ASSIGNMENT 4 - SPARK DATAFRAME

SRUTHY SUJI
23AD140
AI-DS 'C'

a) Create a new Spark Session with new SparkConfig

```
In [7]: sc.stop()

In [8]: from pyspark import SparkConf,SparkContext
        config= SparkConf().setMaster("local[2]").setAppName("Sruthy_SparkAssignm
        sc= SparkContext(conf=config)

In [9]: from pyspark.sql import SparkSession
        spark = SparkSession.builder.appName("SQLSession").getOrCreate()

In [10]: spark
```

Out[10]:

SparkSession - hive

SparkContext

Spark UI

Version
`v2.4.8`

Master
`local[2]`

AppName
`Sruthy_SparkAssignment`

b) Create new instance of Spark SQL session and define new DataFrame using sales_data_sample.csv dataset.

```
In [11]: !hadoop fs -mkdir /assignment

In [12]: !hadoop fs -put /home/hadoop/Downloads/sales_data_sample.csv /assignment

In [13]: sales_df = spark.read.csv("hdfs://localhost:9000/assignment/sales_data_sample.csv",header=True,inferSchema=True)

In [53]: sales_df.rdd.getNumPartitions()

Out[53]: 2

In [57]: sales_df = spark.read.parquet("hdfs://localhost:9000/assignment/parquet_sales")

In [58]: sales_df.toPandas()
```

Out[58]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	ADDRESS
0	10248	32	75.89	4	2428.48	5/7/2004 0:00	Cancelled	2	5	2004	...	897 Long, A
1	10363	43	100.00	9	5154.41	1/6/2005 0:00	Shipped	1	1	2005	...	Sc Engin Center, S
2	10127	20	96.99	7	1939.80	6/3/2003 0:00	Shipped	2	6	2003	...	4092 Furth
3	10276	38	83.79	4	3184.02	8/2/2004 0:00	Shipped	3	8	2004	...	7635 Spi
4	10391	42	100.00	3	4998.00	3/9/2005 0:00	Shipped	1	3	2005	...	201 Miller
...	...	...	...	...	...	...	...	...	...	...	...	...
2818	10389	34	100.00	2	3441.82	2/16/2005	Shipped	1	2	2005		C/ Moral

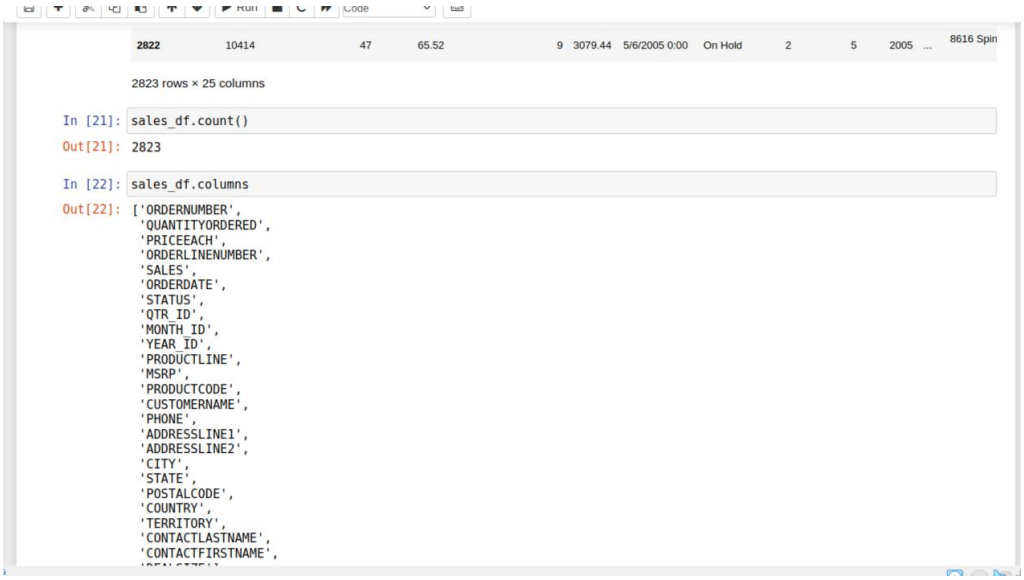
Browse Directory

/assignment/parquet_sales Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hadoop	supergroup	0 B	15/7/2025, 7:06:33 pm	1	128 MB	_SUCCESS
-rw-r--r--	hadoop	supergroup	49.92 KB	15/7/2025, 7:06:33 pm	1	128 MB	part-00000-8df997a2-4617-4f61-8c09-760dd20f8ba1-c000.snappy.parquet
-rw-r--r--	hadoop	supergroup	49.88 KB	15/7/2025, 7:06:33 pm	1	128 MB	part-00001-8df997a2-4617-4f61-8c09-760dd20f8ba1-c000.snappy.parquet

c) Find the shape of DataFrame.

Shape = 2823 rows X 25 columns



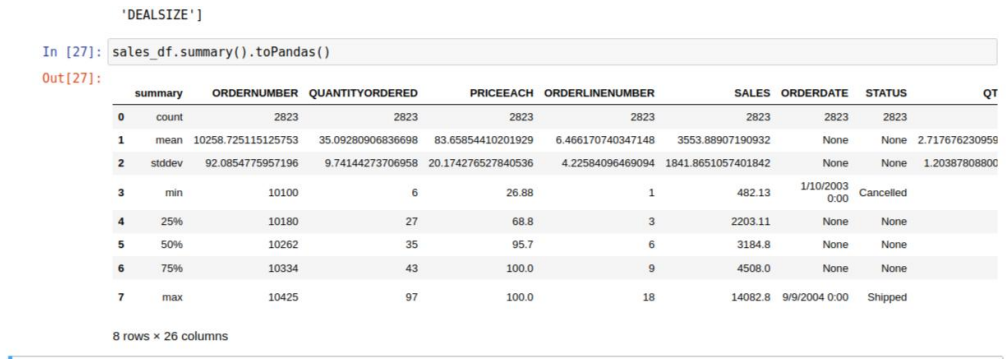
2822	10414	47	65.52	9	3079.44	5/6/2005 0:00	On Hold	2	5	2005	...	8616 Spin

2823 rows × 25 columns

```
In [21]: sales_df.count()
Out[21]: 2823

In [22]: sales_df.columns
Out[22]: ['ORDERNUMBER',
          'QUANTITYORDERED',
          'PRICEEACH',
          'ORDERLINENUMBER',
          'SALES',
          'ORDERDATE',
          'STATUS',
          'QTR_ID',
          'MONTH_ID',
          'YEAR_ID',
          'PRODUCTLINE',
          'MSRP',
          'PRODUCTCODE',
          'CUSTOMERNAME',
          'PHONE',
          'ADDRESSLINE1',
          'ADDRESSLINE2',
          'CITY',
          'STATE',
          'POSTALCODE',
          'COUNTRY',
          'TERRITORY',
          'CONTACTLASTNAME',
          'CONTACTFIRSTNAME',
```

d) Find the Summary of DataFrame for all numerical data columns.



```
'DEALSIZE']

In [27]: sales_df.summary().toPandas()
```

Out[27]:

	summary	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QT
0	count	2823	2823	2823	2823	2823	2823	2823	
1	mean	10258.725115125753	35.09280906836698	83.65854410201929	6.466170740347148	3553.88907190932	None	None	2.717676230959
2	stddev	92.0854775957196	9.74144273706958	20.174276527840536	4.22584096469094	1841.8651057401842	None	None	1.20387808800
3	min	10100	6	26.88	1	482.13	1/10/2003 0:00	Cancelled	
4	25%	10180	27	68.8	3	2203.11	None	None	
5	50%	10262	35	95.7	6	3184.8	None	None	
6	75%	10334	43	100.0	9	4508.0	None	None	
7	max	10425	97	100.0	18	14082.8	9/9/2004 0:00	Shipped	

8 rows × 26 columns

e) Identify and handle missing or null values in the columns.

```
In [33]: from pyspark.sql.functions import col,when,isnull,count
         sales_df.filter(col('ordernumber').isNull()).count()
Out[33]: 0

In [37]: sales_df.filter(col('QUANTITYORDERED').isNull()).count()
Out[37]: 0

In [36]: sales_df.filter(col('PRICEEACH').isNull()).count()
Out[36]: 0

In [38]: sales_df.filter(col('orderlinenumber').isNull()).count()
Out[38]: 0

In [39]: sales_df.filter(col('sales').isNull()).count()
Out[39]: 0

In [40]: sales_df.filter(col('orderdate').isNull()).count()
Out[40]: 0

In [41]: sales_df.filter(col('status').isNull()).count()
Out[41]: 0

In [42]: sales_df.filter(col('QTR_ID').isNull()).count()
Out[42]: 0

In [43]: sales_df.filter(col('MONTH_ID').isNull()).count()
Out[43]: 0

In [44]: sales_df.filter(col('YEAR_ID').isNull()).count()
Out[44]: 0

In [46]: sales_df.select([count(when(isnull(column), column)).alias(column) for column in sales_df.columns]).toPandas()
```

Out[46]:

D	YEAR_ID	...	ADDRESSLINE1	ADDRESSLINE2	CITY	STATE	POSTALCODE	COUNTRY	TERRITORY	CONTACTLASTNAME	CONTACTFIRSTNAME	DEALSIZE
0	0	...	0	2521	0	1486	76	0	0	0	0	0

f) Calculate the total revenue generated per country by combining the columns QUANTITYORDERED and PRICEEACH using Spark DataFrame operations?

```
In [64]: sales_df.cache()
```

```
Out[64]: DataFrame[ORDERNUMBER: int, QUANTITYORDERED: int, PRICEEACH: double, ORDERLINENUMBER: int, SALES: double, ORDERDATE: string, STATUS: string, QTR_ID: int, MONTH_ID: int, YEAR_ID: int, PRODUCTLINE: string, MSRP: int, PRODUCTCODE: string, CUSTOMERNAME: string, PHONE: string, ADDRESSLINE1: string, ADDRESSLINE2: string, CITY: string, STATE: string, POSTALCODE: string, COUNTRY: string, TERRITORY: string, CONTACTLASTNAME: string, CONTACTFIRSTNAME: string, DEALSIZE: string]
```

```
In [65]: productsales = sales_df.withColumn("Revenue", col("QUANTITYORDERED") * col("PRICEEACH"))
productsales.toPandas()
```

Out[65]:

ID	YEAR_ID	...	ADDRESSLINE2	CITY	STATE	POSTALCODE	COUNTRY	TERRITORY	CONTACTLASTNAME	CONTACTFIRSTNAME	DEALSIZE	Revenue
5	2004	...	None	NYC	NY	10022	USA	NA	Yu	Kwai	Small	2428.48
1	2005	...	None	Espoo	None	FIN-02271	Finland	EMEA	Suominen	Kalle	Medium	4300.00
6	2003	...	Suite 400	NYC	NY	10022	USA	NA	Young	Jeff	Small	1939.80
8	2004	...	None	Brickhaven	MA	58339	USA	NA	Barajas	Miguel	Medium	3184.02
3	2005	...	Level 15	North Sydney	NSW	2060	Australia	APAC	O'Hara	Anna	Medium	4200.00
...	...	...	...	...	...	...	...	...	...	...	...	...
2	2005	...	None	Madrid	None	28034	Spain	EMEA	Freyre	Diego	Medium	3400.00
9	2003	...	None	Madrid	None	28034	Spain	EMEA	Freyre	Diego	Small	2573.46
9	2004	...	2nd Floor	Singapore	None	69045	Singapore	APAC	Victorino	Wendy	Small	2328.66
10	2004	...	None	Manchester	None	EC2 5NT	UK	EMEA	Ashworth	Victoria	Medium	2900.00
5	2004	...	None	Madrid	None	28034	Spain	EMEA	Freyre	Diego	Medium	2900.00

g) Determine the top 5 products with the highest total sales revenue using Spark DataFrame?

```
In [67]: productsales.orderBy("Revenue", ascending=False).toPandas()
```

Out[67]:

ID	YEAR_ID	...	ADDRESSLINE2	CITY	STATE	POSTALCODE	COUNTRY	TERRITORY	CONTACTLASTNAME	CONTACTFIRSTNAME	DEALSIZE	Revenue
4	2005	...	None	Strasbourg	None	67000	France	EMEA	Citeaux	Frederique	Large	9048.16
4	2005	...	None	Strasbourg	None	67000	France	EMEA	Citeaux	Frederique	Large	7600.00
4	2005	...	None	San Jose	CA	94217	USA	NA	Frick	Sue	Large	7600.00
4	2005	...	None	Newark	NJ	94019	USA	NA	Brown	William	Large	7543.75
4	2005	...	None	San Jose	CA	94217	USA	NA	Frick	Sue	Large	7182.00
...	...	...	...	...	...	...	...	...	...	...	...	...
4	2005	...	2nd Floor	Singapore	None	69045	Singapore	APAC	Victorino	Wendy	Small	600.00
8	2004	...	None	Torino	None	10100	Italy	EMEA	Accorti	Paolo	Small	577.60
4	2005	...	None	Minato-ku	Tokyo	106-0032	Japan	Japan	Shimamura	Akiko	Small	553.95
4	2005	...	None	San Jose	CA	94217	USA	NA	Frick	Sue	Small	541.14
5	2005	...	None	Nantes	None	44000	France	EMEA	Labrune	Janine	Small	482.13

h)Find the average order quantity for each product using groupBy and agg operations?

```
In [69]: from pyspark.sql.functions import avg

avgquantity = sales_df.groupBy("PRODUCTCODE").agg( avg("QUANTITYORDERED").alias("AverageOrderQuantity"))
avgquantity.orderBy("PRODUCTCODE").show()
```

```
+-----------+--------------------+
|PRODUCTCODE|AverageOrderQuantity|
+-----------+--------------------+
|   S10_1678|   36.30769230769231|
|   S10_1949|   34.32142857142857|
|   S10_2016|   35.69230769230769|
|   S10_4698|   35.42307692307692|
|   S10_4757|   35.25925925925926|
|   S10_4962|  33.285714285714285|
|   S12_1099|               33.52|
|   S12_1108|   37.42307692307692|
|   S12_1666|  34.714285714285715|
|   S12_2823|   37.07692307692308|
|   S12_3148|               35.92|
|   S12_3380|               34.12|
|   S12_3891|   35.42307692307692|
|   S12_3990|                32.0|
|   S12_4473|  37.925925925925924|
|   S12_4675|   37.07692307692308|
|   S18_1097|   35.67857142857143|
|   S18_1129|  35.074074074074076|
|   S18_1342|   38.34615384615385|
|   S18_1367|   34.23076923076923|
+-----------+--------------------+
only showing top 20 rows
```

i)Using Spark DataFrame, filter orders where the SALES value exceeds $10,000 and sort the results by the ORDERDATE column?

```
In [76]: high_sales_df = sales_df.filter(col("SALES") > 10000).orderBy("ORDERDATE")
         high_sales_df.toPandas()
```

Out[76]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	ADDRESSLIN
0	10304	47	100.0	6	10172.7	10/11/2004 0:00	Shipped	4	10	2004	...	67, avenue l'Euro
1	10312	48	100.0	3	11623.7	10/21/2004 0:00	Shipped	4	10	2004	...	5677 Strong
2	10333	46	100.0	2	11336.7	11/18/2004 0:00	Shipped	4	11	2004	...	5557 No Pendale Str
3	10339	55	100.0	13	10758.0	11/23/2004 0:00	Shipped	4	11	2004	...	2-2-8 Roppo
4	10322	50	100.0	6	12536.5	11/4/2004 0:00	Shipped	4	11	2004	...	2304 Lo Airport Aver
5	10375	43	100.0	2	10039.6	2/3/2005 0:00	Shipped	1	2	2005	...	67, rue d Cinquar Otag
6	10388	46	100.0	2	10066.6	3/3/2005 0:00	Shipped	1	3	2005	...	1785 First Str
7	10405	76	100.0	3	11739.7	4/14/2005 0:00	Shipped	2	4	2005	...	24, place Klul
8	10406	65	100.0	1	10468.9	4/15/2005 0:00	Disputed	2	4	2005	...	Vinb'ltet
9	10407	76	100.0	2	14082.8	4/22/2005 0:00	On Hold	2	4	2005	...	3086 Ingle
10	10403	66	100.0	9	11886.6	4/8/2005 0:00	Shipped	2	4	2005	...	Berke Gardens Brew

j) Filter out rows where the STATUS is 'Cancelled' and calculate the total sales from the remaining orders?

```
In [90]: from pyspark.sql.functions import col, sum
         filtered_df = sales_df.filter(col("STATUS") != "Cancelled")
         totalsales=filtered_df.agg(sum("SALES").alias("TotalSales")).show()
```

```
+------------------+
|        TotalSales|
+------------------+
|9838141.370000005|
+------------------+
```

K) Use Spark Data Frame transformations to derive the yearly sales for each customer (CUSTOMERNAME) based on the ORDERDATE column?

```
In [94]: sales_df = sales_df.withColumn("Year", year(col("ORDERDATE")))

         yearly_customer = sales_df.groupBy("Year", "CUSTOMERNAME").agg(sum("SALES").alias("TotalSales")).orderBy("Year", "CUS
         yearly_customer.show()
```

```
+----+--------------------+------------------+
|Year|        CUSTOMERNAME|        TotalSales|
+----+--------------------+------------------+
|null|      AV Stores, Co.|         157807.81|
|null|        Alpha Cognac|          70488.44|
|null|  Amica Models & Co.| 94117.26000000001|
|null|Anna's Decoration...|153996.12999999998|
|null|    Atelier graphique|          24179.96|
|null|Australian Collec...| 64591.46000000001|
|null|Australian Collec...|200995.40999999997|
|null|Australian Gift N...|          59469.12|
|null|  Auto Assoc. & Cie.| 64834.32000000001|
|null|    Auto Canal Petit|          93170.66|
|null|Auto-Moto Classic...|26479.260000000002|
|null|  Baane Mini Imports|         116599.19|
|null|Bavarian Collecta...|          34993.92|
|null|Blauer See Auto, Co.|          85171.59|
|null|   Boards & Toys Co.| 9129.349999999999|
|null|         CAF Imports|          49642.05|
|null|Cambridge Collect...|          36163.62|
|null|Canadian Gift Exc...|          75238.92|
|null|Classic Gift Idea...|          67506.97|
|null|Classic Legends Inc.| 77795.20000000001|
+----+--------------------+------------------+
only showing top 20 rows
```

l) Add a new column to the DataFrame that categorizes orders as "High", "Medium", or "Low" sales based on the SALES value?

```
In [96]: from pyspark.sql.functions import when, col
         sales_df = sales_df.withColumn(
             "SalesCategory",
             when(col("SALES") >= 10000, "High").when(col("SALES") >= 5000, "Medium").otherwise("Low")
         )
         sales_df.select("SALES", "SalesCategory").show()
```

```
+-------+-------------+
|  SALES|SalesCategory|
+-------+-------------+
|2428.48|          Low|
|5154.41|       Medium|
| 1939.8|          Low|
|3184.02|          Low|
| 4998.0|          Low|
| 5875.2|       Medium|
|2694.15|          Low|
| 2443.6|          Low|
|3035.88|          Low|
| 4829.8|          Low|
|4816.08|          Low|
| 4910.4|          Low|
|7048.14|       Medium|
| 3053.7|          Low|
|7016.31|       Medium|
|3584.25|          Low|
|4753.49|          Low|
|5399.55|       Medium|
|1695.96|          Low|
|3402.35|          Low|
```

m) Assume, If you have another DataFrame with customer demographic data, how would you perform a join to compute the total sales per demographic group?

```
In [99]: sales_by_demographics= sales_df.groupBy("COUNTRY").sum("SALES")
         sales_by_demographics = sales_by_demographics.withColumn("TotalSales", col("sum(SALES)"))
         sales_by_demographics.select("COUNTRY", "TotalSales").orderBy("TotalSales", ascending=False).show()
```

```
+-----------+-----------------+
|    COUNTRY|       TotalSales|
+-----------+-----------------+
|        USA|       3627982.83|
|      Spain|1215686.9199999995|
|     France|1110916.5199999998|
|  Australia| 630623.1000000001|
|         UK| 478880.4600000001|
|      Italy|374674.31000000006|
|    Finland|329581.91000000003|
|     Norway|307463.70000000007|
|  Singapore|288488.41000000003|
|    Denmark|245637.14999999997|
|     Canada|224078.56000000006|
|    Germany|220472.09000000003|
|     Sweden|210014.21000000002|
|    Austria|202062.53000000003|
|      Japan|188167.80999999997|
|Switzerland|         117713.56|
|    Belgium|         108412.62|
|Philippines|          94015.73|
|    Ireland|          57756.43|
+-----------+-----------------+
```

n) Can you implement a cumulative distribution function (CDF) over the SALES value for each CUSTOMERNAME? What insights can you gather from analyzing the CDF distribution for each customer?

```
In [100]: from pyspark.sql.functions import percent_rank, col
          from pyspark.sql.window import Window

          window_spec = Window.partitionBy("CUSTOMERNAME").orderBy(col("SALES"))
          cdf_df = sales_df.withColumn("Sales_CDF", percent_rank().over(window_spec))
          cdf_df.select("CUSTOMERNAME", "SALES", "Sales_CDF").orderBy("CUSTOMERNAME", "Sales").show()
```

```
+--------------+-------+---------+
|  CUSTOMERNAME|  SALES|Sales_CDF|
+--------------+-------+---------+
|AV Stores, Co.|  710.2|      0.0|
|AV Stores, Co.|1152.58|     0.02|
|AV Stores, Co.|1264.08|     0.04|
|AV Stores, Co.|1307.32|     0.06|
|AV Stores, Co.|1538.55|     0.08|
|AV Stores, Co.| 1592.0|      0.1|
|AV Stores, Co.| 1608.0|     0.12|
|AV Stores, Co.|1654.56|     0.14|
|AV Stores, Co.|1721.73|     0.16|
|AV Stores, Co.|1729.65|     0.18|
|AV Stores, Co.| 1759.2|      0.2|
|AV Stores, Co.|1859.44|     0.22|
|AV Stores, Co.|1987.74|     0.24|
|AV Stores, Co.|2051.56|     0.26|
|AV Stores, Co.|2082.85|     0.28|
|AV Stores, Co.|2223.52|      0.3|
|AV Stores, Co.|2264.15|     0.32|
|AV Stores, Co.|2315.18|     0.34|
|AV Stores, Co.|2427.03|     0.36|
|AV Stores, Co.|2499.56|     0.38|
+--------------+-------+---------+
```

o) Write spark dataframe code to rank products by total revenue within each country (COUNTRY)?

```
In [104]: country_revenue = productsales.groupBy("COUNTRY", "PRODUCTLINE").sum("Revenue")
          window = Window.partitionBy("COUNTRY").orderBy(col("sum(Revenue)").desc())
          ranked = country_revenue.withColumn("Rank", dense_rank().over(window))

          ranked.select("COUNTRY", "PRODUCTLINE", "sum(Revenue)", "Rank").show()
```

```
+-----------+----------------+------------------+----+
|    COUNTRY|     PRODUCTLINE|      sum(Revenue)|Rank|
+-----------+----------------+------------------+----+
|     Sweden|    Classic Cars|50377.619999999995|   1|
|     Sweden|Trucks and Buses|39562.439999999995|   2|
|     Sweden|    Vintage Cars|          31784.94|   3|
|     Sweden|           Ships|          29514.62|   4|
|     Sweden|     Motorcycles|           12388.6|   5|
|     Sweden|          Planes|           7435.88|   6|
|     Sweden|          Trains|            3200.0|   7|
|Philippines|    Classic Cars|          43815.85|   1|
|Philippines|     Motorcycles|           17491.9|   2|
|Philippines|          Planes|          17048.33|   3|
|Philippines|    Vintage Cars|1935.0900000000001|   4|
|  Singapore|    Classic Cars| 91791.76000000001|   1|
|  Singapore|Trucks and Buses|          75797.19|   2|
|  Singapore|    Vintage Cars|           30221.0|   3|
|  Singapore|           Ships|          13065.74|   4|
|  Singapore|          Trains|          12934.21|   5|
|  Singapore|     Motorcycles|            4175.6|   6|
|    Germany|    Classic Cars|         113357.71|   1|
|    Germany|          Planes|          20786.78|   2|
|    Germany|    Vintage Cars|          18480.53|   3|
+-----------+----------------+------------------+----+
```

p) Calculate a running total of SALES for each customer and show the top 5 customers by this cumulative total?

```
In [105]: customer_sales = sales_df.groupBy("CUSTOMERNAME").sum("SALES")
          customer_sales.orderBy("sum(SALES)", ascending=False).show(5)
```

```
+--------------------+------------------+
|        CUSTOMERNAME|        sum(SALES)|
+--------------------+------------------+
|Euro Shopping Cha...| 912294.1099999998|
|Mini Gifts Distri...| 654858.0599999999|
|Australian Collec...|200995.40999999997|
|  Muscle Machine Inc|         197736.94|
|   La Rochelle Gifts|180124.90000000002|
+--------------------+------------------+
only showing top 5 rows
```

q) Find and handle Invalid and Outliers values in entire DataFrame. [Check for only continuous dataset].

```
In [108]: sales_df.select("SALES", "QUANTITYORDERED", "PRICEEACH").describe().show()
          clean_df = sales_df.filter((sales_df["SALES"] < 20000) & (sales_df["SALES"] > 0) & (sales_df["QUANTITYORDERED"] > 0) &
              (sales_df["PRICEEACH"] > 0) & (sales_df["PRICEEACH"] < 500))
          clean_df.select("CUSTOMERNAME", "SALES").show()
```

```
+-------+-----------------+-----------------+-----------------+
|summary|            SALES| QUANTITYORDERED|        PRICEEACH|
+-------+-----------------+-----------------+-----------------+
|  count|             2823|             2823|             2823|
|   mean|3553.8890719093197|35.09280906836698| 83.65854410201914|
| stddev|1841.8651057401814|9.741442737069589|20.174276527840558|
|    min|           482.13|                6|            26.88|
|    max|          14082.8|               97|            100.0|
+-------+-----------------+-----------------+-----------------+

+--------------------+------+
|        CUSTOMERNAME| SALES|
+--------------------+------+
|   Land of Toys Inc.|2428.48|
| Suominen Souveniers|5154.41|
|  Muscle Machine Inc| 1939.8|
|Online Mini Colle...|3184.02|
|Anna's Decoration...| 4998.0|
|    Handji Gifts& Co| 5875.2|
|    Super Scale Inc.|2694.15|
|Salzburg Collecta...| 2443.6|
|Euro Shopping Cha...|3035.88|
|Technics Stores Inc.| 4829.8|
|Marta's Replicas Co.|4816.08|
|Anna's Decoration...| 4910.4|
|The Sharp Gifts W...|7048.14|
|   Land of Toys Inc.| 3053.7|
|Blauer See Auto, Co.|7916.31|
```

r) How would you cache a DataFrame containing sales data from the top 10 countries by sales to avoid recomputation in subsequent transformations? What persistence level (e.g. MEMORY_ONLY, MEMORY_AND_DISK) would you choose and why?

It would be better to use MEMORY_AND_DISK because if we use this type then when enough memory space is there it will be stored in RAM else it will be stored into the DISK. Hence the data remains safe and saved.

```
In [113]: top_countries = sales_df.groupBy("COUNTRY").sum("SALES")
          top10 = top_countries.orderBy("sum(SALES)", ascending=False).limit(10)
          top10.show()
          top10.cache()
```

```
+---------+------------------+
|  COUNTRY|        sum(SALES)|
+---------+------------------+
|      USA|        3627982.83|
|    Spain|1215686.9199999995|
|   France|1110916.5199999998|
|Australia| 630623.1000000001|
|       UK| 478880.4600000001|
|    Italy|374674.31000000006|
|  Finland|329581.91000000003|
|   Norway| 307463.7000000007|
|Singapore|288488.41000000003|
|  Denmark|245637.14999999997|
+---------+------------------+
```

```
Out[113]: DataFrame[COUNTRY: string, sum(SALES): double]
```

```
In [111]: from pyspark import StorageLevel

          top_sales_df.persist(StorageLevel.MEMORY_AND_DISK)
```

```
Out[111]: DataFrame[COUNTRY: string, ORDERNUMBER: int, QUANTITYORDERED: int, PRICEEACH: double, ORDERLINENUMBER: int, SALES: d
          ouble, ORDERDATE: string, STATUS: string, QTR_ID: int, MONTH_ID: int, YEAR_ID: int, PRODUCTLINE: string, MSRP: int,
          PRODUCTCODE: string, CUSTOMERNAME: string, PHONE: string, ADDRESSLINE1: string, ADDRESSLINE2: string, CITY: string,
          STATE: string, POSTALCODE: string, TERRITORY: string, CONTACTLASTNAME: string, CONTACTFIRSTNAME: string, DEALSIZE: s
          tring, Year: int, SalesCategory: string, sum(SALES): double]
```

s) How would you pivot the data to show PRODUCTLINE as columns and the total SALES for each ORDERDATE as the values? What are the implications of pivoting large datasets in Spark?

```
In [114]: pivot_df = sales_df.groupBy("ORDERDATE").pivot("PRODUCTLINE").sum("SALES")

          pivot_df.show()
```

```
+--------------+------------+------------------+-------+-----------------+-------+-----------------+--------------------+
|     ORDERDATE|Classic Cars|       Motorcycles| Planes|            Ships| Trains|  Trucks and Buses|      Vintage Cars|
+--------------+------------+------------------+-------+-----------------+-------+-----------------+--------------------+
|3/29/2004 0:00|        null|              null|   null|4933.719999999999|   null|             null|              378.4|
|5/30/2005 0:00|        null|              null|   null|             null|   null|             null|             1457.75|
|3/19/2004 0:00|     15330.7|              null|   null|             null|   null|             null|                null|
| 9/7/2004 0:00|        null|              null|   null|             null|   null|             null| 7673.379999999999|
| 5/4/2004 0:00|    15340.86|              null|   null|             null|   null|          18938.3|              269.15|
|11/9/2004 0:00|        null|              null|   null|          6673.29|3807.68|             null|              966.35|
|11/4/2003 0:00|    37852.99|          18877.11|   null|             null|   null|             null|                null|
| 7/1/2003 0:00|        null|25624.880000000005|   null|             null|   null|             null|                null|
|12/1/2003 0:00|        null|          25431.88|7120.96|             null|   null|             null|              113.6|
| 7/2/2004 0:00|    12334.82|              null|   null|             null|   null|             null|                null|
|1/29/2003 0:00|     15263.7|              null|   null|             null|   null|          23041.1|16397.199999999997|
+--------------+------------+------------------+-------+-----------------+-------+-----------------+--------------------+
```

t) How would you calculate the percentage growth of total sales month over month for each PRODUCTLINE using Spark DataFrame?

```
In [115]: from pyspark.sql.functions import lag,round

          monthly = sales_df.groupBy("PRODUCTLINE", "MONTH_ID").sum("SALES")
          win = Window.partitionBy("PRODUCTLINE").orderBy("MONTH_ID")
          monthly = monthly.withColumn("Prev", lag("sum(SALES)").over(win))
          monthly = monthly.withColumn("Growth", round((col("sum(SALES)") - col("Prev")) / col("Prev") * 100, 2))
          monthly.select("PRODUCTLINE", "MONTH_ID", "sum(SALES)", "Prev", "Growth").show()
```

```
+-----------+--------+------------------+------------------+------------+
| PRODUCTLINE|MONTH_ID|       sum(SALES)|          PrevSales|GrowthPercent|
+-----------+--------+------------------+------------------+------------+
| Motorcycles|       1|          81113.88|              null|        null|
| Motorcycles|       2|         122801.68|          81113.88|       51.39|
| Motorcycles|       3|          60469.98|         122801.68|      -50.76|
| Motorcycles|       4|119606.87999999999|          60469.98|        97.8|
| Motorcycles|       5|         108335.97|119606.87999999999|       -9.42|
| Motorcycles|       6|          49879.42|         108335.97|      -53.96|
| Motorcycles|       7| 60698.23000000001|          49879.42|       21.69|
| Motorcycles|       8|         106869.84| 60698.23000000001|       76.07|
| Motorcycles|       9|          45626.63|         106869.84|      -57.31|
| Motorcycles|      10|103649.61000000002|          45626.63|      127.17|
| Motorcycles|      11|261057.36000000004|103649.61000000002|      151.87|
| Motorcycles|      12|          46278.86|261057.36000000004|      -82.27|
|Vintage Cars|       1|         196129.58|              null|        null|
|Vintage Cars|       2|127119.78999999998|         196129.58|      -35.19|
|Vintage Cars|       3|177935.26999999996|127119.78999999998|       39.97|
|Vintage Cars|       4|         115941.63|177935.26999999996|      -34.84|
|Vintage Cars|       5|          146877.0|         115941.63|       26.68|
|Vintage Cars|       6|          74568.35|          146877.0|      -49.23|
|Vintage Cars|       7| 70364.57999999999|          74568.35|       -5.64|
|Vintage Cars|       8|         101424.51| 70364.57999999999|       44.14|
+-----------+--------+------------------+------------------+------------+
only showing top 20 rows
```

u) How can you rebalance the data by portioning based on the COUNTRY column to ensure that large data partitions are avoided?

```
In [120]: balanced_df = sales_df.repartition("COUNTRY")
          balanced_df.toPandas()
```

Out[120]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	CITY
0	10167	33	100.00	16	3812.16	10/23/2003 0:00	Cancelled	4	10	2003	...	Boras
1	10291	28	100.00	6	3256.96	9/8/2004 0:00	Shipped	3	9	2004	...	Boras
2	10167	24	100.00	13	2812.80	10/23/2003 0:00	Cancelled	4	10	2003	...	Boras
3	10320	26	61.23	2	1591.98	11/3/2004 0:00	Shipped	4	11	2004	...	Lule
4	10167	29	100.00	5	2940.02	10/23/2003 0:00	Cancelled	4	10	2003	...	Boras
...	...	...	...	...	...	...	...	...	...	...	...	...
2818	10164	39	81.93	4	3195.27	10/21/2003 0:00	Resolved	4	10	2003	...	Graz
2819	10341	41	100.00	9	7737.93	11/24/2004 0:00	Shipped	4	11	2004	...	Salzburg
2820	10419	34	100.00	4	4594.76	5/17/2005 0:00	Shipped	2	5	2005	...	Salzburg
2821	10164	21	100.00	2	3536.82	10/21/2003 0:00	Resolved	4	10	2003	...	Graz
2822	10419	43	100.00	3	5589.14	5/17/2005 0:00	Shipped	2	5	2005	...	Salzburg

2823 rows × 27 columns

```
In [121]: balanced_df.rdd.getNumPartitions()
Out[121]: 200
```

v) Suppose you have a smaller lookup table with customer details. How would you perform a broadcast join with the large sales_data_sample dataset to improve join performance? What are the key considerations when using broadcast joins?

```
In [126]: from pyspark.sql.functions import broadcast
          joined_df = sales_df.join(broadcast(customer_sales), on="CUSTOMERNAME", how="inner")
          joined_df.toPandas()
```

Out[126]:

	CUSTOMERNAME	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	...	S
0	Land of Toys Inc.	10248	32	75.89	4	2428.48	5/7/2004 0:00	Cancelled	2	5	...	
1	Suominen Souveniers	10363	43	100.00	9	5154.41	1/6/2005 0:00	Shipped	1	1	...	
2	Muscle Machine Inc	10127	20	96.99	7	1939.80	6/3/2003 0:00	Shipped	2	6	...	
3	Online Mini Collectables	10276	38	83.79	4	3184.02	8/2/2004 0:00	Shipped	3	8	...	
4	Anna's Decorations, Ltd	10391	42	100.00	3	4998.00	3/9/2005 0:00	Shipped	1	3	...	
...	...	...	...	...	...	...	...	...	...	...	...	
2818	Euro Shopping Channel	10380	34	100.00	3	3441.82	2/16/2005 0:00	Shipped	1	2	...	
2819	Euro Shopping Channel	10153	29	88.74	9	2573.46	9/28/2003 0:00	Shipped	3	9	...	
2820	Handji Gifts& Co	10288	34	68.49	10	2328.66	9/1/2004 0:00	Shipped	3	9	...	
2821	AV Stores, Co.	10306	29	100.00	7	3207.40	10/14/2004 0:00	Shipped	4	10	...	
2822	Euro Shopping Channel	10246	29	100.00	10	3520.60	5/5/2004 0:00	Shipped	2	5	...	

w) Create a UDF that categorizes the sales values (SALES) into custom buckets like “Low”, “Medium”, “High”. Apply this UDF to the DataFrame and calculate the count of orders in each category per COUNTRY.

```
In [128]: from pyspark.sql.functions import when, col
          sales_df = sales_df.withColumn(
              "SalesCategory",
              when(col("SALES") >= 10000, "High").when(col("SALES") >= 5000, "Medium").otherwise("Low")
          )
          sales_df.select("SALES", "SalesCategory").count()
```

Out[128]: 2823

x) Create a Python UDF to calculate discounts for specific product lines. For example, give a 10% discount for Classic Cars and 5% for Motorcycles. Apply this UDF to derive new discounted sales values.

```
In [129]: sales_df = sales_df.withColumn(
              "DiscountedSales",
              when(col("PRODUCTLINE") == "Classic Cars", col("SALES") * 0.90)
              .when(col("PRODUCTLINE") == "Motorcycles", col("SALES") * 0.95)
              .otherwise(col("SALES"))
          )
          sales_df.select("PRODUCTLINE", "SALES", "DiscountedSales").show()
```

```
+----------------+-------+------------------+
|     PRODUCTLINE|  SALES|   DiscountedSales|
+----------------+-------+------------------+
|           Ships|2428.48|           2428.48|
|     Classic Cars|5154.41|          4638.969|
|     Classic Cars| 1939.8|           1745.82|
|     Classic Cars|3184.02|          2865.618|
|     Vintage Cars| 4998.0|            4998.0|
|     Vintage Cars| 5875.2|            5875.2|
|     Vintage Cars|2694.15|           2694.15|
|           Planes| 2443.6|            2443.6|
|     Vintage Cars|3035.88|           3035.88|
|Trucks and Buses| 4829.8|            4829.8|
|Trucks and Buses|4816.08|           4816.08|
|     Classic Cars| 4910.4|           4419.36|
|     Classic Cars|7048.14|          6343.326|
|     Vintage Cars| 3053.7|            3053.7|
|     Classic Cars|7016.31|          6314.679|
|     Classic Cars|3584.25|3225.8250000000003|
|     Classic Cars|4753.49|          4278.141|
|     Vintage Cars|5399.55|           5399.55|
|      Motorcycles|1695.96|          1611.162|
```

z) How do you implement a cumulative distribution function (CDF) over the SALES value for each CUSTOMERNAME? What insights can you gather from analyzing the CDF distribution for each customer?

In [130]:
```
from pyspark.sql.functions import cume_dist
win = Window.partitionBy("CUSTOMERNAME").orderBy("SALES")
cdf_df = sales_df.withColumn("CDF", cume_dist().over(win))
cdf_df.select("CUSTOMERNAME", "SALES", "CDF").show()
```

```
+------------------+-------+------------------+
|      CUSTOMERNAME|  SALES|               CDF|
+------------------+-------+------------------+
|Suominen Souveniers| 891.03|0.03333333333333333|
|Suominen Souveniers| 1086.6|0.06666666666666667|
|Suominen Souveniers|1103.76|               0.1|
|Suominen Souveniers|1629.04|0.13333333333333333|
|Suominen Souveniers| 1988.4|0.16666666666666666|
|Suominen Souveniers|2140.11|               0.2|
|Suominen Souveniers|2447.76|0.23333333333333334|
|Suominen Souveniers|2632.89|0.26666666666666666|
|Suominen Souveniers| 2773.8|               0.3|
|Suominen Souveniers|2775.08| 0.3333333333333333|
|Suominen Souveniers|2817.87|0.36666666666666664|
|Suominen Souveniers|2851.84|               0.4|
|Suominen Souveniers|2931.98|0.43333333333333335|
|Suominen Souveniers|3128.65| 0.4666666666666667|
|Suominen Souveniers|3288.82|               0.5|
|Suominen Souveniers|3595.62| 0.5333333333333333|
|Suominen Souveniers|3686.54| 0.5666666666666667|
|Suominen Souveniers| 3784.8|               0.6|
|Suominen Souveniers| 4068.7| 0.6333333333333333|
|Suominen Souveniers|4142.64| 0.6666666666666666|
+------------------+-------+------------------+
only showing top 20 rows
```