Code Logic - Retail Data Analysis

In this project we have done the real-time data analysis of the retail data coming from the Kafka broker using the Spark Streaming. The below details for Kafka broker were provided.

- **Bootstrap Server:** ##.###.###.###
- **Port:** ####
- **Topic:** real-time-project

We have written a Python script (**spark-streaming.py**) which includes reading the raw data coming from Kafka broker, writing the summarised input tables to the console, calculating the time-based and country-and-time-based KPIs and then finally writing the KPIs to the JSON files.

- **Initializing Spark session**

  We have initialized the spark session using the **.getOrCreate()** command and set the log level to **ERROR**.

- **Reading the raw data from Kafka broker**

  We have read the raw data from Kafka broker using the **.readStream** by passing the above mentioned details for Kafka broker, format as **“kafka”** and loaded the data using the **.load()** command.

- **Defining the schema**

  We then defined the schema for a single order using the **StructType()** command. Added the required columns using **.add()** command by passing the column names and the data type for the column. For the **“items”** column which is of **ArrayType()**, we have defined the attributes inside that using the **StructField()** command.

- **Writing the utility functions**

  We have then written the utility functions for calculating the total number of items, total cost for single transaction and for determining if the type of transaction is ORDER or RETURN.

- **Defining the UDFs**

We then defined the UDFs (User-Defined Functions) for the utility functions using the **udf()** function.

- **Writing the summarised table values to the console**

We then added the columns created using the UDFs to our stream using the **.withColumn()** command.
We then wrote the summarised table values to the console using the **.writeStream** command with output mode as **“append”** and format as **“console”** with processing time of **“1 minute”**.

- **Calculating time-based and country-and-time-based KPIs**

We then calculated the time-based and country-and-time-based KPIs by adding the watermark on “timestamp” column with 1 minute latency allowance. A tumbling window of 1 minute was added and then aggregation functions were applied on the required columns for calculating the KPIs. For **country-and-time-based KPIs groupBy** function was applied on the **“country”** column.

- **Writing the KPIs to the JSON files**

Finally, the calculated KPIs were written to the JSON files using the **.writeStream** command with format as **“json”** and output mode as **“append”** with processing time of **“1 minute”**. Also, specified the path to store the JSON files for time-based KPIs and country-and-time-based KPIs.

- **Adding the required jar files**

After setting up the EMR cluster, required jar file was downloaded using the following command.

wget https://ds-spark-sql-kafka-jar.s3.amazonaws.com/spark-sql-kafka-0-10_2.11-2.3.0.jar

- **Spark-submit command**

Python script (**spark-streaming.py**) was copied to the EMR cluster using WinSCP tool. Then the python script (**spark-streaming.py**) was executed using the **spark-submit** command. **Spark packages**, **python script name**, **Kafka server**, **port** and **Kafka topic name** are passed as the parameters in the **spark-submit** command. Below is the spark-submit command used.

- **Downloading the JSON files**

After running the spark-submit command, JSON files are created at the specified locations. First, the JSON files are copied from the specified hadoop location to the local user location using the following commands.

```
hadoop fs -get /tmp/KPIsByTime/
```

```
hadoop fs -get /tmp/KPIsByTimeCountry/
```

Then, the JSON files are downloaded from the EMR cluster to our local machine using WinSCP tool.