SaiPrabath Chowdary S

PySpark Coding Challenge 10th Sep 2024

Banking Transactions

PySpark setup

```
[1] ! pip install pyspark
    from pyspark.sql import SparkSession
    from pyspark.sql.functions import col, sum, max, avg, count, rank, to_date, round
```

```
spark = SparkSession.builder.appName("Banking Transactions").getOrCreate()

data = '/content/drive/MyDrive/DataEngineering/PysparkCodingAssessment/BankingData.csv'
banking_df = spark.read.csv(data, header=True, inferSchema=True)
banking_df.show()
```

```
+--------------+-----------+----------------+------+----------------+
|transaction_id|customer_id|transaction_type|amount|transaction_date|
+--------------+-----------+----------------+------+----------------+
|             1|        201|         Deposit|  5000|      2023-09-01|
|             2|        202|      Withdrawal|  2000|      2023-09-01|
|             3|        203|         Deposit|  3000|      2023-09-02|
|             4|        201|      Withdrawal|  1500|      2023-09-02|
|             5|        204|         Deposit| 10000|      2023-09-03|
|             6|        205|      Withdrawal|   500|      2023-09-03|
|             7|        202|         Deposit|  2500|      2023-09-04|
|             8|        206|      Withdrawal|   700|      2023-09-04|
|             9|        203|         Deposit|  4000|      2023-09-05|
|            10|        204|      Withdrawal|  3000|      2023-09-05|
+--------------+-----------+----------------+------+----------------+
```

```
# 1. Calculate the Total Deposit and Withdrawal Amounts
total_amounts_by_type = banking_df.groupBy("transaction_type").agg(sum("amount").alias("total_amount"))
total_amounts_by_type.show()
```

```
+----------------+------------+
|transaction_type|total_amount|
+----------------+------------+
|         Deposit|       24500|
|      Withdrawal|        7700|
+----------------+------------+
```

+ Code + Text

```
[29] # 2. Filter Transactions Greater Than $3,000
     large_transactions = banking_df.filter(col("amount") > 3000)
     large_transactions.show()
```

```
+--------------+-----------+----------------+------+----------------+
|transaction_id|customer_id|transaction_type|amount|transaction_date|
+--------------+-----------+----------------+------+----------------+
|             1|        201|         Deposit|  5000|      2023-09-01|
|             5|        204|         Deposit| 10000|      2023-09-03|
|             9|        203|         Deposit|  4000|      2023-09-05|
+--------------+-----------+----------------+------+----------------+
```

```
[32] # 3. Find the Largest Deposit Made
     largest_deposit = banking_df.filter(col("transaction_type") == "Deposit").orderBy(col("amount").desc()).limit(1)
     largest_deposit.show()
```

```
+--------------+-----------+----------------+------+----------------+
|transaction_id|customer_id|transaction_type|amount|transaction_date|
+--------------+-----------+----------------+------+----------------+
|             5|        204|         Deposit| 10000|      2023-09-03|
+--------------+-----------+----------------+------+----------------+
```

```
# 4. Calculate the Average Transaction Amount for Each Transaction Type
avg_amount_by_type = banking_df.groupBy("transaction_type").agg(avg("amount").alias("avg_amount"))
avg_amount_by_type.show()
```

```
+----------------+----------+
|transaction_type|avg_amount|
+----------------+----------+
|         Deposit|    4900.0|
|      Withdrawal|    1540.0|
+----------------+----------+
```

```
[34] # 5. Find Customers Who Made Both Deposits and Withdrawals
     from pyspark.sql.functions import countDistinct, when

     customers_with_both = banking_df.groupBy("customer_id").agg(
         countDistinct(when(col("transaction_type") == "Deposit", 1)).alias("deposit_count"),
         countDistinct(when(col("transaction_type") == "Withdrawal", 1)).alias("withdrawal_count")
     ).filter((col("deposit_count") > 0) & (col("withdrawal_count") > 0))

     customers_with_both.show()
```

```
+-----------+-------------+----------------+
|customer_id|deposit_count|withdrawal_count|
+-----------+-------------+----------------+
|        202|            1|               1|
|        204|            1|               1|
|        201|            1|               1|
+-----------+-------------+----------------+
```

```
[35] # 6. Calculate the Total Amount of Transactions per Day
     banking_df = banking_df.withColumn("transaction_date", to_date(col("transaction_date"), "yyyy-MM-dd"))

     total_amount_per_day = banking_df.groupBy("transaction_date").agg(sum("amount").alias("total_amount"))
     total_amount_per_day.show()
```

```
+----------------+------------+
|transaction_date|total_amount|
+----------------+------------+
|      2023-09-03|       10500|
|      2023-09-01|        7000|
|      2023-09-05|        7000|
|      2023-09-02|        4500|
|      2023-09-04|        3200|
+----------------+------------+
```

```
# 7. Find the Customer with the Highest Total Withdrawal
highest_withdrawal_customer = banking_df.filter(col("transaction_type") == "Withdrawal") \
            .groupBy("customer_id").agg(sum("amount").alias("total_withdrawal")) \
            .orderBy(col("total_withdrawal").desc()).limit(1)
highest_withdrawal_customer.show()
```

```
+-----------+----------------+
|customer_id|total_withdrawal|
+-----------+----------------+
|        204|            3000|
+-----------+----------------+
```

[38]
```
# 8. Calculate the Number of Transactions for Each Customer
transaction_count_by_customer = banking_df.groupBy("customer_id").agg(count("*").alias("transaction_count"))
transaction_count_by_customer.show()
```

```
+-----------+-----------------+
|customer_id|transaction_count|
+-----------+-----------------+
|        206|                1|
|        205|                1|
|        202|                2|
|        203|                2|
|        204|                2|
|        201|                2|
+-----------+-----------------+
```

[53]
```
# 9. Find All Transactions That Occurred on the Same Day as a Withdrawal Greater Than $1,000

filtered_withdrawals = banking_df.filter(col("transaction_type") == "Withdrawal") \
    .filter(col("amount") > 1000)

same_day_withdrawals = banking_df.join(filtered_withdrawals, banking_df["transaction_date"] == filtered_withdrawals["transaction_date"], "inner")\
    .select(banking_df["*"])
same_day_withdrawals.show()
```

```
+--------------+-----------+----------------+------+----------------+
|transaction_id|customer_id|transaction_type|amount|transaction_date|
+--------------+-----------+----------------+------+----------------+
|             1|        201|         Deposit|  5000|      2023-09-01|
|             2|        202|      Withdrawal|  2000|      2023-09-01|
|             3|        203|         Deposit|  3000|      2023-09-02|
|             4|        201|      Withdrawal|  1500|      2023-09-02|
|             9|        203|         Deposit|  4000|      2023-09-05|
|            10|        204|      Withdrawal|  3000|      2023-09-05|
+--------------+-----------+----------------+------+----------------+
```

```
# 10. Create a New Column to Classify Transactions as "High" or "Low" Value
banking_df = banking_df.withColumn("transaction_value", when(col("amount") > 5000, "High").otherwise("Low"))
banking_df.show()
```

```
+--------------+-----------+----------------+------+----------------+-----------------+
|transaction_id|customer_id|transaction_type|amount|transaction_date|transaction_value|
+--------------+-----------+----------------+------+----------------+-----------------+
|             1|        201|         Deposit|  5000|      2023-09-01|              Low|
|             2|        202|      Withdrawal|  2000|      2023-09-01|              Low|
|             3|        203|         Deposit|  3000|      2023-09-02|              Low|
|             4|        201|      Withdrawal|  1500|      2023-09-02|              Low|
|             5|        204|         Deposit| 10000|      2023-09-03|             High|
|             6|        205|      Withdrawal|   500|      2023-09-03|              Low|
|             7|        202|         Deposit|  2500|      2023-09-04|              Low|
|             8|        206|      Withdrawal|   700|      2023-09-04|              Low|
|             9|        203|         Deposit|  4000|      2023-09-05|              Low|
|            10|        204|      Withdrawal|  3000|      2023-09-05|              Low|
+--------------+-----------+----------------+------+----------------+-----------------+
```