Code Logic - Retail Data Analysis

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

Step-1: Import the required libraries/modules and set-up PySpark environment

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

Step-2: Initialize SparkSession

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

Notes: The above steps(#1 and #2) are being utilized to include all required libraries followed by intialization of Spark session.

Step-3: Reading input data from Kafka server

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

Notes: Step#3 is being utilized to connect Kafka server by leveraging the topic name provided to read the input data.

Step-4: Define the schema for incoming data

```
define_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", DoubleType()),
        StructField("quantity", IntegerType())])))
```

Notes: Step#4 is being utilized to define the schema for incoming data.

Step-5: Create dataframe from the input data

```
order_df = raw_stream_data.select(from_json(col("value").cast("string"),
define_schema).alias("data")).select("data.*")
```

Step-6: Define user-defined functions(UDF's)

- **UDF to calculate total_items**

```
def total_items(items):
  total_items_count = 0
  for item in items:
     total_items_count = total_items_count + item['quantity']
  return total_items_count
```

- **UDF to calculate order type**

```
def is_order(type):
  if type=="ORDER":
     return 1
  else:
     return 0
```

- **UDF to calculate return type**

```
def is_return(type):
  if type=="RETURN":
     return 1
  else:
     return 0
```

- **UDF to calculate total_cost**

```
def total_cost_sum(items,type):
  total_sum = 0
  for item in items:
     total_sum = total_sum + item['unit_price'] * item['quantity']
  if type=="RETURN":
     return total_sum * (-1)
  else:
     return total_sum
```

- **Convert UDF's with utility functions**

```
totalcount = udf(total_items, IntegerType())
isorder = udf(is_order, IntegerType())
isreturn = udf(is_return, IntegerType())
totalcost = udf(total_cost_sum, DoubleType())
```

- **Calculating columns(total_cost,total_items,is_order,is_return)**

```
order_stream_data = order_df \
      .withColumn("total_cost", totalcost(order_df.items, order_df.type)) \
      .withColumn("total_items", totalcount(order_df.items)) \
      .withColumn("is_order", isorder(order_df.type)) \
      .withColumn("is_return", isreturn(order_df.type))
```

Notes: The above steps(#5 and #6) are being utilized to create dataframe from the input data followed by defining user defined functions(UDF's) to calculate total_cost, total_items, is_order and is_return columns.

Step-7: Write intermediate dataset to the console with one-minute interval

```
output_to_console = order_stream_data \
      .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()
```

Notes: Step#7 helps to write intermediate dataset to the console with one-minute interval as per requirement.

Step-8: Calculate time-based KPIs

```
time_based_KPI = order_stream_data \
   .withWatermark("timestamp", "1 minute") \
   .groupby(window("timestamp", "1 minute", "1 minute")) \
   .agg(count("invoice_no").alias("OPM"),
       sum("total_cost").alias("total_sales_volume"),
       avg("total_cost").alias("average_transaction_size"),
       avg("is_return").alias("rate_of_return")) \
   .select("window", "OPM", "total_sales_volume", "average_transaction_size", "rate_of_return")
```

Step-9: Write time based KPI to JSON files

```
time_based_KPI_output_files = time_based_KPI \
   .writeStream \
   .outputMode("Append") \
   .format("json") \
   .option("format","append") \
   .option("truncate", "false") \
   .option("path", "timebasedKPI/") \
   .option("checkpointLocation", "timebasedKPI/checkpoint/") \
   .option("truncate", "False") \
   .trigger(processingTime="1 minute") \
   .start()
```

Notes: The above steps(#8 and #9) are being leveraged to calculate time based KPIs & write to JSON files.

Step-10: Calculate time and country-based KPIs

```
time_and_country_based_KPI = order_stream_data \
    .withWatermark("timestamp", "1 minute") \
    .groupby(window("timestamp", "1 minute", "1 minute"), "country") \
    .agg(count("invoice_no").alias("OPM"),
        sum("total_cost").alias("total_sales_volume"),
        avg("is_return").alias("rate_of_return")) \
    .select("window", "country", "OPM", "total_sales_volume", "rate_of_return")
```

Step-11: Write time and country-based KPI to JSON files

```
time_and_country_based_KPI_output = time_and_country_based_KPI \
    .writeStream \
    .outputMode("Append") \
    .format("json") \
    .option("format","append") \
    .option("truncate", "false") \
    .option("path", "timecountrybasedKPI/") \
    .option("checkpointLocation","timecountrybasedKPI/checkpoint/") \
    .trigger(processingTime="1 minute") \
    .start()
```

Notes: The above steps(#10 and #11) are being leveraged to calculate time and country based KPIs & write to JSON files.

Step-12: Waiting for termination

```
time_and_country_based_KPI_output.awaitTermination()
```

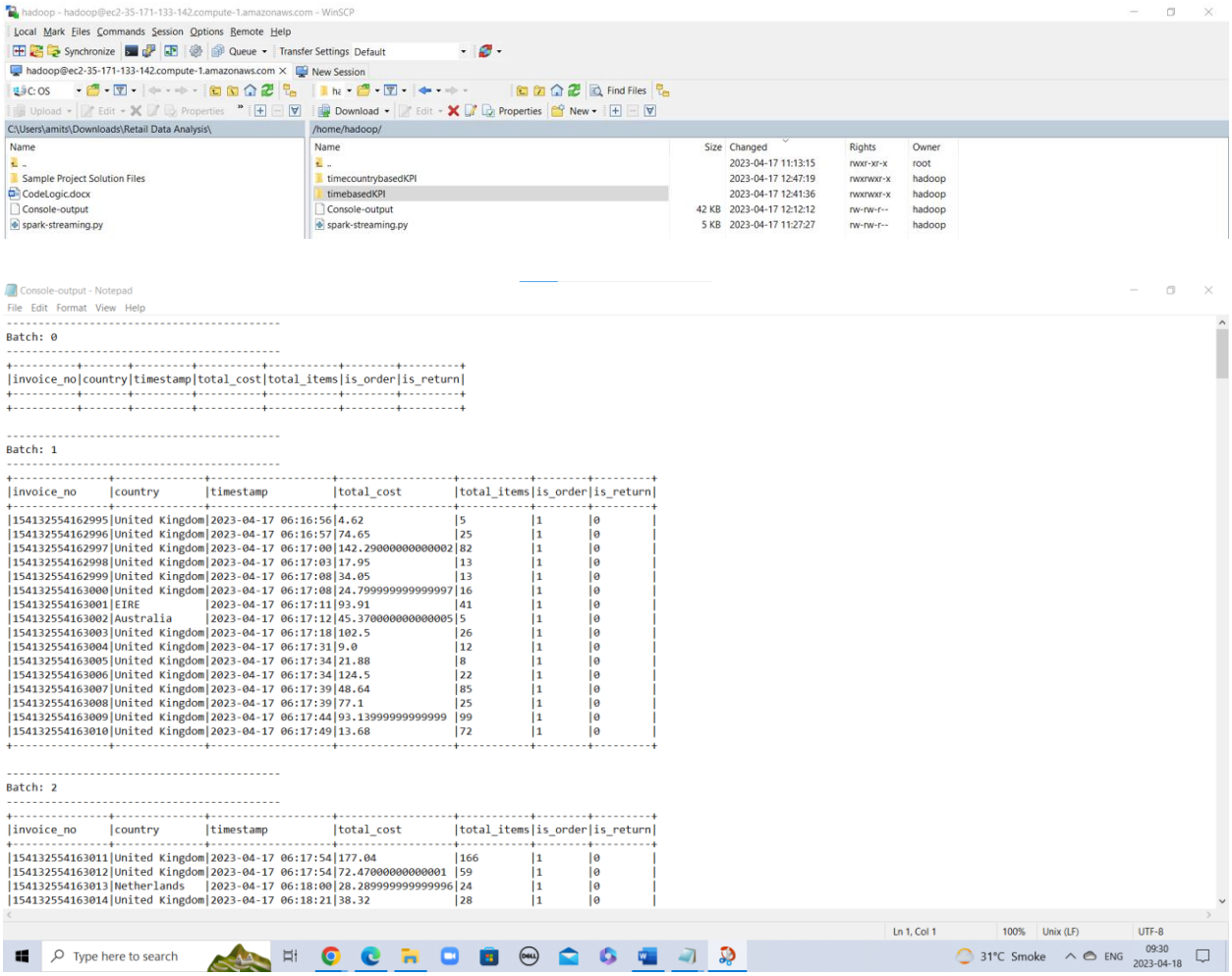
Notes: The above step(#12) waits for the termination signal from the user.

❖ **Snapshot of console commands and validate JSON files generated:**

1. Spark Submit Command to generate console output

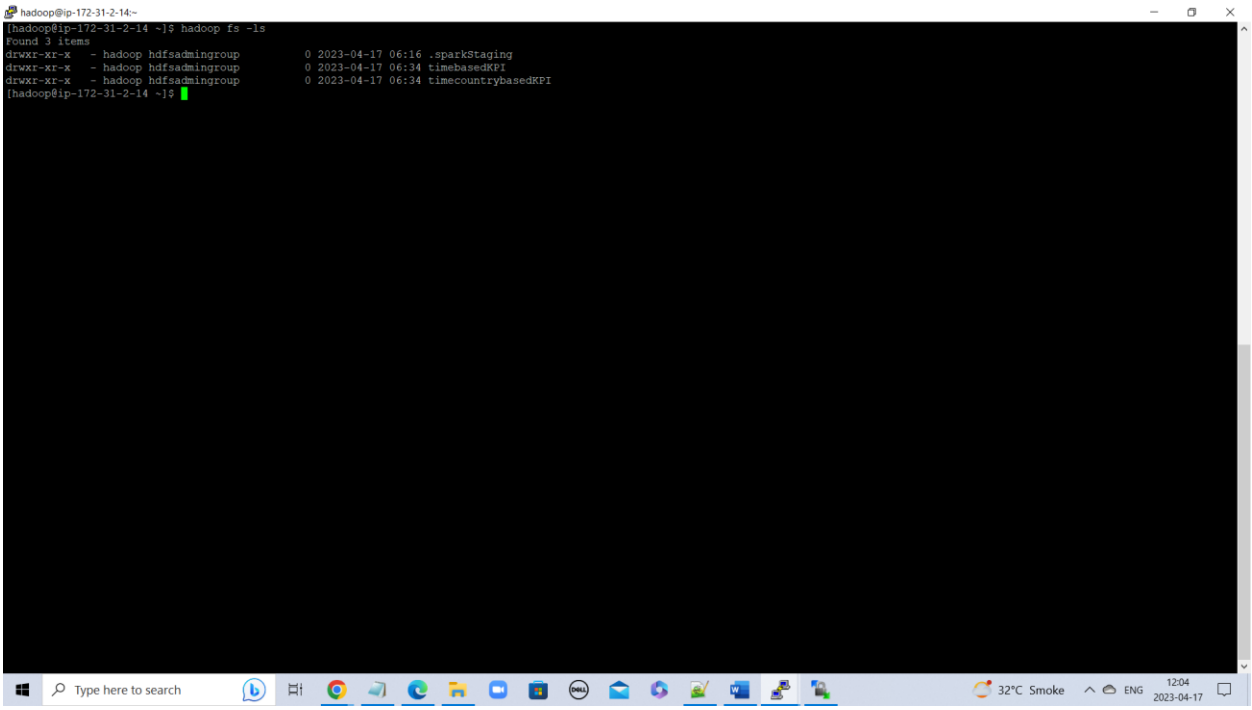
   export SPARK_KAFKA_VERSION=0.10 (use this export command to setup the environment before executing the below command)

   spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5 spark-streaming.py > Console-output

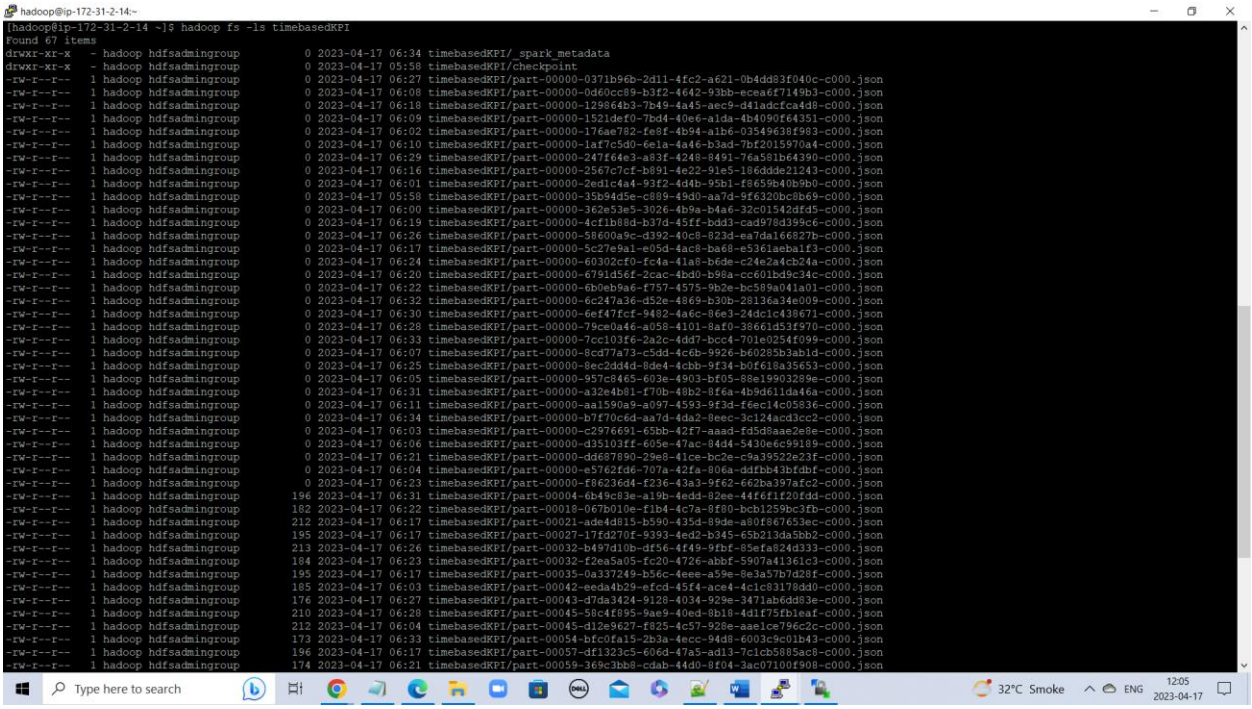


2. Validate the JSON files generated

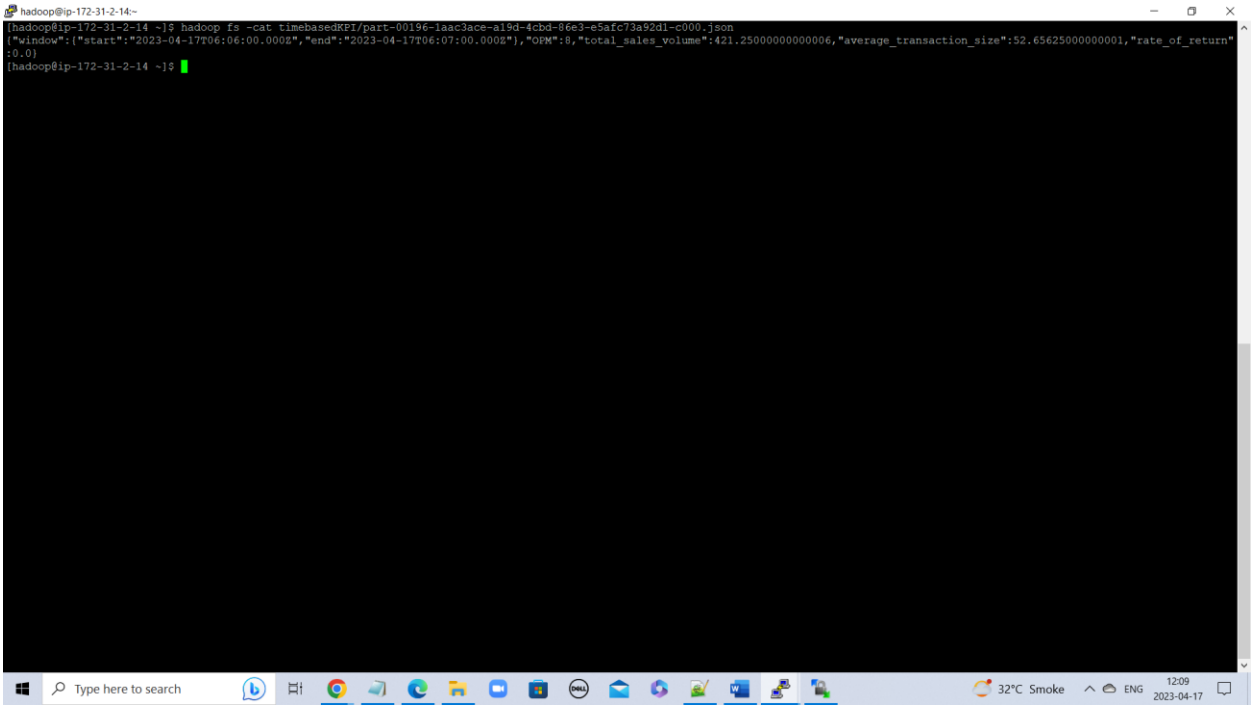
hadoop fs -ls



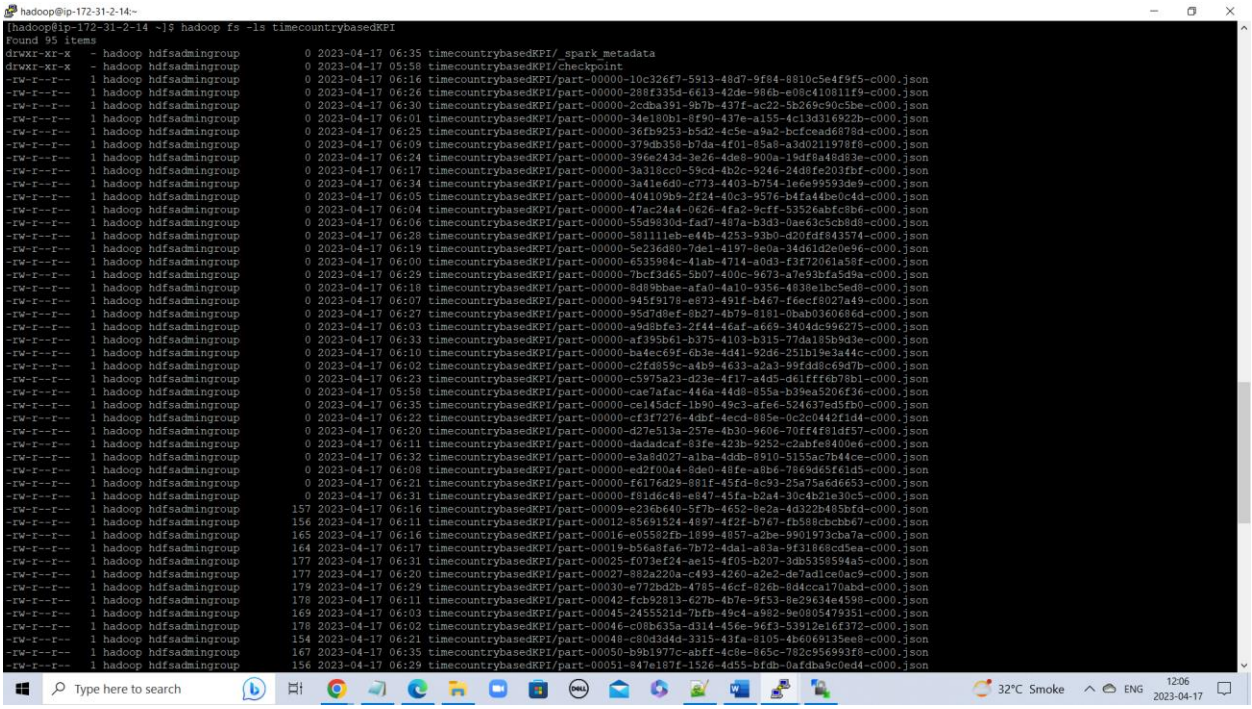
hadoop fs -ls timebasedKPI



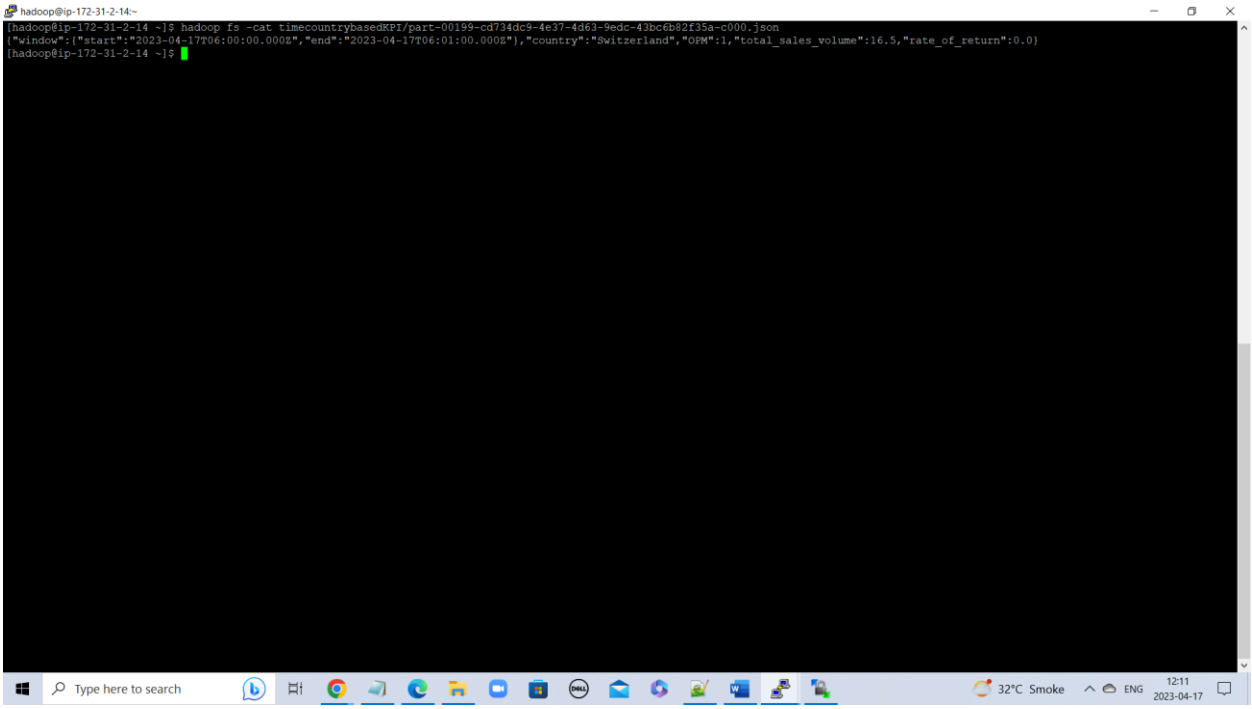
hadoop fs -cat timebasedKPI/part-00196-1aac3ace-a19d-4cbd-86e3-e5afc73a92d1-c000.json



hadoop fs -ls timecountrybasedKPI



hadoop fs -cat timecountrybasedKPI/part-00199-cd734dc9-4e37-4d63-9edc-43bc6b82f35a-c000.json



3. Transfer of files generated from HDFS to Local system