Case Study-3

Name: **Harish Er**

1) Loan Data File:

1. Number of loans in each category:

Just now (3s)

```
# Number of loans in each category
loan_df.groupBy("Loan Category").count().show()
```

(2) Spark Jobs

```
+------------------+-----+
|     Loan Category|count|
+------------------+-----+
|           HOUSING|   67|
|        TRAVELLING|   53|
|       BOOK STORES|    7|
|       AGRICULTURE|   12|
|         GOLD LOAN|   77|
|  EDUCATIONAL LOAN|   20|
|        AUTOMOBILE|   60|
|          BUSINESS|   24|
|COMPUTER SOFTWARES|   35|
|           DINNING|   14|
|          SHOPPING|   35|
|       RESTAURANTS|   41|
|       ELECTRONICS|   14|
|          BUILDING|    7|
|        RESTAURANT|   20|
|   HOME APPLIANCES|   14|
+------------------+-----+
```

Explanation:

- 'groupBy' groups the data based on the 'Loan_Category' column.
- 'count' calculates the number of rows in each group.

2. **Number of people with income greater than 60000 rupees:**

```
Just now (<1s)                                   3                                   Python

# Number of people with income > 60,000
loan_df.filter(loan_df["Income"] > 60000).count()

(2) Spark Jobs

Out[13]: 198
```

Explanation:

- 'filter' selects rows where 'Income' > 60000.

3. **Number of people with 2 or more returned cheques and income less than 50000:**

```
Just now (1s)                                    4                                   Python

# People with 2+ returned cheques and income < 50,000
loan_df.filter((loan_df["` Returned Cheque`"] >= 2) & (loan_df["Income"] < 50000)).count()

(2) Spark Jobs

Out[16]: 137
```

Explanation:

- Use '&' for logical AND.
- Filter rows where 'Returned_Cheques' >= 2 and 'Income' < 50000.

4. **Number of people with expenditure over 50,000 a month:**

```
Just now (<1s)                                   5                                   Python

# People with monthly expenditure > 50,000
loan_df.filter(loan_df["Expenditure"] > 50000).count()

(2) Spark Jobs

Out[20]: 6
```

Explanation:

- Filter rows where 'Expenditure' > 50000.

2) Credit File:

1. Credit card users in Spain:

```
Just now (1s)                                   7                                   Python

# Credit card users in Spain
credit_df.filter(credit_df["`Geography`"] == "Spain").count()

(2) Spark Jobs

Out[26]: 2477
```

Explanation:

- Filter rows where 'Country' is "Spain".

2. Number of members who are eligible and active in the bank:

Assuming "eligible" is determined by a CreditScore > 650

```
Just now (1s)                                   8                                   Python

# Assuming "eligible" is determined by a CreditScore > 650
credit_df.filter((credit_df["CreditScore"] > 650) & (credit_df["IsActiveMember"] == 1)).count()

(2) Spark Jobs

Out[28]: 2655
```

Explanation:

- Filter rows where 'Eligible' and 'Active' columns have "Yes".

3) Transactions File:

1. Maximum withdrawal amount:

```
Just now (1s)                                                    9                                    Python

# Maximum withdrawal amount
txn_df.select("` WITHDRAWAL AMT `").agg({"` WITHDRAWAL AMT `": "max"}).show()

(2) Spark Jobs

+--------------------+
|max( WITHDRAWAL AMT )|
+--------------------+
|        4.594475464E8|
+--------------------+
```

Explanation:

- Use 'agg' to calculate the maximum value in the 'Withdrawal_Amount' column.

2. Minimum withdrawal amount of an account:

```
Just now (1s)                                                    10                                   Python

# Minimum withdrawal amount
txn_df.select("` WITHDRAWAL AMT `").agg({"` WITHDRAWAL AMT `": "min"}).show()

(2) Spark Jobs

+--------------------+
|min( WITHDRAWAL AMT )|
+--------------------+
|                0.01|
+--------------------+
```

Explanation:

- Similar to the above but use 'min' for the minimum value.

3. Maximum deposit amount of an account:

```
Just now (1s)                                                    11                                    Python

# Maximum deposit amount
txn_df.select("` DEPOSIT AMT `").agg({"` DEPOSIT AMT `": "max"}).show()

(2) Spark Jobs

+-----------------+
|max( DEPOSIT AMT )|
+-----------------+
|          5.448E8|
+-----------------+
```

Explanation:

- Find the maximum value in the 'Deposit_Amount' column.

4. Minimum deposit amount of an account:

```
1 minute ago (1s)                                                12                                    Python

# Minimum deposit amount
txn_df.select("` DEPOSIT AMT `").agg({"` DEPOSIT AMT `": "min"}).show()

(2) Spark Jobs

+-----------------+
|min( DEPOSIT AMT )|
+-----------------+
|             0.01|
+-----------------+
```

Explanation:

- Use 'min' for the minimum deposit amount.

5. Sum of balance in every bank account:

```
1 minute ago (2s)                                                13                                    Python

from pyspark.sql import SparkSession

# Initialize Spark session
spark = SparkSession.builder \
    .appName("Sum of Balances from CSV") \
    .getOrCreate()

# Load the CSV file
file_path = "/FileStore/tables/txn.csv"
df = spark.read.csv(file_path, header=True, inferSchema=True)
total_balance = df.agg({"balance amt": "sum"}).collect()[0][0]
print(f"Total balance across all accounts: {total_balance}")

(4) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [Account No: string, TRANSACTION DETAILS: string ... 4 more fields]

Total balance across all accounts: -163245212011488.44
```

Explanation:

- Use 'agg' to calculate the sum of all values in the 'Balance' column.

6. Number of transactions on each date:

```
Just now (1s)                                14                                Python

# Number of transactions per date
txn_df.groupBy("VALUE DATE").count().show()

(2) Spark Jobs

+----------+-----+
|VALUE DATE|count|
+----------+-----+
| 23-Dec-16|  143|
|  7-Feb-19|   98|
| 21-Jul-15|   80|
|  9-Sep-15|   91|
| 17-Jan-15|   16|
| 18-Nov-17|   53|
| 21-Feb-18|   77|
| 20-Mar-18|   71|
| 19-Apr-18|   71|
| 21-Jun-16|   97|
| 17-Oct-17|  101|
|  3-Jan-18|   70|
|  8-Jun-18|  223|
| 15-Dec-18|   62|
|  8-Aug-16|   97|
| 17-Dec-16|   74|
|  3-Sep-15|   83|
| 21-Jan-16|   76|
```

Explanation:

- Group by 'Transaction_Date' and count rows in each group.

7. Customers with withdrawal amount more than 1 lakh:

```
Just now (1s)                                15                                Python

# Customers with withdrawal > 1 lakh
txn_df.filter(txn_df["` WITHDRAWAL AMT `"] > 100000).select("Account No").distinct().show()

(2) Spark Jobs

+-------------+
|   Account No|
+-------------+
|409000438611'|
|     1196711'|
|     1196428'|
|409000493210'|
|409000611074'|
|409000425051'|
|409000405747'|
|409000493201'|
|409000438620'|
|409000362497'|
+-------------+
```

Explanation:

- Filter rows where 'Withdrawal_Amount' > 100000 and select unique 'Customer_ID'.