TASK -1

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

```
scala> val text = sc.textFile("/home/acadgild/Desktop/SPark Assignments/RDD_Deep_Dive.txt")
text: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[71] at textFile at <console>:27

scala> text.count
res8: Long = 23
```

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

```
scala> val total_words = text.flatMap(x => (x.split(","))).count
total_words: Long = 111
```

3. **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.**

TASK 2

PROBLEM STATEMENT 1:

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

```
val text = sc.textFile("/home/acadgild/Desktop/SPark Assignments/RDD_Deep_Dive.txt")
val tpl = text.map(x => x.split(",")).map(ar => (ar(0), ar(1), ar(2), ar(3), ar(4)))
```

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

```
scala> tpl.count
res10: Long = 22
```

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

First split each line (record) from given text file and get the column with only subects

```
val subjects = text.map(x => x.toString().split(",")).map(sub => (sub(1)))
```

Print only distinct subjects

```
scala> subjects.distinct().count
res18: Long = 3
```

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

```
val lines = text.map(x => x.toString().split(","))
```

```
scala> val filter_name = lines.filter(col => (col(0).equalsIgnoreCase("Mathew") && col(3).toInt==55))
filter_name: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[44] at filter at <console>:31

scala> filter_name.collect
res19: Array[Array[String]] = Array(Array(Mathew, history, grade-2, 55, 13), Array(Mathew, science, grade-2, 55, 12))

scala> val filter_name = lines.filter(col => (col(0).equalsIgnoreCase("Mathew") && col(3).toInt==55)).count
filter_name: Long = 2
```

PROBLEM STATEMENT 2:

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

```
val lines = text.map(x =>x.split(","))
```

Fetch only grade column and count all the students who have similar grades

```
val grades = lines.map(col => (col(2))).map(word => (word,1)).reduceByKey( _ + _ )
```

```
scala> grades.collect
res21: Array[(String, Int)] = Array((grade-3,4), (grade-1,9), (grade-2,9))
```

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

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

LOAD

val text = sc.textFile("/home/acadgild/Desktop/Spark Assignments/RDD_Deep_Dive.txt")

Filter the data which contains less than 5 column to avoid ArrayOutOfBound Exception

val col_RDD = text.filter(x => (x.toString.split(",").length>=5)).map(x => x.toString().split(",")).map(col => (col(1),col(3).toInt))

Now find the average score in each subjects

val processedRDD = col_RDD.groupByKey.mapValues{iterator => {(iterator.sum) / (iterator.size)}}

Note:

Here mapValues operates only on values, where as Map operates on entire record.

Result:

```
scala> processedRDD.collect
res11: Array[(String, Int)] = Array((maths,46), (history,69), (science,38))
```

---

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

Step-1:

val text = sc.textFile("/home/acadgild/Desktop/Acadgild_Spark_Assignments/RDD's deep dive.txt")

Step-2:

val col_RDD = text.filter(x => (x.toString.split(",").length>=5)).map(x => x.toString().split(",")).map(col => (col(1),col(3).toInt))

Step -3: Here make (Subject, grade) as key and (Marks as value)

val avg_per_subject_per_grade = col_RDD.groupByKey.mapValues(avg => (avg.sum) / (avg.size))

Result:

```
scala> avg_per_subject_per_grade.foreach(println)
((history,grade-2),79.25)
((history,grade-3),86.0)
((maths,grade-1),46.0)
((science,grade-3),38.333333333333336)
((science,grade-1),50.0)
((science,grade-2),30.333333333333332)
((history,grade-1),51.666666666666664)
((maths,grade-2),48.5)
```

---

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

Step-1:

val text = sc.textFile("/home/acadgild/Desktop/Acadgild_Spark_Assignments/RDD's deep dive.txt")

Step-2:

val col_RDD = text.filter(x => (x.toString.split(",").length>=5)).map(x => x.toString().split(","))

Step-3: Here filter the records which contains students with grade-2 only.

Key – (Student, Grade-2)

Value – Marks

```
val col_grade2 = col_RDD.filter(col => {col(2).equalsIgnoreCase("grade-2") ||
col(2).equalsIgnoreCase("grade 2")}).map(kv => ((kv(0),kv(2)), kv(3) .toDouble ))
```

Step-4: First find the average and then filter the result whose average is greater than 50.

```
val avgValue_above50 = col_grade2.groupByKey.mapValues(av => ((av.sum) / (av.size))).filter(res =>
(res._2 > 50.0))
```

Result:

```
scala> avgValue_above50.foreach(println)
((Lisa,grade-2),61.0)
((Andrew,grade-2),77.0)
((John,grade-2),74.0)
((Mathew,grade-2),65.66666666666667)
```

Problem Statement 3:

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

Step-1:

```
val text = sc.textFile("/home/acadgild/Desktop/Acadgild_Spark_Assignments/RDD's deep dive.txt")
```

Step-2:

```
val   col_RDD = text.filter(x => (x.toString.split(",").length>=5)).map(x => x.toString().split(","))
```

Step-3:

First find out Average score per student across all grades

```
val   avg_acore_per_student = col_RDD.map(col =>
(col(0),col(3).toDouble)).groupByKey.mapValues(av => ((av.sum)/(av.size)))
```

Result:

```
Array((Mark,50.75), (Andrew,46.333333333333336), (Mathew,60.5), (John,47.5), (Lisa,58.0))
```

Find out average score per student_name per grade

```
val   avg_acore_per_student_perGrade  =  col_RDD.map(col => ((col(0),
col(2)),col(3).toDouble)).groupByKey.mapValues(av => ((av.sum)/(av.size)))
```

Result:

```
((Lisa,grade-1),24.0)
((Mark,grade-2),17.5)
((Lisa,grade-2),61.0)
((Mathew,grade-3),45.0)
((Andrew,grade-2),77.0)
((Andrew,grade-1),43.666666666666664)
((Lisa,grade-3),86.0)
((John,grade-1),38.666666666666664)
((John,grade-2),74.0)
((Mark,grade-1),84.0)
((Andrew,grade-3),35.0)
((Mathew,grade-2),65.66666666666667)
```

Step-4: Now compare both and return if any average values are same.

```
val result = (avg_acore_per_student.values).intersection(avg_acore_per_student_perGrade.values)
```

Result:

```
scala> result.collect
res6: Array[Double] = Array()
```

This returns an empty array.