Assignment 19.1 RDD Deep Dive

Task 1:

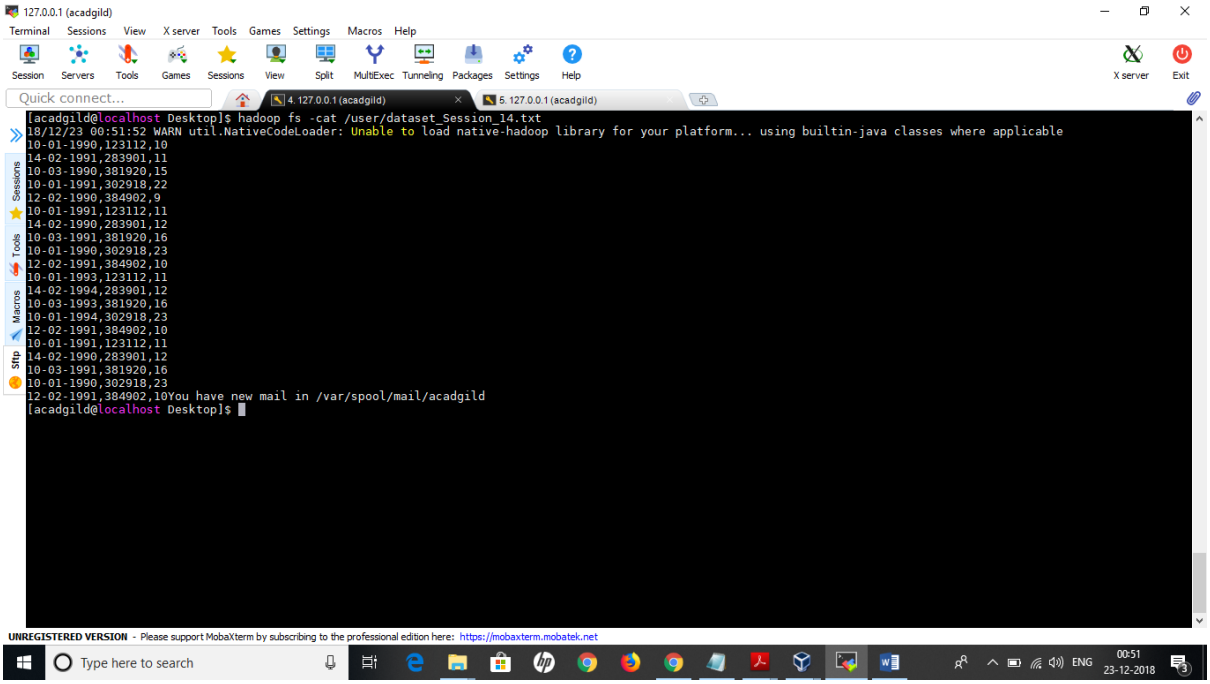
- Write a program to read a text file and print the number of rows of data in the document.

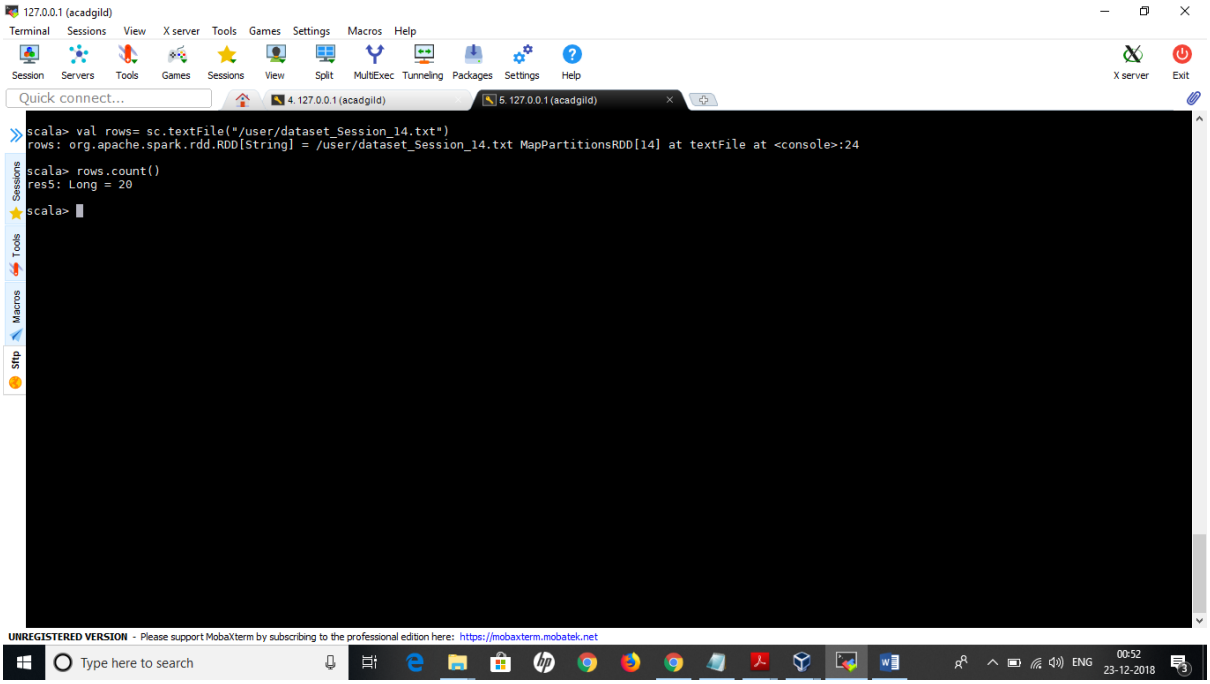
Solution:

val rows= sc.textFile("/user/dataset_Session_14.txt")

rows.count()

Output:





- Write a program to read a text file and print the number of words in the document.

Solution:

cat Spark_word_file.txt

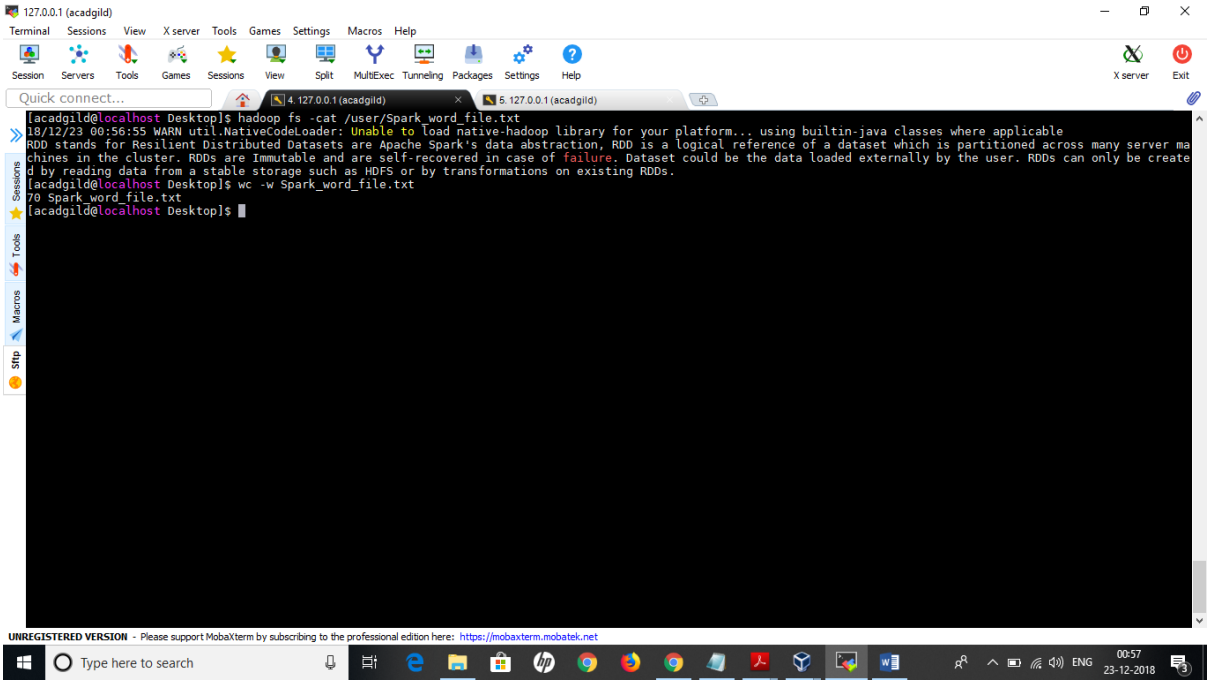
wc -w Spark_word_file.txt

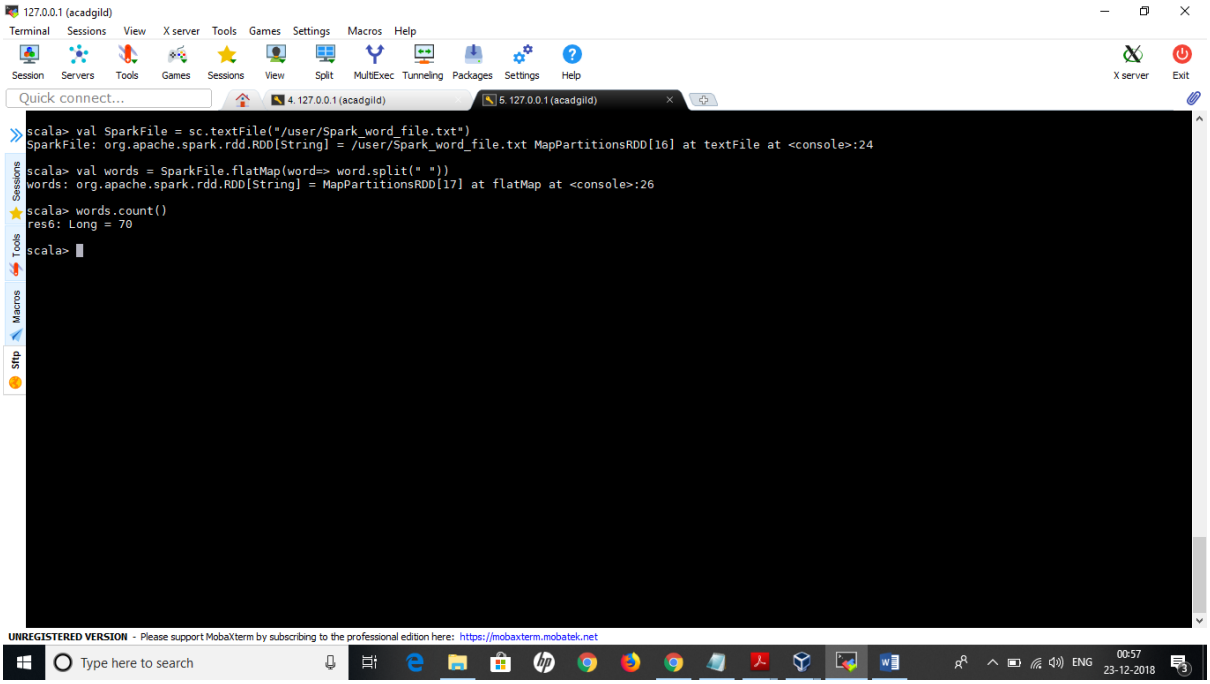
val SparkFile = sc.textFile("/user/Spark_word_file.txt")

val words = SparkFile.flatMap(word=> word.split(" "))

words.count()

Output:





- We have a document where the word separator is -, instead of space. Write a spark code, to obtain the count of the total number of words present in the document.

Solution:

cat Spark_word_file1.txt

wc -w Spark_word_file1.txt

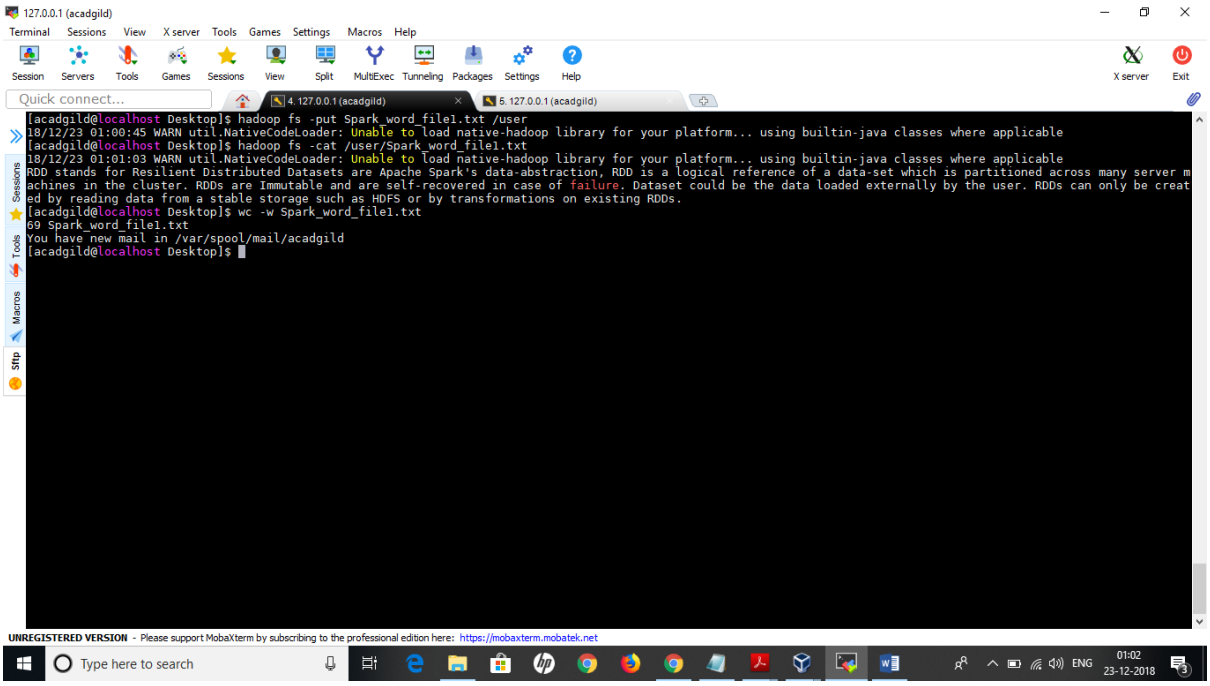
val SparkFile1 = sc.textFile("/user/Spark_word_file1.txt")

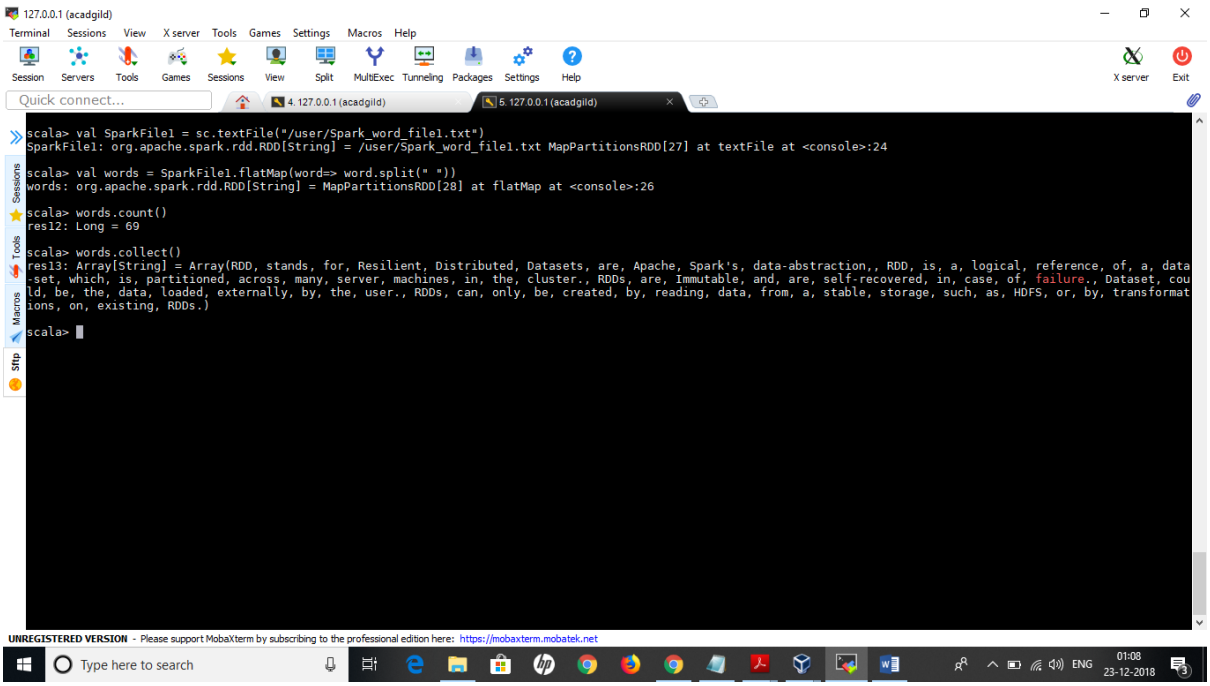
val words = SparkFile1.flatMap(word=> word.split(" "))

words.count()

words.collect()

Output:





Task 2:

Problem Statement 1:

1) Read the text file, and create a tupled rdd.

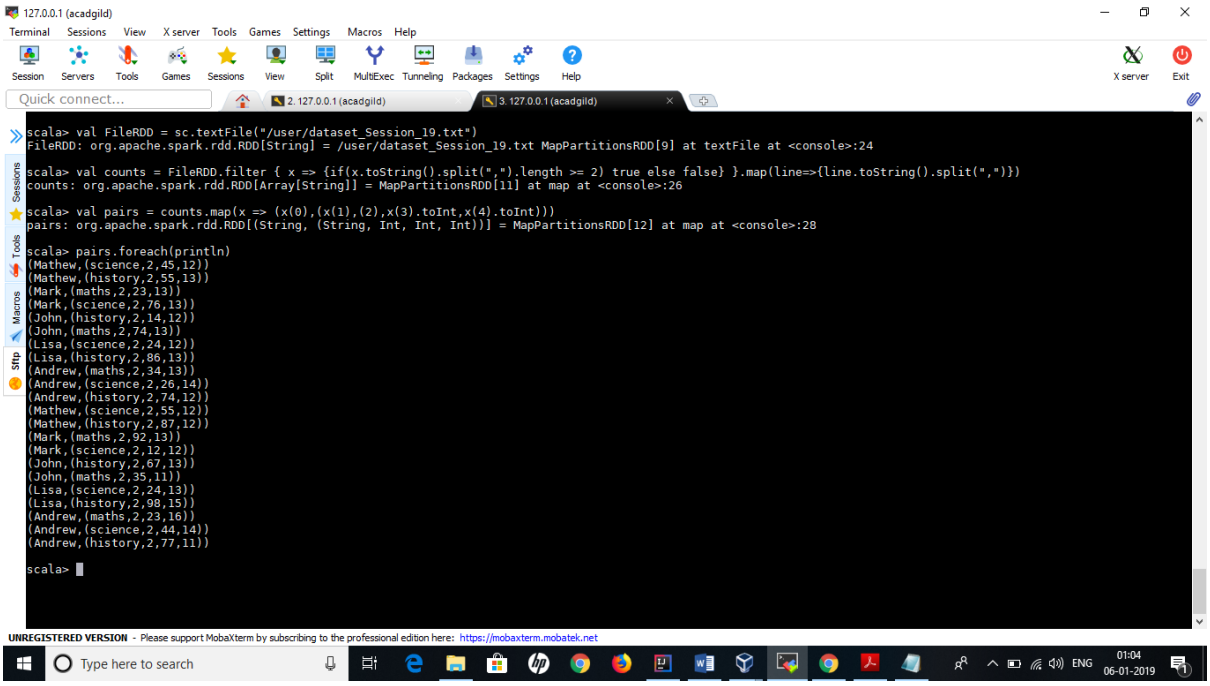
Solution:

val FileRDD = sc.textFile("/user/dataset_Session_19.txt")

val counts = FileRDD.filter { x => {if(x.toString().split(",").length >= 2) true else false} }.map(line=>{line.toString().split(",")})

val pairs = counts.map(x => (x(0),(x(1),(2),x(3).toInt,x(4).toInt)))

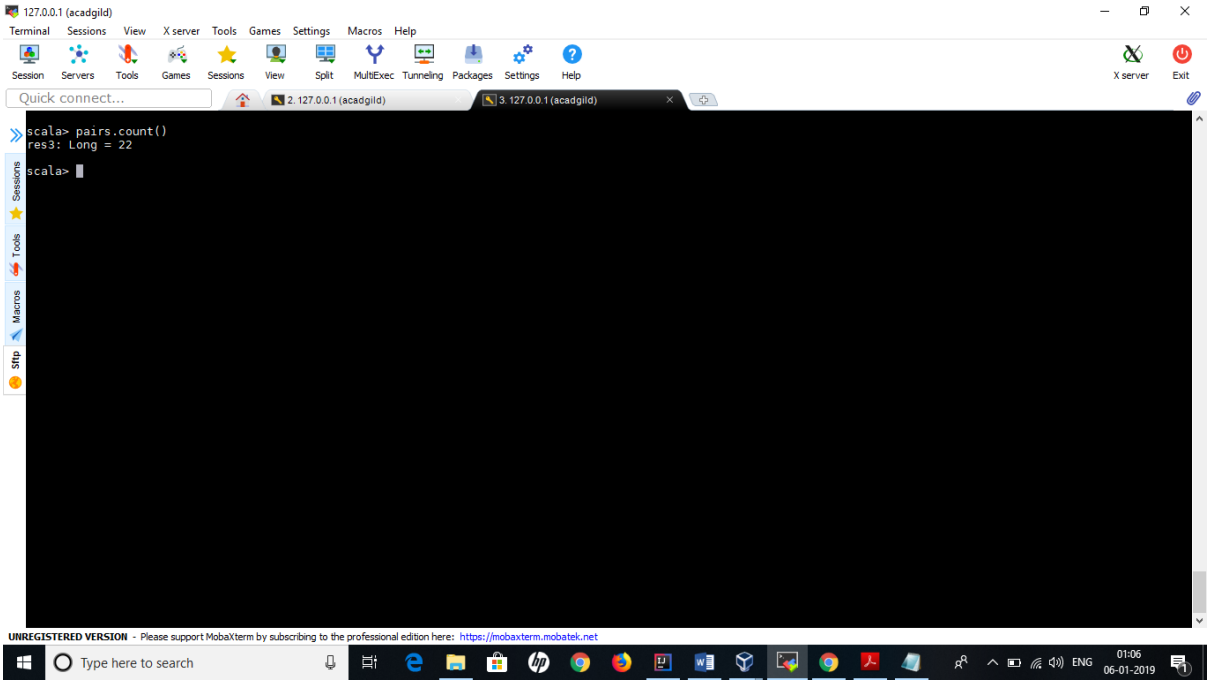
pairs.foreach(println)



2) Find the count of total number of rows present.

Solution:

pairs.count()



3) What is the distinct number of subjects present in the entire school.

Solution:

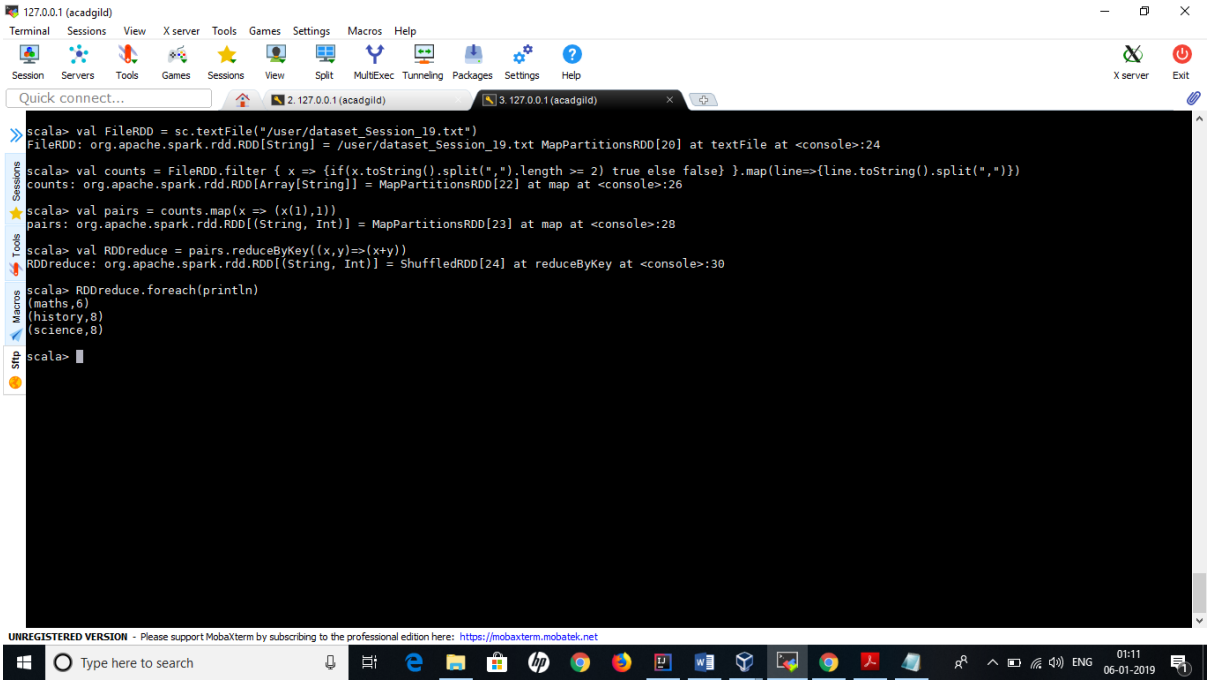
val FileRDD = sc.textFile("/user/dataset_Session_19.txt")

val counts = FileRDD.filter { x => {if(x.toString().split(",").length >= 2) true else false} }.map(line=>{line.toString().split(",")})

val pairs = counts.map(x => (x(1),1))

val RDDreduce = pairs.reduceByKey((x,y)=>(x+y))

RDDreduce.foreach(println)



4) What is the count of the number of students in school, whose name is Mathew and marks is 55.

Solution:

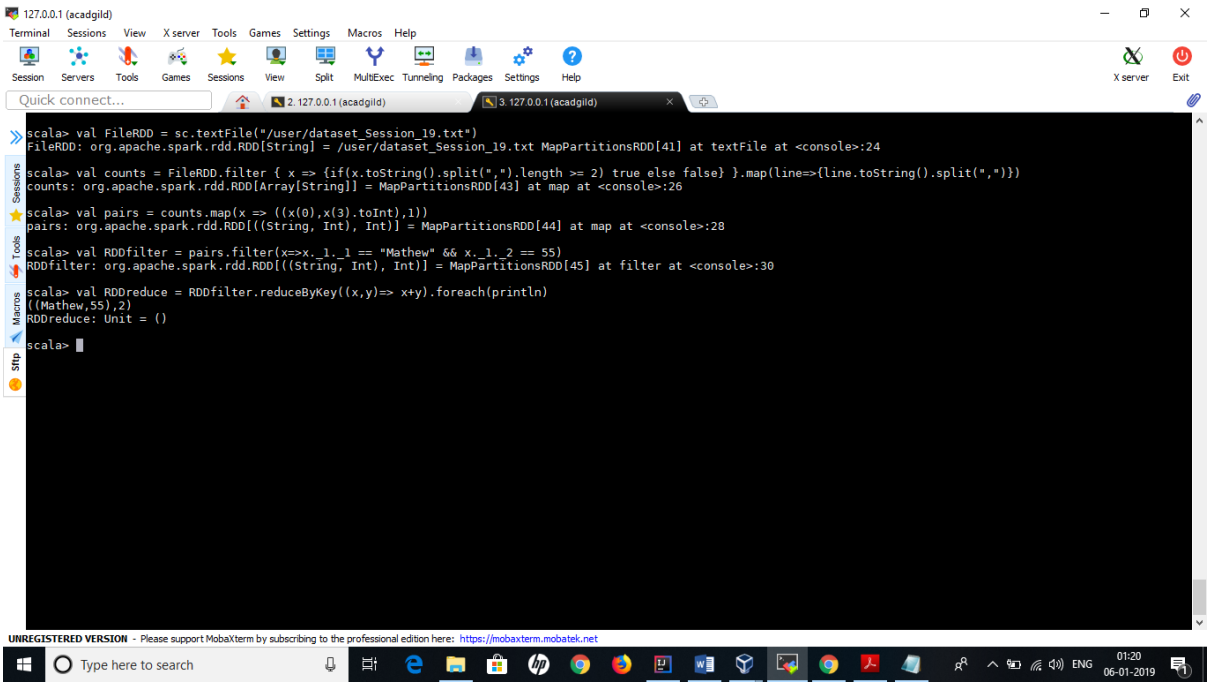
val FileRDD = sc.textFile("/user/dataset_Session_19.txt")

val counts = FileRDD.filter { x => {if(x.toString().split(",").length >= 2) true else false} }.map(line=>{line.toString().split(",")})

val pairs = counts.map(x => ((x(0),x(3).toInt),1))

val RDDfilter = pairs.filter(x=>x._1._1 == "Mathew" && x._1._2 == 55)

val RDDreduce = RDDfilter.reduceByKey((x,y)=> x+y).foreach(println)



Problem Statement 2:

1. What is the count of students per grade in the school?

Solution:

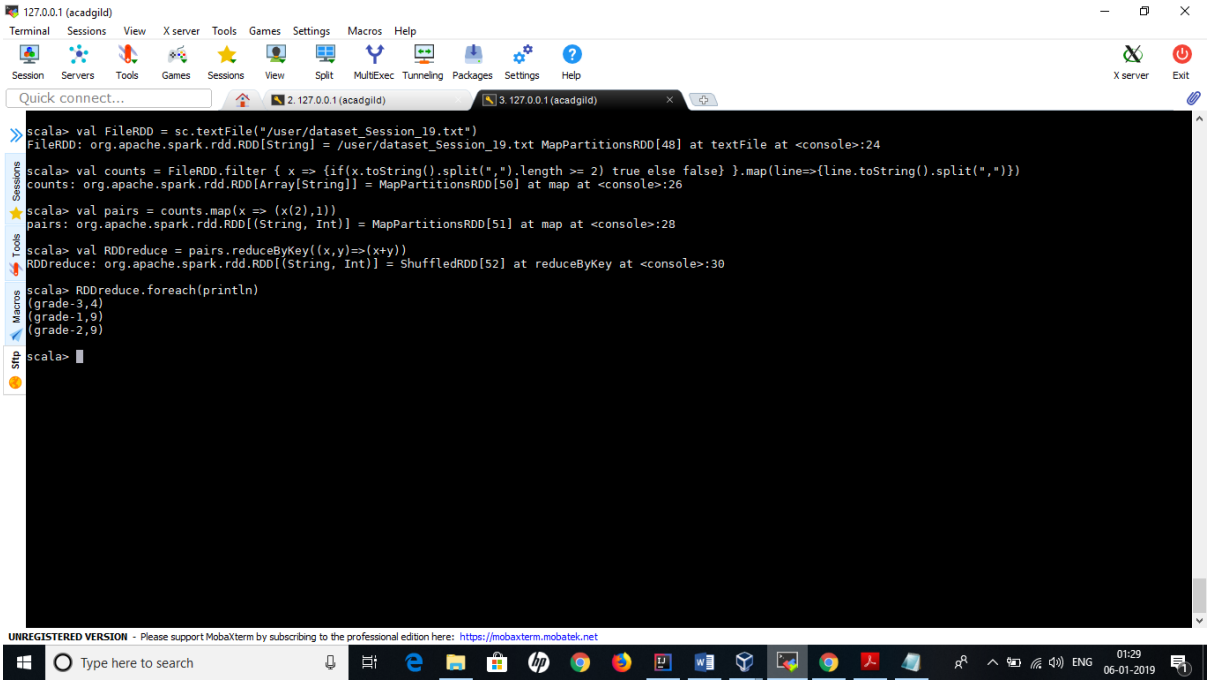
val FileRDD = sc.textFile("/user/dataset_Session_19.txt")

val counts = FileRDD.filter { x => {if(x.toString().split(",").length >= 2) true else false} }.map(line=>{line.toString().split(",")})

val pairs = counts.map(x => (x(2),1))

val RDDreduce = pairs.reduceByKey((x,y)=>(x+y))

RDDreduce.foreach(println)



2. Find the average of each student (Note - Mathew is grade-1, is different from Mathew in some other grade!)

Solution:

val FileRDD = sc.textFile("/user/dataset_Session_19.txt")

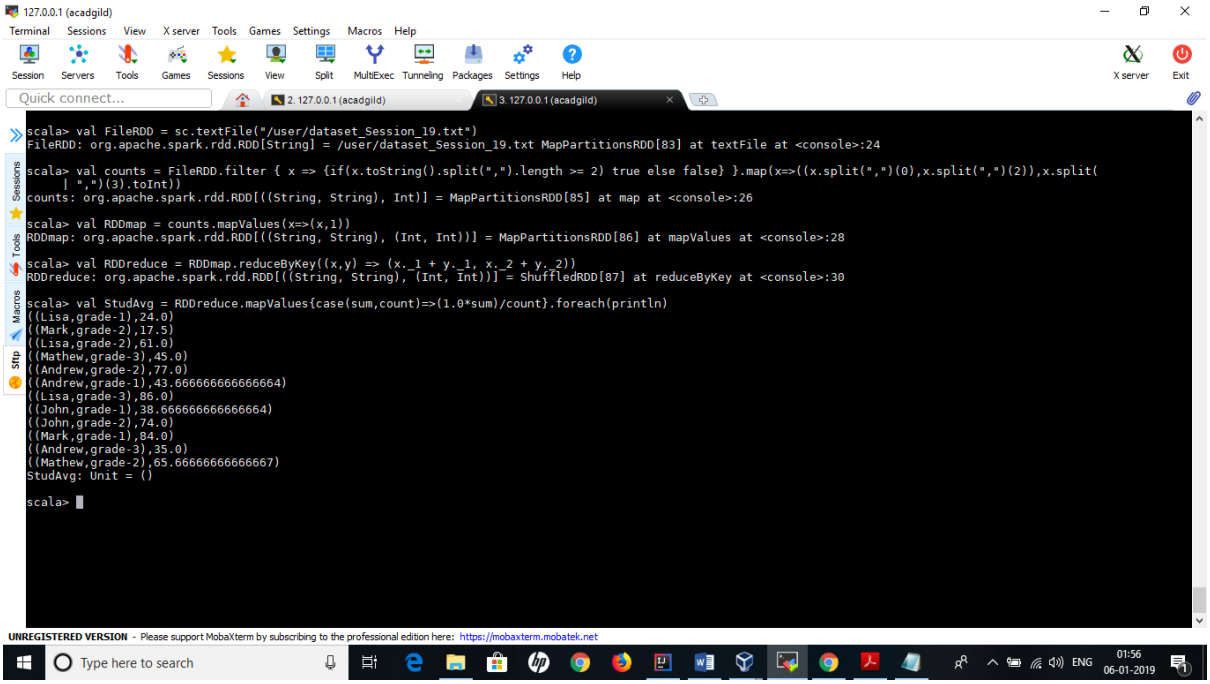
val counts = FileRDD.filter { x => {if(x.toString().split(",").length >= 2) true else false} }.map(x=>((x.split(",")(0),x.split(",")(2)),x.split(

",")(3).toInt))

val RDDmap = counts.mapValues(x=>(x,1))

val RDDreduce = RDDmap.reduceByKey((x,y) => (x._1 + y._1, x._2 + y._2))

val StudAvg = RDDreduce.mapValues{case(sum,count)=>(1.0*sum)/count}.foreach(println)



3. What is the average score of students in each subject across all grades?

Solution:

val FileRDD = sc.textFile("/user/dataset_Session_19.txt")

val counts = FileRDD.filter { x => {if(x.toString().split(",").length >= 2) true else false} }.map(x=>((x.split(",")(0),x.split(",")(1)),x.split(

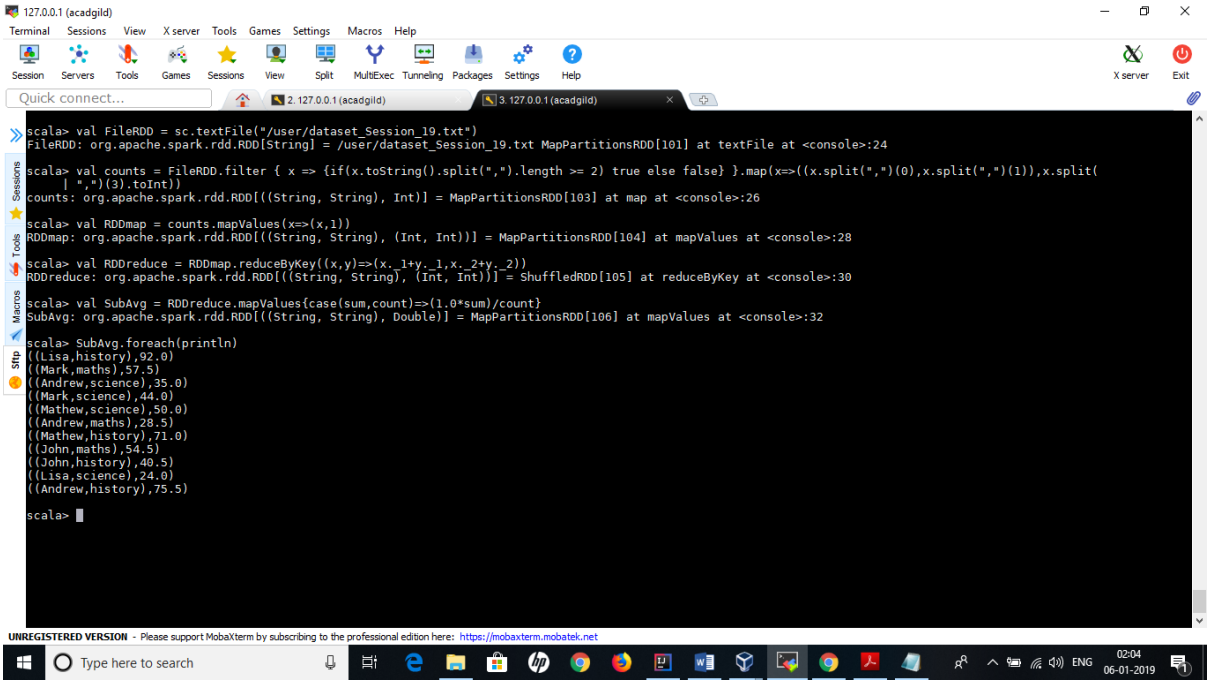
",")(3).toInt))

val RDDmap = counts.mapValues(x=>(x,1))

val RDDreduce = RDDmap.reduceByKey((x,y)=>(x._1+y._1,x._2+y._2))

val SubAvg = RDDreduce.mapValues{case(sum,count)=>(1.0*sum)/count}

SubAvg.foreach(println)



4. What is the average score of students in each subject per grade?

Solution:

val FileRDD = sc.textFile("/user/dataset_Session_19.txt")

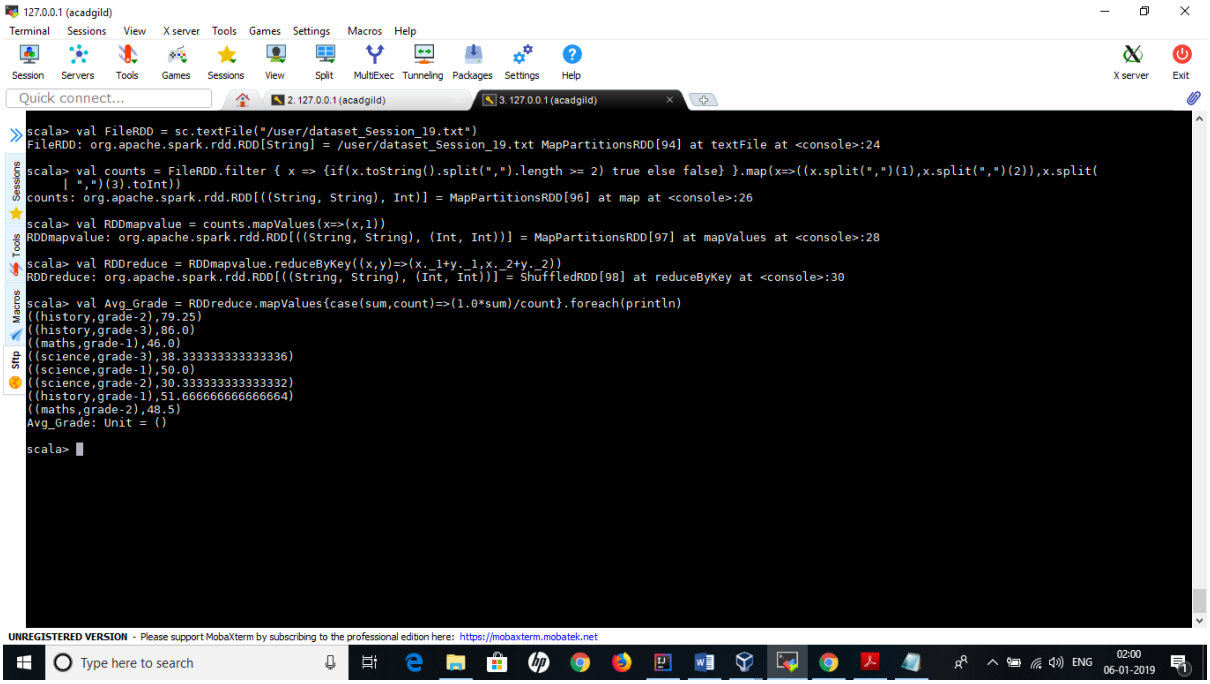
val counts = FileRDD.filter { x => {if(x.toString().split(",").length >= 2) true else false} }.map(x=>((x.split(",")(1),x.split(",")(2)),x.split(

",")(3).toInt))

val RDDmapvalue = counts.mapValues(x=>(x,1))

val RDDreduce = RDDmapvalue.reduceByKey((x,y)=>(x._1+y._1,x._2+y._2))

val Avg_Grade = RDDreduce.mapValues{case(sum,count)=>(1.0*sum)/count}.foreach(println)



5. For all students in grade-2, how many have average score greater than 50?

Solution:

val FileRDD = sc.textFile("/user/dataset_Session_19.txt")

val counts = FileRDD.filter { x => {if(x.toString().split(",").length >= 2) true else false} }.map(x=>((x.split(",")(0),x.split(",")(2)),x.split(

",")(3).toInt))

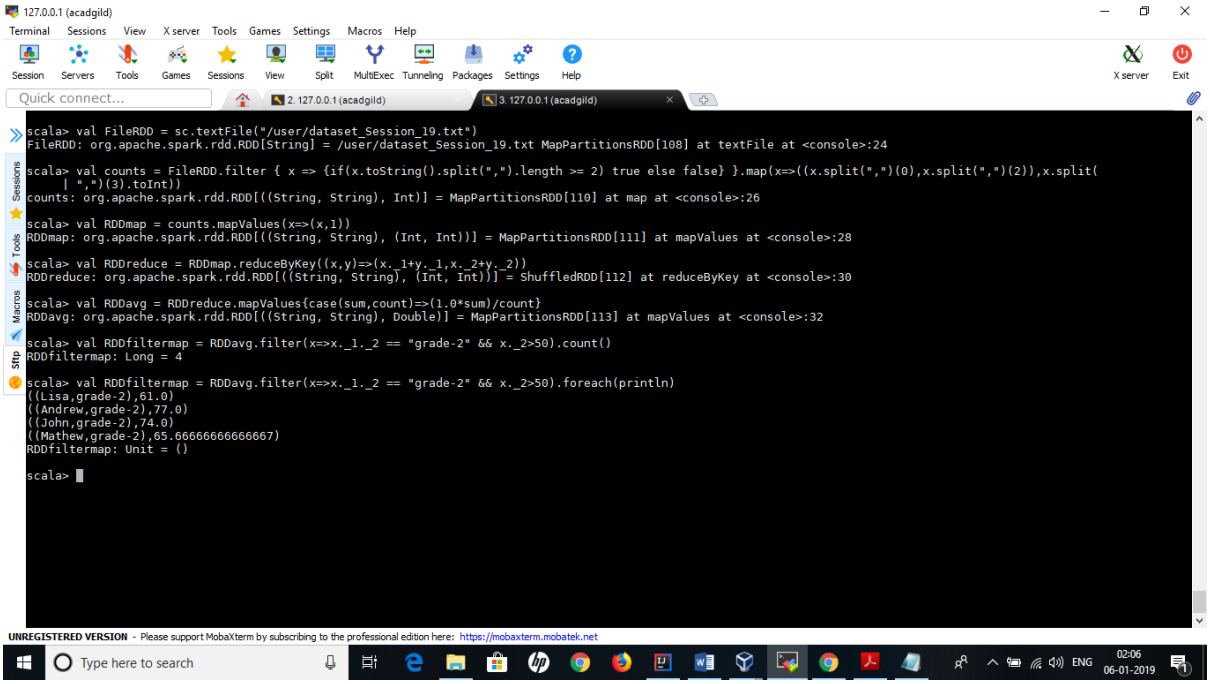
val RDDmap = counts.mapValues(x=>(x,1))

val RDDreduce = RDDmap.reduceByKey((x,y)=>(x._1+y._1,x._2+y._2))

val RDDavg = RDDreduce.mapValues{case(sum,count)=>(1.0*sum)/count}

val RDDfiltermap = RDDavg.filter(x=>x._1._2 == "grade-2" && x._2>50).count()

val RDDfiltermap = RDDavg.filter(x=>x._1._2 == "grade-2" && x._2>50).foreach(println)



Problem Statement 3:

Are there any students in the college that satisfy the below criteria:

1. Average score per student_name across all grades is same as average score per student_name per grade

Solution:

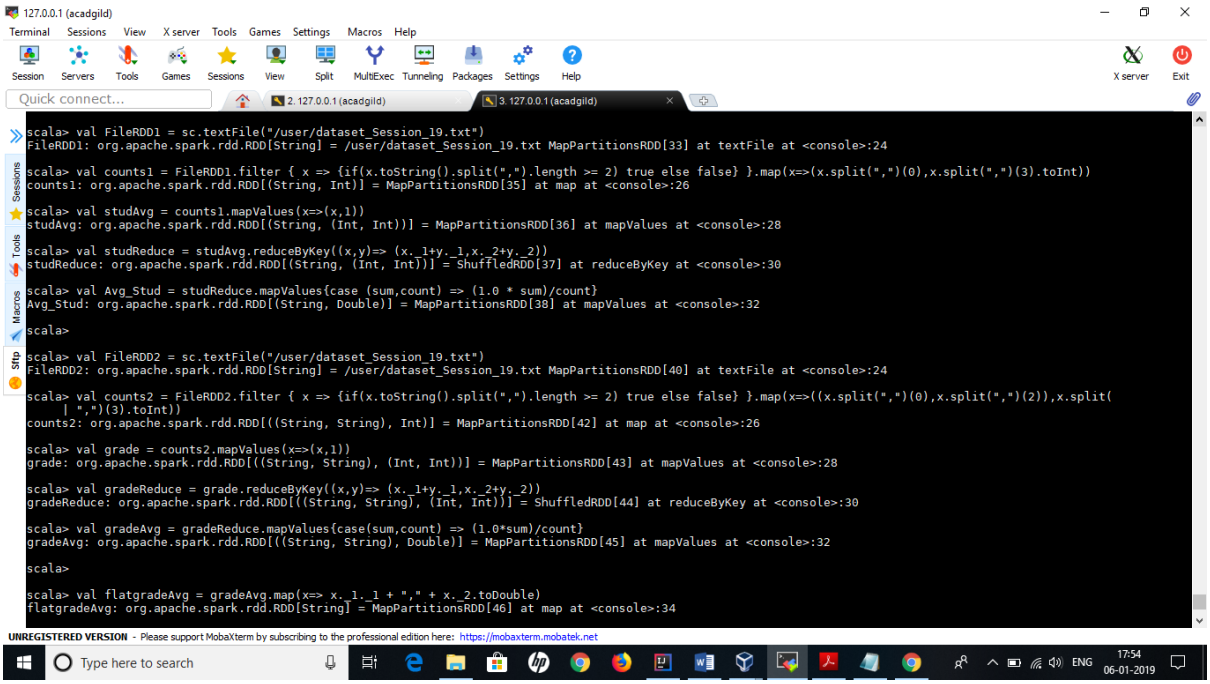
```
val FileRDD1 = sc.textFile("/user/dataset_Session_19.txt")

val counts1 = FileRDD1.filter { x => {if(x.toString().split(",").length >= 2) true else false}
}.map(x=>(x.split(",")(0),x.split(",")(3).toInt))

val studAvg = counts1.mapValues(x=>(x,1))

val studReduce = studAvg.reduceByKey((x,y)=> (x._1+y._1,x._2+y._2))

val Avg_Stud = studReduce.mapValues{case (sum,count) => (1.0 * sum)/count}


val FileRDD2 = sc.textFile("/user/dataset_Session_19.txt")

val counts2 = FileRDD2.filter { x => {if(x.toString().split(",").length >= 2) true else false}
}.map(x=>((x.split(",")(0),x.split(",")(2)),x.split(

",")(3).toInt))

val grade = counts2.mapValues(x=>(x,1))

val gradeReduce = grade.reduceByKey((x,y)=> (x._1+y._1,x._2+y._2))

val gradeAvg = gradeReduce.mapValues{case(sum,count) => (1.0*sum)/count}
```

val flatgradeAvg = gradeAvg.map(x=> x._1._1 + "," + x._2.toDouble)

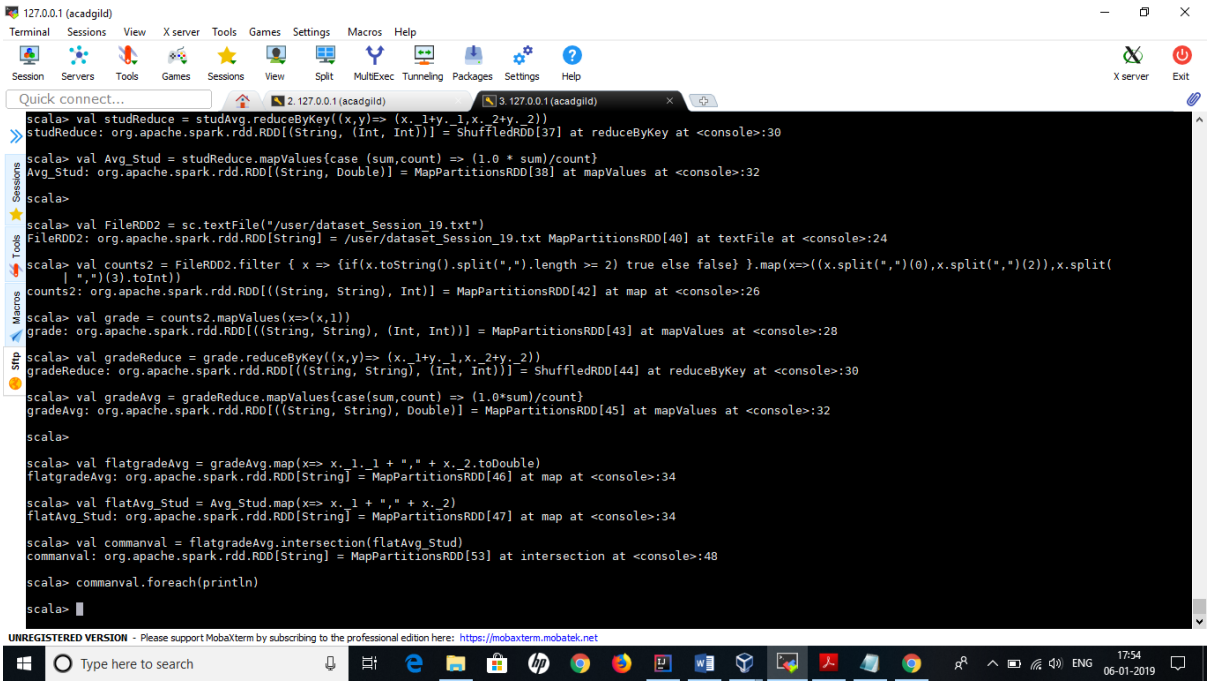
val flatAvg_Stud = Avg_Stud.map(x=> x._1 + "," + x._2)

val commanval = flatgradeAvg.intersection(flatAvg_Stud)

commanval.foreach(println)



```
scala> val FileRDD1 = sc.textFile("/user/dataset_Session_19.txt")
FileRDD1: org.apache.spark.rdd.RDD[String] = /user/dataset_Session_19.txt MapPartitionsRDD[33] at textFile at <console>:24

scala> val counts1 = FileRDD1.filter { x => {if(x.toString().split(",").length >= 2) true else false} }.map(x=>(x.split(",")(0),x.split(",")(3).toInt))
counts1: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[35] at map at <console>:26

scala> val studAvg = counts1.mapValues(x=>(x,1))
studAvg: org.apache.spark.rdd.RDD[(String, (Int, Int))] = MapPartitionsRDD[36] at mapValues at <console>:28

scala> val studReduce = studAvg.reduceByKey((x,y)=> (x._1+y._1,x._2+y._2))
studReduce: org.apache.spark.rdd.RDD[(String, (Int, Int))] = ShuffledRDD[37] at reduceByKey at <console>:30

scala> val Avg_Stud = studReduce.mapValues{case (sum,count) => (1.0 * sum)/count}
Avg_Stud: org.apache.spark.rdd.RDD[(String, Double)] = MapPartitionsRDD[38] at mapValues at <console>:32

scala>

scala> val FileRDD2 = sc.textFile("/user/dataset_Session_19.txt")
FileRDD2: org.apache.spark.rdd.RDD[String] = /user/dataset_Session_19.txt MapPartitionsRDD[40] at textFile at <console>:24

scala> val counts2 = FileRDD2.filter { x => {if(x.toString().split(",").length >= 2) true else false} }.map(x=>((x.split(",")(0),x.split(",")(2)),x.split(
     | ",")(3).toInt))
counts2: org.apache.spark.rdd.RDD[((String, String), Int)] = MapPartitionsRDD[42] at map at <console>:26

scala> val grade = counts2.mapValues(x=>(x,1))
grade: org.apache.spark.rdd.RDD[((String, String), (Int, Int))] = MapPartitionsRDD[43] at mapValues at <console>:28

scala> val gradeReduce = grade.reduceByKey((x,y)=> (x._1+y._1,x._2+y._2))
gradeReduce: org.apache.spark.rdd.RDD[((String, String), (Int, Int))] = ShuffledRDD[44] at reduceByKey at <console>:30

scala> val gradeAvg = gradeReduce.mapValues{case(sum,count) => (1.0*sum)/count}
gradeAvg: org.apache.spark.rdd.RDD[((String, String), Double)] = MapPartitionsRDD[45] at mapValues at <console>:32

scala>

scala> val flatgradeAvg = gradeAvg.map(x=> x._1._1 + "," + x._2.toDouble)
flatgradeAvg: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[46] at map at <console>:34
```



```
scala> val studReduce = studAvg.reduceByKey((x,y)=> (x._1+y._1,x._2+y._2))
studReduce: org.apache.spark.rdd.RDD[(String, (Int, Int))] = ShuffledRDD[37] at reduceByKey at <console>:30

scala> val Avg_Stud = studReduce.mapValues{case (sum,count) => (1.0 * sum)/count}
Avg_Stud: org.apache.spark.rdd.RDD[(String, Double)] = MapPartitionsRDD[38] at mapValues at <console>:32

scala>

scala> val FileRDD2 = sc.textFile("/user/dataset_Session_19.txt")
FileRDD2: org.apache.spark.rdd.RDD[String] = /user/dataset_Session_19.txt MapPartitionsRDD[40] at textFile at <console>:24

scala> val counts2 = FileRDD2.filter { x => {if(x.toString().split(",").length >= 2) true else false} }.map(x=>((x.split(",")(0),x.split(",")(2)),x.split(
     | ",")(3).toInt))
counts2: org.apache.spark.rdd.RDD[((String, String), Int)] = MapPartitionsRDD[42] at map at <console>:26

scala> val grade = counts2.mapValues(x=>(x,1))
grade: org.apache.spark.rdd.RDD[((String, String), (Int, Int))] = MapPartitionsRDD[43] at mapValues at <console>:28

scala> val gradeReduce = grade.reduceByKey((x,y)=> (x._1+y._1,x._2+y._2))
gradeReduce: org.apache.spark.rdd.RDD[((String, String), (Int, Int))] = ShuffledRDD[44] at reduceByKey at <console>:30

scala> val gradeAvg = gradeReduce.mapValues{case(sum,count) => (1.0*sum)/count}
gradeAvg: org.apache.spark.rdd.RDD[((String, String), Double)] = MapPartitionsRDD[45] at mapValues at <console>:32

scala>

scala> val flatgradeAvg = gradeAvg.map(x=> x._1._1 + "," + x._2.toDouble)
flatgradeAvg: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[46] at map at <console>:34

scala> val flatAvg_Stud = Avg_Stud.map(x=> x._1 + "," + x._2)
flatAvg_Stud: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[47] at map at <console>:34

scala> val commanval = flatgradeAvg.intersection(flatAvg_Stud)
commanval: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[53] at intersection at <console>:48

scala> commanval.foreach(println)

scala>
```