**PYSPARK CASE STUDY**

**Name: S. Nikhil Sai**

**Date: 25-11-2024**

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('pyspark-case\_study').getOrCreate()

credit =spark.read.csv("/FileStore/tables/credit\_card.csv", header=True, inferSchema=True)  #original table name = credid\_card.csv

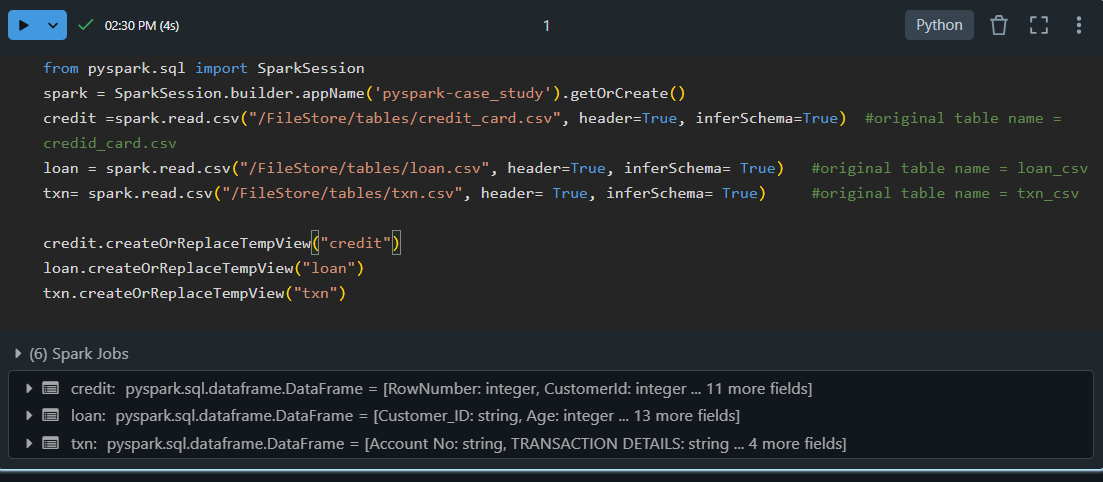
loan = spark.read.csv("/FileStore/tables/loan.csv", header=True, inferSchema= True)   #original table name = loan\_csv

txn= spark.read.csv("/FileStore/tables/txn.csv", header= True, inferSchema= True)     #original table name = txn\_csv

credit.createOrReplaceTempView("credit")

loan.createOrReplaceTempView("loan")

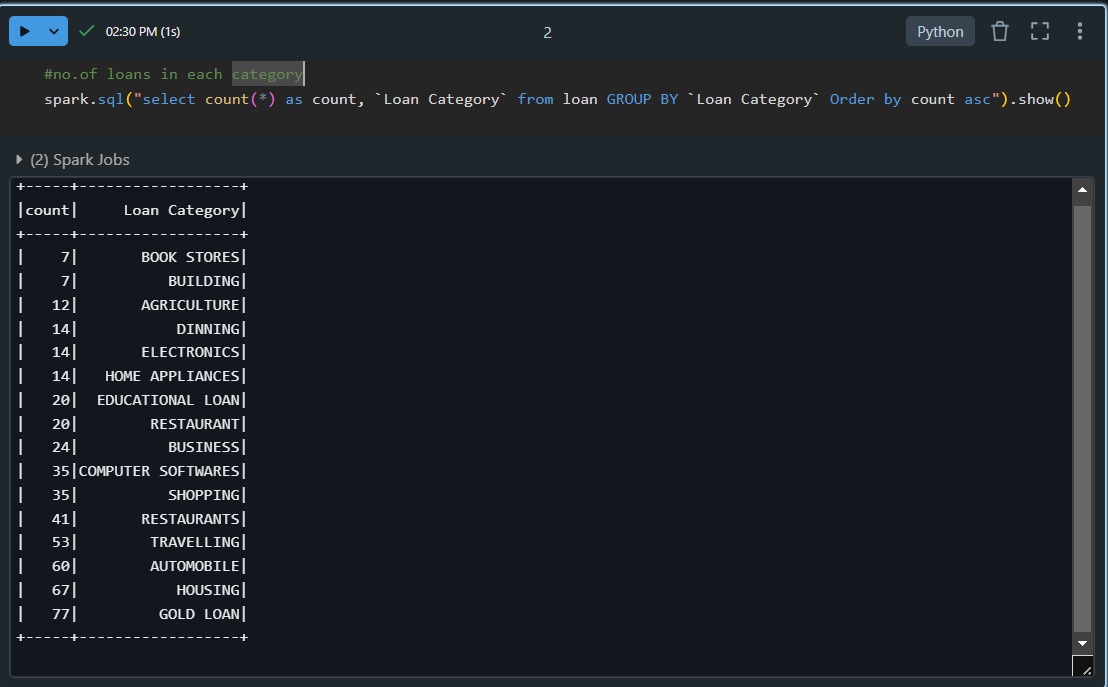
txn.createOrReplaceTempView("txn")

****

**Explanation:** This code sets up a PySpark session named 'pyspark-case\_study' and reads three CSV files (credit\_card.csv, loan.csv, and txn.csv) into Spark DataFrames. The header=True ensures the first row is used as column names, and inferSchema=True detects data types automatically. Each DataFrame is registered as a temporary SQL view (credit, loan, and txn) for executing SQL queries. These views allow seamless integration of Spark SQL for querying the loaded data. This setup enables efficient distributed data processing and analysis.

#no.of loans in each category

spark.sql("select count(\*) as count, `Loan Category` from loan GROUP BY `Loan Category` Order by count asc").show()



**Explanation:** This Spark SQL query calculates the count of records for each unique "Loan Category" in the loan table. It groups the data by the Loan Category column and orders the results in ascending order based on the count of records. The result shows two columns: the count of each loan category and the corresponding loan category itself. The show() function then displays the result as a DataFrame. This is typically used to analyze the distribution of loan categories in the dataset.

#number of people with Loan amount than 100000 rupees

onelakh\_loan= loan.filter(loan["Loan Amount"] > '100000').count()

print('people who took loan greather than 1 lakh: ', onelakh\_loan)

A computer screen shot of a code

Description automatically generated

**Explanation:** This code filters the loan DataFrame to include only the rows where the "Loan Amount" is greater than 100,000 rupees. It then counts the number of such records using the count() function. The result, which represents the number of people who took loans greater than 1 lakh, is printed with the message 'people who took loan greater than 1 lakh: '.

#number of people with income greater than 60000 rupees

sixtyk\_income= loan.filter(loan["Income"] > 60000).count()

print('people whose income is greather than 60000 rupees: ', sixtyk\_income)

A screen shot of a computer

Description automatically generated

**Explanation:** This code filters the loan DataFrame to select the rows where the "Income" is greater than 60,000 rupees. It then counts the number of such records using the count() function. The result, which represents the number of people whose income is greater than 60,000 rupees, is printed with the message 'people whose income is greater than 60000 rupees: '

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

less\_than\_fiftyk= loan.filter((loan[" Returned Cheque"]>=2) & (loan["Income"]<50000)).count()

print('people with 2 or more returned cheques and income less than 50000:',less\_than\_fiftyk)

A screenshot of a computer

Description automatically generated

**Explanation:** This code filters the loan DataFrame to select rows where the "Returned Cheque" is greater than or equal to 2 and the "Income" is less than 50,000 rupees. It then counts the number of such records using the count() function. The result, which represents the number of people with 2 or more returned cheques and an income less than 50,000 rupees, is printed with the message 'people with 2 or more returned cheques and income less than 50000:'. The conditions are combined using the logical AND (&) operator.

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

single\_and\_Returned\_cheque= loan.filter((loan[" Returned Cheque"]>=2) & (loan["Marital Status"]=='SINGLE')).count()

print('number of people with 2 or more returned cheques and are single:',single\_and\_Returned\_cheque)

A computer screen shot of a computer code

Description automatically generated

**Explanation:** This code filters the loan DataFrame to select rows where the "Returned Cheque" is greater than or equal to 2 and the "Marital Status" is 'SINGLE'. It then counts the number of such records using the count() function. The result, which represents the number of people who have 2 or more returned cheques and are single, is printed with the message 'number of people with 2 or more returned cheques and are single:'. The conditions are combined using the logical AND (&) operator.

#number of people with expenditure over 50000 a month",

exp\_over\_fivek=loan.filter(loan["Expenditure"]>50000).count()

print("number of people with expenditure over 50000 a month:", exp\_over\_fivek)

A screenshot of a computer program

Description automatically generated

**Explanation:** This code filters the loan DataFrame to select rows where the "Expenditure" is greater than 50,000. It then counts the number of such records using the count() function. The result, which represents the number of people with an expenditure greater than 50,000 a month, is printed with the message 'number of people with expenditure over 50000 a month:'. This allows you to analyze the number of people with high monthly expenditures in the dataset.

#number of members who are elgible for credit card

credit\_eligible=credit.filter(credit["CreditScore"]>750).count()

print("number of members who are elgible for credit card:", credit\_eligible)

A screenshot of a computer

Description automatically generated

**Explanation:** This code filters the credit DataFrame to select rows where the "CreditScore" is greater than 750, assuming that a credit score above 750 qualifies someone for a credit card. It then counts the number of such records using the count() function. The result, which represents the number of members eligible for a credit card, is printed with the message 'number of members who are eligible for credit card:'. This helps identify individuals who meet the credit score criteria for eligibility.

#credit card users in Spain

spain\_users=credit.filter(credit["Geography"]=='Spain').count()

print("Total credit card users in Spain:", spain\_users)

A computer screen shot of a computer code

Description automatically generated

**Explanation:** This code filters the credit DataFrame to select rows where the "Geography" column is equal to 'Spain', identifying the credit card users based in Spain. It then counts the number of such records using the count() function. The result, which represents the total number of credit card users in Spain, is printed with the message 'Total credit card users in Spain:'. This helps to analyze the number of credit card users specifically in Spain within the dataset.

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

active\_credit= credit.filter((credit["CreditScore"]>750) & (credit["IsActiveMember"]=='1')).count()

print('number of members who are elgible for credit card and active in the bank:',active\_credit)

A screenshot of a computer program

Description automatically generated

**Explanation:** This code filters the credit DataFrame to select rows where the "CreditScore" is greater than 750 (indicating eligibility for a credit card) and the "IsActiveMember" is '1' (indicating active membership in the bank). It then counts the number of such records using the count() function. The result, which represents the number of members eligible for a credit card and active in the bank, is printed with the message 'number of members who are eligible for credit card and active in the bank:'. This helps identify individuals who meet both criteria.

#Maximum withdrawal amount in transaction

max\_withdrawl=spark.sql("select max(` WITHDRAWAL AMT `) as max\_withdrawl\_amount from txn ")

print('Maximum withdrawal amount in transaction:', max\_withdrawl.collect()[0][0])

A screenshot of a computer

Description automatically generated

**Explanation:** This code runs a Spark SQL query to find the maximum value in the "WITHDRAWAL AMT" column from the txn table. The query selects the maximum withdrawal amount and assigns it the alias max\_withdrawl\_amount. The result is then collected using the collect() function, and the maximum value is accessed from the first row ([0][0]) of the result. Finally, it prints the maximum withdrawal amount with the message 'Maximum withdrawal amount in transaction:'. This helps to determine the largest withdrawal amount from the transaction data.

#MINIMUM WITHDRAWAL AMOUNT OF AN ACCOUNT in txn

min\_withdrawl=spark.sql("select min(` WITHDRAWAL AMT `) as min\_withdrawl\_amount from txn ")

print('Min withdrawal amount in transaction:', min\_withdrawl.collect()[0][0])

A screen shot of a computer code

Description automatically generated

**Explanation:** This code runs a Spark SQL query to find the maximum value in the "WITHDRAWAL AMT" column from the txn table. The query selects the maximum withdrawal amount and assigns it the alias max\_withdrawl\_amount. The result is then collected using the collect() function, and the maximum value is accessed from the first row ([0][0]) of the result. Finally, it prints the maximum withdrawal amount with the message 'Maximum withdrawal amount in transaction:'. This helps to determine the largest withdrawal amount from the transaction data.

#MAXIMUM DEPOSIT AMOUNT OF AN ACCOUNT

max\_deposit=spark.sql("select max(` DEPOSIT AMT `) as max\_deposit\_amount from txn ")

print('Maximum deposit amount of an account:', max\_deposit.collect()[0][0])

**A screenshot of a computer program

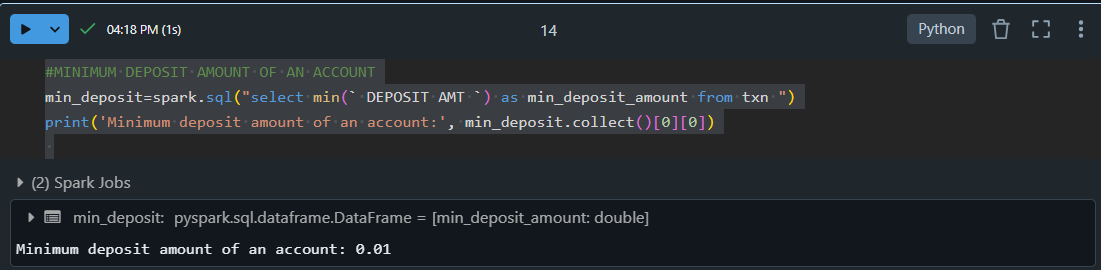
Description automatically generated**

**Explanation:**This code uses a Spark SQL query to calculate the maximum value in the "DEPOSIT AMT" column from the txn table. The query assigns the result the alias max\_deposit\_amount. The maximum deposit amount is then retrieved using the collect() method, which extracts the first row and column value ([0][0]) from the query result. Finally, it prints the maximum deposit amount with the message 'Maximum deposit amount of an account:'. This identifies the highest deposit transaction in the dataset.

#MINIMUM DEPOSIT AMOUNT OF AN ACCOUNT

min\_deposit=spark.sql("select min(` DEPOSIT AMT `) as min\_deposit\_amount from txn ")

print('Minimum deposit amount of an account:', min\_deposit.collect()[0][0])

Explanation: This code executes a Spark SQL query to find the minimum value in the "DEPOSIT AMT" column from the txn table. The query assigns the result an alias min\_deposit\_amount. Using the collect() method, it retrieves the first row and column value ([0][0]) of the result. Finally, it prints the minimum deposit amount with the message 'Minimum deposit amount of an account:'. This helps determine the smallest deposit transaction in the dataset.