Code Logic - Retail Data Analysis

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

- In this Retail-Data-Analysis project our job is to read the data from the kafka stream and do real time processing over the streamed data.

- In order to achieve this, I have written a pySpark code for reading and processing data.

- First I have created a spark session (app name - Structured Streaming) then started reading the data from kafka server (details of kafka server provided in assignment).

```
# Creating spark session spark = SparkSession \
.builder \
.appName("Structured Streaming ")\ .getOrCreate()
spark.sparkContext.setLogLevel("ERROR")
# Code to fetch the data from kafka server df = spark \
.readStream \
.format("kafka")
\ .option("kafka.bootstrap.servers","18.211.252.152:9092
")\ .option("subscribe","real-time-project")\
.load()
df1 = df.selectExpr("CAST(value AS STRING)”)
```

  - Data what we are getting from kafka server are

json so I have defined a schema to read the json data.

```
# Code to define the schema of single order schema =
StructType()\
.add("invoice_no",StringType())
\ .add("country",StringType())
\ .add("timestamp",TimestampType())
\ .add("type",StringType())\ .add("items",StringType())
```

```
df2 = df1.select(from_json(col("value"),schema).alias("orders"))

df3 = df2.select(“orders.*")
```

• Then I have defined some UDFs (qunatityUDF, priceUDF, orderUDF, returnUDF & checkUDF) to create additional columns (total_cost, is_order, is_return and total_items).

```
# Code to calculate the new UDFs and other utility function
def quantity(a):

a = ast.literal_eval(a) quantity = 0
for i in range (len(a)):

quantity = quantity + a[i].get('quantity') return(quantity)

def price(a):
a = ast.literal_eval(a) price = 0.0
for i in range (len(a)):

price = price + a[i].get('quantity')*a[i].get('unit_price')
return(price)

def is_order(a):
if a == "ORDER":
```

```
return(1) else:

return(0)

def is_return(a):
if a == "RETURN":

return(1) else:

return(0)

def check_type(a,b): if a == 1:

b = b * -1.0 return(b)

quantityUDF = udf(lambda t:quantity(t),IntegerType())
priceUDF = udf(lambda t:price(t),DoubleType()) orderUDF =
udf(lambda t:is_order(t),IntegerType())


returnUDF = udf(lambda t:is_return(t),IntegerType()) checkUDF
= udf(lambda t,u:check_type(t,u),DoubleType())

df4 = df3.withColumn("total_items",quantityUDF(df3.items))
df5 = df4.withColumn("total_cost",priceUDF(df4.items))
df6 = df5.withColumn("is_order",orderUDF(df5.type))
df7 = df6.withColumn("is_return",returnUDF(df6.type))

df8 =
df7.withColumn("total_cost",checkUDF(df7.is_return,df7.total
_cost))
df9 =
df8.select(“invoice_no","country","timestamp","total_cost","to
tal_items","is_ord er","is_return")
```

• This will create a summarised table which will be printed in the console.

```
# Code to write final summarised input values to the console
query1 = df9 \
```

.writeStream \ .outputMode("append")\ .format("console") \ .option('truncate',False)\ .trigger(processingTime = '1 minute')\ .start()

• Then finally we have to calculate some KPIs based on time & country. These KPIs have to be calculated for 1 minute window so I have defined a 1 minute tumbling window.

Code to calculate time based KPI with tumbling window of one minute df10 = df9.withWatermark("timestamp","1 minute")\

.groupBy(window('timestamp','1 minute','1 minute')) \ .agg(sum('total_cost'),

count('invoice_no').alias('OPM'),

avg('is_return'))\ .select('window',

'OPM',
format_number('sum(total_cost)',2).alias('total_sale_volume'),

format_number(regexp_replace(format_number('sum(total_cost)',2),",",""))/
format_number('OPM',2),2).alias('average_transaction_size'),

format_number('avg(is_return)',2).alias('rate_of_return'))
# Code to calculate country based KPI with tumbling window of one minute

df11 = df9.withWatermark("timestamp","1 minute") \ .groupBy(window('timestamp','1 minute','1 minute'),'country')\ .agg(sum('total_cost'),

count('invoice_no').alias('OPM'),

avg('is_return'))\ .select('window',

'country',
'OPM',
format_number('sum(total_cost)',2).alias('total_sale_volume'),
format_number(‘avg(is_return)',2).alias('rate_of_return'))

• Finally I am writing the processed query in json format in hdfs.

Code to write time based KPI into one minute window each query4 = df10\

.writeStream\
.format('json')\
.outputMode("append")\ .option('truncate',False)
\ .option('path','time-kpi')\ .option('checkpointLocation','time-cp1')\ .trigger(processingTime = '1 minute')\ .start()

Code to write country based KPI into one minute window each query5 = df11\

.writeStream\
.format('json')\
.outputMode("append")\ .option('truncate',False)
\ .option('path','time-country-kpi')
\ .option('checkpointLocation','time-country-cp1')
\ .trigger(processingTime = '1 minute')\

.start()

1. In order to run this .py file I have created an EMR cluster and ssh into it. 2. To create .py file in cluster -> vi file_name.py

3. To run .py file -> spark-submit --packages org.apache.spark:spark-sql- kafka-0-10_2.11:2.4.5 spark-streaming.py

4. To check whether folders are created or not -> hadoop fs -ls

5. To check json files are created or not -> hadoop fs -ls folder_name/

6. To check the content of json file -> hadoop fs -cat path_to_json_file

7. To move json to s3 bucket -> hadoop distcp folder_name s3://bucket_name/ folder_name/

8. To download folder from s3 -> aws s3 cp s3://bucket_name/folder_name/ local_path