SESSION 21 ASSIGNMENT 2

**PROBLEM :**

1. **Find out the top 5 most visited destinations.**
2. **Which month has seen the most number of cancellations due to bad weather?**
3. **Top ten origins with the highest AVG departure delay.**
4. **Which route (origin & destination) has seen the maximum diversion?**

**SOLUTION :**

import org.apache.log4j.{Level, Logger}

import org.apache.spark.sql.SparkSession

object Asignment\_21\_2 extends App {

Logger.getLogger("org").setLevel(Level.OFF)

Logger.getLogger("akka").setLevel(Level.OFF)

val spark = SparkSession.builder()

.master("local")

.appName("example")

.config("spark.sql.warehouse.dir", "file:///C:")

.getOrCreate()

val sqlContext = spark.sparkContext

val delayed\_flights = sqlContext.textFile("C:/ACADGILD/Big Data/DelayedFlights.csv")

//--------------------------------------------- PROBLEM 1 ------------------------------------------------------------

//Map() operation applies to each element of RDD and it returns the result as new RDD.

//Spark RDD filter function returns a new RDD containing only the elements that satisfy a predicate.

val mapping = delayed\_flights.map(x => x.split(",")).map(x => (x(18),1)).filter(x => x.\_1!=null)

.reduceByKey(\_+\_).map(x => (x.\_2,x.\_1)).sortByKey(false).map(x => (x.\_2,x.\_1)).take(5)

println("The top 5 most visited destinations are : ")

mapping.foreach(println)

//--------------------------------------------- PROBLEM 2 ------------------------------------------------------------

//Spark RDD reduceByKey function merges the values for each key using an associative reduce function.

println("The month has seen the most number of cancellations due to bad weather : ")

val canceled = delayed\_flights.map(x => x.split(",")).filter(x => ((x(22).equals("1"))&&(x(23).equals("B"))))

.map(x => (x(2),1)).reduceByKey(\_+\_).map(x => (x.\_2,x.\_1)).sortByKey(false).take(1).foreach(println)

//--------------------------------------------- PROBLEM 3 ------------------------------------------------------------

val avg = delayed\_flights.map(x => x.split(",")).map(x => (x(17),x(16).filter(x=>x!="").toFloat)).mapValues((\_, 1))

.reduceByKey((x, y) => (x.\_1 + y.\_1, x.\_2 + y.\_2))

.mapValues{ case (sum, count) => (1.0 \* sum)/count}.map(x => (x.\_2,x.\_1))

.sortByKey(false).map(x => (x.\_2,x.\_1)).take(10)

println("Top ten origins with the highest AVG departure delay : ")

avg.foreach(println)

//--------------------------------------------- PROBLEM 4 ------------------------------------------------------------

val diversion = delayed\_flights.map(x => x.split(",")).filter(x => ((x(24).equals("1"))))

.map(x => ((x(17)+","+x(18)),1)).reduceByKey(\_+\_).map(x => (x.\_2,x.\_1))

.sortByKey(false).map(x => (x.\_2,x.\_1)).take(10)

println("The route (origin & destination) which has seen the maximum diversion : ")

diversion.foreach(println)

}

**OUTPUT :**





