**PySpark Assignments**

(By: Y. Kanakaraju, Created: Nov 2022, For: CTS Academy)

**Spark Core – RDD**

1. From the given Car details dataset, compute the ‘Average Weight’ of ‘American Cars’ for each ‘Make’. Do not use ‘groupBy’ transformation

The output should look like: (ford, 3540), (buick, 2800) etc.

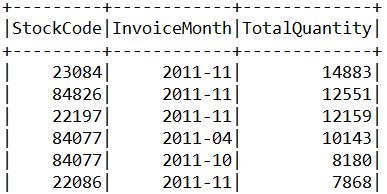
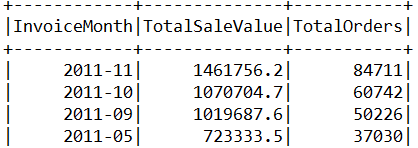
* Dataset: **cars.tsv**

**Spark SQL – DataFrames**

Solve the following two assignments (2 and 3) using the dataset:   
**online-retail-dataset.csv**

The dataset may be downloaded from the following URL: <https://archive.ics.uci.edu/ml/machine-learning-databases/00352/>

Save the excel file as CSV file.

1. Show the **top 25 products that recorded highest monthly sales** by volume in any month. The same product may appear multiple times if it recorded highest sales in different months.
   1. Data required: StockCode, InvoiceMonth and TotalQuantity
   2. A produce is identified by StockCode column.
   3. Total Quantity is the sum of Quantity in that month.
   4. Sample output:  
      
2. Fetch **TotalSaleValue and TotalOrders for each month in the year 2011**. Arrange the data in the descending order of TotalSaleValue. TotalSaleValue is the sum of all SaleValue in each month. Use DataFrame API methods only (do not use SQL queries).
   1. Data required: InvoiceMonth, TotalSaleValue, TotalOrders
   2. SaleValue is derived as UnitPrice \* Quantity
   3. Arrange the data in the descending order of TotalSaleValue.
   4. Round TotalSaleValue to one decimal place.
   5. Sample Output:  
      

**Structured Streaming**Solve the assignments 4 and 5 using Spark Structured Streaming API (do not use DStreams API).

1. Create a real time data pipeline creating a stream of words along with timestamps printed on the console reading from lines of text ingested into a socket.   
   Streaming Source: **Socket** (localhost:9999), Sink: **Console**
   1. Ingest data into socket using the netcat Linux utility (nc –lk 9999)
   2. As you write data into the socket, you should output each word with corresponding timestamp on to the console.
   3. Use ‘append’ output mode.
   4. Use trigger interval of 5 seconds.
2. Create a pipeline to ingest data into Kafka from a Rate source at a rate of 5 rows per second.  
   Streaming Source: **Rate**, Sink: **Kafka**
   1. Start zookeeper and Kafka server services.
   2. Create a Kafka topic called ‘topic1’
   3. Write a structured streaming application to write data from a Rate source into the Kafka topic. Use timestamp as key.

**Weightage:**

* Assignment 1: 15% (RDD API)
* Assignment 2: 15% (Spark SQL)
* Assignment 3: 20% (Spark SQL)
* Assignment 4: 25% (Structured Streaming)
* Assignment 5: 25% (Structured Streaming)

**Assignment Submission Guidelines**

* Please submit all the solutions in **a single text file created using Notepad**.
* Clearly mention your Associate ID the dates of the training batch you attended towards the top of the submitted file.
* Mention the assignment number followed the by source-code. Simply put all your source-code in text format.
* Separate each assignment with a horizontal line.
* No need to show/print the output.
* Even if you practiced on Jupyter Notebook or Databricks, still submit the code in a notepad file only. Just copy and paste all the code in the text file.
* Do not submit notebook files (.pynb files), word documents and image files.

**Sample submission format (for your understanding)**

Associate ID: 123456  
Dates: PySpark from 01-Nov-2022 to 10-Nov-2022

**Assignment 1:**

import pyspark

from pyspark import SparkContext

avgerage\_rdd = sc.textFile('/FileStore/tables/cars-1.tsv').map(lambda x: x.split('\t')).filter(lambda x: x[9] == 'American').map(lambda x: (x[0], int(x[6]))).mapValues(lambda x: (x,1)).reduceByKey(lambda x,y: (x[0]+y[0], x[1]+y[1])).mapValues(lambda x: x[0]/x[1])

avgerage\_rdd.collect()

**Assignment 2**

with s1 as

(

select \*,

CONCAT(YEAR(str\_to\_date(InvoiceDate,"%d-%m-%Y")),'-',MONTH(str\_to\_date(InvoiceDate,"%d-%m-%Y")))

as InvoiceMonth from onlineretail

)

select StockCode, InvoiceMonth, SUM(Quantity) over(partition by InvoiceMonth, StockCode) as TotalQuantity

from s1

order by TotalQuantity desc

limit 25;

**Assignment 3**

import pyspark

from pyspark import SparkContext

from pyspark.sql.functions import \*

df = spark.read.csv("/FileStore/tables/Online\_Retail\_CSV.csv",header = "True",inferSchema= "True").withColumn("salesvalue", round(col("Quantity")\*col("UnitPrice"),2)).withColumn("InvoiceMonth",concat(year(to\_timestamp(col("InvoiceDate"), 'dd-MM-yyyy HH:mm')),lit("-"),month(to\_timestamp(col("InvoiceDate"), 'dd-MM-yyyy HH:mm')))).groupBy("InvoiceMonth").agg(round(sum("salesvalue"),2).alias("TotalSaleValue"),count("InvoiceNo").alias("TotalOrders")).sort("TotalSaleValue",ascending = False).filter("InvoiceMonth like '2011%'")

df.show()

**Assignment 4**

from pyspark import SparkConf, SparkContext

from pyspark.streaming import StreamingContext

conf = SparkConf().setAppName("WordStream").setMaster("local[2]")

sc = SparkContext(conf=conf)

ssc = StreamingContext(sc, 5)

In this example, we are using a local cluster with 2 worker threads and a batch interval of 5 seconds.

lines = ssc.socketTextStream("localhost", 9999)

import time

def add\_timestamp(word):

return (word, time.time())

words = lines.flatMap(lambda line: line.split(" "))

words\_with\_timestamps = words.map(add\_timestamp)

ssc.start()

!nc -lk 9999

ssc.awaitTermination()

**Assignment 5**

from pyspark.sql import SparkSession

from pyspark.sql.functions import from\_json, to\_json, struct

from pyspark.sql.types import StructType, StructField, StringType, DoubleType, TimestampType

spark = SparkSession.builder.appName("KafkaWriter").getOrCreate()

schema = StructType([

StructField("timestamp", TimestampType()),

StructField("value", DoubleType())

])

rateDF = spark.readStream.format("rate").option("rowsPerSecond", 5).load()

# Transform the data and write to Kafka

query = rateDF.selectExpr("CAST(timestamp AS TIMESTAMP) as key", "to\_json(struct(\*)) AS value") \

.writeStream.format("kafka") \

.option("kafka.bootstrap.servers", "localhost:9092") \

.option("topic", "topic1") \

.option("checkpointLocation", "/tmp/checkpoint") \

.start()

query.awaitTermination()