Code Explanation

Imported Libraries

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
```

Explanation:

The code starts by importing necessary libraries. SparkSession is the entry point to any Spark functionality. functions provides a set of predefined functions (like sum, avg etc.) that can be used during DataFrame operations. types contains a set of type classes for defining the schema.

Implemented User-Defined Functions (UDFs)

1. **Total Cost UDF**

```
def calculate_total_cost(items,t_type):
    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 t_type  == 'RETURN':
        return total_cost *-1
    else:
        return total_cost
```

Code Explanation:

To determine the overall revenue generated from each invoice, I calculated the revenue from the sale of individual products. This involved multiplying the unit price of each product by the quantity purchased. By summing up these amounts for all the products within a single invoice, I obtained the total cost

associated with that order. Additionally, to account for return transactions, I ensured that the total cost is represented as a negative value.

2. Total Items UDF

```
def calculate_total_items(items):
    if items is not None:
        item_count =0
        for item in items:
            item_count = item_count + item['quantity']
        return item_count
```

Code Explanation:

To find out the total number of products in each invoice, I summed up the quantities ordered for all the individual products within that specific invoice. By doing this, I was able to determine the overall volume of products associated with each order, providing a comprehensive view of how many items were involved in that transaction.

3. Is Order UDF

```
def is_order(t_type):
    if t_type == 'ORDER':
        return (1)
    else:
        return(0)
```

Code Explanation:

To ascertain whether an invoice corresponds to an order, I employed an if-else conditional statement.

4. Is Return UDF

```
def is_return(t_type):
    if t_type == 'RETURN':
        return(1)
    else:
        return(0)
```

Code Explanation:
To establish whether an invoice is related to a return transaction, I utilized an if-else conditional structure.

Create Spark Session

A Spark session is created with the name "spark-streaming".

```
spark = SparkSession.builder\
         .appName("spark-streaming")\
         .getOrCreate()
```

Connect to Kafka

It connects to a Kafka server and subscribes to the "real-time-project" topic.

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

Schema Definition

The schema for the incoming data is defined.

```
jsonSchema = 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()),
])))
```

Deserialize Kafka Messages

Kafka messages are deserialized from JSON into a Spark DataFrame

```
orders = raw_order.select(from_json(col("value").cast("string"),
jsonSchema).alias("data")).select("data.*")
```

Register UDFs and Add New Columns

The UDFs are applied to the DataFrame to generate new columns: "total_cost", "total_items", "is_order", "is_return".

```
total_cost_udf = udf(calculate_total_cost,DoubleType())
total_items_udf = udf(calculate_total_items,IntegerType())
is_order_udf = udf(is_order, IntegerType())
is_return_udf = udf(is_return, IntegerType())
```

```
orders_with_udfs = orders \
                    .withColumn("total_cost",
total_cost_udf(orders.items,orders.type)) \
                    .withColumn("total_items", total_items_udf(orders.items)) \
                    .withColumn("is_order", is_order_udf(orders.type)) \
                    .withColumn("is_return", is_return_udf(orders.type))
```

Writing to Console

```
extendedOrderQuery = orders_with_udfs \
    .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()
```

Aggregating KPIs by Time

Two different aggregations are performed:

```
aggStreamByTime = orders_with_udfs \
                  .withWatermark("timestamp","1 minute") \
                  .groupBy(window("timestamp","1 minute","1 minute"))\
                  .agg(sum("total_cost").alias("total_sales_volume"),
                       count("invoice_no").alias("OPM"),
                       avg("is_return").alias("rate_of_return"),
                       avg("total_cost").alias("average_transaction_size"))\
                    .select("window","OPM","total_sales_volume","average_trans
action_size","rate_of_return")
```

Writing to HDFS

The aggregated KPIs are written to HDFS in JSON format.

```
queryByTime = aggStreamByTime.writeStream \
                .format('json')\
                .outputMode("append")\
                .option("truncate","false")\
                .option("path","/user/ec2-user/time_kpi")\
                .option("checkpointLocation","/user/ec2-
user/time_kpi_checkpoints")\
                .trigger(processingTime="1 minute")\
                .start()
```

Aggregating KPIs by Time and country

```
# Calculating time-and-country-based KPIs
aggStreamByCountry = orders_with_udfs \
                     .withWatermark("timestamp","1 minute") \
                     .groupBy(window("timestamp","1 minute","1
minute"),"country")\
                    .agg(sum("total_cost").alias("total_sales_volume"),
                           count("invoice_no").alias("OPM"),
                            avg("is_return").alias("rate_of_return"))    \
                      .select("window","country","OPM","total_sales_volume","r
ate_of_return")


# Writing the time-and-country-based KPIs data to HDFS
queryByCountry = aggStreamByCountry.writeStream \
    .format("json") \
    .outputMode("append") \
    .option("truncate","false") \
    .option("path","/user/ec2-user/country_kpi") \
    .option("checkpointLocation","/user/ec2-user/country_kpi_checkpoints") \
    .trigger(processingTime="1 minute") \
    .start()
```

Awaiting Termination

```
extendedOrderQuery.awaitTermination()
queryByTime.awaitTermination()
queryByCountry.awaitTermination()
```