aggregateByKey( ) Transformation

Syntax of this transformation

source_rdd.aggregateByKey(zeroValue, lambda1, lambda2) --> target_rdd

aggregateByKey(zeroValue, seqFunc, combFunc, numPartitions=None, partitionFunc=)

Aggregate the values of each key, using given combine functions and a neutral "zero value". This function can return a different result type, U, than the type of the values in this RDD, V. Thus, we need one operation for merging a V into a U and one operation for merging two U's. The former operation is used for merging values within a partition, and the latter is used for merging values between partitions. To avoid memory allocation, both of these functions are allowed to modify and return their first argument instead of creating a new U.

In [1]:

```
from pyspark.sql import SparkSession

spark = SparkSession \
        .builder \
        .master("local[*]") \
        .appName("aggregrate Transformation") \
        .enableHiveSupport() \
        .getOrCreate()

#path of the data file on the local machine
data_file = '/Users/vaishaliyasala/Desktop/Github/Spark/Exercise_Dependencies/sales_data.csv'

#Read the csv into a dataframe
df = spark.read.csv(data_file, header = True, )

df1 = df.select(df["InvoiceNo"],df["UnitPrice"],df["Quantity"]).repartition(4)

print(df1.printSchema())

#Creating view of the dataframe of with 3 required columns and sample of 3% of data
sample_df = df1.sample(0.02,134)

sample_df.show()
```

```
22/10/13 17:58:44 WARN Utils: Your hostname, Vaishalis-MacBook-Pro.local resolves to a loopback address: 127.0.
0.1; using 192.168.0.105 instead (on interface en0)
22/10/13 17:58:44 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
22/10/13 17:58:44 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builti
n-java classes where applicable
22/10/13 17:58:45 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
22/10/13 17:58:45 WARN Utils: Service 'SparkUI' could not bind on port 4041. Attempting port 4042.

root
 |-- InvoiceNo: string (nullable = true)
 |-- UnitPrice: string (nullable = true)
 |-- Quantity: string (nullable = true)

None
+---------+---------+--------+
|InvoiceNo|UnitPrice|Quantity|
+---------+---------+--------+
|   536464|     2.55|       1|
|   536408|     0.65|      36|
|   536412|     1.65|       5|
|   536412|     1.65|       3|
|   536464|     1.95|       1|
|   536415|     2.95|       3|
|   536399|     1.85|       6|
|   536401|     5.95|       1|
|   536409|     0.65|      12|
|   536520|      2.1|       2|
|   536409|     2.95|       1|
|   536392|      165|       1|
|   536414|        0|      56|
|   536464|     1.25|       3|
|   536420|     2.95|       6|
|   536396|     1.06|       6|
|   536520|     1.95|       5|
|   536389|     4.95|       8|
|   536446|     0.42|      10|
|   536375|     6.95|       4|
+---------+---------+--------+
only showing top 20 rows
```

In [2]:

```
# apply a map() transformation to rdd to create (K, V) pairs

#In this key-value pair, key is the InvoiceNo and the number is the value

#whereas the price is obtained from UnitPrice*Qunatity


import decimal


def get_price(x3):
    try:
        UnitPrice = decimal.Decimal(x3[2])
        convert = UnitPrice * decimal.Decimal(x3[1])
    except decimal.InvalidOperation:
           print("Invalid input")
    key = x3[0]
    price = convert
    return (key, price)


new_rdd = df1.rdd.map(lambda x : get_price(x))
print("Number of elements =",len(new_rdd.collect()))
print("Number of Partitions =",new_rdd.getNumPartitions())
```

```
[Stage 8:>                                                          (0 + 4) / 4]
Number of elements = 999
Number of Partitions = 4
```

In [3]:

```
#Showing the Result for the dataframe sample sample_df
sample_df_rdd = sample_df.rdd.map(lambda x : get_price(x))

print(sample_df_rdd.collect())
```

```
[('536464', Decimal('2.55')), ('536408', Decimal('23.40')), ('536412', Decimal('8.25')), ('536412', Decimal('4.
95')), ('536464', Decimal('1.95')), ('536415', Decimal('8.85')), ('536399', Decimal('11.10')), ('536401', Decim
al('5.95')), ('536409', Decimal('7.80')), ('536520', Decimal('4.2')), ('536409', Decimal('2.95')), ('536392', D
ecimal('165')), ('536414', Decimal('0')), ('536464', Decimal('3.75')), ('536420', Decimal('17.70')), ('536396',
Decimal('6.36')), ('536520', Decimal('9.75')), ('536389', Decimal('39.60')), ('536446', Decimal('4.20')), ('536
375', Decimal('27.80')), ('536373', Decimal('6.36')), ('536408', Decimal('9.90'))]
```

In [4]:

```
# rdd2 = rdd1.aggregateByKey(
#    zero_value,
#    lambda x,y: (x[0] + y,    x[1] + 1),
#    lambda x,y: (x[0] + y[0], x[1] + y[1])
# )

# Where the following is true about the meaning of each x and y
# pair above :
#
# First lambda expression for Within-Partition Reduction Step::
#    x: is a TUPLE that holds: (runningSum, runningCount).
#    y: is a SCALAR that holds the next Value

# Second lambda expression for Cross-Partition Reduction Step::
#    x: is a TUPLE that holds: (runningSum, runningCount).
#    y: is a TUPLE that holds: (nextPartitionsSum, nextPartitionsCount).


# we are showing for each key (invoice), U is sum of prices for all items and no of items
price_and_count = new_rdd.aggregateByKey(
        (0, 0), \
        lambda x, y: (x[0]+y, x[1]+1), \
        lambda rdd1, rdd2: (rdd1[0] + rdd2[0], rdd1[1] + rdd2[1]) \
    )


print("price_and_count.count() = ", price_and_count.count())
print("price_and_count.collect() = ", price_and_count.take(10))
```

```
[Stage 13:>                                                         (0 + 4) / 4]
price_and_count.count() =  66
price_and_count.collect() =  [('536460', (Decimal('295.54'), 14)), ('C536391', (Decimal('-141.48'), 7)), ('5364
12', (Decimal('514.41'), 81)), ('536401', (Decimal('354.23'), 64)), ('536420', (Decimal('233.45'), 14)), ('5363
85', (Decimal('130.85'), 7)), ('536397', (Decimal('279.00'), 2)), ('536416', (Decimal('225.70'), 6)), ('53650
8', (Decimal('155.52'), 2)), ('536376', (Decimal('328.80'), 2))]
```