# Code Logic - Retail Data Analysis

Below are the steps followed to achieve the ask from this assignment

**PRE-REQUISITE:**

- Run the command, “wget https://ds-spark-sql-kafka-jar.s3.amazonaws.com/spark-sql-kafka-0-10_2.11-2.3.0.jar" in the instance we are running this code , to be able to connect to Kafka from spark
- Run the command , “export SPARK_KAFKA_VERSION=0.10”

**STEP 1 :**

Setting up the dependancies by importing the necessary packages, setting up necessary functions to create the UDF later

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

Below four python functions are created for UDF usage later,

- isOrder : This function is written in order to take in the transaction type and return 1 if type is of “ORDER” else 0.
- isReturn : This function is written in order to take in the transaction type and return 1 if type is of “RETURN” else 0.
- itemsOrderTotal : This function is written in order to calculate the total cost of the items purchased within an order.If the transaction type is “ORDER” then we return “(unitPrice * quantity)” else return “(unitPrice * quantity) * -1”
- itemQuantityTotal: This function is responsible to return the sum of the items present in an order

**STEP 2 :**

Setting up the spark session with applicationName “RetailStreamingAnalysis”

```
spark = SparkSession  \
        .builder  \
        .appName("RetailStreamingAnalysis")  \
        .getOrCreate()
```

**STEP 3 :**

Setting up spark to read from Kafka server (we are fetching the latest values to have the output easily reviewed)

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

**STEP 4 :**

Converting incoming Raw Data to string format for processing

Specific Schema structure is defined as per the incoming data.The incoming json data is extracted as string and as per the schema extracted into the respective columns

```
# Defining Schema based on the input provided
    schema =  StructType([
        StructField("country", StringType()),
        StructField("invoice_no", LongType()) ,
        StructField("items", ArrayType(
            StructType([
                StructField("SKU", StringType()),
                StructField("title", StringType()),
                StructField("unit_price", FloatType()),
                StructField("quantity", IntegerType())
            ])
        )),
        StructField("timestamp", TimestampType()),
        StructField("type", StringType()),
        ])

# Casting column value raw data as string and aliasing
rawToStringCastDF = lines.selectExpr('CAST(value AS STRING)') \
    .select(from_json('value', schema).alias("value"))
```

**STEP 5 :**

Flattening the items columns values

Using the UDF functions we are calculating the nested object items and returning a single row entry per order

```
# Using the UDF fetching hte total order cost and total quantity per order
    flattenedDF = rawToStringCastDF.select(“value.country",
                                “value.invoice_no",
                                “value.items.unit_price",
                                “value.items.quantity”,
                                “value.timestamp",
                                "value.type")
    #New UDF function created from the functions created on step1
    item_Order_TotalCost = udf(itemOrderTotal,FloatType())
    order_TotalQuantity = udf(itemQuantityTotal,IntegerType())

    flattenedDF=flattenedDF.withColumn(“cost”,item_Order_TotalCost(
                                        flattenedDF.unit_price,
                                        flattenedDF.quantity,
                                        flattenedDF.type))

    flattenedDF=flattenedDF.withColumn(“total_items",
        order_TotalQuantity(flattenedDF.quantity))
```

**STEP 6 :**

Pre-processing Readiness : Creating the UDF needed for the KPI generation

Creating three UDF based on the functions created in step 1

is_Order:This UDF is linked with isOrder function defined on step 1 return a integer and takes input ordertype

is_Return:This UDF is linked with isReturn function defined on step 1 return a integer and takes input ordertype

Adding the above two UDF to the DataFrame to create the columns is_order, is_return

```
flattenedDF=flattenedDF.withColumn("is_order",is_Order(flattenedDF.type))
flattenedDF=flattenedDF.withColumn("is_return",is_Return(flattenedDF.type))
```

**STEP 7 :**

Generating source table for KPI and the finalised summarised input for console
All the necessary columns needed have been created/calculated,generating the data with new columns for each order for window of 1 minute.

```
#Input for the Time based and Time-Country Based KPI
finalInputValuesDF =
flattenedDF.select("invoice_no","country","timestamp","ts","cost","total_items","is_order","is_return"
)

#Input for the console streaming input
finalInputValueswindowedDF = finalInputValuesDF.withWatermark("ts","10 minutes") \
.groupby(window("ts","1 minute"),"invoice_no","timestamp","country","is_order","is_return") \
.agg(sum("cost").alias("total_cost"),sum("total_items").alias("total_items"))
```

**STEP 8 :**

Generating the Time based KPI

Fetching the necessary columns from the source ,

```
timeBasedInput = finalInputValuesDF.select(“invoice_no",
                                           “timestamp",
                                           “cost","total_items",
                                           "is_order","is_return")
```

Setting the tumbling window of 1 min for the incoming transactions , and processing the necessary columns

```
timeBasedKPI = timeBasedInput.withWatermark("timestamp","10 minutes") \
                .groupby(window("timestamp","1 minute")) \
                .agg(round(sum("cost"),2).alias("total_sale_volume"), \
                        approx_count_distinct("invoice_no").alias("OPM"), \
                        sum("total_items").alias("total_items"), \
                        sum("is_order").alias("total_order"), \
                        sum("is_return").alias("total_return"), \
                )
```

**KPI Calculation:**

**Orders per minute:** Fetching the count of distinct invoices in that 1 min

approx_count_distinct(“invoice_no").alias("OPM")

**Total volume of sales:** UDF item_Order_TotalCost return the cost value of the invoice based on the type ,using the sum of this to get the total volume of sales based on the interval.Below is the calculation performed

round(sum(“cost"),2).alias("total_sale_volume")

**Rate of Returns:** UDF is_Order and is_Return is considered here and its sum is taken here as total_return and total_order.Below is the calculation performed

```
timeBasedKPI = timeBasedKPI.withColumn(“rate_of_return”,
        round(timeBasedKPI.total_return /
        (timeBasedKPI.total_order+timeBasedKPI.total_return),2))
```

**Average Transaction Size:** we take the outputs of the UDF we have created and perform the below calculation to achieve this.

```
timeBasedKPI = timeBasedKPI.withColumn(“average_transaction_size",
                round(timeBasedKPI.total_sale_volume/
                (timeBasedKPI.total_order+timeBasedKPI.total_return),2))
```

**STEP 9 :**

Generating the Time and Country based KPI

Fetching the necessary columns from the source ,

```
timeCountryBasedInput = finalInputValuesDF.select(“invoice_no",
                        “country","timestamp",
                        “total_items","quantity","is_order",
                        "is_return")
```

Setting the tumbling window of 1 min for the incoming transactions , and processing the necessary columns

```
timeAndCountryBasedKPI =  \
timeCountryBasedInput.withWatermark(“timestamp","10 minutes") \
.groupby(window("timestamp","1 minute"),"country") \
.agg(round(sum("cost"),2).alias("total_sale_volume"), \
        approx_count_distinct("invoice_no").alias("OPM"), \
        sum("total_items").alias("total_items"), \
        sum("is_order").alias("total_order"), \
        sum("is_return").alias("total_return"), \
)
```

**KPI Calculation:**

**Orders per minute:** Fetching the count of distinct invoices in that 1 min

approx_count_distinct(“invoice_no").alias("OPM")

**Total volume of sales:** UDF item_Order_TotalCost return the cost value of the invoice based on the type ,using the sum of this to get the total volume of sales based on the interval.Below is the calculation performed

round(sum(“cost"),2).alias("total_sale_volume")

**Rate of Returns:** UDF is_Order and is_Return is considered here and its sum is taken here as total_return and total_order.Below is the calculation performed

```
timeAndCountryBasedKPI =
        timeAndCountryBasedKPI.withColumn(“rate_of_return",
        round(timeAndCountryBasedKPI.total_return/
(timeAndCountryBasedKPI.total_order+timeAndCountryBasedKPI.total_return),2))
```

**STEP 10 :**

Generating the outputs for the input and the KPI’s and terminating the queries

```
# Printing the FinalSummarizedInput Value into the console
finalSummarizedInputQuery = finalInputValueswindowedDF  \
        .writeStream  \
        .outputMode("append")  \
        .option("truncate","false") \
        .format("console")  \
        .start()

# Print time based KPI to HDFS path as a JSON
timeKPIJSONQuery = timeBasedKPI.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()

# Print time and country based KPI to HDFS path as a JSON
timeCountryKPIJSONQuery = timeAndCountryBasedKPI.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()
```

**STEP 11 :**

Save the file in the instance , and run the below commend
spark2-submit --jars spark-sql-kafka-0-10_2.11-2.3.0.jar spark-streaming.py > console_output

**STEP 12 :**

As the program completes,

Fetch the console_output file to local along with the below two folders present in the Hadoop file system,

time_country_KPI

time_KPI