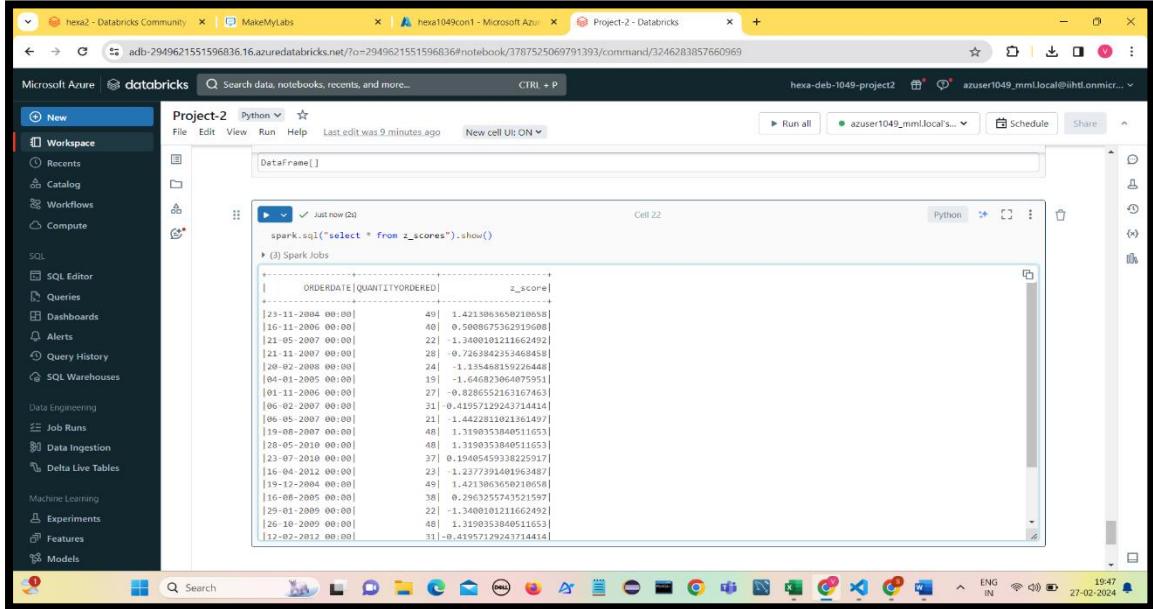
- Cell 21:** Contains the following Python code:

```
# Anomalies Detection

# 1) Calculate Z-scores for each value
spark.sql("""CREATE OR REPLACE TEMPORARY VIEW z_scores AS
SELECT
    ORDERDATE,
    QUANTITYORDERED,
    (QUANTITYORDERED - AVG(QUANTITYORDERED) OVER ()) / STDDEV(QUANTITYORDERED) OVER () AS z_score
FROM sales_view_no_duplicates""")
```
- Cell 22:** Shows the output of the previous command as a DataFrame. The output table has columns 'ORDERDATE' and 'z\_score'. The data is as follows:

ORDERDATE	z_score
23-11-2004 00:00	49  1.4213083659219058
16-11-2006 00:00	40  0.5000875362919008
22-05-2007 00:00	22  -1.340012116624921
12-03-2008 00:00	28  -0.1424224224224224
12-02-2008 00:00	24  -1.175458159253448
04-01-2005 00:00	19  -1.660833664079511
01-11-2006 00:00	27  -0.828552163167463
06-02-2007 00:00	31  0.4195729243714144
10-05-2007 00:00	21  -1.4422211021261497
19-06-2007 00:00	48  1.31903530404511053
28-05-2010 00:00	48  1.31903530404511053
23-07-2010 00:00	37  0.19005459338225917
16-04-2012 00:00	23  -1.2377391401963487
19-12-2004 00:00	45  1.4213083659219058
16-08-2005 00:00	38  0.29532557474521597
29-01-2009 00:00	22  -1.340012116624921
26-10-2009 00:00	48  1.31903530404511053
12-07-2012 00:00	31  -0.41957129243714144

## 2) Display of z-scores



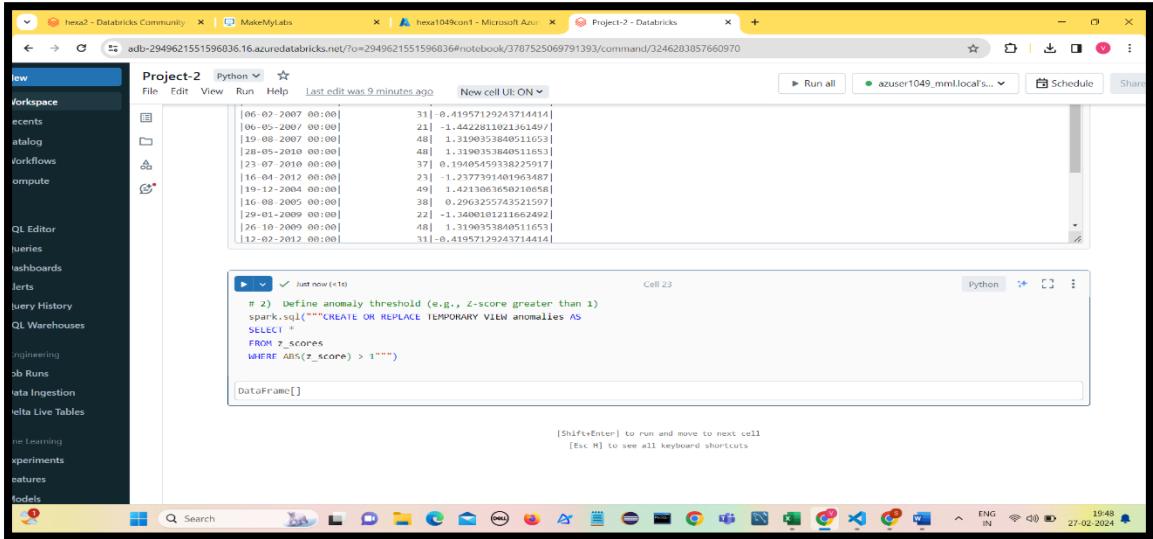
This screenshot shows the same Databricks notebook environment as the previous one. The sidebar and toolbar are identical. The notebook contains two cells:

- Cell 21:** Same as the previous screenshot, showing the creation of a temporary view 'z\_scores'.
- Cell 22:** Shows the output of the 'show()' command on the temporary view 'z\_scores'. The output is identical to the one shown in Cell 21.

## 3) Define Anomaly Threshold

- An anomaly threshold is a predefined value that determines when a data point is considered an anomaly.
- Common thresholds include Z-scores greater than a certain number of standard deviations from the mean.

- We define the threshold as  $\text{ABS}(z\_score) > 1$  to identify anomalies that are more than 1 standard deviations away from the mean.

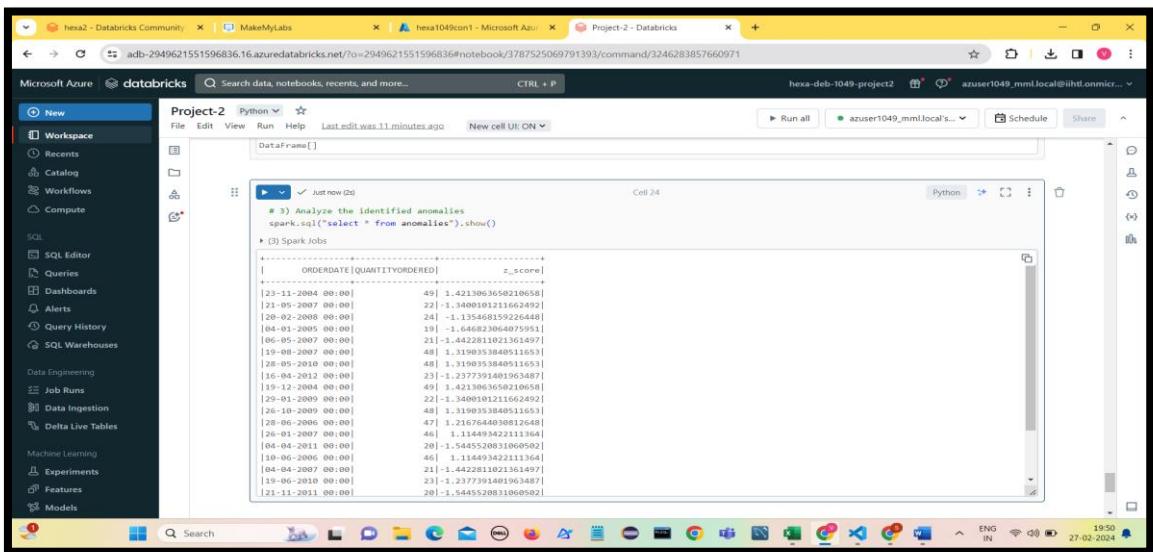


```
# 2) Define anomaly threshold (e.g., z-score greater than 1)
spark.sql("""CREATE OR REPLACE TEMPORARY VIEW anomalies AS
SELECT *
FROM z_scores
WHERE ABS(z_score) > 1""")
```

The screenshot shows a Databricks notebook interface. On the left is a sidebar with various workspace options like events, catalog, workflows, compute, etc. The main area has a code cell labeled 'Cell 23' containing the provided Python code. Below the code cell is a data frame representation: 'DataFrame[]'. At the bottom of the notebook window, there's a toolbar with icons for Run all, Schedule, Share, and other options.

## 4) Analyze Anomalies

Analyzing the identified anomalies involves examining their timestamps, associated values, and potential causes. It's essential to investigate anomalies further to determine whether they are genuine outliers or errors in the data.



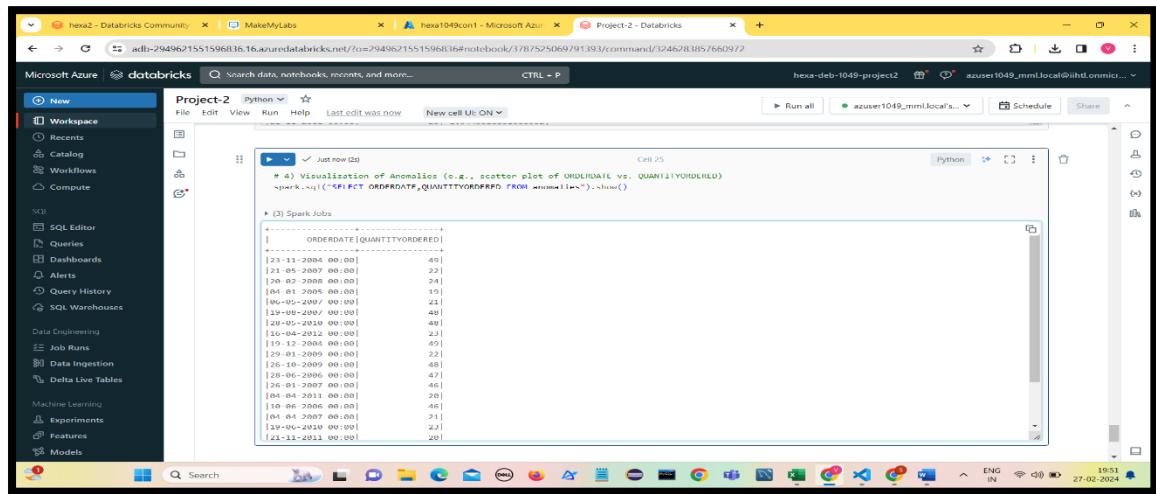
```
# 3) Analyze the identified anomalies
spark.sql("select * from anomalies").show()
```

The screenshot shows a Databricks notebook interface. The sidebar on the left includes 'Recent', 'Catalog', 'Workflows', 'Compute', 'SQL', 'SQL Editor', 'Queries', 'Alerts', 'Query History', and 'SQL Warehouses'. The main area shows the output of the previous code cell, which is a table with columns 'ORDERDATE' and 'QUANTITYORDERED' (with a placeholder 'z\_score'). The table contains numerous rows of data, each representing a specific order record. The bottom of the notebook window features a toolbar with various icons.

## 5) Visualize Anomalies

Visualizing anomalies can provide insights into their distribution and relationship with other variables in the dataset. This step can be helpful for gaining a deeper understanding of the anomalies.

## Visualize using Scatter plot:

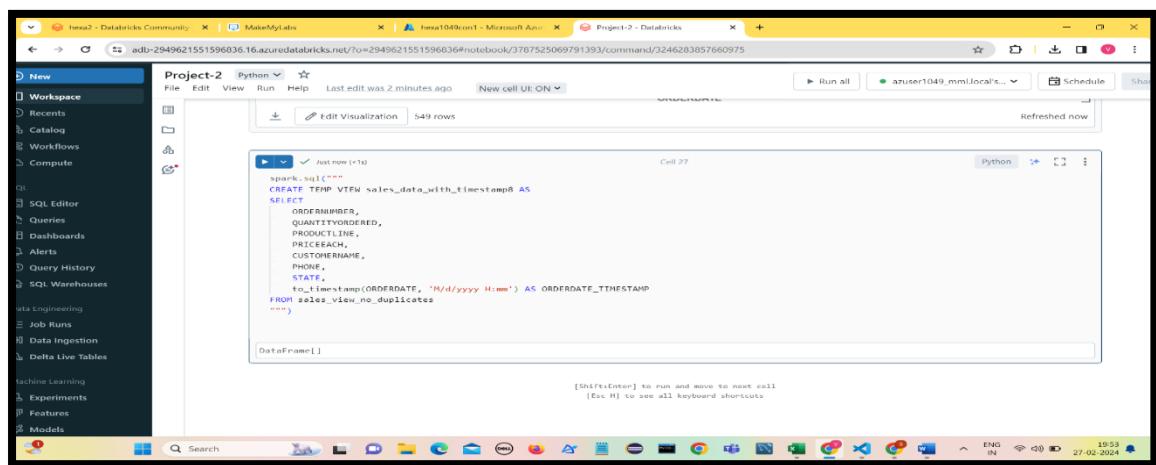
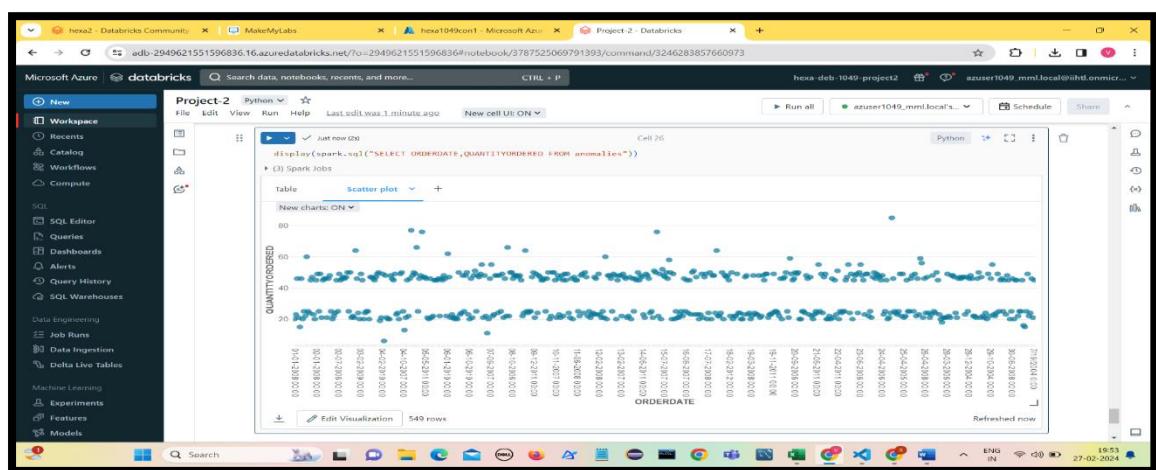


A screenshot of a Microsoft Edge browser window displaying a Databricks notebook titled "Project-2". The notebook contains a single Python cell (Cell 25) with the following code and output:

```
spark.sql("SELECT ORDERDATE, QUANTITYORDERED FROM anomalies").show()
```

The output shows a list of rows from the "anomalies" table, with columns "ORDERDATE" and "QUANTITYORDERED".

ORDERDATE	QUANTITYORDERED
23-11-2004 00:00:00	49
12-05-2005 00:00:00	22
12-06-2006 00:00:00	24
04-01-2005 00:00:00	19
16-05-2007 00:00:00	23
19-08-2008 00:00:00	40
12-02-2010 00:00:00	49
16-04-2012 00:00:00	23
19-12-2004 00:00:00	49
12-05-2006 00:00:00	22
12-10-2009 00:00:00	48
12-06-2006 00:00:00	47
26-01-2007 00:00:00	46
04-05-2008 00:00:00	29
10-06-2006 00:00:00	46
04-04-2007 00:00:00	21
19-04-2010 00:00:00	23
12-11-2011 00:00:00	20



A screenshot of a Microsoft Edge browser window displaying a Databricks notebook titled "Project-2". The notebook contains a single Python cell (Cell 27) with the following code:

```
spark.sql("""CREATE TEMP VIEW sales_data_with_timestamp AS
SELECT
    COUNTRYNAME,
    QUANTITYORDERED,
    PRODUCTLINE,
    PRICEEACH,
    CUSTOMERNAME,
    PHONE,
    STATE,
    to_timestamp(ORDERDATE, 'M/d/yyyy H:mm') AS ORDERDATE_TIMESTAMP
FROM
    sales_view_no_duplicates
""")
```

The output shows the result of the query, indicating 549 rows.

```

spark.sql("select * from sales_data_with_timestamp0").show()

```

ORDERNUMBER	QUANTITYORDERED	PRODUCTLINE	PRICEEACH	CUSTOMERNAME	PHONE	STATE	ORDERDATE_TIMESTAMP
10160	21	Classic Cars	100.0	Classic Legends Inc.	2125558903	NY 2003-11-29 00:00:00	
10161	20	Classic Cars	100.0	Cambridge Catalogue	6175556595	NY 2004-11-29 00:00:00	
10329	42	Motorcycles	100.0	Land of Toys Inc.	2125557818	NY 2004-11-15 00:00:00	
10245	34	Classic Cars	100.0	Super Scale Inc.	2035559545	CT 2003-11-25 00:00:00	
10223	37	Motorcycles	100.0	Australian Collector's...	03 10955 4555	Victoria 2004-02-20 00:00:00	
10215	35	Classic Cars	100.0	West Coast Collect...	3105553722	CA 2004-01-29 00:00:00	
10174	34	Classic Cars	100.0	Australian Gift N...	61 7-3844-6555	Queensland 2004-11-29 00:00:00	
10107	30	Motorcycles	95.7	Land of Toys Inc.	2125557818	NY 2003-02-27 00:00:00	
10285	36	Motorcycles	100.0	Marta's Replicas Co.	6175558951	MA 2004-01-27 00:00:00	
10131	20	Classic Cars	100.0	Classic Gift Ideas...	2125557818	PA 2003-11-27 00:00:00	
10201	22	Motorcycles	98.57	Mini Wheels Co.	6095555707	CA 2003-10-28 00:00:00	
10145	45	Motorcycles	83.26	ToysAGroundups.com	6265557265	CA 2003-08-28 00:00:00	
10145	45	Motorcycles	83.26	ToysAGroundups.com	6265557265	CA 2003-08-25 00:00:00	
10388	42	Motorcycles	76.36	FunGiftIdeas.com	5085552555	MA 2004-10-15 00:00:00	
10107	30	Motorcycles	95.7	Land of Toys Inc.	2125557818	NY 2004-07-23 00:00:00	
10140	37	Classic Cars	100.0	Technics Stores Inc.	6505556809	CA 2003-07-24 00:00:00	
10237	23	Motorcycles	100.0	Vitachrome Inc.	2125551500	NY 2004-10-15 00:00:00	
10318	46	Motorcycles	94.74	Diecast Classics ...	2155551555	PA 2004-10-15 00:00:00	

## Seasonality Detection:

Seasonality refers to the repetitive and predictable patterns or fluctuations in a time series dataset that occur at regular intervals over time. These patterns typically repeat over specific periods, such as days, weeks, months, or years, and are often influenced by external factors such as weather, holidays, or economic cycles.

### 1) Extract Timestamp Components:

Extract year, month, day, and value from the timestamp ORDERDATE.

```

# Seasonality Detection

# 1) Extract year, month, day, and value from the timestamp ORDERDATE
spark.sql("""CREATE OR REPLACE TEMPORARY VIEW timestamp_components AS
SELECT
    to_timestamp(ORDERDATE, 'n/d/yyyy H:m:s') AS ORDERDATE,
    YEAR(to_timestamp(ORDERDATE, 'n/d/yyyy H:m:s')) AS year,
    MONTH(to_timestamp(ORDERDATE, 'n/d/yyyy H:m:s')) AS month,
    DAY(to_timestamp(ORDERDATE, 'n/d/yyyy H:m:s')) AS day,
    QUANTITYORDERED
FROM sales_view_no_duplicates""")

```

A screenshot of a Microsoft Edge browser window displaying a Databricks notebook titled "Project-2". The notebook interface shows a sidebar with various Databricks services like Workspace, Catalog, Workflows, Compute, SQL, and Machine Learning. The main workspace contains two cells. Cell 30 displays the output of a SQL query: "spark.sql("select \* from timestamp\_components").show()". The output shows a DataFrame with columns ORDERDATE, year, month, day, and QUANTITYORDERED. The data consists of approximately 20 rows of timestamp values. Cell 31 contains the Python code for aggregating the data by year and month.

```

spark.sql("select * from timestamp_components").show()
# (2) Spark Jobs
+-----+-----+-----+-----+
| ORDERDATE | year | month | day | QUANTITYORDERED |
+-----+-----+-----+-----+
| 2003-10-20 00:00:00|2003| 10 | 20 | 21 |
| 2004-11-18 00:00:00|2004| 11 | 18 | 29 |
| 2004-11-15 00:00:00|2004| 11 | 15 | 42 |
| 2003-11-25 00:00:00|2003| 11 | 25 | 34 |
| 2004-02-20 00:00:00|2004| 2 | 20 | 37 |
| 2004-01-29 00:00:00|2004| 1 | 29 | 35 |
| 2004-11-18 00:00:00|2004| 11 | 18 | 34 |
| 2003-02-24 00:00:00|2003| 2 | 24 | 38 |
| 2004-08-27 00:00:00|2004| 8 | 27 | 36 |
| 2003-11-13 00:00:00|2003| 11 | 13 | 23 |
| 2003-10-28 00:00:00|2003| 10 | 28 | 22 |
| 2003-10-28 00:00:00|2003| 10 | 28 | 45 |
| 2003-08-25 00:00:00|2003| 8 | 25 | 45 |
| 2004-10-15 00:00:00|2004| 10 | 15 | 42 |
| 2004-07-23 00:00:00|2004| 7 | 23 | 38 |
| 2003-07-24 00:00:00|2003| 7 | 24 | 37 |
| 2004-10-15 00:00:00|2004| 10 | 15 | 23 |
| 2004-10-15 00:00:00|2004| 10 | 15 | 46 |

```

In this step, we extract relevant components such as year, month, day, and value from the timestamp column. These components will be used to analyze the seasonal patterns in the data.

## 2) Aggregate Data

### Aggregate data by year and month

A screenshot of a Microsoft Edge browser window displaying a Databricks notebook titled "Project-2". The sidebar shows various Databricks services. The main workspace contains two cells. Cell 30 shows the raw timestamp data. Cell 31 contains the Python code for aggregation:

```

# 2) Aggregate Data
# Aggregate data by year and month
# Seasonal Component
spark.sql("""
CREATE OR REPLACE TEMPORARY VIEW monthly_aggregation AS
SELECT
    year,
    month,
    AVG(QUANTITYORDERED) AS avg_value
FROM timestamp_components
GROUP BY year, month
ORDER BY year, month""")

```

The output of Cell 31 shows a DataFrame with columns year, month, and avg\_value, containing approximately 12 rows of aggregated data.

In this step, we aggregate the data by year and month, calculating the average value for each month. This aggregated data will help us identify seasonal patterns over time.

```

spark.sql("select * from monthly_aggregation").show()
# (3) Spark Jobs
+-----+-----+
|year/month| avg_value|
+-----+-----+
|NULL| NULL| 35.15242694226328|
|2003| 2| 30.0|
|2003| 7| 37.0|
|2003| 8| 45.0|
|2003| 10| 29.3333333333332|
|2003| 11| 32.7777777777778|
|2004| 1| 35.0|
|2004| 2| 37.0|
|2004| 6| 33.0|
|2004| 7| 25.5|
|2004| 8| 36.0|
|2004| 10| 39.75|
|2004| 11| 33.5|
|2004| 12| 20.0|
+-----+

```

### 3) Analyse Seasonality:

```

# 3) Analyze Seasonality
# Display the aggregated data
spark.sql("SELECT * FROM monthly_aggregation").show()
# (3) Spark Jobs
+-----+-----+
|year/month| avg_value|
+-----+-----+
|NULL| NULL| 35.15242694226328|
|2003| 2| 30.0|
|2003| 7| 37.0|
|2003| 8| 45.0|
|2003| 10| 29.3333333333332|
|2003| 11| 32.7777777777778|
|2004| 1| 35.0|
|2004| 2| 37.0|
|2004| 6| 33.0|
|2004| 7| 25.5|
|2004| 8| 36.0|
|2004| 10| 39.75|
|2004| 11| 33.5|
|2004| 12| 20.0|
+-----+

```

### 4) Visualize Seasonality

- In this step, we analyze the seasonality patterns by examining the aggregated data and visualizing it using line charts or other visualization tools.
- By observing the average value over time (e.g., months or years), we can identify any recurring patterns or trends indicating seasonality.

A screenshot of a Microsoft Edge browser window displaying a Databricks notebook titled "Project-2". The notebook interface includes a left sidebar with navigation links like "New", "Workspace", "Recents", "Catalog", "Workflows", "Compute", "SQL Editor", "Queries", "Dashboards", "Alerts", "Query History", "SQL Warehouses", "Data Engineering", "Job Runs", "Data Ingestion", "Delta Live Tables", "Machine Learning", "Experiments", "Features", and "Models". The main workspace shows a code cell (Cell 34) containing a SQL query:

```
# 4) Visualization of the Seasonality patterns
spark.sql("""
SELECT CONCAT(year, '-', LPAD(CAST(month AS STRING), 2, '0')) AS year_month,
       avg_value
FROM monthly_aggregation
ORDER BY year, month""").show()
```

The output of the query is a table named "year\_month" with "avg\_value" as the column:

year_month	avg_value
NULL	35.15242494226328
2003-02	30.0
2003-07	37.0
2003-08	45.0
2003-10	29.3333333333332
2003-11	32.7777777777778
2004-01	35.0
2004-02	37.0
2004-06	33.0
2004-07	25.5
2004-08	36.0
2004-10	39.75
2004-11	33.5
2004-12	28.0

A screenshot of a Microsoft Edge browser window displaying a Databricks notebook titled "Project-2". The notebook interface includes a left sidebar with navigation links like "New", "Workspace", "Recents", "Catalog", "Workflows", "Compute", "SQL Editor", "Queries", "Dashboards", "Alerts", "Query History", "SQL Warehouses", "Data Engineering", "Job Runs", "Data Ingestion", "Delta Live Tables", "Machine Learning", "Experiments", "Features", and "Models". The main workspace shows a code cell (Cell 35) containing a SQL query:

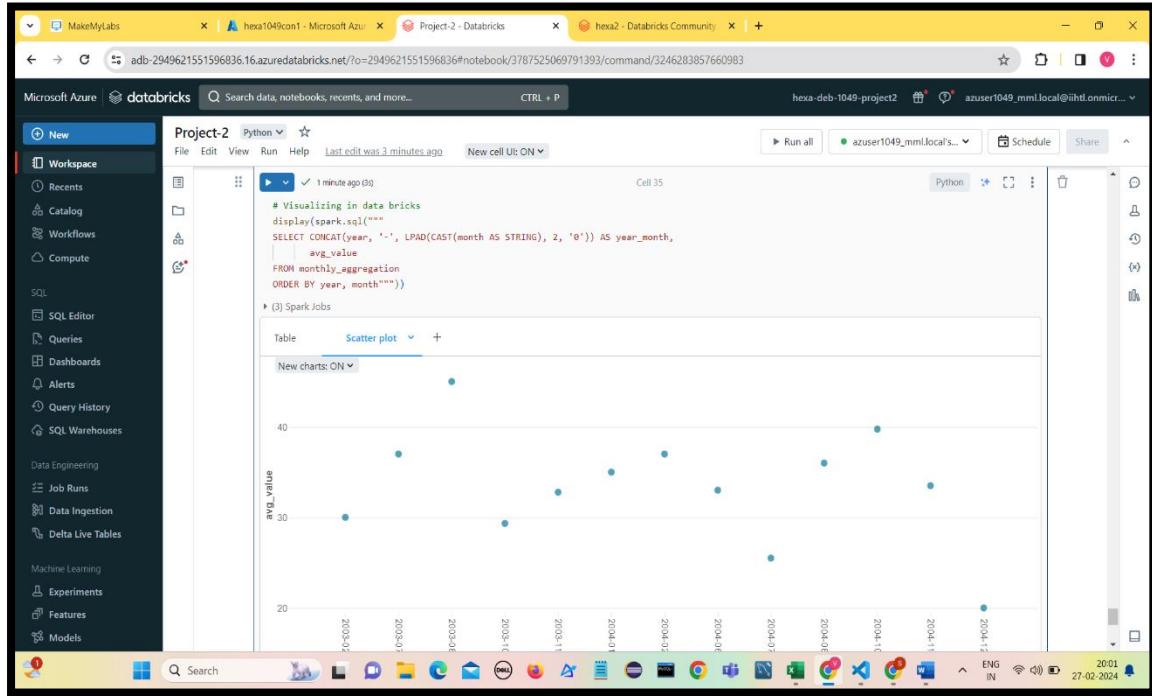
```
# Visualizing in data bricks
display(spark.sql("""
SELECT CONCAT(year, '-', LPAD(CAST(month AS STRING), 2, '0')) AS year_month,
       avg_value
FROM monthly_aggregation
ORDER BY year, month"""))
```

The output of the query is a table named "year\_month" with "avg\_value" as the column, displayed in a tabular format:

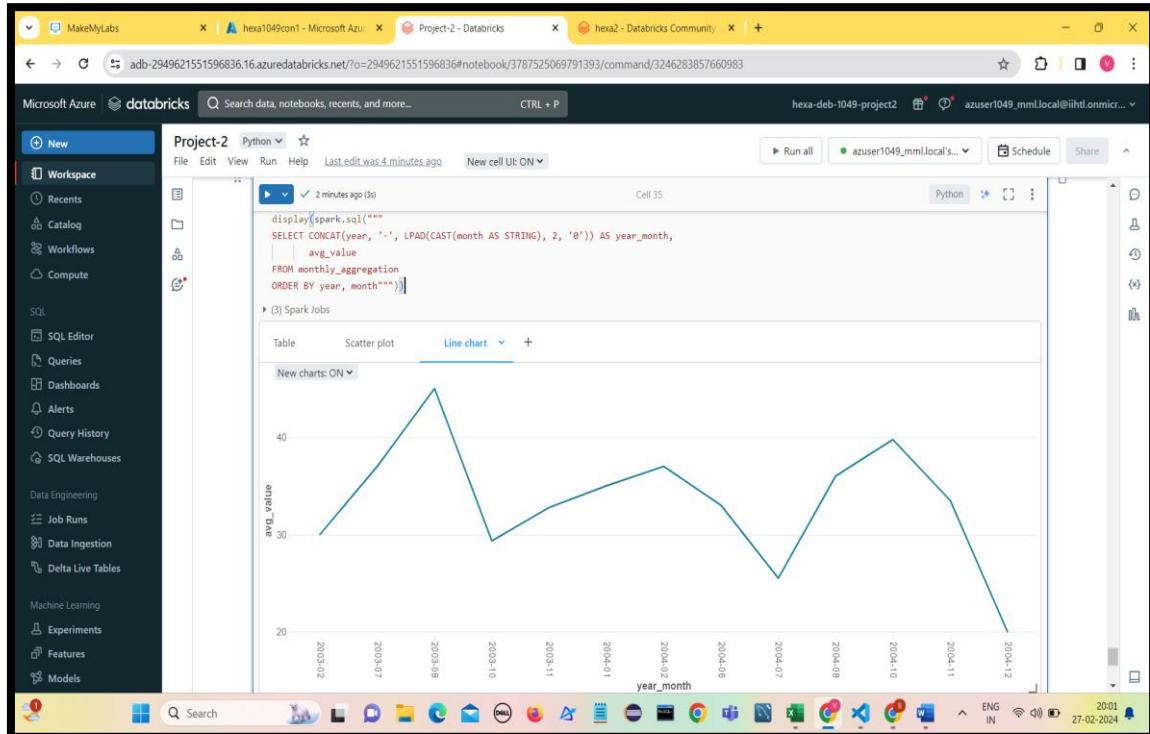
year_month	avg_value
1 null	35.15242494226328
2 2003-02	30
3 2003-07	37
4 2003-08	45
5 2003-10	29.3333333333332
6 2003-11	32.7777777777778
7 2004-01	35

Below the table, it says "14 rows | 2.75 seconds runtime".

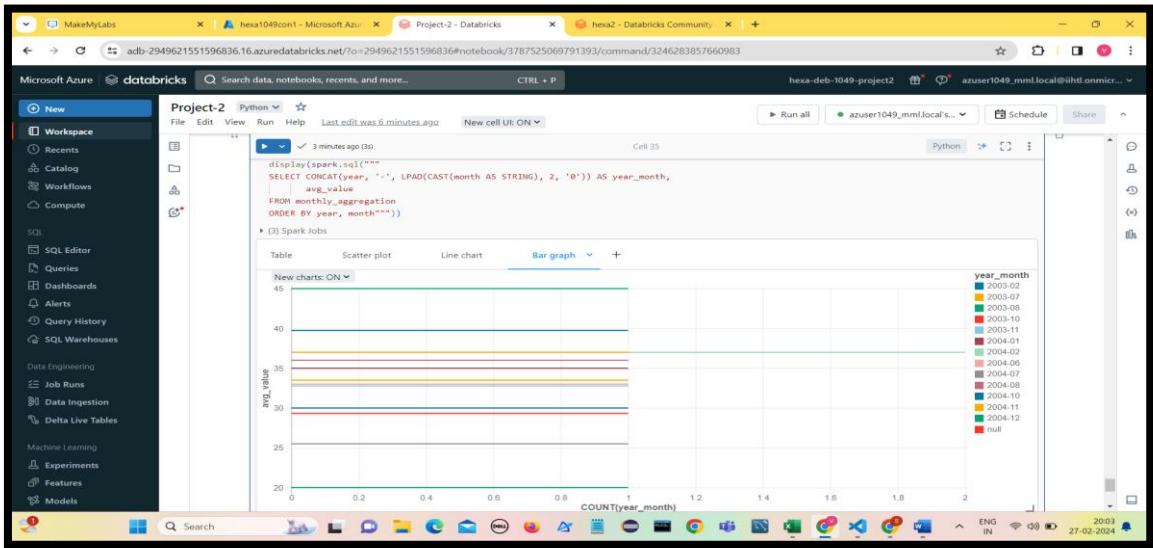
## 1) Scatter plot



## 2) Line Chart



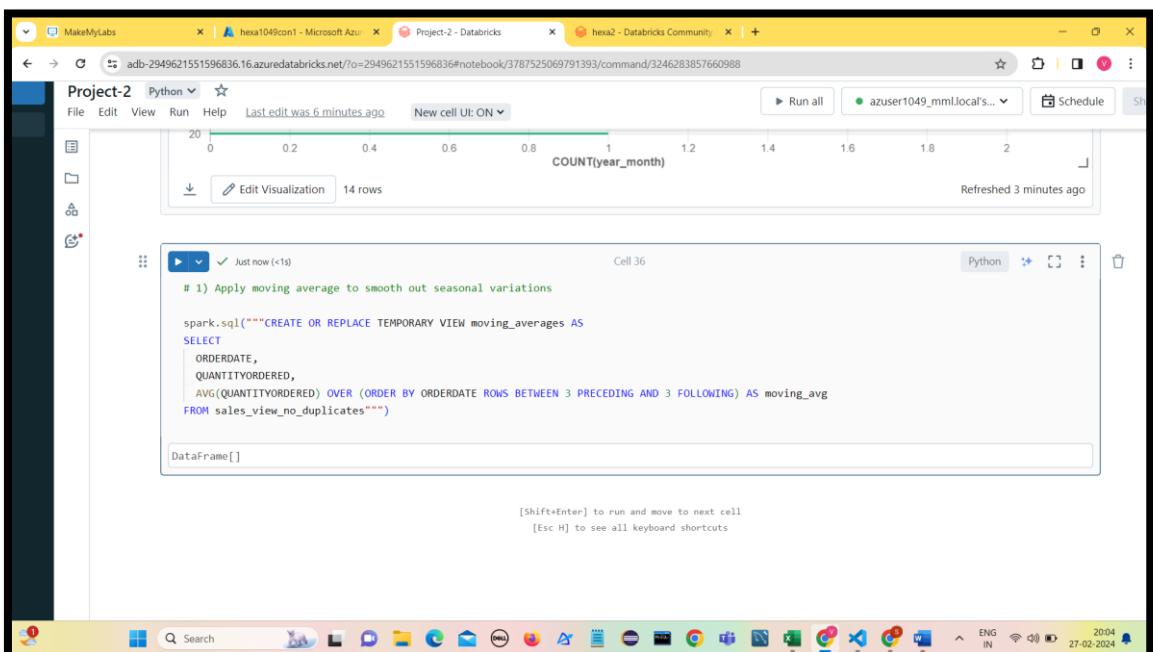
### 3) Bar chart



## Moving Averages

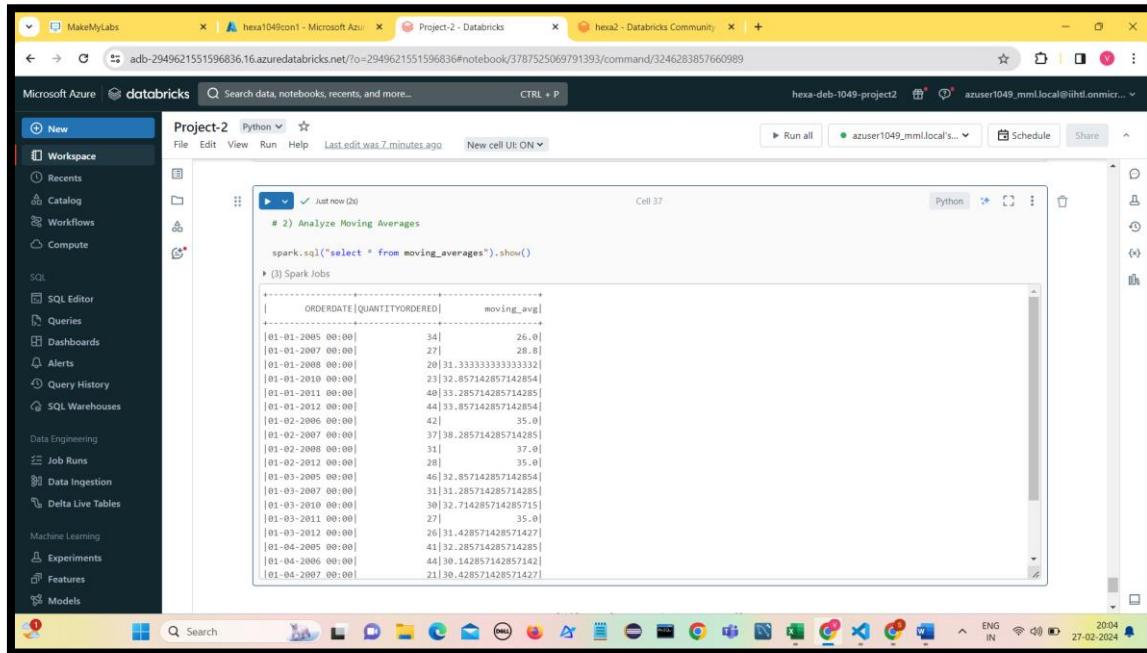
Moving averages are commonly used to smooth out fluctuations in time-series data and identify trends or patterns over a specific period. In the context of detecting seasonality using Spark SQL on Azure Databricks, we can apply moving averages to analyze the trend and smooth out the seasonal variations in the data.

### 1) Calculate moving averages



## 2) Analyze Moving Averages

Once you have calculated the moving averages, you can analyze them to identify trends and smooth out seasonal variations in the data.



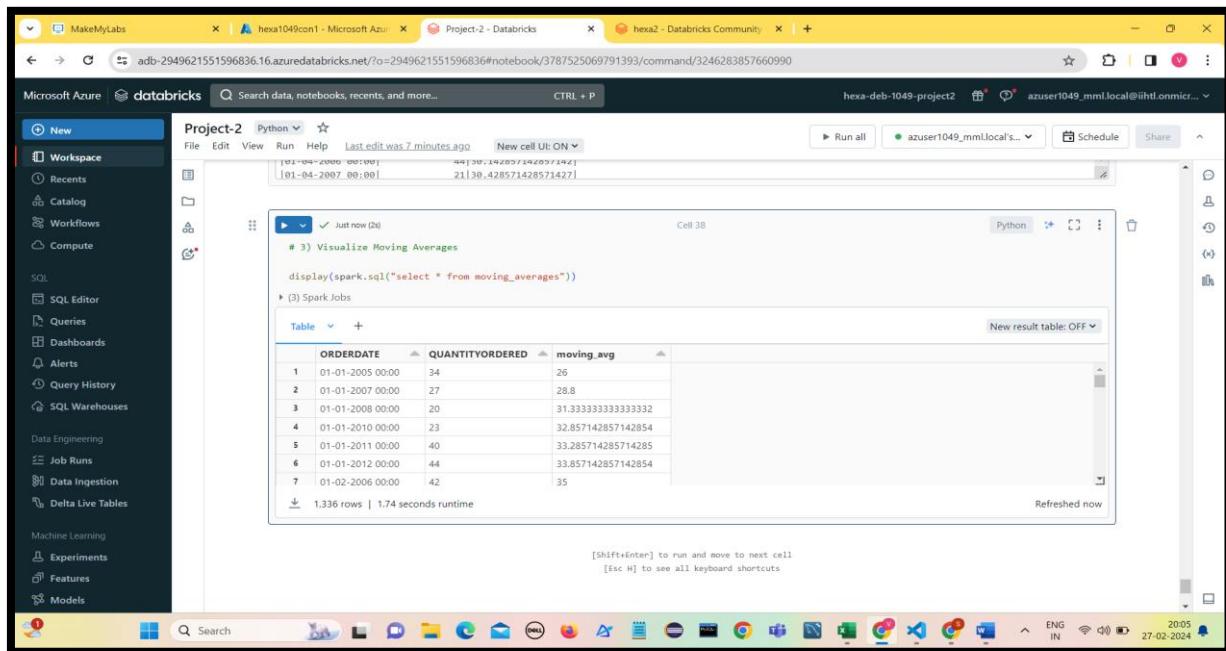
```
Project-2 Python Just now (2s)
# 2) Analyze Moving Averages

spark.sql("select * from moving_averages").show()

+-----+-----+
| ORDERDATE | QUANTITYORDERED | moving_avg |
+-----+-----+
| 01-01-2005 00:00 | 34 | 26.0 |
| 01-01-2007 00:00 | 27 | 28.8 |
| 01-01-2008 00:00 | 20 | 31.33333333333332 |
| 01-01-2010 00:00 | 23 | 32.857142857142854 |
| 01-01-2011 00:00 | 40 | 33.285714285714285 |
| 01-01-2012 00:00 | 44 | 33.857142857142854 |
| 01-02-2006 00:00 | 42 | 35.0 |
| 01-02-2007 00:00 | 37 | 38.285714285714285 |
| 01-02-2008 00:00 | 31 | 37.0 |
| 01-02-2012 00:00 | 28 | 35.0 |
| 01-03-2005 00:00 | 46 | 32.857142857142854 |
| 01-03-2007 00:00 | 31 | 31.285714285714285 |
| 01-03-2010 00:00 | 30 | 32.714285714285715 |
| 01-03-2011 00:00 | 27 | 35.0 |
| 01-03-2012 00:00 | 26 | 31.428571428571427 |
| 01-04-2005 00:00 | 41 | 32.285714285714285 |
| 01-04-2006 00:00 | 44 | 30.42857142857142 |
| 01-04-2007 00:00 | 21 | 30.428571428571427 |
```

## 3) Visualize moving averages

Visualization can provide a clearer understanding of the trend and seasonal variations in the data after applying moving averages.

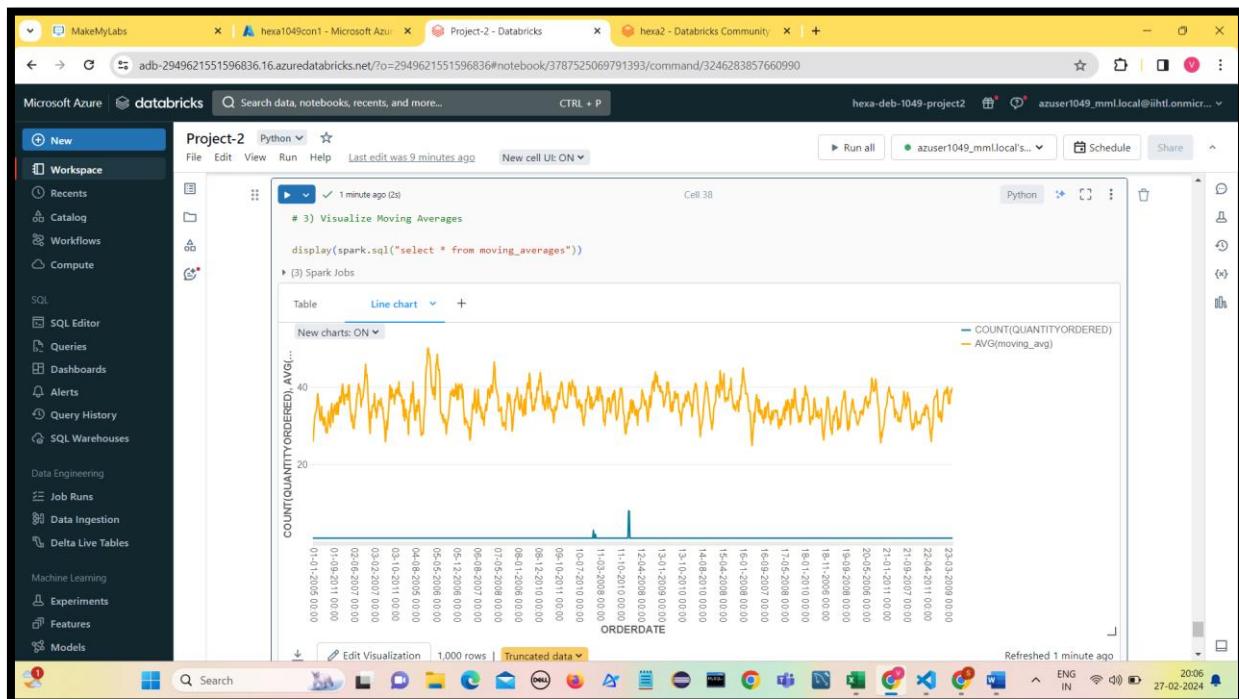


```
Project-2 Python Just now (2s)
# 3) Visualize Moving Averages

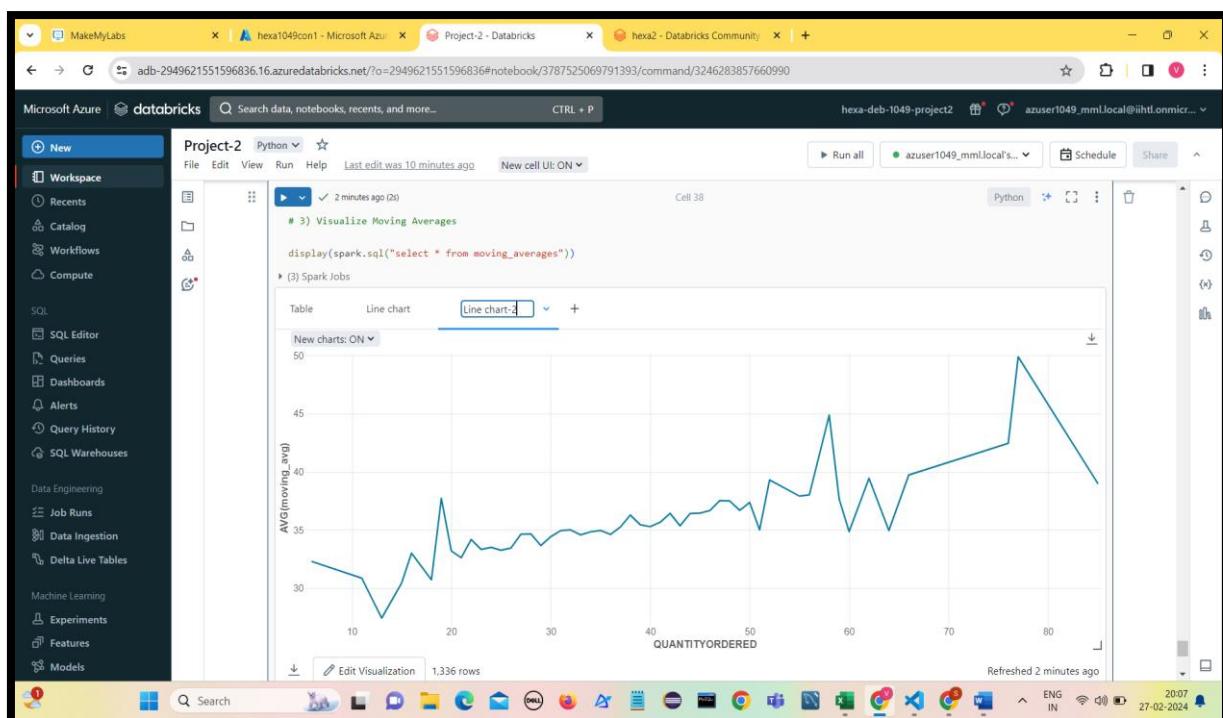
display(spark.sql("select * from moving_averages"))

+-----+-----+
| ORDERDATE | QUANTITYORDERED | moving_avg |
+-----+-----+
| 1 01-01-2005 00:00 | 34 | 26 |
| 2 01-01-2007 00:00 | 27 | 28.8 |
| 3 01-01-2008 00:00 | 20 | 31.33333333333332 |
| 4 01-01-2010 00:00 | 23 | 32.857142857142854 |
| 5 01-01-2011 00:00 | 40 | 33.285714285714285 |
| 6 01-01-2012 00:00 | 44 | 33.857142857142854 |
| 7 01-02-2006 00:00 | 42 | 35 |
+-----+-----+
```

## 1) Line Chart



## 2) Line chart-2



## Create a destination container to write the transformed data.

The screenshot shows the Azure Storage Account 'hexaadls1049' in the 'Containers' blade. A new container named 'destcon123' is being created in the 'New container' dialog. The dialog shows the container name, an anonymous access level set to 'Private (no anonymous access)', and a note that the access level is set to private because anonymous access is disabled on this storage account. The 'Create' button is visible at the bottom.

## Write the transformed data to destcon123 in hexaadls1049 Azure Data Lake Storage

The screenshot shows a Databricks workspace with a Python notebook titled 'Project-2'. The notebook contains a cell with the following code:

```
# Writing the transformed data to a destcon123 in hexaadls1049 Azure Data Lake Storage account.

new_df.write.option("header", "true").csv(path='abfss://destcon123@hexaadls1049.dfs.core.windows.net/output.csv')
```

Below the code cell, there is a table preview showing 1,336 rows of data. The table has columns: ORDERNUMBER, QUANTITYORDERED, PRODUCTLINE, PRICEEACH, CUSTOMERNAME, PHONE, and STATE. The data includes rows for Motorcycles, Vintage Cars, Classic Cars, and Planes from various locations like Australia, Japan, and the USA.

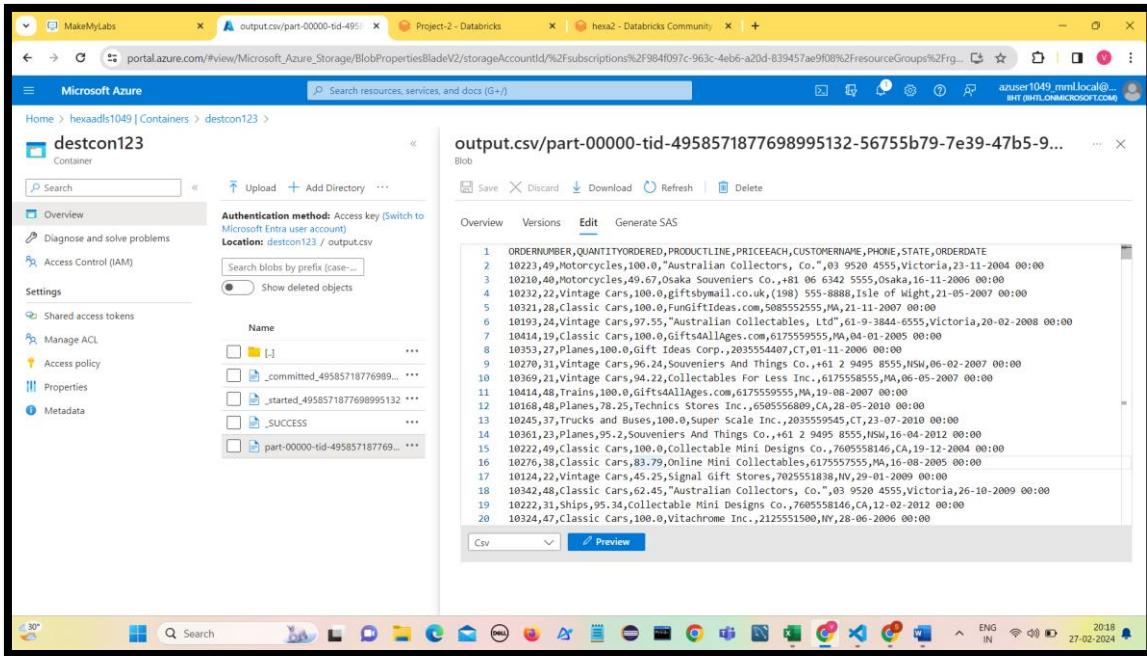
## Writing into output.csv file

The screenshot shows the Microsoft Azure Storage Container Overview page for the 'destcon123' container. The container has one blob named 'output.csv'. The blob details are as follows:

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
output.csv						-

The screenshot shows the Microsoft Azure Storage Container Overview page for the 'destcon123' container. The container now contains several blobs, including log files and a success file. The blob details are as follows:

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
[.]						***
_committed_4958571877698995132	2/27/2024, 8:15:46 PM	Hot (Inferred)		Block blob	112 B	Available
_started_4958571877698995132	2/27/2024, 8:15:44 PM	Hot (Inferred)		Block blob	0 B	Available
_SUCCESS	2/27/2024, 8:15:47 PM	Hot (Inferred)		Block blob	0 B	Available
part-00000-tid-4958571877698995132-56755b79-...	2/27/2024, 8:15:45 PM	Hot (Inferred)		Block blob	109.27 KB	Available



## Conclusion:

In conclusion, the project focused on leveraging Spark SQL on Azure Databricks for data analysis. The project demonstrated the effectiveness of using Spark SQL on Azure Databricks for data analysis tasks, particularly in detecting seasonality and identifying anomalies in time series data. The insights gained from this project can inform strategic decision-making and operational planning for various domains, including retail, finance, healthcare, and more.

**Vuthur Sriganga**

**Hexaware Data Engineering Batch-1**

**vuthursriganga@gmail.com**