Introduction

Welcome to the course project. This project will test your knowledge of the tools related to **real-time data processing**, which you learnt throughout this course. The project mainly revolves around concepts of **Apache Kafka and Apache Spark Streaming**, which are some of the most widely used tools in the industry for real-time data processing.

In this project, you will go through a real-world use case from the retail sector. Your task would be to ingest data from a centralised Kafka server in real-time and process it to calculate various KPIs or key performance indicators.

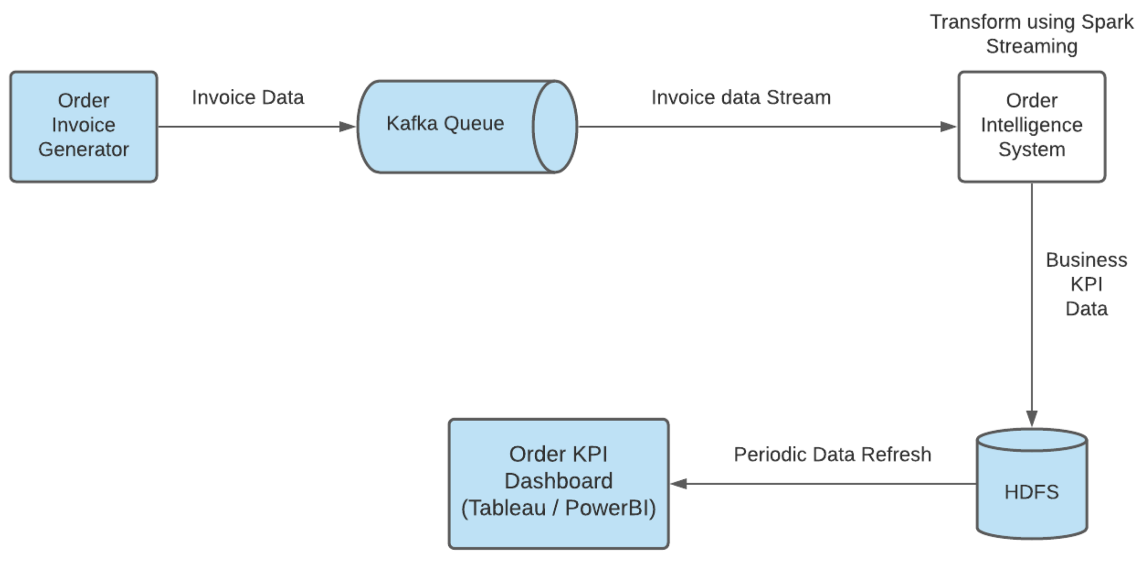
Problem Statement

Digitally enabled customers like to shop on the run, and that is the reason why online shopping is one of the most popular online activities worldwide. In **2019**, global e-commerce sales amounted to **3.53 trillion USD** and are projected to grow to **6.54 trillion USD in 2022**.

The analytical capabilities of big data have had a positive impact across industries, including e-commerce. Big data tools improve business performance by enabling companies to analyse trends and current consumer behavioural patterns and offer better and more customised products.

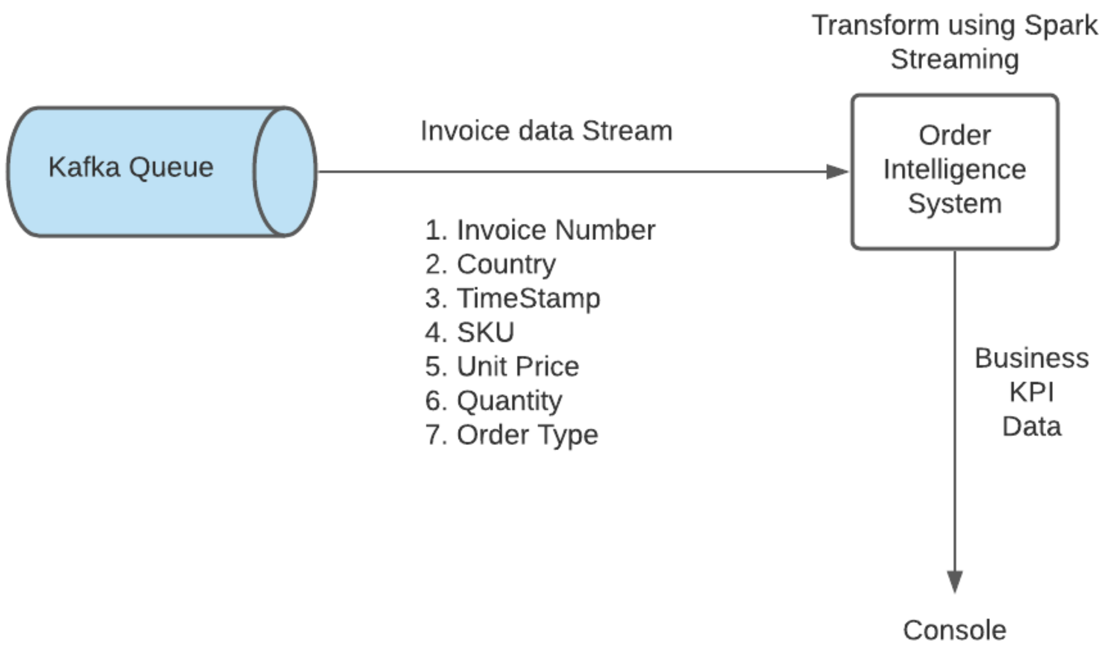
For the purposes of this project, you have been tasked with computing various **Key Performance Indicators (KPIs)** for an e-commerce company, RetailCorp Inc. You have been provided real-time sales data of the company across the globe. The data contains information related to the invoices of orders placed by customers all around the world. You will get to know the details of the data in the next segment.

At the industry level, an end-to-end data pipeline is built for this purpose. Tools such as HDFS(Hadoop Distributed File System) are used to store the data that is processed by the real-time processing framework and then shown on a dashboard with tools such as Tableau and PowerBI. The image given below is an example of such a complete data pipeline.



Data Pipeline in the Industry

For our project, we will be focusing on the '**Order Intelligence**' part of this data pipeline. The image given below shows the architecture of the data pipeline that we will follow in this project.



Architecture of the Project

Broadly, **you will perform the following tasks in this project:**

- Reading the sales data from the Kafka server
- Preprocessing the data to calculate additional derived columns such as total_cost etc
- Calculating the time-based KPIs and time and country-based KPIs
- Storing the KPIs (both time-based and time- and country-based) for a 10-minute interval into separate JSON files for further analysis

Data Explanation

In this segment, you will learn about the data that you will be dealing with in this project in detail.

As discussed in the video, a centralised Kafka server has been set up where the data will be hosted. The details of the Kafka server, such as the broker and topic, will be shared to you in the next segment.

The data is based on an Online Retail Data Set in the UCI Machine Learning Repository. Each order's invoice has been represented in a JSON format. The sample data looks like this.

```json
{
 "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
   }
 ]
}
```

As you can see, the data contains the following information:

- **Invoice number**: Identifier of the invoice
- **Country**: Country where the order is placed
- **Timestamp**: Time at which the order is placed
- **Type**: Whether this is a new order or a return order
- **SKU (Stock Keeping Unit)**: Identifier of the product being ordered
- **Title**: Name of the product is ordered
- **Unit price**: Price of a single unit of the product
- **Quantity**: Quantity of the product being ordered

You will be using these columns to derive some additional columns as well, which will help you in calculating the KPIs. You will learn more about this in the next segment.

Acknowledgement

The data was published in the UCI Machine Learning Repository by Dr Daqing Chen, Director: Public Analytics group.

Key Performance Parameters

In this segment, you will learn about the various key performance parameters that you will be calculating as part of this project.

1. **Total volume of sales:**

By calculating this, you can see where and when demand is higher and try to understand the reason. It helps in creating projections for the future. The total volume of sales is the sum of the transaction value of all orders in the given time window. The total transaction value of a specific order will be calculated as follows:

$$\sum(quantity * unitprice)$$

The total volume of sales in a time interval can be calculated as the summation of the transaction values of all the orders in that time interval. Also, the sales data stream contains a few returns. In the case of returns, the transaction amount needs to be subtracted. So, the equation for the total volume of sales, in this case, will be calculated as follows:

$$\sum_{Order}(quantity * unitprice) - \sum_{Return}(quantity * unitprice)$$

2. **OPM** (**orders per minute**):

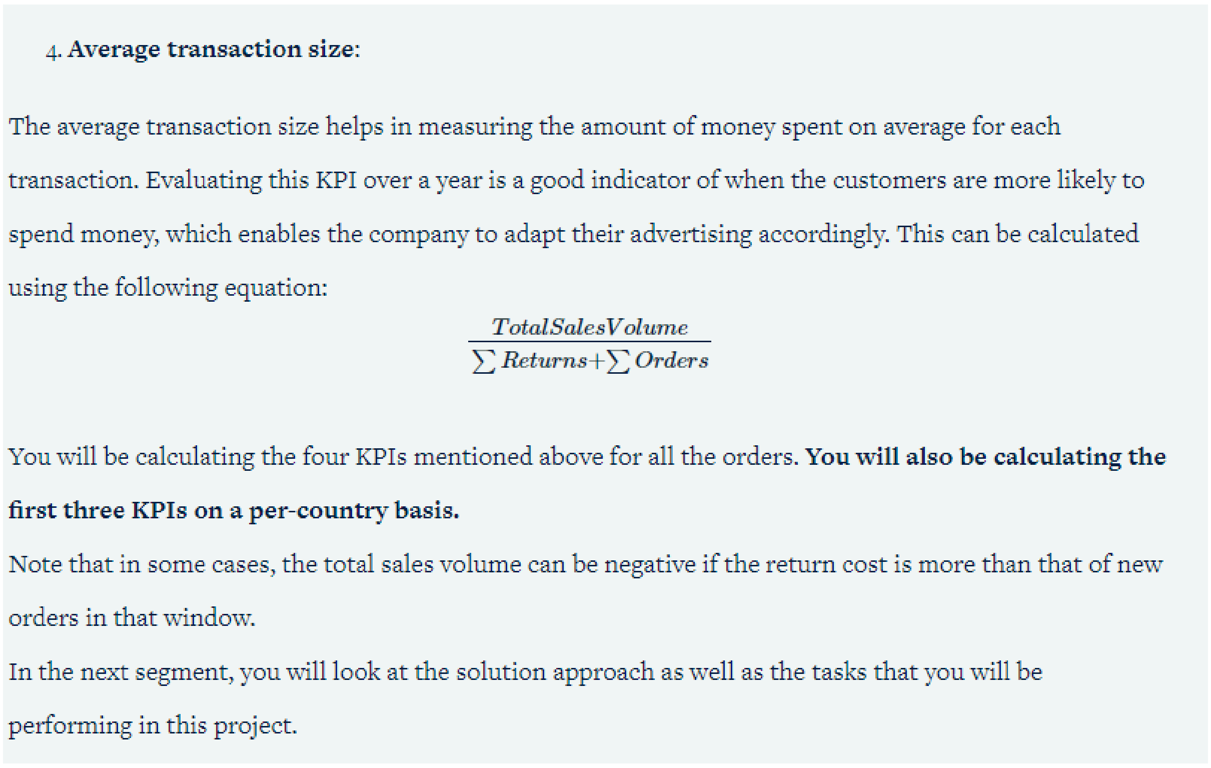
Orders per minute (OPM) is another important metric for e-commerce companies. It is a direct indicator of the success of the company. As the name suggests, it is the total number of orders received in a minute.

$$?Invoices?$$

3. **Rate of return**:

No business likes to see a customer returning their items. Returns are costly because they need to be processed again and have adverse effects on revenue. The rate of return indicates customer satisfaction for the company. The more satisfied the customer is, the lower the rate of return will be. The rate of return is calculated against the total number of invoices using the following equation:

$$\frac{\sum Returns}{\sum Returns + \sum Orders}$$



4. **Average transaction size:**

The average transaction size helps in measuring the amount of money spent on average for each transaction. Evaluating this KPI over a year is a good indicator of when the customers are more likely to spend money, which enables the company to adapt their advertising accordingly. This can be calculated using the following equation:

$$\frac{TotalSalesVolume}{\sum Returns+\sum Orders}$$

You will be calculating the four KPIs mentioned above for all the orders. **You will also be calculating the first three KPIs on a per-country basis.**

Note that in some cases, the total sales volume can be negative if the return cost is more than that of new orders in that window.

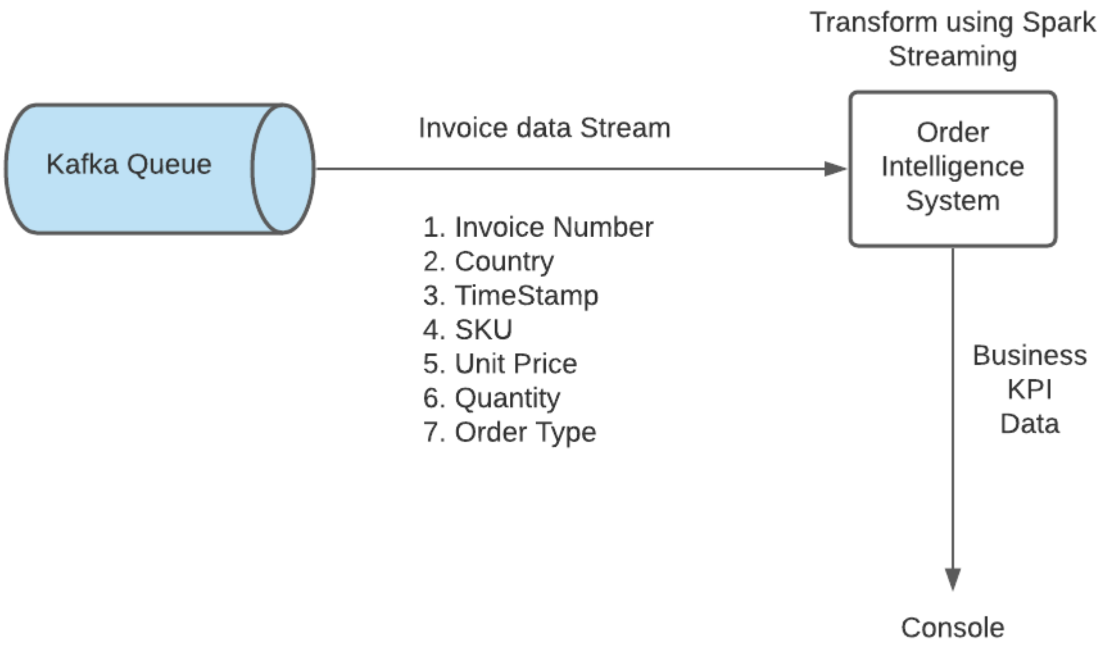
In the next segment, you will look at the solution approach as well as the tasks that you will be performing in this project.

Solution Approach

In this segment, you will learn in detail what tasks you will have to perform in this project.

As discussed in the earlier segments, you will be following this architecture for your project.

Please note that for this project, you will need an EMR cluster configured with a similar configuration that you followed in the **Apache Spark Streaming** module.



Architecture of the Project

The tasks that you will be performing in this project are as follows -

- **Reading input data from Kafka**
    - Code to take raw Stream data from Kafka server
    - Details of the Kafka broker are as follows:
        - **Bootstrap Server** - 18.211.252.152
        - **Port** - 9092
        - **Topic** - real-time-project
- **Calculating additional columns and writing the summarised input table to the console**
    - The following attributes from the raw Stream data have to be taken into account for the project:
        - **invoice_no**: Identifier of the invoice

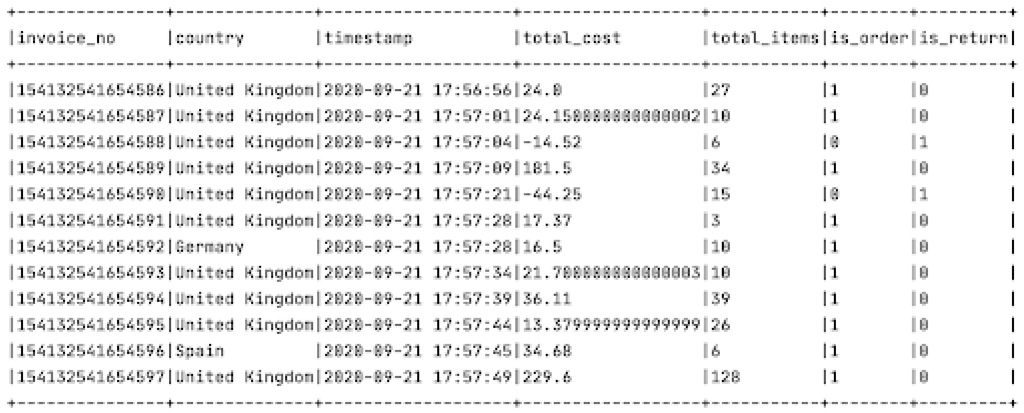
    - **country**: Country where the order is placed
    - **timestamp**: Time at which the order is placed
  - In addition to these attributes, the following **UDFs** have to be calculated and added to the table:
    - **total_cost**: Total cost of an order arrived at by summing up the cost of all products in that invoice (The return cost is treated as a loss. Thus, for return orders, this value will be negative.)
    - **total_items**: Total number of items present in an order
    - **is_order**: This flag denotes whether an order is a new order or not. If this invoice is for a return order, the value should be 0.
    - **is_return**: This flag denotes whether an order is a return order or not. If this invoice is for a new sales order, the value should be 0.
  - The input table must be generated for each **one-minute window**.
  - Code to define the schema of a single order
  - Code to define the aforementioned UDFs and any utility functions are written to calculate them
  - Code to write the final summarised input values to the console. This summarised input table has to be stored in a Console-output file. This can be done by simply appending '**> file-name**' to the Spark-Submit code as follows:

```
spark2-submit_command > file_name
```

-

An example table written to the console is also needed. It should look like below.

Note: The below is just a reference format that can be followed, however, you are free to pick your own approach to solve the problem.



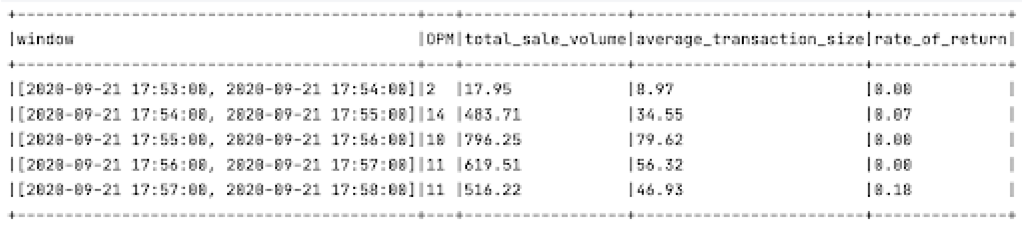
```
+---------------+--------------+-------------------+------------------+-----------+--------+---------+
|invoice_no     |country       |timestamp          |total_cost        |total_items|is_order|is_return|
+---------------+--------------+-------------------+------------------+-----------+--------+---------+
|154132541654586|United Kingdom|2020-09-21 17:56:56|24.0              |27         |1       |0        |
|154132541654587|United Kingdom|2020-09-21 17:57:01|24.150000000000002|10         |1       |0        |
|154132541654588|United Kingdom|2020-09-21 17:57:04|-14.52            |6          |0       |1        |
|154132541654589|United Kingdom|2020-09-21 17:57:09|181.5             |34         |1       |0        |
|154132541654590|United Kingdom|2020-09-21 17:57:21|-44.25            |15         |0       |1        |
|154132541654591|United Kingdom|2020-09-21 17:57:28|17.37             |3          |1       |0        |
|154132541654592|Germany       |2020-09-21 17:57:28|16.5              |10         |1       |0        |
|154132541654593|United Kingdom|2020-09-21 17:57:34|21.700000000000003|10         |1       |0        |
|154132541654594|United Kingdom|2020-09-21 17:57:39|36.11             |39         |1       |0        |
|154132541654595|United Kingdom|2020-09-21 17:57:44|13.379999999999999|26         |1       |0        |
|154132541654596|Spain         |2020-09-21 17:57:45|34.68             |6          |1       |0        |
|154132541654597|United Kingdom|2020-09-21 17:57:49|229.6             |128        |1       |0        |
+---------------+--------------+-------------------+------------------+-----------+--------+---------+
```

Final Summarised Input Values

- **Calculating time-based KPIs**:
  - Code to calculate time-based KPIs **tumbling window of one minute** on orders **across the globe**. These KPIs were discussed in the previous segment.
  - KPIs have to be calculated for a 10-minute interval for evaluation; so, ten 1-minute window batches have to be taken.

- ○ Time-based KPIs can be structured like below. (These tables do not need to be outputted and are just for reference as to how your KPI tables must be structured when all the files are combined.)

Note: The below is just a reference format that can be followed, however, you are free to pick your own approach to solve the problem.

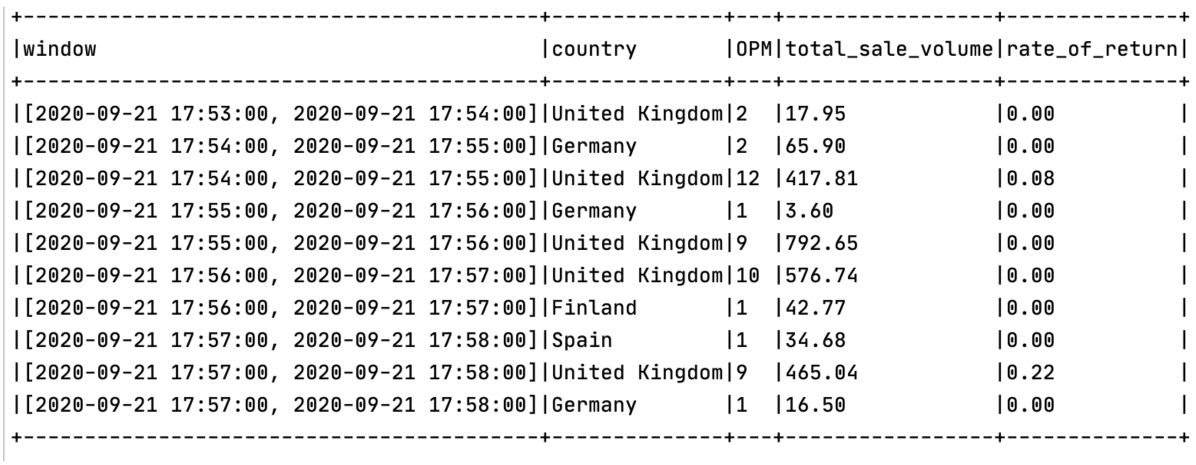


```
+------------------------------------------+---+-----------------+------------------------+--------------+
|window                                    |OPM|total_sale_volume|average_transaction_size|rate_of_return|
+------------------------------------------+---+-----------------+------------------------+--------------+
|[2020-09-21 17:53:00, 2020-09-21 17:54:00]|2  |17.95            |8.97                    |0.00          |
|[2020-09-21 17:54:00, 2020-09-21 17:55:00]|14 |483.71           |34.55                   |0.07          |
|[2020-09-21 17:55:00, 2020-09-21 17:56:00]|10 |796.25           |79.62                   |0.00          |
|[2020-09-21 17:56:00, 2020-09-21 17:57:00]|11 |619.51           |56.32                   |0.00          |
|[2020-09-21 17:57:00, 2020-09-21 17:58:00]|11 |516.22           |46.93                   |0.18          |
+------------------------------------------+---+-----------------+------------------------+--------------+
```

Time-based KPIs

- **Calculating time- and country-based KPIs**:
  - ○ Code to calculate time- and country-based KPIs **tumbling window of one minute on orders on a per-country basis**. These KPIs were discussed in the previous segment.
  - ○ KPIs have to be calculated for a 10-minute interval for evaluation; so, ten 1-minute window batches have to be taken.
  - ○ Time- and country-based KPIs can be structured like below. (These tables do not need to be outputted and are just for reference as to how your KPI tables must be structured.)

Note: The below is just a reference format that can be followed, however, you are free to pick your own approach to solve the problem.



```
+----------------------------------------+-------------+---+-----------------+--------------+
|window                                  |country      |OPM|total_sale_volume|rate_of_return|
+----------------------------------------+-------------+---+-----------------+--------------+
|[2020-09-21 17:53:00, 2020-09-21 17:54:00]|United Kingdom|2  |17.95            |0.00          |
|[2020-09-21 17:54:00, 2020-09-21 17:55:00]|Germany       |2  |65.90            |0.00          |
|[2020-09-21 17:54:00, 2020-09-21 17:55:00]|United Kingdom|12 |417.81           |0.08          |
|[2020-09-21 17:55:00, 2020-09-21 17:56:00]|Germany       |1  |3.60             |0.00          |
|[2020-09-21 17:55:00, 2020-09-21 17:56:00]|United Kingdom|9  |792.65           |0.00          |
|[2020-09-21 17:56:00, 2020-09-21 17:57:00]|United Kingdom|10 |576.74           |0.00          |
|[2020-09-21 17:56:00, 2020-09-21 17:57:00]|Finland       |1  |42.77            |0.00          |
|[2020-09-21 17:57:00, 2020-09-21 17:58:00]|Spain         |1  |34.68            |0.00          |
|[2020-09-21 17:57:00, 2020-09-21 17:58:00]|United Kingdom|9  |465.04           |0.22          |
|[2020-09-21 17:57:00, 2020-09-21 17:58:00]|Germany       |1  |16.50            |0.00          |
+----------------------------------------+-------------+---+-----------------+--------------+
```

Time- and Country-based KPIs

- **Writing all the KPIs to JSON files**:
    - Code to write the KPIs calculated above into JSON files for each one-minute window.
    - These have to be written for a 10-minute interval.

All the resultant JSON files have to be downloaded and then archived into separate ZIP files for time-based and time- and country-based KPIs, respectively, along with the Console-output file.

Refer to the guidelines given in the '**Submission Guidelines**' segment of the next session to get more clarity on what you are required to submit.

Additional Resources

In this segment, you will learn about some of the concepts such as UDF, Watermark and File Sink in Spark Streaming and how to use them. These concepts will help you while working on the project.

UDF

UDF or **User-defined Functions** are functions that are created and modified by the users to fit their particular requirements. Let’s see how you can use UDF in the Spark Structured streaming application.

Let's assume that you are working on a list of quotes by famous people. So, the basic code for reading a set of quotes may look like this:

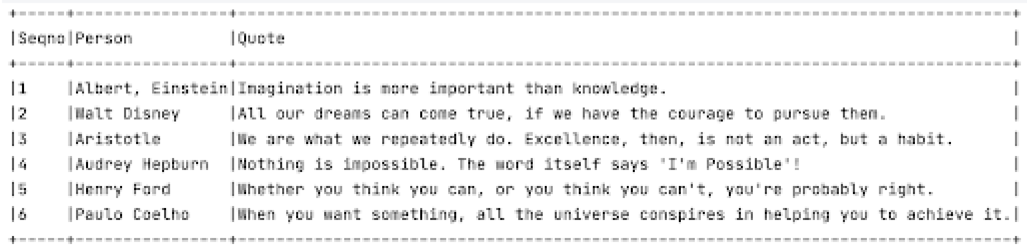
```
# Initialize Spark session
spark = SparkSession \
  .builder \
  .appName("udfExample") \
  .getOrCreate()

# Name of the columns
columns = ["Seqno", "Person", "Quote"]

# The quotes
data = [("1", "Albert, Einstein",
"Imagination is more important than knowledge."),
      ("2", "Walt Disney",
"All our dreams can come true, if we have the courage to pursue them."),
      ("3", "Aristotle",
"We are what we repeatedly do. Excellence, then, is not an act, but a habit."),
      ("4", "Audrey Hepburn",
"Nothing is impossible. The word itself says 'I'm Possible'!"),
      ("5", "Henry Ford",
"Whether you think you can, or you think you can't, you're probably right."),
    ("6", "Paulo Coelho",
"When you want something, all the universe conspires in helping you to achieve it.")]

# Create and display the data frame
df = spark.createDataFrame(data=data,schema=columns)
df.show(truncate=False)
```

At this point, the output will look something like that given below. Here, you can see three columns named **"Seqmo", "Person" and "Quote"** as they are present in the dataset.



```
+-----+----------------+---------------------------------------------------------------------------------+
|Seqno|Person          |Quote                                                                            |
+-----+----------------+---------------------------------------------------------------------------------+
|1    |Albert, Einstein|Imagination is more important than knowledge.                                    |
|2    |Walt Disney     |All our dreams can come true, if we have the courage to pursue them.             |
|3    |Aristotle       |We are what we repeatedly do. Excellence, then, is not an act, but a habit.      |
|4    |Audrey Hepburn  |Nothing is impossible. The word itself says 'I'm Possible'!                      |
|5    |Henry Ford      |Whether you think you can, or you think you can't, you're probably right.        |
|6    |Paulo Coelho    |When you want something, all the universe conspires in helping you to achieve it.|
+-----+----------------+---------------------------------------------------------------------------------+
```

Initial Dataframe

Now, you have a requirement of calculating the number of words inside each quote. You know a good way of doing that in Python, but pySpark library does not have any such standard and foolproof way of doing this calculation. In such cases, you will need a UDF to achieve that. Writing a UDF in pySpark is generally done in two phases:

- **Phase 1**: Writing the utility Python function, which contains the logic for the UDF
- **Phase 2**: Converting the Python function to UDF

You can define the Python function as shown below.

```
# Utility method for calculating word count for a file
def get_word_count(text):
  return len(re.findall(r'\w+', text))
```

Next, you can convert this Python to UDF as given below.

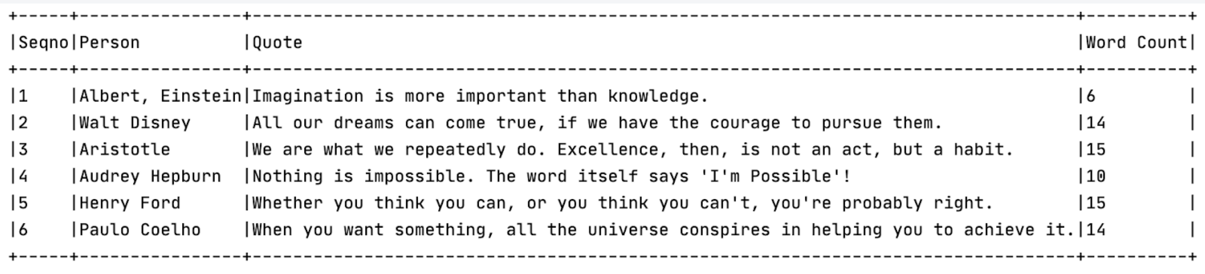
```
# Converting function to UDF
wordCountUDF = udf(lambda t: get_word_count(t), IntegerType())

# Alternatively, you can also use this syntax:
# wordCountUDF = udf(get_word_count, IntegerType())
```

Now, it is time to use the UDF to add a new column named "Word Count".

```
df.withColumn("Word Count", wordCountUDF(df.Quote)) \
  .show(truncate=False)
```

After you run the program, you can see that the new column "Word Count" gets added to the dataset.



```
+-----+----------------+--------------------------------------------------------------------------------+----------+
|Seqno|Person          |Quote                                                                           |Word Count|
+-----+----------------+--------------------------------------------------------------------------------+----------+
|1    |Albert, Einstein|Imagination is more important than knowledge.                                  |6         |
|2    |Walt Disney     |All our dreams can come true, if we have the courage to pursue them.            |14        |
|3    |Aristotle       |We are what we repeatedly do. Excellence, then, is not an act, but a habit.     |15        |
|4    |Audrey Hepburn  |Nothing is impossible. The word itself says 'I'm Possible'!                     |10        |
|5    |Henry Ford      |Whether you think you can, or you think you can't, you're probably right.       |15        |
|6    |Paulo Coelho    |When you want something, all the universe conspires in helping you to achieve it.|14        |
+-----+----------------+--------------------------------------------------------------------------------+----------+
```

Word Count UDF

The complete program with all the import statements is provided below.

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

# Utility method for calculating word count for a file
def get_word_count(text):
  return len(re.findall(r'\w+', text))

# Initialize Spark session
spark = SparkSession \
  .builder \
  .appName("udfExample") \
  .getOrCreate()

# Name of the columns
columns = ["Seqno", "Person", "Quote"]

# The quotes
data = [("1", "Albert, Einstein",
"Imagination is more important than knowledge."),
    ("2", "Walt Disney",
"All our dreams can come true, if we have the courage to pursue them."),
    ("3", "Aristotle",
"We are what we repeatedly do. Excellence, then, is not an act, but a habit."),
    ("4", "Audrey Hepburn",
"Nothing is impossible. The word itself says 'I'm Possible'!"),
    ("5", "Henry Ford",
"Whether you think you can, or you think you can't, you're probably right."),
    ("6", "Paulo Coelho",
"When you want something, all the universe conspires in helping you to achieve it.")]

df = spark.createDataFrame(data=data,schema=columns)

# Converting function to UDF
wordCountUDF = udf(get_word_count, IntegerType())

# Add new column using UDF
df.withColumn("Word Count", wordCountUDF(df.Quote)) \
  .show(truncate=False)
```

Watermark

You have already learnt the concepts of Watermark in Spark streaming. Now, let’s see how we can use it in the code.

Suppose you are working on a stream of network events where you receive requests from various devices. You have been tasked with calculating the request count based on the device model and a sliding time window of 10 minutes interval, and the window slides every five minutes.

Now, as there is a huge load on the system, there can be plenty of cases where an event appears a little late. So, you can put a cap of 10 minutes stating that an event can be at most 10 minutes late. You can use 'Watermark' to handle this specific scenario.

```
windowedStream = eventStream \
    .withWatermark("eventTime", "10 minutes") \ # allows 10 minutes lateness
    .groupBy(
        "device.model",
        window("timestamp", "10 minutes", "5 minutes")
    )
            ...
```

File Sink

Once the calculation is done, you can write the data into various sinks. In this exercise, you will write the code into a file sink. Let's see how to do that in code. For writing the data into file sink, you must specify the following:

- **Format**: It can be either of json, csv or parquet.

- **Output Mode**: The default mode is ‘append’. The other options are complete’ and ‘update’. For streaming use cases, the most widely used mode is ‘append’.
- **Path**: The path to the file where the data will be written.
- **Checkpoint Location**: The path to the checkpoint directory. This checkpoint location has to be a path in an HDFS compatible file system.

This corresponding code segment will be as follows:

```
aggregatedDataStream.writeStream \
  .format("csv") \
  .outputMode("append") \
  .option("truncate", "false") \
  .option("path", "path/to/destination/dir") \
  .option("checkpointLocation", "path/to/checkpoint/dir") \
  .trigger(processingTime="1 minute") \
  .start() \
  .awaitTermination()
```

Submission Guidelines

Submissions Required

Upload a ZIP file containing the following:

- A Python script (**spark-streaming.py**) containing the code for the project, which will process the input data streams into the resultant JSON files. This script should have proper comments, explaining all the steps taken.
- A PDF document (**CodeLogic.pdf**) containing the logic that you applied to solve the entire project. This will also have the commands used to run the

Spark Submit job and any other commands that you used in addition to the Python script discussed above.

- A ZIP file (**Output.zip**) containing the following files:
    - The **Console-output** file that will have the console output from the Spark Submit command containing the summarised input tables for the various batches
    - A ZIP file (**Timebased-KPI.zip**) containing all JSON files created for time-based KPIs
    - A ZIP file (**Country-and-timebased-KPI.zip**) containing all JSON files created for the country- and time-based KPIs