PySpark Coding Challenge

-Sarthak Niranjan Kulkarni (Maverick)

- [sarthakkul2311@gmail.com](mailto:sarthakkul2311@gmail.com) - (+91) 93256 02791

**26/11/2024 (Tuesday)**

**1) Explain ETL (Extract, Transform, Load) with PySpark(in your own words):**

🡪 **ETL:** ETL with PySpark means using Python and Apache Spark to extract data from different sources, clean or transform it as needed, and load it into a storage system like a database or data warehouse. It’s a fast and efficient way to handle large datasets for analysis or reporting. It helps automate and streamline data workflows, making it easier to manage and process big data in real-time or batches.

**Extract:** Retrieve data from various sources like databases, files or APIs.

**Transform:** Clean, aggregate and manipulate data to fit your analysis needs.

**Load:** Store the transformed data into a database or data warehouse for analysis.

**Advantages of ETL: -**

* Data Centralization: Brings scattered data into a single, unified location.
* Improved Data Quality: Cleans and standardizes data, ensuring accuracy and consistency.
* Faster Decision-Making: Prepares data for quick analysis and insights.
* Time and Cost Efficiency: Automates repetitive data processing tasks, saving resources.
* Flexibility: Handles various data types and formats from multiple sources.
* Scalability: Easily adapts to growing data volumes.

**Example:** Extracting sales data from multiple CSV files, cleaning it to remove duplicates, and loading it into a database for reporting using PySpark.

**2) Using Spark SQL - Transformations such as Filter, Join, Simple Aggregations, GroupBy on the case study dataset: -**

**1. Filter: -**

🡪 # Filter customers older than 40

from pyspark.sql import SparkSession

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

filtered\_df = df.filter(df['Age'] > 40)

filtered\_df.show()

A screenshot of a computer program

Description automatically generated

🡪 # Filter married customers with Expenditure greater than 20,000

married\_high\_expenditure\_sql\_df = spark.sql("SELECT \* FROM loan WHERE `Marital Status` = 'MARRIED' AND Expenditure > 20000")

married\_high\_expenditure\_sql\_df.show()

A screenshot of a computer

Description automatically generated

**2. Joins: -** (Performed with Case study Datasets (Loan and Credit\_card))

🡪 **Types Of Joins: -**

* **Inner Join:**

**#** Load the loan and credit tables as DataFrames

loan\_df = spark.table("loan")

credit\_df = spark.table("credit")

# Inner join between loan and credit tables on Customer\_ID

inner\_join\_df = loan\_df.join(credit\_df, loan\_df.Customer\_ID == credit\_df.CustomerId, 'inner')

inner\_join\_df.show()

A screenshot of a computer

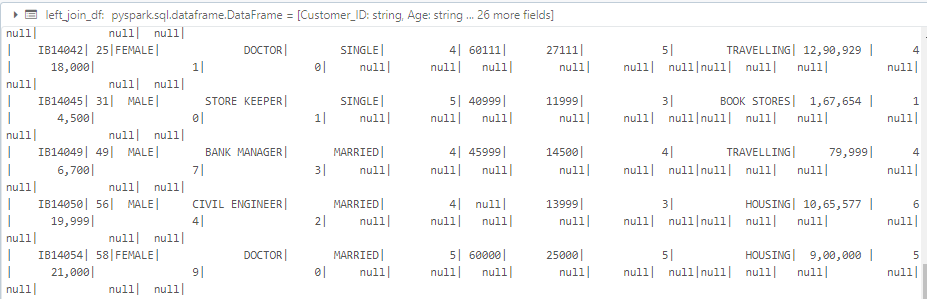
Description automatically generated

* **Left Join: -**

# Left join between loan and credit tables on Customer\_ID

left\_join\_df = loan\_df.join(credit\_df, loan\_df.Customer\_ID == credit\_df.CustomerId, 'left')

left\_join\_df.show()



* **Right Join: -**

# Right join between loan and credit tables on Customer\_ID

right\_join\_df = loan\_df.join(credit\_df, loan\_df.Customer\_ID == credit\_df.CustomerId, 'right')

right\_join\_df.show()

A screenshot of a computer

Description automatically generated

* **Outer Join: -**

# Outer join between loan and credit tables on Customer\_ID

outer\_join\_df = loan\_df.join(credit\_df, loan\_df.Customer\_ID == credit\_df.CustomerId, 'outer')

outer\_join\_df.show()

A screenshot of a computer code

Description automatically generated

**3. Simple Aggregate Functions: -**

🡪 # Count the number of records in the loan table using PySpark

count\_records = spark.sql("SELECT COUNT(\*) AS total\_records FROM loan")

count\_records.show()

# Average Income Amount in the loan table using PySpark

average\_income = spark.sql("""

SELECT AVG(`Income`) AS avg\_income

FROM loan

""")

average\_income.show()

A screenshot of a computer

Description automatically generated

**4. Group By: -**

🡪 # Group by Marital Status and calculate the total Expenditure using PySpark

group\_by\_marital\_status = spark.sql("""

SELECT `Marital Status`, SUM(Expenditure) AS total\_expenditure

FROM loan

GROUP BY `Marital Status`

""")

group\_by\_marital\_status.show()

# Group by Loan Category and calculate the average Expenditure using PySpark

group\_by\_loan\_category = spark.sql("""

SELECT `Loan Category`, AVG(Expenditure) AS avg\_expenditure

FROM loan

GROUP BY `Loan Category`

""")

group\_by\_loan\_category.show()

A screenshot of a computer

Description automatically generated