

# Azure Databricks Assignment

Name: Podutur Lahari – DE126

Date: 28-11-2024

## Practice of Loading Data:-

### 1. Load and Display Loan Table Data

```
# data =spark.read.table("samples.nyctaxi.trips")
```

```
datatable
```

```
=spark.read.table("hive_metastore.default.loan")
```

```
datatable.display()
```

▶ (1) Spark Jobs

▶ datatable: pyspark.sql.dataframe.DataFrame = [Customer\_ID: string, Age: long ... 13 more fields]

Table ▾ + 🔍 🔍 🔍

	<sup>A</sup> <sub>C</sub> Customer_ID	<sup>1</sup> <sub>3</sub> Age	<sup>A</sup> <sub>C</sub> Gender	<sup>A</sup> <sub>C</sub> Occupation	<sup>A</sup> <sub>C</sub> Marital Status	<sup>1</sup> <sub>3</sub> Family Size	<sup>1</sup> <sub>3</sub> Income
1	IB14001	30	MALE	BANK MANAGER	SINGLE	4	50000
2	IB14008	44	MALE	PROFESSOR	MARRIED	6	51000
3	IB14012	30	FEMALE	DENTIST	SINGLE	3	58450
4	IB14018	29	MALE	TEACHER	MARRIED	5	45767
5	IB14022	34	MALE	POLICE	SINGLE	4	43521
6	IB14024	55	FEMALE	NURSE	MARRIED	6	34999
7	IB14025	39	FEMALE	TEACHER	MARRIED	6	46619
8	IB14027	51	MALE	SYSTEM MANAGER	MARRIED	3	49999
9	IB14029	24	FEMALE	TEACHER	SINGLE	3	45008
10	IB14031	37	FEMALE	SOFTWARE ENGINEER	MARRIED	5	55999
11	IB14032	24	MALE	DATA ANALYST	SINGLE	4	60111
12	IB14034	32	MALE	PRODUCT ENGINEER	MARRIED	6	null

### 2. Create RDDs and Load Delta Tables

```
# to create rdds and dataframe from pyspark
```

```
import SparkContext from pyspark.sql import
```

```
SparkSession # Initialize SparkContext and
```

```
SparkSession sc = SparkContext.getOrCreate()
```

```
spark = SparkSession.builder.appName('pyspark
```

```
first program').getOrCreate()
```

```
data = spark.read.format("delta").load("dbfs:/databricks-datasets/nyctaxi-
```

```
withzipcodes/subsampled") datatable =
```

```
spark.read.format("delta").load("dbfs:/user/hive/warehouse/loan")
```

```
data.display() datatable.display()
```

▶ (3) Spark Jobs

▶ data: pyspark.sql.dataframe.DataFrame = [tpep\_pickup\_datetime: timestamp, tpep\_dropoff\_datetime: timestamp ... 4 more fields]

▶ datatable: pyspark.sql.dataframe.DataFrame = [Customer\_ID: string, Age: long ... 13 more fields]

Table ▾ + 🔍 📄 📄

	🕒 tpep_pickup_datetime	🕒 tpep_dropoff_datetime	1.2 trip_distance	1.2 fare_amount	1 <sup>2</sup> <sub>3</sub> pickup_zip	1 <sup>2</sup> <sub>3</sub> dropof
1	2016-02-16T22:40:45.000+00:00	2016-02-16T22:59:25.000+00:00	5.35	18.5	10003	
2	2016-02-05T16:06:44.000+00:00	2016-02-05T16:26:03.000+00:00	6.5	21.5	10282	
3	2016-02-08T07:39:25.000+00:00	2016-02-08T07:44:14.000+00:00	0.9	5.5	10119	
4	2016-02-29T22:25:33.000+00:00	2016-02-29T22:38:09.000+00:00	3.5	13.5	10001	
5	2016-02-03T17:21:02.000+00:00	2016-02-03T17:23:24.000+00:00	0.3	3.5	10028	
6	2016-02-10T00:47:44.000+00:00	2016-02-10T00:53:04.000+00:00	0	5	10038	
7	2016-02-19T03:24:25.000+00:00	2016-02-19T03:44:56.000+00:00	6.57	21.5	10001	
8	2016-02-02T14:05:23.000+00:00	2016-02-02T14:23:07.000+00:00	1.08	11.5	10103	
9	2016-02-20T15:42:20.000+00:00	2016-02-20T15:50:40.000+00:00	0.8	7	10003	

Table ▾ + 🔍 📄 📄

	🔑 Customer_ID	1 <sup>2</sup> <sub>3</sub> Age	🔑 Gender	🔑 Occupation	🔑 Marital Status	1 <sup>2</sup> <sub>3</sub> Family Size	1 <sup>2</sup> <sub>3</sub> Income
1	IB14001	30	MALE	BANK MANAGER	SINGLE	4	50000
2	IB14008	44	MALE	PROFESSOR	MARRIED	6	51000
3	IB14012	30	FEMALE	DENTIST	SINGLE	3	58450
4	IB14018	29	MALE	TEACHER	MARRIED	5	45767
5	IB14022	34	MALE	POLICE	SINGLE	4	43521
6	IB14024	55	FEMALE	NURSE	MARRIED	6	34999
7	IB14025	39	FEMALE	TEACHER	MARRIED	6	46619
8	IB14027	51	MALE	SYSTEM MANAGER	MARRIED	3	49999

## Summary of Loading Data: -

In the first code block, I used PySpark to create a Spark session, which is essential for processing data in Databricks. I then loaded the loan data stored in a Delta format table from the Databricks

File System (DBFS) into a DataFrame using `spark.read.format("delta")`. Delta format offers several advantages such as ACID transactions and time travel, making it a reliable choice for working with large datasets in Databricks. After loading the data, I displayed it to visually inspect the information, which allows me to quickly understand the structure of the dataset.

In the second code block, I accessed two tables from the Databricks metastore using `spark.table()`. This method allows me to easily query tables that have already been registered in the metastore, which is a centralized place to manage metadata for structured data. The first table, `loan_table`, was loaded from the default schema (`hive_metastore.default`), while the second table, `trips_table`, came from the `samples.nyctaxi` schema. By displaying both tables, I can examine the content and start analyzing them for insights. These two tables represent two different kinds of data: financial data in the `loan_table` and transportation data in the `trips_table`.

This entire process showcases the simplicity and flexibility of working with various data formats (like Delta) and managing data in Databricks using PySpark, which is a powerful tool for big data analysis. With this setup, I can perform various analyses, transformations, and queries on the data to derive meaningful insights.

## **Practice on Delta Tables: -**

### **1. Loading and Displaying Data from Delta Table in Azure**

```
spark.table("default.export")
```

```
data = spark.read.format("delta").load("dbfs:/user/hive/warehouse/export")
```

```
data.show()
```

```

data: pyspark.sql.dataframe.DataFrame = [id: long, firstName: string ... 6 more fields]
+---+-----+-----+-----+-----+-----+-----+
| 3|   Quyen|   Marlen|   Dome|   F|1970-10-11 04:00:00|957-57-8246| 53417|
| 4|  Coralie| Antonina|  Marshal|   F|1990-04-11 04:00:00|963-39-4885| 94727|
| 5|   Terrie|   Wava|   Bonar|   F|1980-01-16 05:00:00|964-49-8051| 79908|
| 6| Chassidy|Concepcion|Bourthouloume|   F|1990-11-24 05:00:00|954-59-9172| 64652|
| 7|   Geri|   Tamba|   Mosby|   F|1970-12-19 05:00:00|968-16-4020| 38195|
| 8|   Patria|   Nancy|  Arstall|   F|1985-01-02 05:00:00|984-76-3770|102053|
| 9|   Terese| Alfredia|   Tocque|   F|1967-11-17 05:00:00|967-48-7309| 91294|
|10|   Wava|  Lyndsey|  Jeandon|   F|1963-12-30 05:00:00|997-82-2946| 56521|
|11|  Sophie|  Emerita|   Hearn|   F|1979-09-17 04:00:00|977-66-4483| 90920|
|12|   Jodie|  Tabetha|  Laneham|   F|1959-01-31 05:00:00|923-24-9769| 90634|
|13| Marietta|   Mandi|  Yansons|   F|1974-02-19 04:00:00|900-34-8083| 93162|
|14|  Caridad|   Maire|   Snelle|   F|1960-09-26 04:00:00|992-11-7062| 38859|
|15|  Yasmine|   Meg|  Edworthy|   F|1960-01-29 05:00:00|922-12-9862| 76220|
|16|   Chan|   Jani|   Hartas|   F|1986-12-05 05:00:00|995-51-3115| 75050|
|17|Evangeline|  Wanetta|  Casserley|   F|1961-09-29 04:00:00|926-61-3526| 62814|
|18|   Elnora|   Kecia|   Lipman|   F|1980-02-14 05:00:00|950-23-9739| 71350|
|19|   Adelle|  Kathryn|  Grigoriev|   F|1978-11-14 05:00:00|923-23-5984| 60600|
|20|   Mica|   Zandra|  Challens|   F|1973-11-24 05:00:00|918-66-1232| 51071|
+---+-----+-----+-----+-----+-----+-----+
only showing top 20 rows

```

## 2. Writing DataFrame to Delta Tables in Azure Databricks

```
from pyspark.sql import SparkSession spark =
```

```
SparkSession.builder.appName('Delta Table Write').getOrCreate() data =
```

```
spark.createDataFrame([ (1, "Alice", 1000), (2, "Bob", 2000),
```

```
(3, "Charlie", 3000) ],["id", "name", "salary"])
```

```
# Write the DataFrame as Delta tables
```

```
data.write.format('delta').saveAsTable("mydata_delta", mode="overwrite")
```

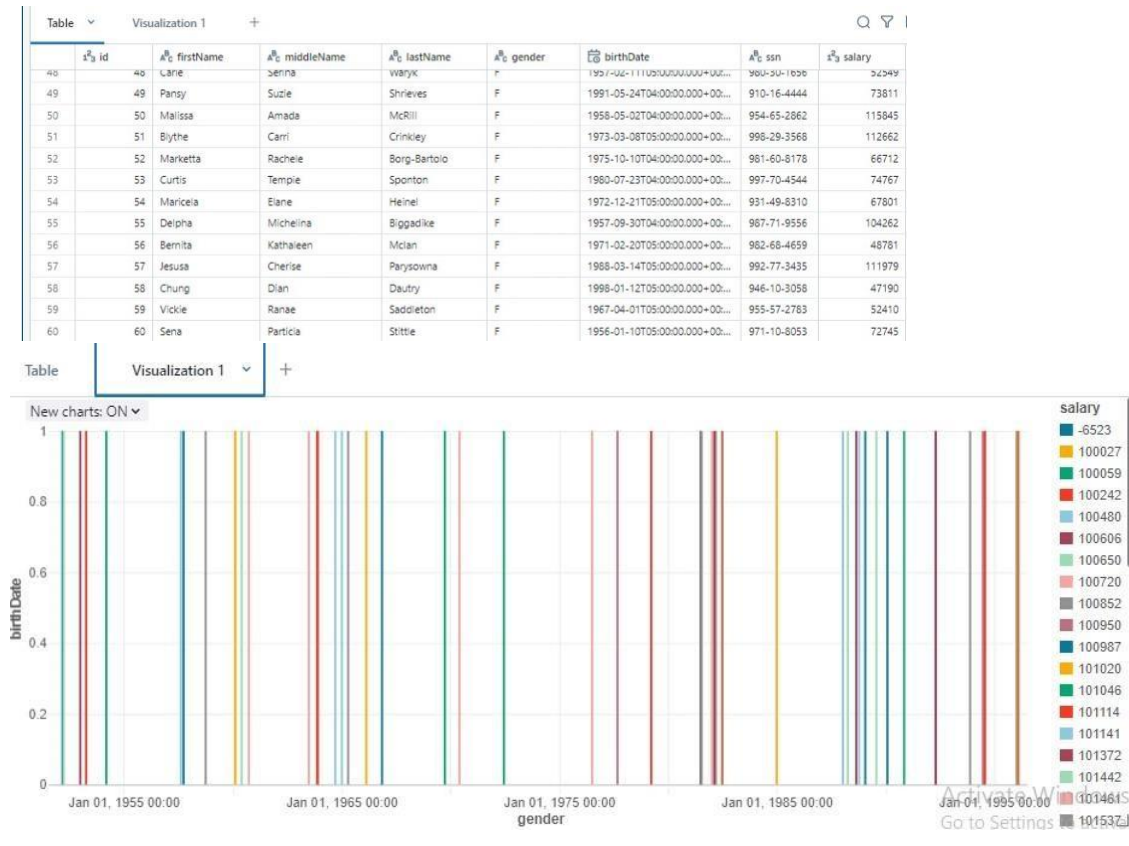
```
data.write.format('delta').saveAsTable("mydata")
```

### 3. Loading and Displaying Data from Delta Table in Databricks

```
spark.table("default.export") data =
```

```
spark.read.format("delta").load("dbfs:/user/hive/warehouse/export")
```

```
data.display()
```



### Summary on Delta Tables:-

In Azure Databricks, Delta tables are used to store data in a structured format that supports efficient querying and data management. The first block of code shows how to read data from an existing Delta table stored in the Databricks metastore or from a path in the DBFS (Databricks File System). Using the `spark.read.format("delta")` method, we load the data from the Delta table into a `DataFrame` and display it. This process allows us to view the content of the Delta table, which is stored in a structured format for analysis.

Next, we see how to create and write a new Delta table from a `DataFrame`. The `data.write.format("delta").saveAsTable("mydata")` command writes the data into a new Delta

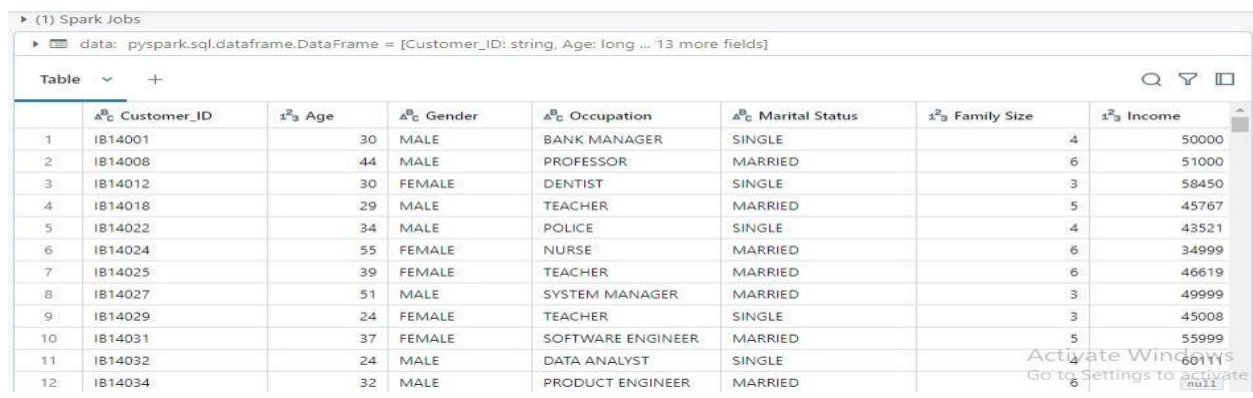
table named "mydata". This code demonstrates the ability to save a DataFrame into a Delta table, making it accessible for future queries and operations. We can specify the mode (like overwrite) to control how existing data is handled when writing the new data.

Finally, the `data.display()` method is used to show the contents of the newly written Delta table in a Databricks notebook. Overall, Delta tables provide a powerful and efficient way to store and manage data in Databricks, with built-in support for ACID transactions, versioning, and schema enforcement. This makes them ideal for data analysis and machine learning tasks where data integrity and fast querying are crucial.

## Practice EDA Analysis: -

### 1. Reading and Displaying Data from the Loan Table in Databricks

```
data = spark.read.table("hive_metastore.default.loan") display(data)
```



The screenshot shows a Databricks notebook interface. At the top, a Spark job is listed with the command: `data: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: long ... 13 more fields]`. Below this, a table view is displayed with the following columns: Customer\_ID, Age, Gender, Occupation, Marital Status, Family Size, and Income. The table contains 12 rows of data.

	Customer_ID	Age	Gender	Occupation	Marital Status	Family Size	Income
1	IB14001	30	MALE	BANK MANAGER	SINGLE	4	50000
2	IB14008	44	MALE	PROFESSOR	MARRIED	6	51000
3	IB14012	30	FEMALE	DENTIST	SINGLE	3	58450
4	IB14018	29	MALE	TEACHER	MARRIED	5	45767
5	IB14022	34	MALE	POLICE	SINGLE	4	43521
6	IB14024	55	FEMALE	NURSE	MARRIED	6	34999
7	IB14025	39	FEMALE	TEACHER	MARRIED	6	46619
8	IB14027	51	MALE	SYSTEM MANAGER	MARRIED	3	49999
9	IB14029	24	FEMALE	TEACHER	SINGLE	3	45008
10	IB14031	37	FEMALE	SOFTWARE ENGINEER	MARRIED	5	55999
11	IB14032	24	MALE	DATA ANALYST	SINGLE	4	60111
12	IB14034	32	MALE	PRODUCT ENGINEER	MARRIED	6	null

### 2. Getting Row Count and Schema Information of the Data

```
# Total row count data.count() # Schema information data.printSchema()
```



## ► (2) Spark Jobs

```
root
|-- Customer_ID: string (nullable = true)
|-- Age: long (nullable = true)
|-- Gender: string (nullable = true)
|-- Occupation: string (nullable = true)
|-- Marital Status: string (nullable = true)
|-- Family Size: long (nullable = true)
|-- Income: long (nullable = true)
|-- Expenditure: long (nullable = true)
|-- Use Frequency: long (nullable = true)
|-- Loan Category: string (nullable = true)
|-- Loan Amount: string (nullable = true)
|-- Overdue: long (nullable = true)
|-- Debt Record: string (nullable = true)
|-- Returned Cheque: long (nullable = true)
|-- Dishonour of Bill: long (nullable = true)
```

---

### 3. Displaying Summary Statistics for 'Income' Column

# Summary statistics for 'Income'

```
data.describe(['Income']).show()
```

## ► (2) Spark Jobs

```
+-----+-----+
|summary|      Income|
+-----+-----+
|  count|          468|
|   mean|68339.49145299145|
| stddev|86796.49367750238|
|    min|          28366|
|    max|          930000|
+-----+-----+
```

---

### 4. Counting Rows Grouped by Gender

```
data.groupBy('gender').count().show()
```

▶ (2) Spark Jobs

gender	count
MALE	280
FEMALE	220

## 5. Displaying Top 5 Highest Incomes

# Top 5 Highest Incomes `data.orderBy(data.Income.desc()).limit(5).show()`

▶ (1) Spark Jobs

Customer_ID	Age	Gender	Occupation	Marital Status	Family Size	Income	Expenditure	Use Frequency	Loan Category	Loan Amount
Overdue	Debt Record	Returned Cheque	Dishonour of Bill							
IBI4157	35	MALE	BANK MANAGER	MARRIED	4	930000	35680	6	HOUSING	6,79,040
5	34,000	5		5						
IBI4107	44	FEMALE	ACCOUNT MANAGER	MARRIED	4	800000	15632	8	AUTOMOBILE	23,65,478
5	20,145	3		4						
IBI4163	44	FEMALE	ACCOUNT MANAGER	MARRIED	4	800000	15632	8	COMPUTER SOFTWARES	23,65,478
5	20,145	3		4						
IBI4256	44	FEMALE	ACCOUNT MANAGER	MARRIED	4	800000	15632	8	COMPUTER SOFTWARES	23,65,478
5	20,145	3		4						
IBI4128	46	FEMALE	CLERK	MARRIED	3	750000	25641	5	GOLD LOAN	2,14,569
4	16,324	3		4						

## 6. Grouping Employees by Salary Buckets and Counting

# Salary Distribution from `pyspark.sql.functions import ceil, col`

# Add salary buckets `data_with_buckets = data.withColumn('salary_bucket',`

`ceil(col('Income') / 20000) * 20000)`

# Count employees in each bucket

`data_with_buckets.groupBy('salary_bucket').count().orderBy('salary_bucket').show()`



▶ (2) Spark Jobs

▶ data\_with\_buckets: pyspark.sql.dataframe.DataFrame = [Customer\_ID: string, Age: long ... 14 more fields]

salary_bucket	count
NULL	32
40000	70
60000	200
80000	136
100000	55
440000	1
700000	1
760000	1
800000	3
940000	1

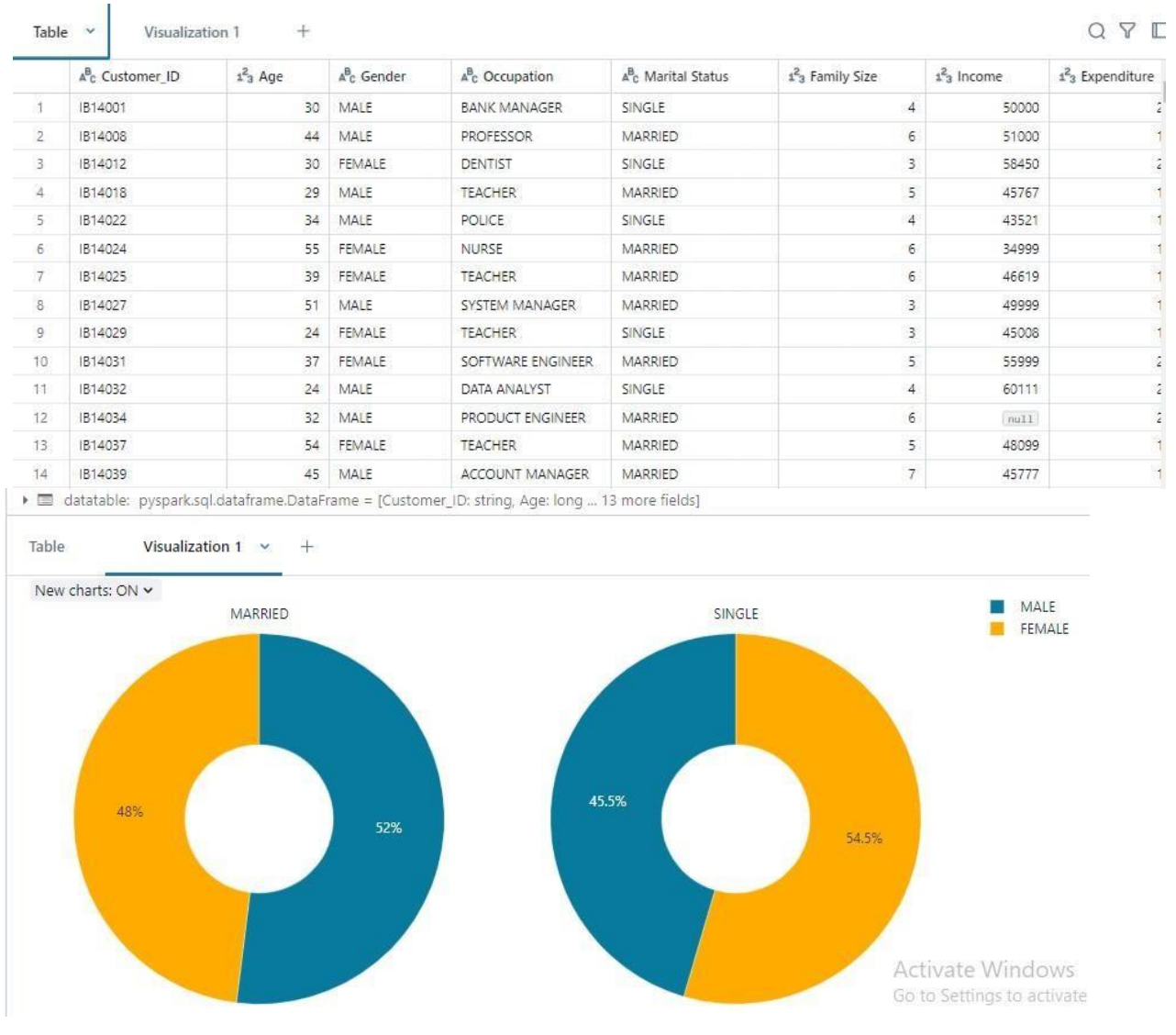
## Summary of EDA Analysis: -

I worked on a dataset from the `hive_metastore.default.loan` table using PySpark in Databricks. First, I loaded the data into a Spark DataFrame and displayed it to get a view of the records. I calculated the total number of rows in the dataset with the `count()` function, which shows how many entries there are. Then, I examined the schema of the data to understand the structure of the table, such as the column names and data types.

I also performed summary statistics for the Income column, which gave me basic measures like the count, mean, and standard deviation. I grouped the data by gender to count how many records fall into each gender category. To further explore the data, I identified the top 5 highest incomes by sorting the data in descending order based on the Income column. Finally, I created salary buckets by dividing the Income into ranges and counted how many employees fall into each bucket, helping me understand the distribution of income within the dataset.

## Practice on Visualization: -

### 1. Loading and Displaying Data from the 'loan' Table

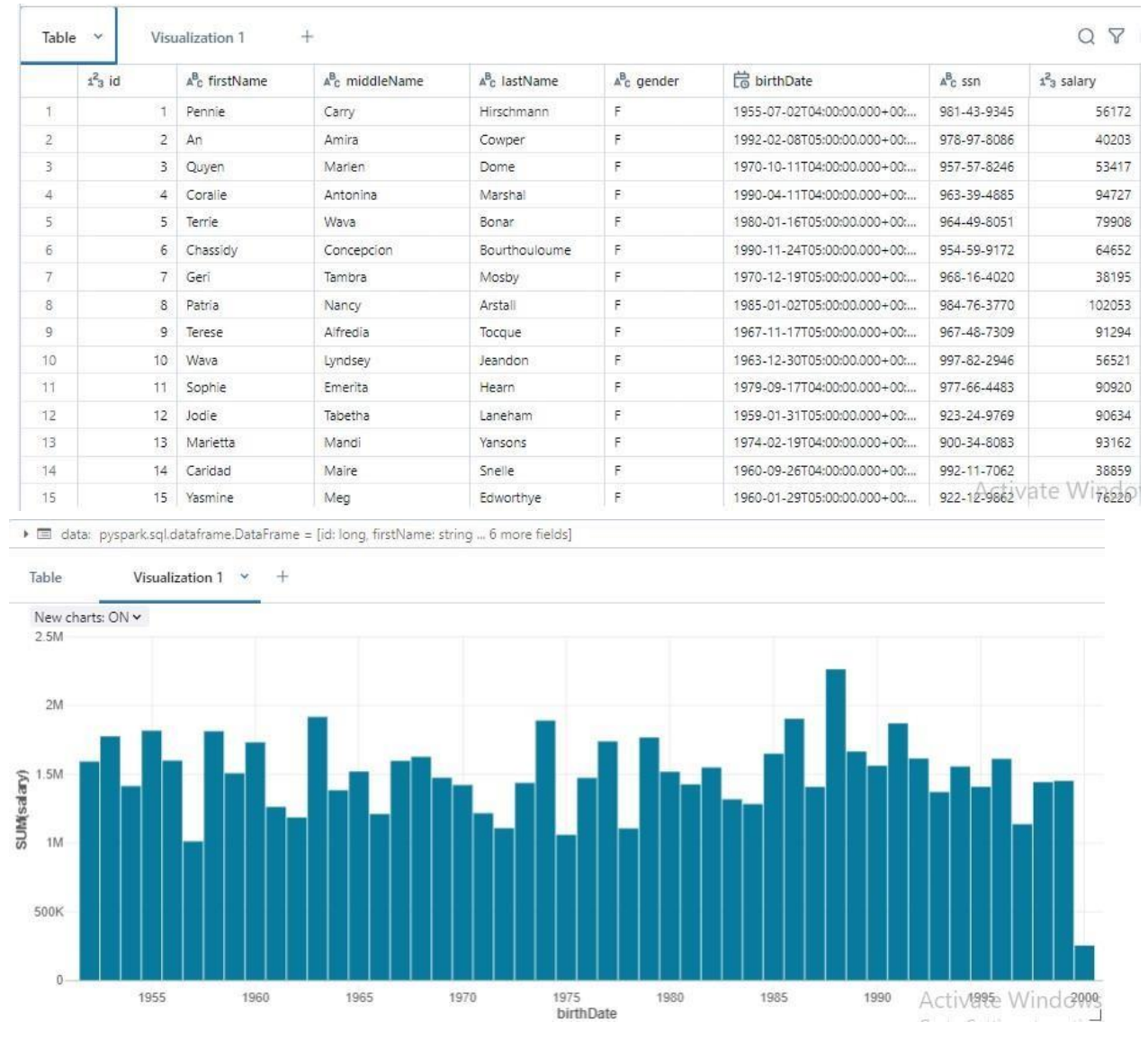


## 2. Loading and Displaying Data from 'export' Table and Delta Location

```
spark.table("default.export") data =
```

```
spark.read.format("delta").load("dbfs:/user/hive/warehouse/export")
```

```
data.display()
```



## **Summary on Visualization: -**

In Azure Databricks, data visualization using PySpark can help you easily interpret and present data insights. PySpark allows you to work with large datasets and perform complex transformations before visualizing the results. The `display()` function in Databricks provides a powerful way to visualize DataFrames directly in the notebook interface. When you load data into PySpark, whether from a table or a Delta file, you can quickly visualize it using Databricks' built-in visualization tools.

Visualizations like bar charts, line graphs, and scatter plots can be created with just a few clicks, providing an intuitive way to explore data patterns. You can create custom visualizations to examine trends over time, compare categories, or understand distributions. Databricks also supports interactive visualization, which means you can drill down into the data, filter values, and adjust axes for better clarity.

By using PySpark for data processing and Databricks for visualization, you can enhance your data exploration experience without switching tools or environments. This integration makes it easier to share insights with others and perform interactive analysis in real-time. Visualizations are crucial for communicating data findings in an understandable and impactful way.