ASSIGNMENT 21.2

Aviation data analysis:

The U.S. Department of Transportation's (DOT) Bureau of Transportation Statistics (BTS) tracks the on-time performance of domestic flights operated by large air carriers. Summary information on the number of on-time, delayed, cancelled, and diverted flights appears in DOT's monthly Air Travel Consumer Report, published about 30 days after the month's end, as well as in summary tables posted on this website. Summary statistics and raw data are made available to the public at the time the Air Travel Consumer Report is released.

You can download the datasets from the following links:

Delayed_Flights.csv

Delayed_Flights.csv Dataset:

There are 29 columns in this dataset. Some of them have been mentioned below:

- Year: 1987 – 2008
- Month: 1 – 12
- FlightNum: Flight number
- Canceled: Was the flight cancelled?
- CancelleationCode: The reason for cancellation.

Problem Statement 1:

Find out the top 5 most visited destinations.

Here is the Spark code in Scala to find this out:

Assignment21_2.scala:

```
// import required Spark packages

import org.apache.spark.{SparkConf, SparkContext}

object SparkTextFileOperations {

   def main(args: Array[String]): Unit = {
```

```
    val conf = new SparkConf().setAppName("SparkSampleTest").setMaster("local[*]")
    val sc = new SparkContext(conf)        // create spark context using spark configuration object
// load input data file – DelayedFlights.csv into Spark's memory
    val delayed_flights = sc.textFile("file:///E:/Acadgild/Session 21/assignment 21.2/ DelayedFlights.csv")
// split each line by comma to access required fields
    val mapping = delayed_flights.map(x => x.split(","))
                .map(x => (x (18), 1))          //get destination column values and initialize count to 1
                .filter(x => x._1! =null)       // filter out null values
                .reduceByKey (_+_)              // get sum of counts for every destination
                .map(x => (x._2, x._1))         // interchange the keys and values to get (count, dest)
                .sortByKey (false)              // sort the counts in descending order
                .map(x => (x._2, x._1))         // interchange the keys and values to get (dest , count)
                .take (5)                       // get first 5 values from result set
    mapping.foreach (println)                   // print the top 5 most visited destinations
    }
}
```

```scala
object Assignment21_2 {

  def main(args: Array[String]) {
    val conf = new SparkConf().setAppName("SparkSampleTest").setMaster("local[*]")
    val sc = new SparkContext(conf)
    val delayed_flights = sc.textFile("file:///E:/Acadgild/Session 21/assignment 21.2/DelayedFlights.csv")

    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)
    mapping.foreach(println)

  }
}
```

Output:

```
(ORD,108984)
(ATL,106898)
(DFW,70657)
(DEN,63003)
(LAX,59969)
```

Problem Statement 2:

Which month has seen the most number of cancellations due to bad weather?

Here is the Spark code snippet in Scala to find this out:

```
// load input data file – DelayedFlights.csv into Spark's memory
  val delayed_flights = sc.textFile("file:///E:/Acadgild/Session 21/assignment 21.2/ DelayedFlights.csv")
// split each line by comma to access required fields
  val mapping = delayed_flights.map(x => x.split(","))
        .filter(x => ((x (22).equals ("1")) &&(x (23).equals ("B"))))
                                    //get records that are cancelled because of bad weather
        .map(x => (x (2), 1))       // get month values from records of previous RDD
        .reduceByKey (_+_)          // compute sum of counts for every month
        .map(x => (x._2, x._1))     // interchange the keys and values to get (count, month)
        .sortByKey (false)          // sort the counts in descending order
        .map(x => (x._2, x._1))     // interchange the keys and values to get (month, count)
        .take (1)                   // get top value from result set
      mapping.foreach (println)     // print the month along having highest number of
                                    // cancellations due to bad weather
```

Assignment21_2.scala

```scala
package org.spark_samples

import org.apache.spark.{SparkConf, SparkContext}

object Assignment21_2 {

  def main(args: Array[String]) {
    val conf = new SparkConf().setAppName("SparkSampleTest").setMaster("local[*]")
    val sc = new SparkContext(conf)
    val delayed_flights = sc.textFile("file:///E:/Acadgild/Session 21/assignment 21.2/DelayedFlights.csv")

    val mapping = 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)
                  .map(x => (x._2,x._1))
                  .take(1)
    mapping.foreach(println)
  }
}
```

Output:

```
(12,250)
```

It is the month of December which has seen most number of cancellations (250) due to bad weather.

Problem Statement 3:

Find the top ten origins with the highest AVG departure delay.

Here is the Spark code snippet in Scala to find this out:

// load input data file – DelayedFlights.csv into Spark's memory

val delayed_flights = sc.textFile("file:///E:/Acadgild/Session 21/assignment 21.2/ DelayedFlights.csv")

// get header of CSV file

val header = inputDatasetWithHeader.first()

// filter out the header from base RDD of input dataset

val delayed_flights = inputDatasetWithHeader.filter(x => x!= header)

```
// split each line by comma to access required fields
   val mapping = delayed_flights.map(x => x.split(","))
       .map(x => (x (17), x (16).toDouble))                    // get origin and departure delay
       .mapValues ((_, 1))                                     // initialize count for each delay
       .reduceByKey ((x, y) => (x._1 + y._1, x._2 + y._2))     // sum up the delay counts for every origin
       .mapValues{ case (sum, count) => (1.0 * sum)/count}     // compute average of delays
       .map(x => (x._2, x._1))                 //interchange keys and values to get (avg(delay), origin)
       .sortByKey (false)                      // sort the counts in descending order
       .map(x => (x._2, x._1))                 //interchange keys and values to get (origin, avg(delay))
       .take (10)                              // get top 10 values from result set
   mapping.foreach(println)                    //print top 10 origins with highest average departure delay
```

Assignment21_2.scala

```scala
import org.apache.spark.{SparkConf, SparkContext}

object Assignment21_2 {
  def main(args: Array[String]) {
    val conf = new SparkConf().setAppName("SparkSampleTest").setMaster("local[*]")
    val sc = new SparkContext(conf)
    val inputDatasetWithHeader = sc.textFile("file:///E:/Acadgild/Session 21/assignment 21.2/DelayedFlights.csv")
    val header = inputDatasetWithHeader.first()
    val delayed_flights = inputDatasetWithHeader.filter(x => x!= header)
    val mapping = delayed_flights.map(x => x.split(","))
                  .map(x => (x(17),x(16).toDouble))
                  .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)
    mapping.foreach(println)
  }
}
```

Output:

```
(CMX,116.1470588235294)
(PLN,93.76190476190476)
(SPI,83.84873949579831)
(ALO,82.2258064516129)
(MQT,79.55665024630542)
(ACY,79.3103448275862)
(MOT,78.66165413533835)
(HHH,76.53005464480874)
(EGE,74.12891986062718)
(BGM,73.15533980582525)
```

Problem Statement 4:

Which route (origin & destination) has seen the maximum diversion?

Here is the Spark code snippet in Scala to find this out:

```
// load input data file – DelayedFlights.csv into Spark's memory

   val delayed_flights = sc.textFile("file:///E:/Acadgild/Session 21/assignment 21.2/ DelayedFlights.csv")

// get header of CSV file

   val header = inputDatasetWithHeader.first()

// filter out the header from base RDD of input dataset

val delayed_flights = inputDatasetWithHeader.filter(x => x!= header)

// split each line by comma to access required fields

   val mapping = delayed_flights.map(x => x.split(","))

        .filter(x => ((x(24).equals("1"))))              // get those records in which 'diverted' set to 1

        .map(x => ((x(17)+","+x(18)),1))                 // form (origin, dest) as key and initialize count

        .reduceByKey (_+_)                               // compute sum of counts for every diverted route

        .map(x => (x._2, x._1))                  // interchange keys and values to get (count, (origin, dest))

        .sortByKey (false)                               // sort the counts in descending order
```

.map(x => (x._2, x._1)) // interchange keys and values to get ((origin, dest), origin)

.take (10) // get top 10 values from result set

mapping.foreach(println) // print top 10 that are having maximum diversion

Assignment21_2.scala

```scala
import org.apache.spark.{SparkConf, SparkContext}

object Assignment21_2 {
  def main(args: Array[String]) {
    val conf = new SparkConf().setAppName("SparkSampleTest").setMaster("local[*]")
    val sc = new SparkContext(conf)
    val inputDatasetWithHeader = sc.textFile("file:///E:/Acadgild/Session 21/assignment 21.2/DelayedFlights.csv")
    val header = inputDatasetWithHeader.first()
    val delayed_flights = inputDatasetWithHeader.filter(x => x!= header)
    val mapping = 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)
    mapping.foreach(println)
  }
}
```

Output:

```
(ORD,LGA,39)
(DAL,HOU,35)
(DFW,LGA,33)
(ATL,LGA,32)
(SLC,SUN,31)
(ORD,SNA,31)
(MIA,LGA,31)
(BUR,JFK,29)
(HRL,HOU,28)
(BUR,DFW,25)
```