Code Logic - Retail Data Analysis

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

Code Explanation:

Start with all the libraries required for the spark streaming

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark.sql import functions as F
from pyspark.sql.functions import from_json
from pyspark.sql.window import Window
```

After we have a target to start spark session and start fetching the data from Kafka

```
# creating spark session
spark = SparkSession  \
.builder  \
.appName("Retail")  \
.getOrCreate()
spark.sparkContext.setLogLevel('ERROR')

# Reading Kafka Stream
lines = spark  \
.readStream  \
.format("kafka")  \
.option("kafka.bootstrap.servers","18.211.252.152:9092")  \
.option("subscribe","real-time-project")  \
.option("startingOffsets", "earliest") \
.load()

lines.printSchema()
```

lines.printSchema() : use to print the data schema on console which help to understand which column contains key or value, etc

Now, data in raw format, next target to fetch the “value” which contain data fetching from kafka and cast into string so that we can ready and operate on it.

```
# Casting value as Stringwhich help to read column better
kafkaDF = lines.selectExpr("cast(value as string)")
kafkaDF.printSchema()
```

If you read data it was in json format but due to casting it is showing each row as String Data type.
Raw format is not in a workable format, so we need to customize it so that we can write different Spark SQL to fetch data for business KPI

First step is update the Schema which include multiple steps as data is in multi hierarchy.
Code snapshot of creating input data stream:

```
# creating user defined schema which help to read operate data better
schema = StructType() \
.add("invoice_no",StringType()) \
.add("country",StringType()) \
.add("timestamp",TimestampType()) \
.add("type",StringType()) \
.add("items", ArrayType(StructType() \
.add("SKU", StringType()) \
.add("title", StringType()) \
.add("unit_price", DoubleType()) \
.add("quantity", IntegerType())))

# creating inital DF as per user define schema
initialDF = kafkaDF.select(from_json(col("value"), schema).alias("data"))

initialDF.printSchema()

explodeInitialDF=initialDF.select(col("data.*"))
```

Initial formatted input schema look like:

```
root
 |-- data: struct (nullable = true)
 |    |-- invoice_no: string (nullable = true)
 |    |-- country: string (nullable = true)
 |    |-- timestamp: timestamp (nullable = true)
 |    |-- type: string (nullable = true)
 |    |-- items: array (nullable = true)
 |    |    |-- element: struct (containsNull = true)
 |    |    |    |-- SKU: string (nullable = true)
 |    |    |    |-- title: string (nullable = true)
 |    |    |    |-- unit_price: double (nullable = true)
 |    |    |    |-- quantity: integer (nullable = true)
```

Now, next step is to create different columns which help to create our KPI queries.

1. Create **Total_Cost** column with help of multiplying the unit_price with quantity

```
#Calculating total_cost by multiplying unit_price with quantity
customDF = finalDF.select(col("invoice_no"),col("country"),col("timestamp"),col("type"), col("SKU"), col("title"), col("unit_price"), col("quantity"), (col("unit_price")*col("quantity")).alias("total_cost"))
```

2. After that we need to create is_order and is_return column.

a. Is_order is 1 if type is ORDER otherwise 0. Code snippet:

```
#Using UDF to create new columns of is_order & is_return
#UDF function for is_order
@udf(returnType=IntegerType())
def is_order(x):
    if x=='ORDER':
        return 1
    else:
        return 0

#applying UDF to create new DF
orderDF=customDF.select(col("*"), \
    is_order(col("type")).alias("is_order") )
```

We use UDF(User Define Function) to create is_oder

b. Is_return is 1 if type is RETURN otherwise 0. Code Snippet:

```
#UDF for is_return column
@udf(returnType=IntegerType())
def is_return(x):
    if x=='ORDER':
        return 0
    else:
        return 1

returndf=orderDF.select(col("*"), \
    is_return(col("type")).alias("is_return") )
```

again, we use UDF for it.

c. Finally is to remove the type column as it is not required. So final schema is :

```
root
 |-- invoice_no: string (nullable = true)
 |-- country: string (nullable = true)
 |-- timestamp: timestamp (nullable = true)
 |-- SKU: string (nullable = true)
 |-- title: string (nullable = true)
 |-- unit_price: double (nullable = true)
 |-- quantity: integer (nullable = true)
 |-- total_cost: double (nullable = true)
 |-- is_order: integer (nullable = true)
 |-- is_return: integer (nullable = true)
```

It time to create the KPI and Input queries

1. Input Query:

```
#Initial qrite stream query to write final input data on console
inputQuery = finalDFInput.groupBy("invoice_no","country","timestamp") \
        .agg(sum("total_cost").alias("total_cost"), \
             sum("quantity").alias("total_items")).writeStream  \
.outputMode("update")  \
.format("console")  \
.trigger(processingTime='1 minute') \
.start()
```

Time Base KPI Spark SQL

```
#Startig KPI Spark Query
#First KPI i.e. Time Based KPI
time_kpi_stream = finalDFInput \
    .withWatermark("timestamp","1 minute") \
    .groupBy(window("timestamp","1 minute" , "1 minute").alias("time")) \
    .agg(sum("total_cost"),avg("total_cost"),count("invoice_no").alias("OPM"), avg("is_return")) \
    .select("time",
            "OPM",
            format_number("sum(total_cost)",2).alias("total_sale_volume"),
            format_number("avg(total_cost)",2).alias("average_transaction_size"),
            format_number("avg(is_return)",2).alias("rate_of_return"))

time_kpi_stream.printSchema()
```

Write Schema query:

```
#Time KPI query to store output steam as json on 1 min window
timeKPIQuery = time_kpi_stream  \
.writeStream  \
.format("json") \
.outputMode("append")  \
.option("truncate", "false") \
.option("path", "time_kpi") \
.option("checkpointLocation","timecp1") \
.trigger(processingTime='1 minute') \
.start()
```

It help to get the data in Json format at location “time_kpi”

Time and Country Base KPI Spark SQL

```
#Second KPI i.e. Time and Country Base KPI
time_country_kpi_stream = finalDFInput \
    .withWatermark("timestamp","1 minute") \
    .groupBy(window("timestamp","1 minute" , "1 minute").alias("time"),"country") \
    .agg(sum("total_cost"), count("invoice_no").alias("OPM"), avg("is_return")) \
    .select("time",
            "OPM",
            format_number("sum(total_cost)",2).alias("total_sale_volume"),
            format_number("avg(is_return)",2).alias("rate_of_return"))

time_country_kpi_stream.printSchema()
```

Write query for time and country kpi:

```
#Time and Country KPI query to store output steam as json on 1 min window
timeCountryQuery = time_country_kpi_stream  \
.writeStream  \
.format("json") \
.outputMode("append")  \
.option("truncate", "false") \
.option("path", "time_country_kpi") \
.option("checkpointLocation","timecountrycp") \
.trigger(processingTime='1 minute') \
.start()

timeCountryQuery.awaitTermination()
```

And final line is to terminate the query using external intervention.