**DATA BRICKS ASSIGNMENT**

**Name: S. Nikhil Sai**

**Date: 28-11-2024**

**🡪Writing Data to csv, json, parquet, delta**

#loading data

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Assignment").getOrCreate()

data = spark.read.table("hive\_metastore.default.export\_1\_csv")

data.createOrReplaceTempView("data")



**Explanation:** This code initializes a **SparkSession**, which is the entry point for executing Spark operations, and sets the application name to "Assignment". It then reads data from the Hive Metastore table **export\_1\_csv** into a PySpark DataFrame using the read.table() method. The DataFrame is subsequently registered as a temporary SQL view named **data**, allowing the dataset to be queried directly using SQL commands within the same session.

#data to csv

data.write.format('csv').saveAsTable("mycsv\_data", mode='overwrite')

data.display()

A screenshot of a computer

Description automatically generated

**Explanation:** This code writes the DataFrame data to a CSV-based table named "mycsv\_data". The write.format('csv') specifies the output format as CSV, and saveAsTable() saves the data as a managed table in the metastore. The mode='overwrite' ensures that if the table already exists, it will be replaced. Finally, data.display() renders the DataFrame as a table-like visualization for easy inspection in the Databricks notebook interface.

#data to json

data.write.format('json').saveAsTable("myjson\_data", mode='overwrite')

data.display()

A screenshot of a computer

Description automatically generated

**Explanation:** This code writes the DataFrame data to a JSON-based table named "myjson\_data". The write.format('json') specifies that the output data format is JSON, while saveAsTable() stores it as a managed table in the Hive Metastore. The mode='overwrite' ensures the table is replaced if it already exists. Finally, data.display() visually displays the DataFrame content within the Databricks notebook for review.

#data to parquet

data.write.format('parquet').saveAsTable("myparquet\_data", mode='overwrite')

data.display()

A screenshot of a computer

Description automatically generated

**Explanation:** This code writes the DataFrame **data** to a **Parquet-based table** named "myparquet\_data". The write.format('parquet') specifies Parquet as the storage format, which is efficient for both storage and querying. The **saveAsTable()** method saves the data as a managed table in the Hive Metastore, and **mode='overwrite'** replaces the table if it already exists. Finally, **data.display()** provides a visual representation of the DataFrame in the Databricks notebook.

#data to delta

data.write.format('delta').saveAsTable("mydelta\_data", mode='overwrite')

data.display()

A screenshot of a computer

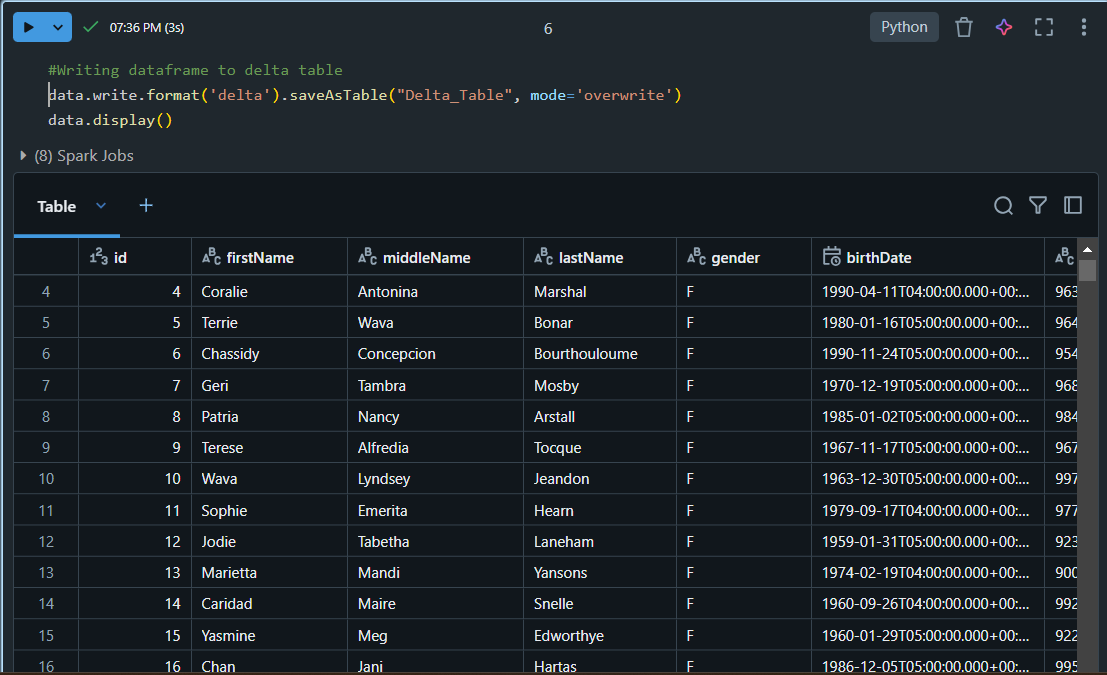
Description automatically generated

**Explanation:** This code saves the DataFrame data as a Delta table named "mydelta\_data". The write.format('delta') specifies the Delta format, which supports ACID transactions and efficient queries. The saveAsTable() method stores the data as a managed Delta table in the Hive Metastore, and mode='overwrite' ensures that any existing table with the same name is replaced. Finally, data.display() visualizes the DataFrame in the Databricks notebook for easy inspection.

**🡪** **Writing dataframe to delta table**

data.write.format('delta').saveAsTable("Delta\_Table", mode='overwrite')

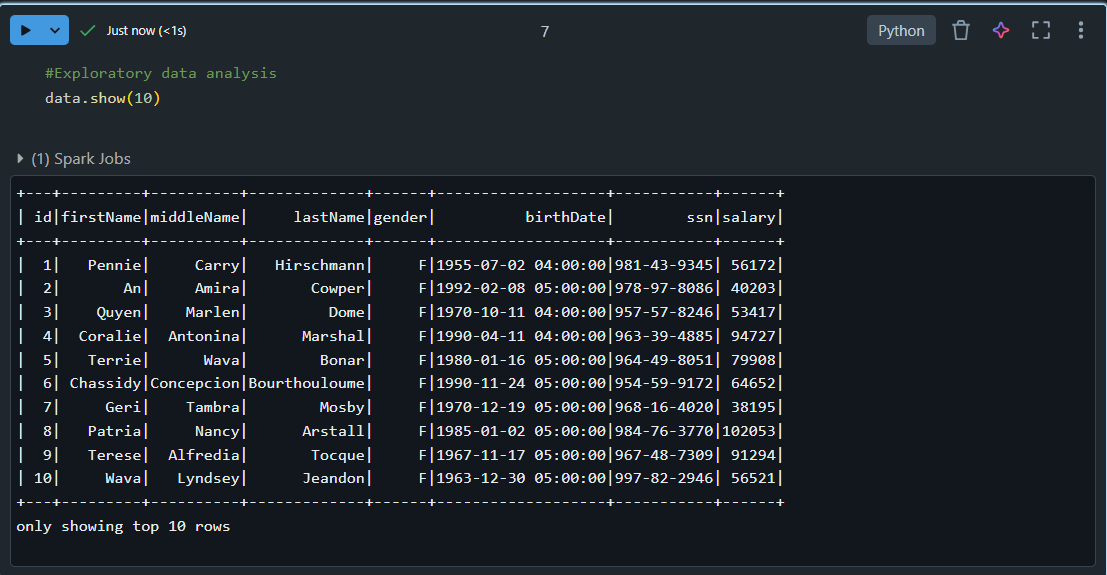
data.display()



**Explanation:** This code writes the DataFrame data into a Delta table named "Delta\_Table". The write.format('delta') specifies the use of the Delta format, which provides features like ACID compliance and versioning for reliable data processing. The saveAsTable() function stores it as a managed Delta table in the Hive Metastore, and mode='overwrite' ensures that any existing table with the same name is replaced. The data.display() command displays the contents of the DataFrame in the Databricks notebook interface for visualization.

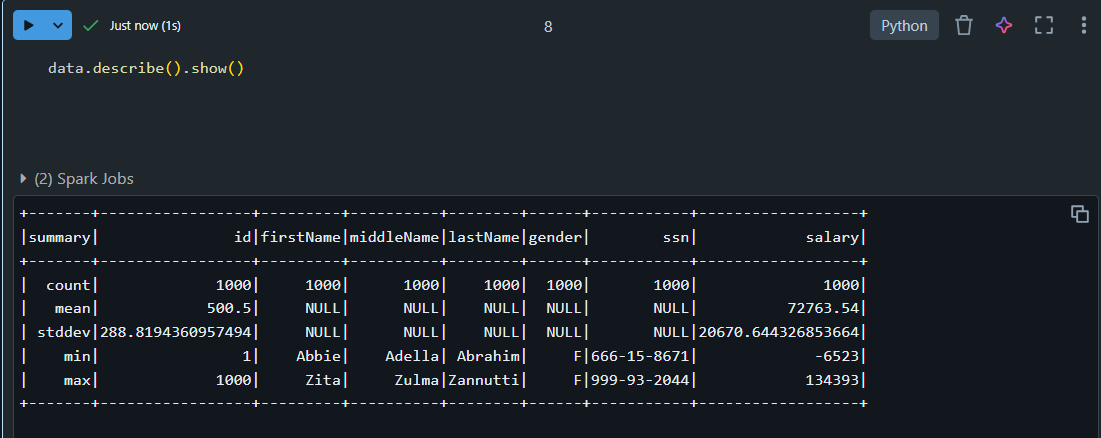
**🡪 Exploratory Data Analysis:**

data.show(10)



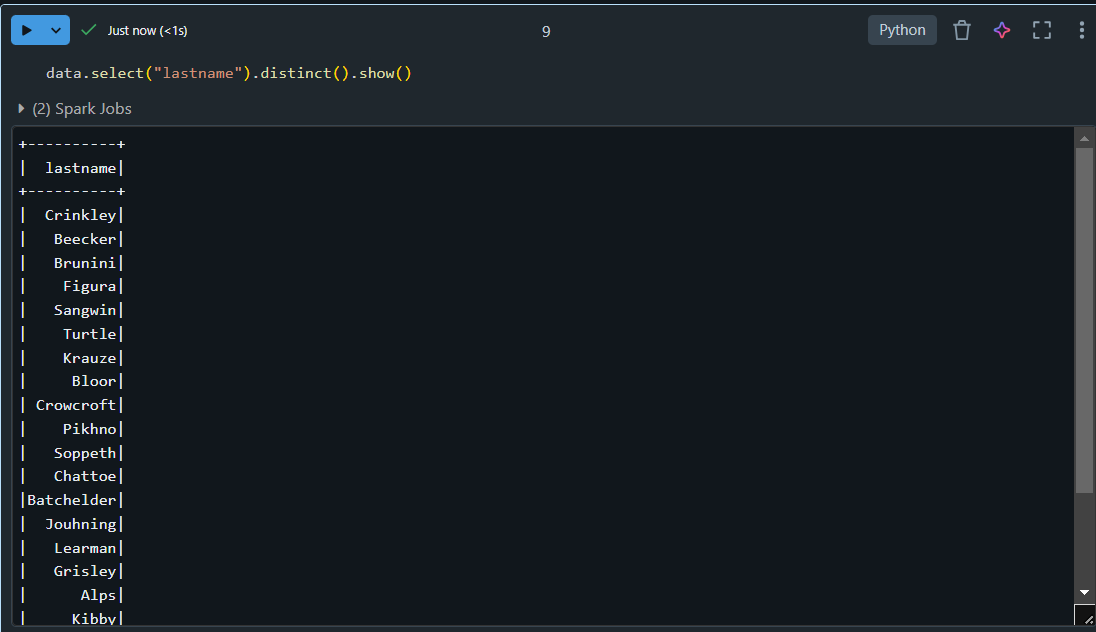
**Explanation:** This code displays the first **10 rows** of the DataFrame **data** in a tabular format. The **show(10)** method provides a quick way to preview a subset of the dataset, helping to inspect its structure and contents, including column values and types. If the dataset contains fewer than 10 rows, all rows will be shown.

data.describe().show()



**Explanation:** This code computes and displays **summary statistics** for the columns in the DataFrame **data**. The **describe()** method calculates metrics like **count**, **mean**, **standard deviation**, **minimum**, and **maximum** for numeric columns. The **show()** function outputs these statistics in a tabular format, providing a quick overview of the dataset's distribution and key statistics.

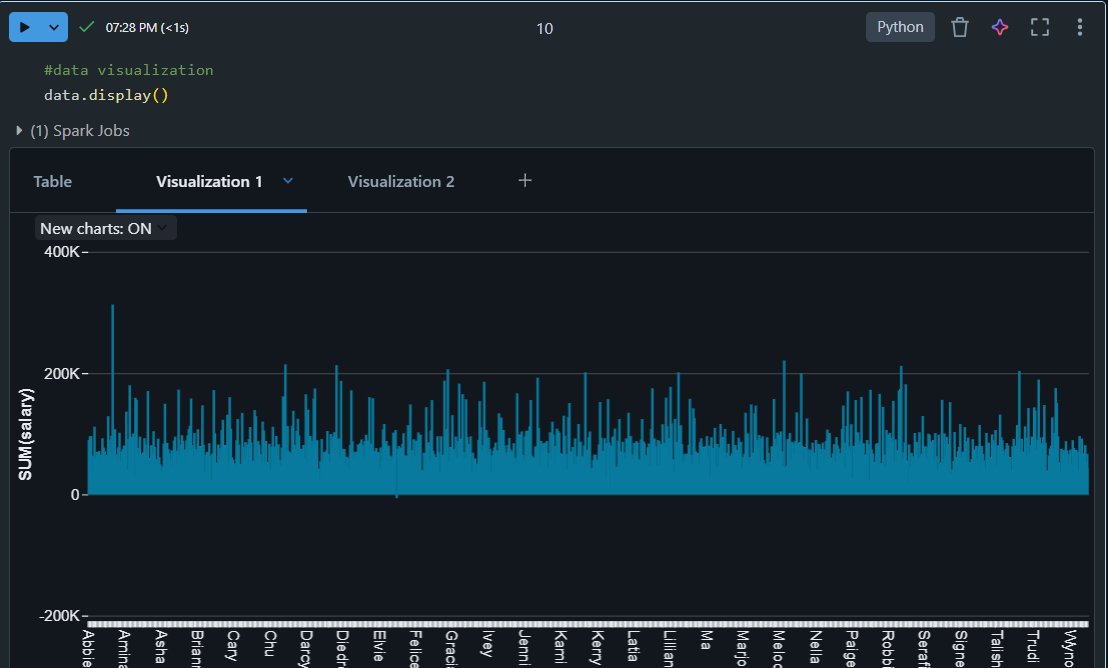
data.select("lastname").distinct().show()



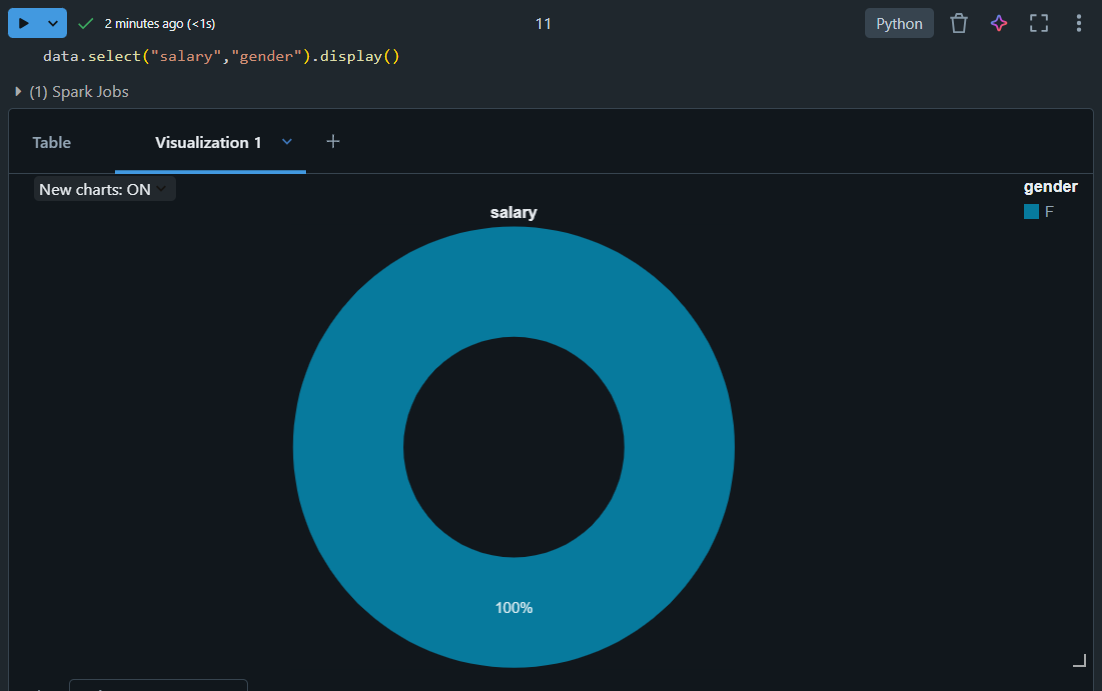
**Explanation:** This code selects the lastname column from the DataFrame data and retrieves only the unique values using the distinct() method. The show() function then displays these distinct values in a tabular format, helping to identify all the unique entries in the lastname column.

* **Data visualization Hands-on:**

data.display()

****

data.select("salary","gender").display()



**Explanation:** The first graph represents the salary of each individual. The second graph is represented in pie chart and it depicts the contribution of salaries of gender. Hence, All the contribution is made by female. So, the 100 percent piechart represents the salary of female.