Credit Card Transaction Analysis

Behind every transaction lies a story – let’s uncover it!

www.streaming.com

Meet Our Team

Ankita Singh

Rohini Goyal

Arman

Sarika Sunuguri

www.streaming.com

Project Overview

ABOUT THE COMPANY

- Leading credit card provider with 100M+ customers, 1000+ merchant partners, and $100B+ annual card spend
- Operating for over 10 years, now aiming for rapid cloud-based expansion

PROJECT OBJECTIVE

- Analyze historical credit card transaction data
- Uncover trends in customer spending and behavior
- Build scalable, cloud-ready data pipelines.
- Deliver business insights through 9 targeted queries

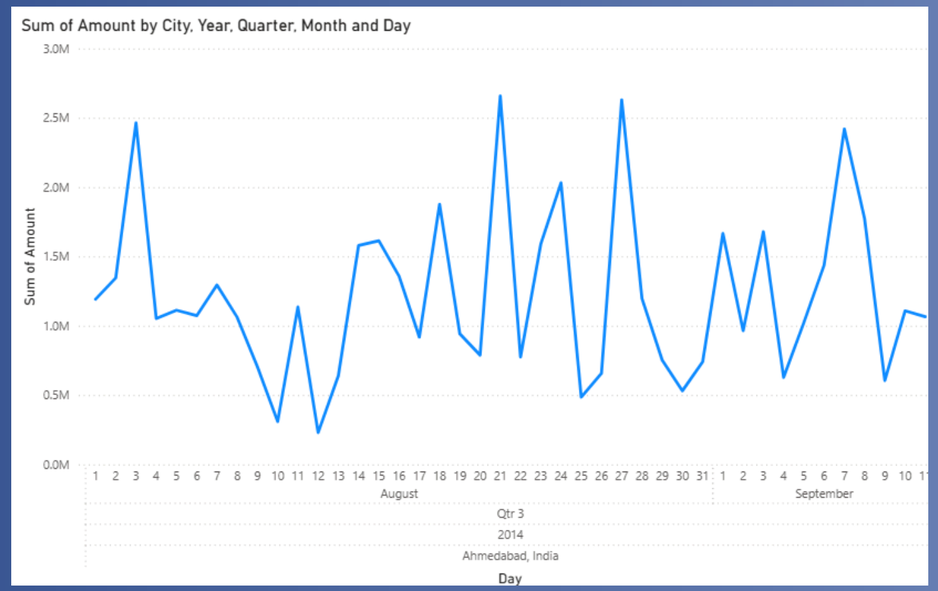
OUR GOAL

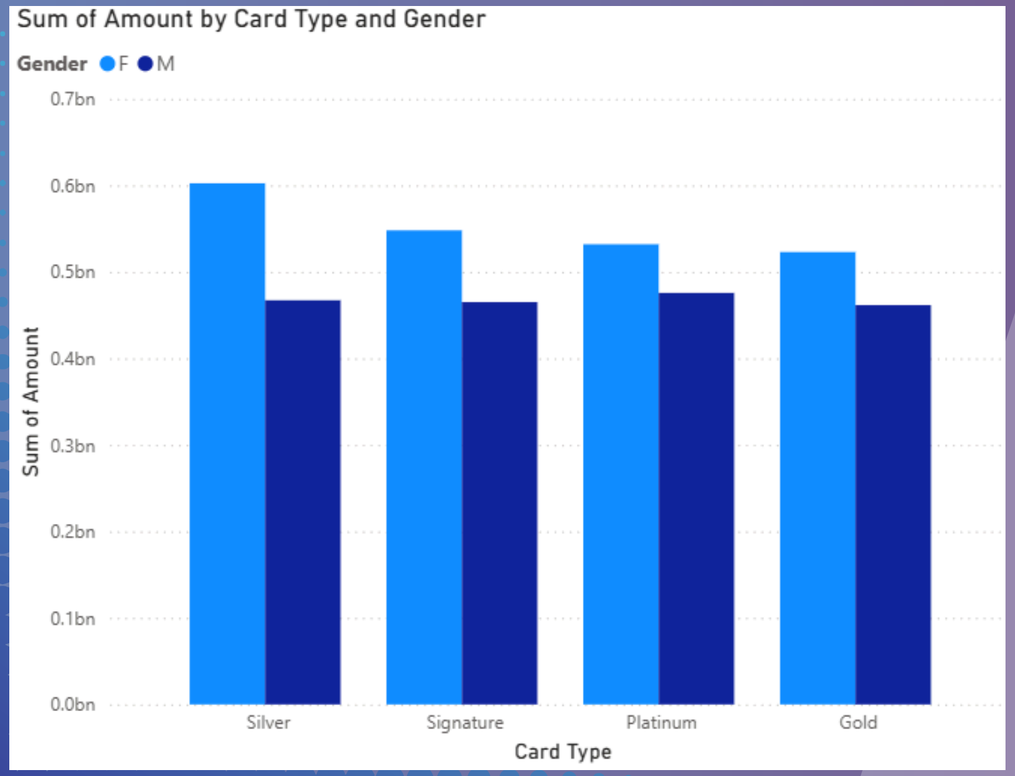
- Convert raw transaction data into actionable insights
- Enable cloud migration with scalable data pipelines
- Support decisions with real-time and accurate analytics

www.streaming.com

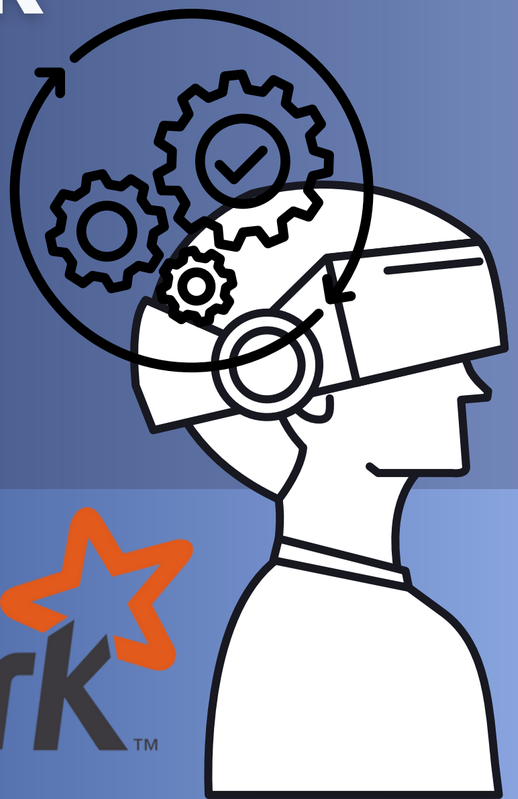
Dataset Description

- Contains credit card transactions made across India
- Captures key variables: City, Gender, Card Type, Expense Type, Amount, and Date
- Offers insights into spending behavior and customer preferences
- Enables discovery of trends, correlations, and valuable business intelligence
- Ideal for exploring unbiased data analysis techniques and real-world financial patterns





Technology Stack



1. PySpark in Google Colab – Used RDD and DataFrame APIs for data processing
2. Spark Scala – Performed operations in Spark Shell and queried data using RDD and DataFrame APIs for data processing
3. Hive in Hadoop YARN Cluster – Used as the execution environment for running Spark and Hive jobs

Query 1 : Write a query to print top 5 cities with highest spends and their percentage contribution of total credit card spends

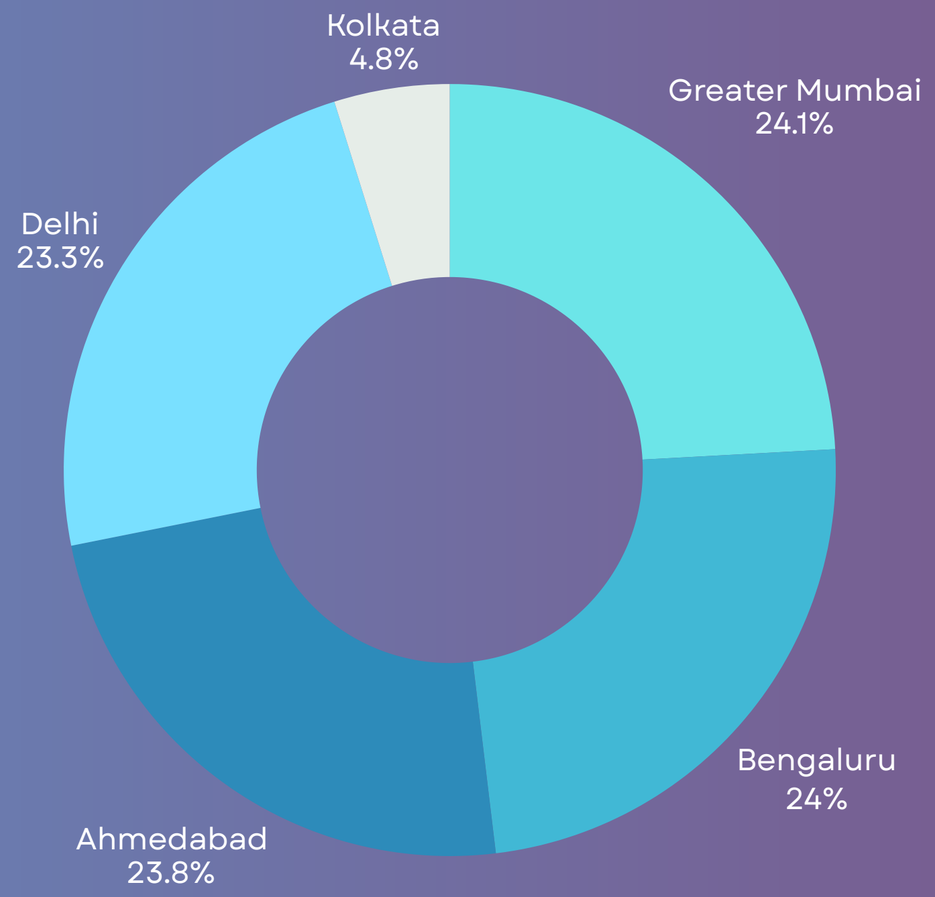
Coding in Hive

```
hive> WITH city_spends AS (
    >   SELECT city, SUM(amount) AS total_spend
    >   FROM credit_card_transactions
    >   GROUP BY city
    > ),
    > total_spend AS (
    >   SELECT SUM(amount) AS total_amount
    >   FROM credit_card_transactions
    > )
    > SELECT
    >   cs.city,
    >   cs.total_spend,
    >   ROUND(cs.total_spend / ts.total_amount * 100, 2) AS percentage_contribut
ion
    > FROM city_spends cs
    > CROSS JOIN total_spend ts
    > ORDER BY cs.total_spend DESC
    > LIMIT 5;
```

Output

```
OK
"Greater Mumbai 576751476        14.15
"Bengaluru      572326739        14.05
"Ahmedabad      567794310        13.93
"Delhi  556929212        13.67
"Kolkata        115466943        2.83
Time taken: 186.853 seconds, Fetched: 5 row(s)
```

Pictorial Visualization



Query 2 : Write a query to print the transaction details (all columns from the table) for each card type when- - it reaches a cumulative of 1000000 total spends

Coding in Pyspark

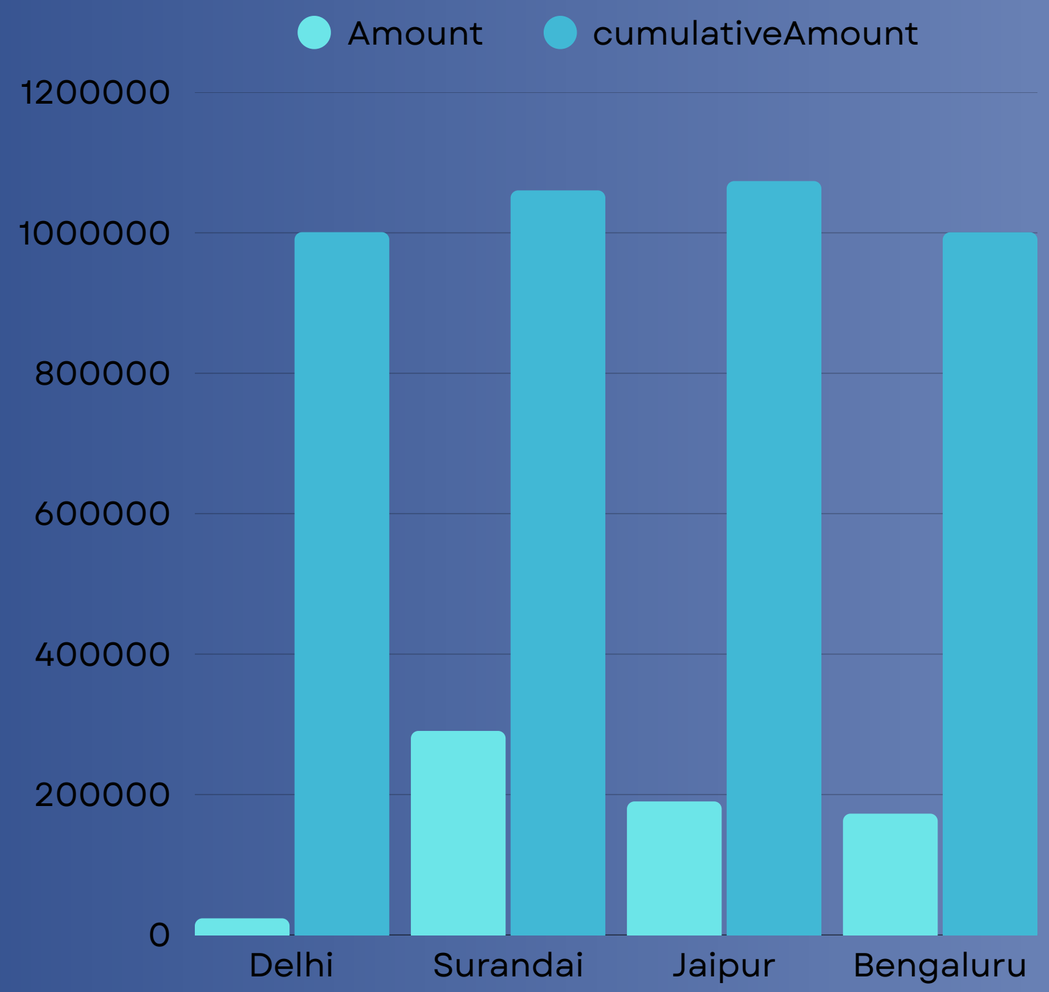
```
from pyspark.sql.functions import sum, col, row_number

cardTypeWindow = Window.partitionBy("Card Type").orderBy("Date").rowsBetween(Window.unboundedPreceding, Window.currentRow)

cumulativeSpendDF = cumulativeDF.withColumn("cumulativeAmount", sum("Amount").over(cardTypeWindow))

filteredDF = cumulativeSpendDF.filter(col("cumulativeAmount") >= 1000000)

rowWindow = Window.partitionBy("Card Type").orderBy("Date")
resultDF = filteredDF.withColumn("rowNum", row_number().over(rowWindow)) \
                     .filter(col("rowNum") == 1) \
                     .drop("rowNum")
```

Output

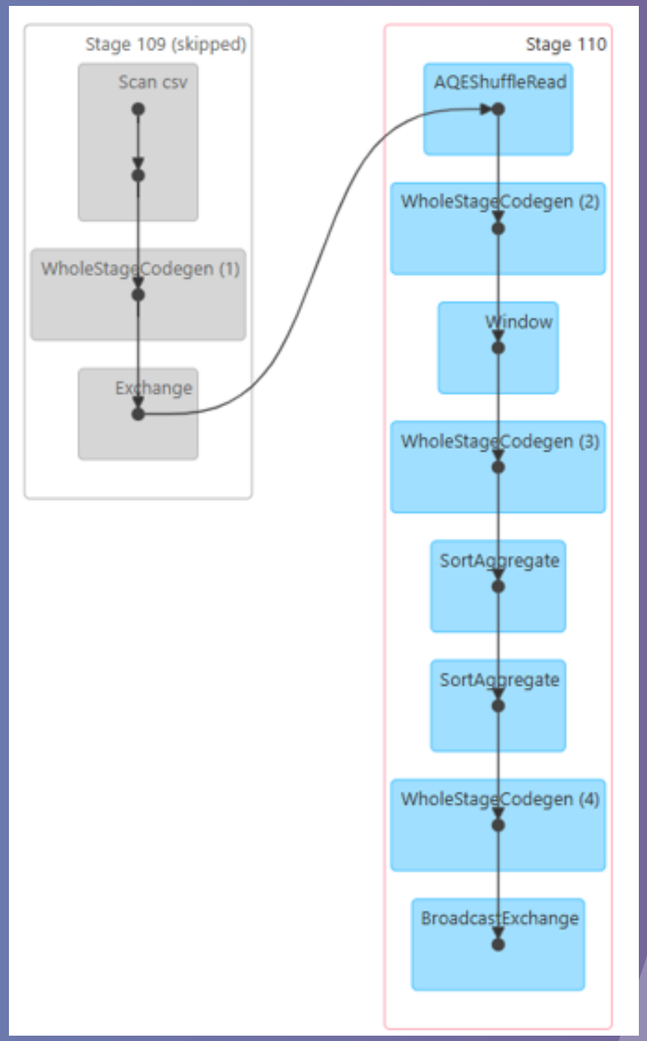
index	City	Date	Card Type	Exp Type	Gender	Amount	cumulativeAmount
13431	Delhi, India	1-Apr-14	Gold	Grocery	M	23697	1000956
18394	Surandai, India	1-Apr-14	Platinum	Entertainment	F	290618	1060394
20659	Jaipur, India	1-Apr-14	Signature	Bills	F	190198	1073671
13960	Bengaluru, India	1-Apr-14	Silver	Entertainment	F	172789	1000757

Query 2

Pictorial Visualization



DAG Visualization



Query 3 : Write a query to find city which had lowest percentage spend for gold card type

Coding in Pyspark

```
[41] lowestDF = spark.read.csv("/content/Sprint Data/Credit card transactions - India - Simple.csv", header=True, inferSchema=True)
```

```
goldDF = lowestDF.filter(col("Card Type") == "Gold")

citySpendDF = goldDF.groupBy("City").agg(sum("Amount").alias("Gold Spend"))

totalGoldSpend = goldDF.agg(sum("Amount").alias("totalGold")).collect()[0]["totalGold"]

lowestGoldCity = citySpendDF.withColumn(
    "percentage",
    round((col("Gold Spend") / totalGoldSpend) * 100, 3)
).orderBy("Gold Spend").limit(1)

lowestGoldCity.show()
```

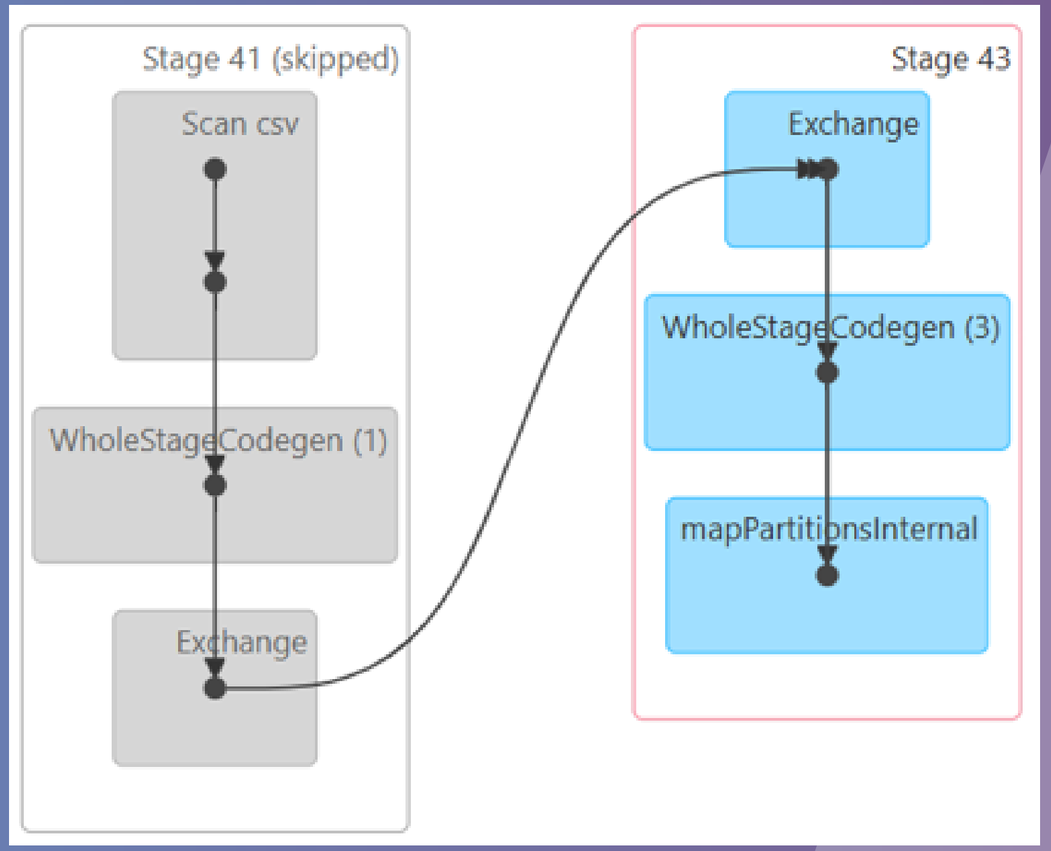
Output

```
+---------------+----------+----------+
|           City|Gold Spend|percentage|
+---------------+----------+----------+
|Dhamtari, India|      1416|       0.0|
+---------------+----------+----------+
```

Query 3

Pictorial Visualization

DAG Visualization



Query 4 : Write a query to print 3 columns : city, highest_expense_type, lowest_expense_city.

Coding in SparkScala

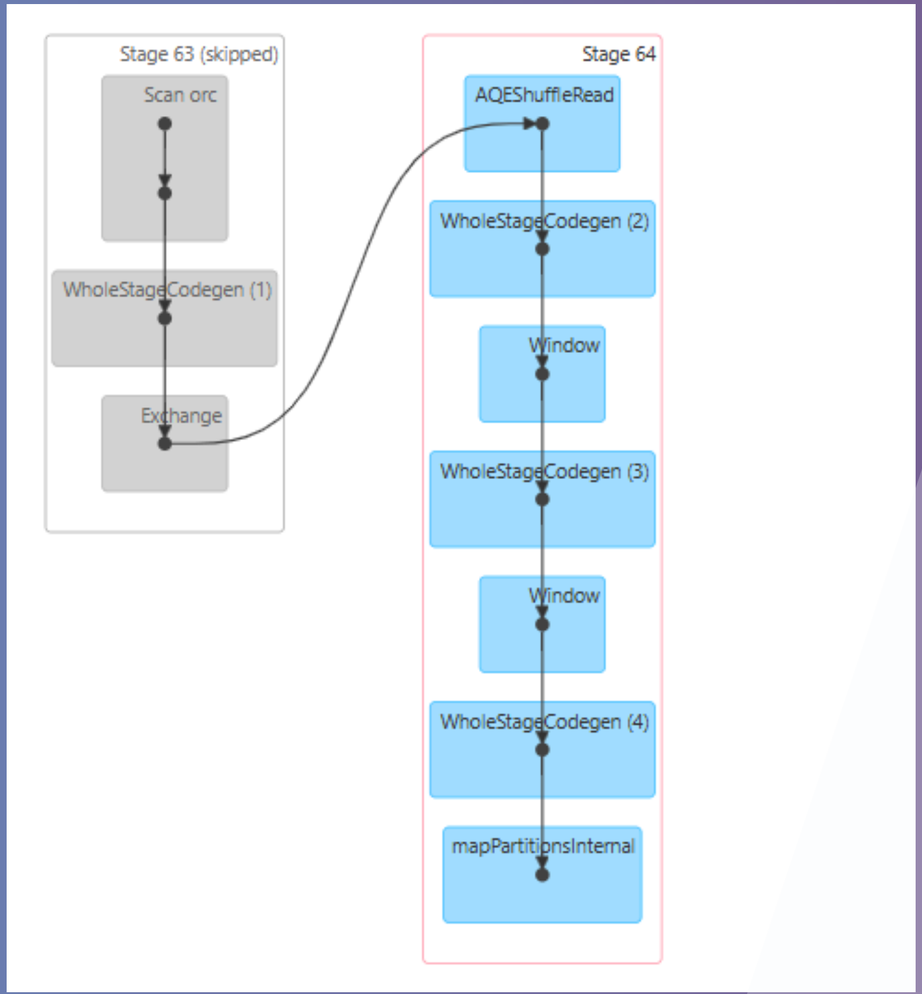
```
val dataorc = spark.read.option("header", "true").orc("C:/Users/ROGOYAL/Downloads/SpringProject/Answers/5C
import org.apache.spark.sql.functions._
import org.apache.spark.sql.expressions.Window
val desccity = Window.partitionBy("City").orderBy(col("Amount").desc)
val asccity = Window.partitionBy("City").orderBy(col("Amount").asc)
val descasc = dataorc.withColumn("highest_expense", first("Exp Type").o
ver(desccity)).withColumn("lowest_expense", first("Exp Type").over(asccity))
val result = descasc.select("City", "highest_expense", "lowest_expense").distinct()
result.show()
result.write.option("header", "true").csv("C:/Users/ROGOYAL/Downloads/S
pring Project/Answers/5Question/5quesOutput.csv")
```

Query 4

Output

```
scala> result.show()
[Stage 62:>

+----------------+--------------+--------------+
|            City|highest_expense|lowest_expense|
+----------------+--------------+--------------+
|  Achalpur, India|  Entertainment|          Fuel|
|  Adilabad, India|          Food| Entertainment|
| Adityapur, India|          Fuel|       Grocery|
|     Adoni, India|       Grocery| Entertainment|
|     Adoor, India|  Entertainment|          Food|
|  Afzalpur, India|  Entertainment|          Fuel|
|  Agartala, India|       Grocery|       Grocery|
|      Agra, India|         Bills|       Grocery|
| Ahmedabad, India|          Food|       Grocery|
|Ahmednagar, India|       Grocery| Entertainment|
|    Aizawl, India|          Food|          Food|
|     Ajmer, India|          Fuel|         Bills|
|     Akola, India|         Bills| Entertainment|
|      Akot, India|          Fuel|          Food|
| Alappuzha, India|         Bills|          Fuel|
|   Aligarh, India|          Fuel|         Bills|
|Alipurduar, India|          Fuel| Entertainment|
| Alirajpur, India|  Entertainment| Entertainment|
| Allahabad, India|       Grocery| Entertainment|
|     Alwar, India|          Food| Entertainment|
+----------------+--------------+--------------+
only showing top 20 rows
```

DAG Visualization



Query 5 : Write a query to find percentage contribution of spends by females for each expense type

Coding in SparkScala - RDD

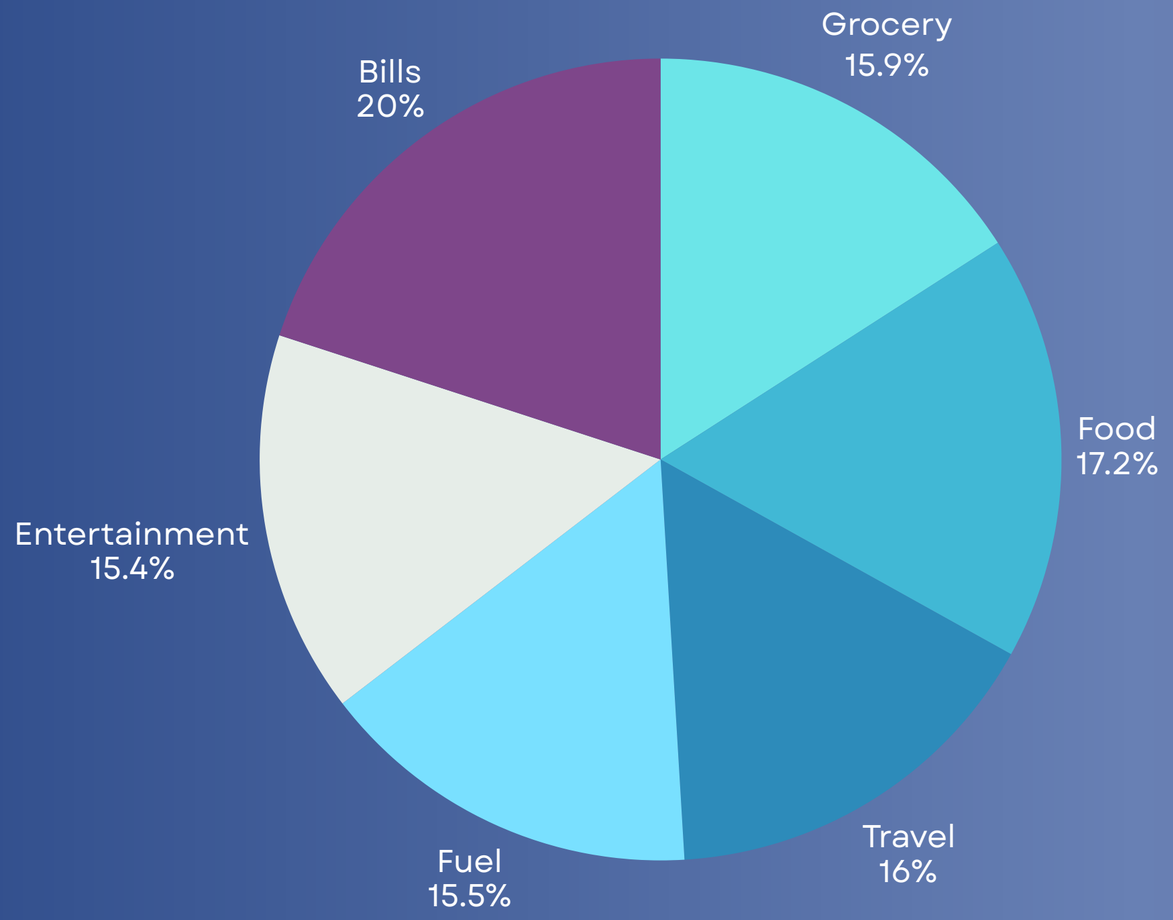
```
import org.apache.spark.sql.SparkSession
val spark = SparkSession.builder.appName("spark with scala example").master("local[*]").getOrCreate()
val df = spark.read.option("header", "true").option("inferSchema", "true").option("multiLine", "true")
    .option("quote", "\"").option("escape", "\"").csv("C:/Users/ROGOYAL/Downloads/Spring Project/Simple.csv")

val cleanedDF = df.selectExpr("`Exp Type` as expType", "Gender as gender", "Amount as amount")
    .filter("expType is not null and gender is not null and amount is not null")

val rdd = cleanedDF.rdd.map(row => {
     |   val expType = row.getString(0)
     |   val gender = row.getString(1)
     |   val amount = row.get(2) match {
     |     case d: Double => d
     |     case i: Int => i.toDouble
     |     case l: Long => l.toDouble
     |     case f: Float => f.toDouble
     |     case s: String => s.toDouble
     |     case other => throw new RuntimeException(s"Unexpected type: $other")}(expType, gender, amount)})

val totalByExpType = rdd.map { case (expType, _, amount) => (expType, amount) }.reduceByKey(_ + _)

val femaleByExpType = rdd.filter { case (_, gender, _) => gender == "F"}.map { case (expType, _, amount) => (expType, amount) }.reduceByKey(_ + _)

val percentageByExpType = femaleByExpType.join(totalByExpType).mapValues { case (femaleAmt, totalAmt) => (femaleAmt / totalAmt) * 100 }

percentageByExpType.collect().foreach {case (expType, percent) =>println
n(f"$expType: $percent%.2f%%")}

val output = percentageByExpType.map {case (expType, percent) => f"$exp
Type: $percent%.2f%%"}

output.saveAsTextFile("C:/Users/ROGOYAL/Downloads/Spring Project/Answer
s/6Question/6quesOutput.txt")
```

Output

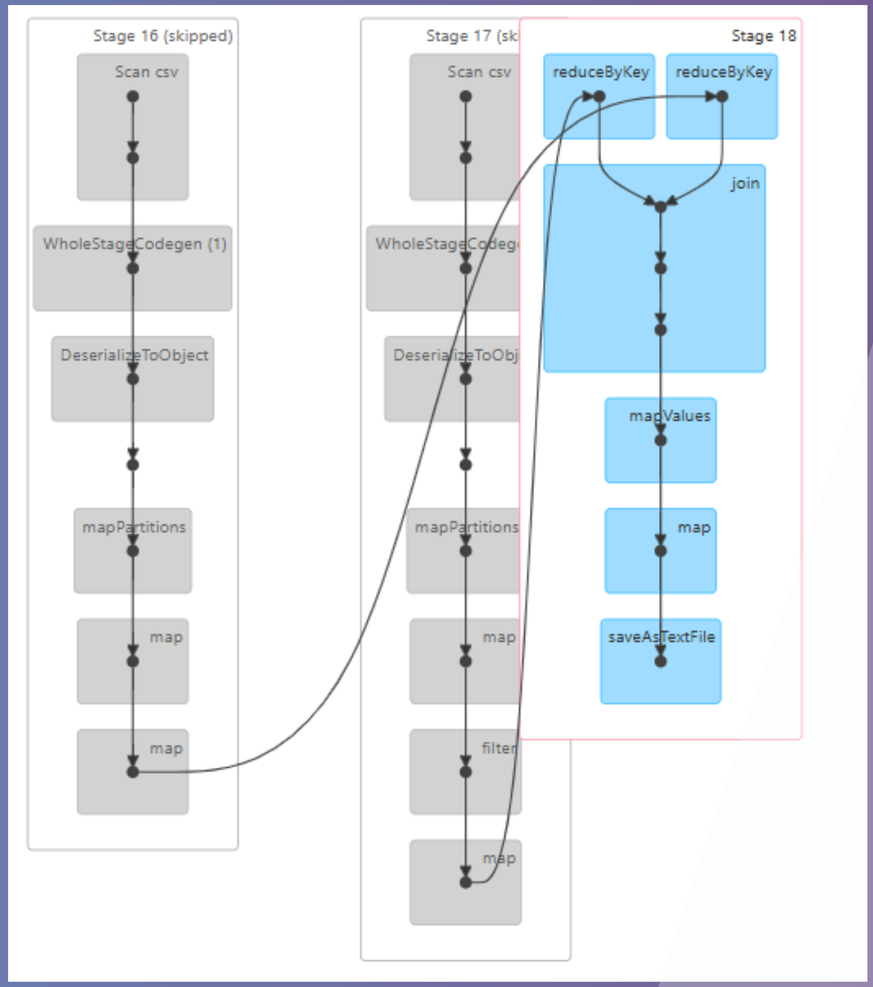
```
Grocery: 50.91%
Food: 54.91%
Travel: 51.13%
Fuel: 49.71%
Entertainment: 49.37%
Bills: 63.95%
```

Query 5

Pictorial Visualization



DAG Visualization



Query 6 : Which card and expense type combination saw highest month over month growth in Jan-2014

Coding in SparkScala

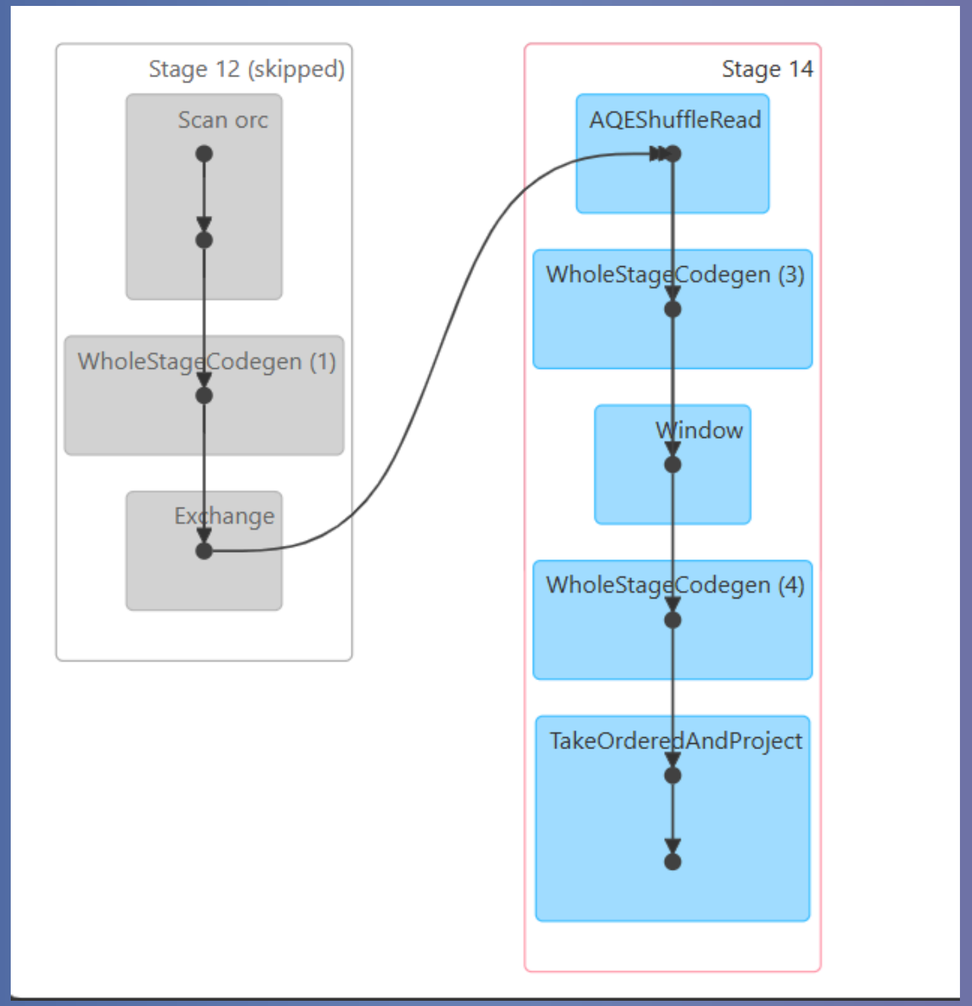
```
import org.apache.spark.sql.functions._
import org.apache.spark.sql.expressions.Window
val df = mydforc.withColumn("ParsedDate",coalesce(to_date(col("Date"), "d-MMM-yy"), to_date(col("Date"), "dd-MMM-yy")))
val dfWithYM = df.withColumn("month", month(col("ParsedDate"))).withColumn("year", year(col("ParsedDate")))
val monthlySpend = dfWithYM.groupBy("Card Type", "Exp Type", "year", "month").agg(sum("Amount").alias("MonthlyAmount"))
val windSpec = Window.partitionBy("Card Type", "Exp Type").orderBy("year", "month")
val withLag = monthlySpend.withColumn("PrevMonthAmount", lag("MonthlyAmount", 1).over(windSpec))
val withGrowth = withLag.withColumn("MoMGrowth", (col("MonthlyAmount") - col("PrevMonthAmount")) / col("PrevMonthAmount"))
val jan2014 = withGrowth.filter(col("year") === 2014 && col("month") === 1)
val topGrowthCombo = jan2014.orderBy(col("MoMGrowth").desc).limit(1)
topGrowthCombo.show()
```

Output

```
scala> topGrowthCombo.show()
+---------+--------+----+-----+-------------+---------------+-----------------+
|Card Type|Exp Type|year|month|MonthlyAmount|PrevMonthAmount|        MoMGrowth|
+---------+--------+----+-----+-------------+---------------+-----------------+
|     Gold|  Travel|2014|    1|    2092554.0|      1113534.0|0.8792008147034577|
+---------+--------+----+-----+-------------+---------------+-----------------+
```

Query 6

DAG Visualization



Query 7 : During weekends which city has highest total spend to total no of transaction ratio.

Coding in Pyspark

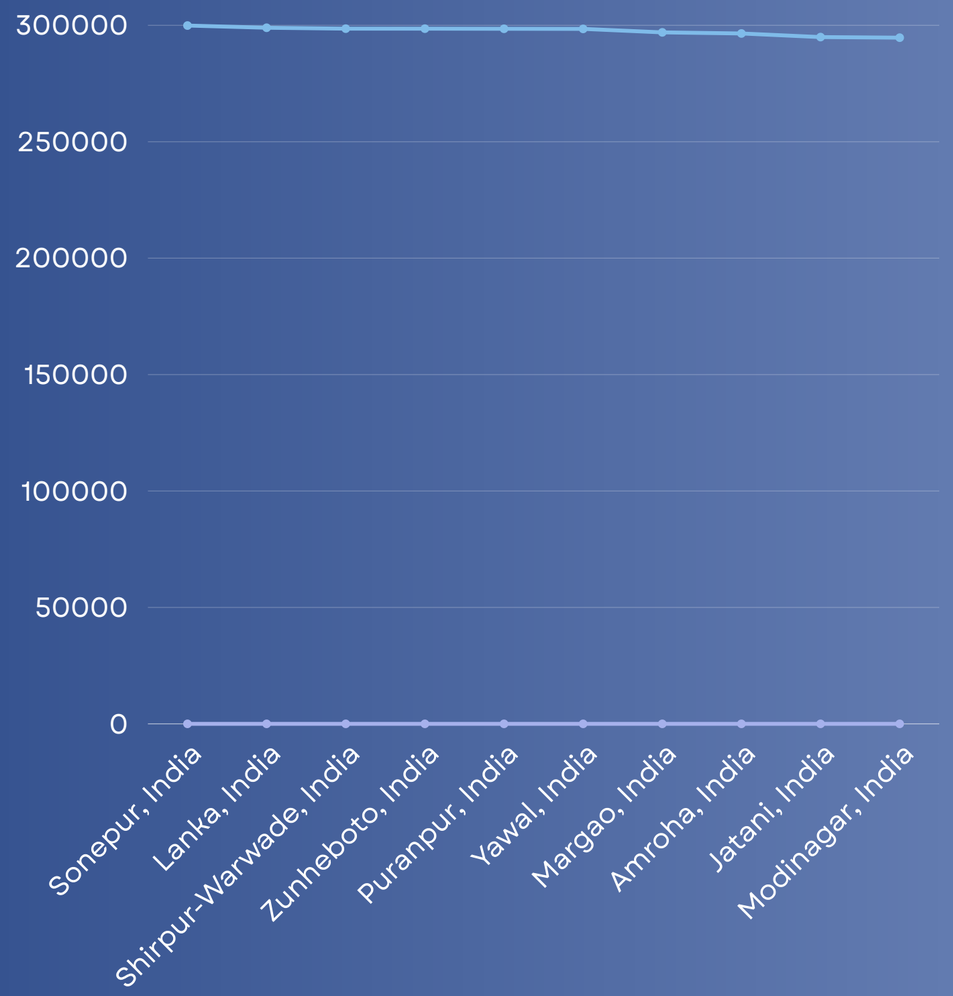
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, to_date, dayofweek, sum as _sum, count
spark = SparkSession.builder.appName("CreditCardWeekendAnalysis").config("spark.sql.legacy.timeParserPolicy", "LEGACY").getOrCreate()
df = spark.read.csv("/content/sample_data/Credit card transactions - India - Simple.csv", inferSchema = True, header = True)
df = df.withColumn("Amount", col("Amount").cast("int")).withColumn("Date", to_date(col("Date"), "d-MMM-yy"))
weekend_df = df.withColumn("dayofweek", dayofweek("Date")).filter((col("dayofweek") == 1) | (col("dayofweek") == 7))
agg_df = weekend_df.groupBy("City").agg(_sum("Amount").alias("total_spend"),count("*").alias("total_transactions"))
ratio_df = agg_df.withColumn("spend_transaction_ratio", col("total_spend") / col("total_transactions"))
top_10_cities = ratio_df.orderBy(col("spend_transaction_ratio").desc()).limit(1)
top_10_cities.show(truncate=False)
top_10_cities.coalesce(1).write.option("header", True).csv("/content/Q8py2.csv")
```

Output

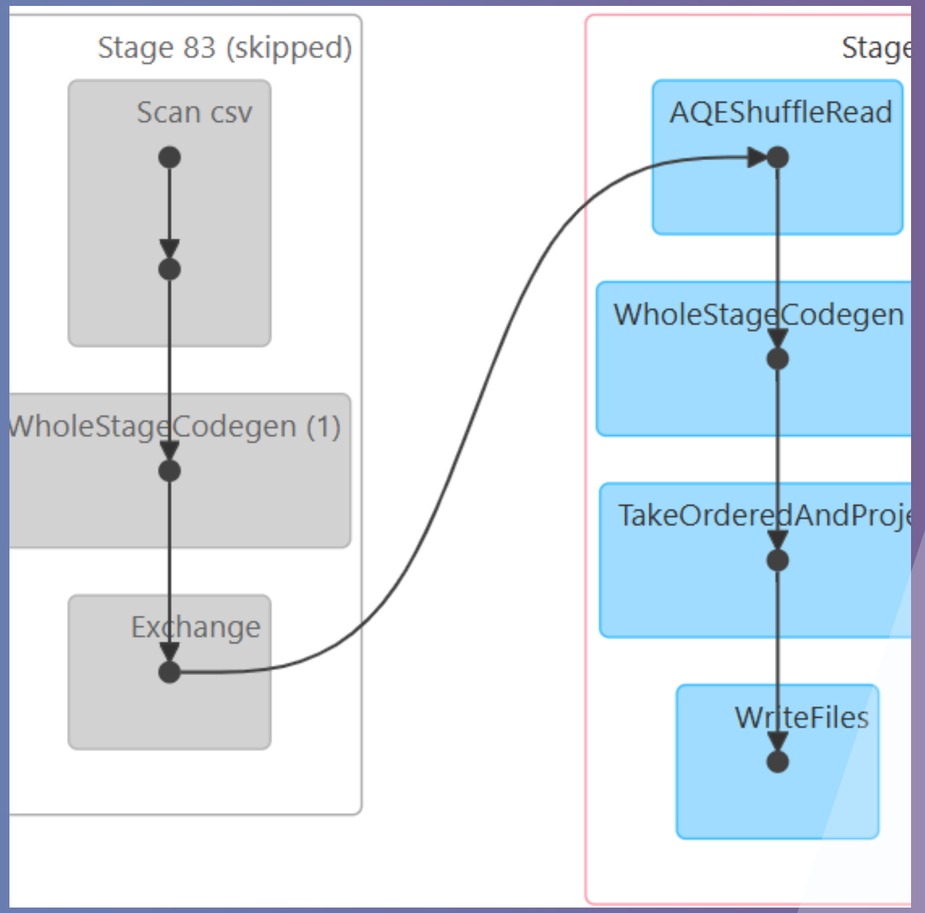
```
+--------------+-----------+------------------+-----------------------+
|City          |total_spend|total_transactions|spend_transaction_ratio|
+--------------+-----------+------------------+-----------------------+
|Sonepur, India|299905     |1                 |299905.0               |
+--------------+-----------+------------------+-----------------------+
```

Query 7

Pictorial Visualization



DAG Visualization



Query 8 : Which city took least number of days to reach its 500th transaction after the first transaction in that city

Coding in Pyspark

```
            sorted_dates = sorted(dates)
            diff = (sorted_dates[499] - sorted_dates[0]).days
            return (city, diff)
        else:
            return None

city_diffs = combined.map(getDiff500).filter(lambda x: x is not None)
```

```
fastestCity = city_diffs.takeOrdered(1, key=lambda x: x[1])

print("City that reached 500th transaction the fastest: ['City Name', Number of days]")
print(fastestCity)
```

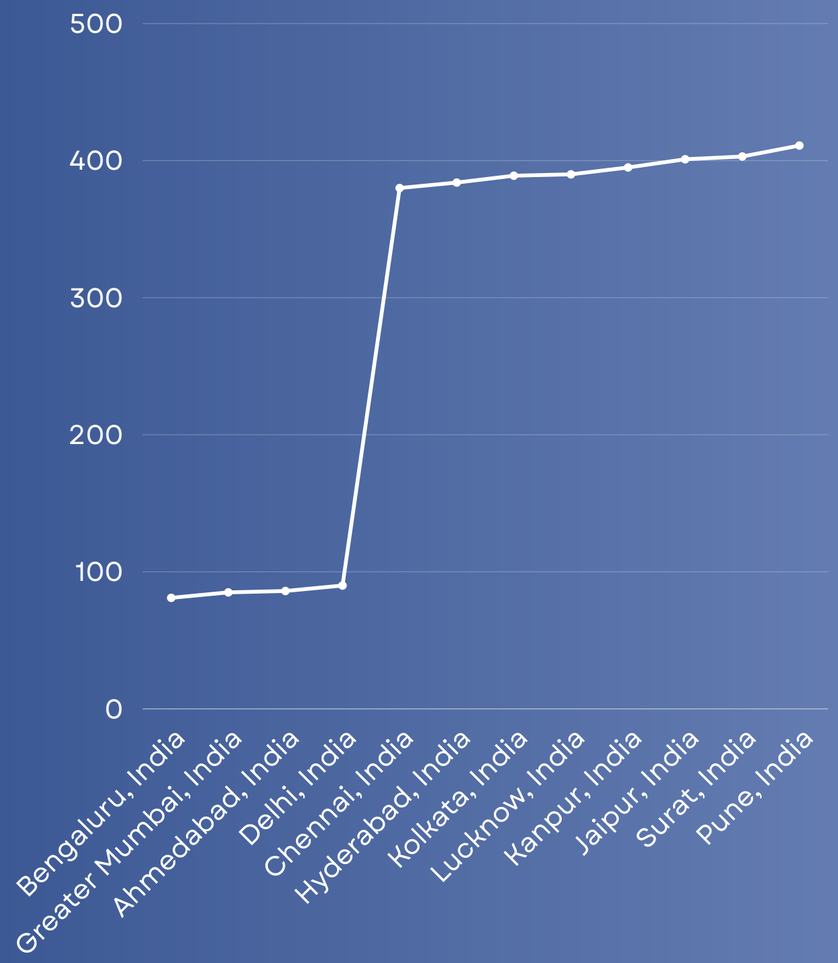
Output

```
result.collect()

#result.coalesce(1).saveAsTextFile("/content/city_to_500_transaction_fastest") # for saving in 1 part I used coalesce.
```

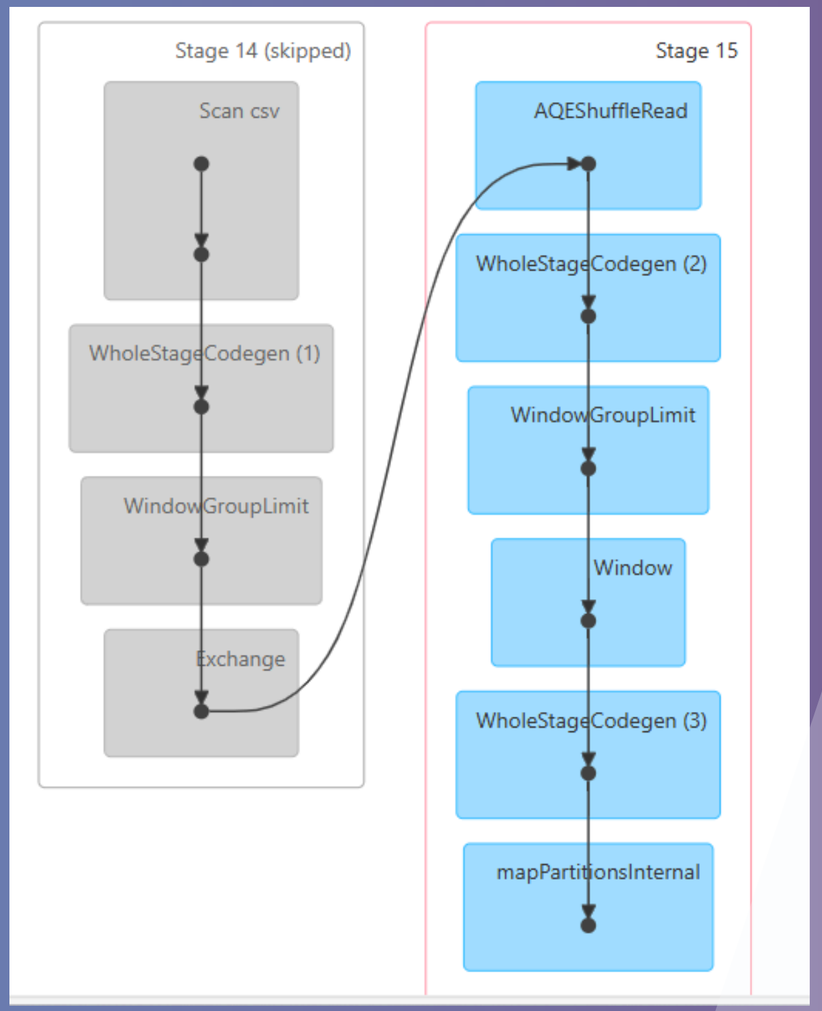
```
['City Name , Days', 'Bengaluru , 81']
```

Query 8

Pictorial Visualization



DAG Visualization



Performance Optimization

1. Efficient Storage
   - Used Parquet and ORC formats for faster read/write and compression
2. Optimized Computation
   - Avoided groupByKey(), used reduceByKey() and aggregations
   - Applied cache(), persist() for reusing data efficiently
   - Used partitioning to improve parallelism and data locality
3. Execution Efficiency
   - Filtered early (filter()) to reduce data load
   - Selected only needed columns with select()
   - Managed data skew with repartition() / coalesce()

Challenges

1. **Software Compatibility Issues** – Encountering systems that are not compatible with the required software or tools, leading to delays and workarounds.

2. **Complexity in Logic Development** – Facing difficulties in designing or implementing efficient and scalable logic for problem-solving.

3. **Limited Knowledge of Built-in Functions** – Struggling due to insufficient familiarity with available functions or libraries, which affects productivity and code optimization.

Conclusion & Insights

- Metro cities like Greater Mumbai and Bengaluru dominate overall credit card spends
- Some Metropolitan cities reached their 500th transaction very quickly, highlighting early adoption and usage trends
- Female users significantly contribute to spending in key categories like Bills and Food

So, we can say that metro cities lead in credit card spending, some cities adopted usage faster, and spending habits vary across different expense categories.

Thank You For Your Attention!

We are open to any questions or feedback you may have.