**Banking Transaction:**

from pyspark.sql import SparkSession

from pyspark.sql.functions import col

from pyspark.sql import functions as F

spark = SparkSession.builder.appName("BankingTransaction").getOrCreate()

banking\_df = spark.read.format("csv").option("header","true").option("inferSchema","true").load("/content/sample\_data/bank\_transaction.csv")

**# 1. Calculate the Total Deposit and Withdrawal Amounts**

amounts\_df = banking\_df.groupBy("transaction\_type").agg(F.sum("amount").alias("total\_amount"))

amounts\_df.show()

**# 2. Filter Transactions Greater Than $3,000**

high\_transaction = banking\_df.filter(col("amount") > 3000)

high\_transaction.show()

**# 3. Find the Largest Deposit Made**

largest\_deposit = banking\_df.filter(col("transaction\_type") == "Deposit").orderBy(col("amount").desc()).limit(1)

largest\_deposit.show()

**# 4. Calculate the Average Transaction Amount for Each Transaction Type**

avg\_transaction\_df = banking\_df.groupBy("transaction\_type").agg(F.avg("amount").alias("average\_amount"))

avg\_transaction\_df.show()

**# 5. Find Customers Who Made Both Deposits and Withdrawals**

deposit\_customers = banking\_df.filter(col("transaction\_type") == "Deposit").select("customer\_id").distinct()

withdrawal\_customers = banking\_df.filter(col("transaction\_type") == "Withdrawal").select("customer\_id").distinct()

deposit\_customers.intersect(withdrawal\_customers).show()

**# 6. Calculate the Total Amount of Transactions per Day**

total\_transaction\_perday = banking\_df.groupBy("transaction\_date").agg(F.sum("amount").alias("total\_amount"))

total\_transaction\_perday.show()

**# 7. Find the Customer with the Highest Total Withdrawal**

highest\_withdrawal = banking\_df.filter(col("transaction\_type") == "Withdrawal").groupBy("customer\_id").agg(F.sum("amount").alias("total\_withdrawn")) \

.orderBy(col("total\_withdrawn").desc()).limit(1)

highest\_withdrawal.show()

**# 8. Calculate the Number of Transactions for Each Customer**

transaction\_per\_customer = banking\_df.groupBy("customer\_id").agg(F.count("transaction\_id").alias("transaction\_count"))

transaction\_per\_customer.show()

**# 9. Find All Transactions That Occurred on the Same Day as a Withdrawal Greater**

**# Than $1,000**

withdrawal\_dates = banking\_df.filter((col("transaction\_type") == "Withdrawal") & (col("amount") > 1000)) \

.select("transaction\_date").distinct()

banking\_df.join(withdrawal\_dates, on="transaction\_date").show()

**# 10. Create a New Column to Classify Transactions as "High" or "Low" Value**

banking\_df = banking\_df.withColumn("transaction\_value", F.when(col("amount") > 5000, "High").otherwise("Low"))

banking\_df.show()