# Code Logic - Retail Data Analysis

In the Retail Data analysis, We will first create a folder, download required jar files and export kafka.

```
> mkdir coding-labs
> cd coding-lab/
> wget https://ds-spark-sql-kafka-jar.s3.amazonaws.com/spark-sql-kafka-0-10_2.11-2.3.0.jar
> export SPARK_KAFKA_VERSION=0.10
```

After the above steps start coding.

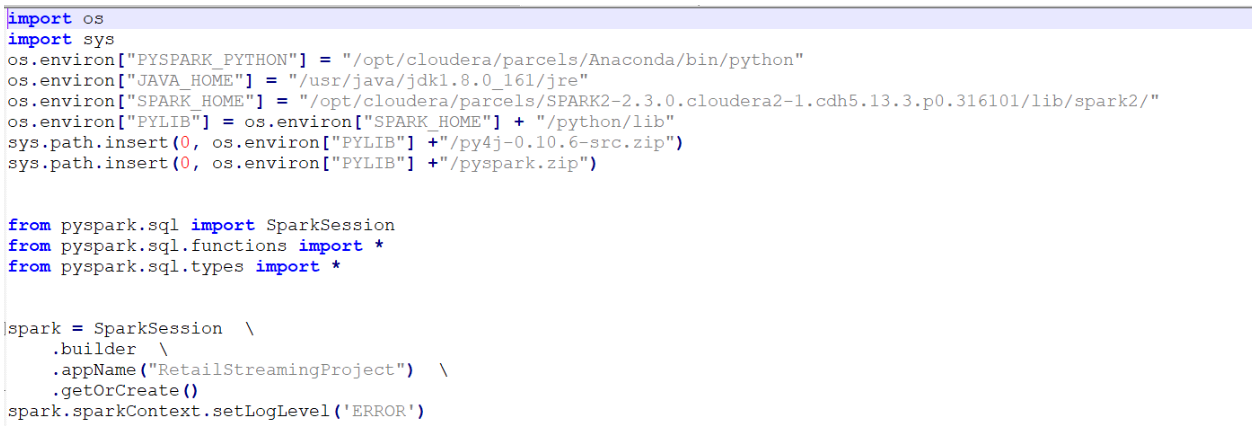
```
>vi spark-streaming.py
```

To execute run the below scripts to debug and spark submit to do the ETL task.

```
>python spark-streaming.py
>spark2-submit --jars spark-sql-kafka-0-10_2.11-2.3.0.jar spark-streaming.py
```

CODE FLOW:

# initialise the packages and libraries

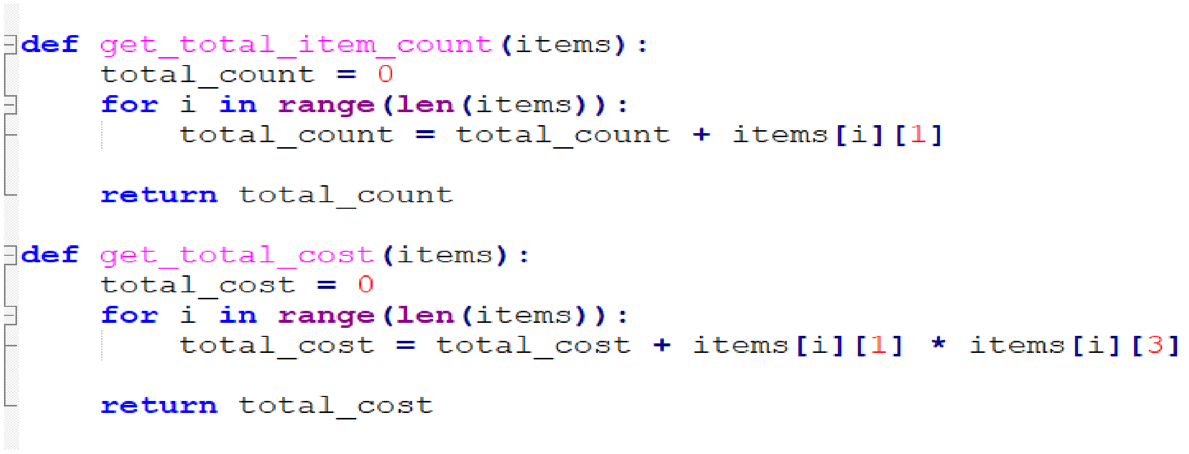


```python
import os
import sys
os.environ["PYSPARK_PYTHON"] = "/opt/cloudera/parcels/Anaconda/bin/python"
os.environ["JAVA_HOME"] = "/usr/java/jdk1.8.0_161/jre"
os.environ["SPARK_HOME"] = "/opt/cloudera/parcels/SPARK2-2.3.0.cloudera2-1.cdh5.13.3.p0.316101/lib/spark2/"
os.environ["PYLIB"] = os.environ["SPARK_HOME"] + "/python/lib"
sys.path.insert(0, os.environ["PYLIB"] +"/py4j-0.10.6-src.zip")
sys.path.insert(0, os.environ["PYLIB"] +"/pyspark.zip")


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


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

#pre define the functions for getting total items and total cost in an invoice



```python
def get_total_item_count(items):
    total_count = 0
    for i in range(len(items)):
        total_count = total_count + items[i][1]

    return total_count

def get_total_cost(items):
    total_cost = 0
    for i in range(len(items)):
        total_cost = total_cost + items[i][1] * items[i][3]

    return total_cost
```

#get the raw data from spark to schema based data frame

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


newschema = StructType() \
     .add("country",StringType()) \
     .add("invoice_no", LongType()) \
     .add("items", ArrayType(StructType()
        .add("SKU",StringType()) \
        .add("quantity",LongType()) \
        .add("title",StringType()) \
        .add("unit_price",DoubleType()))) \
     .add("timestamp",TimestampType()) \
     .add("type",StringType())

structdatastream = rawdata.select(from_json(col("value").cast(StringType()), newschema).alias("data")).select("data.*")
```

#initialise the udf and call the functions.

```
structexpand1stream =  structdatastream \
    .withColumn('total_items',add_total_item_count(structdatastream.items))

add_total_cost = udf(get_total_cost,DoubleType())

structexpand2stream = structexpand1stream \
    .withColumn('total_cost', add_total_cost(structexpand1stream.items))

structexpand2stream = structexpand2stream.withColumn('is_ordered',when(col('type') == 'ORDER',1).otherwise(0))
structexpand2stream = structexpand2stream.withColumn('is_returned',when(col('type') == 'ORDER',0).otherwise(1))

structexpand2stream= structexpand2stream.withColumn('total_cost' , when(col('type') == 'ORDER', col('total_cost')).otherwise(-1*col('total_cost')))
```

# add new columns for invoice is ordered or returned and negate the total cost if returned.

```
structexpand2stream = structexpand2stream.withColumn('is_ordered',when(col('type') == 'ORDER',1).otherwise(0))
structexpand2stream = structexpand2stream.withColumn('is_returned',when(col('type') == 'ORDER',0).otherwise(1))

structexpand2stream= structexpand2stream.withColumn('total_cost' , when(col('type') == 'ORDER', col('total_cost')).otherwise(-1*col('total_cost')))
```

# query for console stream

```
streamDFinal = structexpand2stream.select("invoice_no","country","timestamp","type","total_items","total_cost","is_ordered","is_returned")

query = streamDFinal  \
        .writeStream  \
        .outputMode("append")  \
        .format("console")  \
        .option("truncate","false") \
        .trigger(processingTime="1 minute") \
        .start()
```

# group by time window of 1 min and get the kpi values to json.

```
aggstreambytime = streamDFinal \
           .withWatermark("timestamp","1 minute") \
           .groupBy(window("timestamp","1 minute","1 minute")) \
           .agg(sum("total_cost").alias("TSV"),count("invoice_no").alias("OPM"),sum("is_returned").alias("rateofreturn")) \
           .select("window","OPM","TSV","rateofreturn")
aggstreambytime = aggstreambytime.withColumn('rateofreturn',col("rateofreturn")/col("OPM"))
aggstreambytime = aggstreambytime.withColumn('avgtranssize',col("TSV")/col("OPM"))

querybytime = aggstreambytime  \
          .writeStream  \
          .outputMode("append")  \
          .format("json")  \
          .option("truncate","false") \
          .option("path","/tmp/finaltime") \
          .option("checkpointLocation","/tmp/time") \
          .trigger(processingTime="1 minute") \
          .start()
```

#similarly group by time and country based window and get the kpi values to json.

```
aggstreambycountry = streamDFinal \
                    .withWatermark("timestamp","1 minute") \
                    .groupBy(window("timestamp","1 minute","1 minute"),"country") \
                    .agg(sum("total_cost").alias("TSV"),count("invoice_no").alias("OPM"),sum("is_returned").alias("rateofreturn")) \
                    .select("window","country","OPM","TSV","rateofreturn")

aggstreambycountry = aggstreambycountry.withColumn('rateofreturn',col("rateofreturn")/col("OPM"))


querybycountry = aggstreambycountry  \
         .writeStream  \
         .outputMode("append")  \
         .format("json")  \
         .option("truncate","false") \
         .option("path","/tmp/finalcountry") \
         .option("checkpointLocation","/tmp/country") \
         .trigger(processingTime="1 minute") \
         .start().awaitTermination()
```