Code Logic - Retail Data Analysis

1. Data is read from Kafka using the defined schema, parsed as JSON, and stored in the orderStream DataFrame.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *

# Initial Spark session creation
spark = SparkSession  \
        .builder  \
        .appName("StructuredSocketRead")  \
        .getOrCreate()
spark.sparkContext.setLogLevel('ERROR')

# Reading the streaming data
orderRaw = spark  \
        .readStream  \
        .format("kafka")  \
        .option("kafka.bootstrap.servers","ec2-18-211-252-152.compute-1.amazonaws.com:9092")  \
        .option("subscribe","real-time-project")  \
        .load()
```

```
# Define schema of a single order
jsonSchema = StructType() \
      .add("invoice_no", StringType()) \
      .add("country", StringType()) \
      .add("timestamp", TimestampType()) \
      .add("type", StringType()) \
      .add("items", ArrayType(StructType([
      StructField("SKU", StringType()),
      StructField("title", StringType()),
      StructField("unit_price", DoubleType()),
      StructField("quantity", DoubleType())
  ])))

# Parsing the Streaming data using from_json and schema
orderStream = orderRaw.select(from_json(col("value").cast("string"), jsonSchema).alias("data")).select("data.*")
```

2. Created UDFs for calculating total_items, total_cost, is_order, and is_return.

```
# UDF for calculating Total_Items
def get_total_item_count(items):
    total_items = 0
    for item in items:
        total_items = total_items + item[2]
    return total_items

# UDF for calculating Total_cost
def get_total_cost_per_record(items):
    total_cost = 0
    for item in items:
        total_cost = total_cost + (item[2] * item[3])
    return total_cost

# UDF for calculating order type flag
def get_is_order_type(type):
    order_type_flag = 0
    if type == 'ORDER':
        order_type_flag = 1
    else:
        order_type_flag = 0
    return order_type_flag

# UDF for calculating return type flag
def get_is_order_return_type(type):
    order_return_type_flag = 0
    if type == 'ORDER':
        order_return_type_flag = 0
    else:
        order_return_type_flag = 1
    return order_return_type_flag
```

3. User-defined functions were defined with utility, and new columns (Total Cost, Total_Items, Is_order, and Is_return) were created using UDF. The intermediary datasets were written to the console.

```
# Define the UDFs with the utility functions
add_total_count = udf(get_total_item_count, DoubleType())
add_total_cost = udf(get_total_cost_per_record, DoubleType())
add_is_order_flg = udf(get_is_order_type, IntegerType())
add_is_return_flg = udf(get_is_order_return_type, IntegerType())

# Deriving the Required new attributes using the UDF
Data_Frame_Total_Items_Cost= orderStream \
    .withColumn("Total_Items", add_total_count(orderStream.items)) \
    .withColumn("Total_Cost", add_total_cost(orderStream.items)) \
    .withColumn("is_order", add_is_order_flg(orderStream.type)) \
    .withColumn("is_return", add_is_return_flg(orderStream.type)).select("invoice_no","country","timestamp","Total_Items", "Total_Cost","is_order

# Writing the Inetermediary data into Console
query = Data_Frame_Total_Items_Cost  \
        .writeStream  \
        .outputMode("append")  \
        .format("console")  \
        .option("truncate", "false") \
        .start()
```

4. Calculating time based and country based KPI's using with Watermark and groupBy

```
# Calculate time based KPIs
aggStreamByTime = Data_Frame_Total_Items_Cost \
    .withWatermark("timestamp", "1 minute") \
    .groupBy(window("timestamp", "1 minute", "1 minute")) \
    .agg(sum("Total_Cost").alias("total_volume_of_sales"),count("invoice_no").alias("OPM"),avg("is_return").alias("avg_rate_of_return")).select("


# Calculate Country based KPIs
aggStreamByCountry= Data_Frame_Total_Items_Cost \
    .withWatermark("timestamp", "1 minute") \
    .groupBy(window("timestamp", "1 minute", "1 minute"), "country") \
    .agg(sum("Total_Cost").alias("total_volume_of_sales"),count("invoice_no").alias("OPM"),avg("is_return").alias("avg_rate_of_return")).select("
```

5. Writing the Time based and country based KPI’s to HDFS

```
# Writing the Time Based KPIs into HDFS
queryByTime= aggStreamByTime.writeStream \
    .format("json") \
    .outputMode("append") \
    .option("truncate", "false") \
    .option("path", "time-wise-kp1") \
    .option("checkpointLocation", "time-cp1") \
    .trigger(processingTime="1 minute") \
    .start()


# Writing the Country Based KPIs into HDFS
queryByCountry = aggStreamByCountry.writeStream \
    .format("json") \
    .outputMode("append") \
    .option("truncate", "false") \
    .option("path", "time-country-wise-kp1") \
    .option("checkpointLocation", "time-country-cp1") \
    .trigger(processingTime="1 minute") \
    .start()

queryByCountry.awaitTermination()
```