PYSPARK CODING CHALLENGE

HARSHINI V

ETL WITH PYSPARK:

ETL (Extract, Transform, Load) is a fundamental process for organizing and preparing data in a streamlined and effective manner. Using PySpark, this process benefits from distributed computing, enabling the handling of massive datasets with exceptional speed and efficiency.

1. Extract (E):
In PySpark, extraction involves retrieving raw data from various sources, including databases, cloud storage, APIs, or local files. PySpark supports numerous data formats such as CSV, JSON, Parquet, and Avro, making it highly adaptable for different data workflows. The process begins by utilizing the SparkSession, which facilitates smooth connections to and reading from these sources.

Example:

```
df = spark.read.csv("data.csv", header=True, inferSchema=True)
```

2. Transform (T):
During this step, the extracted data is cleansed, enriched, and restructured to make it suitable for further analysis. PySpark offers a rich set of APIs for transformations, including filtering, grouping, joining, and handling missing data. These transformations are optimized for distributed processing, ensuring efficiency even with very large datasets.

Example:

```
transformed_df = df.filter(df["column"] > 100).groupBy("category").agg({"value": "sum"})
```

3. Load (L):
After transformation, the data is saved to its target destination, such as a data warehouse, relational database, or storage system. PySpark provides compatibility with a variety of storage systems, enabling seamless data integration. You can save the output in formats like Parquet or Hive for optimized querying or load it directly into databases.

Example:

```
transformed_df.write.mode("overwrite").parquet("output_path")
```

By utilizing PySpark for ETL, organizations can efficiently process, transform, and load data, making it a preferred choice for big data pipelines. PySpark not only accelerates ETL operations but also ensures they are scalable, dependable, and versatile.

Benefits of PySpark for ETL

- **Scalability:** PySpark is designed to manage extensive datasets by distributing computations across multiple nodes.
- **Flexibility:** It integrates seamlessly with various data sources and formats.
- **Fault Tolerance:** Built on Apache Spark, PySpark ensures reliability by recovering from node failures automatically.
- **User-Friendly:** The PySpark API combines the simplicity of Python with the robust capabilities of Spark.

USING SPARK SQL - TRANSFORMATIONS SUCH AS FILTER, JOIN, SIMPLE AGGREGATIONS, GROUPBY ON THE CASE STUDY DATASET:

FILTER:

Filtering is a crucial transformation in Spark SQL that extracts specific rows from a dataset based on given conditions. Filters help narrow down data to include only the relevant records for analysis. For instance, in the loan dataset, you can filter customers with an income greater than 50,000 or those with more than two returned cheques. This is done using the WHERE clause in SQL queries. Filters can also combine multiple conditions using logical operators such as AND or OR, enabling complex queries.

EXAMPLES:

Filter customers with income greater than 60,000

loan_df.filter(loan_df['income'] > 60000).show()

```
# Filter customers with income greater than 60,000
loan_df.filter(loan_df['income'] > 60000).show()
```

Customer_Id	Age	Gender	Occupation	Marital Status	Family Size	Income	Expenditure	Use Frequency	Loan Category	Loan Amount	Overdue	Debt Record	Returned Cheque	Dishonour of Bill
15767821	24	MALE	DATA ANALYST	SINGLE	4	60111	28999	6	AUTOMOBILE	35,232	5	33,333	1	2
15643966	25	FEMALE	DOCTOR	SINGLE	4	60111	27111	5	TRAVELLING	12,90,929	4	18,000	1	0
15738191	60	FEMALE	TEACHER	MARRIED	5	70000	40000	9	GOLD LOAN	2,57,789	4	10,058	4	3
15728693	25	MALE	PROFESSOR	SINGLE	5	62145	31254	4	BOOK STORES	12,45,789	6	48,596	6	5
15706552	49	MALE	ASSISTANT PROFESSOR	MARRIED	5	65214	42589	5	HOUSING	9,85,412	5	11,254	1	2
15659428	47	FEMALE	DOCTOR	MARRIED	4	72154	45286	4	AUTOMOBILE	7,54,126	2	19,524	5	2
15794171	54	MALE	AIRPORT OFFICER	MARRIED	6	80000	32541	2	AUTOMOBILE	20,45,789	1	16,599	2	3
15729599	44	FEMALE	ACCOUNT MANAGER	MARRIED	4	800000	15632	8	AUTOMOBILE	23,65,478	5	20,145	3	4
15738148	41	MALE	BANK MANAGER	MARRIED	6	64125	21246	6	TRAVELLING	6,52,147	5	16,524	3	3
15684171	33	MALE	DOCTOR	MARRIED	6	70000	12541	8	HOUSING	7,45,213	4	19,541	1	3
15766205	46	FEMALE	CLERK	MARRIED	3	750000	25641	5	GOLD LOAN	2,14,569	4	16,324	3	4
15616550	33	MALE	DOCTOR	MARRIED	6	70000	33541	8	BUILDING	7,45,213	4	19,541	1	3
15630053	56	MALE	FIRE DEPARTMENT	MARRIED	6	67890	34567	5	TRAVELLING	6,78,500	5	13,560	3	4
15804771	58	MALE	SYSTEM ENGINEER	MARRIED	6	76800	null	5	TRAVELLING	16,59,080	6	29,000	5	3
15773469	35	MALE	BANK MANAGER	MARRIED	4	930000	35680	6	HOUSING	6,79,040	5	34,000	5	5
15702014	54	MALE	AIRPORT OFFICER	MARRIED	6	80000	62541	2	AUTOMOBILE	20,45,789	1	16,599	2	3
15592461	44	FEMALE	ACCOUNT MANAGER	MARRIED	4	800000	15632	8	COMPUTER SOFTWARES	23,65,478	5	20,145	3	4
15638424	47	FEMALE	DOCTOR	MARRIED	4	72154	45286	4	AUTOMOBILE	7,54,126	2	19,524	5	2
15703793	54	MALE	AIRPORT OFFICER	MARRIED	6	80000	62541	2	HOUSING	20,45,789	1	16,599	2	3
15770811	24	MALE	DATA ANALYST	SINGLE	4	60111	28999	6	RESTAURANTS	35,232	5	33,333	1	2

only showing top 20 rows

Filter customers with more than 2 returned cheques and income less than 50,000

loan_df.filter((loan_df[' Returned Cheque'] >= 2) & (loan_df['Income'] < 50000)).show()

```python
# Filter customers with more than 2 returned cheques and income less than 50,000
loan_df.filter((loan_df[' Returned Cheque'] >= 2) & (loan_df['Income'] < 50000)).show()
```

Customer_Id	Age	Gender	Occupation	Marital Status	Family Size	Income	Expenditure	Use Frequency	Loan Category	Loan Amount	Overdue	Debt Record	Returned Cheque	Dishonour of Bill
15592531	39	FEMALE	TEACHER	MARRIED	6	46619	18675	4	HOUSING	12,09,867	8	29,999	6	8
15656148	51	MALE	SYSTEM MANAGER	MARRIED	3	49999	19111	5	RESTAURANTS	60,676	8	13,000	2	5
15792365	24	FEMALE	TEACHER	SINGLE	3	45008	17454	4	AUTOMOBILE	3,99,435	9	51,987	4	7
15632264	54	FEMALE	TEACHER	MARRIED	5	48099	19999	4	RESTAURANTS	30,999	1	12,000	7	5
15691483	45	MALE	ACCOUNT MANAGER	MARRIED	7	45777	18452	4	GOLD LOAN	9,87,611	7	39,999	8	1
15788218	49	MALE	BANK MANAGER	MARRIED	4	45999	14500	4	TRAVELLING	79,999	4	6,700	7	3
15597945	36	FEMALE	CLERK	MARRIED	4	35000	15000	3	HOUSING	3,00,000	2	5,600	4	8
15699309	40	MALE	PUBLIC WORKS	MARRIED	4	38000	20000	3	GOLD LOAN	4,00,000	9	19,954	3	2
15725737	45	FEMALE	FIRE DEPARTMENT	MARRIED	4	40000	18888	4	AUTOMOBILE	70,000	1	0	2	1
15736816	30	MALE	ELECTRICIAN	MARRIED	4	30000	15000	5	HOUSING	3,54,789	5	32,154	5	5
15700772	51	FEMALE	TECHNICIAN	MARRIED	5	30000	null	5	RESTAURANTS	1,25,463	7	52,634	4	10
15589475	21	FEMALE	MANAGER	SINGLE	3	42516	24567	7	AUTOMOBILE	25,69,874	8	89,652	2	3
15750181	33	FEMALE	CLERK	MARRIED	3	35684	15247	3	RESTAURANTS	14,52,637	3	13,547	3	2
15788448	29	MALE	FIRE DEPARTMENT	MARRIED	5	45213	32457	9	TRAVELLING	15,24,789	7	90,000	2	5
15717426	56	MALE	DRIVER	MARRIED	5	30000	15426	7	TRAVELLING	9,21,456	6	20,000	4	6
15619360	49	MALE	ASSISTANT MANAGER	MARRIED	7	45612	39542	3	SHOPPING	5,87,412	7	65,412	3	2
15754849	36	MALE	ELECTRICIAN	MARRIED	2	36985	25648	6	AUTOMOBILE	9,85,413	7	20,000	5	3
15768193	36	MALE	ELECTRICIAN	MARRIED	2	36985	25648	6	ELECTRONICS	9,85,413	7	20,000	5	3
15683553	27	FEMALE	SOFTWARE ENGINEER	SINGLE	4	40000	22000	4	GOLD LOAN	4,00,000	4	15,647	5	3
15569590	34	FEMALE	TEACHER	MARRIED	4	45389	null	5	HOME APPLIANCES	3,50,050	4	24,000	4	3

only showing top 20 rows

Filter credit card users in Spain

credit_df.filter(credit_df['Geography'] == 'Spain').show()

```python
# Filter credit card users in Spain
credit_df.filter(credit_df['Geography'] == 'Spain').show()
```

RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	IsActiveMember	EstimatedSalary	Exited
2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	1	112542.58	0
5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	79084.1	0
6	15574012	Chu	645	Spain	Male	44	8	113755.78	2	0	149756.71	1
12	15737173	Andrews	497	Spain	Male	24	3	0.0	2	0	76390.01	0
15	15600882	Scott	635	Spain	Female	35	7	0.0	2	1	65951.65	0
18	15788218	Henderson	549	Spain	Female	24	9	0.0	2	1	14406.41	0
19	15661507	Muldrow	587	Spain	Male	45	6	0.0	1	0	158684.81	0
22	15597945	Dellucci	636	Spain	Female	32	8	0.0	2	0	138555.46	0
23	15699309	Gerasimov	510	Spain	Female	38	4	0.0	1	0	118913.53	1
31	15589475	Azikiwe	591	Spain	Female	39	3	0.0	3	0	140469.38	1
34	15659428	Maggard	520	Spain	Female	42	6	0.0	2	1	34410.55	0
35	15732963	Clements	722	Spain	Female	29	9	0.0	2	1	142033.07	0
37	15788448	Watson	490	Spain	Male	31	3	145260.23	1	1	114066.77	0
38	15729599	Lorenzo	804	Spain	Male	33	7	76548.6	1	1	98453.45	0
41	15619360	Hsiao	472	Spain	Male	40	4	0.0	1	0	70154.22	0
45	15684171	Bianchi	660	Spain	Female	61	5	155931.11	1	1	158338.39	0
59	15623944	T'ien	511	Spain	Female	66	4	0.0	1	0	1643.11	1
63	15702014	Jeffrey	555	Spain	Male	33	1	56084.69	2	0	178798.13	0
64	15751208	Pirozzi	684	Spain	Male	56	8	78707.16	1	1	99398.36	0
73	15812518	Palermo	657	Spain	Female	37	0	163607.18	1	1	44203.55	0

only showing top 20 rows

Filter loans with expenditure greater than 50,000 per month

loan_df.filter(loan_df['expenditure'] > 50000).show()

```python
# Filter loans with expenditure greater than 50,000 per month
loan_df.filter(loan_df['expenditure'] > 50000).show()
```

Customer_Id	Age	Gender	Occupation	Marital Status	Family Size	Income	Expenditure	Use Frequency	Loan Category	Loan Amount	Overdue	Debt Record	Returned Cheque	Dishonour of Bill
15702014	54	MALE	AIRPORT OFFICER	MARRIED	6	80000	62541	2	AUTOMOBILE	20,45,789	1	16,599	2	3
15703793	54	MALE	AIRPORT OFFICER	MARRIED	6	80000	62541	2	HOUSING	20,45,789	1	16,599	2	3
15805254	54	MALE	AIRPORT OFFICER	MARRIED	6	81000	62541	2	DINNING	20,45,789	1	16,599	2	3
15693683	54	MALE	AIRPORT OFFICER	MARRIED	6	80000	62541	2	HOUSING	20,45,789	1	16,599	2	3
15782688	41	MALE	BANK MANAGER	MARRIED	6	64125	51246	6	TRAVELLING	6,52,147	5	16,524	3	3
15663252	26	MALE	DIETICIAN	SINGLE	3	95425	53086	2	HOUSING	4,88,076	4	61227	5	2

JOINS:

Joins combine two or more datasets based on a common key, enabling richer analyses by bringing together data from different sources. In Spark SQL, joins can be performed using different types such as INNER JOIN, LEFT JOIN, or RIGHT JOIN. For example, joining the loan and credit card datasets on customer IDs allows us to analyze customer loan amounts along with their credit scores.

EXAMPLES:

Join loan and credit card datasets on customer_id and customerid

loan_df.join(credit_df, loan_df['Customer_Id'] == credit_df['CustomerId'], 'inner') \

.select(loan_df['Customer_Id'], loan_df['Loan Amount'], credit_df['Balance']).show()

```
# Join loan and credit card datasets on customer_id and customerid
loan_df.join(credit_df, loan_df['Customer_Id'] == credit_df['CustomerId'], 'inner') \
.select(loan_df['Customer_Id'], loan_df['Loan Amount'], credit_df['Balance']).show()
```

```
+-----------+-----------+---------+
|Customer_Id|Loan Amount|  Balance|
+-----------+-----------+---------+
|   15634602|  10,00,000|      0.0|
|   15647311|     50,000| 83807.86|
|   15619304|     75,000| 159660.8|
|   15701354|   6,00,000|      0.0|
|   15737888|   2,00,000|125510.82|
|   15574012|     47,787|113755.78|
|   15592531|  12,09,867|      0.0|
|   15656148|     60,676|115046.74|
|   15792365|   3,99,435|142051.07|
|   15592389|     60,999|134603.88|
|   15767821|     35,232|102016.72|
|   15737173|     80,660|      0.0|
|   15632264|     30,999|      0.0|
|   15691483|   9,87,611|      0.0|
|   15600882|   5,99,934|      0.0|
|   15643966|  12,90,929|143129.41|
|   15737452|   1,67,654|132602.88|
|   15788218|     79,999|      0.0|
|   15661507|  10,65,577|      0.0|
|   15568982|   9,00,000|      0.0|
+-----------+-----------+---------+
only showing top 20 rows
```

Join loan dataset with credit card dataset to get loan amount and estimated salary

loan_df.join(credit_df, loan_df['Customer_Id'] == credit_df['CustomerId'], 'inner') \

.select(loan_df['Customer_Id'], loan_df['Loan Amount'], credit_df['EstimatedSalary']).show()

```
# Join loan dataset with credit card dataset to get loan amount and estimated salary
loan_df.join(credit_df, loan_df['Customer_Id'] == credit_df['CustomerId'], 'inner') \
.select(loan_df['Customer_Id'], loan_df['Loan Amount'], credit_df['EstimatedSalary']).show()
```

```
+-----------+-----------+---------------+
|Customer_Id|Loan Amount|EstimatedSalary|
+-----------+-----------+---------------+
|   15634602|  10,00,000|      101348.88|
|   15647311|     50,000|      112542.58|
|   15619304|     75,000|      113931.57|
|   15701354|   6,00,000|       93826.63|
|   15737888|   2,00,000|        79084.1|
|   15574012|     47,787|      149756.71|
|   15592531|  12,09,867|        10062.8|
|   15656148|     60,676|      119346.88|
|   15792365|   3,99,435|        74940.5|
|   15592389|     60,999|       71725.73|
|   15767821|     35,232|       80181.12|
|   15737173|     80,660|       76390.01|
|   15632264|     30,999|       26260.98|
|   15691483|   9,87,611|      190857.79|
|   15600882|   5,99,934|       65951.65|
|   15643966|  12,90,929|       64327.26|
|   15737452|   1,67,654|        5097.67|
|   15788218|     79,999|       14406.41|
|   15661507|  10,65,577|      158684.81|
|   15568982|   9,00,000|       54724.03|
+-----------+-----------+---------------+
only showing top 20 rows
```

Join loan dataset with credit dataset and filter by customers aged 30 and above

loan_df.join(credit_df, loan_df['Customer_Id'] == credit_df['CustomerId'], 'inner') \
.filter(loan_df['age'] >= 30) \
.select(loan_df['Customer_Id'], loan_df['Loan Amount'], credit_df['age']).show()

```
# Join loan dataset with credit dataset and filter by customers aged 30 and above
loan_df.join(credit_df, loan_df['Customer_Id'] == credit_df['CustomerId'], 'inner') \
.filter(loan_df['age'] >= 30) \
.select(loan_df['Customer_Id'], loan_df['Loan Amount'], credit_df['age']).show()
```

```
+-----------+-----------+---+
|Customer_Id|Loan Amount|age|
+-----------+-----------+---+
|   15634602|  10,00,000| 42|
|   15647311|     50,000| 41|
|   15619304|     75,000| 42|
|   15737888|   2,00,000| 43|
|   15574012|     47,787| 44|
|   15592531|  12,09,867| 50|
|   15656148|     60,676| 29|
|   15592389|     60,999| 27|
|   15737173|     80,660| 24|
|   15632264|     30,999| 34|
|   15691483|   9,87,611| 25|
|   15600882|   5,99,934| 35|
|   15737452|   1,67,654| 58|
|   15788218|     79,999| 24|
|   15661507|  10,65,577| 45|
|   15568982|   9,00,000| 24|
|   15597945|   3,00,000| 32|
|   15699309|   4,00,000| 38|
|   15725737|     70,000| 46|
|   15738191|   2,57,789| 25|
+-----------+-----------+---+
only showing top 20 rows
```

Join loan dataset with credit dataset to get credit score for customers with housing loans

loan_df.join(credit_df, loan_df['Customer_Id'] == credit_df['CustomerId'], 'inner') \
.filter(loan_df['loan category'] == 'HOUSING') \
.select(loan_df['Customer_Id'], loan_df['Loan Amount'], credit_df['CreditScore']).show()

```
# Join loan dataset with credit dataset to get credit score for customers with housing loans
loan_df.join(credit_df, loan_df['Customer_Id'] == credit_df['CustomerId'], 'inner') \
.filter(loan_df['loan category'] == 'HOUSING') \
.select(loan_df['Customer_Id'], loan_df['Loan Amount'], credit_df['CreditScore']).show()
```

```
+-----------+-----------+-----------+
|Customer_Id|Loan Amount|CreditScore|
+-----------+-----------+-----------+
|   15634602|  10,00,000|        619|
|   15592531|  12,09,867|        822|
|   15661507|  10,65,577|        587|
|   15568982|   9,00,000|        726|
|   15597945|   3,00,000|        636|
|   15736816|   3,54,789|        756|
|   15706552|   9,85,412|        533|
|   15684171|   7,45,213|        660|
|   15773469|   6,79,040|        687|
|   15703793|  20,45,789|        738|
|   15625759|   3,00,000|        729|
|   15738721|  10,65,577|        773|
|   15693683|  20,45,789|        814|
|   15715951|  20,45,789|        562|
|   15740404|   3,00,000|        758|
|   15712543|   4,77,870|        789|
|   15640905|  20,45,789|        579|
|   15724944|  10,65,577|        663|
|   15628145|   3,54,789|        682|
|   15754105|   9,85,412|        650|
+-----------+-----------+-----------+
only showing top 20 rows
```

Join loan dataset with credit dataset and show customers with overdue loans and high credit score

loan_df.join(credit_df, loan_df['Customer_Id'] == credit_df['CustomerId'], 'inner') \
.filter(loan_df['overdue'] > 0) \
.filter(credit_df['creditscore'] > 700) \
.select(loan_df['Customer_Id'], loan_df['Loan Amount'], credit_df['CreditScore']).show()

```
# Join loan dataset with credit dataset and show customers with overdue loans and high credit score
loan_df.join(credit_df, loan_df['Customer_Id'] == credit_df['CustomerId'], 'inner') \
.filter(loan_df['overdue'] > 0) \
.filter(credit_df['creditscore'] > 700) \
.select(loan_df['Customer_Id'], loan_df['Loan Amount'], credit_df['CreditScore']).show()
```

```
+-----------+-----------+-----------+
|Customer_Id|Loan Amount|CreditScore|
+-----------+-----------+-----------+
|   15737888|   2,00,000|        850|
|   15592531|  12,09,867|        822|
|   15568982|   9,00,000|        726|
|   15577657|   4,00,000|        732|
|   15625047|   1,00,000|        846|
|   15736816|   3,54,789|        756|
|   15732963|   8,52,416|        722|
|   15729599|  23,65,478|        804|
|   15717426|   9,21,456|        850|
|   15755196|   6,54,120|        834|
|   15754849|   9,85,413|        776|
|   15602280|  52,14,789|        829|
|   15771873|   7,85,241|        776|
|   15683553|   4,00,000|        788|
|   15647091|   8,54,000|        725|
|   15651280|   7,89,000|        742|
|   15789484|   9,21,456|        751|
|   15641582|   5,87,412|        735|
|   15703793|  20,45,789|        738|
|   15620344|     60,676|        813|
+-----------+-----------+-----------+
only showing top 20 rows
```

AGGREGATIONS:

Aggregation transforms datasets by summarizing data using functions like SUM, AVG, MAX, MIN, and COUNT. In Spark SQL, aggregations provide insights into the overall characteristics of the data. For instance, you can calculate the total loan amount for each loan category or find the average income of customers.

EXAMPLES:

Calculate the average income for each occupation in the loan dataset

loan_df.groupBy('Occupation').avg('Income').withColumnRenamed('avg(Income)', 'Average_Income').show()

```
# Calculate the average income for each occupation in the loan dataset
loan_df.groupBy('Occupation').avg('Income').withColumnRenamed('avg(Income)', 'Average_Income').show()
```

```
+--------------------+------------------+
|          Occupation|    Average_Income|
+--------------------+------------------+
|      CIVIL ENGINEER|60359.666666666664|
|     FIRE DEPARTMENT|55357.916666666664|
|          ACCOUNTANT| 56623.28571428572|
|        BANK MANAGER|           92191.0|
|      SYSTEM OFFICER|           56780.0|
|           NUTRITION|           55650.0|
|           DIETICIAN| 72599.16666666667|
|               CLERK|         76871.125|
|   SOFTWARE ENGINEER|           61107.8|
|AGRICULTURAL ENGI...|         82060.625|
|   ASSISTANT MANAGER|54866.166666666664|
|             TEACHER| 52812.73333333333|
| ASSISTANT PROFESSOR|53319.333333333336|
|     SYSTEM ENGINEER|60509.333333333336|
| CHARTERED APPRAISER| 76456.72727272728|
|                NAVY|        71190.9375|
|              POLICE| 49049.88888888889|
|            BUSINESS|        56682.5625|
|              FARMER| 74906.85714285714|
|              DRIVER|64450.833333333336|
+--------------------+------------------+
only showing top 20 rows
```

Find the maximum balance in the credit card dataset

credit_df.agg({"Balance": "max"}).withColumnRenamed("max(Balance)", "Max_Balance").show()

```
# Find the maximum balance in the credit card dataset
credit_df.agg({"Balance": "max"}).withColumnRenamed("max(Balance)", "Max_Balance").show()
```

```
+-----------+
|Max_Balance|
+-----------+
|  250898.09|
+-----------+
```

Find the total number of products for each customer in the credit card dataset

credit_df.groupBy('CustomerId').sum('NumOfProducts').withColumnRenamed('sum(NumOfProducts)', 'Total_Products').show()

```
# Find the total number of products for each customer in the credit card dataset
credit_df.groupBy('CustomerId').sum('NumOfProducts').withColumnRenamed('sum(NumOfProducts)', 'Total_Products').show()
```

```
+----------+--------------+
|CustomerId|Total_Products|
+----------+--------------+
|  15632264|             2|
|  15613854|             2|
|  15662403|             2|
|  15672012|             1|
|  15724563|             2|
|  15793949|             1|
|  15721292|             2|
|  15763612|             2|
|  15734491|             2|
|  15590268|             1|
|  15747980|             2|
|  15574167|             1|
|  15671766|             1|
|  15576928|             1|
|  15630661|             1|
|  15612893|             1|
|  15760121|             1|
|  15694890|             1|
|  15661330|             1|
|  15806913|             1|
+----------+--------------+
only showing top 20 rows
```

Calculate the average credit score by geography in the credit card dataset

credit_df.groupBy('Geography').avg('CreditScore').withColumnRenamed('avg(CreditScore)', 'Average_Credit_Score').show()

```
# Calculate the average credit score by geography in the credit card dataset
credit_df.groupBy('Geography').avg('CreditScore').withColumnRenamed('avg(CreditScore)', 'Average_Credit_Score').show()
```

```
+---------+--------------------+
|Geography|Average_Credit_Score|
+---------+--------------------+
|  Germany|   651.4535671582304|
|   France|   649.6683286796969|
|    Spain|   651.3338716188938|
+---------+--------------------+
```

GROUPBY:

The GROUP BY transformation organizes data into groups based on one or more columns and applies aggregation functions to each group. It is especially useful for analyzing patterns and comparisons within subsets of data.

EXAMPLES:

Group by Marital Status and count the number of customers in each category in loan dataset

loan_df.groupBy('Marital Status').count().withColumnRenamed('count', 'Customer_Count').show()

```
# Group by Marital Status and count the number of customers in each category in loan dataset
loan_df.groupBy('Marital Status').count().withColumnRenamed('count', 'Customer_Count').show()
```

```
+--------------+--------------+
|Marital Status|Customer_Count|
+--------------+--------------+
|        SINGLE|           146|
|       MARRIED|           354|
+--------------+--------------+
```

Group by Loan Category and calculate the average expenditure for each category in loan dataset

loan_df.groupBy('Loan Category').agg({'Expenditure': 'avg'}).withColumnRenamed('avg(Expenditure)', 'Average_Expenditure').show()

```
# Group by Loan Category and calculate the average expenditure for each category in loan dataset
loan_df.groupBy('Loan Category').agg({'Expenditure': 'avg'}).withColumnRenamed('avg(Expenditure)', 'Average_Expenditure').show()
```

```
+-----------------+-------------------+
|    Loan Category|Average_Expenditure|
+-----------------+-------------------+
|          HOUSING| 29052.666666666668|
|        TRAVELLING|          26211.125|
|       BOOK STORES|            21221.0|
|       AGRICULTURE|            30573.5|
|         GOLD LOAN|  26168.61842105263|
|  EDUCATIONAL LOAN|            31088.6|
|        AUTOMOBILE| 26787.660714285714|
|          BUSINESS|            31431.0|
|COMPUTER SOFTWARES| 26157.363636363636|
|           DINNING| 27934.285714285714|
|          SHOPPING| 26654.272727272728|
|       RESTAURANTS|            25398.0|
|       ELECTRONICS|  26123.46153846154|
|          BUILDING| 36014.857142857145|
|        RESTAURANT|           30609.75|
|   HOME APPLIANCES| 27622.384615384617|
+-----------------+-------------------+
```

Group by Occupation and calculate the number of customers for each occupation in loan dataset

loan_df.groupBy('Occupation').count().withColumnRenamed('count', 'Customer_Count').show()

```
# Group by Occupation and calculate the number of customers for each occupation in loan dataset
loan_df.groupBy('Occupation').count().withColumnRenamed('count', 'Customer_Count').show()
```

```
+--------------------+--------------+
|          Occupation|Customer_Count|
+--------------------+--------------+
|      CIVIL ENGINEER|             6|
|     FIRE DEPARTMENT|            12|
|          ACCOUNTANT|             7|
|        BANK MANAGER|            28|
|      SYSTEM OFFICER|             4|
|           NUTRITION|             1|
|           DIETICIAN|            13|
|               CLERK|            26|
|   SOFTWARE ENGINEER|            35|
|AGRICULTURAL ENGI...|             8|
|   ASSISTANT MANAGER|             6|
|             TEACHER|            63|
| ASSISTANT PROFESSOR|             9|
|     SYSTEM ENGINEER|             3|
| CHARTERED APPRAISER|            11|
|                NAVY|            16|
|              POLICE|            18|
|            BUSINESS|            16|
|              FARMER|             7|
|              DRIVER|            18|
+--------------------+--------------+
only showing top 20 rows
```