Large Scale Data Processing

Project Overview

In today's data-driven world, the ability to efficiently process and analyze large volumes of data is crucial for businesses to gain insights and make informed decisions. This project aims to leverage the power of Azure Databricks and PySpark to perform large-scale data processing tasks, including Extract, Transform, and Load (ETL) operations, on massive datasets. By utilizing Databricks clusters, we ensure scalability and parallel processing capabilities.

About Project

This project leverages Azure Databricks and PySpark for large-scale data processing on a sample CSV file containing employee data. The dataset consists of approximately 100,000 records with 11 fields including employee ID, full name, job title, department, gender, ethnicity, age, hire date, annual salary, bonus percentage, country, and exit date.

Architectural Diagram

Key-Components/Requirements of the projects

1. Azure Databricks:

- Azure Databricks provides a cloud-based platform for big data analytics and machine learning. It offers a collaborative environment for data engineers, data scientists, and analysts to work together seamlessly.
- Databricks provides managed Spark clusters, eliminating the need for infrastructure management and allowing teams to focus on data processing tasks.

2. PySpark**:**

- PySpark is the Python API for Apache Spark, a powerful open-source framework for distributed data processing. PySpark simplifies development tasks by providing a Python interface to Spark's capabilities.
- With PySpark, developers can write concise and expressive code to perform complex data transformations, aggregations, and analytics on large datasets.

3. ETL Operations:

- **Extract**: Data ingestion from various sources such as databases, data lakes, streaming platforms, or external APIs.
- **Transform**: Data transformation tasks including cleansing, filtering, aggregating, joining, and enriching datasets to prepare them for analysis.
- **Load**: Storing processed data into target systems such as data warehouses, data lakes, or serving layers for downstream consumption.

4. Scalability with Databricks Clusters:

- Databricks clusters dynamically allocate computational resources based on workload requirements, ensuring optimal performance and resource utilization.
- Autoscaling capabilities automatically adjust cluster size to accommodate changes in workload demand, allowing for seamless scalability without manual intervention.
- Databricks Runtime optimizes performance with built-in optimizations, caching, and tuning for various workloads, resulting in faster processing times.

Azure Resources Used for this Project

1. Azure Data Lake Storage Gen2:

- This is where the Transformed data is Loaded. Azure Data Lake Storage Gen2 provides a scalable and secure platform for storing large volumes of data. It enables us to manage, access, and analyse data effectively

2. Azure Blob Storage:

- This is where the raw data is stored. Azure Blob Storage integral to Microsoft Azure's storage service, is a cloud-based solution tailored for managing vast amounts of unstructured data, encompassing both text and binary data. Termed "Blob" for "Binary Large Object," it signifies a compilation of binary data treated as a singular entity within a database.

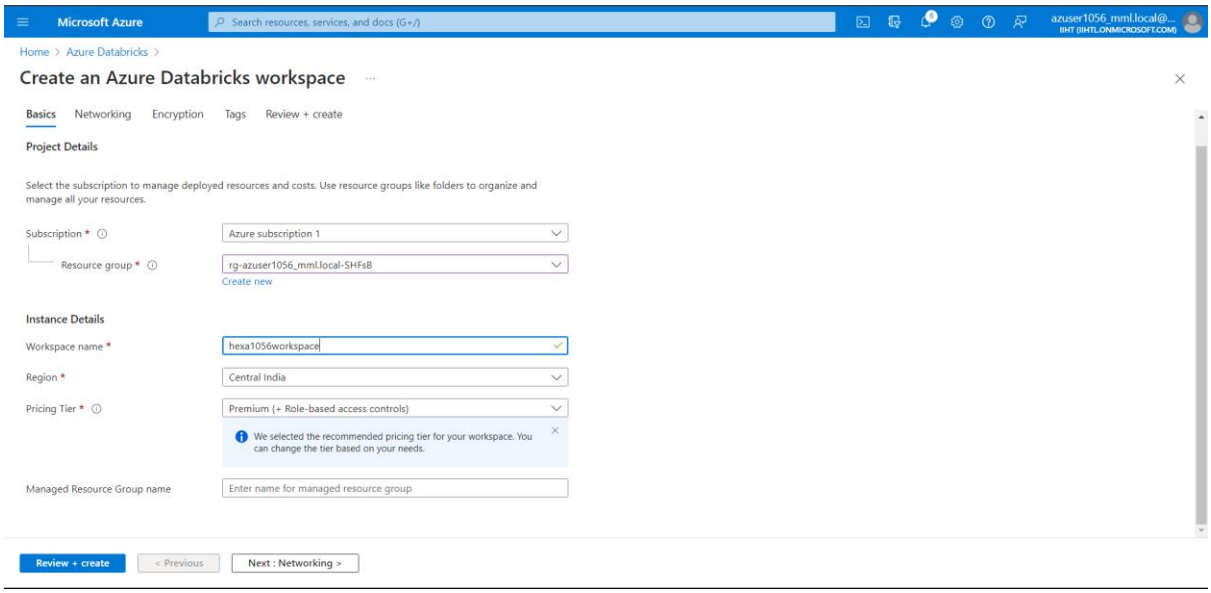
3. Databricks Cluster

- An Azure Databricks cluster process the data depending on the user instructions in the Azure Notebook. It serves as a computational resource facilitating the processing of extensive data and execution of analytics workloads through the Apache Spark platform within the Microsoft Azure cloud.

How It works

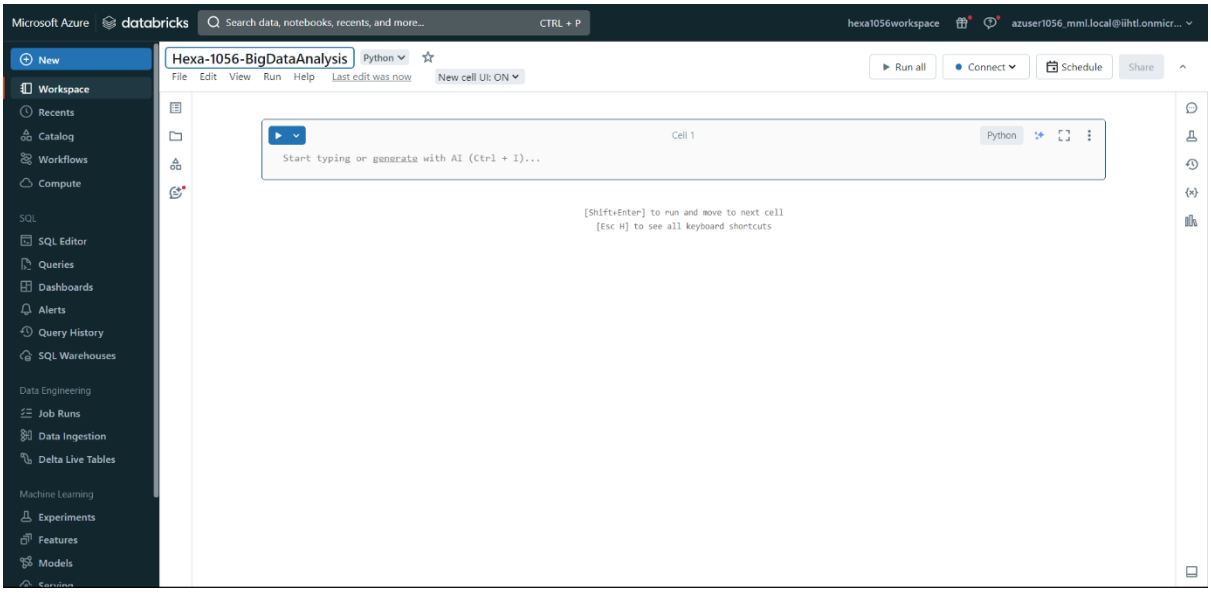
1. Setting Up Azure Databricks Environment:

- Sign in to the Azure portal and create an Azure Databricks workspace. Configure workspace settings, including pricing tier, region, and workspace name



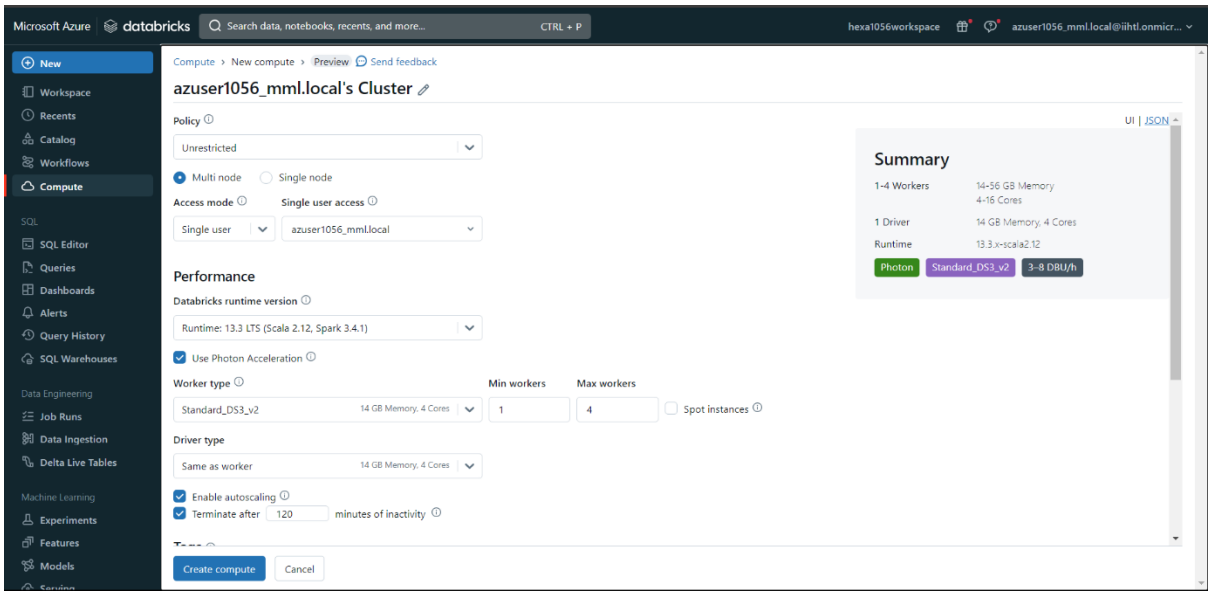
2. Developing PySpark Notebooks:

- Create a new PySpark notebook within the Databricks workspace. Begin writing PySpark code to perform ETL operations, data transformations, and other data processing tasks.



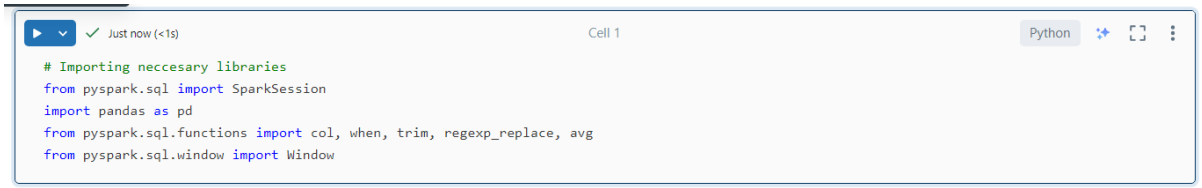
3. Create Cluster and Connecting to notebook

- The cluster is created with 4 working nodes and autoscaling is enabled which automatically adjust cluster size to accommodate changes in workload demand, allowing for seamless scalability without manual intervention.



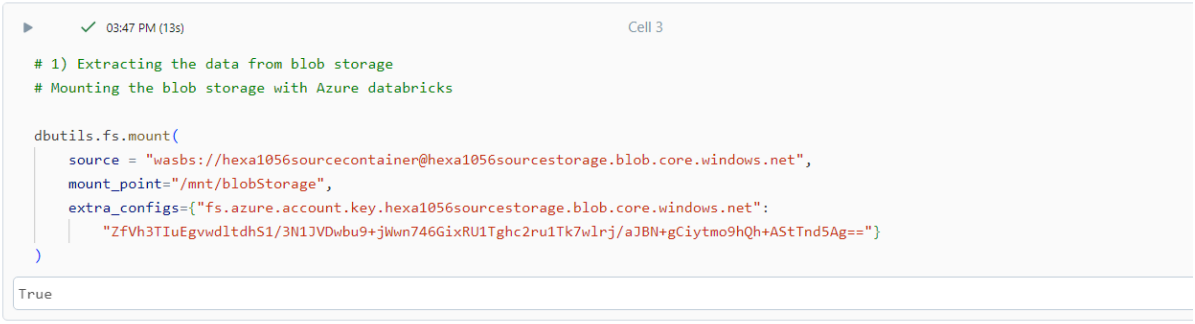
4. Importing Necessary libraries and Creating Spark Session:

- Use SparkSession.builder to configure and create a SparkSession. specify the application name using .appName() and configure any additional Spark options using .config(). Finally, call .getOrCreate() to either create a new SparkSession

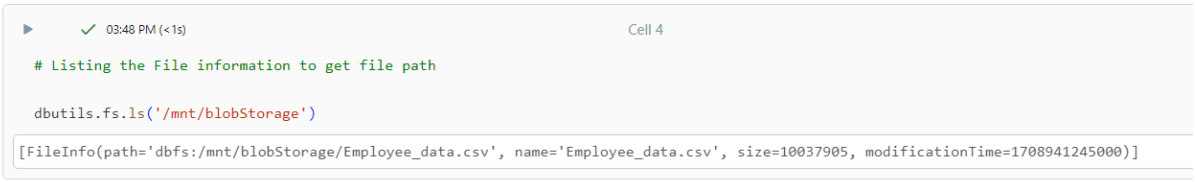


5. Extracting Data from Source storage

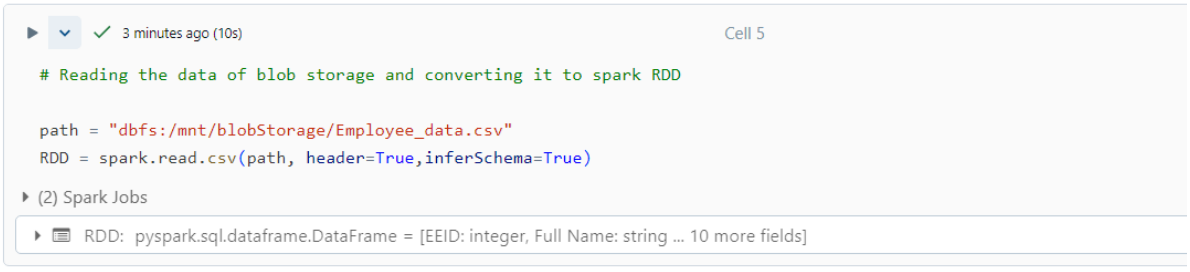
- Connecting data source (Azure Blob Storage) by mounting it to the Databricks File System (DBFS) to simplify data access
- It helps to retrieve raw data for processing and analysis within the PySpark environment.



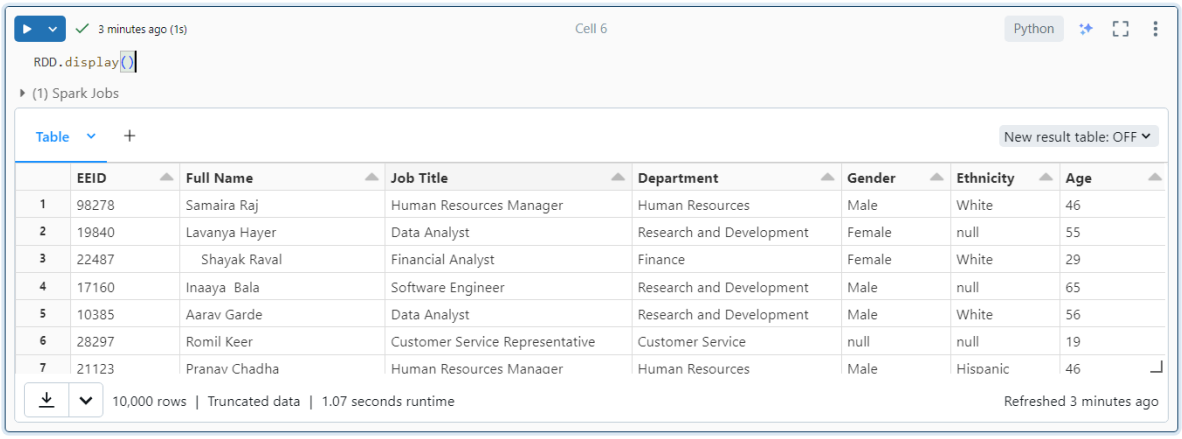
```
03:47 PM (13s)                                                    Cell 3

# 1) Extracting the data from blob storage
# Mounting the blob storage with Azure databricks

dbutils.fs.mount(
    source = "wasbs://hexa1056sourcecontainer@hexa1056sourcestorage.blob.core.windows.net",
    mount_point="/mnt/blobStorage",
    extra_configs={"fs.azure.account.key.hexa1056sourcestorage.blob.core.windows.net":
        "ZfVh3TIuEgvwdltdhS1/3N1JVDwbu9+jWwn746GixRU1Tghc2ru1Tk7wlrj/aJBN+gCiytmo9hQh+AStTnd5Ag=="}
)

True
```



```
03:48 PM (<1s)                                                    Cell 4

# Listing the File information to get file path

dbutils.fs.ls('/mnt/blobStorage')

[FileInfo(path='dbfs:/mnt/blobStorage/Employee_data.csv', name='Employee_data.csv', size=10037905, modificationTime=1708941245000)]
```



```
3 minutes ago (10s)                                               Cell 5

# Reading the data of blob storage and converting it to spark RDD

path = "dbfs:/mnt/blobStorage/Employee_data.csv"
RDD = spark.read.csv(path, header=True,inferSchema=True)

(2) Spark Jobs
RDD:  pyspark.sql.dataframe.DataFrame = [EEID: integer, Full Name: string ... 10 more fields]
```



```
3 minutes ago (1s)                                                Cell 6                    Python

RDD.display()

(1) Spark Jobs
```

Table + New result table: OFF

	EEID	Full Name	Job Title	Department	Gender	Ethnicity	Age
1	98278	Samaira Raj	Human Resources Manager	Human Resources	Male	White	46
2	19840	Lavanya Hayer	Data Analyst	Research and Development	Female	null	55
3	22487	Shayak Raval	Financial Analyst	Finance	Female	White	29
4	17160	Inaaya Bala	Software Engineer	Research and Development	Male	null	65
5	10385	Aarav Garde	Data Analyst	Research and Development	Male	White	56
6	28297	Romil Keer	Customer Service Representative	Customer Service	null	null	19
7	21123	Pranav Chadha	Human Resources Manager	Human Resources	Male	Hispanic	46

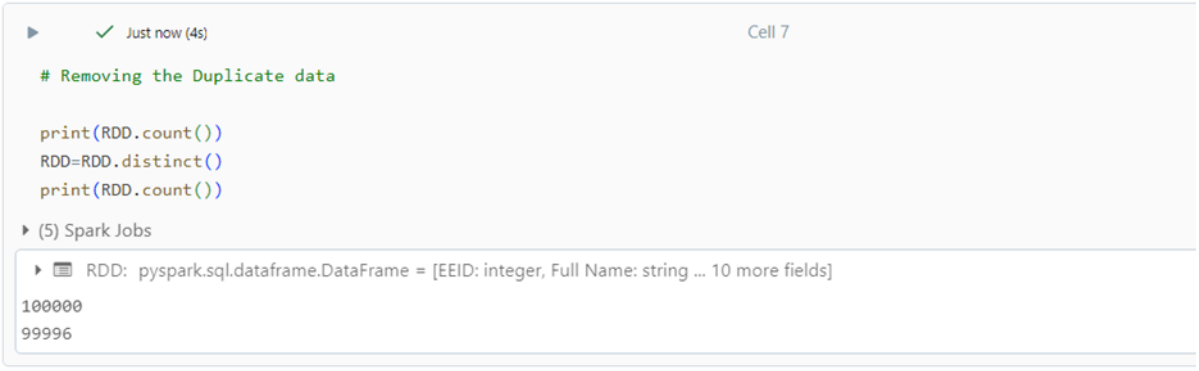
10,000 rows | Truncated data | 1.07 seconds runtime Refreshed 3 minutes ago

6. Transforming the raw data

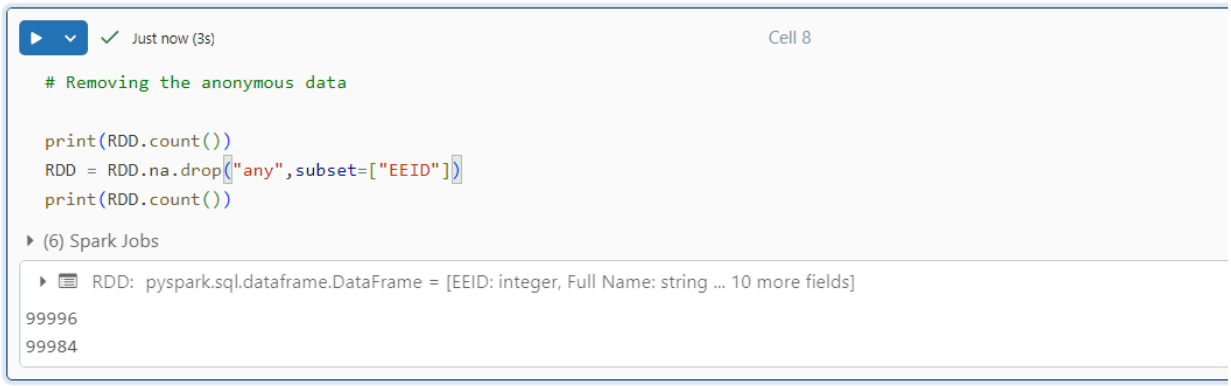
- Utilize PySpark DataFrame transformations and functions to cleanse, transform, and prepare the data for analysis.
- Implement business logic and data processing steps to transform raw dataset up to mark for data analysis purpose.

Transformations done:-

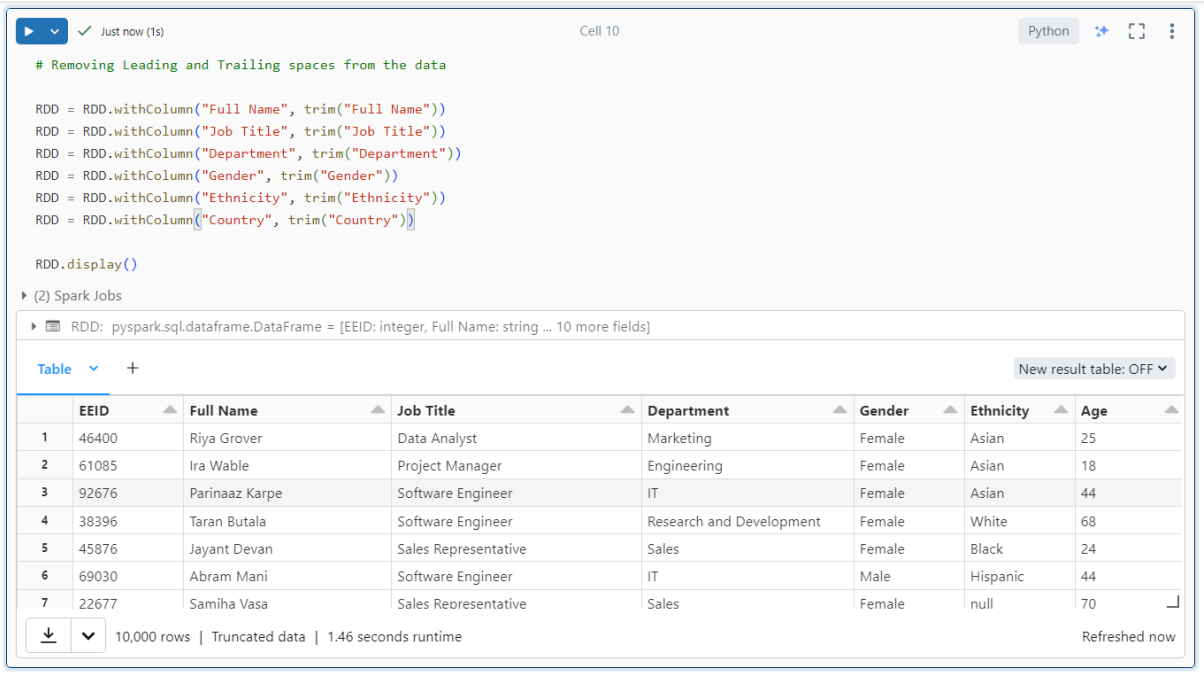
- Removing the Duplicate records



- Handling anonymous data



- Removing Extra spaces form the data



- Filling null values with proper messages and data

Just now (1s) Cell 12

```
RDD.display()
```

(2) Spark Jobs

Table + New result table: OFF

	EEID	Full Name	Job Title	Department	Gender	Ethnicity	Age
1	46400	Riya Grover	Data Analyst	Marketing	Female	Asian	25
2	61085	Ira Wable	Project Manager	Engineering	Female	Asian	18
3	92676	Parinaaz Karpe	Software Engineer	IT	Female	Asian	44
4	38396	Taran Butala	Software Engineer	Research and Development	Female	White	68
5	45876	Jayant Devan	Sales Representative	Sales	Female	Black	24
6	69030	Abram Mani	Software Engineer	IT	Male	Hispanic	44
7	22677	Samiha Vasa	Sales Representative	Sales	Female	Not Known	70

10,000 rows | Truncated data | 1.15 seconds runtime Refreshed now

Just now (6s) Cell 13 Python

```
# Filling the numerical column's null values with proper average values

window_spec = Window.partitionBy()
RDD = RDD.withColumn('Age',when(col('Age').isNull(),avg(col('Age')).over(window_spec)).otherwise(col('Age')))

average_salaries = RDD.groupBy("Country", "Department").avg("Annual Salary")
RDD = RDD.join(average_salaries, ["Country", "Department"], "left").withColumnRenamed("avg(Annual Salary)", "Average Salary")
RDD = RDD.withColumn('Annual Salary', when(col('Annual Salary').isNull(), col('Average Salary'))
                                        .otherwise(col('Annual Salary'))).drop("Average Salary")

RDD.display()
```

(8) Spark Jobs

average_salaries: pyspark.sql.dataframe.DataFrame = [Country: string, Department: string ... 1 more field]

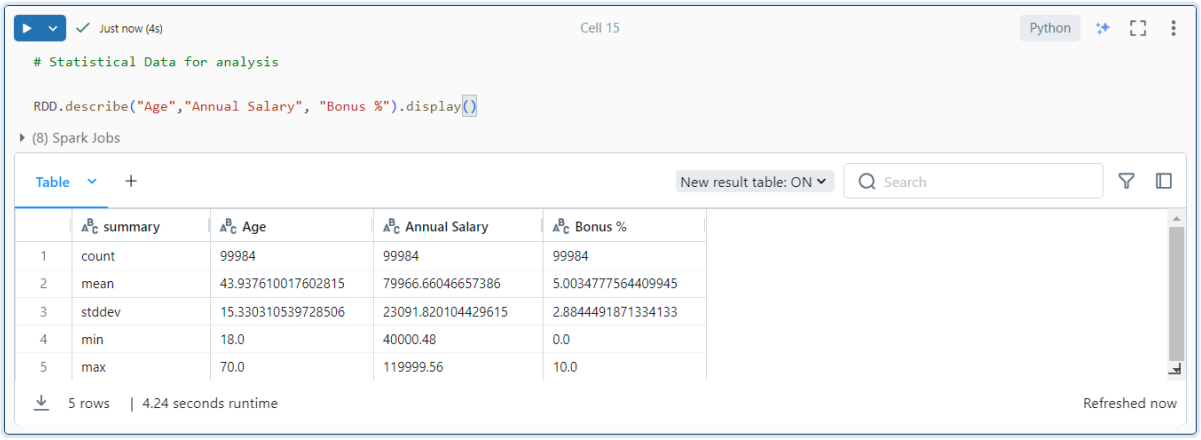
RDD: pyspark.sql.dataframe.DataFrame = [Country: string, Department: string ... 10 more fields]

Table + New result table: ON Search

	Country	Department	EEID	Full Name	Job Title	Gender	Ethnicity
1	Canada	Marketing	46400	Riya Grover	Data Analyst	Female	Asian
2	UK	Engineering	61085	Ira Wable	Project Manager	Female	Asian
3	USA	IT	92676	Parinaaz Karpe	Software Engineer	Female	Asian
4	Canada	Research and Development	38396	Taran Butala	Software Engineer	Female	White
5	Australia	Sales	45876	Jayant Devan	Sales Representative	Female	Black
6	Canada	IT	69030	Abram Mani	Software Engineer	Male	Hispanic
7	Canada	Sales	22677	Samiha Vasa	Sales Representative	Female	Not Known
8	USA	Customer Service	94951	Kabir Dey	Customer Service Representative	Prefer Not to say	Not Known

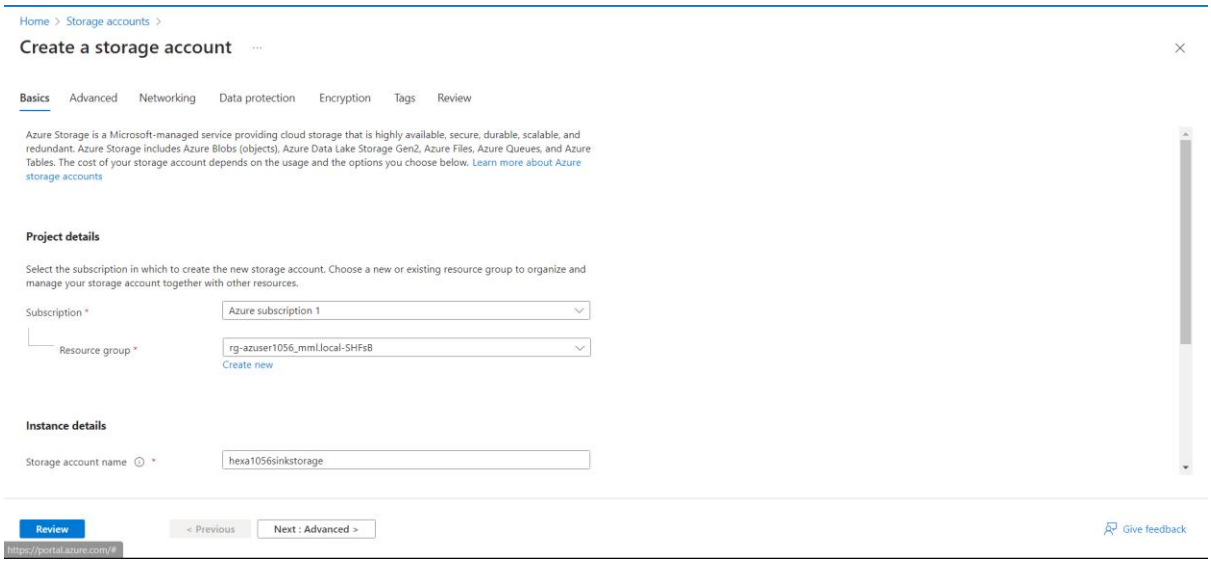
- Renaming USA to US to make dataset consistent

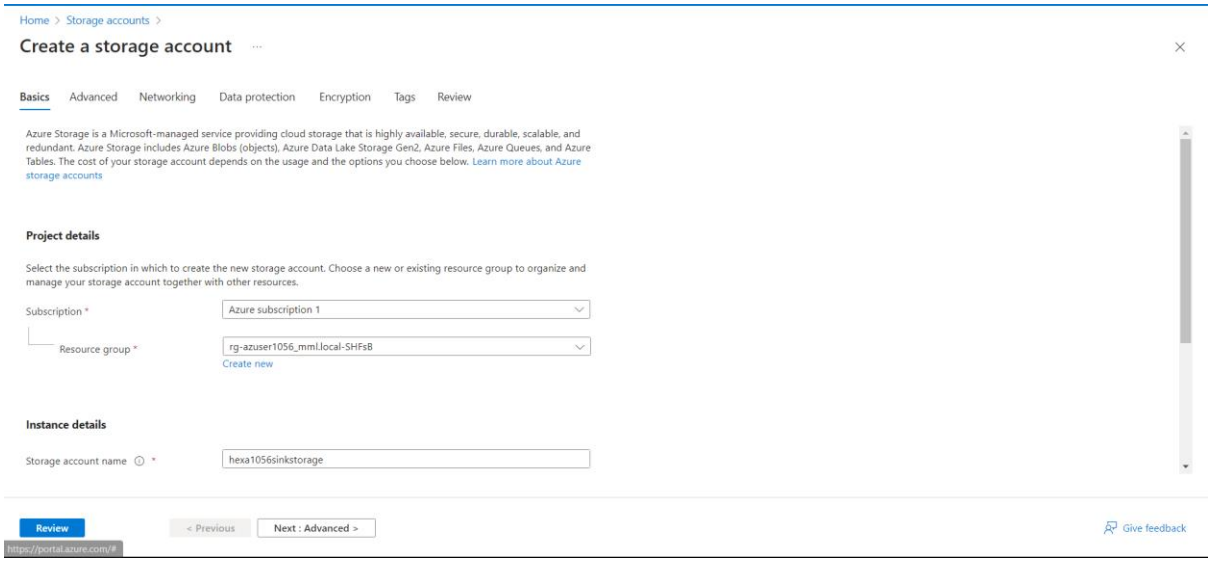
- Statistical data

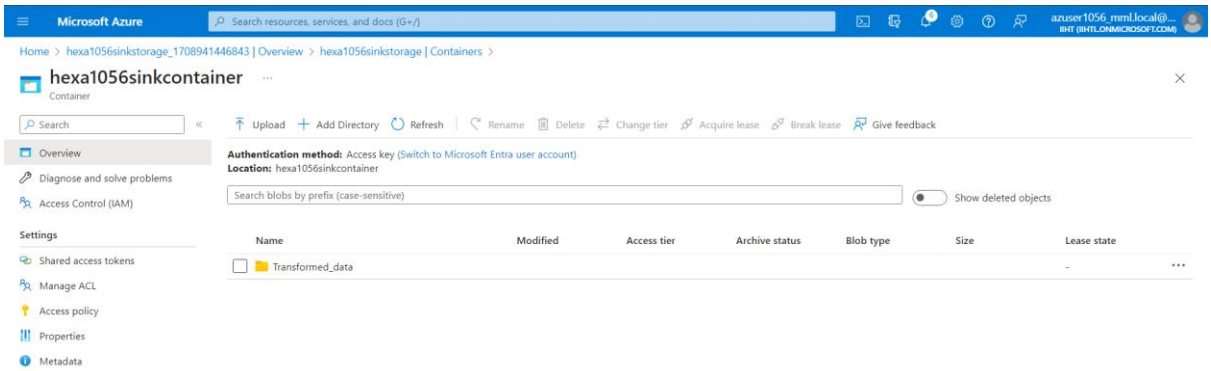


7. Loading Data into Sink Storage

- To store the transformed data we need to create sink storage (Azure Data Lake) and Container
- Connecting data source (ADLS) by mounting it to the Databricks notebook to load the data







Just now (11s) Cell 16

```
# 3) Loading Data in Azure Data Lake

# Mounting the sink storage(Azure Data Lake) with Azure databricks

dbutils.fs.mount(
    source = "wasbs://hexa1056sinkcontainer@hexa1056sinkstorage.blob.core.windows.net",
    mount_point="/mnt/blobStorage1",
    extra_configs={"fs.azure.account.key.hexa1056sinkstorage.blob.core.windows.net":
        "EWHBv0tr0q7pGTHfIliOtqk9iU+JTba373jdDdBQGOylx6Nj9wVmtoAjiHtjd5tvcUvfZpXDTm3u+AStZniWHw=="}
)
```

```
True
```

04:37 PM (<1s) Cell 17

```
dbutils.fs.ls("/mnt/blobStorage1")
```

```
[FileInfo(path='dbfs:/mnt/blobStorage1/Transformed_data/', name='Transformed_data/', size=0, modificationTime=0)]
```

04:37 PM (1s) Cell 18

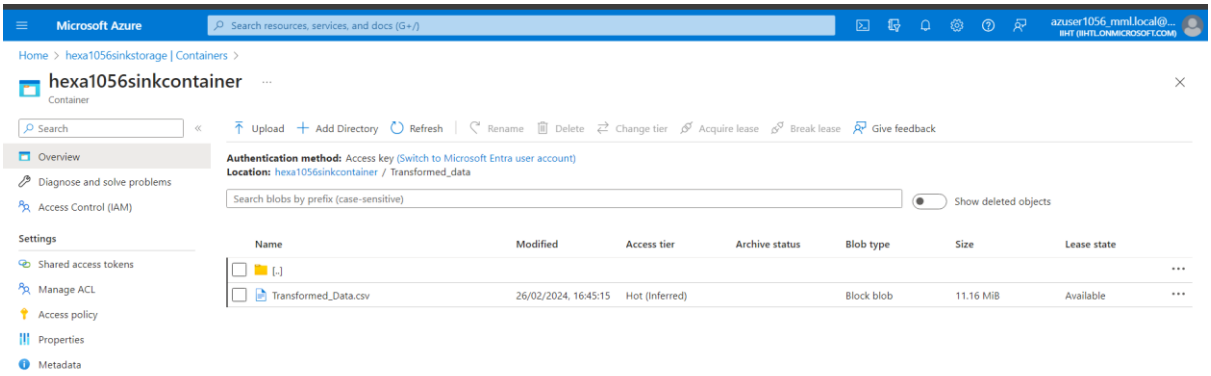
```
# Converting RDD to pandas dataframe

pandas_df=RDD.toPandas()
```

(1) Spark Jobs

5 minutes ago (1s) Cell 19 Python

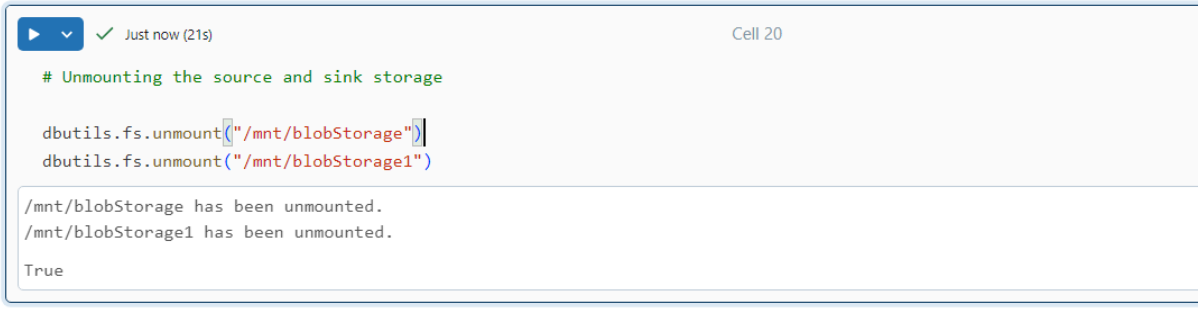
```
# Loading the transformed data in Azure Data Lake

pandas_df.to_csv('/dbfs/mnt/blobStorage1/Transformed_data/Transformed_Data.csv',index=False)
```

- Data Successfully Loaded

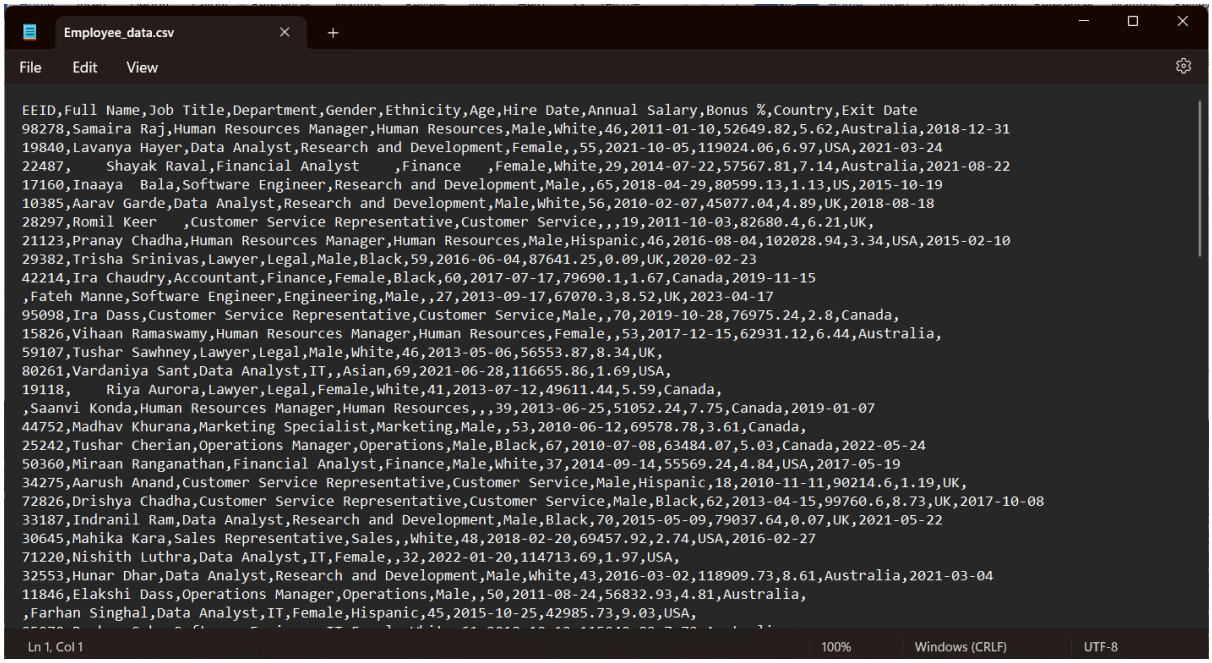


---

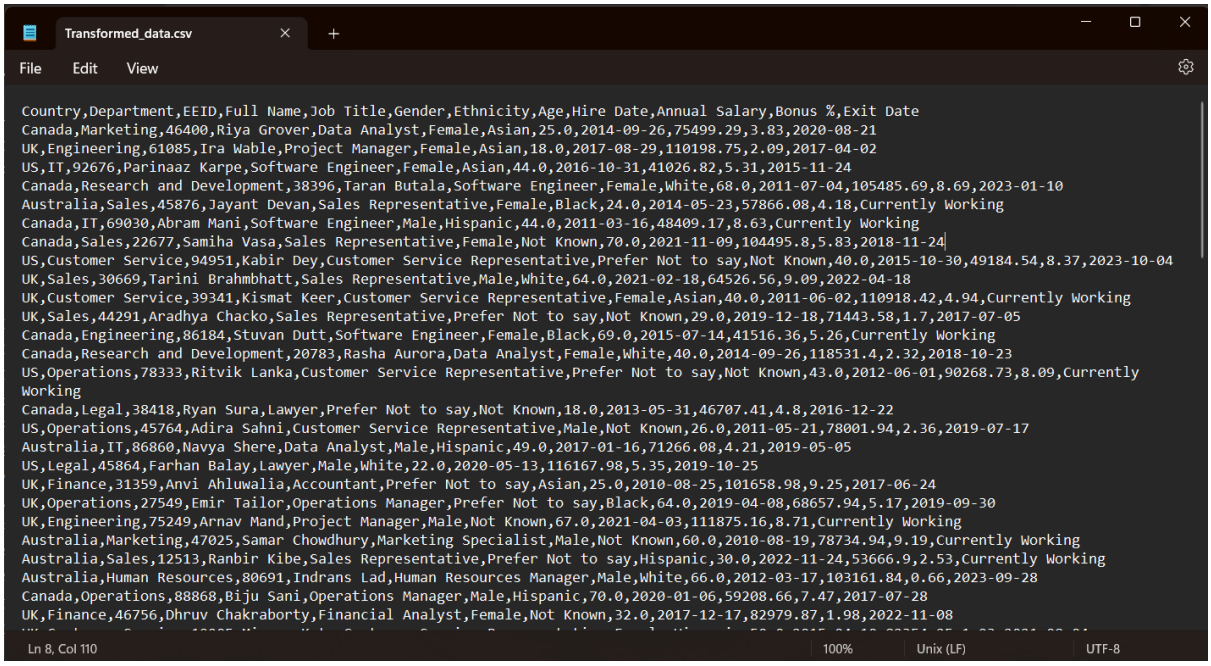
8. Unmounting the Source and Sink storage



Raw Data



Transformed Data



Conclusion

In conclusion, this project successfully demonstrated the utilization of Azure Databricks and PySpark for large-scale data processing on a sample CSV file containing employee data. Through the implementation of Extract, Transform, Load (ETL) operations, the dataset was cleaned, transformed, and prepared for further analysis or reporting. Overall, this project serves as a practical demonstration of how Azure Databricks and PySpark can be effectively utilized for large-scale data processing