Code Logic - Retail Data Analysis

Calculating UDFs

for Total_Items

```
def total_item_count(items):
  total_items = 0
  for item in items:
     total_items = total_items + item[2]
  return total_items
```

for Total_cost

```
def total_cost_per_record(items):
  total_cost = 0
  for item in items:
     total_cost = total_cost + (item[2] * item[3])
  return total_cost
```

for order type flag

```
def order_type(type):
   order_type_flag = 0
   if type == 'ORDER':
      order_type_flag = 1
   else:
      order_type_flag = 0
   return order_type_flag
```

for return type flag

```
def 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
```

Reading Data from Kafka using the Read Stream

```
spark = SparkSession  \
        .builder  \
        .appName("StructuredSocketRead")  \
        .getOrCreate()
spark.sparkContext.setLogLevel('ERROR')

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()
```

Defining UDFs

```
adding_total_cost = udf(total_cost_per_record, DoubleType())
adding_total_item = udf(total_item_count, DoubleType())
adding_is_order_flg = udf(order_type, IntegerType())
adding_is_return_flg = udf(order_return_type, IntegerType())
```

Defining Schema and Parsing from JSON

```
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())
    ])))

ordersStream = orderRaw.select(from_json(col("value").cast("string"),
jsonSchema).alias("data")).select("data.*")
```

Creating New Columns

```
DF_Total_Item_Cost = ordersStream \
    .withColumn("total_cost", adding_total_cost(ordersStream.items)) \
    .withColumn("total_items", adding_total_item(ordersStream.items)) \
    .withColumn("is_order", adding_is_order_flg(ordersStream.type)) \
    .withColumn("is_return",
adding_is_return_flg(ordersStream.type)).select("invoice_no","country","timestamp","total_cost",
"total_items","is_order", "is_return")
```

Writing Data into Console

```
query = DF_Total_Item_Cost  \
        .writeStream  \
        .outputMode("append")  \
        .format("console")  \
        .option("truncate", "false") \
        .start()
```

Calculating Time Based KPIs

```
aggStreamByTime = DF_Total_Item_Cost \
    .withWatermark("timestamp", "1 minute") \
    .groupBy(window("timestamp", "1 minute", "1 minute")) \

.agg(count("invoice_no").alias("OPM"),sum("total_cost").alias("total_sale_volume"),avg("is_retur
n").alias("rate_of_return")).select("window", "OPM", "total_sale_volume", "rate_of_return")
```

Calculating TimeCountryBasedKPIs

```
aggStreamByTimeCountry= DF_Total_Item_Cost \
    .withWatermark("timestamp", "1 minute") \
    .groupBy(window("timestamp", "1 minute", "1 minute"), "country") \

.agg(count("invoice_no").alias("OPM"),sum("total_cost").alias("total_sale_volume"),avg("is_retur
n").alias("rate_of_return")).select("window","country", "OPM", "total_sale_volume",
"rate_of_return")
```

Writing Time Based KPIs into HDFS

```
queryByTime= aggStreamByTime.writeStream \
    .format("json") \
    .outputMode("append") \
    .option("truncate", "false") \
    .option("path", "time-wise-kpi") \
    .option("checkpointLocation", "time-wise-cp") \
    .trigger(processingTime="1 minute") \
    .start()
```

Writing Time Country Based KPIs into HDFS

```
queryByTimeCountry = aggStreamByTimeCountry.writeStream \
    .format("json") \
    .outputMode("append") \
    .option("truncate", "false") \
    .option("path", "time-country-wise-kpi") \
    .option("checkpointLocation", "time-country-wise-cp") \
    .trigger(processingTime="1 minute") \
    .start()

queryByTimeCountry.awaitTermination()
```

Spark Submit

```
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
18.211.252.152 9092 real-time-project
```