

Big Data Processing Lab-4

Name: Sumit Vishwakarma

Roll No: 202201320

Colab Link:

<https://colab.research.google.com/drive/15CLflprF4Cubc9vsUGJi95E8dzBFDCwCscrollTo=ZIKlh41VQEqr>

Q1: Compute top-10 selling products in terms of numbers

Code and Output:

Q1: Compute top-10 selling products in terms of numbers

```
from pyspark.sql.functions import col, sum
orders_df = spark.read \
    .option("header", "true") \
    .csv( input )
qty_df = orders_df.groupBy("ProductID").agg(sum("OrderQuantity").alias("TotalQuantity"))
top_10_selling_qty = qty_df.orderBy(col("TotalQuantity").desc()).limit(10)
top_10_selling_qty.show()
```

```
↔ +-----+-----+
|ProductID|TotalQuantity|
+-----+-----+
|      23|      956.0|
|      37|      896.0|
|       8|      879.0|
|       4|      878.0|
|      40|      855.0|
|      41|      854.0|
|      22|      837.0|
|      38|      832.0|
|      27|      830.0|
|      12|      827.0|
+-----+-----+
```

Q2: Compute top-10 selling products in terms of value

Code and Output:

```
+ Code + Text

Q2: Compute top-10 selling products in terms of value

from pyspark.sql.functions import expr
input = "/content/gdrive/My Drive/us-sales/orders.csv"
orders_df = spark.read \
    .option("header", "true") \
    .csv( input )
value_df = orders_df.withColumn("value", col("UnitPrice") * col("OrderQuantity"))
value_df = value_df.groupBy("ProductID").agg(sum("value").alias("total_value"))
top_10_selling_value = value_df.orderBy(col("total_value").desc()).show(10)

+-----+-----+
|ProductID|    total_value|
+-----+-----+
|      23| 2358788.599999999|
|      40|    2130841.2|
|       4| 2071546.1999999993|
|      37|    2052886.7|
|      41|    2049958.8|
|       5| 2011333.3000000012|
|       2|    2005638.3|
|      35| 1981973.900000001|
|       8| 1976895.2999999996|
|      17| 1925111.000000002|
+-----+-----+
only showing top 10 rows

✓ 1s    completed at 10:23 PM
```

Q3: Compute top-10 profit making products. Profit = sum(qty*(price-cost))

Code and Output:

```
Q3: Compute top-10 profit making products. Profit = sum(qty*(price-cost))

orders_df = spark.read \
    .option("header", "true") \
    .csv( input )
profit_df = orders_df.withColumn("profit", col("OrderQuantity") * (col("UnitPrice") - col("UnitCost")))
profit_df = profit_df.groupBy("ProductID").agg(sum("profit").alias("total_profit"))
top_10_profit = profit_df.orderBy(col("total_profit").desc()).limit(10)
top_10_profit.show()

+-----+-----+
|ProductID|    total_profit|
+-----+-----+
|      23| 908818.7699999997|
|       8| 796037.5299999998|
|       4| 786277.2900000003|
|       2| 783599.7399999995|
|      40| 767278.9100000005|
|      41| 761318.8800000004|
|       5| 750981.4400000005|
|      37| 743189.7299999997|
|      35| 741447.8800000004|
|      11| 741098.0599999997|
+-----+-----+

54

✓ 1s    completed at 10:24 PM
```

Q4: Give top-3 stores selling product number 25

Code and Output:

Q4: Give top-3 stores selling product product number 25

```
from pyspark.sql.functions import col, sum
orders_df = spark.read \
    .option("header", "true") \
    .csv( input )
product_25_df = orders_df.filter(col("ProductID") == 25)
store_sales_df = product_25_df.groupBy("StoreID").agg(sum("OrderQuantity").alias("total_qty_sold"))
top_3_stores = store_sales_df.orderBy(col("total_qty_sold").desc()).limit(3)
top_3_stores.show()
```

```
+-----+-----+
|StoreID|total_qty_sold|
+-----+-----+
|      56|           16.0|
|      26|           14.0|
|     350|           13.0|
+-----+-----+
```

Q5: Give top-3 products sold in mid-west region

Code and Output:

Q5: Give top-3 products sold in midwest region

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

spark = SparkSession.builder \
    .appName("Top Products in Midwest") \
    .getOrCreate()

sales_df = spark.read.csv("/content/gdrive/My Drive/us-sales/sales_teams.csv", header=True, inferSchema=True)
orders_df = spark.read.csv("/content/gdrive/My Drive/us-sales/orders.csv", header=True, inferSchema=True)

sales_orders_df = sales_df.join(orders_df, on="SalesTeamID", how="inner")
midwest_df = sales_orders_df.filter(col("Region") == "Midwest")
product_sales_midwest_df = midwest_df.groupBy("ProductID").agg(sum("OrderQuantity").alias("total_qty_sold"))
top_3_products_midwest = product_sales_midwest_df.orderBy(col("total_qty_sold").desc()).limit(3)
top_3_products_midwest.show()
```

```
+-----+-----+
|ProductID|total_qty_sold|
+-----+-----+
|        5|           285|
|       40|           279|
|       27|           276|
+-----+-----+
```

Q6: Give region wise quantity sold for each product. Compute: Region, Product ID, Sum(Qty). Region is related to a order through sales team.

Code and Output:

```
Q6: Give region wise quantity sold for each product. Compute: Region, Product ID, Sum(Qty). Region is related to a order through sale ↑ ↓
```

```

from pyspark.sql import SparkSession
from pyspark.sql.functions import col, sum

spark = SparkSession.builder \
    .appName("Region Wise Quantity Sold") \
    .getOrCreate()
sales_df = spark.read.csv("/content/gdrive/My Drive/us-sales/sales_teams.csv", header=True, inferSchema=True)
orders_df = spark.read.csv("/content/gdrive/My Drive/us-sales/orders.csv", header=True, inferSchema=True)

sales_orders_df = sales_df.join(orders_df, on="SalesTeamID", how="inner")
region_product_sales_df = sales_orders_df.groupBy("ProductID", "Region").agg(sum("OrderQuantity").alias("total_qty_sold"))
region_product_sales_df.show()

```

| ProductID | Region | total_qty_sold |
|-----------|-----------|----------------|
| 28 | South | 210 |
| 4 | Midwest | 244 |
| 22 | Northeast | 209 |
| 6 | South | 71 |
| 19 | West | 229 |
| 16 | Midwest | 241 |
| 5 | South | 160 |
| 46 | West | 197 |

✓ 1s completed at 10:23 PM

Q-7: Compute Average monthly sale in terms of numbers at each store; that , that is on average what numbers of a product are sold on a store in a month.

Code and Output:

```
Q-7: Compute Average monthly sale in terms of numbers at each store; that , that is on average what numbers of a product are sold on a store in a month.
```

```

[ ] from pyspark.sql.functions import month, year, date_format, avg
df = df.withColumn("YearMonth", date_format("OrderDate", "yyyy-MM"))
monthly_sales_df = df.groupBy("StoreID", "YearMonth").agg(sum("OrderQuantity").alias("MonthlyQuantity"))
average_monthly_sales_df = monthly_sales_df.groupBy("StoreID").agg(avg("MonthlyQuantity").alias("AverageMonthlyQuantity"))

average_monthly_sales_df.show()

```

| StoreID | AverageMonthlyQuantity |
|---------|------------------------|
| 148 | 6.333333333333333 |
| 243 | 5.428571428571429 |
| 31 | 5.5 |
| 85 | 7.470588235294118 |
| 251 | 6.533333333333333 |
| 137 | 5.6 |
| 65 | 6.6875 |
| 255 | 6.0588235294117645 |
| 53 | 2.923076923076923 |
| 133 | 5.357142857142857 |
| 296 | 5.176470588235294 |
| 322 | 5.894736842105263 |
| 78 | 5.923076923076923 |

✓ 1s completed at 10:23 PM

Q-8: Compute sales bifurcation of each warehouse; that total sales amount through each channel

Code and Output:

```
Q-8: Compute sales bifurcation of each warehouse; that total sales amount through each channel
```

```
sales_amount_df = df.withColumn("SalesAmount", col("OrderQuantity") * col("UnitPrice"))
warehouse_sales_df = sales_amount_df.groupBy("WarehouseCode", "Sales_Channel").agg(sum("SalesAmount").alias("TotalSalesAmount"))
warehouse_sales_df.show()
```

| WarehouseCode | Sales_Channel | TotalSalesAmount |
|---------------|---------------|----------------------|
| WARE-PUJ1005 | Distributor | 2998256.6999999993 |
| WARE-UHY1004 | Distributor | 2269785.8 |
| WARE-NBV1002 | In-Store | 2856270.3000000003 |
| WARE-MKL1006 | Wholesale | 837071.2 |
| WARE-PUJ1005 | Wholesale | 1713940.4000000004 |
| WARE-NMK1003 | Wholesale | 3155331.4999999998 |
| WARE-MKL1006 | In-Store | 3554832.4000000004 |
| WARE-UHY1004 | In-Store | 5670873.299999997 |
| WARE-NBV1002 | Distributor | 1248377.5000000002 |
| WARE-NMK1003 | Distributor | 4455593.800000001 |
| WARE-XYS1001 | Online | 3827207.5000000005 |
| WARE-XYS1001 | Distributor | 2099110.0000000005 |
| WARE-XYS1001 | In-Store | 5346941.6999999996 |
| WARE-UHY1004 | Wholesale | 1501536.9999999998 |
| WARE-NMK1003 | In-Store | 1.0728823899999984E7 |
| WARE-MKL1006 | Distributor | 1738784.0000000002 |

✓ 1s completed at 10:23 PM

Q9: Compute average "product retention period" (i. e. the difference between procurement date and order date) at each warehouse

Code and Output:

```
Q9: Compute average "product retention period" (i. e. the difference between procurement date and order date) at each warehouse
```

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import avg, to_date, datediff
orders_df = spark.read \
    .option("header", "true") \
    .csv(input)
orders_df = orders_df.withColumn("OrderDate", to_date(orders_df["OrderDate"], "yyyy-MM-dd")).withColumn("ProcuredDate", to_date(orders_df["ProcuredDate"], "yyyy-MM-dd"))
orders_with_retention = orders_df.withColumn("retention_period", datediff(orders_df["OrderDate"], orders_df["ProcuredDate"]))
avg_retention_by_warehouse = orders_with_retention.groupBy("WarehouseCode").agg(avg("retention_period").alias("avg_retention_period"))
avg_retention_by_warehouse.show()
```

| WarehouseCode | avg_retention_period |
|---------------|----------------------|
| WARE-XYS1001 | 109.35679214402619 |
| WARE-PUJ1005 | 109.51274982770504 |
| WARE-MKL1006 | 108.2532088681447 |
| WARE-NMK1003 | 109.46986027944112 |
| WARE-UHY1004 | 108.73359683794466 |
| WARE-NBV1002 | 109.81476121562952 |

Q10: Give Year-Month sales of all products. Here you actually print 'Year-Month', ProductID, sum(qty).

Code and Output:

```
Q10: Give Year-Month sales of all products. Here you actually print 'Year-Month', ProductID, sum(qty). Use Order Date for extracting Year and Month of sale.
```

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, sum, date_format
orders_df = spark.read \
    .option("header", "true") \
    .csv( input )
sales_df = orders_df.withColumn("YearMonth", date_format(col("OrderDate"), "yyyy-MM")).groupBy("YearMonth", "ProductID").agg(sum("OrderQuantity").alias("sum_qty"))
sales_df.show()
```

| YearMonth | ProductID | sum_qty |
|-----------|-----------|---------|
| 2018-11 | 3 | 21.0 |
| 2018-12 | 44 | 12.0 |
| 2019-05 | 46 | 22.0 |
| 2019-05 | 45 | 16.0 |
| 2020-09 | 37 | 8.0 |
| 2020-09 | 9 | 15.0 |
| 2020-02 | 33 | 26.0 |
| 2020-03 | 29 | 5.0 |
| 2020-07 | 24 | 29.0 |
| 2018-08 | 44 | 12.0 |
| 2019-08 | 23 | 30.0 |

✓ 1s completed at 10:23 PM

Q11: Compute a fact file with the dimensions of "store_id", "product_id", and "month_year".

Code and output:

```
Q11: Compute a fact file with the dimensions of "store_id", "product_id", and "month_year".`
```

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, sum, date_format
orders_df = spark.read \
    .option("header", "true") \
    .csv( input )
orders_df = orders_df.withColumn("month_year", date_format(col("OrderDate"), "yyyy-MM"))
fact_file_df = orders_df.groupBy("StoreID", "ProductID", "month_year").agg(sum("OrderQuantity").alias("total_quantity"), sum("OrderAmount").alias("total_amount"))
fact_file_df.show()
```

| StoreID | ProductID | month_year | total_quantity | total_amount |
|---------|-----------|------------|----------------|--------------|
| 306 | 11 | 2018-06 | 8.0 | 857.6 |
| 311 | 46 | 2018-06 | 1.0 | 1829.1 |
| 297 | 9 | 2018-06 | 2.0 | 207.7 |
| 90 | 21 | 2018-08 | 1.0 | 1098.8 |
| 350 | 30 | 2018-08 | 5.0 | 1118.9 |
| 329 | 29 | 2018-09 | 1.0 | 2525.9 |
| 281 | 32 | 2018-11 | 3.0 | 5862.5 |
| 116 | 36 | 2018-11 | 6.0 | 2438.8 |
| 124 | 29 | 2018-12 | 2.0 | 1098.8 |
| 256 | 32 | 2018-12 | 2.0 | 3825.7 |
| 99 | 21 | 2019-01 | 6.0 | 5701.7 |
| 362 | 44 | 2019-01 | 2.0 | 201.0 |
| 145 | 37 | 2019-01 | 5.0 | 2653.2 |

✓ 1s completed at 10:23 PM