Code Logic - Retail Data Analysis

Setup:

1. Setting up spark environment before running the spark job

export SPARK\_KAFKA\_VERSION=0.10

1. Run the spark job to execute spark-streaming.py script

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

org.apache.spark:spark-sql-kafka-0-10\_2.11:2.4.5 - Spark Kafka Sql Jar version spark-streaming.py - Script to be executed

Code Logic:

spark-streaming.py script executes sequentially with following steps:

➔ Import the necessary modules

from pyspark.sql import SparkSession

from pyspark.sql.functions import \*

from pyspark.sql.functions import \*

➔ Initialising Spark Session

spark = SparkSession \

.builder \

.appName("Retail\_Project") \

.getOrCreate()

➔ Setting the log level to be ERROR

spark.sparkContext.setLogLevel('ERROR')

➔ Reading data from Kafka

raw\_data = spark \

.readStream \

.format("kafka") \

.option("kafka.bootstrap.servers","18.211.252.152:9092") \

.option("subscribe","real-time-project") \

.option("startingOffsets", "latest") \

.load()

**Format** - Kafka

**Bootstrap Server** - 18.211.252.152:9092

**Subscribe to Topic** - real-time-project

**Starting Offsets** – Earliest

➔ Construct schema in a format provided as a input:

Input

{

"invoice\_no": 154132541653705,

"country": "United Kingdom",

"timestamp": "2020-09-18 10:55:23",

"type": "ORDER",

"items": [

{ "SKU": "21485",

"title": "RETROSPOT HEART HOT WATER BOTTLE",

"unit\_price": 4.95,

"quantity": 6

},

{ "SKU": "23499",

"title": "SET 12 VINTAGE DOILY CHALK",

"unit\_price": 0.42,

"quantity": 2

}

]

}

**Schema**

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", FloatType()),

StructField("quantity", IntegerType())

])))

➔ Read the input data with respect to schema provided

order\_stream = raw\_data.select(from\_json(col("value").cast("string"), schema).alias("data")).select("data.\*")

➔ Add the following utility functions

is\_order - A user defined function to return whether it’s type is ORDER or not

def is\_order(type):

if type=="ORDER":

return 1

else:

return 0

is\_return - A user defined function to return whether it’s type is RETURN or not

def is\_return(type):

if type=="RETURN":

return 1

else:

return 0

total\_items\_count - A user defined function to return the total count of items ordered.

def total\_items\_count(items):

total\_count = 0

for item in items:

total\_count = total\_count + item['quantity']

return total\_count

calculate\_total\_cost - A user defined function to return the total cost of items ordered.

def calculate\_total\_cost(items,type):

total\_price = 0

for item in items:

total\_price = total\_price + item['unit\_price'] \* item['quantity']

if type=="RETURN":

return total\_price \* -1

else:

return total\_price

# Define the UDFs with the utility functions

flag\_order = udf(is\_order, IntegerType()) f

lag\_return = udf(is\_return, IntegerType())

calculate\_item = udf(total\_items\_count, IntegerType())

calculate\_cost = udf(calculate\_total\_cost, FloatType())

➔ Append additional columns to the order\_stream with the user defined functions

order\_stream\_extended = order\_stream \

.withColumn("total\_items", calculate\_item(order\_stream.items)) \ .withColumn("total\_cost",calculate\_cost(order\_stream.items,order\_stream.type))\ .withColumn("is\_order", flag\_order(order\_stream.type)) \

.withColumn("is\_return", flag\_return(order\_stream.type))

➔ Select columns required to be logged as console. Every 1 minute the new data gets processed in a stream.

order\_table\_console = order\_stream\_extended \

.select("invoice\_no", "country", "timestamp","type","total\_items","total\_cost","is\_order","is\_return") \

.writeStream \

.outputMode("append") \

.format("console") \

.option("truncate", "false") \

.trigger(processingTime="1 minute") \

.start()

➔ Calculate Time and Country based KPIs

# Calculate time based KPIs

agg\_time = order\_stream\_extended \

.withWatermark("timestamp","1 minute") \

.groupby(window("timestamp", "1 minute",”1 minute”)) \ .agg(sum("total\_cost").alias("total\_volume\_of\_sales"), avg("total\_cost").alias("average\_transaction\_size"), avg("is\_return").alias("rate\_of\_return")) \ .select("window.start","window.end","total\_volume\_of\_sales","average\_transaction\_size", "rate\_of\_return")

# Calculate time and country based KPIs

agg\_time\_country = order\_stream\_extended \

.withWatermark("timestamp", "1 minute") \

.groupBy(window("timestamp", "1 minute",”1 minute”), "country") \ .agg(sum("total\_cost").alias("total\_volume\_of\_sales"), count("invoice\_no").alias("OPM"), avg("is\_return").alias("rate\_of\_return")) \ .select("window.start","window.end","country", "OPM","total\_volume\_of\_sales","rate\_of\_return")

➔ Logging Time and Country based KPIs

# Write time based KPI values

console\_by\_time = agg\_time.writeStream \

.format("json") \

.outputMode("append") \

.option("truncate", "false") \

.option("path", "time\_based\_kpi/") \

.option("checkpointLocation", "time\_based\_kpi/cp/") \ .trigger(processingTime="1 minute") \

.start()

# Write time and country based KPI values

console\_by\_country = agg\_time\_country.writeStream \

.format("json") \

.outputMode("append") \

.option("truncate", "false") \

.option("path", "time\_country\_based\_kpi/") \

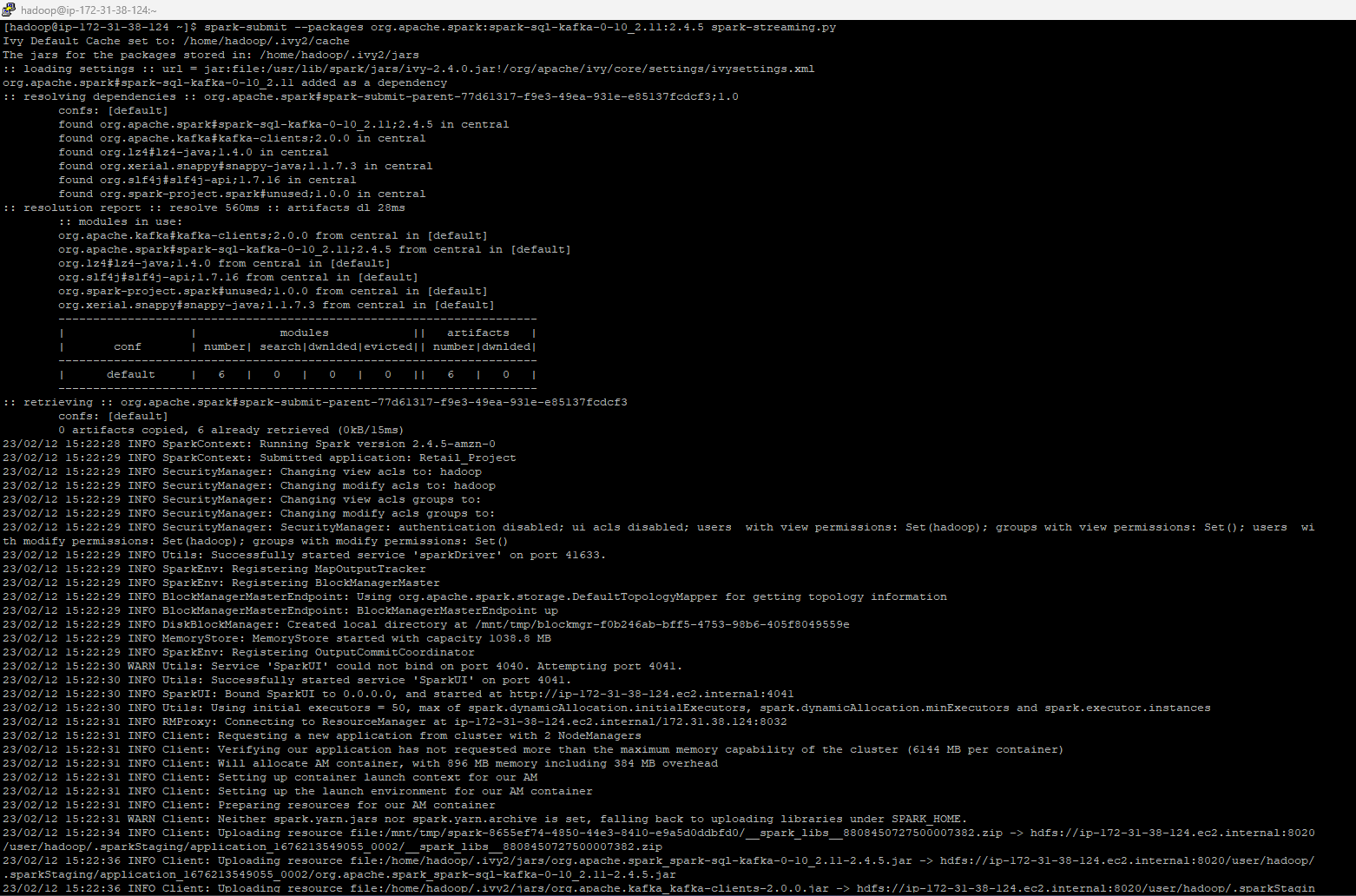
.option("checkpointLocation", "time\_country\_based\_kpi/cp/") \ .trigger(processingTime="1 minute") \

.start()

➔ Await Termination execution for writestream queries order\_table\_console.awaitTermination()

console\_by\_time.awaitTermination()

console\_by\_country.awaitTermination()



Text

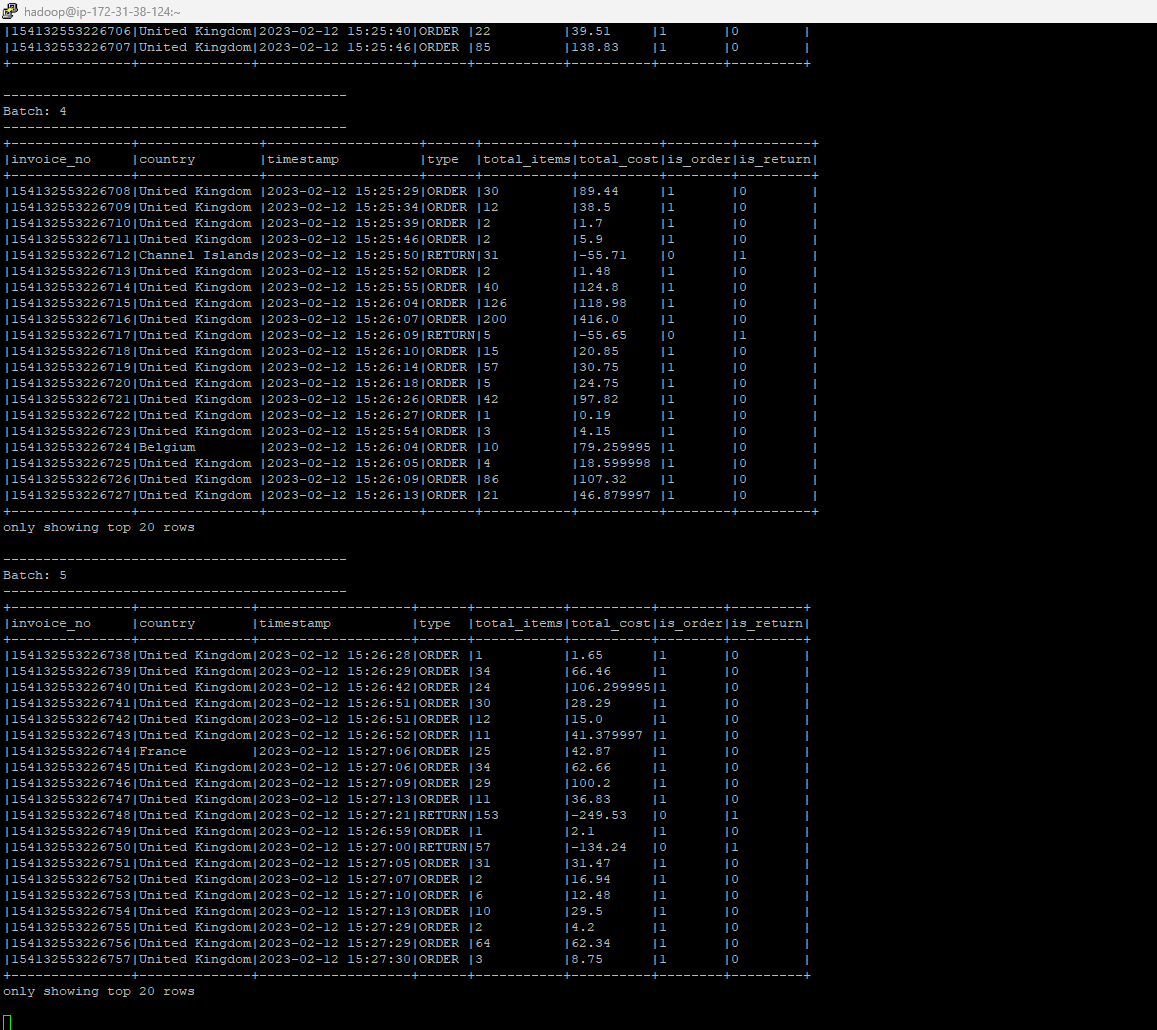
Description automatically generated

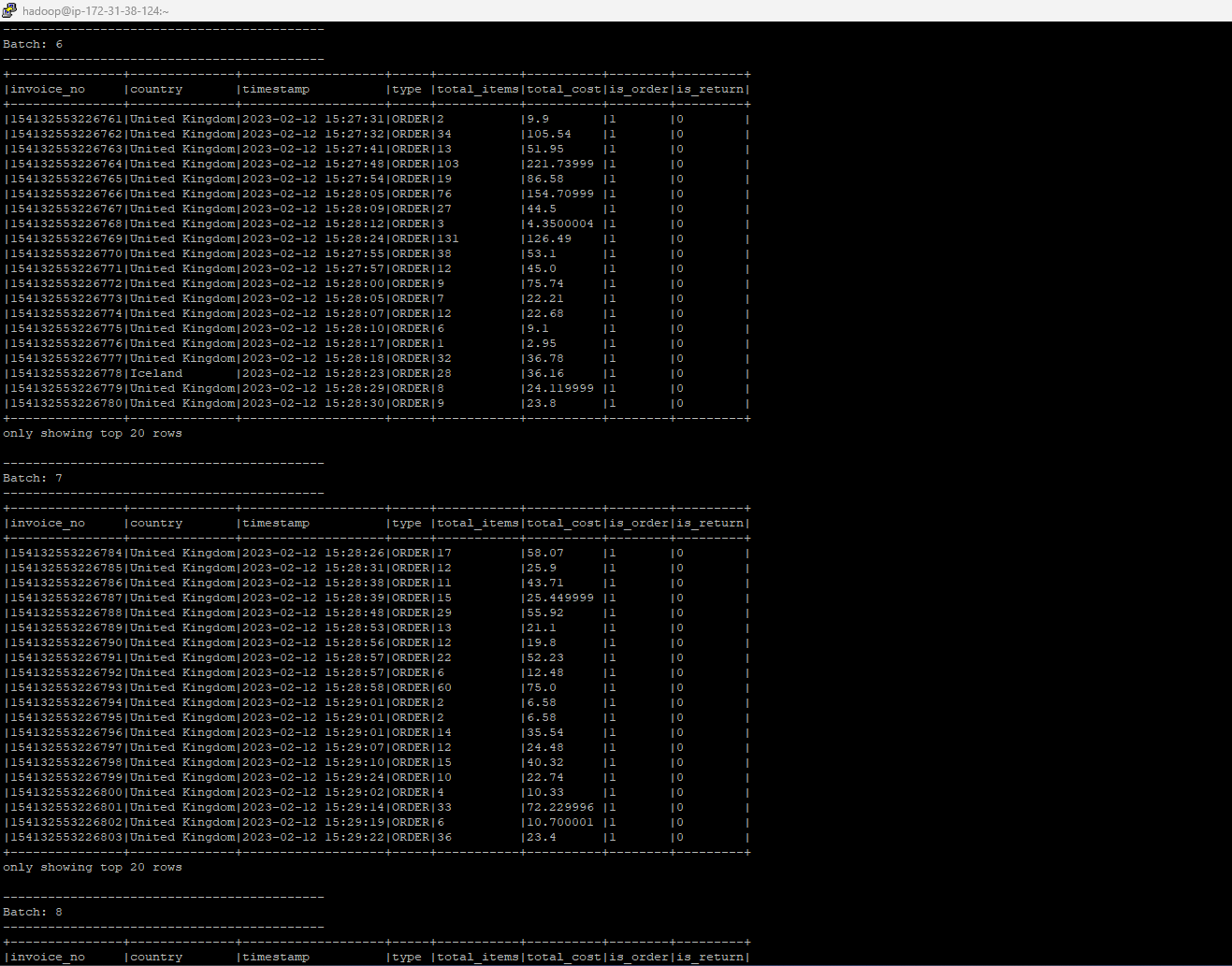
Text

Description automatically generated

Graphical user interface, text

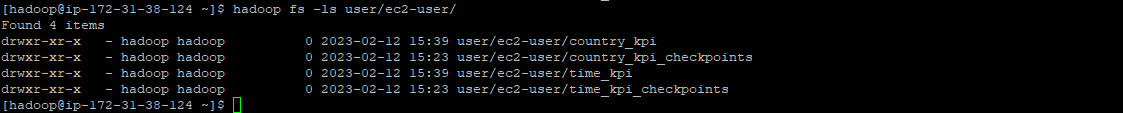
Description automatically generated





I checked HDFS to make sure the KPI files were present.

hadoop fs -ls /user/ec2-user/



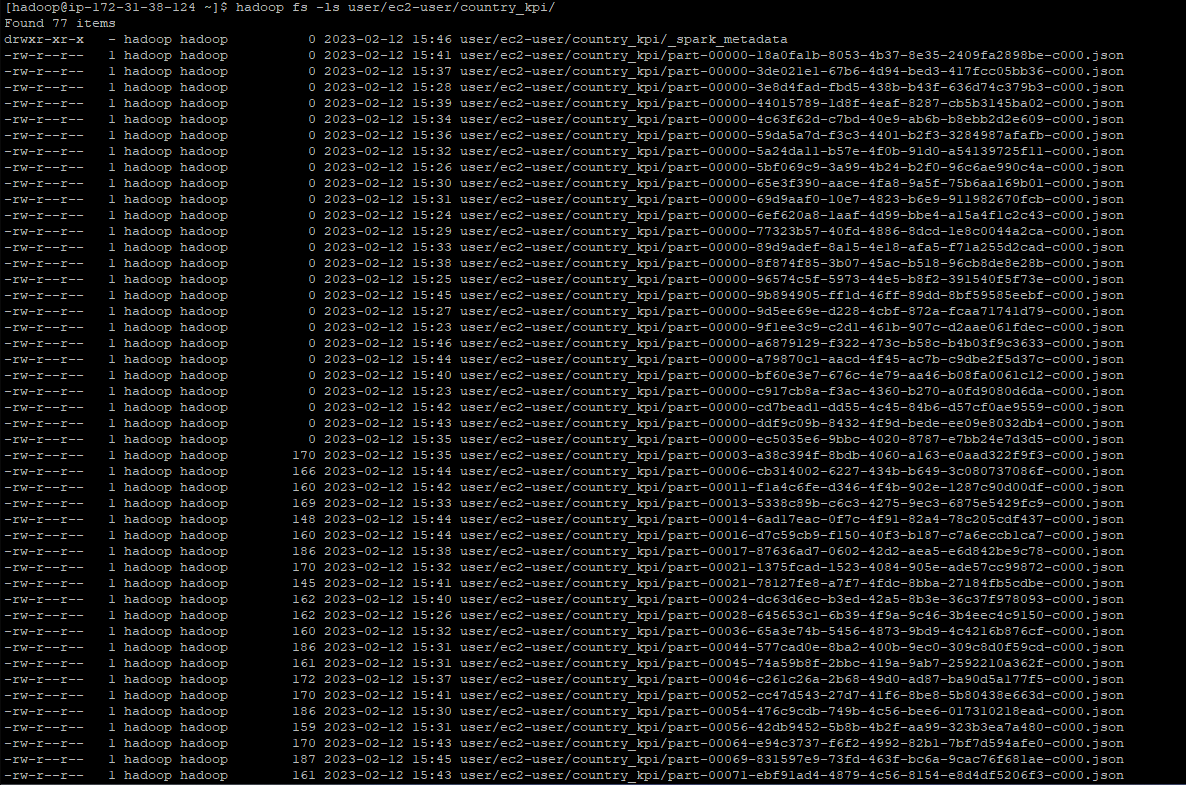
I also checked the folders to see the JSON files:

hadoop fs -ls /user/ec2-user/time\_kpi

Text

Description automatically generated

hadoop fs -ls user/ec2-user/country\_kpi



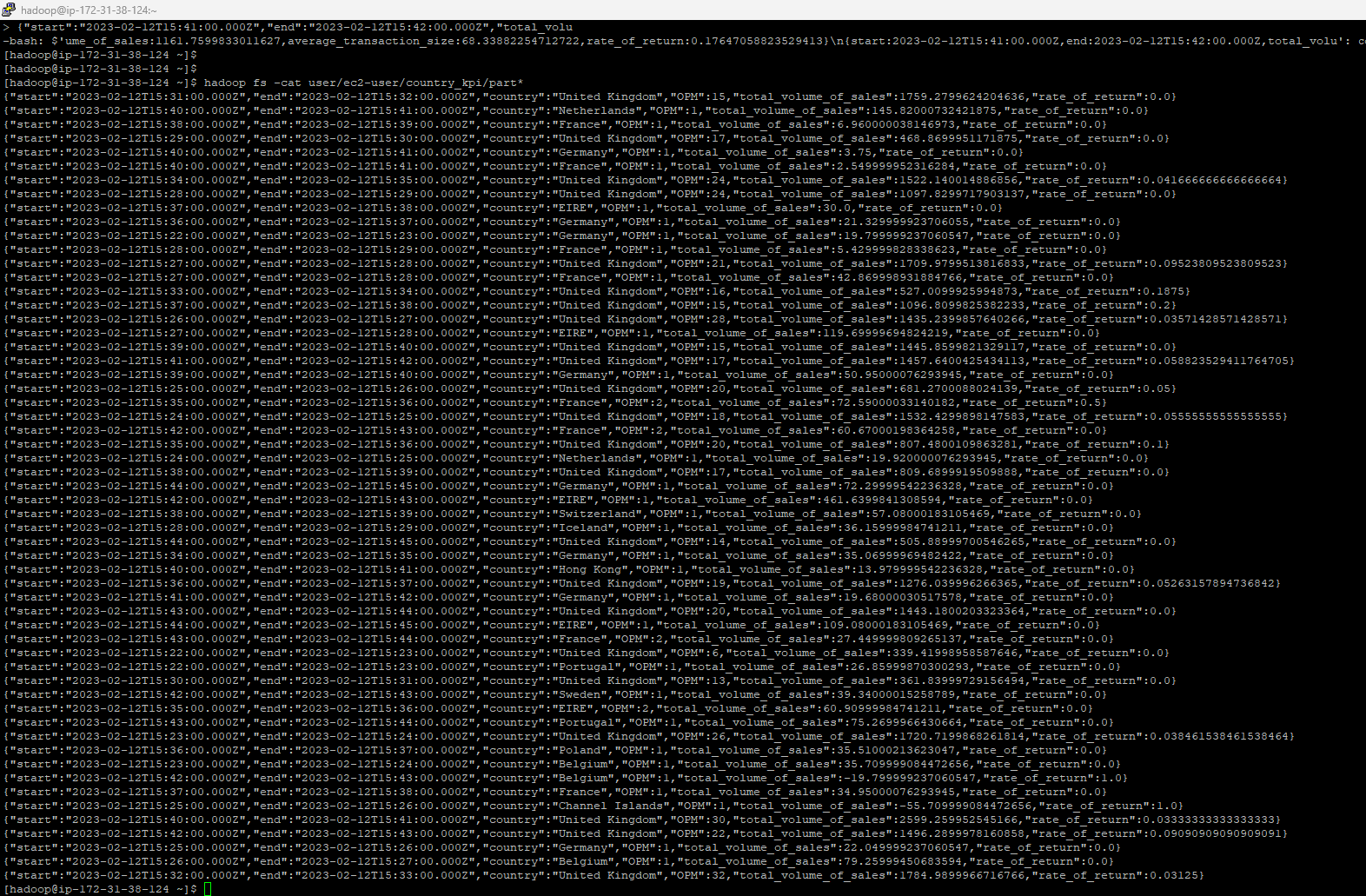
And used ‘cat’ command to take a look at the data:

hadoop fs -cat user/ec2-user/time\_kpi/part\*

Text

Description automatically generated

hadoop fs -cat user/ec2-user/country\_kpi/part\*



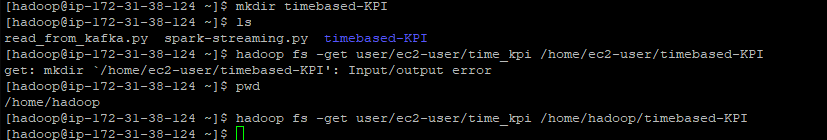
Transfer of files from EMR Instance on AWS to my system, using WinSCP :

First, I needed to transfer the JSON files from HDFS into the the EC2 system I created directories for time-based and then time-and-country-based KPIs as ec2-user.

Using the ‘get’ command I copied the contents of the output folders into the EC2 system.

mkdir timebased-KPI

hadoop fs -get user/ec2-user/time\_kpi /home/hadoop/timebased-KPI



mkdir country-and-timebased-KPI

hadoop fs -get user/ec2-user/country\_kpi /home/hadoop/country-and-timebased-KPI



Thereafter I used WinSCP to establish a connection between the EC2 instance and my local file system to transfer all the required files into my system.