

# **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:**

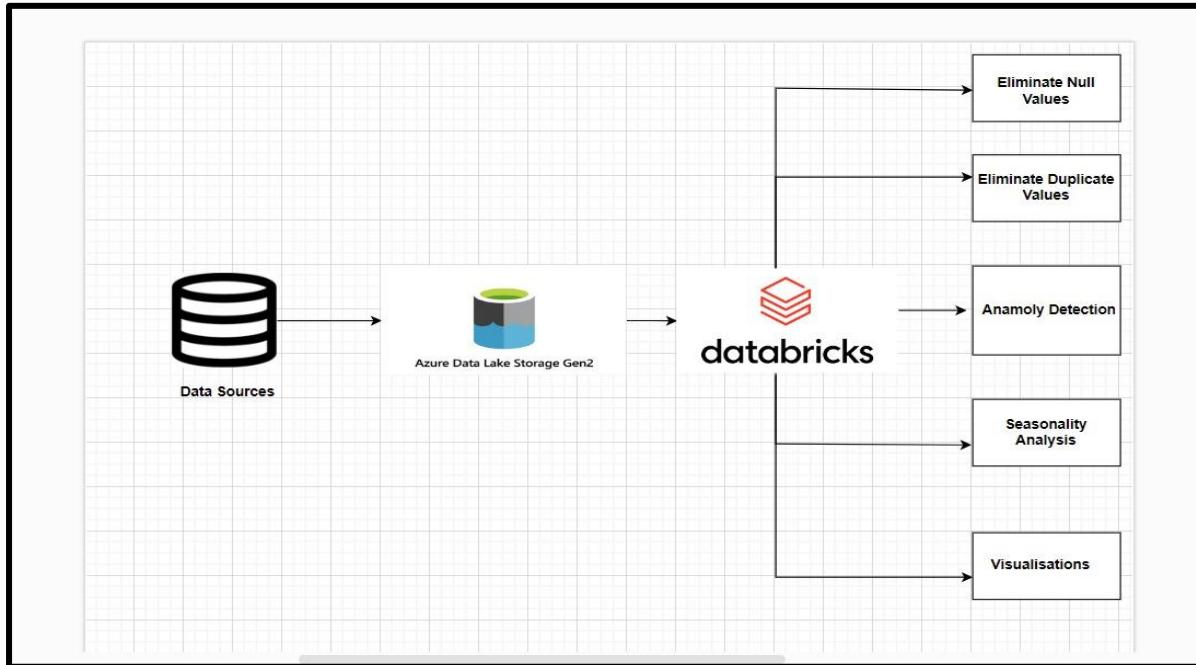
The project involves implementing data analysis using Spark SQL on Azure Databricks and processing the data for errors, seasonality, and anomalies. This initiative aims to leverage the scalability and performance of Spark on the Azure Databricks platform for efficient data processing and analysis.

## **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 behaviour.

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

- Analyse 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.
- Analyse 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.

## **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:**

- Analyse 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:**

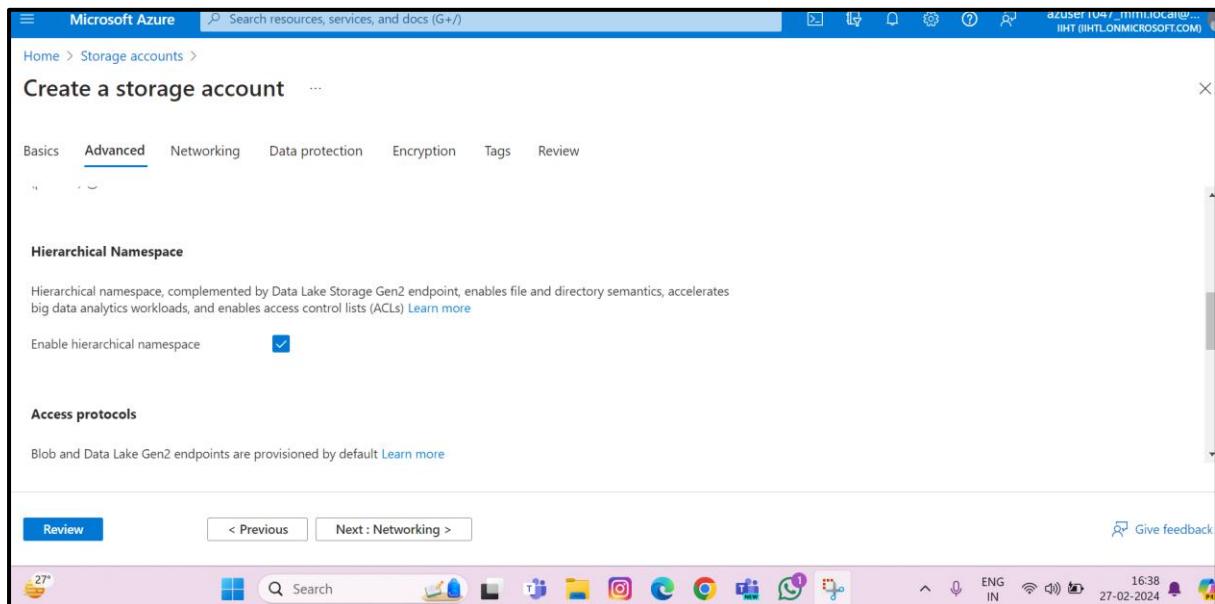
- Visualize analysed data and insights using interactive visualizations or reports.
- Communicate findings to stakeholders for data-driven decision-making.

## **Tasks Performed**

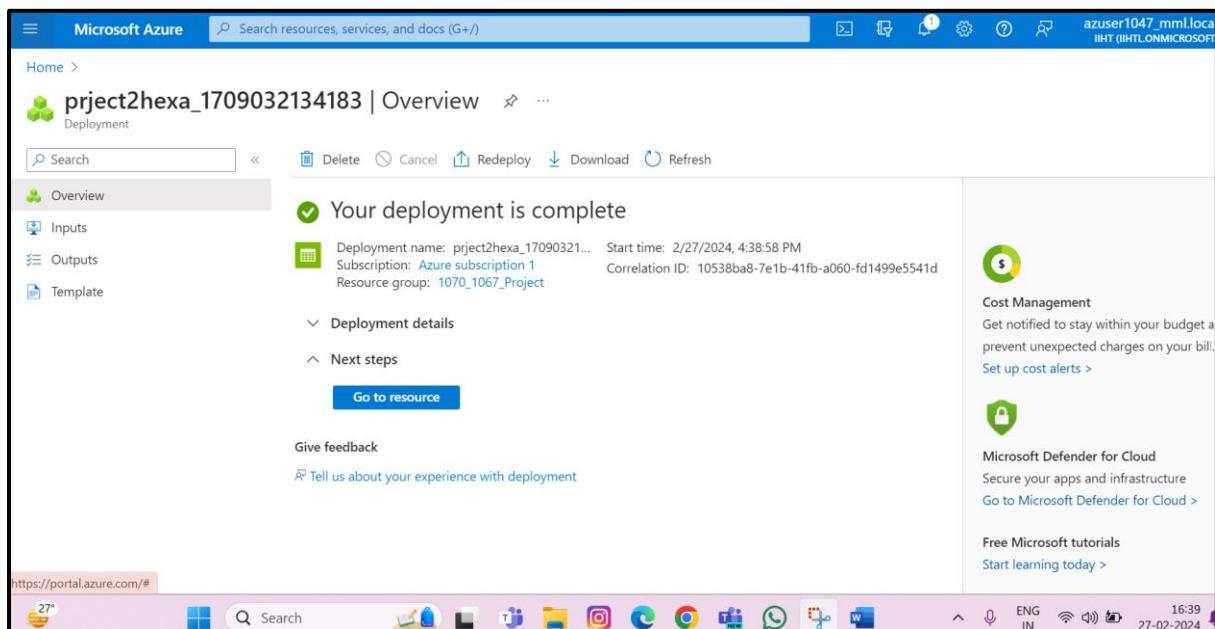
- 1) Provision Azure Databricks workspace in Azure portal.
- 2) Ingest raw data from sources into Azure Databricks.
- 3) Explore data using Databricks notebooks for structure and quality.
- 4) Clean data by handling missing values, duplicates, and inconsistencies.
- 5) Transform cleaned data into suitable 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

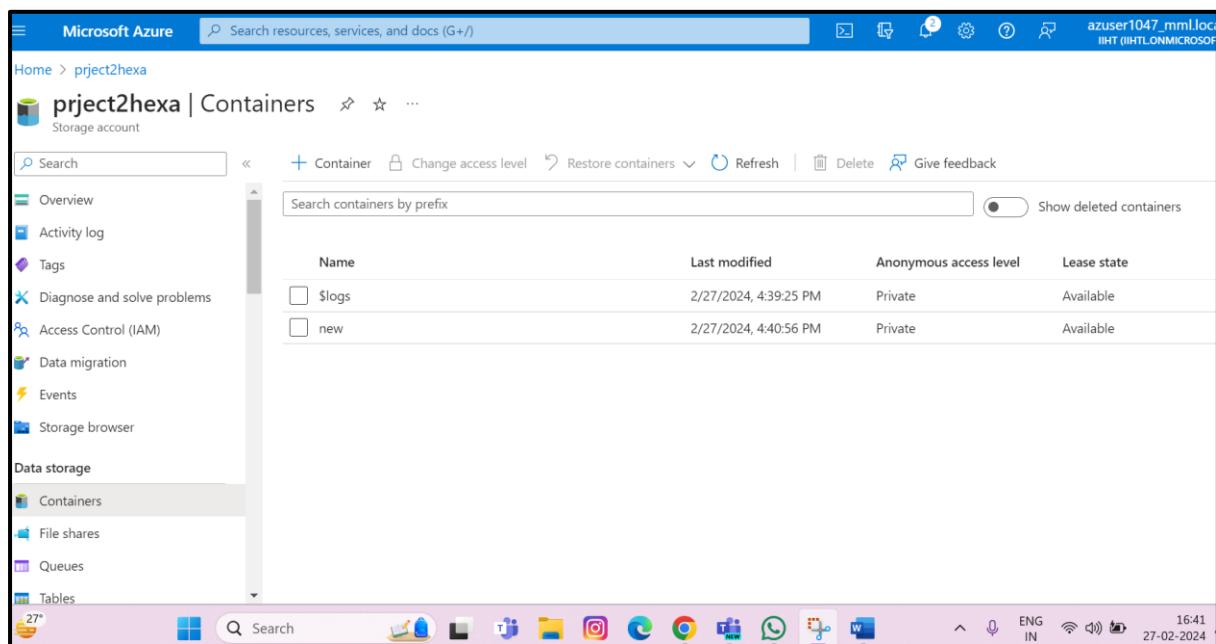
### Creating an Azure Data Lake Storage GEN2 Account by enabling Hierarchical Namespace



Account Created successfully:

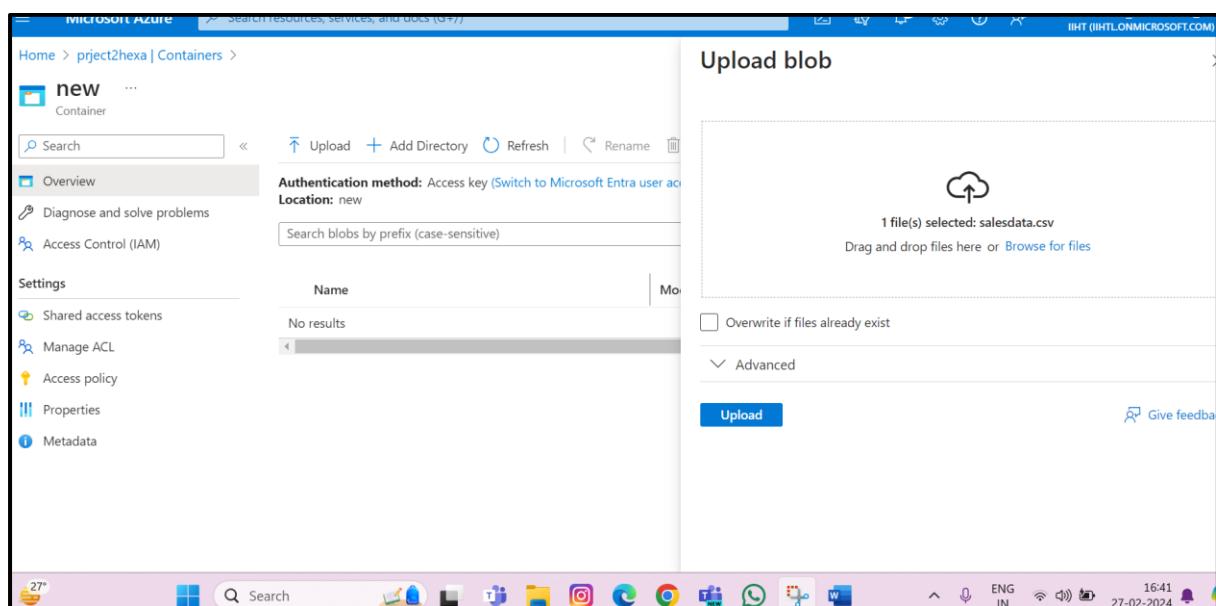


**Inside the ADLS account create a container to upload file:**



A screenshot of the Microsoft Azure Storage Account interface. The left sidebar shows 'Containers' selected under 'Data storage'. The main area displays a list of containers with columns for Name, Last modified, Anonymous access level, and Lease state. Two containers are listed: '\$logs' (Last modified 2/27/2024, 4:39:25 PM, Private, Available) and 'new' (Last modified 2/27/2024, 4:40:56 PM, Private, Available). The top navigation bar includes a search bar, a 'Container' button, and other account-related links.

**Uploading file in Container:**



A screenshot of the Microsoft Azure Storage Account interface showing the 'Upload blob' dialog. The dialog is overlaid on the 'Containers' list. It shows a file named 'salesdata.csv' has been selected for upload. The 'Upload' button is visible at the bottom right of the dialog. The background shows the same container list as the previous screenshot.

## Creating Azure Databricks Workspace

The screenshot shows the Microsoft Azure Deployment Overview page for a deployment named "rg-azuser1047\_mml.local-1vvZg\_project2". The status is "Your deployment is complete". Deployment details include a name of "rg-azuser1047\_mml.local-1vv... ", a start time of "2/27/2024, 4:42:58 PM", a subscription of "Azure subscription 1", a correlation ID of "e0ee7a84-a7bf-4c92-9d08-28...", and a resource group of "rg-azuser1047\_mml.local-1vv... ". There are sections for "Deployment details" and "Next steps", with a "Go to resource" button. On the right, there are links for "Cost management", "Microsoft Defender for Cloud", and "Free Microsoft tutorials". The bottom of the screen shows the Windows taskbar with various pinned icons.

## Creating a cluster after launching Databricks notebook:

The screenshot shows the Microsoft Azure Databricks Compute New Compute page. The sidebar on the left lists "Compute", "SQL", "Data Engineering", "Job Runs", "Data Ingestion", and "Delta Live Tables". The main area shows settings for a new compute cluster: "Runtime: 14.3 LTS ML (Scala 2.12, Spark 3.5.0)", "Node type: Standard\_DS3\_v2 (14 GB Memory, 4 Cores)", and "Terminate after 4320 minutes of inactivity". There are sections for "Tags" (Add tags, Value) and "Advanced options". At the bottom are "Create compute" and "Cancel" buttons. The bottom of the screen shows the Windows taskbar.

## Cluster created successfully:

The screenshot shows the Databricks Compute interface for creating a cluster. The top navigation bar includes 'Compute > Preview' and 'Send feedback'. The main title is 'azuser1047\_mml.local's Cluster' with a green checkmark icon. A 'More' button and a 'Terminate' button are visible. Below the title, there are tabs for 'Configuration', 'Notebooks (0)', 'Libraries', 'Event log', 'Spark UI', 'Driver logs', 'Metrics', 'Apps', and 'Spark compute UI - Master'. The 'Configuration' tab is selected. Under 'Databricks Runtime Version', it says '14.3 LTS ML (includes Apache Spark 3.5.0, Scala 2.12)'. There is an unchecked checkbox for 'Use Photon Acceleration'. Under 'Node type', it shows 'Standard\_DS3\_v2' with '14 GB Memory, 4 Cores'. A checked checkbox indicates 'Terminate after 4320 minutes of inactivity'. The 'Tags' section shows 'No custom tags' and a link to 'Automatically added tags'. A 'Advanced options' link is also present. The bottom of the window shows a taskbar with various icons and system status information like 'ENG IN', '16:52', and '27-02-2024'.

## Creating a Databricks workspace:

The screenshot shows the Databricks workspace interface. The left sidebar has a 'New' button highlighted, followed by 'Workspace', 'Recents', 'Catalog', 'Workflows', 'Compute', 'SQL', 'SQL Editor', 'Queries', 'Dashboards', 'Alerts', 'Query History', and 'SQL Warehouses'. The main area is titled 'Project2' with 'Python' selected. It shows a single notebook cell labeled 'Cell 1' with the instruction 'Start typing or generate with AI (Ctrl + I)...'. Below the cell, keyboard shortcuts are listed: '[Shift+Enter] to run and move to next cell' and '[Esc H] to see all keyboard shortcuts'. The top navigation bar includes 'File', 'Edit', 'View', 'Run', 'Help', 'Last edit was now', 'New cell UI: ON', 'Run all', 'Schedule', and 'Share'. The bottom taskbar shows a search bar, file icons, and system status information like 'ENG IN', '16:54', and '27-02-2024'.

## Connecting Databricks to source file through SAS token:

The screenshot shows a Databricks notebook interface. The top navigation bar includes 'File', 'Edit', 'View', 'Run', 'Help', 'Last edit was 3 minutes ago', 'New cell UI: ON', 'Run all', 'azuser1047\_mml.local's...', 'Schedule', and 'Share'. The sidebar on the left has icons for 'Notebook', 'File', 'Delta', and 'Edit'. The main area contains three cells:

- Cell 1:** Contains the Python code: 

```
spark.conf.set("fs.azure.sas.new.project2hexa.blob.core.windows.net", "sv=2022-11-02&ss=bfqt&srt=sco&sp=rw&lac=upx&se=2024-02-27T19:25:15Z&st=2024-02-27T11:25:15Z&spr=https&sig=ta6T1BqdATJrmj5TiVo2ErnG57dcf1WivTcruzYXULc%3D")
```
- Cell 2:** An empty cell.
- Cell 3:** Contains the Python code: 

```
from pyspark.sql import SparkSession
```

 followed by a comment: 

```
# Create SparkSession
```

. Below this, the code reads a CSV file: 

```
spark = SparkSession.builder \
    .appName("Data Analysis") \
    .getOrCreate()
```

, 

```
df = spark.read.csv('wasbs://new@project2hexa.blob.core.windows.net/salesdata.csv', header=True, inferSchema=True)
```

, and displays the DataFrame: 

```
display(df)
```

.

The status bar at the bottom shows the URL 'redatabricks.net/?o=7244331283462641', the date '27-02-2024', the time '17:05', and system status indicators.

## Data Exploration

### Displaying Schema:

The screenshot shows a Databricks notebook interface with the same top navigation bar and sidebar as the previous screenshot. The main area shows the continuation of Cell 3 from the previous screenshot:

```
from pyspark.sql import SparkSession
```

```
# Create SparkSession
```

```
spark = SparkSession.builder \
    .appName("Data Analysis") \
    .getOrCreate()
```

```
df = spark.read.csv('wasbs://new@project2hexa.blob.core.windows.net/salesdata.csv', header=True, inferSchema=True)
```

```
display(df)
```

Below the code, it shows the output of the `display(df)` command:

df: pyspark.sql.dataframe.DataFrame = [ORDERNUMBER: integer, QUANTITYORDERED: integer ... 7 more fields]

A table view of the DataFrame is displayed:

	ORDERNUMBER	QUANTITYORDERED	PRODUCTLINE	PRICEEACH	CUSTOMERNAME	PHONE
1	10107	30	Motorcycles	95.7	Land of Toys Inc.	2125557
2	10121	34	Motorcycles	81.35	Reims Collectables	26.47.15
3	10134	41	Motorcycles	94.74	Lyon Souveniers	+33 1 4€
4	10145	45	Motorcycles	83.26	Toys4GrownUps.com	6265557
5	10145	45	Motorcycles	83.26	Toys4GrownUps.com	6265557
6	10107	30	Motorcycles	95.7	Land of Toys Inc.	2125557

The status bar at the bottom shows the URL 'redatabricks.net/?o=7244331283462641', the date '27-02-2024', the time '17:09', and system status indicators.

## Creating a temp view and selecting all the rows:

The screenshot shows the Microsoft Azure Databricks workspace. On the left, the sidebar includes options like Workspace, Recents, Catalog, Workflows, Compute, SQL, and Data Engineering. The main area displays a Python notebook cell titled 'Cell 3' with the following code:

```
df.createOrReplaceTempView("sales")
spark.sql("select * from sales").display()
```

The resulting table contains 7 rows of sales data:

ORDERNUMBER	QUANTITYORDERED	PRODUCTLINE	PRICEEACH	CUSTOMERNAME	PHONE
1	10107	Motorcycles	95.7	Land of Toys Inc.	2125557
2	10121	Motorcycles	81.35	Reims Collectables	2647.15
3	10134	Motorcycles	94.74	Lyon Souveniers	+33 1 46
4	10145	Motorcycles	83.26	Toys4GrownUps.com	6265557
5	10145	Motorcycles	83.26	Toys4GrownUps.com	6265557
6	10107	Motorcycles	95.7	Land of Toys Inc.	2125557
7	10180	Motorcycles	86.13	Daedalus Desins Imports	20.16.13

At the bottom of the table, it says '2,823 rows | 0.38 seconds runtime'. The status bar at the bottom right shows 'Refreshed now'.

## Data Cleaning

### Removing null values:

Code:

The screenshot shows the Microsoft Azure Databricks workspace. The sidebar is identical to the previous screenshot. The main area displays a Python notebook cell titled 'Cell 4' with the following code:

```
# 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
    ORDERNUMBER IS NOT NULL AND
    QUANTITYORDERED IS NOT NULL AND
    PRODUCTLINE IS NOT NULL AND
    PRICEEACH IS NOT NULL AND
    CUSTOMERNAME IS NOT NULL AND
    PHONE IS NOT NULL AND
    ORDERDATE IS NOT NULL""")
spark.sql("SELECT * FROM sales_null").display()
```

The resulting table is identical to the one in the previous screenshot, containing 7 rows of sales data.

## Output:

The screenshot shows a Databricks notebook titled "Project2" in Python. Cell 4 contains a table named "remove\_null\_df" with the following data:

	ORDERNUMBER	QUANTITYORDERED	PRODUCTLINE	PRICEEACH	CUSTOMERNAME	PHONE
1	10107	30	Motorcycles	95.7	Land of Toys Inc.	21255578
2	10145	45	Motorcycles	83.26	Toys4GrownUps.com	62655572
3	10145	45	Motorcycles	83.26	Toys4GrownUps.com	62655572
4	10107	30	Motorcycles	95.7	Land of Toys Inc.	21255578
5	10201	22	Motorcycles	98.57	Mini Wheels Co.	65055557
6	10201	22	Motorcycles	98.57	Mini Wheels Co.	65055557
7	10223	37	Motorcycles	100	Australian Collectors. Co.	03 9520

1,338 rows | 0.60 seconds runtime

[Shift+Enter] to run and move to next cell  
[Esc H] to see all keyboard shortcuts

17:23 27-02-2024

## Count of null Values:

The screenshot shows a Databricks notebook titled "Project2" in Python. Cell 5 contains the following SQL code:

```
# Count of null values
state_null_count_df = spark.sql("""SELECT COUNT(*) AS state_null_count
| FROM sales_null
| WHERE state IS NULL""")
state_null_count_df.show()
```

The output shows a single row with a value of 0.

[Shift+Enter] to run and move to next cell  
[Esc H] to see all keyboard shortcuts

17:25 27-02-2024

## Count of null values in all columns

The screenshot shows a Databricks notebook titled "Project2" in Python. The code in Cell 6 is:

```
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 PRICEEACH IS NULL THEN 1 END) AS PRICEEACH_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"""
)
null_values_counts_df.display()
```

The result of the query is displayed in a table:

	ORDERNUMBER_null_count	QUANTITYORDERED_null_count	PRODUCTLINE_null_count	PRICEEACH_null_count
1	0	0	0	0

## Distinct Values :

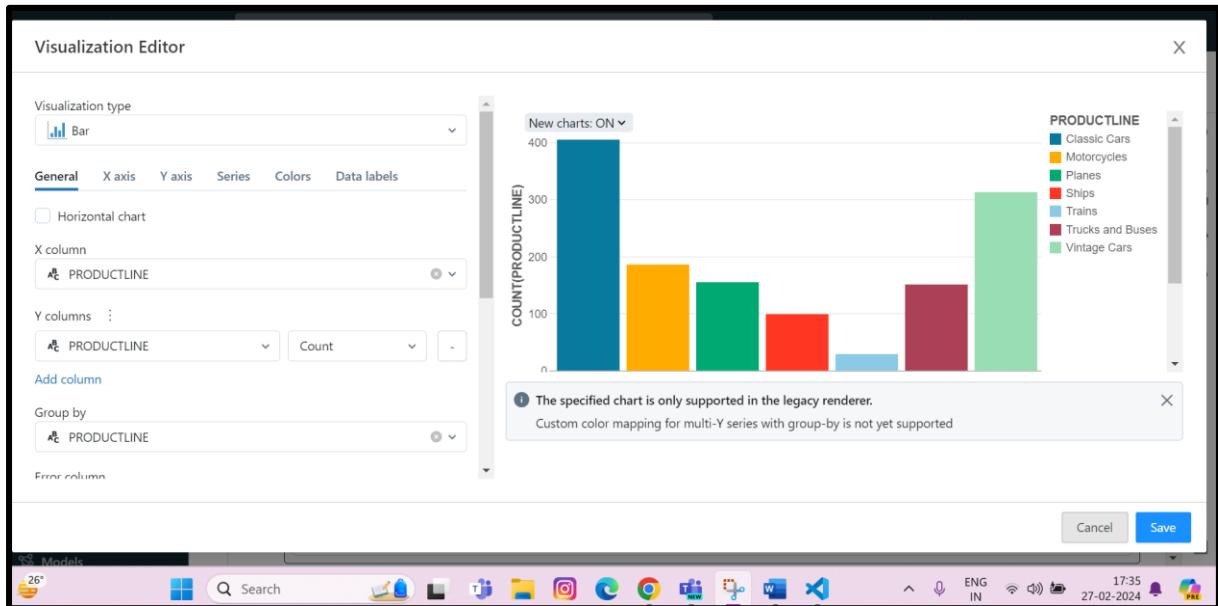
The screenshot shows a Databricks notebook titled "Project2" in Python. The code in Cell 7 is:

```
# 2) Selecting distinct values
spark.sql("select distinct PRODUCTLINE from sales_null").display()
```

The result of the query is displayed in a table:

PRODUCTLINE
1 Motorcycles
2 Vintage Cars
3 Ships
4 Trucks and Buses
5 Classic Cars
6 Trains
7 Planes

Visualisation that shows count of each product grouped by the product names:



## Error Detection

Removing duplicate values:

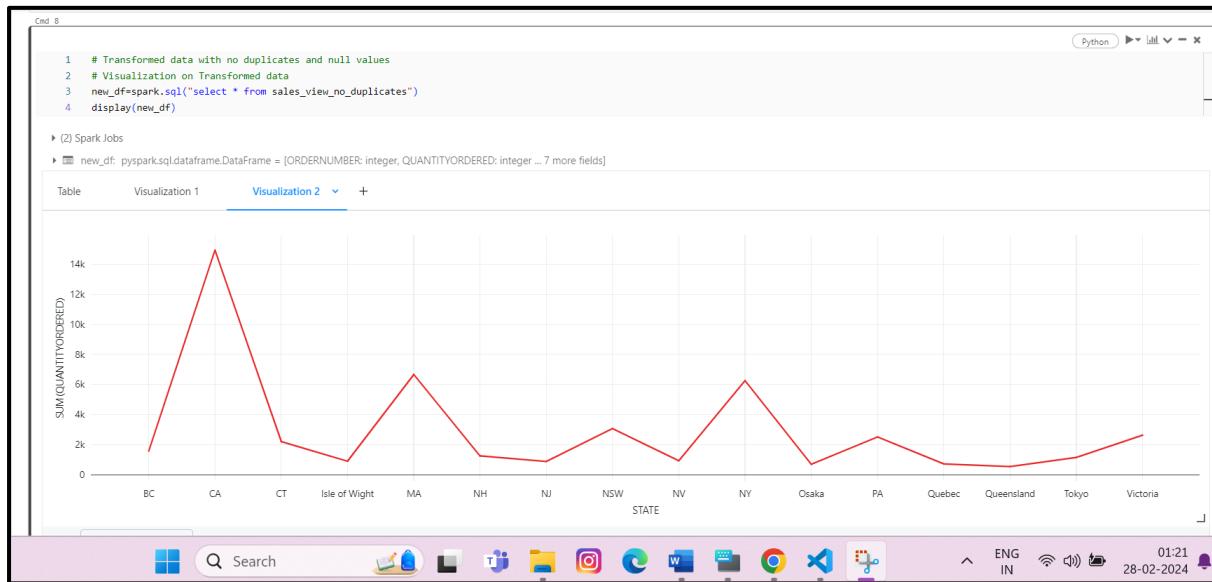
The screenshot shows a Databricks notebook titled 'Project2'. The sidebar on the left includes 'Workflows', 'Compute', 'SQL' (with 'SQL Editor', 'Queries', 'Dashboards', 'Alerts', 'Query History', and 'SQL Warehouses'), 'Data Engineering' (with 'Job Runs', 'Data Ingestion', 'Delta Live Tables'), and 'Machine Learning' (with 'Experiments', 'Features', and 'Models'). The main area shows a cell with the following SQL code:

```
distinct_df=spark.sql("""  
CREATE OR REPLACE TEMP VIEW sales_view_no_duplicates AS  
SELECT DISTINCT *  
FROM sales_null """)  
spark.sql("SELECT * FROM sales_view_no_duplicates").display()
```

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

ORDERNUMBER	QUANTITYORDERED	PRODUCTLINE	PRICEACH	CUSTOMERNAME	PHONE
1	10223	Motorcycles	100	Australian Collectors, Co.	03 9520 4
2	10210	Motorcycles	49.67	Osaka Souvenirs Co.	+81 06 6:
3	10232	Vintage Cars	100	giftsbymail.co.uk	(198) 555
4	10321	Classic Cars	100	FunGiftIdeas.com	50855525
5	10193	Vintage Cars	97.55	Australian Collectables, Ltd	61-9-384
6	10414	Classic Cars	100	Gifts4AllAges.com	61755595
7	10353	Planes	100	Gift Ideas Corp.	2035554

## Visualise Transformed Data with no null and no duplicate values:



## Operations On Transformed Data

### Filtering:

The figure shows a Jupyter Notebook interface with a Python code cell and a table output. The code cell contains:1 # Operations on Transformed data  
2  
3 # 1) Filtering  
4 spark.sql("select \* from sales\_view\_no\_duplicates where STATE='PA' ").display()The table output shows a filtered dataset for the state PA. The columns are ORDERNUMBER, QUANTITYORDERED, PRODUCTLINE, PRICEEACH, CUSTOMERNAME, PHONE, STATE, and ORDERDATE. The data includes:

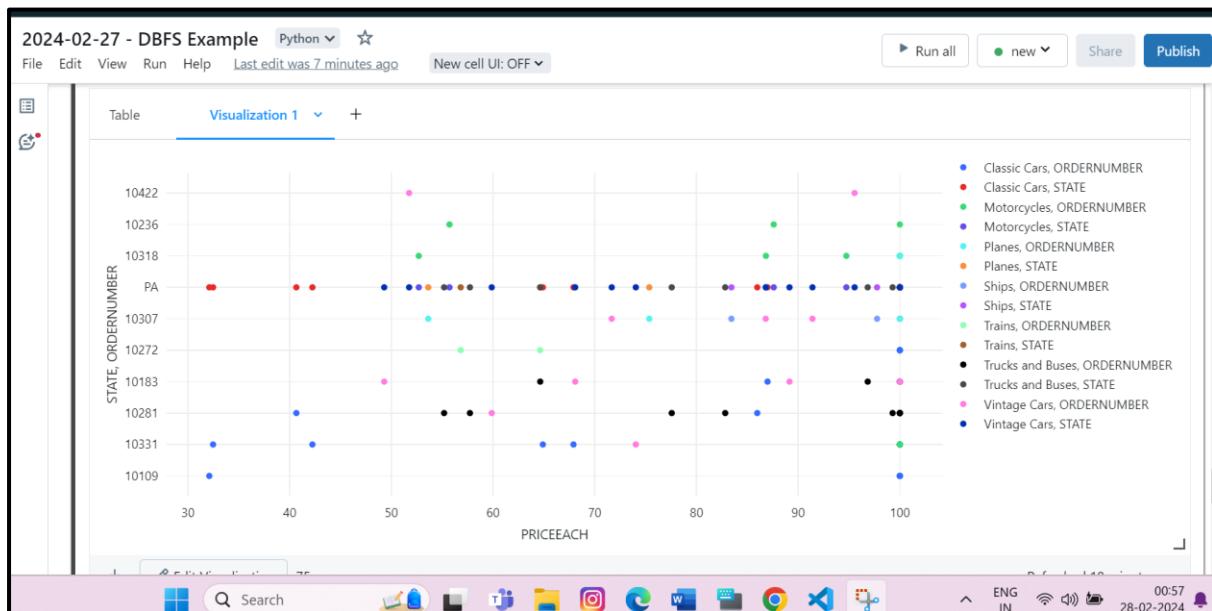
	ORDERNUMBER	QUANTITYORDERED	PRODUCTLINE	PRICEEACH	CUSTOMERNAME	PHONE	STATE	ORDERDATE
1	10318	31	Planes	100	Diecast Classics Inc.	215551555	PA	29-10-2006
2	10422	51	Vintage Cars	95.55	Diecast Classics Inc.	215551555	PA	07-01-2006
3	10281	36	Trucks and Buses	77.57	Diecast Classics Inc.	215551555	PA	03-12-2010
4	10318	26	Motorcycles	86.83	Diecast Classics Inc.	215551555	PA	24-12-2008
5	10236	36	Motorcycles	87.6	Motor Mint Distributors Inc.	215559857	PA	17-12-2008
6	10183	21	Trucks and Buses	96.84	Classic Gift Ideas, Inc	215554695	PA	26-10-2005
7	10272	39	Classic Cars	100	Diecast Classics Inc.	215551555	PA	20-07-2003

75 rows | 1.31 seconds runtime

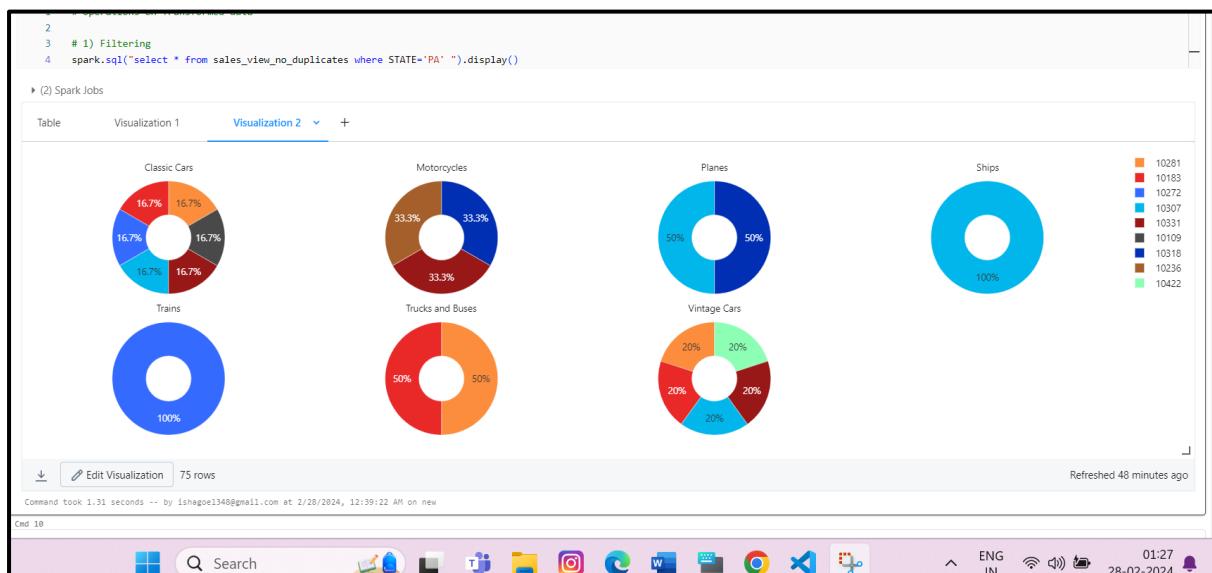
Refreshed now

## Visualization for filter:

### Scatter:



### Pie Chart:



## Aggregations

### Sum():

```
1 # 2) Aggregations
2 # sum()
3 spark.sql("select sum(PRICEEACH) as Price from sales_view_no_duplicates").display()
```

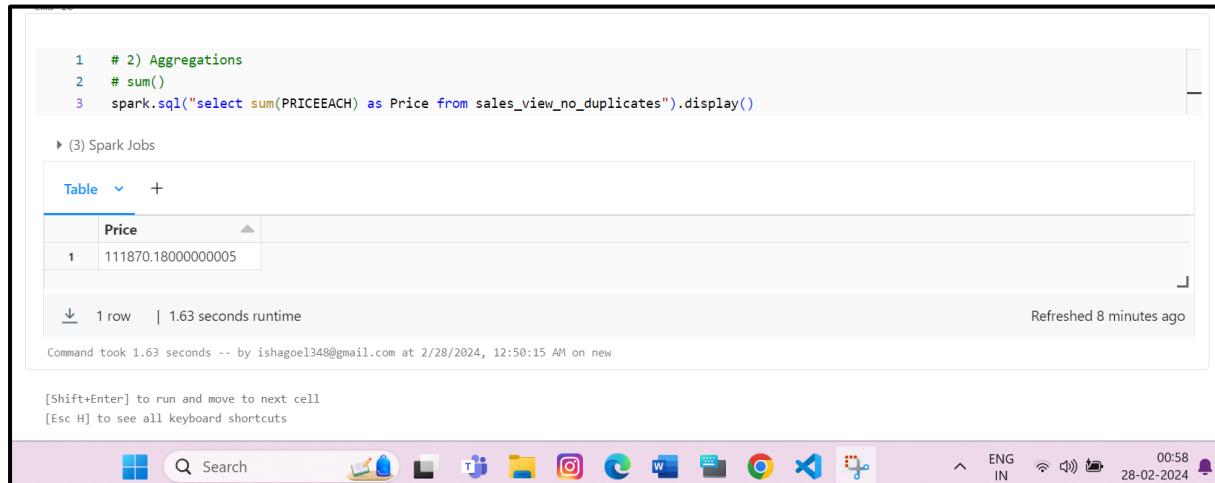
► (3) Spark Jobs

Price
1 111870.18000000005

↓ 1 row | 1.63 seconds runtime Refreshed 8 minutes ago

Command took 1.63 seconds -- by ishagoel348@gmail.com at 2/28/2024, 12:50:15 AM on new

[Shift+Enter] to run and move to next cell  
[Esc H] to see all keyboard shortcuts



### Avg():

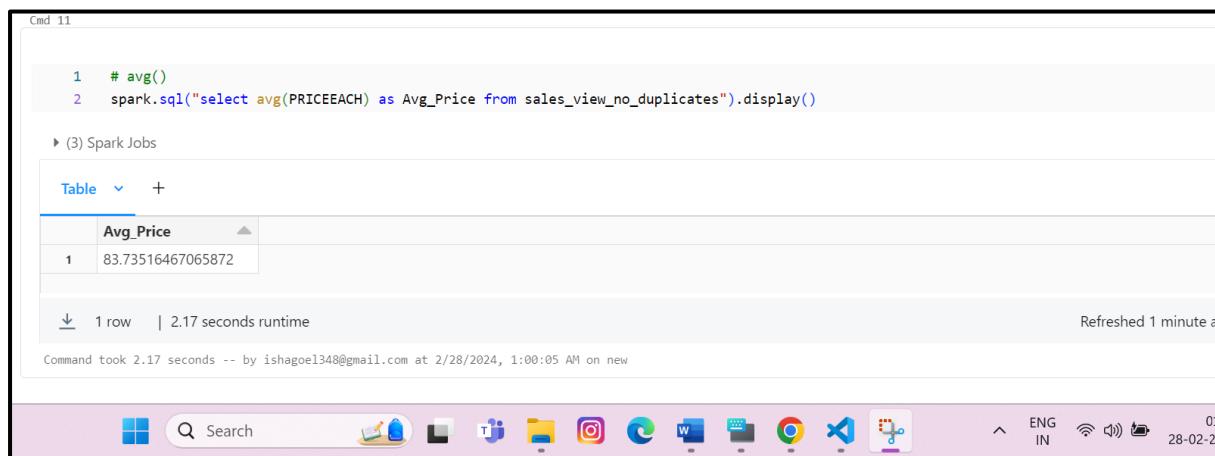
```
1 # avg()
2 spark.sql("select avg(PRICEEACH) as Avg_Price from sales_view_no_duplicates").display()
```

► (3) Spark Jobs

Avg_Price
1 83.73516467065872

↓ 1 row | 2.17 seconds runtime Refreshed 1 minute ago

Command took 2.17 seconds -- by ishagoel348@gmail.com at 2/28/2024, 1:00:05 AM on new



### Mean():

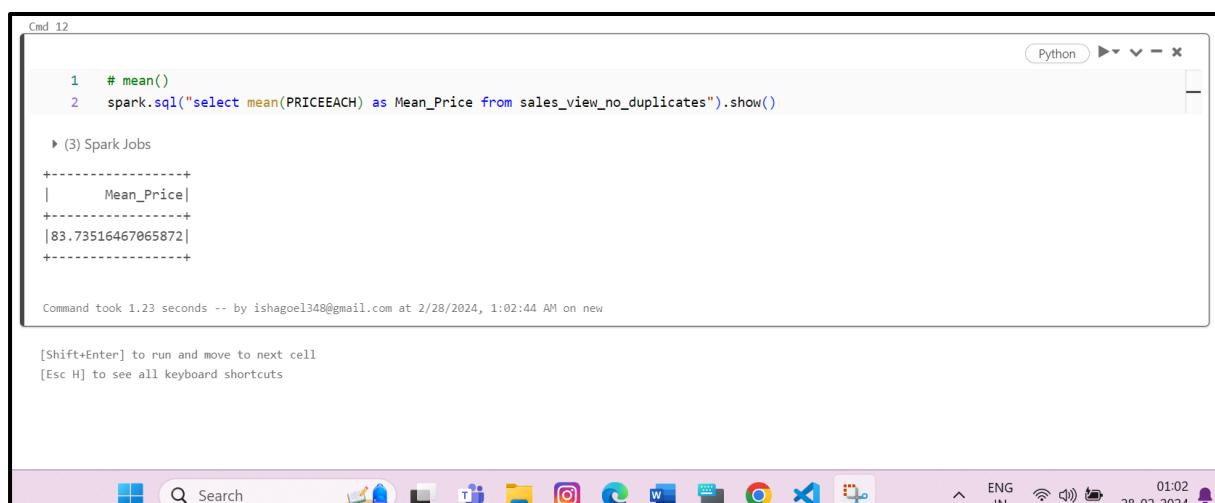
```
1 # mean()
2 spark.sql("select mean(PRICEEACH) as Mean_Price from sales_view_no_duplicates").show()
```

► (3) Spark Jobs

Mean_Price
83.73516467065872

Command took 1.23 seconds -- by ishagoel348@gmail.com at 2/28/2024, 1:02:44 AM on new

[Shift+Enter] to run and move to next cell  
[Esc H] to see all keyboard shortcuts



# Anomaly 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

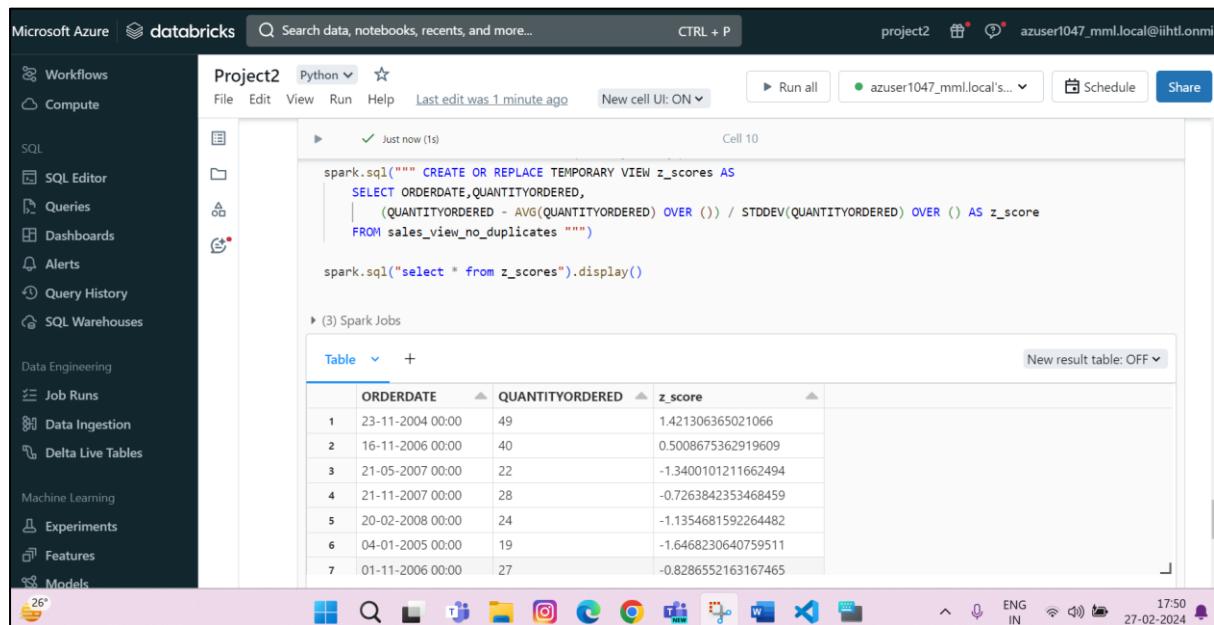
X is the data point,

$\mu$  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 the Databricks interface with a Python notebook titled "Project2". The sidebar on the left contains links for Workflows, Compute, SQL (SQL Editor, Queries, Dashboards, Alerts, Query History, SQL Warehouses), Data Engineering (Job Runs, Data Ingestion, Delta Live Tables), Machine Learning (Experiments, Features, Models), and a system status bar at the bottom.

In the main area, a code cell labeled "Cell 10" contains the following SQL query:

```
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 """)

spark.sql("select * from z_scores").display()
```

Below the code cell, a table titled "Table" displays the results of the query:

	ORDERDATE	QUANTITYORDERED	z_score
1	23-11-2004 00:00	49	1.421306365021066
2	16-11-2006 00:00	40	0.5008675362919609
3	21-05-2007 00:00	22	-1.3400101211662494
4	21-11-2007 00:00	28	-0.7263842353468459
5	20-02-2008 00:00	24	-1.1354681592264482
6	04-01-2005 00:00	19	-1.6468230640759511
7	01-11-2006 00:00	27	-0.8286552163167465

## Display of z-scores

The screenshot shows a Microsoft Azure Databricks notebook titled "Project2" in Python. The code cell contains the following SQL query:

```
# -- 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""")
spark.sql("select * from anomalies").display()
```

The resulting table displays 549 rows of data with columns: ORDERDATE, QUANTITYORDERED, and z\_score. The data shows various dates from 2004 to 2010 and corresponding order quantities.

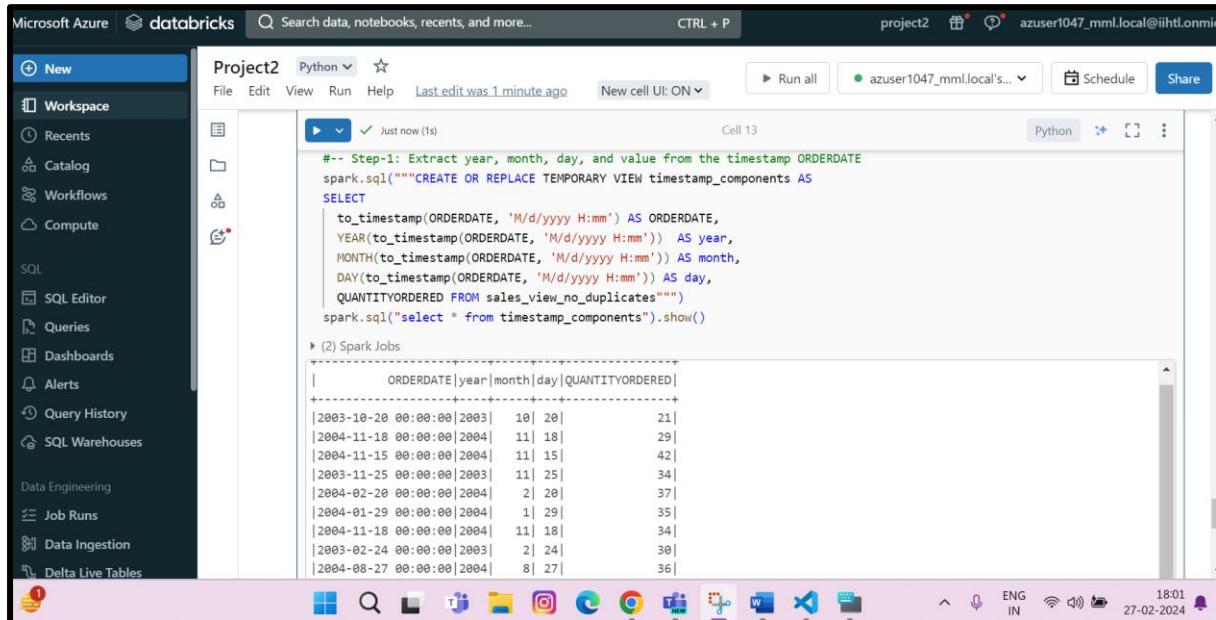
## Visualise anomaly:

The screenshot shows the "Visualization Editor" interface. The "General" tab is selected, showing a "Bar" chart type. The X axis is set to "QUANTITYORDERED" and the Y axis is also set to "QUANTITYORDERED". A legend on the right maps colors to values: 11 (dark blue), 13 (orange), 15 (green), 16 (red), 18 (light blue), 19 (dark red), 20 (light green), 21 (pink), 22 (grey), 23 (purple), 24 (dark grey), 25 (yellow-green). A tooltip message indicates "Not all data shown" and suggests filtering or aggregating data to improve performance.

## 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.

## Extract Timestamp Components: Extract year, month, day, and value from the timestamp ORDERDATE.



```
Cell 13
#-- Step-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, 'M/d/yyyy H:mm') AS ORDERDATE,
    YEAR(to_timestamp(ORDERDATE, 'M/d/yyyy H:mm')) AS year,
    MONTH(to_timestamp(ORDERDATE, 'M/d/yyyy H:mm')) AS month,
    DAY(to_timestamp(ORDERDATE, 'M/d/yyyy H:mm')) AS day,
    QUANTITYORDERED FROM sales_view_no_duplicates""")
spark.sql("select * from timestamp_components").show()
```

ORDERDATE	year	month	QUANTITYORDERED
2003-10-20 00:00:00	2003	10	20
2004-11-18 00:00:00	2004	11	18
2004-11-15 00:00:00	2004	11	15
2003-11-25 00:00:00	2003	11	25
2004-02-20 00:00:00	2004	2	20
2004-01-29 00:00:00	2004	1	29
2004-11-18 00:00:00	2004	11	18
2003-02-24 00:00:00	2003	2	24
2004-08-27 00:00:00	2004	8	27

## Aggregate Data: Aggregate data by year and month

Code:



```
Cell 14
# Step 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""")
spark.sql("select * from monthly_aggregation").show()
```

## Output:

The screenshot shows the Microsoft Azure Databricks workspace interface. On the left, the sidebar includes options like New, Workspace, Recents, Catalog, Workflows, Compute, SQL, SQL Editor, Queries, Dashboards, Alerts, Query History, SQL Warehouses, Data Engineering, Job Runs, Data Ingestion, and Delta Live Tables. The main area displays a Python notebook titled 'Project2'. Cell 14 shows the output of a Spark job with the following data:

year month	avg_value
[NULL] [NULL]	35.15242494226328
[2003] [2]	30.0
[2003] [7]	37.0
[2003] [8]	45.0
[2003] [10]	29.33333333333332
[2003] [11]	32.77777777777778
[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

## Analyse Seasonality:

The screenshot shows the Microsoft Azure Databricks workspace interface. The sidebar and notebook structure are identical to the previous screenshot. Cell 15 contains the following Python code and its output:

```
# Step 3: Analyze Seasonality
##-- Display the aggregated data
spark.sql("SELECT * FROM monthly_aggregation").show()
```

year month	avg_value
[NULL] [NULL]	35.15242494226328
[2003] [2]	30.0
[2003] [7]	37.0
[2003] [8]	45.0
[2003] [10]	29.33333333333332
[2003] [11]	32.77777777777778
[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

## Visualize Seasonality:

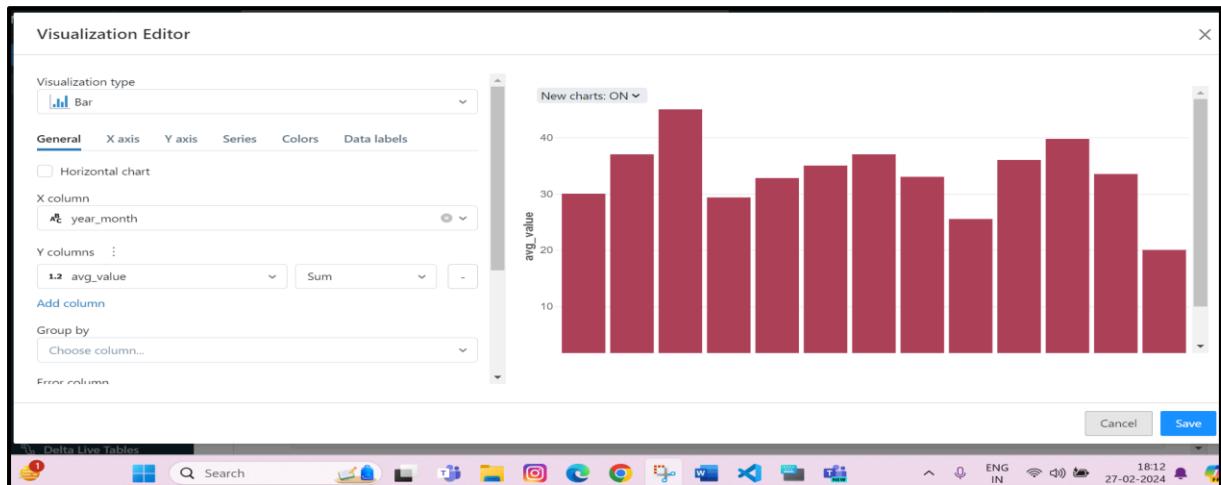
The screenshot shows the Microsoft Azure Databricks workspace interface. The sidebar and notebook structure are identical to the previous screenshots. Cell 16 contains the following Python code and its output:

```
#-- Visualize 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""").display()
```

The output shows a table with two columns: 'year\_month' and 'avg\_value'. The data is as follows:

year_month	avg_value
1	35.15242494226328
2	30
3	37
4	45
5	29.33333333333332
6	32.77777777777778
7	35

## Through 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.

### Calculate moving averages

The screenshot shows a Python code cell in an Azure Databricks notebook. The code applies a moving average to smooth out seasonal variations in sales data. It uses a temporary view named 'moving\_averages' with a window of 3 preceding and 3 following rows for each ORDERDATE. The code is as follows:

```
1 # 1) Apply moving average to smooth out seasonal variations
2
3 spark.sql("""CREATE OR REPLACE TEMPORARY VIEW moving_averages AS
4 SELECT
5     ORDERDATE,
6     QUANTITYORDERED,
7     AVG(QUANTITYORDERED) OVER (ORDER BY ORDERDATE ROWS BETWEEN 3 PRECEDING AND 3 FOLLOWING) AS moving_avg
8     FROM sales_view_no_duplicates""")
9
```

The output of the cell is 'Out[24]: DataFrame[]'.

At the bottom of the screen, the taskbar shows various application icons and the system tray indicates the date as 28-02-2024 and the time as 01:11.

# Analyse Moving Averages

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

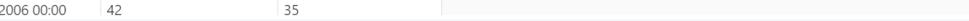
```
1 # 2) Analyze Moving Averages
2
3 spark.sql("select * from moving_averages").display()
```

▶ (3) Spark Jobs

Table +

	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

↓ 1,336 rows | 1.19 seconds runtime Refreshed now



# Visualize moving averages

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

Child 15

Python   

```
1 # 3) Visualize Moving Averages
2
3 display(spark.sql("select * from moving_averages"))
```

▶ (3) Spark Jobs

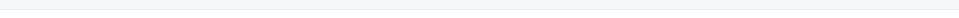
Table  +

	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	31.333333333333332
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

1,336 rows | 1.02 seconds runtime

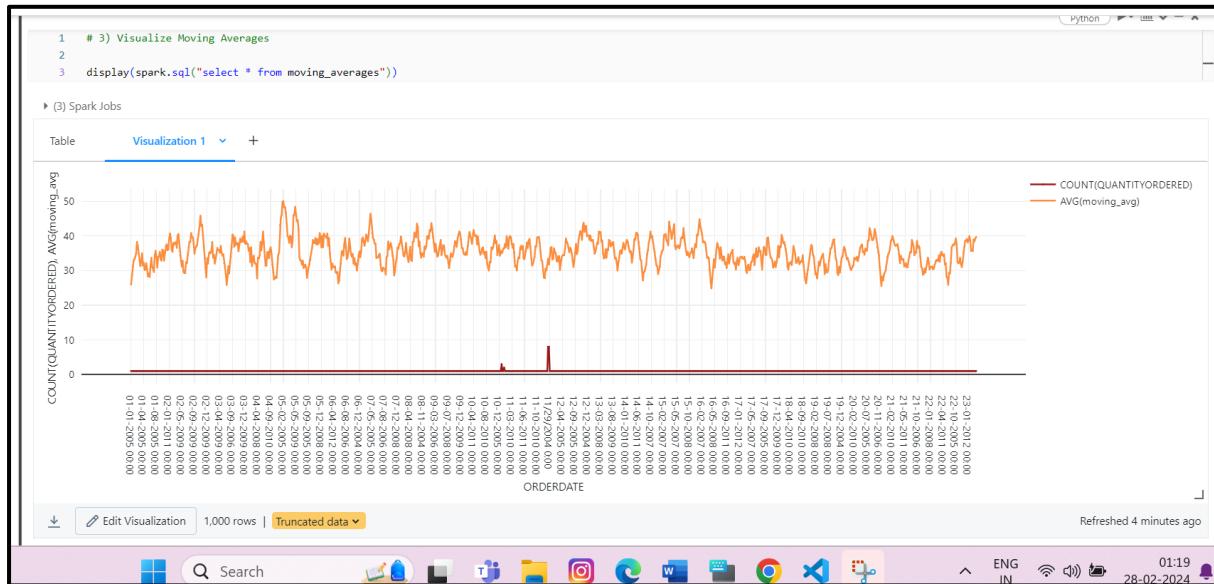
Refreshed now

Command took 1.02 seconds on 12/20/2024 at 1:15 AM AMERICA/SAO\_PAULO



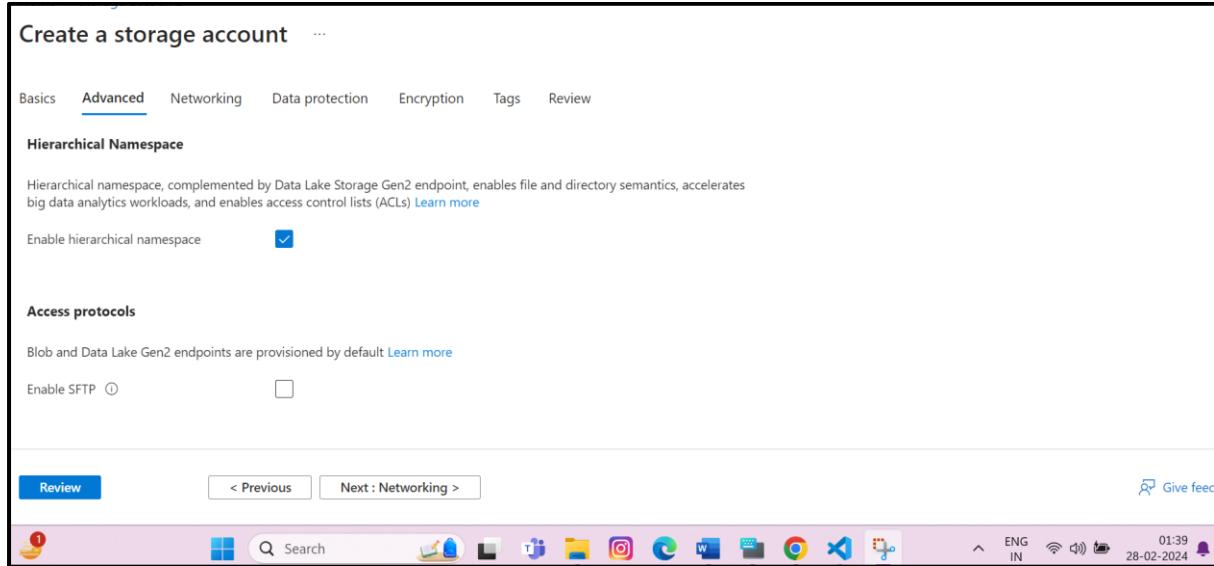
01:15  
28-02-2024

## Through Line Chart:



## Writing the transformed file to destination

Create a destination storage account to write the transformed data.



In the destination storage account create a container to write the transformed data:

The screenshot shows the Azure Storage Account 'destinationhexa' interface. In the left sidebar, under 'Data storage', 'Containers' is selected. The main area displays a table of existing containers, including '\$logs'. On the right, a modal window titled 'New container' is open, prompting for a name ('new') and access level ('Private (no anonymous access)'). A note indicates that anonymous access is disabled. At the bottom of the modal are 'Create' and 'Give feedback' buttons.

Write the transformed data to “new” in “destination” Azure Data Lake Storage:

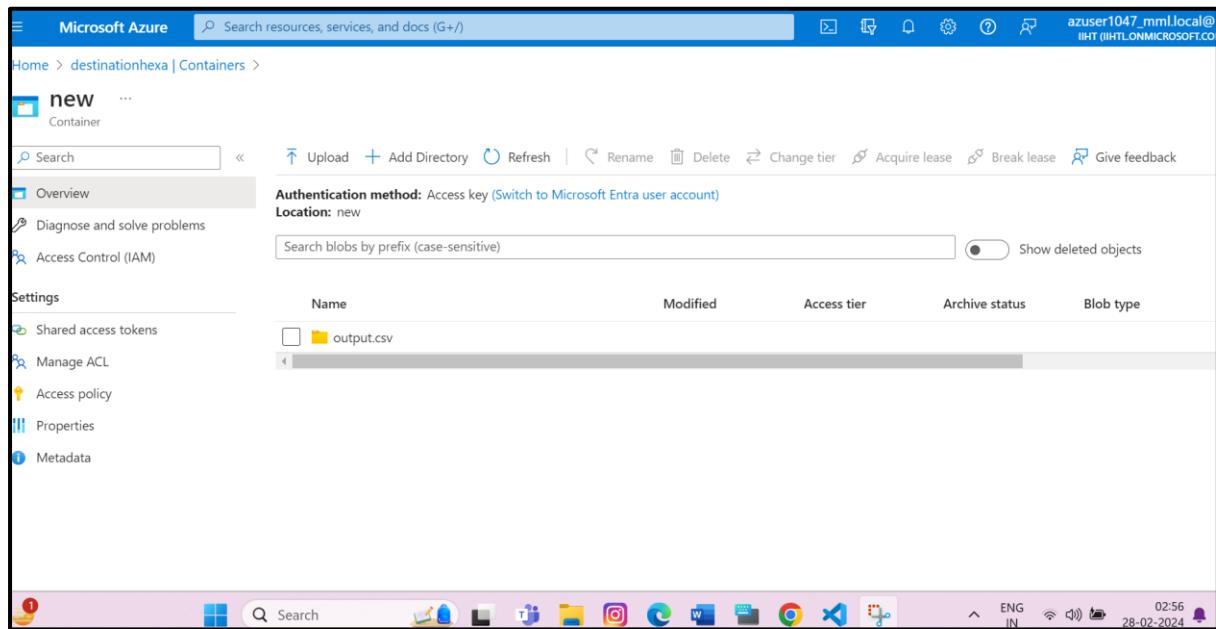
The screenshot shows an Azure Databricks notebook titled 'Explore hive\_metastore.default.salesdata'. The code cell (Cell 10) contains the following Python code:

```
# Writing the transformed data to a destination in Azure Data Lake Storage account.
# Replace 'your_storage_account_key' with the actual storage account key

new_df.write \
    .option("header", "true") \
    .option("fs.azure.account.key.destinationhexa.dfs.core.windows.net",
           "hQDzWDjTDm2FAizIKS1RNQckPi9y0Q6F4HfwtsODRxjuUC/GNFosd1BA4tuJ3ttokmdgd1AJJhiV+AStyyom7Q==") \
    .mode("overwrite") \
    .csv("abfss://new@destinationhexa.dfs.core.windows.net/output.csv")
```

Below the code cell, it says '(2) Spark Jobs'. At the bottom of the notebook interface, there are keyboard shortcuts: [Shift+Enter] to run and move to next cell, and [Esc H] to see all keyboard shortcuts.

## Writing into output.csv file inside the “new” container:



The screenshot shows the Microsoft Azure Storage Explorer interface. The left sidebar displays navigation options like Home, destinationhexa, Containers, and new (selected). The main area shows a table of blobs in the 'new' container. The table includes columns for Name, Modified, Access tier, Archive status, and Blob type. One blob named 'output.csv' is listed. The top navigation bar includes a search bar, upload, add directory, refresh, rename, delete, change tier, acquire lease, break lease, and give feedback buttons. The bottom taskbar shows various application icons and system status.

Name	Modified	Access tier	Archive status	Blob type
output.csv				

## Files have been written to output.csv:

The screenshot shows the Microsoft Azure Storage Explorer interface. The left sidebar displays the 'new' container with various settings like Shared access tokens, Manage ACL, Access policy, Properties, and Metadata. The main area lists blobs with the following details:

Name	Modified	Access tier	Archive status	Blob type	Size
[...]	2/28/2024, 2:55:55 AM	Hot (Inferred)		Block blob	112
_committed_6463178006761651032	2/28/2024, 2:55:45 AM	Hot (Inferred)		Block blob	0 B
_started_6463178006761651032	2/28/2024, 2:55:55 AM	Hot (Inferred)		Block blob	0 B
_SUCCESS				Block blob	0 B
part-00000-tid-6463178006761651032-b91b39db-9e...	2/28/2024, 2:55:55 AM	Hot (Inferred)		Block blob	109.

The screenshot shows the Microsoft Azure Storage Explorer interface focusing on the 'output.csv/part-00000-tid-6463178006761651032-b...' blob. The right pane displays the CSV data with the following rows:

ORDERNUMBER	QUANTITYORDERED	PRODUCTLINE	PRICEEACH	CUSTOMERNAME	PHONE	STATE					
10107,30	Motorcycles,95.7	Land of Toys Inc.	2125557818	NY	2/24/2003	0:00					
10376,35	Classic Cars,100.0	Boards & Toys Co.	3105552373	CA	04-07-2005	00					
10210,31	Planes,100.0	Osaka Souveniers Co.,	+81 06 6342 5555	Osaka,	09-03-200	00					
10228,31	Vintage Cars,100.0	Cambridge Collectables Co.	,6175555555	MA	24-06-200	00					
10357,28	Trucks and Buses,100.0	Mini Gifts Distributors Ltd.	,4155551450	CA	10182,44	Classic Cars,69.84	Mini Gifts Distributors Ltd.	,4155551450	CA	23-06-200	00
10193,25	Classic Cars,76.26	Australian Collectables, Ltd	,61-9-3844-6555								
10274,40	Planes,65.08	Collectables For Less Inc.	,6175558555	MA	17-06-2009	00					
10319,43	Classic Cars,85.69	Microscale Inc.	,2125551957	NY	04-07-2010	00:00					
10263,31	Motorcycles,79.91	Gift Depot Inc.	,2035552570	CT	17-08-2010	00:00					
10283,20	Ships,94.14	"Royal Canadian Collectables, Ltd."	,(604) 555-4555	BC							

## **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.