Code Logic - Retail Data Analysis

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

Logic for 'spark-streaming.py'

First, we import the sparksession and other modules necessary for further use.

```
#importing necessary modules
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
```

Writing the Python functions, which contain the logic for the UDFs

1. Total Cost UDF - To calculate the total income from every invoice I needed to calculate the income from sale of each product, so I multiplied the unit price of the product with the quantity of the product purchased. The sum of this cost across the products in that invoice gives me the total cost of the order. I also made sure that if the transaction is a return transaction, the total cost is negative.

```
def find_total_order_cost(items, trn_type):
    if items is not None:
        total_cost = 0

        for item in items:
            item_price = item['quantity'] * item['unit_price']
            total_cost += item_price
```

```
        if trn_type == "RETURN":
            return total_cost * -1
        else:
            return total_cost
    else:
        return 0  # Return 0 if items is None
```

2. Total Items UDF - To calculate the number of products in every invoice I added the quantity ordered of each product in that invoice.

```
def find_total_item_count(items):
    if items is not None:
        total_count = 0
        for item in items:
            total_count = total_count + item['quantity']
        return total_count
```

3. Is Order UDF - To determine if invoice is for an order or not I used an if-else statement.

```
def flag_isOrder(trn_type):
    if trn_type == "ORDER":
        return(1)
    else:
        return(0)
```

4. Is Return UDF - To determine if invoice is for a return or not I used an if-else statement.

```
def flag_isReturn(trn_type):
    if trn_type == "RETURN":
        return(1)
    else:
        return(0)
```

Initializing the Spark session and setting the log level to warn as a good practice.

```
spark = SparkSession  \
    .builder  \
    .appName("spark-streaming")  \
    .getOrCreate()
spark.sparkContext.setLogLevel('WARN')
```

Reading input data from Kafka mentioning the details of the Kafka broker, such as bootstrap server, port and topic name.

```
rawData = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "18.211.252.152:9092") \
    .option("startingOffsets","earliest") \
    .option("failOnDataLoss", "false") \
    .option("subscribe", "real-time-project") \
    .load()
```

Defining JSON schema of each order, using appropriate datatypes and StructField in the case of the item attributes.

```
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()),
])))
```

Reading the raw JSON data from Kafka as 'order stream' by casting it to string and storing it into the alias 'data'.

```
orderStream = rawData.select(from_json(col("value").cast("string"),
schema).alias("data")).select("data.*")
```

Defining the UDFs by Converting the Python functions I defined earlier, and assigning the appropriate return datatype.

```
sum_total_order_cost = udf(find_total_order_cost, FloatType())
sum_total_item_count = udf(find_total_item_count, IntegerType())
sum_isOrder = udf(flag_isOrder, IntegerType())
sum_isReturn = udf (flag_isReturn, IntegerType())
```

Calculating the additional columns according to the required input values.

```
newOrderStream = orderStream \
    .withColumn("total_cost", sum_total_order_cost(orderStream.items, orderStream.type)) \
    .withColumn("total_items", sum_total_item_count(orderStream.items)) \
    .withColumn("is_order", sum_isOrder(orderStream.type)) \
    .withColumn("is_return", sum_isReturn(orderStream.type))
```

Writing the summarized input values to console, using 'append' output method and applying truncate as false and setting the processing time to 1 minute.

```
orderQuery = newOrderStream \
    .select("invoice_no", "country", "timestamp", "total_cost", "total_items", "is_order",
"is_return") \
    .writeStream \
    .outputMode("append") \
    .format("console") \
    .option("truncate", "false") \
    .trigger(processingTime = "1 minute") \
    .start()
```

Calculating time-based KPIs (Total sale volume, OPM, Rate of return, Average transaction size) having tumbling window of one minute and watermark of one minute.

```
aggStreamByTime = newOrderStream \
    .withWatermark("timestamp","1 minute") \
    .groupBy(window("timestamp", "1 minute", "1 minute")) \
    .agg(sum("total_cost").alias("total_sale_volume"),
         count("invoice_no").alias("OPM"),
         avg("is_return").alias("rate_of_return"),
         avg("total_cost").alias("average_transaction_size")
        ) \
    .select("window", "OPM", "total_sale_volume", "average_transaction_size",
"rate_of_return" )
```

Writing the time-based KPIs data to HDFS - into JSON files for each one-minute window, using 'append' output mode, setting truncate as false, and specifying the HDFS output path for both the KPI files and for their checkpoints. Ten 1-minute window batches were taken.

```
queryByTime = aggStreamByTime.writeStream \
    .format("json") \
    .outputMode("append") \
    .option("truncate","false") \
    .option("path","/home/hadoop/time_kpi") \
    .option("checkpointLocation","/home/hadoop/folder/time_kpi_checkpoints") \
    .trigger(processingTime="1 minute") \
    .start()
```

Calculating time-and-country-based KPIs (Total sale volume, OPM, Rate of return) having tumbling window of one minute and watermark of one minute. Here I grouped by window and country both.

```
aggStreamByCountry = newOrderStream \
    .withWatermark("timestamp", "1 minute") \
    .groupBy(window("timestamp", "1 minute", "1 minute"), "country") \
    .agg(sum("total_cost").alias("total_sale_volume"),
         count("invoice_no").alias("OPM"),
         avg("is_return").alias("rate_of_return")) \
    .select("window", "country", "OPM", "total_sale_volume", "rate_of_return" )
```

Writing the the time-and-country-based KPIs data to HDFS into JSON files for each one-minute window, using 'append' output mode, setting truncate as false, and specifying the HDFS output path for both the KPI files and for their checkpoints. Ten 1-minute window batches were taken.

```
queryByCountry = aggStreamByCountry.writeStream \
    .format("json") \
    .outputMode("append") \
    .option("truncate","false") \
    .option("path","/home/hadoop/country_kpi") \
    .option("checkpointLocation","/home/hadoop/folder/country_kpi_checkpoints") \
    .trigger(processingTime="1 minute") \
    .start()
```

Indicating Spark to await termination.

```
orderQuery.awaitTermination()
queryByTime.awaitTermination()
queryByCountry.awaitTermination()
```

Console commands

First, I created a cluster and logged in as 'hadoop' user.
Then, I transferred the spark-streaming.py file to emr cluster using WinScp.

Then, I used the following commands:

```
export SPARK_KAFKA_VERSION=0.10
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5 spark-streaming.py
```

Screenshot of summarized input values:

```
-------------------------------------------
Batch: 10
-------------------------------------------
+--------------+--------------+-------------------+----------+-----------+--------+---------+
|invoice_no    |country       |timestamp          |total_cost|total_items|is_order|is_return|
+--------------+--------------+-------------------+----------+-----------+--------+---------+
|154132556402368|United Kingdom|2023-09-24 13:16:25|-3.81     |3          |0       |1        |
|154132556402369|United Kingdom|2023-09-24 13:16:38|34.92     |26         |1       |0        |
|154132556402370|United Kingdom|2023-09-24 13:16:43|-24.67    |12         |0       |1        |
|154132556402371|United Kingdom|2023-09-24 13:16:48|1.6500001 |3          |1       |0        |
|154132556402372|United Kingdom|2023-09-24 13:16:56|57.600002 |80         |1       |0        |
|154132556402373|United Kingdom|2023-09-24 13:17:06|21.619999 |50         |1       |0        |
|154132556402374|United Kingdom|2023-09-24 13:17:10|4.92      |2          |1       |0        |
+--------------+--------------+-------------------+----------+-----------+--------+---------+
```

Screenshot of HDFS to make sure the KPI files were present:

hadoop fs -ls /home/hadoop/

```
[hadoop@ip-172-31-44-120 ~]$ hadoop fs -ls /home/hadoop/folder/
Found 2 items
drwxr-xr-x   - hadoop hadoop          0 2023-09-24 13:03 /home/hadoop/folder/country_kpi_checkpoints
drwxr-xr-x   - hadoop hadoop          0 2023-09-24 13:03 /home/hadoop/folder/time_kpi_checkpoints
[hadoop@ip-172-31-44-120 ~]$ hadoop fs -ls /home/hadoop/
Found 3 items
drwxr-xr-x   - hadoop hadoop          0 2023-09-24 13:13 /home/hadoop/country_kpi
drwxr-xr-x   - hadoop hadoop          0 2023-09-24 13:02 /home/hadoop/folder
drwxr-xr-x   - hadoop hadoop          0 2023-09-24 13:13 /home/hadoop/time_kpi
[hadoop@ip-172-31-44-120 ~]$
```

Screenshot of the folders to see the JSON files*:*

hadoop fs -ls /home/hadoop/time_kpi

```
[hadoop@ip-172-31-44-120 ~]$ hadoop fs -ls /home/hadoop/time_kpi
Found 222 items
drwxr-xr-x   - hadoop hadoop          0 2023-09-24 13:13 /home/hadoop/time_kpi/_spark_metadata
-rw-r--r--   1 hadoop hadoop          0 2023-09-24 13:08 /home/hadoop/time_kpi/part-00000-08724920-d3e6-42ca-97a1-10b8e5ddf0cf-c000.json
-rw-r--r--   1 hadoop hadoop          0 2023-09-24 13:05 /home/hadoop/time_kpi/part-00000-269a2984-1860-4bd9-a939-3ee4f6a33736-c000.json
-rw-r--r--   1 hadoop hadoop          0 2023-09-24 13:09 /home/hadoop/time_kpi/part-00000-49bfeb8e-3f65-4d4e-b0a2-4500aa1de0b7-c000.json
-rw-r--r--   1 hadoop hadoop          0 2023-09-24 13:03 /home/hadoop/time_kpi/part-00000-4ce4ae44-d862-4a2f-bbe4-15414a1a2bae-c000.json
-rw-r--r--   1 hadoop hadoop          0 2023-09-24 13:04 /home/hadoop/time_kpi/part-00000-62bc69b7-0ebc-4f47-a9b9-df21a99bf14a-c000.json
-rw-r--r--   1 hadoop hadoop          0 2023-09-24 13:13 /home/hadoop/time_kpi/part-00000-78d853e5-b614-4f03-9f9c-3f1480fe1ad4-c000.json
-rw-r--r--   1 hadoop hadoop      10980 2023-09-24 13:03 /home/hadoop/time_kpi/part-00000-7e1abcb5-44cb-407c-9bdb-3b021b525ba5-c000.json
-rw-r--r--   1 hadoop hadoop          0 2023-09-24 13:10 /home/hadoop/time_kpi/part-00000-c441084d-b2c2-42f1-9ce1-15a839bfc792-c000.json
-rw-r--r--   1 hadoop hadoop          0 2023-09-24 13:06 /home/hadoop/time_kpi/part-00000-c5a53e1c-e502-4537-95ca-5bb5611c3876-c000.json
-rw-r--r--   1 hadoop hadoop          0 2023-09-24 13:12 /home/hadoop/time_kpi/part-00000-cce973f2-c79f-48ea-b8a8-be4b361edc95-c000.json
-rw-r--r--   1 hadoop hadoop          0 2023-09-24 13:07 /home/hadoop/time_kpi/part-00000-edfa691b-c148-43bd-a2c8-fb06d3ac249d-c000.json
-rw-r--r--   1 hadoop hadoop          0 2023-09-24 13:11 /home/hadoop/time_kpi/part-00000-f45dbeac-5b71-47a8-a3ca-fbf1d87aeeff-c000.json
-rw-r--r--   1 hadoop hadoop       9193 2023-09-24 13:03 /home/hadoop/time_kpi/part-00001-e1cb6f4a-d11b-4e38-9333-8fd83ae5d63e-c000.json
-rw-r--r--   1 hadoop hadoop       5913 2023-09-24 13:03 /home/hadoop/time_kpi/part-00002-f6c1aee4-68ab-4f7d-ad1e-77d5aa72ed73-c000.json
-rw-r--r--   1 hadoop hadoop       5785 2023-09-24 13:03 /home/hadoop/time_kpi/part-00003-77ea7c1d-9f15-4cdf-9322-57a08c72ad59-c000.json
```

hadoop fs -ls /home/hadoop/country_kpi

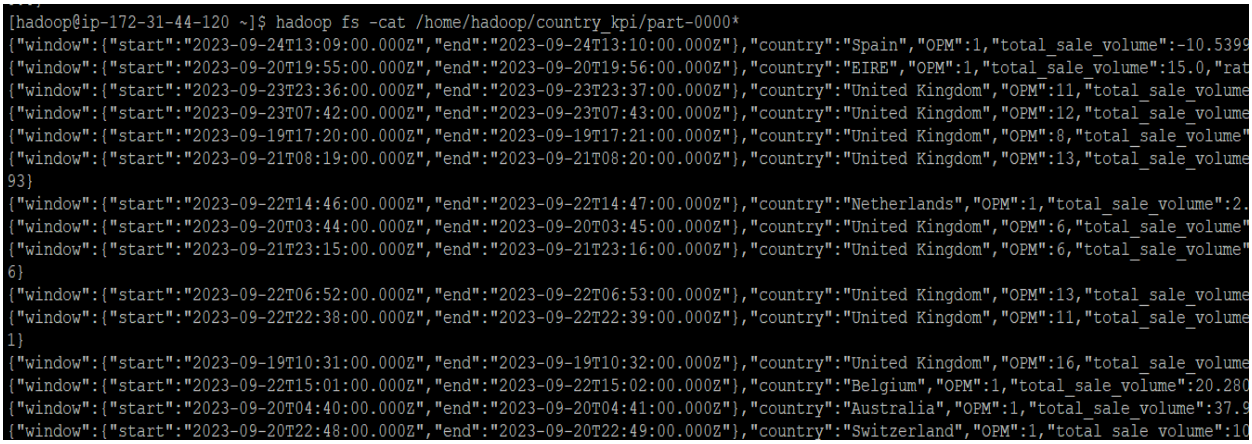
```
[hadoop@ip-172-31-44-120 ~]$ hadoop fs -ls /home/hadoop/country_kpi
Found 225 items
drwxr-xr-x   - hadoop hadoop          0 2023-09-24 13:13 /home/hadoop/country_kpi/_spark_metadata
-rw-r--r--   1 hadoop hadoop          0 2023-09-24 13:04 /home/hadoop/country_kpi/part-00000-1141cfc6-8382-4d55-b76a-6fa07a992acc-c000.json
-rw-r--r--   1 hadoop hadoop          0 2023-09-24 13:03 /home/hadoop/country_kpi/part-00000-50840071-25ec-45aa-9669-af167bc455af-c000.json
-rw-r--r--   1 hadoop hadoop          0 2023-09-24 13:05 /home/hadoop/country_kpi/part-00000-56d9d450-02bf-4f5a-9b3b-bf3d482a63f9-c000.json
-rw-r--r--   1 hadoop hadoop          0 2023-09-24 13:08 /home/hadoop/country_kpi/part-00000-588304e5-1f20-40e2-8dba-b2ed596eb19b-c000.json
-rw-r--r--   1 hadoop hadoop          0 2023-09-24 13:06 /home/hadoop/country_kpi/part-00000-67a13e40-0e7b-40bb-a79c-11fe4f1ce89a-c000.json
-rw-r--r--   1 hadoop hadoop          0 2023-09-24 13:13 /home/hadoop/country_kpi/part-00000-6f3cc959-f369-4e3a-884e-8194b1f1829d-c000.json
-rw-r--r--   1 hadoop hadoop          0 2023-09-24 13:12 /home/hadoop/country_kpi/part-00000-7387b26b-b447-4c07-b0cb-8b2099097be4-c000.json
-rw-r--r--   1 hadoop hadoop          0 2023-09-24 13:10 /home/hadoop/country_kpi/part-00000-7ea85c25-1de0-48d3-b7db-e6b6d9f707b3-c000.json
-rw-r--r--   1 hadoop hadoop        168 2023-09-24 13:09 /home/hadoop/country_kpi/part-00000-9728491e-b70e-42a4-a52e-ccb1ad59e436-c000.json
-rw-r--r--   1 hadoop hadoop          0 2023-09-24 13:11 /home/hadoop/country_kpi/part-00000-c0be3a34-ebaa-4a51-afd2-641134a66eb5-c000.json
-rw-r--r--   1 hadoop hadoop      13915 2023-09-24 13:03 /home/hadoop/country_kpi/part-00000-c90edde1-9e10-4146-a5be-f55028c27bd0-c000.json
-rw-r--r--   1 hadoop hadoop          0 2023-09-24 13:07 /home/hadoop/country_kpi/part-00000-e5a43fc3-a41c-4c25-a4f1-13277f62a7b6-c000.json
-rw-r--r--   1 hadoop hadoop      14968 2023-09-24 13:03 /home/hadoop/country_kpi/part-00001-8c37925b-2c26-4393-adce-9c9b3a0d4365-c000.json
-rw-r--r--   1 hadoop hadoop      14903 2023-09-24 13:03 /home/hadoop/country_kpi/part-00002-f107b3fa-1a95-4644-a5bb-7c8637a5f555-c000.json
-rw-r--r--   1 hadoop hadoop      11758 2023-09-24 13:03 /home/hadoop/country_kpi/part-00003-148a4610-bc84-4a71-8f1c-6d984b2e06f7-c000.json
-rw-r--r--   1 hadoop hadoop      13369 2023-09-24 13:03 /home/hadoop/country_kpi/part-00004-8f2bee44-e1c2-4c50-ad05-efe68577d95b-c000.json
-rw-r--r--   1 hadoop hadoop      15465 2023-09-24 13:03 /home/hadoop/country_kpi/part-00005-35986cf7-d1a2-40cf-94a8-1c7e8c079b85-c000.json
-rw-r--r--   1 hadoop hadoop      10051 2023-09-24 13:03 /home/hadoop/country_kpi/part-00006-1d0e5dbb-452c-4fea-a88e-e11af771d428-c000.json
-rw-r--r--   1 hadoop hadoop      17337 2023-09-24 13:03 /home/hadoop/country_kpi/part-00007-7d7c708a-e5b4-409e-aafc-4ff5ed22bc30-c000.json
-rw-r--r--   1 hadoop hadoop      12985 2023-09-24 13:03 /home/hadoop/country_kpi/part-00008-b036f60d-c4a6-46f1-b7a0-59d0e5b196ee-c000.json
-rw-r--r--   1 hadoop hadoop      15037 2023-09-24 13:03 /home/hadoop/country_kpi/part-00009-ba2cdba2-4a28-47e8-b532-059d4fea0a9a-c000.json
```

Screenshot of 'cat' command to take a look at the data:

hadoop fs -cat /home/hadoop/time_kpi/part-0000*

```
[hadoop@ip-172-31-44-120 ~]$ hadoop fs -cat /home/hadoop/time_kpi/part-0000*
{"window":{"start":"2023-09-22T21:54:00.000Z","end":"2023-09-22T21:55:00.000Z"},"OPM":10,"total_sale_volume":686.3700330257416,"average_tran
0.0}
{"window":{"start":"2023-09-22T23:17:00.000Z","end":"2023-09-22T23:18:00.000Z"},"OPM":14,"total_sale_volume":1098.4699983596802,"average_tra
:0.2857142857142857}
{"window":{"start":"2023-09-19T10:21:00.000Z","end":"2023-09-19T10:22:00.000Z"},"OPM":18,"total_sale_volume":2714.0899543762207,"average_tra
":0.05555555555555555}
{"window":{"start":"2023-09-19T17:24:00.000Z","end":"2023-09-19T17:25:00.000Z"},"OPM":17,"total_sale_volume":3267.5999822616577,"average_tra
":0.11764705882352941}
{"window":{"start":"2023-09-19T20:39:00.000Z","end":"2023-09-19T20:40:00.000Z"},"OPM":13,"total_sale_volume":467.5199947357178,"average_trans
:0.15384615384615385}
{"window":{"start":"2023-09-20T20:54:00.000Z","end":"2023-09-20T20:55:00.000Z"},"OPM":5,"total_sale_volume":164.03000116348267,"average_tran
0.0}
{"window":{"start":"2023-09-21T16:08:00.000Z","end":"2023-09-21T16:09:00.000Z"},"OPM":9,"total_sale_volume":1044.1299837827682,"average_tran
0.0}
{"window":{"start":"2023-09-22T19:23:00.000Z","end":"2023-09-22T19:24:00.000Z"},"OPM":7,"total_sale_volume":562.7900018692017,"average_transa
.0}
{"window":{"start":"2023-09-20T00:43:00.000Z","end":"2023-09-20T00:44:00.000Z"},"OPM":15,"total_sale_volume":675.0599926710129,"average_tran
:0.0}
{"window":{"start":"2023-09-22T12:05:00.000Z","end":"2023-09-22T12:06:00.000Z"},"OPM":6,"total_sale_volume":560.1399917602539,"average_transa
.16666666666666666}
{"window":{"start":"2023-09-19T18:32:00.000Z","end":"2023-09-19T18:33:00.000Z"},"OPM":11,"total_sale_volume":1194.5299520492554,"average_tra
":0.0}
```

*hadoop fs -cat /home/hadoop/country_kpi/part-0000**



Then, I created a new folder using WinScp and transferred the directory containing time-kpi and country-kpi from HDFS to the new folder.
Then, I downloaded the folder using WinScp.

THANK YOU