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_am
ount"))

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("trans
action_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()
```