Assignment Solution

Week12: Apache Spark - Structured API
Part-2

Spark StructuredAPIs -Assignment Solutions

Assignment 1 :

Given 2 Datasets employee.json and dept.json
We need to calculate the count of employees against each department. Use Structured APIs.

Code:

```
//Find the count of employees against each department

import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.log4j.Level
import org.apache.log4j.Logger
import org.apache.spark.sql.functions._

object Assignment1_Week12 extends App{

  //Setting the Log Level
  Logger.getLogger("org").setLevel(Level.ERROR)

  //Setting the spark conf
  val sparkConf = new SparkConf()
  sparkConf.set("spark.app.name","Assignment1_Week12")
  sparkConf.set("spark.master","local[2]")

  //Creating Spark Session
  val spark = SparkSession.builder()
  .config(sparkConf)
  .getOrCreate()

  //Load the department data into a Dataframe using dataframe reader API

   val deptDf = spark.read
  .format("json")
  .option("path","C:/TrendyTech/SparkExamples/dept.json")
  .load()

//   deptDf.show()
//   deptDf.printSchema()
```

```
//Load the employee data into a Dataframe using dataframe reader API

   val employeeDf = spark.read
.format("json")
.option("path","C:/TrendyTech/SparkExamples/employee.json")
.load()

// employeeDf.show()
// employeeDf.printSchema()

//Joining of two dataframes using left outer join, with department dataframe on left side

   val joinCondition = deptDf.col("deptid") === employeeDf.col("deptid")//join condition

   val joinType = "left" //joinType

   val joinedDf = deptDf.join(employeeDf, joinCondition, joinType) //Joining of two dataframes

//drop the ambiguous column deptid of employee dataframe,from the joined Dataframe

 val   joinedDfNew = joinedDf.drop(employeeDf.col("deptid"))

//Use first function so as to get other columns also along with aggregated columns

joinedDfNew.groupBy("deptid").agg(count("empname").as("empcount"),first("deptName").as ("deptName")).dropDuplicates("deptName").show()

spark.stop()
}
```

Output:

```
|deptid|empcount| deptName|
+------+--------+---------+
|    21|       1|       HR|
|    41|       2|      Fin|
|    51|       0|    Admin|
|    31|       1|Marketing|
|    11|       1|       IT|
```

Assignment 2

Find the top movies as shown in spark practical 18 using broadcast join. Use Dataframes or Datasets to solve it this time.

Code:

```
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.log4j.Level
import org.apache.log4j.Logger
import org.apache.spark.sql.functions._


object Assignment2_Week12 extends App {

  //Setting the Log Level
  Logger.getLogger("org").setLevel(Level.ERROR)

  //Setting the spark conf
  val sparkConf = new SparkConf()
  sparkConf.set("spark.app.name","Assignment2_Week12")
  sparkConf.set("spark.master","local[2]")

  //Creating Spark Session
  val spark = SparkSession.builder()
  .config(sparkConf)
  .getOrCreate()


  //Creation of a ratings dataframe using a case class approach

  case class Ratings(userid:Int,movieid:Int,rating:Int,timestamp:String)//create a
case-class that represents the schema

  //Creation of base RDD for ratings data
  val ratingsRDD =
spark.sparkContext.textFile("C:/TrendyTech/SparkExamples/ratings.dat")//ratings data
does not have a schema, so first loading to an RDD

  // map the RDD elements into instances of the case class

val caseClassSchemaRDD  =  ratingsRDD.map(x => x.split("::")).map(x =>
Ratings(x(0).toInt,x(1).toInt,x(2).toInt,x(3)) )
```

```scala
//Transform to a Dataframe:

import spark.implicits._

  val ratingsDf = caseClassSchemaRDD.toDF()

//   ratingsDf.show()
//   ratingsDf.printSchema()

//Creation of base RDD for movies data

  val moviesRDD =
spark.sparkContext.textFile("C:/TrendyTech/SparkExamples/movies.dat")

//defining the schema using case class

  case class Movies(movieid:Int,moviename:String,genre:String)

  val moviestransformedRDD = moviesRDD.map(line => line.split("::")).map(fields =>
Movies(fields(0).toInt,fields(1),fields(2)) )

  val  moviesNewDf  =
moviestransformedRDD.toDF().select("movieid","moviename")

  // moviesNewDf.show()
  //moviesNewDf.printSchema()


  val transformedmovieDf =   ratingsDf.groupBy("movieid")
  .agg(count("rating").as("movieViewCount"),avg("rating").as("avgMovieRating"))
  .orderBy(desc("movieViewCount"))


//transformedmovieDf.show()


    val popularMoviesDf = transformedmovieDf.filter("movieViewCount > 1000 AND
avgMovieRating > 4.5")


  // popularMoviesDf.show()
```

```
//Now we want to associate the Movie names also, so we use a broadcast join

    spark.sql("SET spark.sql.autoBroadcastJoinThreshold    = -1")

    val joinCondition = popularMoviesDf.col("movieid") ===
moviesNewDf.col("movieid") //join condition

    val joinType = "inner"                                              //type of
join

    val  finalPopularMoviesDf =
popularMoviesDf.join(broadcast(moviesNewDf),joinCondition,joinType).drop(popularM
oviesDf.col("movieid")).sort(desc("avgMovieRating")) //joining the 2 dataframes using
broadcast join where movies data is the smaller dataset


finalPopularMoviesDf.drop("movieViewCount","movieid","avgMovieRating").show(false
)

  spark.stop()



}
```

Output:

```
+----------------------------------+
|moviename                         |
+----------------------------------+
|Shawshank Redemption, The (1994)  |
|Godfather, The (1972)             |
|Usual Suspects, The (1995)        |
|Schindler's List (1993)           |
+----------------------------------+
```

Assignment 3

File A is a text file of size 1.2 GB in HDFS at location /loc/x. It contains match by match statistics of runs scored by all the batsman in the history of cricket.
File B is a text file of size 1.2 MB present in local dir /loc/y. It contains list of batsman playing in cricket world cup 2019.
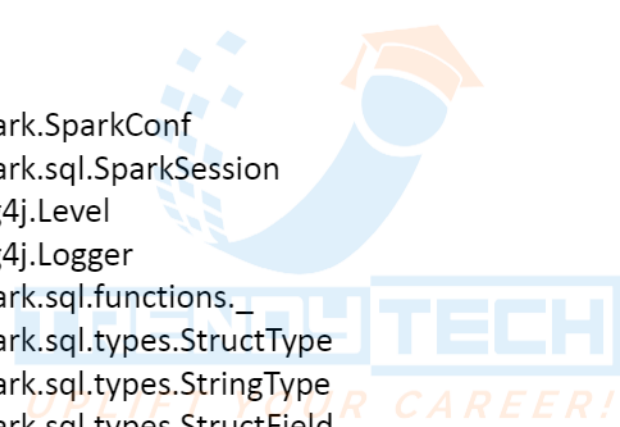
File A:
1 Rohit_Sharma India 200 100.2
1 Virat_Kohli India 100 98.02
1 Steven_Smith Aus 77 79.23
35 Clive_Lloyd WI 29 37.00
243 Rohit_Sharma India 23 150.00
243 Faf_du_Plesis SA 17 35.06
File B:
Rohit_Sharma India
Steven_Smith Aus
Virat_Kohli **India**

Find the batsman participating in 2019 who has the best average of scoring runs in his career.   Solve this using Dataframes or Datasets.

** File is tab separated.Headers not part of file

Code:

```
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.log4j.Level
import org.apache.log4j.Logger
import org.apache.spark.sql.functions._
import org.apache.spark.sql.types.StructType
import org.apache.spark.sql.types.StringType
import org.apache.spark.sql.types.StructField
import org.apache.spark.sql.Row

object Assignment3_Week12 extends App {

  //Setting the Log Level
  Logger.getLogger("org").setLevel(Level.ERROR)

  //Setting the spark conf
  val sparkConf = new SparkConf()
  sparkConf.set("spark.app.name","Assignment3_Week12")
  sparkConf.set("spark.master","local[2]")

  //Creating Spark Session
  val spark = SparkSession.builder()
  .config(sparkConf)
  .getOrCreate()
```

//Case class creation

```
case class BatsmenHistory(MatchNumber:Int,Batsman:String,Team:String,
RunsScored:Int,StrikeRate:Double)
```

//Creation of base RDD for historical data

```
val batsmenHistoryRDD =
spark.sparkContext.textFile("C:/TrendyTech/SparkExamples/FileA_BatsmenDetails_Histo
ry.txt")
```

```
val batsmenHistorySchemaRDD = batsmenHistoryRDD.map(line =>
line.split("\t")).map(fields =>
BatsmenHistory(fields(0).toInt,fields(1),fields(2),fields(3).toInt,fields(4).toDouble) )
```

// Dataframe creation

```
import spark.implicits._
```

```
val batsmenHistoryDf =    batsmenHistorySchemaRDD.toDF()
```

//batsmenHistoryDf.show()

//batsmenHistoryDf.printSchema()

//Calculating Average runs scored by a batsman in history, with highest average at top

```
val batsmenBestRunsAvgHistoryDf =
batsmenHistoryDf.groupBy("Batsman").agg(avg("RunsScored").as("AverageRunsScored"))
.select("Batsman","AverageRunsScored")
```

//batsmenBestRunsAvgHistoryDf.sort(col("AverageRunsScored").desc).show()

```
//create a base RDD from input data of worldcup
val batsmenWorldCupRDD =
spark.sparkContext.textFile("C:/TrendyTech/SparkExamples/FileB_BatsemenDetails_Wor
ldcup2019.txt")
```

//Alternative Approach instead of using case class ,though case class can also be used
instead-

//Programmatically create an explicit schema of the worldcup 2019 file:

```
val batsmenworldcupSchema = StructType(List(
      StructField("batsman",StringType,false),
      StructField("team",StringType)
      ))
```

//Convert RDD[Array(String)] to RDD[Row].

```
val batsmenWorldCupRowRDD = batsmenWorldCupRDD.map(line =>
line.split("\t")).map( fields => Row(fields(0),fields(1)))
```

//Apply the explicitly defined Struct Type schema to the RDD[Row]

```
val batsmenWorldCupDf = spark.createDataFrame(batsmenWorldCupRowRDD,
batsmenworldcupSchema)

batsmenWorldCupDf.show()
batsmenWorldCupDf.printSchema()
```

//autoBroadcast Join is turned off
```
  spark.sql("SET spark.sql.autoBroadcastJoinThreshold    = -1")

  val joinCondition = batsmenBestRunsAvgHistoryDf.col("Batsman") ===
batsmenWorldCupDf.col("batsman")

  val joinType = "inner"
```

//Using broadcast join

```
  val    finalBestBatsmenPlayingWorldCupDf =
batsmenBestRunsAvgHistoryDf.join(broadcast(batsmenWorldCupDf),joinCondition,joinT
ype).drop (batsmenBestRunsAvgHistoryDf.col("Batsman"))

finalBestBatsmenPlayingWorldCupDf.orderBy(desc("AverageRunsScored")).show()

spark.stop()

}
```

Output:
```
+-----------------+------------+
|AverageRunsScored|     batsman|
```

```
+----------------+-----------+
|           111.5|Rohit_Sharma|
|           100.0| Virat_Kohli|
|            77.0|Steven_Smith|
+----------------+-----------+
```

****