Code Logic - Retail Data Analysis

Code Logic

A step by step explanation of flow involved in reading streaming data and achieving KPI's as asked.

Step 1:

Include all necessary libraries and establish a spark session

Step 2:

Connect to the kafka server where streaming data is available along with topic name that you want to read from.

Step 3:

Since now you read the data from server, we now make it more readable by mapping lines of streaming data into a schema, along with data types against each column.
Columns in raw data are "country", "invoice_no", "timestamp", "type", "items" which is an array which is sub-divided into "SKU", "title", "unit_price", "quantity" for each occurence

Step 4:

Unveil array items now as individual array items are used in KPI and new column derivations. Use option explode for this purpose.
df1 =
new_df.select(col("type"),col("country"),col("invoice_no"),col("timestamp"),explode(col("items ")))

Step 5:

Rename columns like SKU , title, unit price & quantity which were earlier represented as col.SKU or col.title.

Step 6:

Defining UDF's and executing them.
UDF 1: This function accepts "type" as input and determines if the nature of request is "Order" or "Return"
If Type comes as "ORDER", then it returns a flag is_order = 1 which says it is an order.
UDF 2: This function accepts "type" as input and determines if the nature of request is "Order" or "Return"
If Type comes as "RETURN", then it returns a flag is_return = 1 which says it is a return request.

UDF 3: This function accepts 3 attributes in input. Unit price, quantity & type as input. If type is "Order", it returns a value of Unit price multiplied by quantity. If type is "Return", it returns a value of Unit price multiplied by quantity toggled by "-" symbol.

Step 7:
Since we calculated all new columns required, we need to add them to our existing data frame.
Output the data with new columns for each order for window of 1 minute.

```
df3 = df2.withWatermark("timestamp","10 minutes") \
.groupby(window("timestamp","1 minute"),"invoice_no","country","is_Order","is_Return") \
.sum("Cost","quantity")
```

Output from this step is used for calculating KPI's in next steps.

Step 8:
Calculation of KPI's:
Orders per Minute:
Calculated as count of distinct invoices as below >

```
F.approx_count_distinct("invoice_no").alias("OPM")
```

Total volume of sales:
Calculated from UDF3 where we derived cost value of order. Use this cost value and get a sum of all order cost's for window period to get total volume of sales >

```
agg(sum("Cost").alias("Total_sales_vol")
```

Rate of Return:
For this KPI, we utilize is_return & is_order flags derived from UDF1 & UDF2.
We take sum of is_return & is_order for window period and store them as total_Order & total_return.
Below formula is used to calculate Rate of Return. Note that same formula is used twice once after grouping by country to get country specific rate of return.
Second time to generate time based rate of return.

Final_time =
Final_time.withColumn("rate_of_return",Final_time.total_return/(Final_time.total_Order+Final_time.total_return))
Average Transaction Size:
For this KPI, we utilize is_return & is_order flags derived from UDF1 & UDF2.
We take sum of is_return & is_order for window period and store them as total_Order & total_return.
Below formula is used to calculate average transaction size. This needs to be calculated only on time basis, not country basis.
Final_time.withColumn("Avg_trans_size",Final_time.Total_sales_vol/(Final_time.total_Order+Final_time.total_return))

Step 9:
Now that KPI calculation is done, there are three 3 ways to let out our data.
1) Printing all input and derived columns to console which can be collected in **Console-output** using below code. This can be achieved by submitting the location as below in spark submit job using jar file.

spark2-submit --jars spark-sql-kafka-0-10_2.11-2.3.0.jar put_jsont.py > **console_print**
Code:

```
query_1 = df3 \
.writeStream \
.outputMode("complete") \
.format("console") \
.option("truncate", "False") \
.start()
```

2) Print time based KPI to HDFS path as a JSON file using below code.

```
query_2 = Final_time.writeStream \
.outputMode("Append") \
.format("json") \
.option("format","append") \
.option("truncate", "false") \
.option("path","time_KPI") \
.option("checkpointLocation", "time_KPI_json") \
.trigger(processingTime="1 minute") \
.start()
```

Here, HDFS location path is given as time_KPI where KPI data shall be written to.

3) Print time and country based KPI to HDFS path as a JSON file using below code.

```
query_3 = Final_country_time.writeStream \
.outputMode("Append") \
.format("json") \
.option("format","append") \
.option("truncate", "false") \
.option("path","time_country_KPI") \
.option("checkpointLocation", "time_country_KPI_json") \
.trigger(processingTime="1 minute") \
.start()
```

Here, HDFS location path is given as time_country_KPI where KPI data shall be written to.

Step 10:
Once code is run, execute below command to create a jar file that acts as instance to run our spark code.
wget https://ds-spark-sql-kafka-jar.s3.amazonaws.com/spark-sql-kafka-0-10_2.11-2.3.0.jar
export spark version using command - export SPARK_KAFKA_VERSION=0.10
Submit the spark job using command below , tag the object created above to the python file that has program to execute and all console data shall be written to console_print
spark2-submit --jars spark-sql-kafka-0-10_2.11-2.3.0.jar read_write.py > console_print

Step 11:
As the program goes towards completion, open Hadoop file system and list out files written from program.
Files written here are at paths “time_country_KPI” & “time_KPI”.
These paths contain all KPI related data identified from streaming data.
Copy these files to local and zip them attach to output file for submission.