```
package ChicksWightAnalysis
import java.io._
import java.util.logging.{Level, Logger}

import scala.io.Source
import scala.math.random
import org.apache.spark._
import org.apache.spark.rdd.RDD.doubleRDDToDoubleRDDFunctions
import org.apache.spark.sql.types.{DoubleType, IntegerType,
StructField, StructType}
import org.apache.spark.sql.{DataFrame, Row, SQLContext, SparkSession}

import scala.collection


case class ChicksClass(Id: Int, chickWeight: Int, chickAge: String,
chickID: String, dietType: Int)
case class Pair(chickWeight: String,dietType: String)
//case class MeanVariancePair(mean: Double,variance: double)

object ChicksWeightByDiet extends App {
  override def main(args: Array[String]) {
    val conf2 = new SparkConf().setAppName("Spark and
SparkSql").setMaster("local")
    val sc = new SparkContext(conf2)
    sc.setLogLevel("WARN")

    val sqlContext = new org.apache.spark.sql.SQLContext(sc)


    def getDiet(in: String): String = {
      in.split("\"")(1)
    }


    var myHashMap = collection.mutable.Map[String, collection.Map[Int,
(Double, Double)]]()
    var populationMap = collection.Map[Int, (Double, Double)]()
    var hasMapForResample = collection.mutable.Map[Int,
collection.Map[Int, (Double, Double)]]()

    val myData = sc.textFile("src/main/java/ChicksWightAnalysis/
ChickWeight.csv")
    val myLines = myData.map(line => line.split(",").map(_.trim))
    val header = myLines.first()
    val chickDataWithNoHeader = myLines.filter(_ (0) != header(0))
    val populationData = chickDataWithNoHeader.map(x =>
(getDiet(x(4)), x(1))).cache()
    val cleanPopulationData = populationData.map(x => (x._1.toInt,
x._2.toDouble))
```

```
    val computation = cleanPopulationData.groupByKey().map(x => (x._1,
(x._2.count(_ => true), x._2.reduce(_ + _), x._2.map(x => x *
x).reduce((a, b) => a + b))))
    val results = computation.map(x => (x._1, (x._2._2 / x._2._1,
(x._2._3 / x._2._1 - (x._2._2 / x._2._1 * x._2._2 /
x._2._1))))).sortBy(_._1, true)
    // println("===========Printing Aggregate of the population
============")
    populationMap = results.sortBy(_._1, true).collectAsMap()
    myHashMap.put("Pupulation", populationMap)

    val writer3 = new PrintWriter(new File("src/main/java/
ChicksWightAnalysis/finalResample.txt"))
    val writer2 = new PrintWriter(new File("src/main/java/
ChicksWightAnalysis/sample.txt"))

    import org.apache.spark.sql.functions._

     for (i <- 1 to 4) {
      val sampleFromEachCategory = cleanPopulationData.filter(x =>
x._1 == i).sample(false, 0.25)
      val sampleCompute = sampleFromEachCategory.groupByKey().map(x =>
(x._1, (x._2.count(_ => true), x._2.reduce(_ + _), x._2.map(x => x *
x).reduce((a, b) => a + b))))
      val SampleResults = sampleCompute.map(x => (x._1, (x._2._2 /
x._2._1, (x._2._3 / x._2._1 - (x._2._2 / x._2._1 * x._2._2 /
x._2._1))))).sortBy(_._1, true)
      var finalSampleResults = SampleResults.sortBy(_._1,
true).collect()
       for (value <- finalSampleResults)
         writer2.println(value.toString)

      val writer1 = new PrintWriter(new File("src/main/java/
ChicksWightAnalysis/resample.txt" ))
      val resumplesNumber:Int=100
      for (j <- 1 to resumplesNumber) {
        val resampleFromEachCategory = sampleFromEachCategory.filter(x
=> x._1 == i).sample(true, 1.0)
        val resampleCompute =
resampleFromEachCategory.groupByKey().map(x => (x._1, (x._2.count(_ =>
true), x._2.reduce(_ + _), x._2.map(x => x * x).reduce((a, b) => a +
b))))
        val reSampleResults = resampleCompute.map(x => (x._1,
(x._2._2 / x._2._1, (x._2._3 / x._2._1 - (x._2._2 / x._2._1 *
x._2._2 / x._2._1))))).sortBy(_._1, true).cache()
        val toFile = reSampleResults.sortBy(_._1, true).collect()
        for (value <- toFile)
        writer1.println(value.toString)

      }
```

```
       writer1.close()
      val resampledFromFile = sc.textFile("src/main/java/
ChicksWightAnalysis/resample.txt")
      val resampleLines = resampledFromFile.map(line =>
line.split(","))
     // val resampleData = resampleLines.map(x => (x(0),
(x(1),x(2)))).cache()
      val cleanResamplesFromFile = resampleLines
        .map(
          line=>
          (
            line(0).replaceAll("\\(","").toInt,
            (
              line(1).replaceAll("\\(","").toDouble,
              line(2).replaceAll("\\)","").toDouble
            )
          )
        ).cache()

     val computeFromFile = cleanResamplesFromFile.groupByKey()
        .map(x =>
          (
            x._1,(x._2.map(y=>y._1).reduce(_+_)/
resumplesNumber,x._2.map(y=>y._2).reduce(_+_)/resumplesNumber)
          ))

       val finalToFile = computeFromFile.sortBy(_._1, true).collect()
       for (value <- finalToFile)
       writer3.println(value.toString)

    }
    writer2.close()
    val finalSampleFile = sc.textFile("src/main/java/
ChicksWightAnalysis/sample.txt")
    val finalSampleLines = finalSampleFile.map(line =>
line.split(","))
    val finalCleanSamples = finalSampleLines
      .map(
        line=>
          (
            line(0).replaceAll("\\(","").toInt,
            (
              line(1).replaceAll("\\(","").toDouble,
              line(2).replaceAll("\\)","").toDouble
            )
          )
      ).cache()

    val finalSampledData = finalCleanSamples.sortBy(_._1,
true).collectAsMap()
```

```
    myHashMap.put("Sample", finalSampledData)

    writer3.close()
    val finalResampledFile = sc.textFile("src/main/java/
ChicksWightAnalysis/finalResample.txt")
    val finalResampleLines = finalResampledFile.map(line =>
line.split(","))
     val finalCleanResamplesFromFile = finalResampleLines
      .map(
        line=>
          (
            line(0).replaceAll("\\(","").toInt,
            (
              line(1).replaceAll("\\(","").toDouble,
              line(2).replaceAll("\\)","").toDouble
            )
          )
      ).cache()

    val finalResampledData = finalCleanResamplesFromFile.sortBy(_._1,
true).collectAsMap()

    myHashMap.put("ReSample",finalResampledData)
val HashKeys = Array("Pupulation","Sample","ReSample")
    for (key <- HashKeys) {
      println("===========Printing Aggregate of the " + key + "
============")
      for (i <- 1 to 4) {
        println(i, myHashMap.get(key).get(i))
      }
    }
  }
}
```