1. **SparkSession Setup:**

```
spark = SparkSession.builder.appName("RetailDataAnalysis").getOrCreate()
spark.sparkContext.setLogLevel('ERROR')
```

   It initializes a `SparkSession` named "RetailDataAnalysis" and sets the log level to ERROR.

2. **User-Defined Functions (UDFs):**

   - `is_a_order(type)` and `is_a_return(type)`: UDFs that return 1 if the type is 'ORDER' or 'RETURN', respectively, and 0 otherwise.
   - `total_item_count(items)`: UDF that calculates the total quantity of items in a list.
   - `total_cost(items, type)`: UDF that calculates the total cost of items, considering the type as 'RETURN' for negative values.

3. **Register UDFs:**

```
is_order = udf(is_a_order, IntegerType())
is_return = udf(is_a_return, IntegerType())
add_total_item_count = udf(total_item_count, IntegerType())
add_total_cost = udf(total_cost, FloatType())
```

   It registers the UDFs to be used in Spark DataFrame operations.

4. **Kafka Options:**

```
kafka_options = {
    "kafka.bootstrap.servers": "18.211.252.152:9092",
    "subscribe": "real-time-project",
    "startingOffsets": "latest"
}
```

   It defines the Kafka configuration options, such as bootstrap servers, topic subscription, and starting offsets.

5. **Read Input Data from Kafka:**

```
raw_stream = spark.readStream.format("kafka").options(**kafka_options).load()
```

   It reads data from Kafka using the specified options.

6. **Define Schema for JSON Data:**

```
JSON_Schema = StructType().add("invoice_no", LongType()).add("country", StringType()).add("timestamp", TimestampType()).add("ty
```

   It defines the schema for the JSON data read from Kafka.

7. **Extract Data from JSON Format:**

```
order_stream_data = raw_stream.select(from_json(col("value").cast("string"), JSON_Schema).alias("data")).select("data.*")
```

   It extracts structured data from the JSON format.

8. **Calculate Additional Columns for the Stream:**

```
enriched_stream_output = order_stream_data \
    .withColumn("total_cost", add_total_cost(order_stream_data.items, order_stream_data.type)) \
    .withColumn("total_items", add_total_item_count(order_stream_data.items)) \
    .withColumn("is_order", is_order(order_stream_data.type)) \
    .withColumn("is_return", is_return(order_stream_data.type))
```

   It calculates additional columns such as total cost, total items, and flags for order and return.

9. **Write Summarized Input Table to Console:**

```
order_batch = enriched_stream_output \
    .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()
```

It writes the summarized input table to the console and specified path in append mode.

10. **Calculate Time-Based KPIs:**

```
agg_time = enriched_stream_output \
    .withWatermark("timestamp", "1 minute") \
    .groupby(window("timestamp", "1 minute")) \
    .agg(sum("total_cost").alias("total_volume_of_sales"), avg("total_cost").alias("average_transaction_size"), count("invoice_
    .select("window.start", "window.end", "OPM", "total_volume_of_sales", "average_transaction_size", "rate_of_return")
```

It calculates time-based Key Performance Indicators (KPIs) using a 1-minute window.

11. **Calculate Time and Country-Based KPIs:**

```
agg_country_time = enriched_stream_output \
    .withWatermark("timestamp", "1 minute") \
    .groupBy(window("timestamp", "1 minute"), "country") \
    .agg(sum("total_cost").alias("total_volume_of_sales"), count("invoice_no").alias("OPM"), avg("is_return").alias("rate_of_re
    .select("window.start", "window.end", "country", "OPM", "total_volume_of_sales", "rate_of_return")
```

It calculates time and country-based KPIs using a 1-minute window.

12. **Write Time-Based KPI Values to Console:**

```
by_time = agg_time.writeStream \
    .format("json") \
    .outputMode("append") \
    .option("truncate", "false") \
    .option("path", "Timebased-KPI") \
    .option("checkpointLocation", "Timebased-KPI-checkpoints") \
    .trigger(processingTime="1 minute") \
    .start()
```

It writes time-based KPI values to the console and specified path in append mode.

13. **Write Time and Country-Based KPI Values to Console:**

```
by_time_country = agg_country_time.writeStream \
    .format("json") \
    .outputMode("append") \
    .option("truncate", "false") \
    .option("path", "Country-and-timebased-KPI") \
    .option("checkpointLocation", "Country-and-timebased-KPI_checkpoints") \
    .trigger(processingTime="1 minute") \
    .start()
```

It writes time and country-based KPI values to the console and specified path in append mode.

14. **Await Termination of Streams:**

```
order_batch.awaitTermination()
by_time.awaitTermination()
by_time_country.awaitTermination()
```

It waits for the termination of the streaming queries.