Coding Challenge-3

Name: **Harish Er**

Q1) Explain ETL (Extract, Transform, Load) with PySpark(in your own words).

Ans: ETL (Extract, Transform, Load) is a process in data engineering where data is taken from a source system, processed or transformed to fit analytical or operational needs, and then loaded into a target system like a database or data warehouse. Using PySpark, which is a Python interface for Apache Spark, this process becomes scalable and efficient for handling large datasets. Here's how ETL works in PySpark:

1. **Extract**

This phase involves reading data from different sources like CSV files, JSON files, databases, or even real-time streams. PySpark uses its DataFrame API to simplify this step. For example, you can use spark.read to load data into a DataFrame:

 # Extract data from a CSV file

df = spark.read.csv("path/to/data.csv", header=True, inferSchema=True)

2. **Transform**

Once the data is extracted, it often requires cleaning, filtering, aggregating, or reshaping. PySpark provides a rich set of functions for data manipulation through its SQL-like operations and built-in methods. Common transformations include:

- Filtering rows: df.filter(df['column'] > value)
- Adding or modifying columns: df.withColumn('new_col', df['existing_col'] * 2)
- Aggregations: df.groupBy('column').agg({'value': 'sum'})
- Joining datasets: df1.join(df2, 'common_column', 'inner')

An example of transformation:

transformed_df = df.withColumn("total", df["quantity"] * df["price"])

3. **Load**

In this phase, the transformed data is saved into a target system like a database, a data warehouse, or another file format. PySpark's write APIs allow saving data in various formats like CSV, Parquet, or directly to databases.

 # Save data to a Parquet file

transformed_df.write.parquet("path/to/output.parquet")

Advantages of PySpark for ETL

1. **Scalability:** Processes large datasets distributed across clusters.
2. **Efficiency:** Optimized for high-performance batch and stream processing.
3. **Flexibility**: Supports various data formats and transformation operations.
4. **Ease of Use:** Python-friendly API makes it accessible for Python developers.

PySpark's distributed nature makes it ideal for ETL workflows involving massive data volumes, ensuring faster and more efficient processing.

Q2) Using Spark SQL - Transformations such as Filter, Join, Simple Aggregations, GroupBy on the case study dataset.

1. Pyspark:

1. Filter:

```python
# Filter: Customers with income greater than 50,000 and single marital status
filtered_df = loan_df.filter((loan_df["Income"] > 50000) & (loan_df["Marital Status"] == "SINGLE"))
filtered_df.show(5)
```

(1) Spark Jobs

filtered_df: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 13 more fields]

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
IB14012	30	FEMALE	DENTIST	SINGLE	3	58450	27675	5	TRAVELLING	75,000	6	20,876	3	1
IB14032	24	MALE	DATA ANALYST	SINGLE	4	60111	28999	6	AUTOMOBILE	35,232	5	33,333	1	2
IB14042	25	FEMALE	DOCTOR	SINGLE	4	60111	27111	5	TRAVELLING	12,90,929	4	18,000	1	0
IB14089	25	MALE	PROFESSOR	SINGLE	5	62145	31254	4	BOOK STORES	12,45,789	6	48,596	6	5
IBI4155	24	FEMALE	SOFTWARE ENGINEER	SINGLE	4	55680	29000	5	AUTOMOBILE	7,89,000	5	24,000	6	4

only showing top 5 rows

Explanation: Filtering retrieves specific rows based on conditions. In this case, we filter customers with an income greater than 50,000 and marital status as 'SINGLE'.

2. Join:

- **Inner Join:**

```python
# Example second dataset
demographics = [(1, "Urban"), (2, "Rural"), (3, "Urban")]
columns = ["Customer_ID", "Area"]

# Convert to PySpark DataFrame
demographics_df = spark.createDataFrame(demographics, columns)

# Inner join: Matches rows in both datasets
inner_joined = loan_df.join(demographics_df, "Customer_ID", "inner")
inner_joined.show()
```

(4) Spark Jobs

demographics_df: pyspark.sql.dataframe.DataFrame = [Customer_ID: long, Area: string]

inner_joined: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 14 more fields]

```
|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|Area|
```

- **Left Join:**

```
# Left join: All rows from the left DataFrame, matching rows from the right
left_joined = loan_df.join(demographics_df, "Customer_ID", "left")
left_joined.show(5)
```

```
(4) Spark Jobs
demographics_df: pyspark.sql.dataframe.DataFrame = [Customer_ID: long, Area: string]
left_joined: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 14 more fields]
+-----------+---+------+------------+--------------+-----------+------+-----------+-------------+-------------+-----------+-------+-----------+---------------+---
---------------+----+
|Customer_ID|Age|Gender|  Occupation|Marital Status|Family Size|Income|Expenditure|Use Frequency|Loan Category|Loan Amount|Overdue| Debt Record| Returned Cheque| D
ishonour of Bill|Area|
+-----------+---+------+------------+--------------+-----------+------+-----------+-------------+-------------+-----------+-------+-----------+---------------+---
---------------+----+
|    IB14001| 30|  MALE|BANK MANAGER|        SINGLE|          4| 50000|      22199|            6|      HOUSING| 10,00,000 |      5|     42,898|              6|
9|null|
|    IB14008| 44|  MALE|   PROFESSOR|       MARRIED|          6| 51000|      19999|            4|     SHOPPING|     50,000|      3|     33,999|              1|
5|null|
|    IB14012| 30|FEMALE|     DENTIST|        SINGLE|          3| 58450|      27675|            5|   TRAVELLING|     75,000|      6|     20,876|              3|
1|null|
|    IB14018| 29|  MALE|     TEACHER|       MARRIED|          5| 45767|      12787|            3|    GOLD LOAN|  6,00,000 |      7|     11,000|              0|
4|null|
|    IB14022| 34|  MALE|      POLICE|        SINGLE|          4| 43521|      11999|            3|   AUTOMOBILE|  2,00,000 |      2|     43,898|              1|
2|null|
+-----------+---+------+------------+--------------+-----------+------+-----------+-------------+-------------+-----------+-------+-----------+---------------+---
---------------+----+
only showing top 5 rows
```

- **Full Outer Join:**

```
# Full outer join: All rows from both DataFrames, with nulls for non-matches
outer_joined = loan_df.join(demographics_df, "Customer_ID", "outer")
outer_joined.show(5)
```

```
(3) Spark Jobs
demographics_df: pyspark.sql.dataframe.DataFrame = [Customer_ID: long, Area: string]
outer_joined: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 14 more fields]
+-----------+---+------+------------+--------------+-----------+------+-----------+-------------+-------------+-----------+-------+-----------+---------------+---
---------------+----+
|Customer_ID|Age|Gender|  Occupation|Marital Status|Family Size|Income|Expenditure|Use Frequency|Loan Category|Loan Amount|Overdue| Debt Record| Returned Cheque| D
ishonour of Bill|Area|
+-----------+---+------+------------+--------------+-----------+------+-----------+-------------+-------------+-----------+-------+-----------+---------------+---
---------------+----+
|    IB14001| 30|  MALE|BANK MANAGER|        SINGLE|          4| 50000|      22199|            6|      HOUSING| 10,00,000 |      5|     42,898|              6|
9|null|
|    IB14008| 44|  MALE|   PROFESSOR|       MARRIED|          6| 51000|      19999|            4|     SHOPPING|     50,000|      3|     33,999|              1|
5|null|
|    IB14012| 30|FEMALE|     DENTIST|        SINGLE|          3| 58450|      27675|            5|   TRAVELLING|     75,000|      6|     20,876|              3|
1|null|
|    IB14018| 29|  MALE|     TEACHER|       MARRIED|          5| 45767|      12787|            3|    GOLD LOAN|  6,00,000 |      7|     11,000|              0|
4|null|
|    IB14022| 34|  MALE|      POLICE|        SINGLE|          4| 43521|      11999|            3|   AUTOMOBILE|  2,00,000 |      2|     43,898|              1|
2|null|
+-----------+---+------+------------+--------------+-----------+------+-----------+-------------+-------------+-----------+-------+-----------+---------------+---
---------------+----+
only showing top 5 rows
```

Explanation:

- **Inner Join**: Rows are included only if Customer_ID exists in both DataFrames.
- **Left Join**: Preserves all rows from the loan dataset (loan_data), with nulls for missing matches in demographics.
- **Full Outer Join**: Combines rows from both DataFrames, filling nulls where no match exists.

3. Simple Aggregations

```
Just now (1s)                                   4                                   Python

# Total loan amount
loan_df.filter(loan_df["Expenditure"] > 50000).count()

(2) Spark Jobs
Out[15]: 6
```

Explanation: Aggregations calculate metrics like sum, average, or count. Here, we compute the total loan amount.

4. GroupBy

```
Just now (2s)                                   5                                   Python

grouped_df = loan_df.groupBy("Loan Category").avg("Income")
grouped_df.show(5)

(2) Spark Jobs
grouped_df: pyspark.sql.dataframe.DataFrame = [Loan Category: string, avg(Income): double]
+-------------+-----------------+
|Loan Category|      avg(Income)|
+-------------+-----------------+
|      HOUSING| 74728.19354838709|
|   TRAVELLING| 57016.58490566038|
|  BOOK STORES|50903.142857142855|
|  AGRICULTURE|60372.666666666664|
|    GOLD LOAN| 70838.31506849315|
+-------------+-----------------+
only showing top 5 rows
```

Explanation: Grouping organizes data into categories and applies aggregate functions (e.g., average, sum). Here, we calculate the average loan amount for each loan category.

2. Spark SQL:

1. Filter:

```python
loan_df.createOrReplaceTempView("loan_data")
filtered_sql = spark.sql("""
    SELECT *
    FROM loan_data
    WHERE Income > 50000 AND `Marital Status` = 'SINGLE'
""")
filtered_sql.show(5)
```

(1) Spark Jobs

filtered_sql: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 13 more fields]

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
IB14012	30	FEMALE	DENTIST	SINGLE	3	58450	27675	5	TRAVELLING	75,000	6	20,876	3	1
IB14032	24	MALE	DATA ANALYST	SINGLE	4	60111	28999	6	AUTOMOBILE	35,232	5	33,333	1	2
IB14042	25	FEMALE	DOCTOR	SINGLE	4	60111	27111	5	TRAVELLING	12,90,929	4	18,000	1	0
IB14089	25	MALE	PROFESSOR	SINGLE	5	62145	31254	4	BOOK STORES	12,45,789	6	48,596	6	5
IBI4155	24	FEMALE	SOFTWARE ENGINEER	SINGLE	4	55680	29000	5	AUTOMOBILE	7,89,000	5	24,000	6	4

Explanation: Filtering retrieves specific rows based on conditions. In this case, we filter customers with an income greater than 50,000 and marital status as 'SINGLE'.

2. Join:

- **Inner Join:**

```python
demographics_df.createOrReplaceTempView("demographics")
joined_sql = spark.sql("""
    SELECT l.*, d.Area
    FROM loan_data l
    INNER JOIN demographics d
    ON l.Customer_ID = d.Customer_ID
""")
joined_sql.show()
```

(4) Spark Jobs

joined_sql: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 14 more fields]

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	Area

- **Left Join:**

```
Just now (2s)                                                7                                   Python

demographics_df.createOrReplaceTempView("demographics")
left_sql = spark.sql("""
    SELECT l.*, d.Area
    FROM loan_data l
    LEFT JOIN demographics d
    ON l.Customer_ID = d.Customer_ID
""")
left_sql.show(5)
```

```
(4) Spark Jobs
left_sql:  pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 14 more fields]
+-----------+---+------+------------+--------------+-----------+------+-----------+-------------+-------------+-----------+-------+-----------+----------------+--
----------------+----+
|Customer_ID|Age|Gender|  Occupation|Marital Status|Family Size|Income|Expenditure|Use Frequency|Loan Category|Loan Amount|Overdue| Debt Record| Returned Cheque| D
ishonour of Bill|Area|
+-----------+---+------+------------+--------------+-----------+------+-----------+-------------+-------------+-----------+-------+-----------+----------------+--
----------------+----+
|    IB14001| 30|  MALE|BANK MANAGER|        SINGLE|          4| 50000|      22199|            6|      HOUSING| 10,00,000 |      5|     42,898|               6|
9|null|
|    IB14008| 44|  MALE|   PROFESSOR|       MARRIED|          6| 51000|      19999|            4|     SHOPPING|     50,000|      3|     33,999|               1|
5|null|
|    IB14012| 30|FEMALE|     DENTIST|        SINGLE|          3| 58450|      27675|            5|   TRAVELLING|     75,000|      6|     20,876|               3|
1|null|
|    IB14018| 29|  MALE|     TEACHER|       MARRIED|          5| 45767|      12787|            3|    GOLD LOAN|  6,00,000 |      7|     11,000|               0|
4|null|
|    IB14022| 34|  MALE|      POLICE|        SINGLE|          4| 43521|      11999|            3|   AUTOMOBILE|  2,00,000 |      2|     43,898|               1|
2|null|
+-----------+---+------+------------+--------------+-----------+------+-----------+-------------+-------------+-----------+-------+-----------+----------------+--
----------------+----+
```

- **Full Outer Join:**

```
Just now (1s)                                                7                                   Python

demographics_df.createOrReplaceTempView("demographics")
outer_sql = spark.sql("""
    SELECT l.*, d.Area
    FROM loan_data l
    FULL OUTER JOIN demographics d
    ON l.Customer_ID = d.Customer_ID
""")
outer_sql.show(5)
```

```
(3) Spark Jobs
outer_sql:  pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 14 more fields]
+-----------+---+------+------------+--------------+-----------+------+-----------+-------------+-------------+-----------+-------+-----------+----------------+--
----------------+----+
|Customer_ID|Age|Gender|  Occupation|Marital Status|Family Size|Income|Expenditure|Use Frequency|Loan Category|Loan Amount|Overdue| Debt Record| Returned Cheque| D
ishonour of Bill|Area|
+-----------+---+------+------------+--------------+-----------+------+-----------+-------------+-------------+-----------+-------+-----------+----------------+--
----------------+----+
|    IB14001| 30|  MALE|BANK MANAGER|        SINGLE|          4| 50000|      22199|            6|      HOUSING| 10,00,000 |      5|     42,898|               6|
9|null|
|    IB14008| 44|  MALE|   PROFESSOR|       MARRIED|          6| 51000|      19999|            4|     SHOPPING|     50,000|      3|     33,999|               1|
5|null|
|    IB14012| 30|FEMALE|     DENTIST|        SINGLE|          3| 58450|      27675|            5|   TRAVELLING|     75,000|      6|     20,876|               3|
1|null|
|    IB14018| 29|  MALE|     TEACHER|       MARRIED|          5| 45767|      12787|            3|    GOLD LOAN|  6,00,000 |      7|     11,000|               0|
4|null|
|    IB14022| 34|  MALE|      POLICE|        SINGLE|          4| 43521|      11999|            3|   AUTOMOBILE|  2,00,000 |      2|     43,898|               1|
2|null|
+-----------+---+------+------------+--------------+-----------+------+-----------+-------------+-------------+-----------+-------+-----------+----------------+--
----------------+----+
```

Explanation:

- **Inner Join:** Includes rows where keys match in both datasets.
- **Left Join:** Includes all rows from the left dataset and matching rows from the right.
- **Full Outer Join:** Includes all rows from both datasets, with nulls where no match is found.

3. Simple Aggregations:

Just now (1s) 8 Python

```python
aggregation_sql = spark.sql("""
    SELECT SUM(`Expenditure`) AS Total_Loan_Amount
    FROM loan_data
""")
aggregation_sql.show()
```

(2) Spark Jobs

aggregation_sql: pyspark.sql.dataframe.DataFrame = [Total_Loan_Amount: long]

```
+-----------------+
|Total_Loan_Amount|
+-----------------+
|         13243460|
+-----------------+
```

Explanation: Aggregations calculate metrics like sum, average, or count. Here, we compute the total loan amount. SQL uses the SUM() function.

4. GroupBy:

Just now (1s) 9 Python

```python
grouped_sql = spark.sql("""
    SELECT `Loan Category`, AVG(`Income`) AS Avg_Income
    FROM loan_data
    GROUP BY `Loan Category`
""")
grouped_sql.show()
```

(2) Spark Jobs

grouped_sql: pyspark.sql.dataframe.DataFrame = [Loan Category: string, Avg_Income: double]

```
+-----------------+-----------------+
|    Loan Category|       Avg_Income|
+-----------------+-----------------+
|          HOUSING| 74728.19354838709|
|        TRAVELLING| 57016.58490566038|
|       BOOK STORES|50903.142857142855|
|       AGRICULTURE|60372.666666666664|
|         GOLD LOAN| 70838.31506849315|
|  EDUCATIONAL LOAN| 62057.64705882353|
|        AUTOMOBILE| 68285.63157894737|
|          BUSINESS| 70246.54166666667|
|COMPUTER SOFTWARES|134376.66666666666|
|           DINNING| 67617.54545454546|
|          SHOPPING|       50466.34375|
|       RESTAURANTS|55228.794871794875|
|       ELECTRONICS| 54728.42857142857|
|          BUILDING| 69700.16666666667|
|        RESTAURANT|           64647.1|
|   HOME APPLIANCES| 58895.42857142857|
+-----------------+-----------------+
```

Explanation: Grouping organizes data into categories and applies aggregate functions (e.g., average, sum). Here, we calculate the average loan amount for each loan category. SQL groups data with GROUP BY and calculates the average with AVG().

5. Calculate Average Income by Gender:

Just now (11s) 9 Python

```python
from pyspark.sql import SparkSession

# Initialize Spark Session
spark = SparkSession.builder.appName("Local File Loading").getOrCreate()


loan_df = spark.read.csv("/FileStore/tables/loan.csv", header=True, inferSchema=True)

loan_df.createOrReplaceTempView("loan_data")
average_loan_by_gender_sql = spark.sql("""
    SELECT Gender, AVG(`Income`) AS Avg_Income
    FROM loan_data
    GROUP BY Gender
    ORDER BY Avg_Income DESC
""")
average_loan_by_gender_sql.show()
```

(4) Spark Jobs

loan_df: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 13 more fields]

average_loan_by_gender_sql: pyspark.sql.dataframe.DataFrame = [Gender: string, Avg_Income: double]

```
+------+----------------+
|Gender|      Avg_Income|
+------+----------------+
|FEMALE|76242.93779904307|
|  MALE|61961.80694980695|
+------+----------------+
```

Explanation: This query calculates the average Income for each gender and sorts the result in descending order of the average Income. Here's what each part does:

- SELECT Gender, AVG(`Income`): Retrieves the gender and the average loan amount.
- FROM loan_data: Specifies the dataset to query.
- GROUP BY Gender: Groups the rows by gender to compute averages for each group.
- ORDER BY Avg_Income DESC: Sorts the results in descending order based on the calculated average.