1. **Read the data from kafka using the readStream**

```
spark = SparkSession  \
        .builder  \
        .appName("StructuredSocketRead")  \
        .getOrCreate()
spark.sparkContext.setLogLevel('ERROR')
orderRaw = spark  \
        .readStream  \
        .format("kafka")  \
        .option("kafka.bootstrap.servers","ec2-18-211-252-152.compute-
1.amazonaws.com:9092")  \
        .option("subscribe","real-time-project")  \
        .load()
```

2. **Defined schema and parsed the JSON and created orderStream dataframe**

```
    jsonSchema = StructType() \
        .add("invoice_no", StringType()) \
        .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", DoubleType())
   ])))
orderStream = orderRaw.select(from_json(col("value").cast("string"),
jsonSchema).alias("data")).select("data.*")
```

3. **Created the user defined functions for total_items, total_cost, is_order and is_return**

```
  def get_total_item_count(items):
   total_items = 0
   for item in items:
       total_items = total_items + item[2]
   return total_items
def get_total_cost_per_record(items):
   total_cost = 0
   for item in items:
```

```
        total_cost = total_cost + (item[2] * item[3])
    return total_cost
def get_is_order_type(type):
    order_type_flag = 0
    if type == 'ORDER':
        order_type_flag = 1
    else:
        order_type_flag = 0
    return order_type_flag
def get_is_order_return_type(type):
    order_return_type_flag = 0
    if type == 'ORDER':
        order_return_type_flag = 0
    else:
        order_return_type_flag = 1
    return order_return_type_flag
```

4. **Defined the user defined functions with utility**

```
# Define the UDFs with the utility functions
add_total_count = udf(get_total_item_count, DoubleType())
add_total_cost = udf(get_total_cost_per_record, DoubleType())
add_is_order_flg = udf(get_is_order_type, IntegerType())
add_is_return_flg = udf(get_is_order_return_type, IntegerType())
```

5. **Created new columnsTotal Cost, Total_Items, Is_order and Is_return using UDF**

```
Data_Frame_Total_Items_Cost= orderStream \
    .withColumn("Total_Items", add_total_count(orderStream.items)) \
    .withColumn("Total_Cost", add_total_cost(orderStream.items)) \
    .withColumn("is_order", add_is_order_flg(orderStream.type)) \
    .withColumn("is_return",
add_is_return_flg(orderStream.type)).select("invoice_no","country","timestamp",
"Total_Items", "Total_Cost","is_order", "is_return")
```

6. **Written the intermediary datasets into the console**

```
query = Data_Frame_Total_Items_Cost  \
        .writeStream  \
        .outputMode("append")  \
        .format("console")  \
```

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

7. **Calculating time based KPI using withWatermark and groupBy**

```
aggStreamByTime = Data_Frame_Total_Items_Cost \
    .withWatermark("timestamp", "1 minute") \
    .groupBy(window("timestamp", "1 minute", "1 minute")) \

.agg(sum("Total_Cost").alias("total_volume_of_sales"),count("invoice_no").alias
("OPM"),avg("is_return").alias("avg_rate_of_return")).select("window.start",
"window.end", "OPM", "total_volume_of_sales", "avg_rate_of_return")
```

8. **Calculating country based KPI using withWatermark and groupBy**

```
# Calculate Country based KPIs
aggStreamByCountry= Data_Frame_Total_Items_Cost \
    .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("avg_rate_of_return")).select("window.start",
"window.end","country", "OPM", "total_volume_of_sales", "avg_rate_of_return")
```

9. **Writing the Time based KPI to HDFS**

```
    # Writing the Time Based KPIs into HDFS
queryByTime= aggStreamByTime.writeStream \
    .format("json") \
    .outputMode("append") \
    .option("truncate", "false") \
    .option("path", "time-wise-kp1") \
    .option("checkpointLocation", "time-cp1") \
    .trigger(processingTime="1 minute") \
    .start()
```

10. **Writing the country based KPI to HDFS**

```
    queryByCountry = aggStreamByCountry.writeStream \
    .format("json") \
    .outputMode("append") \
    .option("truncate", "false") \
    .option("path", "time-country-wise-kp1") \
    .option("checkpointLocation", "time-country-cp1") \
    .trigger(processingTime="1 minute") \
    .start()
queryByCountry.awaitTermination()
```

11. Navigated to HDFS path and downloaded the output files

```
[ec2-user@ip-10-0-0-176 ~]$ hadoop fs -get /user/hadoop/time-wise-kpi ./time-kpi-2
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/hadoop/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/share/aws/emr/emrfs/lib/slf4j-log4j12-1.7.12.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
get: `/user/hadoop/time-wise-kpi': No such file or directory
[ec2-user@ip-10-0-0-176 ~]$ hadoop fs -get /user/hadoop/time-wise-kpi ./time-kpi-2
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/hadoop/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/share/aws/emr/emrfs/lib/slf4j-log4j12-1.7.12.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
[ec2-user@ip-10-0-0-176 ~]$ hadoop fs -get /user/hadoop/time-country-wise-kpi ./cntry-kpi-2
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/hadoop/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/share/aws/emr/emrfs/lib/slf4j-log4j12-1.7.12.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
[ec2-user@ip-10-0-0-176 ~]$ ls
cntry-kpi-2  country-kpi  streamtest2.py  streamtest.py  time-kpi  time-kpi-2  time-wise-kpi
[ec2-user@ip-10-0-0-176 ~]$
```

12. Used WinSCP to move the files from Hadoop to Desktop