Code Logic - Retail Data Analysis

In this document, you will describe the code and the overall steps taken to solve the project.

#### Calculation Logic

1. **OPM**: agg(count(df5['invoice\_no']).alias('OPM')) – Aggregated total records in 1 minute window
2. **Rate\_of\_return**: agg((sum(df5['is\_return'])/(sum(df5['is\_return']+df5['is\_order']))).alias('rate\_of\_return')) – Aggregated return orders and then divide them by total orders
3. **Total\_sale\_volume**: agg(sum(df5['total\_cost']).alias('total\_sale\_volume')) – Aggregated total cost of all orders
4. **Average\_transaction\_size**: agg(sum(df5['total\_cost'])/(sum(df5['is\_return']+df5['is\_order']))).alias('average\_transaction\_size')) – Aggregated total cost and then divided them by total orders

#### Run Steps

**shell commands to run the program:**

1. export SPARK\_KAFKA\_VERSION=0.10 [Sparka Kafka 10 version is supported]
2. spark2-submit --packages org.apache.spark:spark-sql-kafka-0-10\_2.11:2.3.0 spark-streaming.py 2>&1 | tee console\_output.txt

#### Code Walkthrough

**STEP 1: Defining the json schema**

The records that are read from kafka topic are defined a schema, since we want to parse the incoming JSON messages and doing this would simplify flattening json schema to fit in to the data frame

schema = StructType([

StructField("type", StringType()),

StructField("country", StringType()),

StructField("invoice\_no", LongType()),

StructField("timestamp", TimestampType()),

StructField("items", ArrayType(

StructType([

StructField("SKU", StringType()),

StructField("title", StringType()),

StructField("unit\_price", DoubleType()),

StructField("quantity", IntegerType())

])

))

])

**STEP 2: Read from Kafka topic**

Read from kafka topic from the bootstrap server given in the assignment and parse json to the already defined schema

parsed = spark \

.readStream \

.format("kafka") \

.option("kafka.bootstrap.servers", "18.211.252.152:9092") \

.option("subscribe", "real-time-project") \

.option("startingOffsets","earliest") \

.load() \

.select(from\_json(col("value").cast("string"), schema).alias("parsed\_value"))

**STEP 3: The full json is nested in one single column, so we will spread that out using the below command**

parsed\_1=parsed.select(col("parsed\_value.\*"))

**STEP 4: Writing UDFs required to calculate total cost and total items**

@udf('float')

def getTotalCost(x,type):

sum = 0

for val in x:

sum += val.quantity\*val.unit\_price

if type == 'ORDER':

return sum

return -1\*sum

@udf('int')

def getTotalItems(x):

sum = 0

for val in x:

sum += val.quantity

return sum

**STEP 5: Creating the new columns total\_cost and total\_items using the UDFs defined above**

df4 = parsed\_1.withColumn('total\_cost',getTotalCost(parsed\_1['items'],parsed\_1['type'])).withColumn('total\_items',getTotalItems(parsed\_1['items']))

**STEP 6: Writing UDFs required to classify order or return**

@udf("int")

def isOrder(type):

if type == 'ORDER':

isOrder = 1

else:

isOrder = 0

return isOrder

@udf("int")

def isReturn(type):

if type == 'RETURN':

isReturn = 1

else:

isReturn = 0

return isReturn

**STEP 7: Calculating input data frame with all the final summarized input values**

df5 = df4.withColumn('is\_order',isOrder(df4['type'])).withColumn('is\_return',isReturn(df4['type'])).select('invoice\_no','country','timestamp','total\_cost','total\_items','is\_order','is\_return')

**STEP 8: Time based kpi aggregation to calculate OPM, rate\_of\_return, total\_sale\_volume, average\_transaction\_size**

df6 = df5.withWatermark("timestamp", "1 minute").groupBy(window('timestamp','1 minute')).agg(count(df5['invoice\_no']).alias('OPM'),(sum(df5['is\_return'])/(sum(df5['is\_return']+df5['is\_order']))).alias('rate\_of\_return'),sum(df5['total\_cost']).alias('total\_sale\_volume'),(sum(df5['total\_cost'])/(sum(df5['is\_return']+df5['is\_order']))).alias('average\_transaction\_size'))

**STEP 9: Time and country based kpi aggregation to calculate OPM, rate\_of\_return, total\_sale\_volume**

df7 = df5.withWatermark("timestamp", "1 minute").groupBy(window('timestamp','1 minute'),'country').agg(count(df5['invoice\_no']).alias('OPM'),(sum(df5['is\_return'])/(sum(df5['is\_return']+df5['is\_order']))).alias('rate\_of\_return'),sum(df5['total\_cost']).alias('total\_sale\_volume'))

**STEP 10: Writing to console for summarized input in 1 minute batches**

query\_in = df5.writeStream \

.outputMode("append") \

.format("console") \

.trigger(processingTime='1 minute') \

.start()

**STEP 11: Writing to json for time based kpi in 1 minute batches**

query\_timebased = df6.writeStream \

.outputMode("append") \

.format("json") \

.option("truncate", "false") \

.option("path", "user/root/time\_kpi") \

.option("checkpointLocation", "user/root/checkpoint2") \

.trigger(processingTime='1 minute') \

.start()

**STEP 12: Writing to json for time and country based kpi in 1 minute batches**

query\_timecountrybased = df7.writeStream \

.outputMode("append") \

.format("json") \

.option("truncate", "false") \

.option("path", "user/root/country\_kpi") \

.option("checkpointLocation", "user/root/checkpoint3") \

.trigger(processingTime='1 minute') \

.start()

**STEP 13: Wait for a 10 minute = 600 seconds interval for the process to finish up**

query\_in.awaitTermination(600)

query\_timebased.awaitTermination(600)

query\_timecountrybased.awaitTermination(600)