CASE STUDY - ONLINE BANKING ANALYSIS

Import libraries & Initiate session

2 minutes ago (<1s) 3

```
# initialize the session
from pyspark import SparkContext
from pyspark.sql import SparkSession

sc = SparkContext.getOrCreate()
spark = SparkSession.builder.appName('Case study program').getOrCreate()
```

Upload dataset

2 minutes ago (3s) 5

```
data_credit =spark.read.csv("/FileStore/tables/creditCard.csv",inferSchema=True,header=True)
data_txn =spark.read.csv("/FileStore/tables/txn.csv",inferSchema=True,header=True)
data_loan =spark.read.csv("/FileStore/tables/bankloan.csv",inferSchema=True,header=True)
```

(6) Spark Jobs

```
data_credit: pyspark.sql.dataframe.DataFrame = [RowNumber: integer, CustomerId: integer ... 11 more fields]
data_txn: pyspark.sql.dataframe.DataFrame = [Account No: string, TRANSACTION DETAILS: string ... 4 more fields]
data_loan: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 13 more fields]
```

Exploring data

Loan Data

2 minutes ago (<1s)

```
# Print Schema
data_loan.printSchema()
```

```
root
 |-- Customer_ID: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Occupation: string (nullable = true)
 |-- Marital Status: string (nullable = true)
 |-- Family Size: integer (nullable = true)
 |-- Income: integer (nullable = true)
 |-- Expenditure: integer (nullable = true)
 |-- Use Frequency: integer (nullable = true)
 |-- Loan Category: string (nullable = true)
 |-- Loan Amount: string (nullable = true)
 |-- Overdue: integer (nullable = true)
 |--  Debt Record: string (nullable = true)
 |--  Returned Cheque: integer (nullable = true)
 |--  Dishonour of Bill: integer (nullable = true)
```

Transaction data

4 minutes ago (<1s)

```
# Print Schema
data_txn.printSchema()
```

```
root
 |-- Account No: string (nullable = true)
 |-- TRANSACTION DETAILS: string (nullable = true)
 |-- VALUE DATE: string (nullable = true)
 |--  WITHDRAWAL AMT : double (nullable = true)
 |--  DEPOSIT AMT : double (nullable = true)
 |-- BALANCE AMT: double (nullable = true)
```

Credit data

07:53 PM (<1s)

```
# Print Schema
data_credit.printSchema()
```

```
root
 |-- RowNumber: integer (nullable = true)
 |-- CustomerId: integer (nullable = true)
 |-- Surname: string (nullable = true)
 |-- CreditScore: integer (nullable = true)
 |-- Geography: string (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Tenure: integer (nullable = true)
 |-- Balance: double (nullable = true)
 |-- NumOfProducts: integer (nullable = true)
 |-- IsActiveMember: integer (nullable = true)
 |-- EstimatedSalary: double (nullable = true)
 |-- Exited: integer (nullable = true)
```

In LOAN DATA

1. number of loans in each category

07:53 PM (1s)

```
data_loan.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|
+------------------+-----+
```

07:53 PM (1s)

```
clean_loan.groupBy("Loan Category").count().show()
```

(2) Spark Jobs

```
+------------------+-----+
|     Loan Category|count|
+------------------+-----+
|           HOUSING|   61|
|        TRAVELLING|   48|
|       BOOK STORES|    7|
|       AGRICULTURE|   12|
|         GOLD LOAN|   72|
|  EDUCATIONAL LOAN|   17|
|        AUTOMOBILE|   53|
|          BUSINESS|   24|
|COMPUTER SOFTWARES|   25|
|           DINNING|   11|
|          SHOPPING|   30|
|       RESTAURANTS|   37|
|       ELECTRONICS|   13|
|          BUILDING|    6|
|        RESTAURANT|   20|
|   HOME APPLIANCES|   13|
+------------------+-----+
```

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

07:53 PM (1s) 35

```
from pyspark.sql.functions import col, regexp_replace
# since here , present in loan amount column we are replacing the comma
# then cast it as integer
# Remove commas and cast the Loan Amount column to integer
loan_with_null_cast = data_loan.withColumn(
    "Loan Amount",
    regexp_replace(col("Loan Amount"), ",", "").cast("int")
)
loan_with_null_cast.printSchema()
loan_02 = loan_with_null_cast.filter(col("Loan Amount") > 100000)
num_rows111 = loan_02.count()
print(f"Number of people taken more then 1 lakh in raw data: {num_rows111}")
```

```
▸ (2) Spark Jobs
▸ loan_with_null_cast: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 13 more fields]
▸ loan_02: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 13 more fields]
root
 |-- Customer_ID: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Occupation: string (nullable = true)
 |-- Marital Status: string (nullable = true)
 |-- Family Size: integer (nullable = true)
 |-- Income: integer (nullable = true)
 |-- Expenditure: integer (nullable = true)
 |-- Use Frequency: integer (nullable = true)
 |-- Loan Category: string (nullable = true)
 |-- Loan Amount: integer (nullable = true)
 |-- Overdue: integer (nullable = true)
 |--  Debt Record: string (nullable = true)
 |--  Returned Cheque: integer (nullable = true)
 |--  Dishonour of Bill: integer (nullable = true)

Number of people taken more then 1 lakh in raw data: 450
```

```
▸ loan_clean_cast: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 13 more fields]
▸ loan_04: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 13 more fields]
root
 |-- Customer_ID: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Occupation: string (nullable = true)
 |-- Marital Status: string (nullable = true)
 |-- Family Size: integer (nullable = true)
 |-- Income: integer (nullable = true)
 |-- Expenditure: integer (nullable = true)
 |-- Use Frequency: integer (nullable = true)
 |-- Loan Category: string (nullable = true)
 |-- Loan Amount: integer (nullable = true)
 |-- Overdue: integer (nullable = true)
 |--  Debt Record: string (nullable = true)
 |--  Returned Cheque: integer (nullable = true)
 |--  Dishonour of Bill: integer (nullable = true)

Number of people taken more then 1 lakh in clean data: 409
```

Here we are changing the datatype of Loan Amount from string to Integer for performing aggregate functions.

3. number of people with income greater than 60000 rupees

07:53 PM (<1s) 38

```
loan_05 = data_loan.filter(col("Income") > 60000)
num_rows114 = loan_05.count()
print(f"Number of people with income greater than 60000 rupees on raw data: {num_rows114}")
```

(2) Spark Jobs

loan_05: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 13 more fields]

```
Number of people with income greater than 60000 rupees on raw data: 198
```

07:53 PM (<1s) 39

```
loan_06 = clean_loan.filter(col("Income") > 60000)
num_rows115 = loan_06.count()
print(f"Number of people with income greater than 60000 rupees on clean data: {num_rows115}")
```

(2) Spark Jobs

loan_06: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 13 more fields]

```
Number of people with income greater than 60000 rupees on clean data: 192
```

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

07:53 PM (1s) 41

```
loan_07 = data_loan.filter((col(" Returned Cheque") >=2)&(col("Income") <50000))
num_rows007 = loan_07.count()
print(f"No of people with returned cheq>=2 & salary<50000data (raw): {num_rows007}")
```

(2) Spark Jobs

loan_07: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 13 more fields]

```
No of people with returned cheq>=2 & salary<50000data (raw): 137
```

07:53 PM (<1s) 42

```
loan_08 = clean_loan.filter((col(" Returned Cheque") >=2)&(col("Income") <50000))
num_rows008 = loan_08.count()
print(f"No of people with returned cheq>=2 & salary<50000data (clean): {num_rows008}")
```

(2) Spark Jobs

loan_08: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 13 more fields]

```
No of people with returned cheq>=2 & salary<50000data (clean): 132
```

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

07:53 PM (<1s) 44

```
loan_09 = data_loan.filter((col(" Returned Cheque") >= 2) & (col("Marital Status") == "Single"))
num_rows009 = loan_09.count()
print(f"No of people with returned cheq>=2 & single: {num_rows009}")
```

(2) Spark Jobs

loan_09: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 13 more fields]

```
No of people with returned cheq>=2 & single: 0
```

6. number of people with expenditure over 50000 a month

07:53 PM (<1s) 46

```
loan_10 = data_loan.filter(col("Expenditure") >50000)
num_rows010 = loan_10.count()
print(f"No of people with expenditure>50000 (raw): {num_rows010}")
```

(2) Spark Jobs

loan_10: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 13 more fields]

```
No of people with expenditure>50000 (raw): 6
```

+ Code + Text

07:53 PM (<1s) 47

```
loan_11 = clean_loan.filter(col("Expenditure") >50000)
num_rows011 = loan_11.count()
print(f"No of people with expenditure>50000(clean): {num_rows011}")
```

(2) Spark Jobs

loan_11: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 13 more fields]

```
No of people with expenditure>50000(clean): 6
```

7. number of members who are elgible for credit card

07:53 PM (<1s) 49

```
eligible_customers1 = data_loan.filter(
    (col("Income") > 20000) &
    (col(" Returned Cheque") == 0) &  # No returned cheques
    (col(" Dishonour of Bill") == 0)
)
# Count the number of eligible members
eligible_count1 = eligible_customers1.count()
print(f"No of people eligible for loan (raw): {eligible_count1}")
```

(2) Spark Jobs

eligible_customers1: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 13 more fields]

```
No of people eligible for loan (raw): 3
```

07:53 PM (1s) 50

```
eligible_customers2 = clean_loan.filter(
    (col("Income") > 20000) &
    (col(" Returned Cheque") == 0) &
    (col(" Dishonour of Bill") == 0)
)
# Count the number of eligible members
eligible_count2 = eligible_customers2.count()
print(f"No of people eligible for loan (clean): {eligible_count2}")
```

(2) Spark Jobs

eligible_customers2: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 13 more fields]

```
No of people eligible for loan (clean): 2
```

In credit.csv file

1. credit card users in Spain

07:53 PM (<1s) 53

```
credit_01 = data_credit.filter(col("Geography") =='Spain')
count01 = credit_01.count()
print(f"No of Credit card users in Spain: {count01}")
```

(2) Spark Jobs

credit_01: pyspark.sql.dataframe.DataFrame = [RowNumber: integer, CustomerId: integer ... 11 more fields]

```
No of Credit card users in Spain: 2477
```

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

07:53 PM (<1s) 55

```
#Works based on certain assumptions
eligible_active_customers = data_credit.filter(
    (col("CreditScore") >= 600) &        # Credit score threshold
    (col("Balance") > 0) &               # Non-zero balance
    (col("EstimatedSalary") >= 20000) & # Minimum salary threshold
    (col("Exited") == 0) &               # Customer has not exited
    (col("IsActiveMember") == 1)         # Customer is active
)

# Count the number of eligible and active customers
eligible_active_count = eligible_active_customers.count()

print(f"Number of eligible and active customers: {eligible_active_count}")
```

(2) Spark Jobs

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

```
Number of eligible and active customers: 1803
```

In Transactions file

1. Maximum withdrawal amount in transactions Minimum withdrawal amount of an account

07:53 PM (1s) 58 Python

```python
from pyspark.sql.functions import col, max, min

# Use PySpark's max and min functions correctly
withdrawal_stats = txn_all_filled.agg(
    max(col(" WITHDRAWAL AMT ")).alias("MaxWithdrawal"),
    min(col(" WITHDRAWAL AMT ")).alias("MinWithdrawal")
)
display(withdrawal_stats)
```

(2) Spark Jobs

withdrawal_stats: pyspark.sql.dataframe.DataFrame = [MaxWithdrawal: double, MinWithdrawal: double]

Table

	1.2 MaxWithdrawal	1.2 MinWithdrawal
1	459447546.4	0

2. maximum deposit amount of an account

07:53 PM (1s)

```python
from pyspark.sql.functions import col, max, min

# Use PySpark's max and min functions correctly
deposit_stats = txn_all_filled.agg(
    max(col(" DEPOSIT AMT ")).alias("MaxDeposit"),
)
display(deposit_stats)
```

(2) Spark Jobs

deposit_stats: pyspark.sql.dataframe.DataFrame = [MaxDeposit: double]

Table

	1.2 MaxDeposit
1	544800000

3. minimum deposit amount of an account

07:53 PM (1s)

```
from pyspark.sql.functions import col, max, min

# Use PySpark's max and min functions correctly
deposit_stats1 = txn_all_filled.agg(
    min(col(" DEPOSIT AMT ")).alias("MinDeposit"),
)
display(deposit_stats1)
```

(2) Spark Jobs

deposit_stats1: pyspark.sql.dataframe.DataFrame = [MinDeposit: double]

Table

	1.2 MinDeposit
1	0

4. sum of balance in every bank account

07:53 PM (1s) 64

```
from pyspark.sql.functions import col, sum

# Group by "Account No" and calculate the sum of "BALANCE AMT"
balance_sum = txn_all_filled.groupBy("Account No").agg(
    sum(col("BALANCE AMT")).alias("TotalBalance")
)
display(balance_sum)
```

(2) Spark Jobs

balance_sum: pyspark.sql.dataframe.DataFrame = [Account No: string, TotalBalance: double]

Table

	Account No	TotalBalance
1	409000438611'	-2494865770683.3955
2	1196711'	-16047649810127.5
3	1196428'	-81418498130721
4	409000493210'	-3275849521320.9575
5	409000611074'	1615533622
6	409000425051'	-3772118411.6499877
7	409000405747'	-24310804706.700016

5. Number of transaction on each date

07:53 PM (1s) 66

```
from pyspark.sql.functions import col, count

# Group by "VALUE DATE" and calculate the COUNT
txn_c = txn_all_filled.groupBy("VALUE DATE").    count().alias("Transaction count")
display(txn_c)
```

(2) Spark Jobs

txn_c: pyspark.sql.dataframe.DataFrame = [VALUE DATE: string, count: long]

Table

	VALUE DATE	count
1	23-Dec-16	143
2	7-Feb-19	98
3	21-Jul-15	80
4	9-Sep-15	91
5	17-Jan-15	16
6	18-Nov-17	53
7	21-Feb-18	77
8	20-Mar-18	71

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

07:53 PM (1s) 68

```
cust = txn_all_filled.filter(col(" WITHDRAWAL AMT ") >100000.0)
display(cust)
```

(1) Spark Jobs

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

Table

	Account No	TRANSACTION DETAILS	VALUE DATE	WITHDRAWAL AMT	DEPOSIT AMT	BALANCE AMT
1	409000611074'	INDO GIBL Indiaforensic STL01071	16-Aug-17	133900	0	8366100
2	409000611074'	INDO GIBL Indiaforensic STL04071	16-Aug-17	195800	0	8147300
3	409000611074'	INDO GIBL Indiaforensic STL10071	16-Aug-17	143800	0	7781600
4	409000611074'	INDO GIBL Indiaforensic STL11071	16-Aug-17	331650	0	7449950
5	409000611074'	INDO GIBL Indiaforensic STL12071	16-Aug-17	129000	0	7320950
6	409000611074'	INDO GIBL Indiaforensic STL13071	16-Aug-17	230013	0	7090937
7	409000611074'	INDO GIBL Indiaforensic STL14071	16-Aug-17	367900	0	6723037
8	409000611074'	INDO GIBL Indiaforensic STL15071	16-Aug-17	108000	0	6615037
9	409000611074'	INDO GIBL Indiaforensic STL17071	16-Aug-17	141000	0	6409237
10	409000611074'	INDO GIBL Indiaforensic STL22071	16-Aug-17	206000	0	5959817