Spark Submit Code Explanation

Imported Libraries

Importing all required libraries for our project

```
# Import necessary modules
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark.sql.window import Window
```

Spark Session Creation

We start the spark application named RetailDataAnalyticsStreamingProject.

```
# SparkSession Creation
spark = SparkSession.builder.appName("RetailDataAnalyticsStreamingProject").getOrCreate()

spark.sparkContext.setLogLevel('ERROR')
```

Loading Data and Creating Schema

We load the raw stream data by connecting to kafka server and then create a schema to give it proper structure. Kafka raw stream data is deserialized from JSON into a Spark DataFrame

```
# Lets load data

# Reading input data from Kafka server with the given credentials
raw_stream_data = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers","18.211.252.152:9092") \
    .option("subscribe","real-time-project") \
    .option("startingOffsets", "latest")  \
    .load()


# Define File Schema
JSON_Schema = StructType() \
    .add("invoice_no", LongType()) \
    .add("country",StringType()) \
    .add("timestamp", TimestampType()) \
    .add("type", StringType()) \
    .add("items", ArrayType(StructType([
        StructField("SKU", StringType()),
        StructField("title", StringType()),
        StructField("unit_price", FloatType()),
        StructField("quantity", IntegerType())
        ])))


order_stream_data = raw_stream_data.select(from_json(col("value").cast("string"), JSON_Schema).alias("data")).select("data.*")
```

Creating User defined functions

We create the User defined functions in accordance with the formulas given.

order_checker function confirms if the order is indeed an order and not a return one. We give 1 to orders and 0 to others.

return_checker function confirms if the order is indeed an. We give 1 to orders and 0 to others.

net_item_count function adds up the count of all items in an order

net_cost function calculates the net cost of an order by doing quantity of item*price of item calculation (positive if normal , negative if return)

```
# Creating Utility Functions for further steps

# Check whether new order
def order_checker(type):
    return 1 if type == 'ORDER' else 0

# Checking whether return order
def return_checker(type):
    return 1 if type == 'RETURN' else 0

# Calculating Total Item Count in an order
def net_item_count(items):
    if items is not None:
        item_count =0
        for item in items:
            item_count = item_count + item['quantity']
        return item_count

# Calculating Total Cost of an order
def net_cost(items,type):
    if items is not None:
        total_cost =0
        item_price =0
    for item in items:
        item_price = (item['quantity']*item['unit_price'])
        total_cost = total_cost+ item_price
        item_price=0

    if type  == 'RETURN':
        return total_cost *-1
    else:
        return total_cost


# Lets create the user defined functions now
is_order = udf(order_checker, IntegerType())
is_return = udf(return_checker, IntegerType())
add_net_item_count = udf(net_item_count, IntegerType())
add_net_cost = udf(net_cost, FloatType())
```

Using UDFs to create new columns and get the final stream dataset to be used for analytics

```
# Using above UDFs to add new columns to our input stream data
processed_order_stream = order_stream_data \
    .withColumn("total_cost", add_net_cost(order_stream_data.items,order_stream_data.type)) \
    .withColumn("total_items", add_net_item_count(order_stream_data.items)) \
    .withColumn("is_order", is_order(order_stream_data.type)) \
    .withColumn("is_return", is_return(order_stream_data.type))
```

Writing Processes Input Table to console

```
# Writing the processed input table to the console
order_batch_console = processed_order_stream \
    .select("invoice_no", "country", "timestamp","total_cost","total_items","is_order","is_return") \
    .writeStream.outputMode("append").format("console").option("truncate", "false") \
    .option("path", "/Console_output").option("checkpointLocation", "/Console_output_checkpoints") \
    .trigger(processingTime="1 minute") \
    .start()
```

Time Based KPI

We first create the aggregated time stream level by clubbing data into 1 min groups and calculate the KPIs Order per Minute, Total Sales Volume, Average Transaction size and Rate of Return. We then write the output to HDFS.

```
# Calculating the time based KPIs
aggregated_time_stream = processed_order_stream \
    .withWatermark("timestamp","1 minutes").groupby(window("timestamp", "1 minute")) \
    .agg(count("invoice_no").alias("OPM"),sum("total_cost").alias("total_sale_volume"),avg("total_cost").alias("average_transaction_size"),avg("is_Return").alias("rate_of_return")) \
    .select("window.start","window.end","OPM","total_sale_volume","average_transaction_size","rate_of_return")
```

```
# Writing the time based KPI values to the Console
time_stream_console = aggregated_time_stream.writeStream \
    .format("json").outputMode("append") \
    .option("truncate", "false").option("path", "timeKPIvalue").option("checkpointLocation", "timeKPIvalue_checkpoints") \
    .trigger(processingTime="1 minutes") \
    .start()
```

Time Country Based KPI

We first create the aggregated time stream level by clubbing data into 1 min groups and by country and calculate the KPIs Order per Minute, Total Sales Volume and Rate of Return. We then write the output to HDFS.

```
# Calculating the time and country based KPIs
aggregated_time_country_stream = processed_order_stream \
    .withWatermark("timestamp", "1 minutes") \
    .groupBy(window("timestamp", "1 minutes"), "country") \
    .agg(count("invoice_no").alias("OPM"),sum("total_cost").alias("total_sale_volume"),avg("is_Return").alias("rate_of_return")) \
    .select("window.start","window.end","country", "OPM","total_sale_volume","rate_of_return")
```

```
# Writing the time and country based KPI values to the Console
time_country_stream_console = aggregated_time_country_stream.writeStream \
    .format("json").outputMode("append") \
    .option("truncate", "false").option("path", "time_countryKPIvalue").option("checkpointLocation", "time_countryKPIvalue_checkpoints") \
    .trigger(processingTime="1 minutes") \
    .start()
```

Await Termination

We started the spark job and stop it through keyboard interrupt after 10 minutes.

```
# Executing the spark jobs for console outputs and storage
order_batch_console.awaitTermination()
time_stream_console.awaitTermination()
time_country_stream_console.awaitTermination()
```

Spark submit Command

spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5 spark-streaming.py> Console-output.txt

Other Relevant Commands Used

SSH into Amazon EMR- ssh -i "PEM.pem" hadoop@ec2-52-3-245-82.compute-1.amazonaws.com

Copy spark submit notebook from s3 to Amazon EMR- aws s3 cp s3://aws-emr-studio-783482579529-us-east-1/Scriptfileretaildata/spark-streaming.py /home/hadoop/

See Console Output- cat Console-output.txt

Download KPI values from HDFS-

- hadoop fs -get timeKPIvalue
- hadoop fs -get time_countryKPIvalue

Download KPI files from EMR to local machine-

- **scp -i "E:\MS in Data Science\MS-DataScience\Login Files\PEM.pem" -r hadoop@107.21.57.234:/home/hadoop/timeKPIvalue "E:\timeKPIvalue"**
- **scp -i "E:\MS in Data Science\MS-DataScience\Login Files\PEM.pem" -r hadoop@107.21.57.234:/home/hadoop/time_countryKPIvalue "E:\time_countryKPIvalue"**