PySpark Coding Challenge Data Engineering

J Jatin

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

ETL (Extract, Transform, Load) is a process used in data engineering to prepare and move data from various sources into a centralized system, such as a data warehouse or a big data platform. When implemented with PySpark, a distributed computing framework, ETL becomes highly scalable and efficient for processing large datasets.

1. Extract

This step involves retrieving data from various sources, such as databases, APIs, files, or streams. In PySpark:

Example Code:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("ETL").getOrCreate()
# Extracting data from a CSV file
df = spark.read.csv("data/source.csv", header=True,
inferSchema=True)
```

2. Transform

This step cleans, filters, enriches, and reshapes the data to make it suitable for analysis. PySpark's **DataFrame API** and **SQL** capabilities are often used for this.

E.g.: Removing duplicates, handling missing values, aggregating data, and applying complex transformations.

3. Load

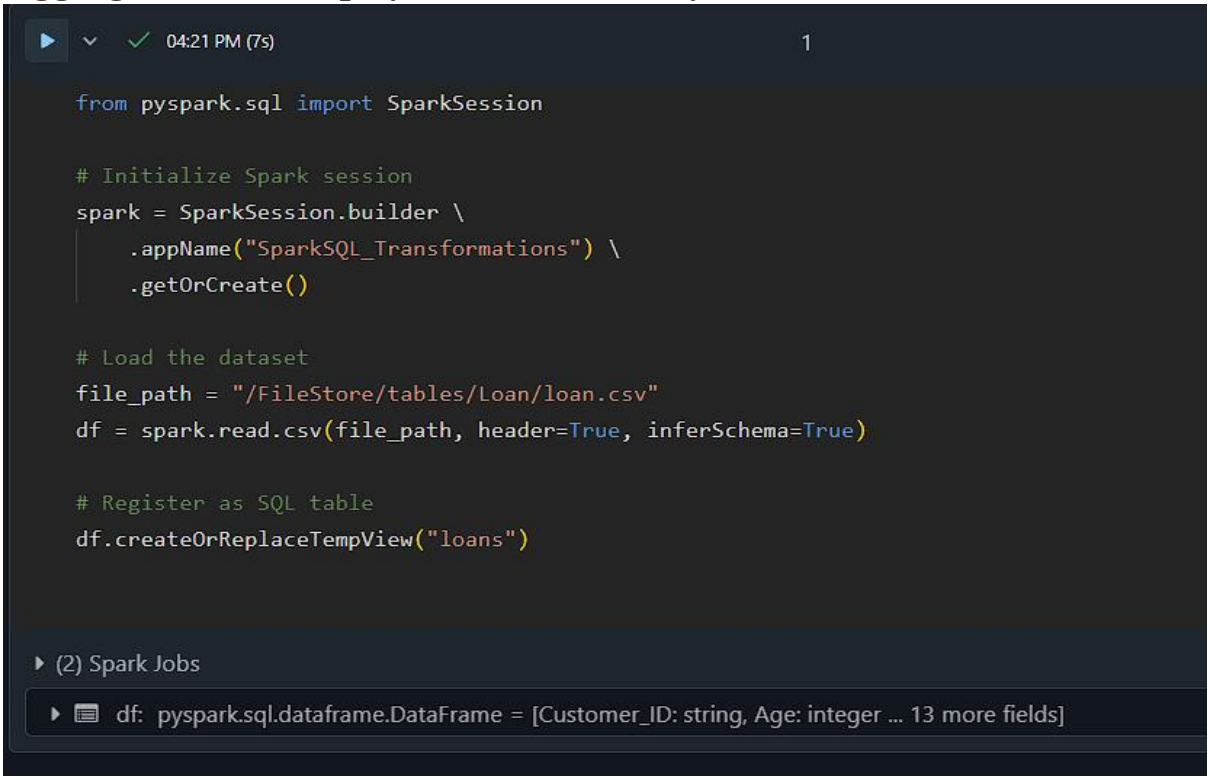
This step saves the transformed data to a target system, such as a database, data warehouse, or file storage, for further use.

Advantages of Using PySpark for ETL

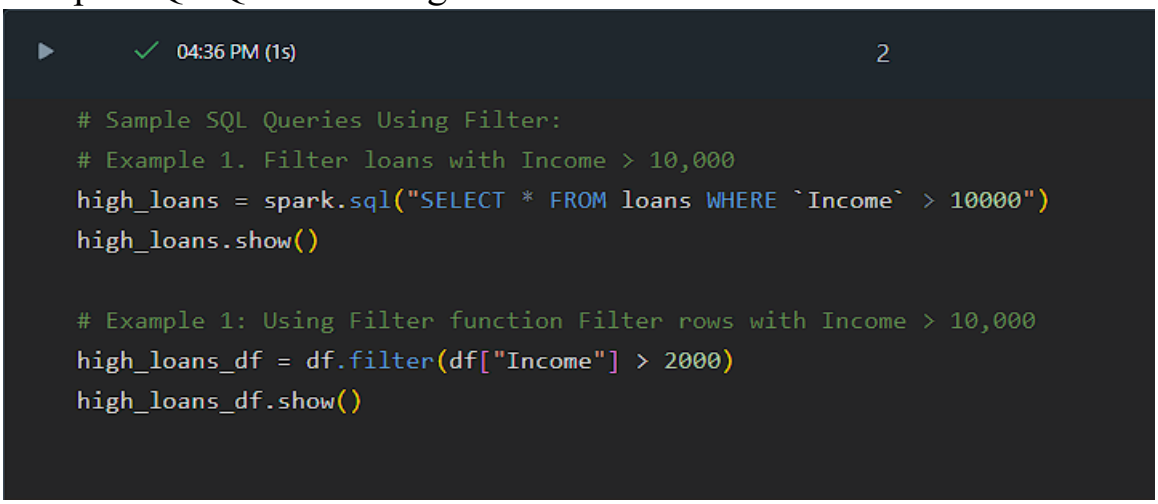
Scalability: PySpark's distributed nature allows handling terabytes or petabytes of data efficiently.

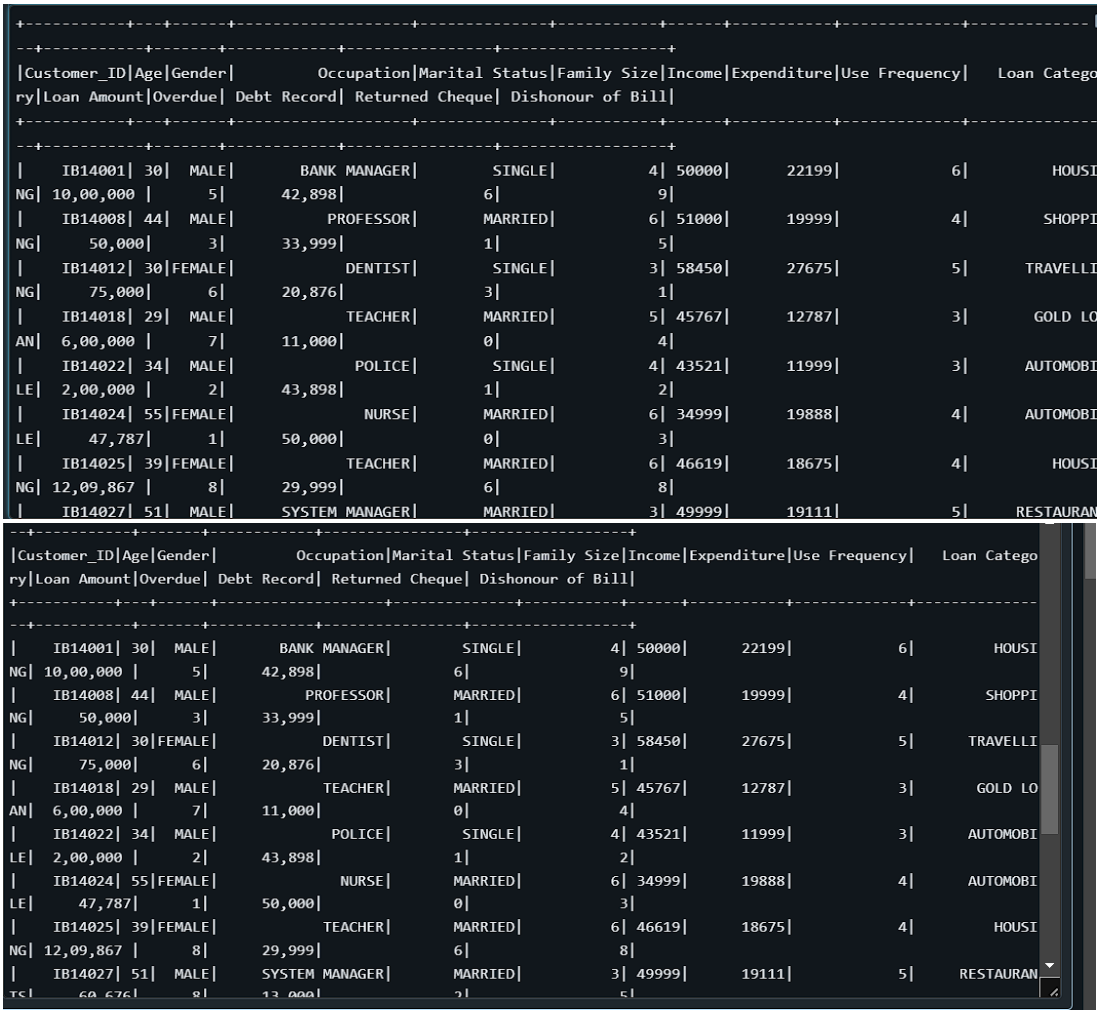
Fault Tolerance: Built-in mechanisms ensure recovery from failures during the process.
Versatility: Supports diverse data formats and integrates with big data ecosystems like Hadoop and Hive.
Performance: Optimized execution plans via Catalyst and Tungsten improve speed and resource utilization.

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



1. Sample SQL Queries Using Filter:

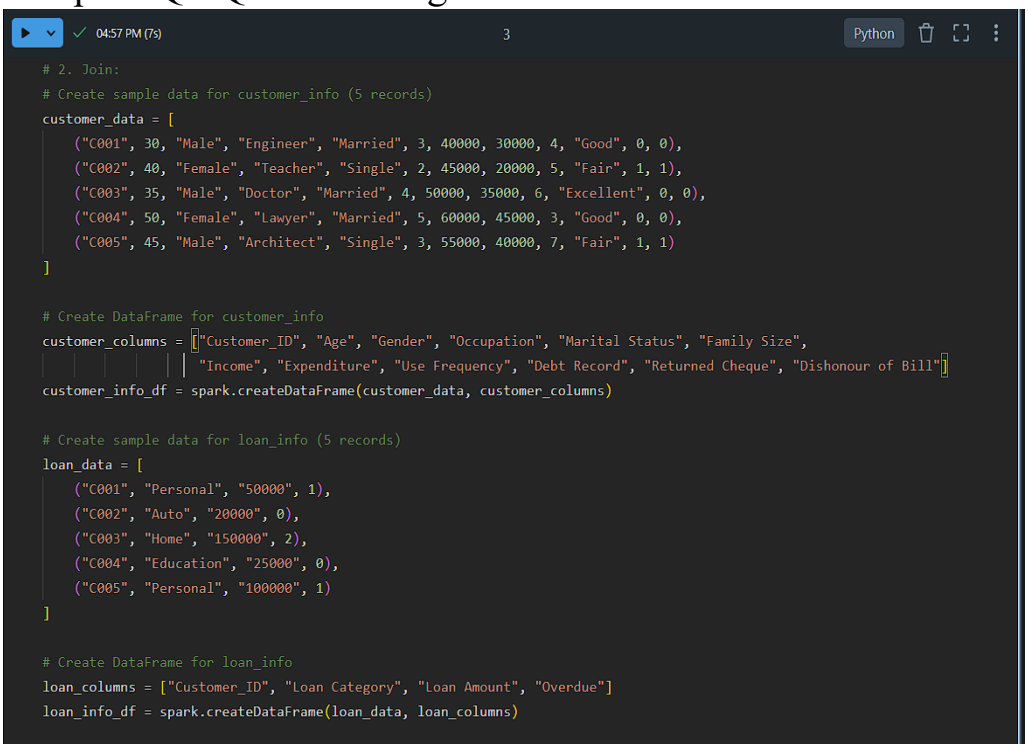




```
+-----------+---+------+------------------+--------------+-----------+------+-----------+-------------+-------------
--+-----------+-------+------------+----------------+------------------+
|Customer_ID|Age|Gender|        Occupation|Marital Status|Family Size|Income|Expenditure|Use Frequency|   Loan Catego
ry|Loan Amount|Overdue| Debt Record| Returned Cheque| Dishonour of Bill|
+-----------+---+------+------------------+--------------+-----------+------+-----------+-------------+-------------
--+-----------+-------+------------+----------------+------------------+
|    IB14001| 30|  MALE|      BANK MANAGER|        SINGLE|          4| 50000|      22199|            6|         HOUSI
NG| 10,00,000 |      5|      42,898|               6|                 9|
|    IB14008| 44|  MALE|         PROFESSOR|       MARRIED|          6| 51000|      19999|            4|        SHOPPI
NG|     50,000|      3|      33,999|               1|                 5|
|    IB14012| 30|FEMALE|           DENTIST|        SINGLE|          3| 58450|      27675|            5|      TRAVELLI
NG|     75,000|      6|      20,876|               3|                 1|
|    IB14018| 29|  MALE|           TEACHER|       MARRIED|          5| 45767|      12787|            3|       GOLD LO
AN|  6,00,000 |      7|      11,000|               0|                 4|
|    IB14022| 34|  MALE|            POLICE|        SINGLE|          4| 43521|      11999|            3|      AUTOMOBI
LE|  2,00,000 |      2|      43,898|               1|                 2|
|    IB14024| 55|FEMALE|             NURSE|       MARRIED|          6| 34999|      19888|            4|      AUTOMOBI
LE|     47,787|      1|      50,000|               0|                 3|
|    IB14025| 39|FEMALE|           TEACHER|       MARRIED|          6| 46619|      18675|            4|         HOUSI
NG| 12,09,867 |      8|      29,999|               6|                 8|
|    IB14027| 51|  MALE|    SYSTEM MANAGER|       MARRIED|          3| 49999|      19111|            5|    RESTAURAN
```


```
|Customer_ID|Age|Gender|        Occupation|Marital Status|Family Size|Income|Expenditure|Use Frequency|   Loan Catego
ry|Loan Amount|Overdue| Debt Record| Returned Cheque| Dishonour of Bill|
```


2. Sample SQL Queries Using Join:



```python
# 2. Join:
# Create sample data for customer_info (5 records)
customer_data = [
    ("C001", 30, "Male", "Engineer", "Married", 3, 40000, 30000, 4, "Good", 0, 0),
    ("C002", 40, "Female", "Teacher", "Single", 2, 45000, 20000, 5, "Fair", 1, 1),
    ("C003", 35, "Male", "Doctor", "Married", 4, 50000, 35000, 6, "Excellent", 0, 0),
    ("C004", 50, "Female", "Lawyer", "Married", 5, 60000, 45000, 3, "Good", 0, 0),
    ("C005", 45, "Male", "Architect", "Single", 3, 55000, 40000, 7, "Fair", 1, 1)
]

# Create DataFrame for customer_info
customer_columns = ["Customer_ID", "Age", "Gender", "Occupation", "Marital Status", "Family Size",
                    "Income", "Expenditure", "Use Frequency", "Debt Record", "Returned Cheque", "Dishonour of Bill"]
customer_info_df = spark.createDataFrame(customer_data, customer_columns)

# Create sample data for loan_info (5 records)
loan_data = [
    ("C001", "Personal", "50000", 1),
    ("C002", "Auto", "20000", 0),
    ("C003", "Home", "150000", 2),
    ("C004", "Education", "25000", 0),
    ("C005", "Personal", "100000", 1)
]

# Create DataFrame for loan_info
loan_columns = ["Customer_ID", "Loan Category", "Loan Amount", "Overdue"]
loan_info_df = spark.createDataFrame(loan_data, loan_columns)
```

```
# Register both DataFrames as temp views (tables)
customer_info_df.createOrReplaceTempView("customer_info")
loan_info_df.createOrReplaceTempView("loans")
```

Inner Join:

```
# Inner Join: Get the customer details along with their loan information
inner_join_query = """
SELECT
    c.Customer_ID,
    c.Age,
    c.Gender,
    c.Occupation,
    c.Income,
    l.`Loan Category`,
    l.`Loan Amount`,
    l.Overdue
FROM customer_info c
INNER JOIN loans l
ON c.Customer_ID = l.Customer_ID
"""

inner_join_result = spark.sql(inner_join_query)
inner_join_result.show()
```

```
+-----------+---+------+----------+------+-------------+-----------+-------+
|Customer_ID|Age|Gender|Occupation|Income|Loan Category|Loan Amount|Overdue|
+-----------+---+------+----------+------+-------------+-----------+-------+
|       C001| 30|  Male|  Engineer| 40000|     Personal|      50000|      1|
|       C002| 40|Female|   Teacher| 45000|         Auto|      20000|      0|
|       C003| 35|  Male|    Doctor| 50000|         Home|     150000|      2|
|       C004| 50|Female|    Lawyer| 60000|    Education|      25000|      0|
|       C005| 45|  Male| Architect| 55000|     Personal|     100000|      1|
+-----------+---+------+----------+------+-------------+-----------+-------+
```

Right Join:

```
# Left Join: Get all customers and their loan information (if available)
left_join_query = """
SELECT
    c.Customer_ID,
    c.Age,
    c.Gender,
    c.Occupation,
    c.Income,
    l.`Loan Category`,
    l.`Loan Amount`,
    l.Overdue
FROM customer_info c
LEFT JOIN loans l
ON c.Customer_ID = l.Customer_ID
"""

left_join_result = spark.sql(left_join_query)
left_join_result.show()
```

```
+-----------+---+------+----------+------+-------------+-----------+-------+
|Customer_ID|Age|Gender|Occupation|Income|Loan Category|Loan Amount|Overdue|
+-----------+---+------+----------+------+-------------+-----------+-------+
|       C001| 30|  Male|  Engineer| 40000|     Personal|      50000|      1|
|       C002| 40|Female|   Teacher| 45000|         Auto|      20000|      0|
|       C003| 35|  Male|    Doctor| 50000|         Home|     150000|      2|
|       C004| 50|Female|    Lawyer| 60000|    Education|      25000|      0|
|       C005| 45|  Male| Architect| 55000|     Personal|     100000|      1|
+-----------+---+------+----------+------+-------------+-----------+-------+
```

3. Sample SQL Queries Using Aggregation:

```
# 1. Aggregation: Calculate the average Income
avg_loan = spark.sql("SELECT AVG(Income) AS Avg_Income FROM loans")
avg_loan.show()
```

```
+-----------------+
|       Avg_Income|
+-----------------+
|68339.49145299145|
+-----------------+
```

```
# 2. Aggregation: Calculate the Average Loan Amount:
avg_loan_amount = spark.sql("SELECT AVG(`Loan Amount`) as Average_Loan_Amount FROM loans")
avg_loan_amount.show()
```

4. Sample SQL Queries Using Group-By:

```
Just now (1s)                                                    4

# Sample SQL Queries Using GroupBy:
# 1. Grouping by Occupation and calculating the average Income for each occupation
# SQL Query to group by 'Occupation' and calculate average 'Income'
group_by_occupation_query = """
SELECT
    Occupation,
    AVG(Income) AS Avg_Income
FROM customer_info
GROUP BY Occupation
"""
```

group_by_gender_result: pyspark.sql.d

```
+----------+----------+
|Occupation|Avg_Income|
+----------+----------+
|  Engineer|   40000.0|
|   Teacher|   45000.0|
|    Doctor|   50000.0|
|    Lawyer|   60000.0|
| Architect|   55000.0|
+----------+----------+
```

```
#2. Grouping by Gender and calculating the total Expenditure for each gender
group_by_gender_query = """
SELECT
    Gender,
    SUM(Expenditure) AS Total_Expenditure
FROM customer_info
GROUP BY Gender
"""

group_by_gender_result = spark.sql(group_by_gender_query)
group_by_gender_result.show()
```

```
+------+-----------------+
|Gender|Total_Expenditure|
+------+-----------------+
|  Male|           105000|
|Female|            65000|
+------+-----------------+
```