**AZURE DATA BRICKS**

**data=spark.read.csv('file:/Workspace/Users/d-user4@heu.ai/Educational\_Universities\_data 5.csv')**

1. Importing and exporting data

Azure Blob Storage is trying to access your storage account using anonymous credentials, but your storage account is not set up to allow public access. To resolve this, you need to authenticate with Azure using a proper method like a **SAS token**, **account key**, or **Azure Active Directory (AAD) token**.

spark.conf.set( "fs.azure.account.key.bharathiwork.blob.core.windows.net", "<your-storage-account-key>" )

where is storage account key:

* In the Azure Portal, go to "Storage accounts" in the left-hand menu.
* Select your storage account (in your case, bharathiwork).
* Once in your storage account, go to the "Security + networking" section in the left-hand menu.
* Click on "Access keys" under the "Settings" menu.
* You’ll see two keys, key1 and key2. You can use either of them.
* Click on the "Show keys" button to reveal the key.
* Copy the value of the key you want to use.

**IMPORTING DATA**:

df=spark.read.format("csv").option("header","true").load("wasbs://input@bharathiwork.blob.core.windows.net/data.csv")

**EXPORTING DATA:**

df.write.format("csv").option("header", "true").save("wasbs://input@bharathiwork.blob.core.windows.net/output/results.csv")

1. Exploring and analyzing data using Spark DataFrame API

 Load your data into a DataFrame.

 Explore your data using **show, printSchema, and describe**.

 Analyse your data with operations like filtering, grouping, and aggregation.

 Save the results for further use.

**View Data**

* Quickly inspect the first few rows of your data.

python

df.show(5)

A screenshot of a computer program

Description automatically generated

**Understand the Data Structure**

* Check the schema (column names and data types) of your DataFrame.

python

df.printSchema()

A screenshot of a computer program

Description automatically generated

**Summary Statistics**

* Get summary statistics like mean, count, min, max, etc., for each column.

python

df.describe().show()

A screenshot of a computer program

Description automatically generated

**Select and Filter Data**

* **Select specific columns**:

df.select("column\_name").show()

A screenshot of a computer

Description automatically generated

1. Visualizing data with Databricks notebooks

import matplotlib.pyplot as plt

# Convert Spark DataFrame to Pandas DataFrame for easier plotting

pdf = df.toPandas()

# Plot a simple bar chart

plt.figure(figsize=(10, 6))

pdf['column\_name'].value\_counts().plot(kind='bar')

plt.title("Bar Chart Example")

plt.show()



1. Data Engineering with Databricks

**Extract, Load, Transform (ELT)**: Data is ingested from various sources into a data lake or storage in its raw form and later transformed as needed.

filtered\_df = df.filter(df["Ranking"] > 500)

# Select specific columns

transformed\_df = filtered\_df.select("Technologies/Courses Offered", "University Name", "Costing")

A screenshot of a computer

Description automatically generated

# Save transformed data back to Blob Storage or Delta Lake for further use

df.write.format("csv").option("header", "true").save("wasbs://output@bharathiwork.blob.core.windows.net/transformed\_data")

A screenshot of a computer

Description automatically generated

transformed\_df.cache()

**Example:**

Let's say you have a DataFrame transformed\_df that you need to use in several different operations:

python

Copy code

# Cache the DataFrame for quicker access in future operations

transformed\_df.cache()

# Now use the cached DataFrame in multiple operations

result1 = transformed\_df.filter("country = 'USA'")

result2 = transformed\_df.groupBy("age").count()

* **Without caching**: Every time transformed\_df is used, Databricks would reload or recalculate the data.
* **With caching**: After the first operation, transformed\_df is stored in memory, so subsequent operations are much faster.

1. Data ingestion and integration

Basic data ingestion (reading from storage), integration (joining two datasets), and exporting the integrated data back to the storage

**Step 1: Ingest Data**

# Reading data from Blob Storage

df1 = spark.read.format("csv").option("header", "true").load("wasbs://input@bharathiwork.blob.core.windows.net/data1.csv")

df2 = spark.read.format("csv").option("header", "true").load("wasbs://input@bharathiwork.blob.core.windows.net/data2.csv")

**Step 2: Integrate Data**

# Joining the two DataFrames on a common column, say "id"

joined\_df = df1.join(df2, df1["id"] == df2["id"], "inner")

**Step 3: Export the Integrated Data**

# Writing the integrated data back to Blob Storage as a CSV

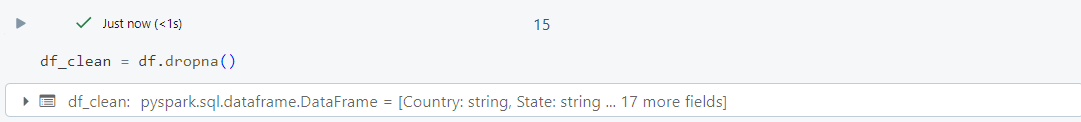
joined\_df.write.format("csv").option("header", "true").save("wasbs://output@bharathiwork.blob.core.windows.net/integrated\_data.csv")

1. Data cleaning and pre-processing

**Drop Missing Values**:

* Remove rows with any missing (null) values.

df\_clean = df.dropna()



**Remove Duplicates**:

* Remove duplicate rows.

df\_clean = df\_clean.dropDuplicates()



**Handle Missing Values (Alternative to Dropping)**:

* Fill missing values in a specific column, say city, with a default value "Unknown".

df\_clean = df\_clean.fillna({"city": "Unknown"})

1. Transforming and aggregating data using Spark SQL

**Step 1: Ingest Data**

* **Load the data into a DataFrame**:

# Load the CSV data into a DataFrame

df = spark.read.format("csv").option("header", "true").load("wasbs://input@bharathiwork.blob.core.windows.net/sales\_data.csv")

**Step 2: Register the DataFrame as a Temporary SQL Table**

* **Create a temporary table to use SQL queries**:

# Register the DataFrame as a SQL temporary view

df.createOrReplaceTempView("sales")

A screenshot of a computer

Description automatically generated

**Step 3: Transform and Aggregate Data Using SQL Queries**

1. **Transform Data**:
   * Select specific columns and apply transformations (e.g., calculate total\_sales by multiplying quantity and price).

transformed\_df = spark.sql("""

SELECT

product\_id,

quantity,

price,

(quantity price) AS total\_sales

FROM

sales

""")

A screenshot of a computer

Description automatically generated

1. **Aggregate Data**:

Calculate the total sales for each product.

aggregated\_df = spark.sql("""

SELECT

product\_id,

SUM(quantity) AS total\_quantity,

SUM(total\_sales) AS total\_sales\_amount

FROM

sales

GROUP BY

product\_id

""")

A screenshot of a computer program

Description automatically generated

1. **Joining and Merging Datasets**

**Joining**: Combine columns from two or more DataFrames based on a related column (e.g., INNER JOIN, LEFT JOIN).

**Merging**: Combining multiple DataFrames into a single DataFrame.

df1 = spark.read.csv("path/to/data1.csv")

df2 = spark.read.csv("path/to/data2.csv")

# Inner Join

joined\_df = df1.join(df2, df1["id"] == df2["id"], "inner")

1. Distributed Computing with Databricks

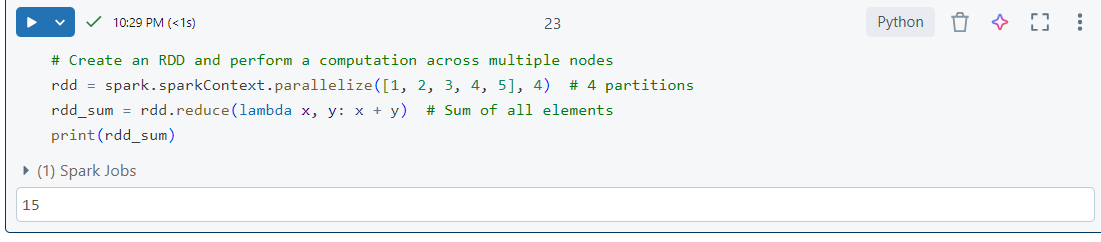
Apache Spark distributes data and computation across multiple nodes in a cluster to handle large datasets efficiently.

# Create an RDD and perform a computation across multiple nodes

rdd = spark.sparkContext.parallelize([1, 2, 3, 4, 5], 4)  # 4 partitions

rdd\_sum = rdd.reduce(lambda x, y: x + y)  # Sum of all elements

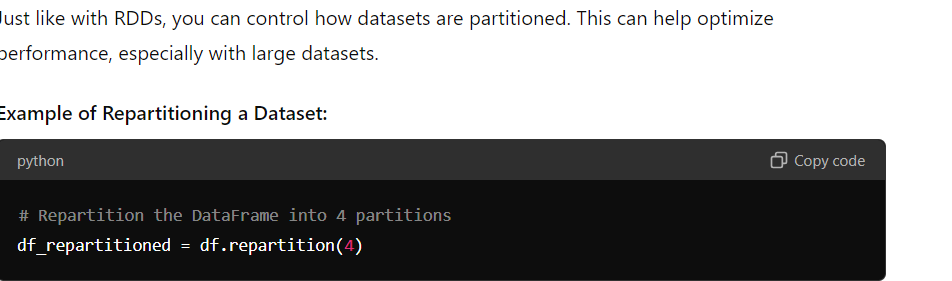
print(rdd\_sum)



Example with dataset:

A screenshot of a computer program

Description automatically generated



1. Understanding the concepts of distributed computing

Distributed computing involves dividing a task into smaller subtasks that are executed on multiple computers. This approach improves processing speed and handles large volumes of data efficiently.

Distributed Computing is a way of solving problems by splitting them into smaller pieces and using multiple computers to work on these pieces at the same time. This makes processing faster and allows handling more data.

Key Points:

1. Divide and Conquer :

- Break a big task into smaller, manageable chunks.

- Each chunk is worked on by a different computer.

2. Work Together :

- Multiple computers process the chunks simultaneously.

- This speeds up the overall task.

3. Combine Results :

- After processing, results from all computers are combined to get the final answer.

4. Scalable and Reliable :

- The system can add more computers if needed.

- It can handle failures by redistributing tasks.

Example :

Imagine you have a large pile of books to sort. Instead of sorting them all by yourself, you give stacks to several friends. Each friend sorts their stack at the same time, and then you combine all the sorted stacks into one organized collection. This way, the job gets done faster and more efficiently.

1. **Leveraging the Power of Apache Spark**

 **In-Memory Computing**: Spark keeps the data in memory during processing, which speeds up operations like grouping and aggregation.

 **Distributed Computing**: The data is divided into partitions and processed in parallel across the cluster nodes.

 **Fault Tolerance**: If a node fails, Spark will use other nodes to recompute the lost data from checkpoints or lineage information.

1. **Partitioning and Parallel Processing of Data**

**Explanation:**

**Partitioning**: This divides a DataFrame or RDD into smaller chunks (partitions). Each partition can be processed independently in parallel, which speeds up data processing.

**Parallel Processing**: Multiple partitions are processed at the same time on different nodes, increasing efficiency and reducing processing time.

**Code Example:**

# Repartition a DataFrame into 5 partitions

df\_repartitioned = df.repartition(5)

1. **Optimizing Spark Jobs for Performance**

**Explanation:**

**Caching**: Storing intermediate data in memory to avoid recomputing it.

**Reducing Shuffles**: Minimize data movement between nodes during operations like joins or aggregations.

**Code Example:**

# Cache DataFrame to improve performance

df.cache()

1. **Data Visualization and Collaboration in Databricks**
2. **Sharing and Collaborating on Notebooks**

**Data Visualization**:

* **Creating Charts and Graphs**: Allows you to convert data into visual formats like charts, graphs, and plots to better understand trends, patterns, and relationships.

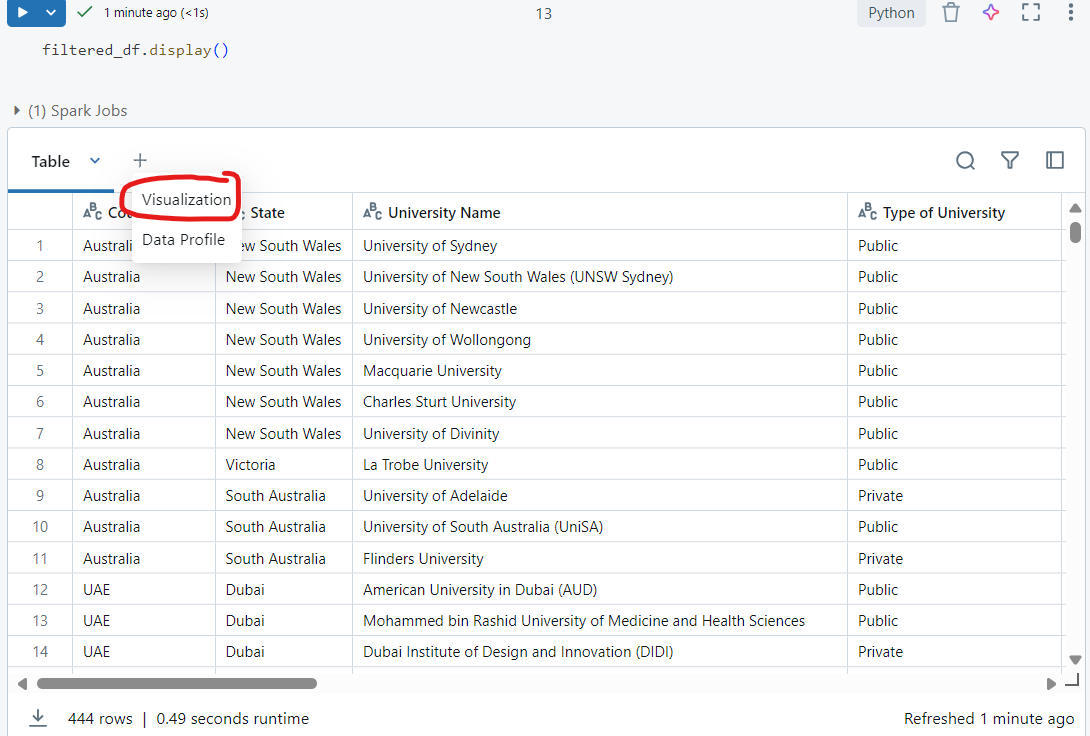
**Collaboration**:

* **Sharing Notebooks**: Enables team members to work together by sharing notebooks, comments, and insights.
* **Joint Analysis**: Multiple users can edit and analyze data in real-time, making it easier to work on data projects as a team.

1. Data

A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated