PySpark Case Study

-Sarthak Niranjan Kulkarni (Maverick)

- [sarthakkul2311@gmail.com](mailto:sarthakkul2311@gmail.com) - (+91) 93256 02791

**25/11/2024 (Monday)**

**1. Loandata.csv file**

**1. Number of loans in each category:**

🡪 # Group by 'Loan Category' and count the number of loans in each category

loan\_category\_counts = loan\_df.groupBy("Loan Category").count()

# Show the result

loan\_category\_counts.show()

A screenshot of a computer

Description automatically generated

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

🡪 from pyspark.sql.functions import col

# Filter the rows where 'Income' is greater than 60,000

people\_above\_60k = loan\_df.filter(col("Income") > 60000)

# Count the number of such people

num\_people\_above\_60k = people\_above\_60k.count()

# Display the result

print(f"Number of people with income greater than 60,000 rupees: {num\_people\_above\_60k}")

A screenshot of a computer

Description automatically generated

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

🡪 from pyspark.sql.functions import col

# Remove any leading/trailing spaces from the column names if necessary

loan\_df\_cleaned = loan\_df.select([col(c).alias(c.strip()) for c in loan\_df.columns])

# Filter the rows where 'Returned Cheque' >= 2 and 'Income' < 50,000

people\_filtered = loan\_df\_cleaned.filter((col("Returned Cheque") >= 2) & (col("Income") < 50000))

# Count the number of such people

num\_people\_filtered = people\_filtered.count()

# Display the result

print(f"Number of people with 2 or more returned cheques and income less than 50,000 rupees: {num\_people\_filtered}")

A white background with black text

Description automatically generated

**4. Number of people with 2 or more returned cheques and are single**

🡪 from pyspark.sql.functions import col

# Remove leading/trailing spaces from column names if needed

loan\_df\_cleaned = loan\_df.select([col(c).alias(c.strip()) for c in loan\_df.columns])

# Filter the rows where 'Returned Cheque' >= 2 and 'Marital Status' is 'SINGLE'

people\_filtered = loan\_df\_cleaned.filter((col("Returned Cheque") >= 2) & (col("Marital Status") == "SINGLE"))

# Count the number of such people

num\_people\_filtered = people\_filtered.count()

# Display the result

print(f"Number of people with 2 or more returned cheques and are single: {num\_people\_filtered}")

A screenshot of a computer

Description automatically generated

**5. Number of people with expenditure over 50000 a month**

🡪 from pyspark.sql.functions import col

# Filter the rows where 'Expenditure' > 50,000

people\_filtered = loan\_df.filter(col("Expenditure") > 50000)

# Count the number of such people

num\_people\_filtered = people\_filtered.count()

# Display the result

print(f"Number of people with expenditure over 50,000 a month: {num\_people\_filtered}")



**6. Number of members who are elgible for credit card**

🡪 from pyspark.sql.functions import col

# Filter the rows where 'Income' > 30,000 and 'Overdue' == 0 (eligible for credit card)

eligible\_members = loan\_df.filter((col("Income") > 30000) & (col("Overdue") == 0))

# Count the number of eligible members

num\_eligible\_members = eligible\_members.count()

# Display the result

print(f"Number of members eligible for a credit card: {num\_eligible\_members}")



**2. Credit.csv File**

1. **Credit card users in Spain**

🡪 from pyspark.sql.functions import col

# Filter the DataFrame for users in Spain with at least one product (credit card users)

credit\_card\_users\_spain = credit\_df.filter((col("Geography") == "Spain") & (col("NumOfProducts") > 0))

# Count the number of credit card users in Spain

num\_credit\_card\_users\_spain = credit\_card\_users\_spain.count()

# Display the result

print(f"Number of credit card users in Spain: {num\_credit\_card\_users\_spain}")



1. **Number of members who are eligible and active in the bank**

🡪 from pyspark.sql.functions import col

# Filter the DataFrame for eligible and active members

eligible\_active\_members = credit\_df.filter((col("NumOfProducts") > 0) & (col("IsActiveMember") == 1))

# Count the number of eligible and active members

num\_eligible\_active\_members = eligible\_active\_members.count()

# Display the result

print(f"Number of eligible and active members: {num\_eligible\_active\_members}")



**3. Txn.csv File: -**

1. **Maximum withdrawal amount in transactions**

🡪 from pyspark.sql.functions import col

# Find the maximum withdrawal amount

max\_withdrawal = txn\_df.agg({" WITHDRAWAL AMT ": "max"}).collect()[0][0]

# Show the result

print("Maximum Withdrawal Amount:", max\_withdrawal)



1. **Minimum withdrawal amount of an account in txn.csv**

🡪 from pyspark.sql.functions import col, min

# Find the minimum withdrawal amount grouped by Account No

min\_withdrawal = txn\_df.groupBy("Account No").agg(min(" WITHDRAWAL AMT ").alias("Min Withdrawal")).show()

A screenshot of a computer screen

Description automatically generated

1. **Maximum deposit amount of an account**

🡪 from pyspark.sql.functions import col, max

# Find the maximum deposit amount grouped by Account No

max\_deposit = txn\_df.groupBy("Account No").agg(max(" DEPOSIT AMT ").alias("Max Deposit")).show()

A screenshot of a computer

Description automatically generated

1. **Minimum deposit amount of an account**

🡪 from pyspark.sql.functions import col, min

# Find the minimum deposit amount grouped by Account No

min\_deposit = txn\_df.groupBy("Account No").agg(min(" DEPOSIT AMT ").alias("Min Deposit")).show()

A screenshot of a computer screen

Description automatically generated

1. **Sum of balance in every bank account**

🡪 from pyspark.sql.functions import sum

# Find the sum of balance grouped by Account No

sum\_balance = txn\_df.groupBy("Account No").agg(sum("BALANCE AMT").alias("Total Balance")).show()

A screenshot of a computer screen

Description automatically generated

1. **Number of transaction on each date**

🡪 from pyspark.sql.functions import col

# Find the number of transactions grouped by VALUE DATE

transaction\_count\_per\_date = txn\_df.groupBy("VALUE DATE").count().alias("Number of Transactions").show()

A screenshot of a computer

Description automatically generated

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

🡪 from pyspark.sql.functions import col

# Filter the data for withdrawal amount greater than 1 lakh

customers\_with\_high\_withdrawal = txn\_df.filter(col(" WITHDRAWAL AMT ") > 100000).select("Account No", "TRANSACTION DETAILS", "VALUE DATE", " WITHDRAWAL AMT ").show()

A screenshot of a computer

Description automatically generated

**4. Summary of Case Study: -**

1. **Reading Data**:
   * PySpark's spark.read.option is used to load CSV data into a DataFrame with headers enabled.
2. **Grouping and Aggregation**:
   * groupBy and count functions are applied to analyze loan data by categories and transaction counts by dates.
3. **Filtering Data**:
   * The filter function is extensively used to extract subsets of data based on conditions, such as income thresholds, marital status, and overdue payments.
4. **Column Operations**:
   * The col function allows dynamic column references, while column names are cleaned using .alias() for consistent processing.
5. **Aggregation Functions**:
   * Functions like max, min, sum, and agg are used to calculate metrics for transactional data (e.g., maximum withdrawals and total balances).
6. **Efficient Counting**:
   * count() is applied to count rows meeting specific criteria, such as active members, high expenditures, or high-value transactions.
7. **DataFrames and SQL Integration**:
   * PySpark seamlessly combines SQL-like operations (select, groupBy, and filtering) with DataFrame-based data manipulation for scalable analysis.