Case Study – 3 (PySpark)

ONLINE BANKING ANALYSIS

Name: Aathirainathan P

Date: 22-11-2024

I. Loading data:

07:32 PM (5s) 1

```
#Loading Data
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Case Study").getOrCreate()

loan_file_path = "/FileStore/tables/loan.csv"
credit_card_file_path = "/FileStore/tables/credit_card.csv"
txn_file_path = "/FileStore/tables/txn.csv"

loan_df = spark.read.format("csv") \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .load(loan_file_path)

credit_df = spark.read.format("csv") \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .load(credit_card_file_path)

txn_df = spark.read.format("csv") \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .load(txn_file_path)
```

```python
# Display loaded DataFrames
loan_df.show(5)  # Display first 5 rows of loan data
credit_df.show(5)  # Display first 5 rows of credit card data
txn_df.show(5)  # Display first 5 rows of transaction data
```

▸ (9) Spark Jobs

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

▸ credit_df: pyspark.sql.dataframe.DataFrame = [RowNumber: integer, CustomerId: integer ... 11 more fields]

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

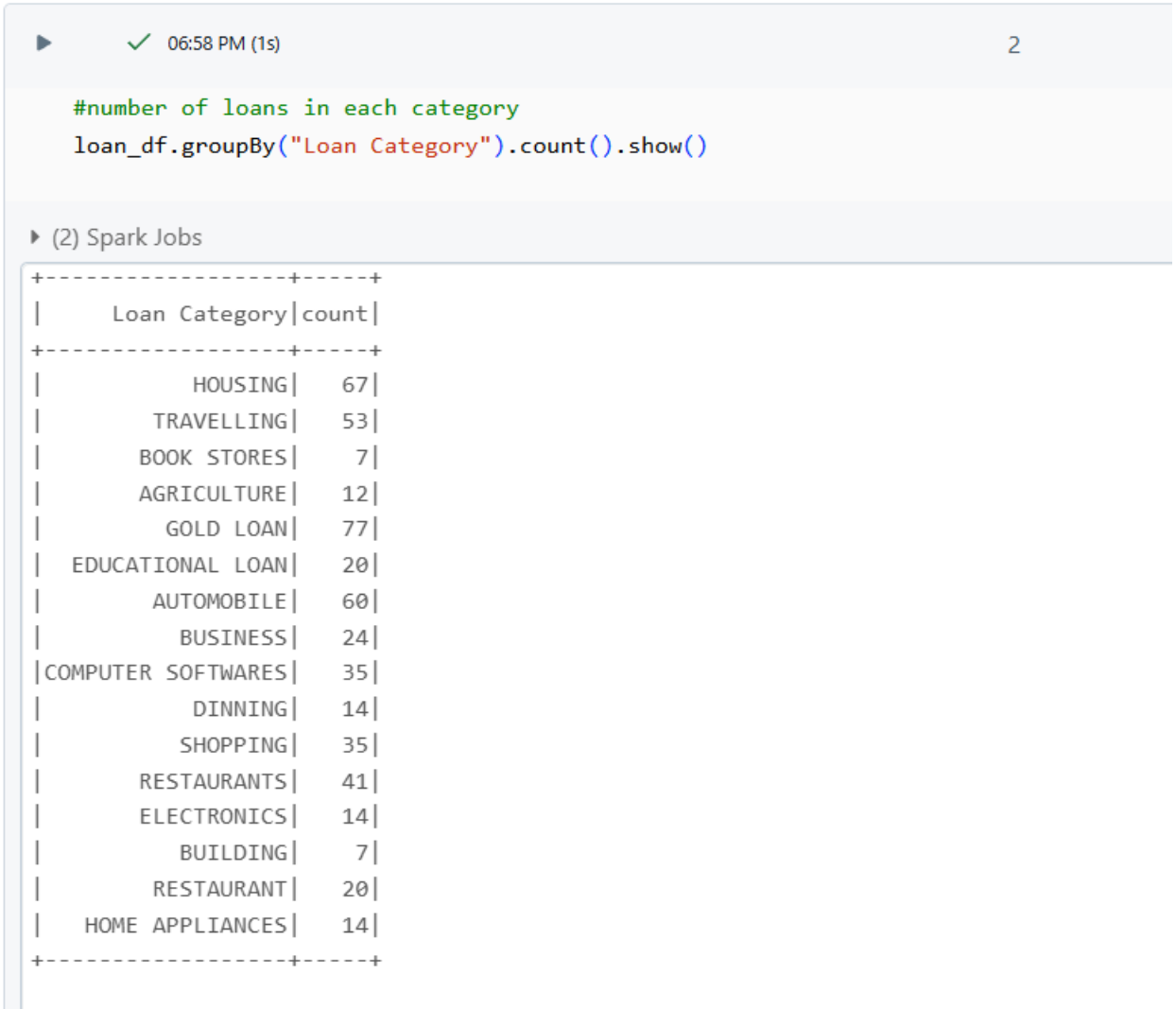
```
|RowNumber|CustomerId| Surname|CreditScore|Geography|Gender|Age|Tenure|  Balance|NumOfProducts|IsActiveMember|EstimatedSalary|Exited|
+---------+----------+--------+-----------+---------+------+---+------+---------+-------------+--------------+---------------+------+
|        1|  15634602|Hargrave|        619|   France|Female| 42|     2|      0.0|            1|             1|      101348.88|     1|
|        2|  15647311|    Hill|        608|    Spain|Female| 41|     1| 83807.86|            1|             1|      112542.58|     0|
|        3|  15619304|    Onio|        502|   France|Female| 42|     8| 159660.8|            3|             0|      113931.57|     1|
|        4|  15701354|    Boni|        699|   France|Female| 39|     1|      0.0|            2|             0|       93826.63|     0|
|        5|  15737888|Mitchell|        850|    Spain|Female| 43|     2|125510.82|            1|             1|        79084.1|     0|
+---------+----------+--------+-----------+---------+------+---+------+---------+-------------+--------------+---------------+------+
only showing top 5 rows

+-------------+--------------------+----------+----------------+-------------+-----------+
|   Account No| TRANSACTION DETAILS|VALUE DATE| WITHDRAWAL AMT | DEPOSIT AMT |BALANCE AMT|
+-------------+--------------------+----------+----------------+-------------+-----------+
|'409000611074'|TRF FROM  Indiafo...| 29-Jun-17|            NULL|    1000000.0|  1000000.0|
|'409000611074'|TRF FROM  Indiafo...|  5-Jul-17|            NULL|    1000000.0|  2000000.0|
|'409000611074'|FDRL/INTERNAL FUN...| 18-Jul-17|            NULL|     500000.0|  2500000.0|
|'409000611074'|TRF FRM  Indiafor...|  1-Aug-17|            NULL|    3000000.0|  5500000.0|
|'409000611074'|FDRL/INTERNAL FUN...| 16-Aug-17|            NULL|     500000.0|  6000000.0|
+-------------+--------------------+----------+----------------+-------------+-----------+
```

II. Questions:

A. loandata.csv file:

Execution Screenshots:

1. number of loans in each category:



06:58 PM (1s) 2

```
#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|
+------------------+-----+
```

2. number of people who have taken more than 1 lack loan:

```
07:20 PM (1s)                                   3                                   Python

#Number of people who have taken more than 1 lakh loan

from pyspark.sql.functions import col, regexp_replace

# First we Convert Loan Amount form String to Float
loan_df_cleaned = loan_df.withColumn("Loan Amount", regexp_replace(col("Loan Amount"), ",", "").cast("float"))

# Now, we use filter for people who have taken more than 1 lakh loan
ans = loan_df_cleaned.filter(col("Loan Amount") > 100000).count()

print(f"Number of people who have taken more than 1 lakh loan: {ans}")

▸ (2) Spark Jobs
▸ loan_df_cleaned:  pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 13 more fields]
Number of people who have taken more than 1 lakh loan: 450
```

3.number of people with income greater than 60000 rupees:

```
07:21 PM (1s)                                   4

#number of people with income greater than 60000 rupees

ans=loan_df.filter(col("Income") > 60000).count()
print(f"The Number of people with income greater than 60000 rupees: {ans}")

▸ (2) Spark Jobs
The Number of people with income greater than 60000 rupees: 198
```

4. number of people with 2 or more returned cheques and income less than 50000:

```
07:21 PM (1s)                                   5

# number of people with 2 or more returned cheques and income less than 50000

from pyspark.sql.functions import col

ans=loan_df.filter((col(" Returned Cheque") >= 2) & (col("Income") < 50000)).count()
print(f"The Number of people with 2 or more returned cheques and income less than 50000: {ans}")

▸ (2) Spark Jobs
The Number of people with 2 or more returned cheques and income less than 50000: 137
```

5. number of people with 2 or more returned cheques and are single:

```
07:22 PM (1s)                                                    6

from pyspark.sql.functions import col

# number of people with 2 or more returned cheques and are single
ans=loan_df.filter((col(" Returned Cheque") >= 2) & (col("Marital Status") == "SINGLE")).count()
print(f"The Number of people with 2 or more returned cheques and are single: {ans}")

(2) Spark Jobs

The Number of people with 2 or more returned cheques and are single: 111
```

6. number of people with expenditure over 50000 a month:

```
07:25 PM (1s)                                                    7

#number of people with expenditure over 50000 a month

ans=loan_df.filter(col("Expenditure") > 50000).count()
print(f"The number of people with expenditure over 50000 a month: {ans}")

(2) Spark Jobs

The number of people with expenditure over 50000 a month: 6
```

Execution Summary:

The queries on loandata.csv used PySpark's DataFrame API to filter and aggregate data. Key functions included:

- **filter()**: Applied to select rows based on conditions (e.g., income > 60,000, loan amount > 100,000).
- **groupBy()**: Grouped data by categories (e.g., loan category) to compute aggregate values.
- **count()**: Used to count rows that met specific criteria (e.g., number of people with high loans or returned cheques).
- **agg()**: Aggregated data for summary statistics (e.g., sum, count) after grouping.
- **alias()**: Renamed columns for clarity in the final output.

These functions allowed us to extract insights about loan distribution, income, and customer behaviour.

B.credit.csv file:

Execution Screenshots:

1.credit card users in Spain:

```
07:33 PM (1s)                                                    8

#credit card users in Spain

ans=credit_df.filter(col("Geography") == "Spain").count()
print(f"Total Credit card users in Spain: {ans}")

(2) Spark Jobs

Total Credit card users in Spain: 2477
```

2.number of members who are elgible and active in the bank:

```
07:36 PM (1s)                                                    9

#number of members who are elgible and active in the bank

ans=credit_df.filter((col("IsActiveMember") == 1)).count()
print(f"The number of members who are elgible and active in the bank: {ans}")

(2) Spark Jobs

The number of members who are elgible and active in the bank: 5151
```

Execution Summary:

The queries on credit.csv used PySpark's DataFrame operations for filtering and counting. Key functions included:

- **filter()**: Applied conditions to select active members and credit card eligibility based on specific criteria (e.g., IsActiveMember == 1).
- **count()**: Counted the number of records satisfying the filter conditions, such as the number of eligible active bank members.
- **groupBy() and agg()**: Grouped and aggregated data when required for summarization.

These operations helped us analyze the credit card usage, member activity, and eligibility within the dataset.

C. Transaction file(txn.csv):

Execution Screenshots:

1. Maximum withdrawal amount in transactions:

07:39 PM (1s) 10

```
#Maximum withdrawal amount in transactions
from pyspark.sql.functions import max

max_withdrawal = txn_df.select(max(col(" WITHDRAWAL AMT "))).collect()[0][0]
print(f"Maximum withdrawal amount in transactions: {max_withdrawal}")
```

(2) Spark Jobs

```
Maximum withdrawal amount in transactions: 459447546.4
```

2. MINIMUM WITHDRAWAL AMOUNT OF AN ACCOUNT in txn.csv:

07:40 PM (1s) 11

```
#MINIMUM WITHDRAWAL AMOUNT OF AN ACCOUNT in txn.csv
from pyspark.sql.functions import min

min_withdrawal = txn_df.select(min(col(" WITHDRAWAL AMT "))).collect()[0][0]
print(f"Minimum withdrawal amount in transactions: {min_withdrawal}")
```

(2) Spark Jobs

```
Minimum withdrawal amount in transactions: 0.01
```

3. MAXIMUM DEPOSIT AMOUNT OF AN ACCOUNT:

07:42 PM (1s) 12

```
#MAXIMUM DEPOSIT AMOUNT OF AN ACCOUNT

max_deposit = txn_df.select(max(col(" DEPOSIT AMT "))).collect()[0][0]
print(f"Maximum deposit amount in transactions: {max_deposit}")
```

(2) Spark Jobs

```
Maximum deposit amount in transactions: 544800000.0
```

4. MINIMUM DEPOSIT AMOUNT OF AN ACCOUNT:

07:43 PM (1s) 13

```
#MINIMUM DEPOSIT AMOUNT OF AN ACCOUNT

min_deposit = txn_df.select(min(col(" DEPOSIT AMT "))).collect()[0][0]
print(f"Minimum deposit amount in transactions: {min_deposit}")
```

(2) Spark Jobs

```
Minimum deposit amount in transactions: 0.01
```

5. Number of transactions on each date:

07:50 PM (1s) 14 Python

```
#Number of transaction on each date

print("Number of transactions on each date:")
txn_count_by_date = txn_df.groupBy("VALUE DATE").count().show()
```

(2) Spark Jobs

```
Number of transactions on each date:
+----------+-----+
|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|
```

6.List of customers with withdrawal amount more than 1 lakh

07:51 PM (2s) 15 Python

```
#List of customers with withdrawal amount more than 1 lakh

print("List of customers with withdrawal amount more than 1 lakh:")
customers_with_high_withdrawals = txn_df.filter(col(" WITHDRAWAL AMT ") > 100000).select("Account No").distinct().show()
```

(2) Spark Jobs

```
List of customers with withdrawal amount more than 1 lakh:
+--------------+
|   Account No|
+--------------+
|409000438611'|
|     1196711'|
|     1196428'|
|409000493210'|
|409000611074'|
|409000425051'|
|409000405747'|
|409000493201'|
|409000438620'|
|409000362497'|
+--------------+
```

Execution Summary:

The queries on txn.csv involved using PySpark DataFrame operations for aggregating and analyzing transaction data. The key functions used include:

- **groupBy()**: Grouped the data by Account No to perform aggregations like sum and count.
- **agg()**: Applied aggregation functions like sum() to calculate total balances and other metrics for each account.
- **filter()**: Filtered data based on conditions like withdrawal amounts and dates.
- **show()**: Displayed the results in a readable format.

These functions enabled us to compute aggregate transaction metrics, such as maximum withdrawal, deposit amounts, and total balances for each account, as well as track transactions by date and customer.

Submitted By:
Aathirainathan P