



Ejercicio de Laboratorio 4: Databricks

Instituto Politécnico Nacional.
Escuela Superior de Cómputo.
Licenciatura en ciencia de datos.
Bases de Datos Avanzadas

Emiliano López Méndez.

Introduccion

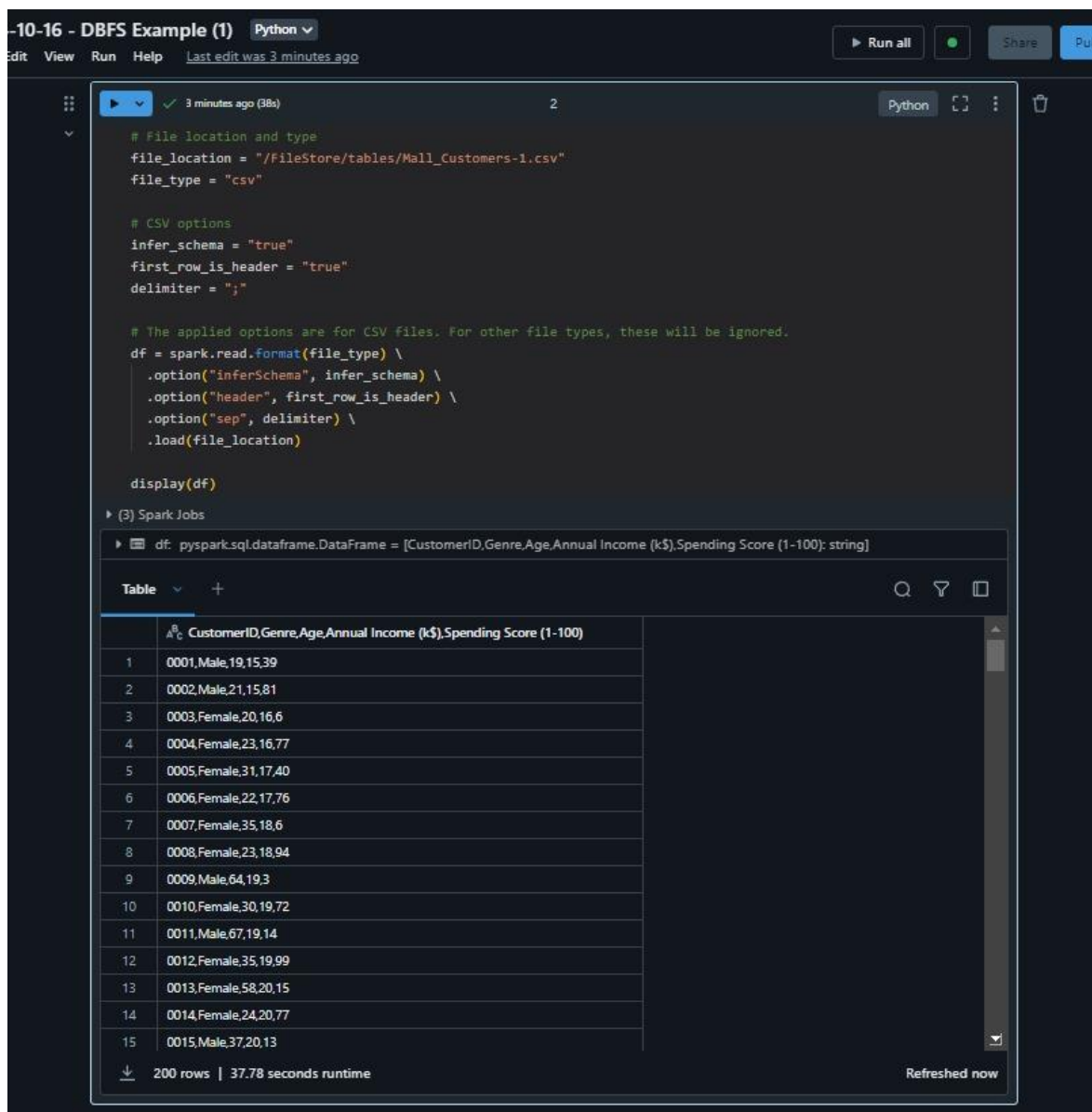
En este laboratorio se explorarán las capacidades de Databricks, una plataforma basada en la nube diseñada para procesar grandes volúmenes de datos utilizando Apache Spark. La actividad consiste en cargar un conjunto de datos, ejecutar consultas tanto en PySpark como en SQL, y almacenar los resultados en formato Parquet. A través de este ejercicio, se pretende reforzar las habilidades en el uso de herramientas de análisis avanzado de datos y comprender cómo utilizar vistas temporales y tablas en un entorno distribuido. Además, se practicará el manejo eficiente de datos estructurados, esencial para proyectos de Big Data.

Desarrollo de la Actividad:

El desarrollo de este laboratorio se estructura en varias etapas para garantizar un aprendizaje progresivo en el uso de Databricks y Apache Spark. En la primera parte, se carga el dataset `Mall_customers.csv` en el entorno de Databricks, permitiendo explorar su integración con un espacio de almacenamiento en la nube. Posteriormente, se crea una vista temporal con PySpark, lo que habilita la ejecución de consultas mediante la función `spark.sql`. En la siguiente fase, se aprovecha el modo SQL del notebook para realizar consultas adicionales y generar tablas temporales. Finalmente, se almacena el resultado de las consultas en archivos Parquet, un formato eficiente para datos analíticos, consolidando así la comprensión del manejo de datos en un entorno distribuido.

Parte 1 Carga de archivo

En esta sección se procede a cargar el dataset Mall_customers.csv en el entorno de Databricks. La carga del archivo es fundamental para que los datos estén disponibles en el espacio de almacenamiento de la plataforma y puedan ser utilizados en consultas posteriores. A través de este paso, se garantiza que el archivo se integre correctamente y sea accesible mediante PySpark, facilitando así el análisis de la información en un entorno de Big Data.



The screenshot shows a Databricks notebook titled "-10-16 - DBFS Example (1)" in Python. The code defines file location and type, CSV options, and uses Spark's read.format method to load the data into a DataFrame. The DataFrame is then displayed, showing a table with 15 rows of customer data.

```
# File location and type
file_location = "/FileStore/tables/Mall_Customers-1.csv"
file_type = "csv"

# CSV options
infer_schema = "true"
first_row_is_header = "true"
delimiter = ";"

# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
    .option("inferSchema", infer_schema) \
    .option("header", first_row_is_header) \
    .option("sep", delimiter) \
    .load(file_location)

display(df)
```

df: pyspark.sql.dataframe.DataFrame = [CustomerID,Genre,Age,Annual Income (k\$),Spending Score (1-100): string]

	CustomerID,Genre,Age,Annual Income (k\$),Spending Score (1-100)
1	0001,Male,19,15,39
2	0002,Male,21,15,81
3	0003,Female,20,16,6
4	0004,Female,23,16,77
5	0005,Female,31,17,40
6	0006,Female,22,17,76
7	0007,Female,35,18,6
8	0008,Female,23,18,94
9	0009,Male,64,19,3
10	0010,Female,30,19,72
11	0011,Male,67,19,14
12	0012,Female,35,19,99
13	0013,Female,58,20,15
14	0014,Female,24,20,77
15	0015,Male,37,20,13

200 rows | 37.78 seconds runtime Refreshed now

```
# Importar librerías necesarias
from pyspark.sql import SparkSession

# Crear una sesión de Spark
spark = SparkSession.builder.appName("MallCustomers").getOrCreate()

# Cargar el dataset desde el almacenamiento
df = spark.read.option("header", "true").csv("/FileStore/tables/Mall_Customers-1.csv")

# Crear una vista temporal
df.createOrReplaceTempView("customers_view")
```

▶ (1) Spark Jobs

df: pyspark.sql.dataframe.DataFrame

- CustomerID: string
- Genre: string
- Age: string
- Annual Income (k\$): string
- Spending Score (1-100): string

Parte 2 Consultas con spark.sql

En esta etapa, se realizan consultas sobre los datos utilizando PySpark mediante la función `spark.sql`. Esta función permite ejecutar instrucciones SQL sobre las vistas temporales generadas, proporcionando un entorno flexible y eficiente para el análisis de datos. La ejecución de estas consultas permite obtener conocimientos específicos del dataset, como estadísticas descriptivas y segmentación de los clientes. De este modo, se pone en práctica la capacidad de trabajar con datos estructurados mediante Spark en Databricks.

```
# Consultar la cantidad de clientes por género
spark.sql("SELECT Genre, COUNT(*) AS Total FROM customers_view GROUP BY Genre").show()
```

▶ (2) Spark Jobs

Genre	Total
Female	112
Male	88

Just now (2s)

5

Python

```
# Consultar el gasto promedio por género
spark.sql("SELECT Genre, AVG(`Spending Score (1-100)`) AS Avg_Spending FROM customers_view GROUP BY Genre").show()
```

▶ (2) Spark Jobs

Genre	Avg_Spending
Female	51.526785714285715
Male	48.51136363636363

Just now (1s)

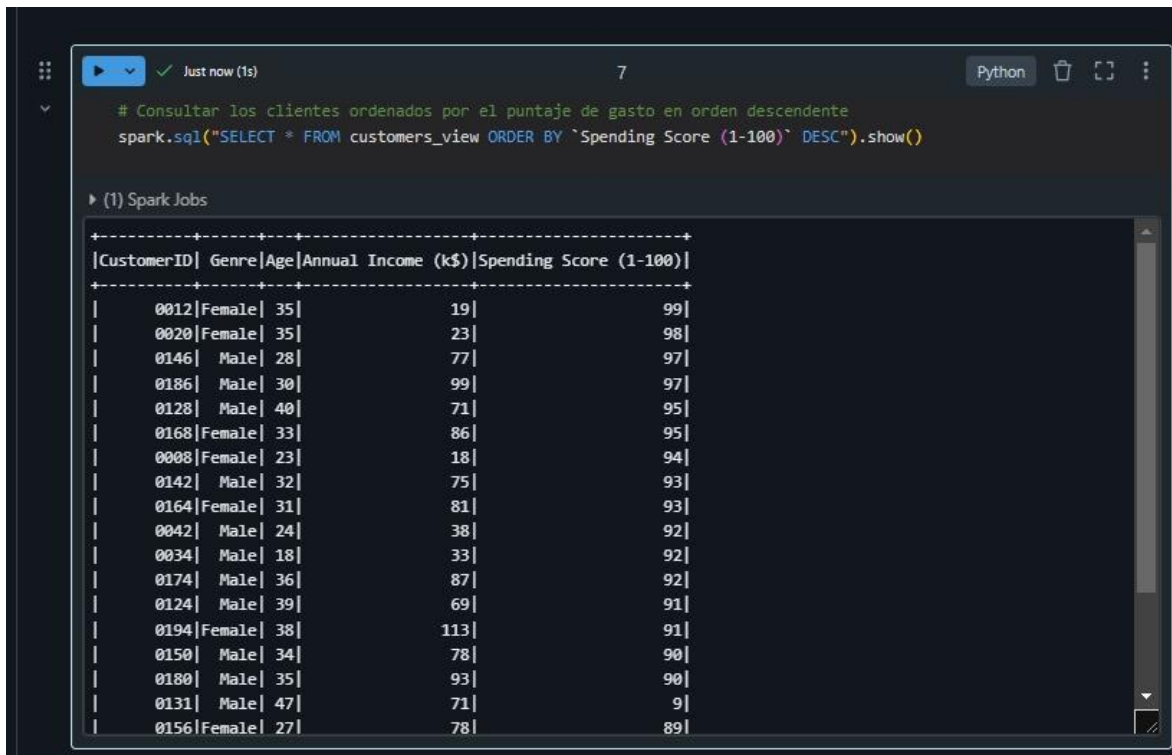
6

Python

```
# Consultar los clientes que tienen un ingreso anual mayor a 70,000
spark.sql("SELECT * FROM customers_view WHERE `Annual Income (k$)` > 70").show()
```

▶ (1) Spark Jobs

CustomerID	Genre	Age	Annual Income (k\$)	Spending Score (1-100)
0127	Male	43	71	35
0128	Male	40	71	95
0129	Male	59	71	11
0130	Male	38	71	75
0131	Male	47	71	9
0132	Male	39	71	75
0133	Female	25	72	34
0134	Female	31	72	71
0135	Male	20	73	5
0136	Female	29	73	88
0137	Female	44	73	7
0138	Male	32	73	73
0139	Male	19	74	10
0140	Female	35	74	72
0141	Female	57	75	5
0142	Male	32	75	93
0143	Female	28	76	40
0144	Female	32	76	87



The screenshot shows a Databricks notebook interface. At the top, there's a status bar with a play button, a checkmark, the text 'Just now (1s)', a tab labeled '7', and a 'Python' button with some icons. Below this, a code cell contains a Spark SQL query: `# Consultar los clientes ordenados por el puntaje de gasto en orden descendente` followed by `spark.sql("SELECT * FROM customers_view ORDER BY `Spending Score (1-100)` DESC").show()`. Below the code cell, there's a section titled '(1) Spark Jobs' which displays the results of the query as a table. The table has five columns: CustomerID, Genre, Age, Annual Income (k\$), and Spending Score (1-100). The results are sorted by the Spending Score in descending order.

CustomerID	Genre	Age	Annual Income (k\$)	Spending Score (1-100)
0012	Female	35	19	99
0020	Female	35	23	98
0146	Male	28	77	97
0186	Male	30	99	97
0128	Male	40	71	95
0168	Female	33	86	95
0008	Female	23	18	94
0142	Male	32	75	93
0164	Female	31	81	93
0042	Male	24	38	92
0034	Male	18	33	92
0174	Male	36	87	92
0124	Male	39	69	91
0194	Female	38	113	91
0150	Male	34	78	90
0180	Male	35	93	90
0131	Male	47	71	9
0156	Female	27	78	89

Parte 3 Utilizando el modo sql realizar 4 consultas y tabla temporal

En esta parte del laboratorio se utiliza el modo SQL del notebook para realizar consultas adicionales sobre los datos cargados. Se aprovecha la familiaridad con las consultas SQL para explorar la información disponible en el dataset. Además, se procede a la creación de una tabla temporal mediante la inserción de los datos desde la vista temporal generada previamente. Esta tabla temporal facilita la organización y reutilización de datos en consultas posteriores, mejorando la eficiencia del análisis y la manipulación de la información dentro del entorno de Databricks.

2 minutes ago (2s)

8

SQL

%sql

-- Clientes por genero

SELECT Genre, COUNT(*) AS Total FROM customers_view GROUP BY Genre;

▶ (2) Spark Jobs

▶

_sqldf: pyspark.sql.dataframe.DataFrame = [Genre: string, Total: long]

Table

	Genre	Total
1	Female	112
2	Male	88

2 rows

|

1.53 seconds runtime

Refreshed 2 minutes ago

Just now (2s)

9

SQL

%sql

--Gasto promedio por genero

SELECT Genre, AVG(`Spending Score (1-100)`) AS Avg_Spending

FROM customers_view

GROUP BY Genre;

▶ (2) Spark Jobs

▶

_sqldf: pyspark.sql.dataframe.DataFrame = [Genre: string, Avg_Spending: double]

Table

	Genre	Avg_Spending
1	Female	51.526785714285715
2	Male	48.51136363636363

2 rows

|

2.35 seconds runtime

Refreshed now

This result is stored as `_sqldf` and can be used in other `Python` cells.

Just now (1s) 10 SQL

```
%sql
--Consultar los clientes que tienen un ingreso anual mayor a 70,000
SELECT *
FROM customers_view
WHERE `Annual Income (k$)` > 70;
```

▶ (1) Spark Jobs

▶ `_sqldf`: pyspark.sql.dataframe.DataFrame = [CustomerID: string, Genre: string ... 3 more fields]

Table +

Q Y □

	^A _C CustomerID	^A _C Genre	^A _C Age	^A _C Annual Income (k\$)	^A _C Spending Score (1-100)
1	0127	Male	43	71	35
2	0128	Male	40	71	95
3	0129	Male	59	71	11
4	0130	Male	38	71	75
5	0131	Male	47	71	9
6	0132	Male	39	71	75
7	0133	Female	25	72	34
8	0134	Female	31	72	71
9	0135	Male	20	73	5
10	0136	Female	29	73	88
11	0137	Female	44	73	7
12	0138	Male	32	73	73
13	0139	Male	19	74	10
14	0140	Female	35	74	72
15	0141	Female	57	75	5

↓ 74 rows | 0.84 seconds runtime

Refreshed now

! This result is stored as `_sqldf` and can be used in other Python cells.

Just now (<1s) 12

```
%sql
--Crear una tabla temporal con los datos de la vista temporal:
CREATE TEMPORARY VIEW temp_customers AS
SELECT *
FROM customers_view;
```

_sqldf: pyspark.sql.dataframe.DataFrame
OK

This result is stored as `_sqldf` and can be used in other Python cells.

Just now (1s) 13 SQL

```
%sql
--Crear una tabla temporal con los datos de la vista temporal:
SELECT * FROM temp_customers;
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [CustomerID: string, Genre: string ... 3 more fields]

Table + 🔍 🏠

	CustomerID	Genre	Age	Annual Income (k\$)	Spending Score (1-100)
1	0001	Male	19	15	39
2	0002	Male	21	15	81
3	0003	Female	20	16	6
4	0004	Female	23	16	77
5	0005	Female	31	17	40
6	0006	Female	22	17	76
7	0007	Female	35	18	6
8	0008	Female	23	18	94
9	0009	Male	64	19	3
10	0010	Female	30	19	72
11	0011	Male	67	19	14

Parte 4 Almacenar resultados en archivo parquet

En esta última fase, se explora cómo almacenar los resultados de las consultas en archivos Parquet. Este formato es ampliamente utilizado en Big Data debido a su capacidad para manejar grandes volúmenes de datos de manera eficiente, con compresión y particionado incorporados. Al finalizar esta parte, los resultados generados estarán disponibles en Parquet, lo que facilita su reutilización y optimiza su almacenamiento para futuras consultas y análisis.

<>

+ Code + Text

▶ ✓ 2 minutes ago (6s) 14

```
# Guardar el DataFrame en formato Parquet
df.write.parquet("/ruta/al/almacenamiento/customers_parquet")
```

▶ (1) Spark Jobs

▶ ✓ just now (11s) 15 Python

```
# Leer el archivo Parquet
df_parquet = spark.read.parquet("/ruta/al/almacenamiento/customers_parquet")

# Mostrar el contenido
display(df_parquet)
```

▶ (2) Spark Jobs

▶ df_parquet: pyspark.sql.dataframe.DataFrame = [CustomerID: string, Genre: string ... 3 more fields]

Table +

🔍 🔼 📄

	CustomerID	Genre	Age	Annual Income (k\$)	Spending Score (1-100)
1	0001	Male	19	15	39
2	0002	Male	21	15	81
3	0003	Female	20	16	6
4	0004	Female	23	16	77
5	0005	Female	31	17	40
6	0006	Female	22	17	76
7	0007	Female	35	18	6
8	0008	Female	23	18	94
9	0009	Male	64	19	3
10	0010	Female	30	19	72
11	0011	Male	67	19	14
12	0012	Female	35	19	99
13	0013	Female	58	20	15
14	0014	Female	24	20	77
15	0015	Male	37	20	13

↓ 200 rows | 11.12 seconds runtime Refreshed now

Conclusión

A lo largo de este laboratorio se exploraron las capacidades de Databricks para cargar, consultar y almacenar datos de manera eficiente utilizando PySpark y SQL. El trabajo permitió reforzar conocimientos sobre vistas temporales, consultas relacionales y el uso de tablas temporales, además de familiarizarse con el formato Parquet para el almacenamiento de resultados. Estas habilidades son fundamentales en el análisis de grandes volúmenes de datos y preparan al estudiante para enfrentar retos en entornos de Big Data, optimizando tanto el procesamiento como la gestión de la información.