Code Logic - Retail Data Analysis

Spark Streaming Code Implementation:

To get started, import the essential **PySpark** modules including the '**SparkSession**' class, along with all the functions and data types from the '**pyspark.sql.functions**' and '**pyspark.sql.types**' modules, respectively.

```
# Importing necessary libraries
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark.sql.window import Window

```

Create a **SparkSession**, which serves as the entry point for interacting with a Spark application.

```
# Creating a spark session
spark = SparkSession \
        .builder \
        .appName("RetailDataAnalysis") \
        .getOrCreate()

# Setting log level to ERROR
spark.sparkContext.setLogLevel('ERROR')
```

Connect to the **Kafka server** and read the retail data in **JSON** format. Define the appropriate schema or data structure to convert the raw data into a structured DataFrame.

```
# Reading Retail Dataset from Kafka topic : real-time-project
retail_data_raw = spark  \
            .readStream  \
            .format("kafka")  \
            .option("kafka.bootstrap.servers","18.211.252.152:9092")  \
            .option("subscribe","real-time-project")  \
            .load()

# Defining the Data Structure for source retail data (json format).
retail_data_JSON = StructType([
                        StructField("invoice_no", LongType()),
                        StructField("country", StringType()),
                        StructField("timestamp", TimestampType()),
                        StructField("type", StringType()),
                        StructField("items", ArrayType(StructType([
                                                StructField("SKU", StringType()),
                                                StructField("title", StringType()),
                                                StructField("unit_price", FloatType()),
                                                StructField("quantity", IntegerType())
                                            ])))
                    ])

# Converting source data from json format to dataframe (table).
retail_data_formatted =  retail_data_raw \
                .select(from_json(col("value").cast("string"), retail_data_JSON).alias("retail_data")) \
                .select("retail_data.*")
```

Define the following **Utility Functions** to calculate key metrics and handle order types:

1. **order_checker:** This function takes **type** as an argument and maps the type of order. If the category is **ORDER**, it returns 1. If the category is **RETURN**, it returns 0.

```
# Checking whether new order
def order_checker(type):
    '''
        To check whether the type is Order or not
            1 - ORDER Type
            0 - Not an Order Type
     '''
    return 1 if type == 'ORDER' else 0
```

2. **return_checker:** This function takes **type** as an argument and maps the type of return. If the category is **ORDER**, it returns 0. If the category is **RETURN**, it returns 1.

```
# Checking whether return order
def return_checker(type):
    '''
        To check whether the type is Return or not
            1 - Return Type
            0 - Not a Return Type
     '''
    return 1 if type == 'RETURN' else 0
```

3. **net_item_count:** This function takes **items** as an argument and retrieves the total number of items by summing up the quantities of all products.

```
# Calculating Total Item Count in an order
def net_item_count(items):
    '''
        Calculating total no. of items present in each invoice (quantity).
    '''
    if items is not None:
        item_count =0
        for item in items:
            item_count = item_count + item['quantity']
        return item_count
```

4. **net_cost:** This function takes **items and type** as an argument and calculates the net cost of an order by multiplying the quantity of each item by its price. The cost is positive for normal orders and negative for returns.

```
# Calculating Total Cost of an order
def net_cost(items,type):
    '''
        Calculating the total cost associated with each invoice.
        'total_cost' is the sum of '(unit_price * quantity)' for all items.
        If the invoice type is "Return," the total cost will be negative, representing the amount lost.
    '''
    if items is not None:
        total_cost =0
        item_price =0
    for item in items:
        item_price = (item['quantity']*item['unit_price'])
        total_cost = total_cost+ item_price
        item_price=0

    if type  == 'RETURN':
        return total_cost *-1
    else:
        return total_cost
```

Transform the **Utility Functions** into **User Defined Functions (UDFs)**, allowing them to be utilized in subsequent transformations to derive the desired metrics.

```
# Converting utility functions to UDFs
is_order = udf(order_checker, IntegerType())
is_return = udf(return_checker, IntegerType())
add_net_item_count = udf(net_item_count, IntegerType())
add_net_cost = udf(net_cost, FloatType())
```

Utilize the UDFs to generate four new columns: '**total_cost**', '**total_items**', '**is_order**', and '**is_return**', and store them in the '**retail_data_transformed**' dataframe.

```
# Extracting the columns/metrics: total_cost, total_items, is_order, and is_return from the source dataframe
retail_data_transformed = retail_data_formatted \
                          .withColumn("total_cost", add_net_cost(retail_data_formatted.items,retail_data_formatted.type)) \
                          .withColumn("total_items", add_net_item_count(retail_data_formatted.items)) \
                          .withColumn("is_order", is_order(retail_data_formatted.type)) \
                          .withColumn("is_return", is_return(retail_data_formatted.type))
```

Output the summarized input values to the console using the 'append' output method, with truncation set to false and a processing time of 1 minute.

```
# Selecting the required columns and writing the retail transformed data to console
retail_data_sink = retail_data_transformed \
                   .select("invoice_no", "country", "timestamp","total_cost","total_items","is_order","is_return") \
                   .writeStream.outputMode("append").format("console").option("truncate", "false") \
                   .option("path", "/Console_output").option("checkpointLocation", "/Console_output_checkpoints") \
                   .trigger(processingTime="1 minute") \
                   .start()
```

Compute **time-based Key Performance Indicators (KPIs)** such as **Operating Profit Margin (OPM), Total Sale Volume, Average Transaction Size, and Rate of Return** using a tumbling window of one minute and a watermark of one minute.

```
# Calculating time based KPIs: OPM (Orders Per Minute), total_sale_volume, average_transaction_size, rate_of_return.
retail_time_kpi = retail_data_transformed \
                  .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("total_cost").alias("average_transaction_size"),
                       avg("is_return").alias("rate_of_return")) \
                  .select("window",
                          "OPM",
                          "total_sale_volume",
                          "average_transaction_size",
                          "rate_of_return")

```

Write the **time-based KPIs** data to **HDFS** as **JSON** files for each one-minute window, using the 'append' output mode, with truncation set to false. Specify the HDFS output path for both the KPI files and their checkpoints.

```
# Writing retail data time based KPI in HDFS in JSON format
retail_time_kpi_sink = retail_time_kpi \
                       .writeStream \
                       .format("json") \
                       .outputMode("append") \
                       .option("truncate", "false") \
                       .option("path", "Timebased-KPI") \
                       .option("checkpointLocation", "Timebased-Checkpoint") \
                       .trigger(processingTime="1 minute") \
                       .start()
```

Compute **time-and-country-based Key Performance Indicators (KPIs**) such as **Operating Profit Margin (OPM), Total Sale Volume, and Rate of Return** using a tumbling window of one minute and a watermark of one minute. Group the data by both window and country.

```
# Calculating time-country based KPIs: OPM (Orders Per Minute), total_sale_volume, rate_of_return.
retail_time_country_kpi = retail_data_transformed \
                  .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_return").alias("rate_of_return")) \
                  .select("window",
                          "country",
                          "OPM",
                          "total_sale_volume",
                          "rate_of_return")
```

Output the **time-and-country-based KPIs** data to **HDFS** as **JSON** files for each one-minute window, using the 'append' output mode, with truncation set to false. Specify the HDFS output path for both the KPI files and their checkpoints.

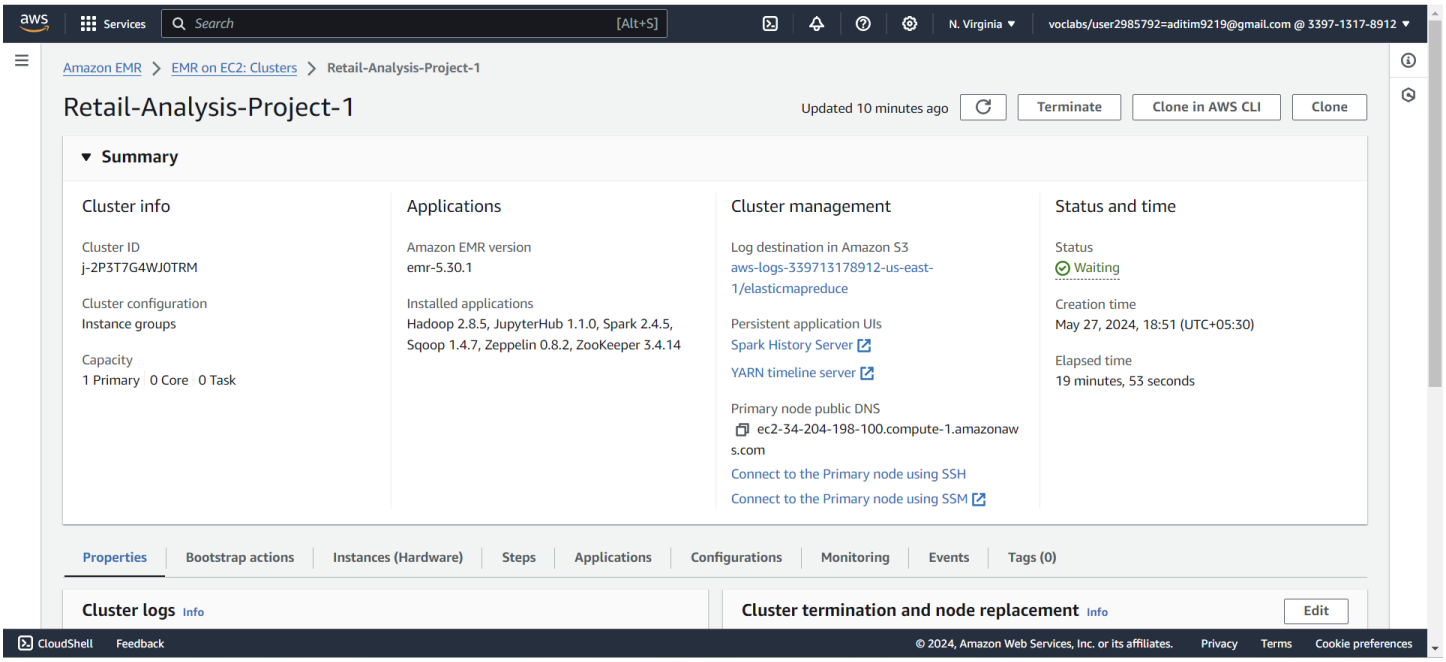
```
155
156  # Writing retail data time and country based KPI in HDFS in JSON format
157  retail_time_country_kpi_sink = retail_time_country_kpi \
158                                 .writeStream \
159                                 .format("json") \
160                                 .outputMode("append") \
161                                 .option("truncate", "false") \
162                                 .option("path", "Country-and-timebased-KPI") \
163                                 .option("checkpointLocation", "Country-and-timebased-Checkpoint") \
164                                 .trigger(processingTime="1 minute") \
165                                 .start()
166
```

Instruct Spark to await termination.

```
166
167  # Executing the spark jobs for console outputs and storage
168  retail_data_sink.awaitTermination()
169  retail_time_kpi_sink.awaitTermination()
170  retail_time_country_kpi_sink.awaitTermination()
171
```

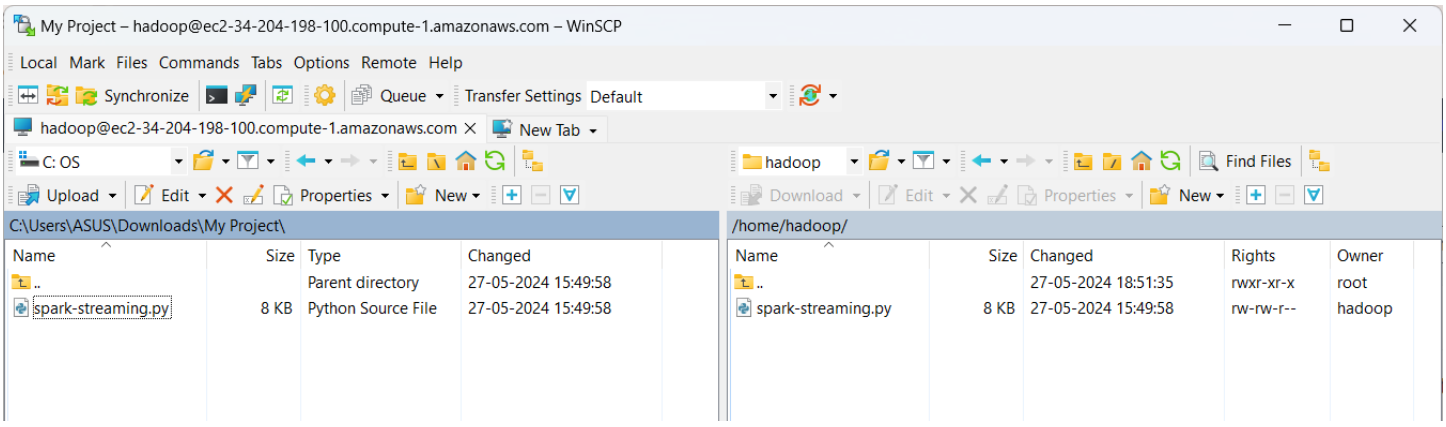
Code Deployment and Execution Steps:

Create an **EMR Cluster** with following configurations:



Initiated by logging into the **EMR Cluster** as the user ‘hadoop’.

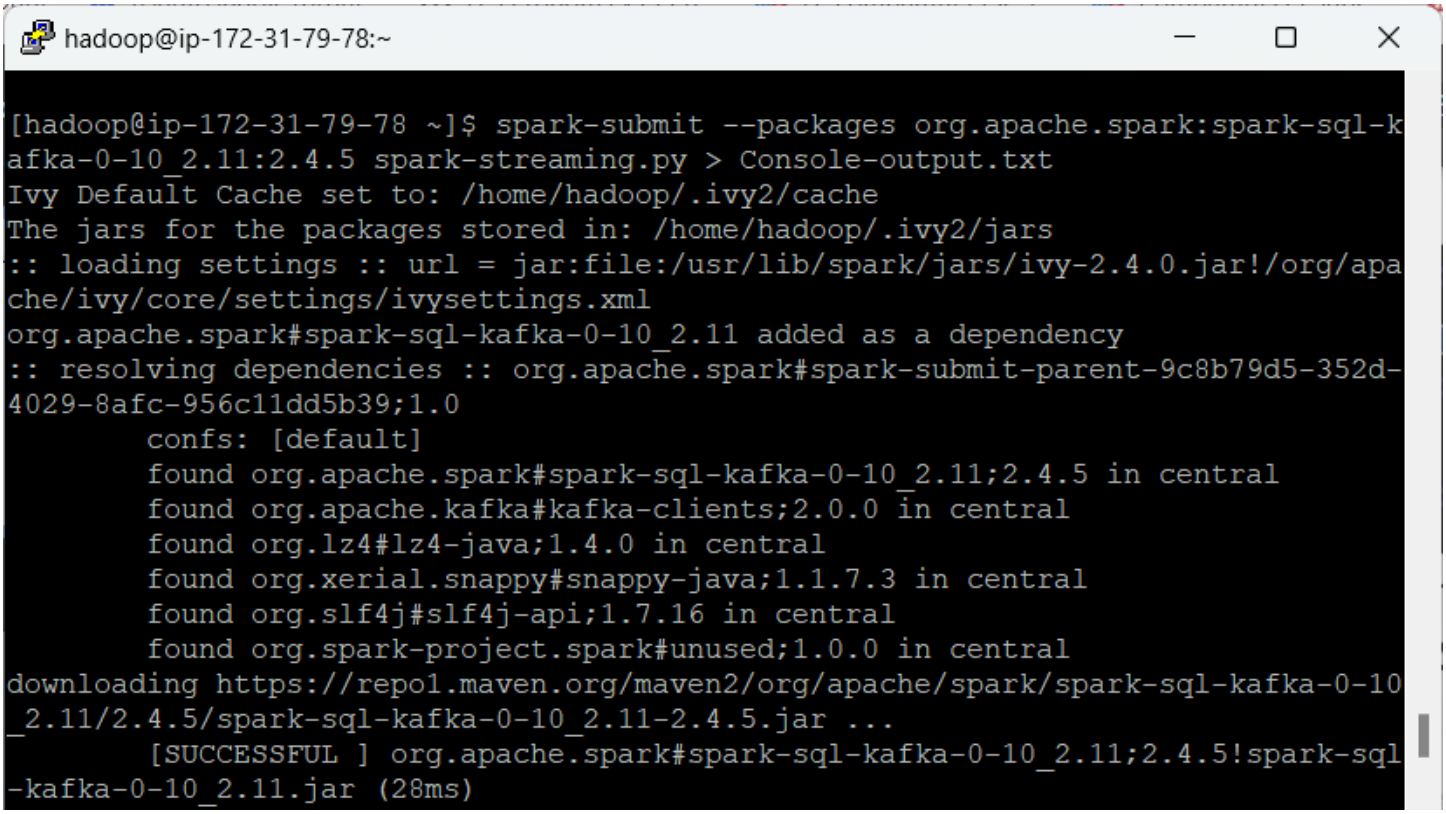
Transfer the Python file (spark-streaming.py) to the directory /home/hadoop using the WinSCP file transfer application.



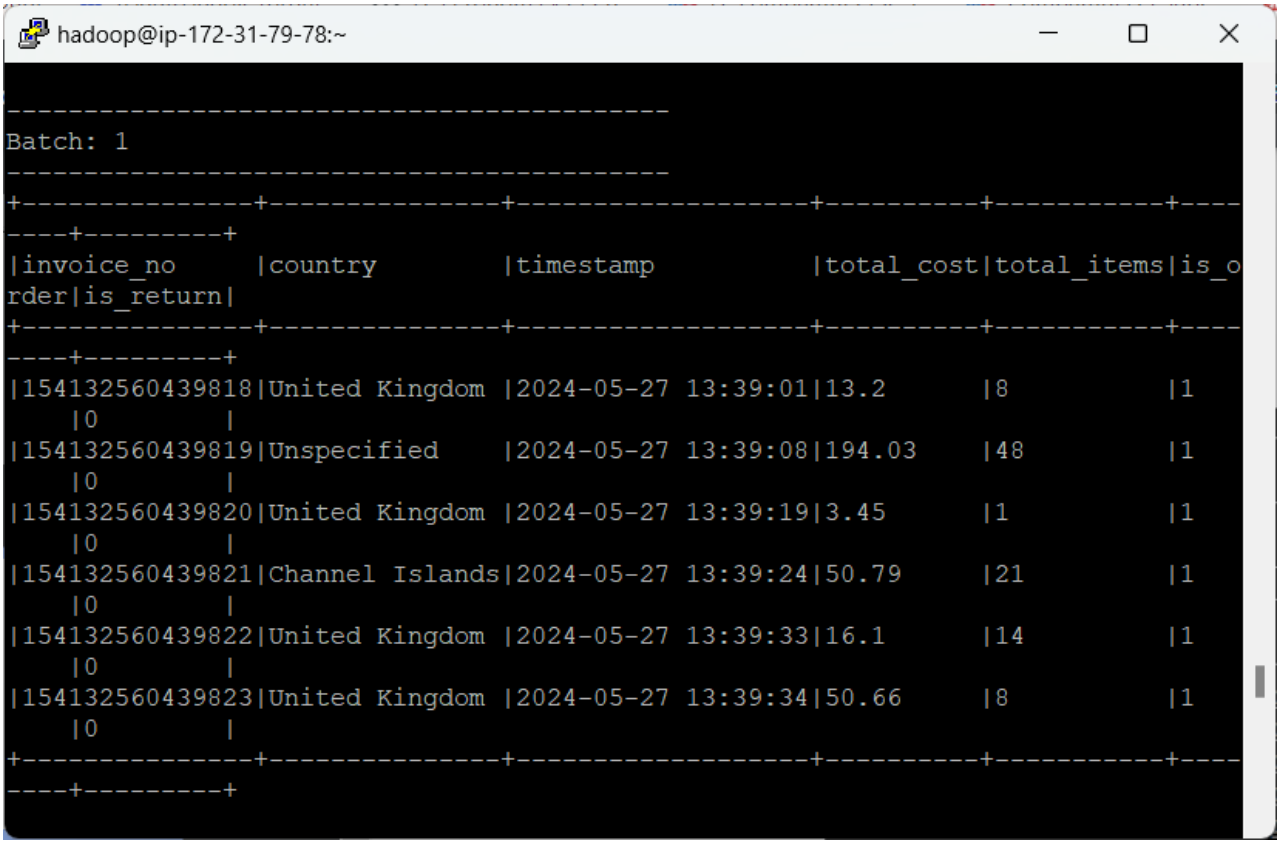

```
[hadoop@ip-172-31-79-78 ~]$ ls -ltr *.py
-rw-rw-r-- 1 hadoop hadoop 7614 May 27 10:19 spark-streaming.py
```

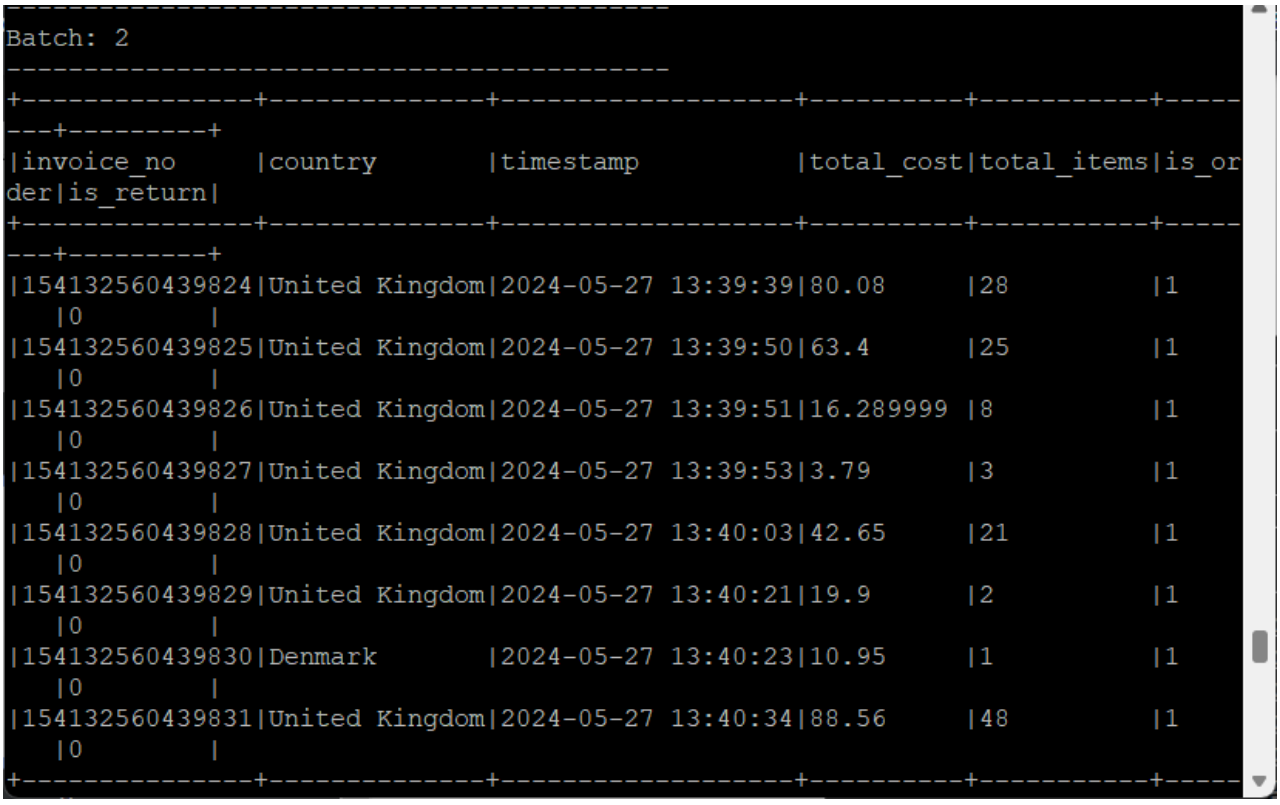
Execute the following command to enable Kafka integration with Apache Spark:
export SPARK_KAFKA_VERSION=0.10

Run the Python file using the spark-submit command, providing the Kafka jar package as an argument. Save the console output to a text file named **Console-output.txt**.



Read the console output file using the command: '**cat Console-output.txt**' and verify if the transformed data meets our requirements.





Now check whether all the JSON files are created for **Time based** and **Time-and-Country based KPIs** in **HDFS** (Path: /user/hadoop/).

```
hadoop@ip-172-31-79-78:~
KPI
[hadoop@ip-172-31-79-78 ~]$ hadoop fs -ls /user/hadoop
Found 5 items
drwxr-xr-x   - hadoop hadoop          0 2024-05-27 13:52 /user/hadoop/.sparkStaging
drwxr-xr-x   - hadoop hadoop          0 2024-05-27 13:35 /user/hadoop/Country-and-timebased-Checkpoint
drwxr-xr-x   - hadoop hadoop          0 2024-05-27 13:52 /user/hadoop/Country-and-timebased-KPI
drwxr-xr-x   - hadoop hadoop          0 2024-05-27 13:35 /user/hadoop/Timebased-Checkpoint
drwxr-xr-x   - hadoop hadoop          0 2024-05-27 13:52 /user/hadoop/Timebased-KPI
[hadoop@ip-172-31-79-78 ~]$
```

Read Timebased-KPI JSON files:

hadoop fs -ls /user/hadoop/Timebased-KPI

```
hadoop@ip-172-31-79-78:~
[hadoop@ip-172-31-79-78 ~]$ hadoop fs -ls /user/hadoop/Timebased-KPI
Found 32 items
drwxr-xr-x   - hadoop hadoop          0 2024-05-27 13:52 /user/hadoop/Timebased-KPI/_spark_metadata
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:35 /user/hadoop/Timebased-KPI/part-00000-15bb4305-7f2f-4d9c-936f-32353d32cda4-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:45 /user/hadoop/Timebased-KPI/part-00000-1cb4443a-def6-4c3d-bfc2-0040e1a49839-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:37 /user/hadoop/Timebased-KPI/part-00000-1f1d964b-eb14-4ca8-a488-6754e65540a1-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:39 /user/hadoop/Timebased-KPI/part-00000-3ffd9808-1e18-4c48-a1ff-bcccc3f3d72e-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:47 /user/hadoop/Timebased-KPI/part-00000-42b5cc07-8041-449c-b950-7e39b27f4e9a-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:43 /user/hadoop/Timebased-KPI/part-00000-4b7470e4-ecad-4587-8edb-cbd0519b06d4-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:40 /user/hadoop/Timebased-KPI/part-00000-51d41b8b-b762-4c7a-b799-0fddce798559-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:51 /user/hadoop/Timebased-KPI/part-00000-6772bde1-fb78-4613-82ec-01ebbda43c47-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:38 /user/hadoop/Timebased-KPI/part-00000-8413f36c-ed13-4479-8991-35f3cba97cfd-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:44 /user/hadoop/Timebased-KPI/part-00000-8f8be1ae-febb-49de-aed9-934d2a44ee4c-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:52 /user/hadoop/Timebased-KPI/part-00000-963b38c6-8f73-4145-a600-efd6d5008de7-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:49 /user/hadoop/Timebased-KPI/part-00000-9906832b-b442-43aa-b32a-b9399c0f2f74-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:41 /user/hadoop/Timebased-KPI/part-00000-9c4df0f0-d2f3-426d-a390-8314f75302db-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:42 /user/hadoop/Timebased-KPI/part-00000-a2f2ebd5-2f79-4c23-8289-278e6ce86a0b-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:50 /user/hadoop/Timebased-KPI/part-00000-b33c75a3-1c98-42bb-a471-9e5193e260f7-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:46 /user/hadoop/Timebased-KPI/part-00000-c44a94d7-74aa-4821-add2-4495f72c7dac-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:48 /user/hadoop/Timebased-KPI/part-00000-d0bc05c9-8742-4ccf-b919-51c0a63db94c-c000.json
-rw-r--r--   1 hadoop hadoop        194 2024-05-27 13:46 /user/hadoop/Timebased-KPI/part-00015-c0b65828-7a94-4faa-9c6e-b372d02b6bdd-c000.json
-rw-r--r--   1 hadoop hadoop        194 2024-05-27 13:51 /user/hadoop/Timebased-KPI/part-00023-699623b2-c2a2-498b-bdcd-768c7fc26c6e-c000.json
-rw-r--r--   1 hadoop hadoop        195 2024-05-27 13:40 /user/hadoop/Timebased-KPI/part-00027-16a040e7-f632-45f4-be45-bb4cb75c9d96-c000.json
-rw-r--r--   1 hadoop hadoop        194 2024-05-27 13:41 /user/hadoop/Timebased-KPI/part-00029-221d0aea-62ff-45a9-9f2a-874120787d6a-c000.json
-rw-r--r--   1 hadoop hadoop        193 2024-05-27 13:45 /user/hadoop/Timebased-KPI/part-00047-c62683f9-6e69-4d3c-8c6d-609e8242c9ab-c000.json
-rw-r--r--   1 hadoop hadoop        194 2024-05-27 13:43 /user/hadoop/Timebased-KPI/part-00049-5f46e649-d38f-438d-91fc-2a82905b2051-c000.json
-rw-r--r--   1 hadoop hadoop        192 2024-05-27 13:44 /user/hadoop/Timebased-KPI/part-00062-8746a69e-affd-4255-aa1f-8bf694d258aa-c000.json
-rw-r--r--   1 hadoop hadoop        194 2024-05-27 13:47 /user/hadoop/Timebased-KPI/part-00116-9531e147-72a4-4d55-bd16-76a8c09bc7de-c000.json
```

Use **hadoop fs -cat /user/hadoop/Timebased-KPI/part*** to read all the **JSON** files

```
hadoop@ip-172-31-79-78:~
[hadoop@ip-172-31-79-78 ~]$ hadoop fs -cat /user/hadoop/Timebased-KPI/part*
{"window":{"start":"2024-05-27T13:45:00.000Z","end":"2024-05-27T13:46:00.000Z"},"OPM":9,"total_sale_volume":956.0299909114838,"average_transaction_size":106.
22555454572041,"rate_of_return":0.0}
{"window":{"start":"2024-05-27T13:51:00.000Z","end":"2024-05-27T13:52:00.000Z"},"OPM":11,"total_sale_volume":403.4200038909912,"average_transaction_size":36.
67454580827193,"rate_of_return":0.0}
{"window":{"start":"2024-05-27T13:39:00.000Z","end":"2024-05-27T13:40:00.000Z"},"OPM":10,"total_sale_volume":491.79000210762024,"average_transaction_size":49
.17900021076203,"rate_of_return":0.0}
{"window":{"start":"2024-05-27T13:40:00.000Z","end":"2024-05-27T13:41:00.000Z"},"OPM":7,"total_sale_volume":724.6199884414673,"average_transaction_size":103.
51714120592389,"rate_of_return":0.0}
{"window":{"start":"2024-05-27T13:44:00.000Z","end":"2024-05-27T13:45:00.000Z"},"OPM":7,"total_sale_volume":196.1200076341629,"average_transaction_size":28.0
1714394773756,"rate_of_return":0.0}
{"window":{"start":"2024-05-27T13:42:00.000Z","end":"2024-05-27T13:43:00.000Z"},"OPM":12,"total_sale_volume":860.3599934577942,"average_transaction_size":71.
69666612148285,"rate_of_return":0.0}
{"window":{"start":"2024-05-27T13:43:00.000Z","end":"2024-05-27T13:44:00.000Z"},"OPM":8,"total_sale_volume":788.939998626709,"average_transaction_size":98.61
749982833862,"rate_of_return":0.0}
{"window":{"start":"2024-05-27T13:47:00.000Z","end":"2024-05-27T13:48:00.000Z"},"OPM":6,"total_sale_volume":379.64998638629913,"average_transaction_size":63.
27499773104986,"rate_of_return":0.0}
{"window":{"start":"2024-05-27T13:52:00.000Z","end":"2024-05-27T13:53:00.000Z"},"OPM":8,"total_sale_volume":334.5799980163574,"average_transaction_size":41.8
2249975204468,"rate_of_return":0.0}
{"window":{"start":"2024-05-27T13:46:00.000Z","end":"2024-05-27T13:47:00.000Z"},"OPM":10,"total_sale_volume":808.7799695730209,"average_transaction_size":80.
87799695730209,"rate_of_return":0.1}
{"window":{"start":"2024-05-27T13:49:00.000Z","end":"2024-05-27T13:50:00.000Z"},"OPM":3,"total_sale_volume":220.03999483585358,"average_transaction_size":73.
34666494528453,"rate_of_return":0.0}
{"window":{"start":"2024-05-27T13:50:00.000Z","end":"2024-05-27T13:51:00.000Z"},"OPM":12,"total_sale_volume":864.7299990653992,"average_transaction_size":72.
060833255449994,"rate_of_return":0.0}
{"window":{"start":"2024-05-27T13:41:00.000Z","end":"2024-05-27T13:42:00.000Z"},"OPM":3,"total_sale_volume":340.05999755859375,"average_transaction_size":113
.35333251953125,"rate_of_return":0.3333333333333333}
{"window":{"start":"2024-05-27T13:48:00.000Z","end":"2024-05-27T13:49:00.000Z"},"OPM":10,"total_sale_volume":793.249997138977,"average_transaction_size":79.3
2499971389771,"rate_of_return":0.0}
[hadoop@ip-172-31-79-78 ~]$
```

Read Country-and-timebased-KPI JSON files:

hadoop fs -ls /user/hadoop/Country-and-timebased-KPI

```
hadoop@ip-172-31-79-78:~
[hadoop@ip-172-31-79-78 ~]$ hadoop fs -ls /user/hadoop/Country-and-timebased-KPI
Found 47 items
drwxr-xr-x   - hadoop hadoop          0 2024-05-27 13:52 /user/hadoop/Country-and-timebased-KPI/_spark_metadata
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:49 /user/hadoop/Country-and-timebased-KPI/part-00000-00636caa-fcaf-49d7-acaf-7e04b42d6f3c-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:41 /user/hadoop/Country-and-timebased-KPI/part-00000-08be9d73-e72f-4841-a786-f2b00d27582f-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:38 /user/hadoop/Country-and-timebased-KPI/part-00000-107138dd-4c89-4a97-88c4-b81dfde4a365-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:51 /user/hadoop/Country-and-timebased-KPI/part-00000-152ea9fc-1e28-4b31-b4a7-563edf3ed730-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:52 /user/hadoop/Country-and-timebased-KPI/part-00000-2703efd7-d36d-4982-a341-2463945b0bb5-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:35 /user/hadoop/Country-and-timebased-KPI/part-00000-3a129574-ccf9-418b-aed0-f7e84896e060-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:48 /user/hadoop/Country-and-timebased-KPI/part-00000-678ee326-c1e9-4c57-a356-29a931e0da6d-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:39 /user/hadoop/Country-and-timebased-KPI/part-00000-6819bdbf-67c3-453c-85cf-f349ee6ab252-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:42 /user/hadoop/Country-and-timebased-KPI/part-00000-6dd366a4-6e11-41f1-88a6-24e0ac3d8d7a-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:37 /user/hadoop/Country-and-timebased-KPI/part-00000-6eaa48ca-b799-4dd5-a2fd-b8f05333da03-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:45 /user/hadoop/Country-and-timebased-KPI/part-00000-91eec1af-31b5-4558-8564-d8bd259d42ad-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:46 /user/hadoop/Country-and-timebased-KPI/part-00000-ae30681d-983c-47da-b9e0-5257a13e73fe-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:47 /user/hadoop/Country-and-timebased-KPI/part-00000-bbe2d733-5130-4f31-b856-26f4fb3f1fc2-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:50 /user/hadoop/Country-and-timebased-KPI/part-00000-c3106616-0d9b-4726-bae3-3489104f7933-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:40 /user/hadoop/Country-and-timebased-KPI/part-00000-d2c20408-9683-4dc9-9b20-5fb840ae33ba-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:44 /user/hadoop/Country-and-timebased-KPI/part-00000-ef00acbe-7bbf-4f98-a1e7-5cfe4703b5e8-c000.json
-rw-r--r--   1 hadoop hadoop          0 2024-05-27 13:43 /user/hadoop/Country-and-timebased-KPI/part-00000-f314214c-9f0b-4f49-b60d-c92f63edc598-c000.json
-rw-r--r--   1 hadoop hadoop        165 2024-05-27 13:49 /user/hadoop/Country-and-timebased-KPI/part-00007-37e797f2-ecac-4b94-a361-a8d02ec5998f-c000.json
-rw-r--r--   1 hadoop hadoop        176 2024-05-27 13:49 /user/hadoop/Country-and-timebased-KPI/part-00012-c4f0d08e-2901-41be-92c2-188f43db6e11-c000.json
-rw-r--r--   1 hadoop hadoop        165 2024-05-27 13:47 /user/hadoop/Country-and-timebased-KPI/part-00013-3d4d1b70-eae8-4d38-8838-ed0de5606ef8-c000.json
-rw-r--r--   1 hadoop hadoop        168 2024-05-27 13:49 /user/hadoop/Country-and-timebased-KPI/part-00020-4d6ef1b1-f40e-4324-846c-adcfcd0d7a13-c000.json
-rw-r--r--   1 hadoop hadoop        175 2024-05-27 13:52 /user/hadoop/Country-and-timebased-KPI/part-00046-b06de176-bc59-480d-a193-86052804d233-c000.json
-rw-r--r--   1 hadoop hadoop        169 2024-05-27 13:41 /user/hadoop/Country-and-timebased-KPI/part-00050-51d9ed1a-8613-48f0-9575-75d4c0352a7c-c000.json
-rw-r--r--   1 hadoop hadoop        177 2024-05-27 13:40 /user/hadoop/Country-and-timebased-KPI/part-00086-8d47e538-b24f-40a8-b49b-3b9043da0a36-c000.json
```

Use **hadoop fs -cat /user/hadoop/Country-and-timebased-KPI/part*** to read all the **JSON** files

```
hadoop@ip-172-31-79-78:~
[hadoop@ip-172-31-79-78 ~]$ hadoop fs -cat /user/hadoop/Country-and-timebased-KPI/part*
{"window":{"start":"2024-05-27T13:48:00.000Z","end":"2024-05-27T13:49:00.000Z"},"country":"EIRE","OPM":1,"total_sale_volume":87.62000274658203,"rate_of_retur
n":0.0}
{"window":{"start":"2024-05-27T13:48:00.000Z","end":"2024-05-27T13:49:00.000Z"},"country":"United Kingdom","OPM":7,"total_sale_volume":505.96999073028564,"ra
te_of_return":0.0}
{"window":{"start":"2024-05-27T13:46:00.000Z","end":"2024-05-27T13:47:00.000Z"},"country":"EIRE","OPM":1,"total_sale_volume":1.649999976158142,"rate_of_retur
n":0.0}
{"window":{"start":"2024-05-27T13:48:00.000Z","end":"2024-05-27T13:49:00.000Z"},"country":"Belgium","OPM":1,"total_sale_volume":83.87000274658203,"rate_of_re
turn":0.0}
{"window":{"start":"2024-05-27T13:52:00.000Z","end":"2024-05-27T13:53:00.000Z"},"country":"United Kingdom","OPM":8,"total_sale_volume":334.5799980163574,"rat
e_of_return":0.0}
{"window":{"start":"2024-05-27T13:40:00.000Z","end":"2024-05-27T13:41:00.000Z"},"country":"Denmark","OPM":1,"total_sale_volume":10.949999809265137,"rate_of_r
eturn":0.0}
{"window":{"start":"2024-05-27T13:39:00.000Z","end":"2024-05-27T13:40:00.000Z"},"country":"Channel Islands","OPM":1,"total_sale_volume":50.790000915527344,"r
ate_of_return":0.0}
{"window":{"start":"2024-05-27T13:39:00.000Z","end":"2024-05-27T13:40:00.000Z"},"country":"United Kingdom","OPM":8,"total_sale_volume":246.97000241279602,"ra
te_of_return":0.0}
{"window":{"start":"2024-05-27T13:41:00.000Z","end":"2024-05-27T13:42:00.000Z"},"country":"United Kingdom","OPM":3,"total_sale_volume":340.05999755859375,"ra
te_of_return":0.3333333333333333}
{"window":{"start":"2024-05-27T13:43:00.000Z","end":"2024-05-27T13:44:00.000Z"},"country":"United Kingdom","OPM":6,"total_sale_volume":644.2399978637695,"rat
e_of_return":0.0}
{"window":{"start":"2024-05-27T13:44:00.000Z","end":"2024-05-27T13:45:00.000Z"},"country":"United Kingdom","OPM":7,"total_sale_volume":196.1200076341629,"rat
e_of_return":0.0}
{"window":{"start":"2024-05-27T13:42:00.000Z","end":"2024-05-27T13:43:00.000Z"},"country":"Sweden","OPM":1,"total_sale_volume":127.31999969482422,"rate_of_re
turn":0.0}
{"window":{"start":"2024-05-27T13:51:00.000Z","end":"2024-05-27T13:52:00.000Z"},"country":"France","OPM":1,"total_sale_volume":10.079999923706055,"rate_of_re
turn":0.0}
```

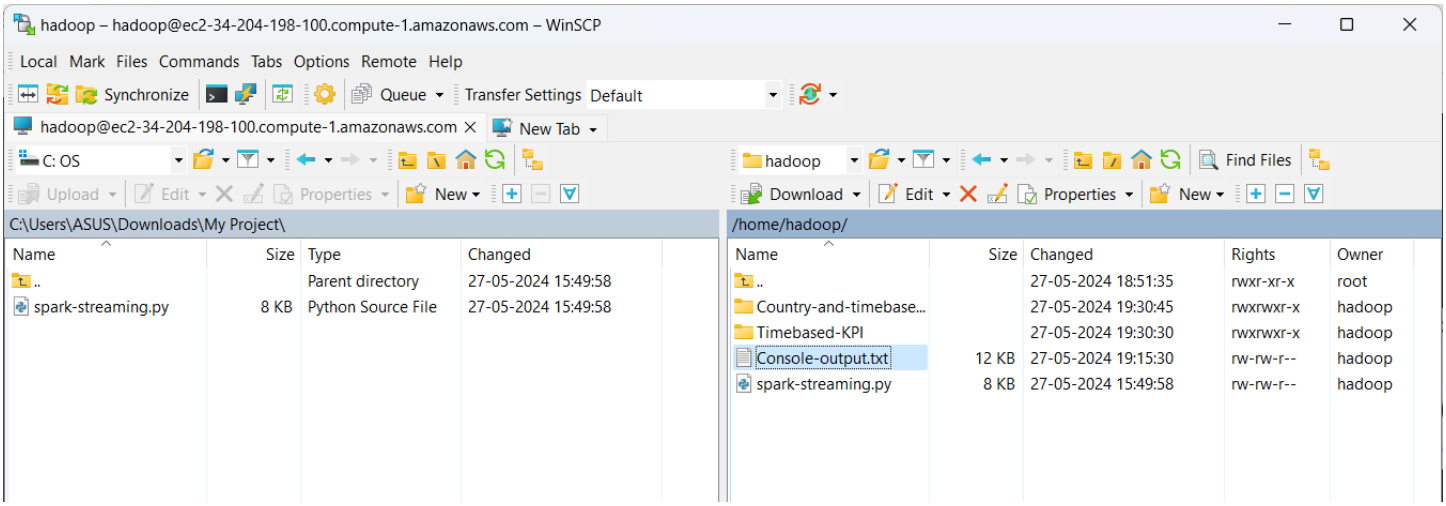
Copy the directories for Time-based and Country-and-Time-based KPIs from HDFS to the local path (/home/hadoop) on the EMR instance using the following commands:

hadoop fs -get /user/hadoop/Timebased-KPI /home/hadoop/

hadoop fs -get /user/hadoop/Country-and-timebased-KPI /home/hadoop/

```
hadoop@ip-172-31-79-78:~
[hadoop@ip-172-31-79-78 ~]$ hadoop fs -get /user/hadoop/Timebased-KPI /home/hadoop/
[hadoop@ip-172-31-79-78 ~]$ hadoop fs -get /user/hadoop/Country-and-timebased-KPI /home/hadoop/
[hadoop@ip-172-31-79-78 ~]$ ls -ltr
total 32
-rw-rw-r-- 1 hadoop hadoop  7614 May 27 10:19 spark-streaming.py
-rw-rw-r-- 1 hadoop hadoop 11922 May 27 13:45 Console-output.txt
drwxrwxr-x 3 hadoop hadoop  4096 May 27 14:00 Timebased-KPI
drwxrwxr-x 3 hadoop hadoop  4096 May 27 14:00 Country-and-timebased-KPI
[hadoop@ip-172-31-79-78 ~]$
```

Transfer all the console and JSON output files from the EMR instance to the local machine using WinSCP.



After completing all the steps, terminate the EMR instance from the AWS console.