**Big Data Analytics-Lab-CSE6034\_Lab-Assessment - 5:**

**Implementation of Programming on Scala**

**( ( i) Word Count, ii)RDD, iii)DDL )**

**Submitted By: 20MAI0001 - NIHARIKA MAITRA**

**Git Repo link :**

<https://github.com/Niharika-20-MAI-01/Winter-2021-Big-Data-Analytics-LabCSE6034-Niharika-20MAI01>

**Activity - 1 : Write a scala program, to implement word count from a document.**

**Scala Program Code and the corresponding Output obtained on executing the “Word Count” Program:**

val filePath = "dbfs:/databricks-datasets/SPARK\_README.md" // path in Databricks File System

val lines = sc.textFile(filePath) // read the file into the cluster

lines.take(10).mkString("\n") // display first 10 lines in the file

(1) Spark Jobs

filePath: String = dbfs:/databricks-datasets/SPARK\_README.md lines: org.apache.spark.rdd.RDD[String] = dbfs:/databricks-datasets/SPARK\_README.md MapPartitionsRDD[3] at textFile at command-3691639510777405:2 res3: String = # Apache Spark Spark is a fast and general cluster computing system for Big Data. It provides high-level APIs in Scala, Java, Python, and R, and an optimized engine that supports general computation graphs for data analysis. It also supports a rich set of higher-level tools including Spark SQL for SQL and DataFrames, MLlib for machine learning, GraphX for graph processing, and Spark Streaming for stream processing. <http://spark.apache.org/>

Command took 1.14 seconds -- by maitraniharika@gmail.com at 21/5/2021, 2:29:18 pm on My Cluster

val numPartitions = lines.partitions.length // get the number of partitions

println(s"Number of partitions (workers) storing the dataset = $numPartitions")

Number of partitions (workers) storing the dataset = 2 numPartitions: Int = 2

Command took 0.49 seconds -- by maitraniharika@gmail.com at 21/5/2021, 2:29:25 pm on My Cluster

val words = lines.flatMap(x => x.split(' ')) // split each line into a list of words

words.take(10).mkString("\n") // display the first 10 words

(1) Spark Jobs

words: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[5] at flatMap at command-3691639510777407:1 res6: String = # Apache Spark Spark is a fast and general

Command took 1.27 seconds -- by maitraniharika@gmail.com at 21/5/2021, 2:36:47 pm on My Cluster

val stopWords = Seq("","a","\*","and","is","of","the","a") // define the list of stop words

val filteredWords = words.filter(x => !stopWords.contains(x.toLowerCase())) // filter the words

filteredWords.take(10).mkString("\n") // display the first 10 filtered words

(1) Spark Jobs

stopWords: Seq[String] = List("", a, \*, and, is, of, the, a) filteredWords: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[6] at filter at command-3691639510777408:2 res7: String = # Apache Spark Spark fast general cluster computing system for

Command took 1.17 seconds -- by maitraniharika@gmail.com at 21/5/2021, 2:37:26 pm on My Cluster

filteredWords.cache() // cache filtered dataset into memory across the cluster worker nodes

filteredWords.count() // materialize the cache

(1) Spark Jobs

res8: Long = 387

Command took 0.95 seconds -- by maitraniharika@gmail.com at 21/5/2021, 2:38:11 pm on My Cluster

val word1Tuples = filteredWords.map(x => (x, 1)) // map the words into (word,1) tuples

word1Tuples.take(10).mkString("\n") // display the (word,1) tuples

(1) Spark Jobs

word1Tuples: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[7] at map at command-3691639510777411:1 res9: String = (#,1) (Apache,1) (Spark,1) (Spark,1) (fast,1) (general,1) (cluster,1) (computing,1) (system,1) (for,1)

Command took 0.77 seconds -- by maitraniharika@gmail.com at 21/5/2021, 2:38:44 pm on My Cluster

val wordCountTuples = word1Tuples.reduceByKey{case (x, y) => x + y} // aggregate counts for each word

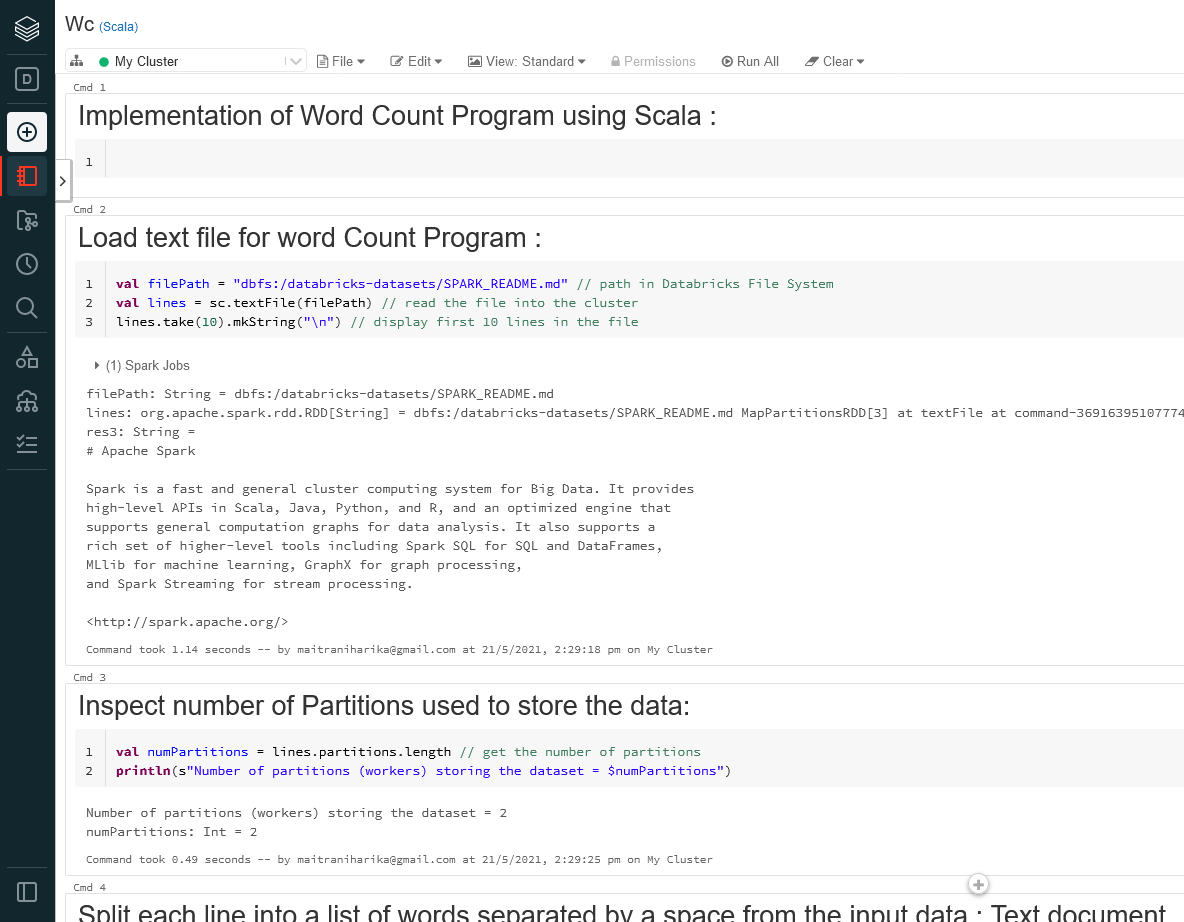
wordCountTuples.take(10).mkString("\n") // display the first 10 (word,count) tuples

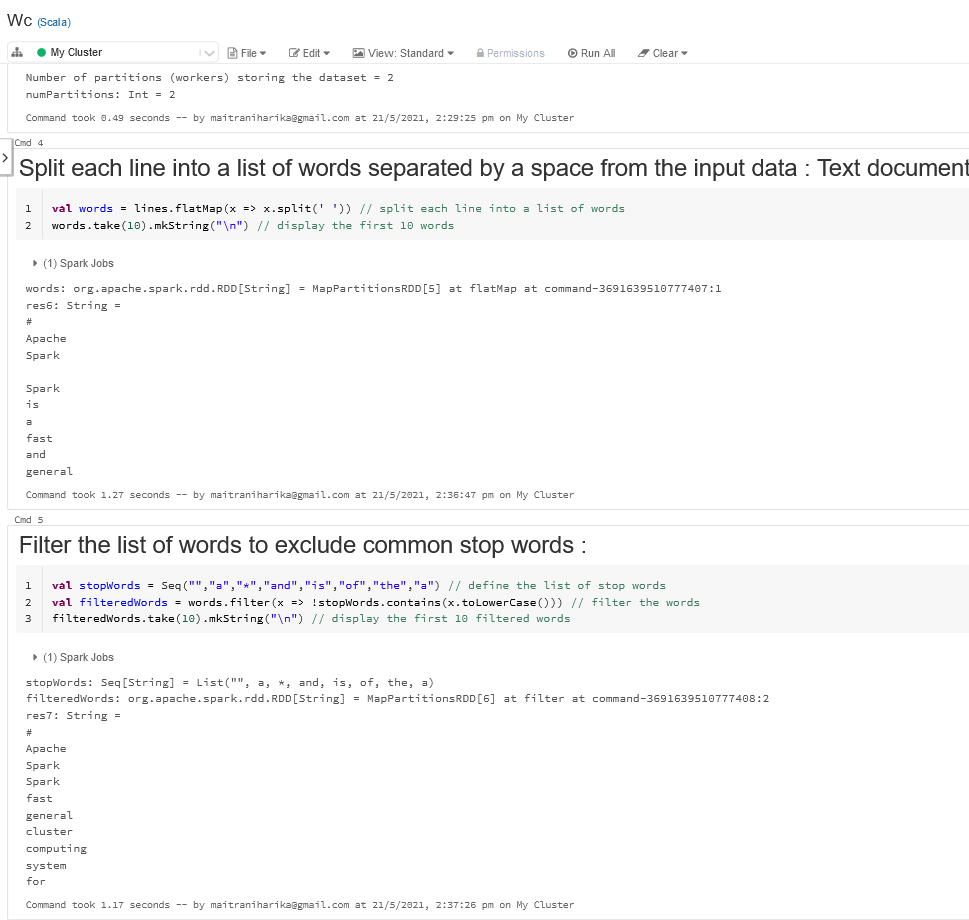
(1) Spark Jobs

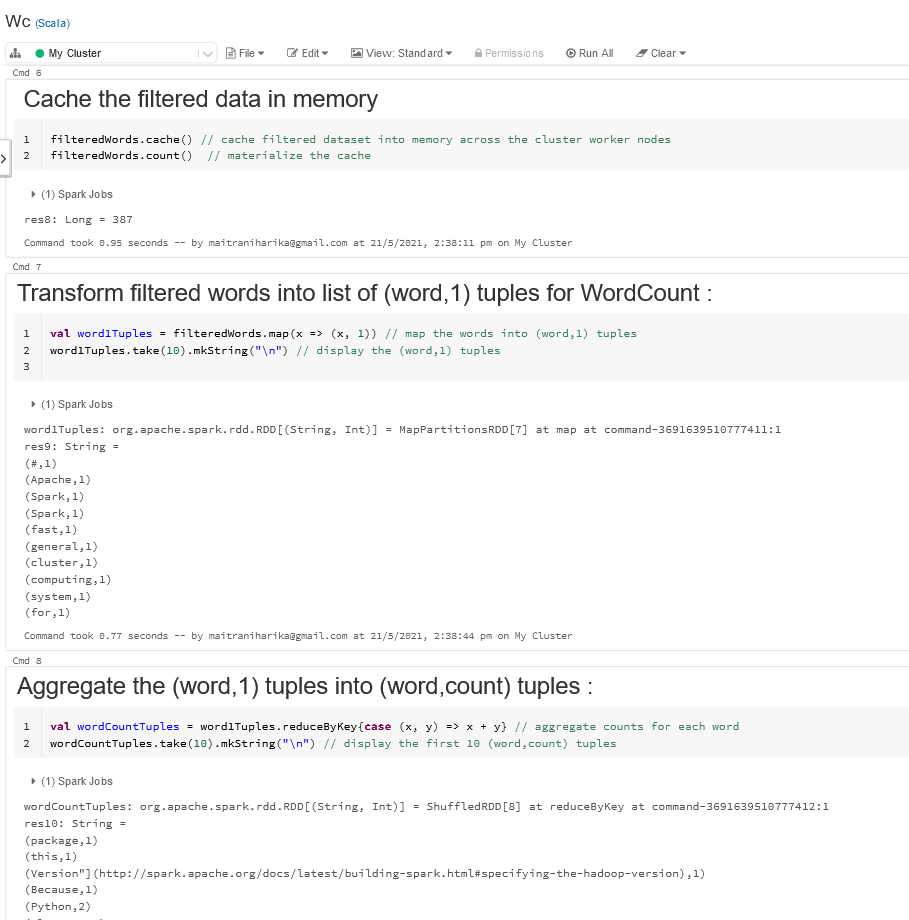
sortedWordCountTuples: String = (to,14) (Spark,13) (for,11) (##,8) (run,7) (can,6) (in,5) (on,5) (also,4) (you,4)

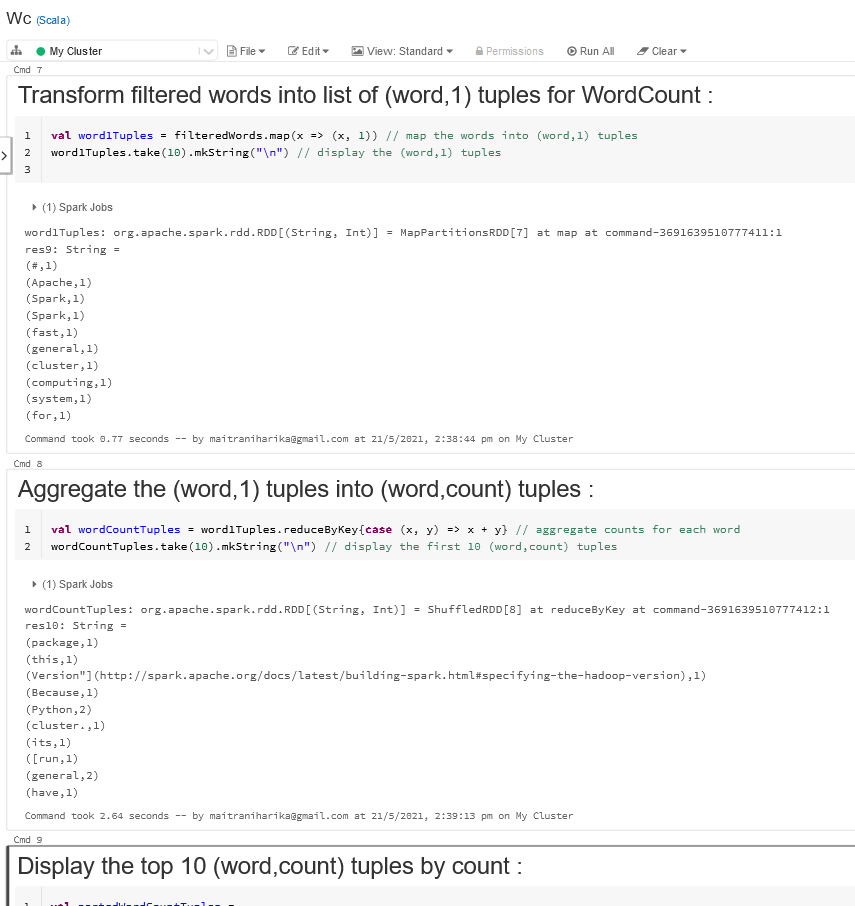
Command took 0.70 seconds -- by maitraniharika@gmail.com at 21/5/2021, 2:39:51 pm on My Cluster

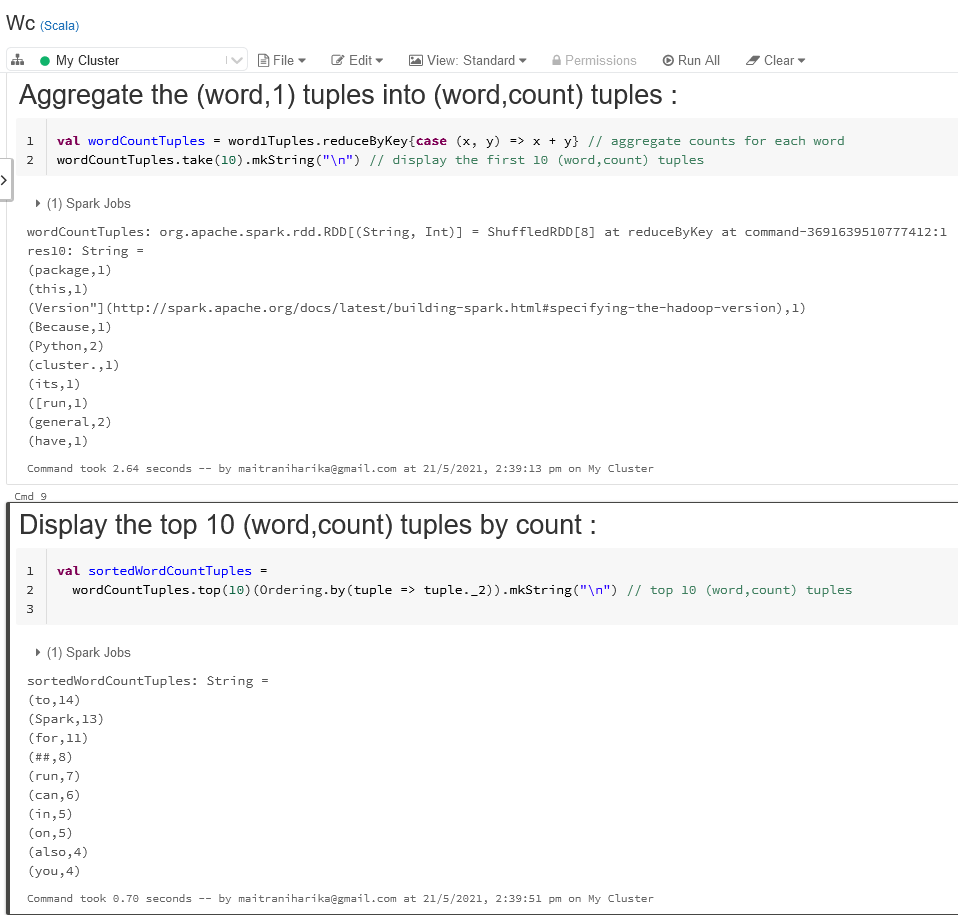
**Screenshot of the Scala Program Code and the corresponding Outputs obtained on executing the Word Count Program:**



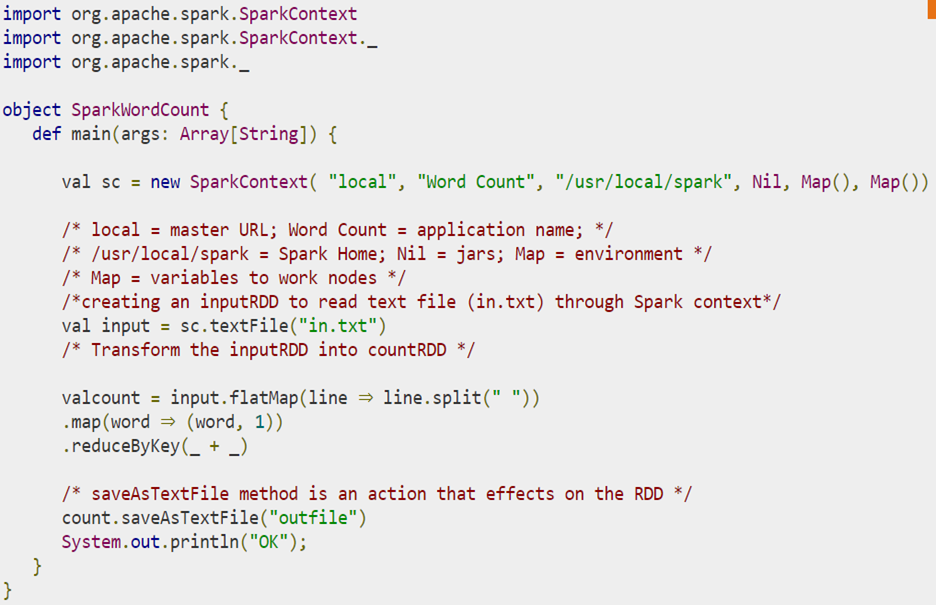








**Java Code Screenshot for the Spark-Scala Word Count Program:**



**Activity - 2 : Implement at least 5 Transformations and 5 Actions on an RDD created by reading datasets from a “csv/json” file**

**Scala Program Code and the corresponding Output obtained on implementing : the 5 transformations and 5 actions on RDD created by reading dataset from a “csv/json” file :**

**val ds = spark.read.json("/databricks-datasets/iot/iot\_devices.json")**

**//Spark does not know the structure of input data. That is, it doesn’t know how it is required to organize inputdata into a**

**//typed-specific JVM object. It attempts to infer the schema from the JSON file and creates a DataFrame = Dataset[Row] of generic Row objects**

(1) Spark Jobs

ds:org.apache.spark.sql.DataFrame = [battery\_level: long, c02\_level: long ... 13 more fields]

ds: org.apache.spark.sql.DataFrame = [battery\_level: bigint, c02\_level: bigint ... 13 more fields]

Command took 4.67 seconds -- by maitraniharika@gmail.com at 21/5/2021, 5:40:16 pm on My Cluster

**// define a case class that represents the device data.**

**case class DeviceIoTData (**

**battery\_level: Long,**

**c02\_level: Long,**

**cca2: String,**

**cca3: String,**

**cn: String,**

**device\_id: Long,**

**device\_name: String,**

**humidity: Long,**

**ip: String,**

**latitude: Double,**

**longitude: Double,**

**scale: String,**

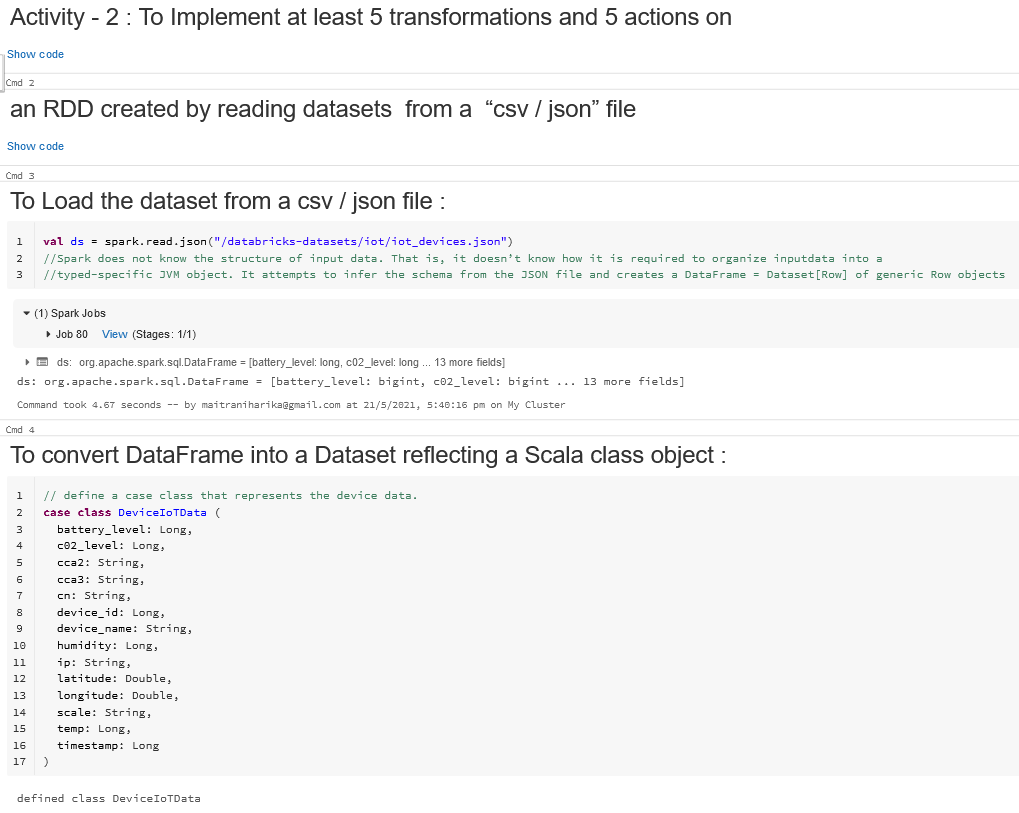
**temp: Long,**

**timestamp: Long**

**)**

defined class DeviceIoTData

Command took 0.38 seconds -- by maitraniharika@gmail.com at 21/5/2021, 5:40:16 pm on My Cluster



**// read the JSON file and create the Dataset from the ``case class`` DeviceIoTData**

**// ds is now a collection of JVM Scala objects DeviceIoTData**

**val ds = spark.read.json("/databricks-datasets/iot/iot\_devices.json").as[DeviceIoTData]**

**(1) Spark Jobs**

**ds:org.apache.spark.sql.Dataset[DeviceIoTData] = [battery\_level: long, c02\_level: long ... 13 more fields]**

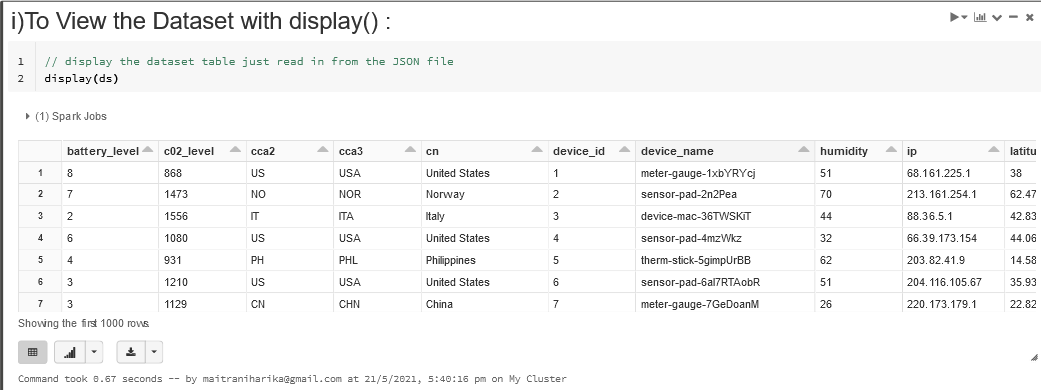
**ds: org.apache.spark.sql.Dataset[DeviceIoTData] = [battery\_level: bigint, c02\_level: bigint ... 13 more fields]**

**Command took 5.14 seconds -- by maitraniharika@gmail.com at 21/5/2021, 5:40:16 pm on My Cluster**

****

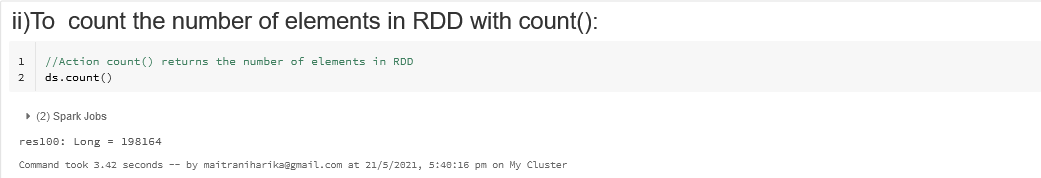
**// display the dataset table just read in from the JSON file**

**display(ds)**

****

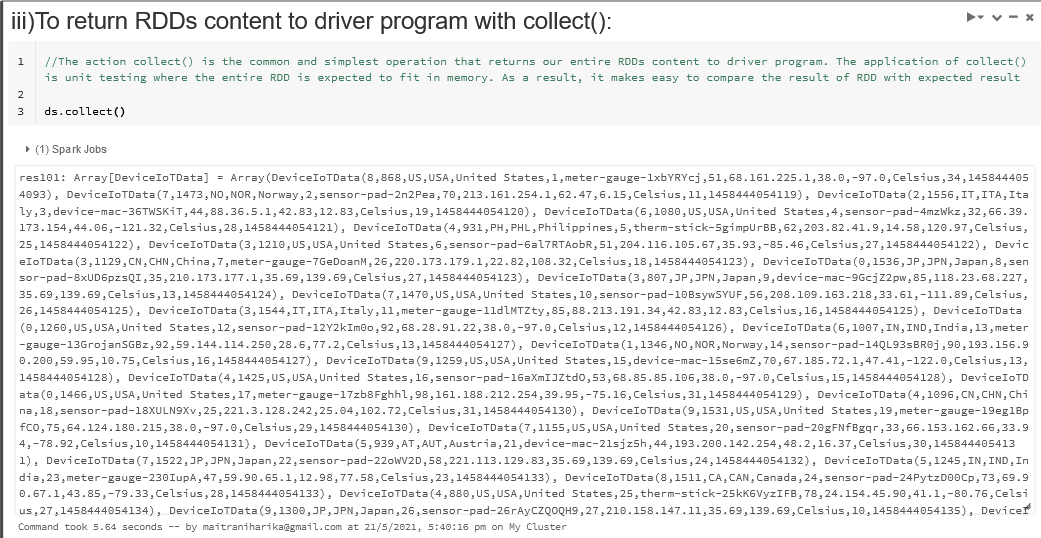
**//Action count() returns the number of elements in RDD**

**ds.count()**

****

**//The action collect() is the common and simplest operation that returns our entire RDDs content to driver program. The application of collect() is unit testing where the entire RDD is expected to fit in memory. As a result, it makes easy to compare the result of RDD with expected result**

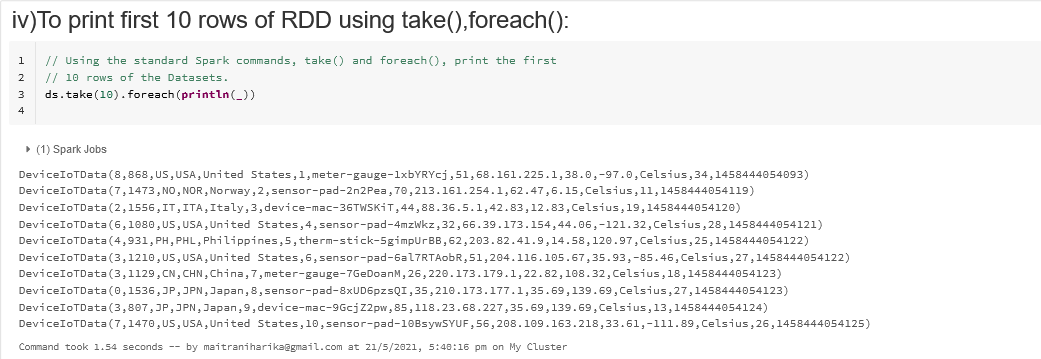
**ds.collect()**

****

**// Using the standard Spark commands, take() and foreach(), print the first**

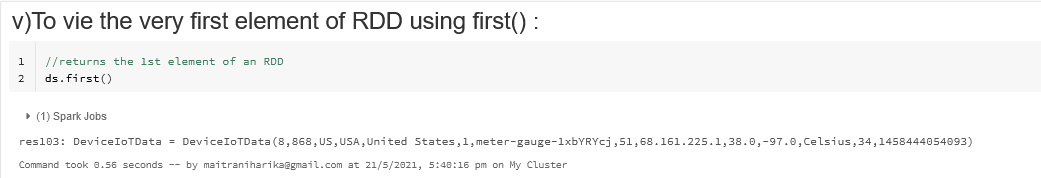
**// 10 rows of the Datasets.**

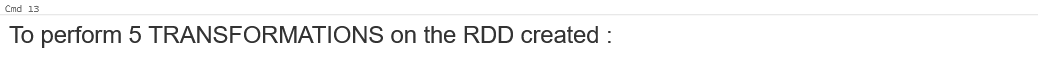
**ds.take(10).foreach(println(\_))**

****

**//returns the 1st element of an RDD**

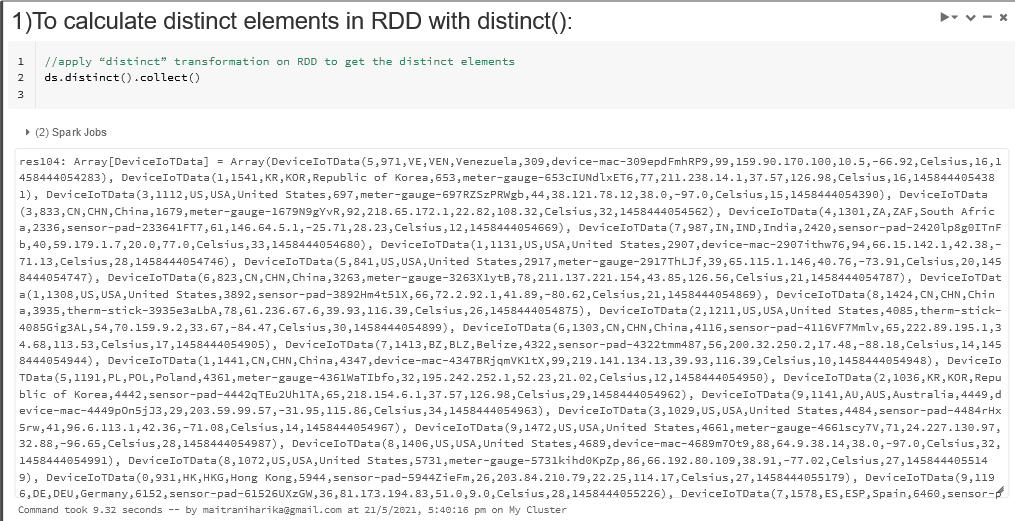
**ds.first()**

****

****

**//apply “distinct” transformation on RDD to get the distinct elements**

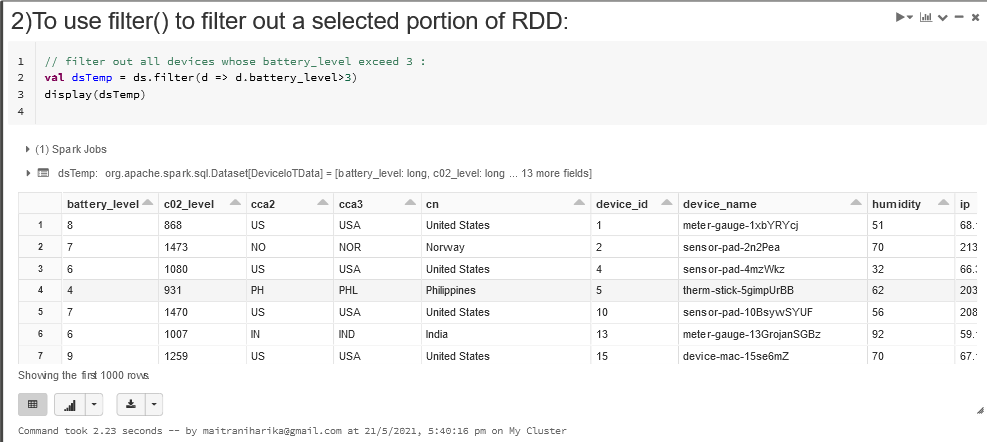
**ds.distinct().collect()**

****

**// filter out all devices whose battery\_level exceed 3 :**

**val dsTemp = ds.filter(d => d.battery\_level>3)**

**display(dsTemp)**

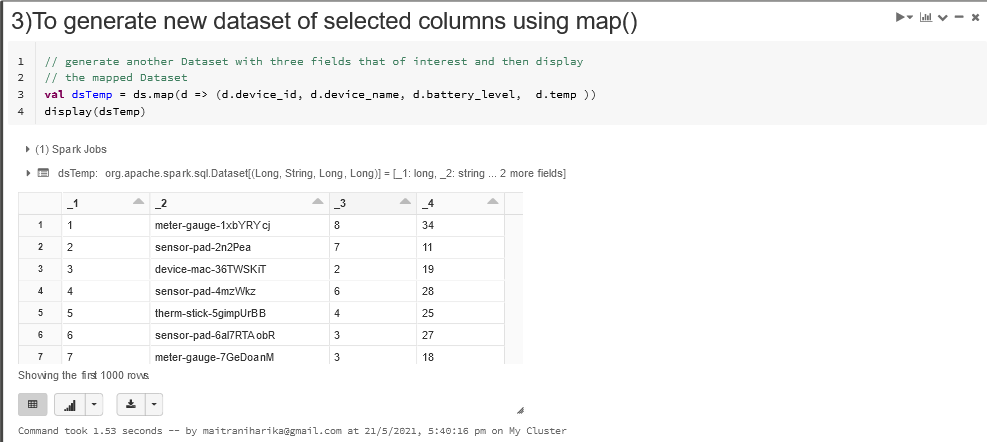
****

**// generate another Dataset with fields that of interest and then display**

**// the mapped Dataset**

**val dsTemp = ds.map(d => (d.device\_id, d.device\_name, d.battery\_level, d.temp ))**

**display(dsTemp)**

****

**val dsTemp = ds.groupBy("battery\_level").count()**

**display(dsTemp)**

****

**// Apply methods such as groupBy() and avg().**

**// Filter temperatures > 25, along with their corresponding**

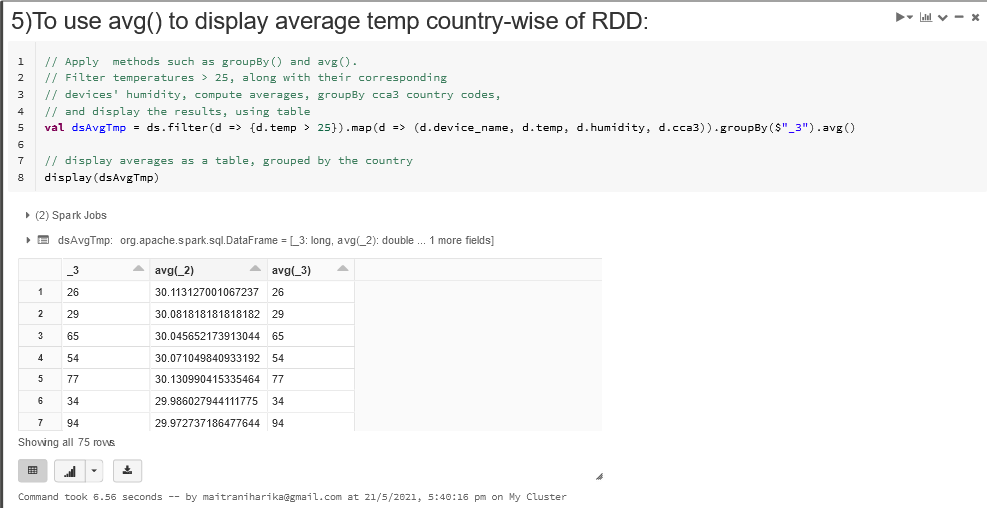
**// devices' humidity, compute averages, groupBy cca3 country codes,**

**// and display the results, using table**

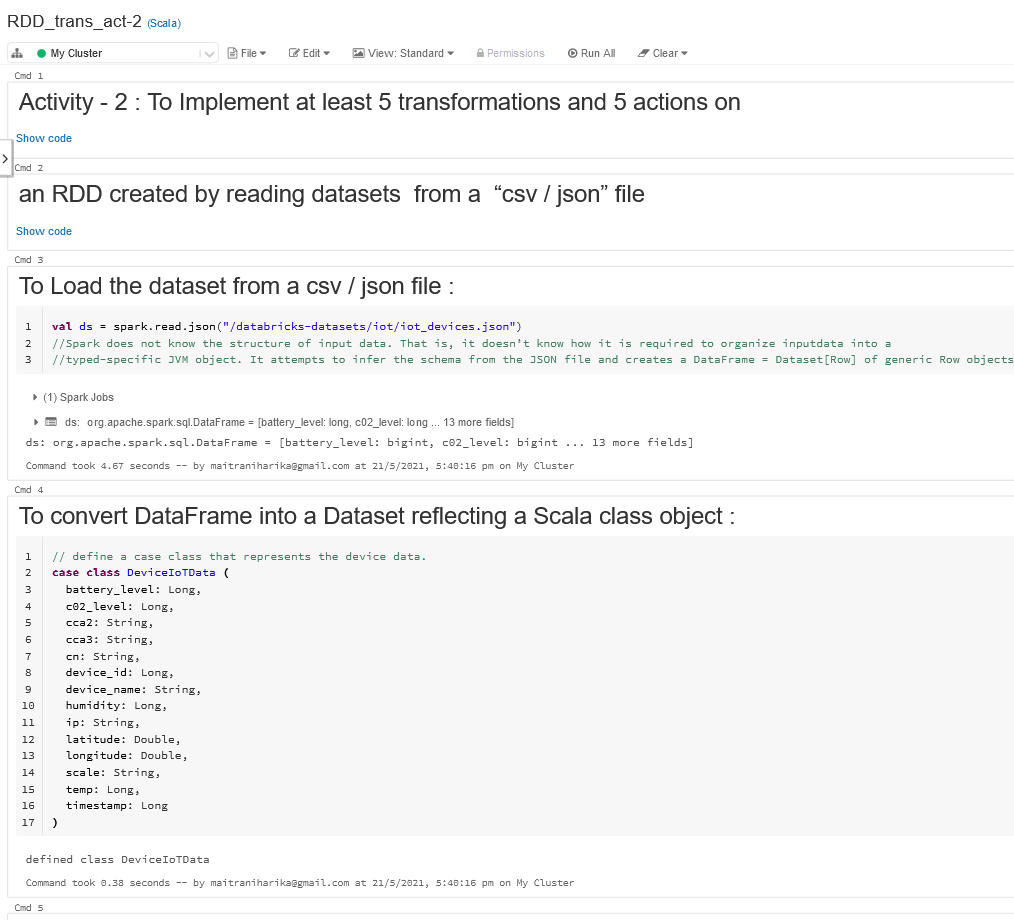
**val dsAvgTmp = ds.filter(d => {d.temp > 25}).map(d => (d.device\_name, d.temp, d.humidity, d.cca3)).groupBy($"\_3").avg()**

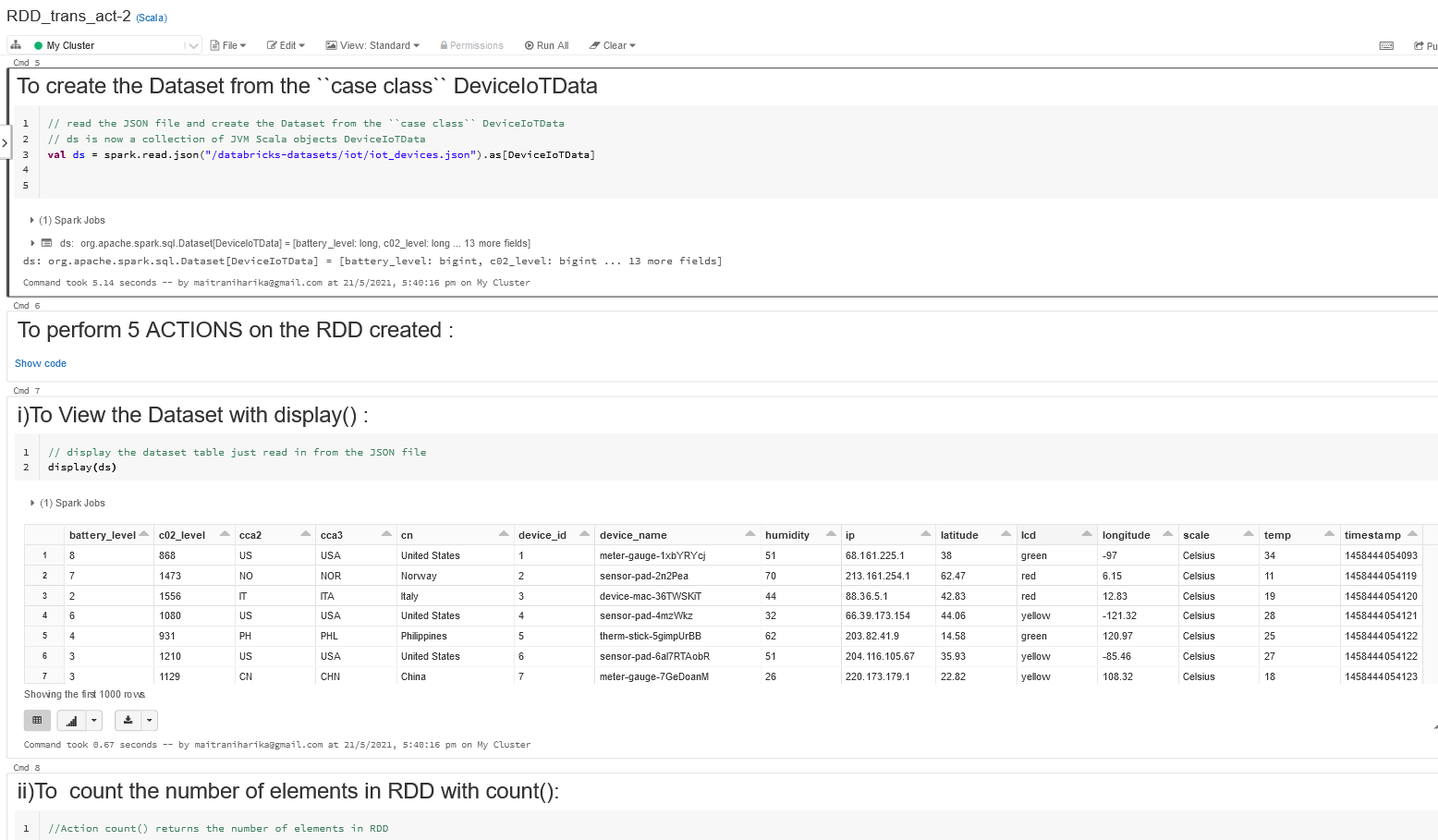
**// display averages as a table, grouped by the country**

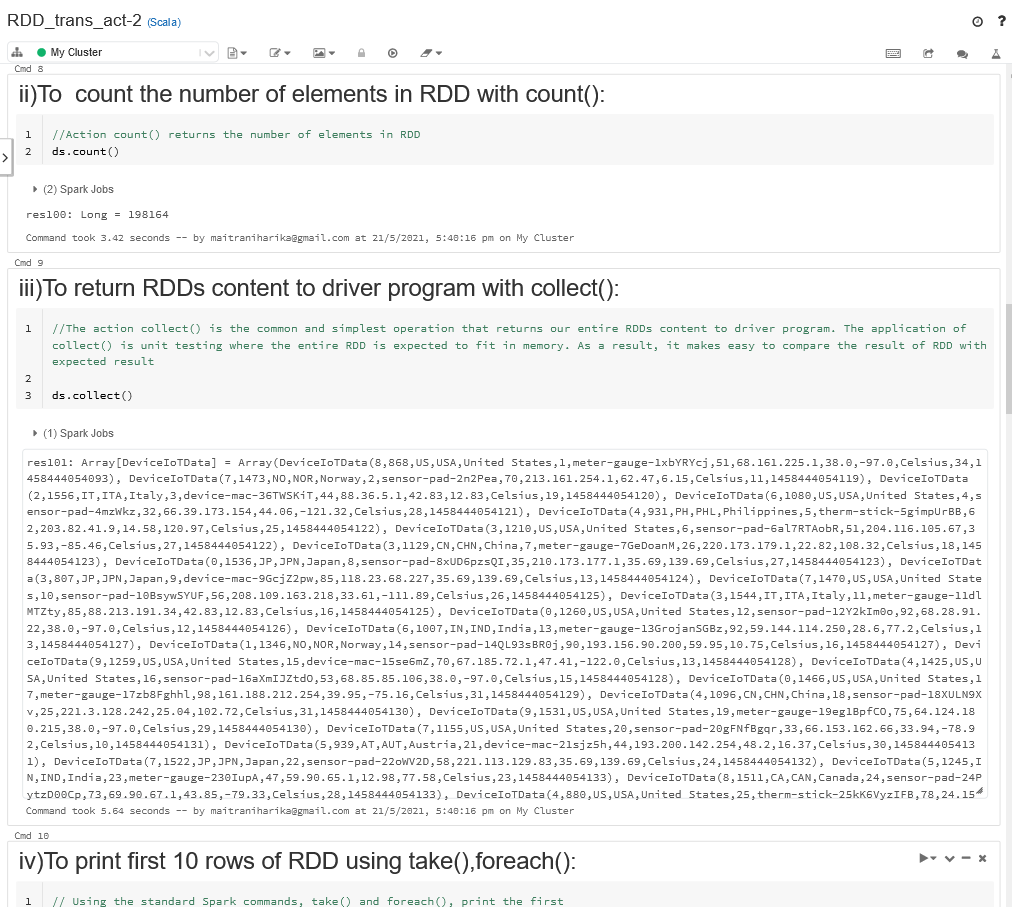
**display(dsAvgTmp)**

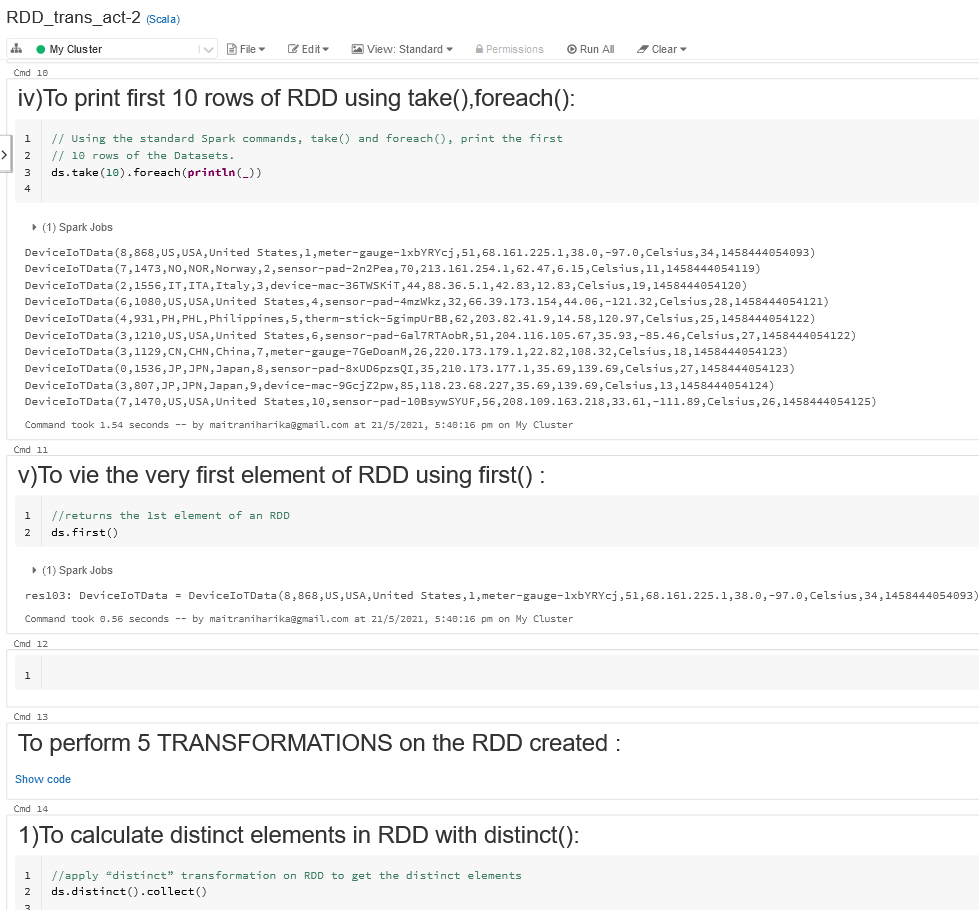
****

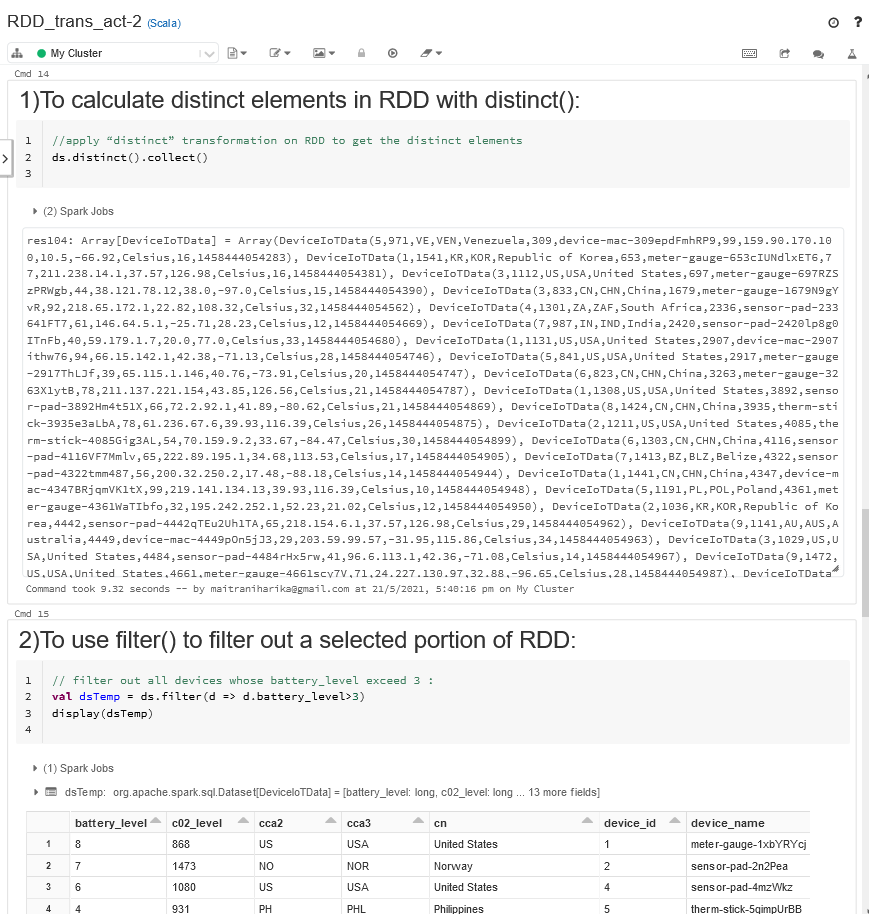
**Screenshot of the Scala Program Codes and the corresponding Outputs obtained on implementing : the 5 transformations and 5 actions on RDD created by reading datasets from a “csv/json” file Program:**

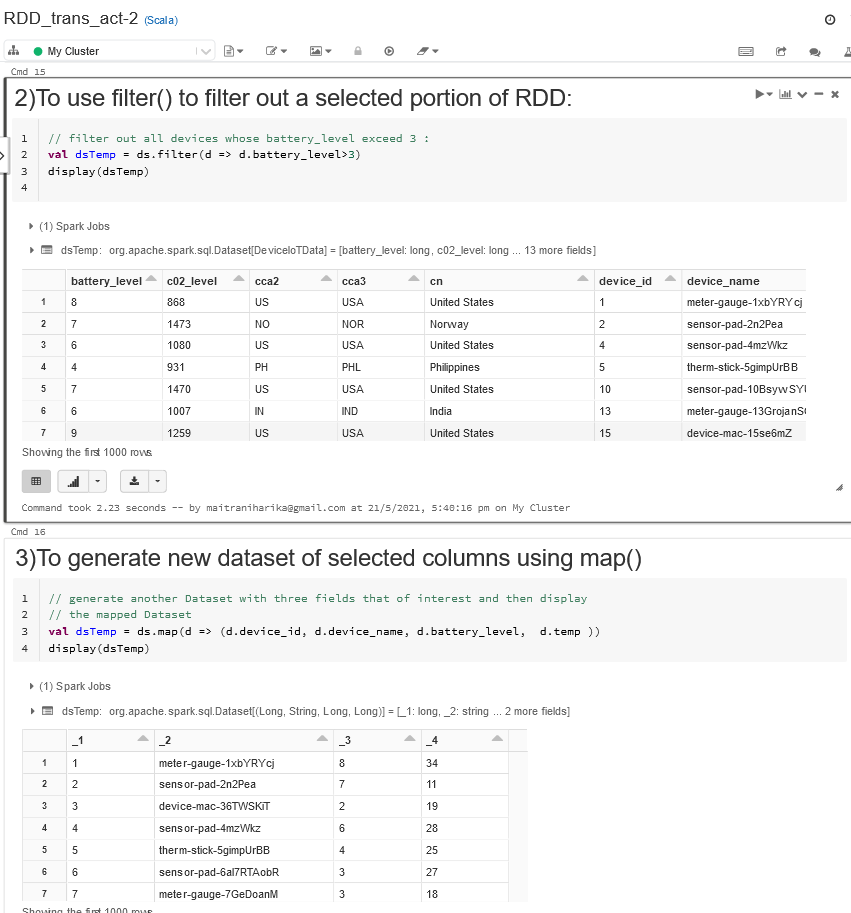
****

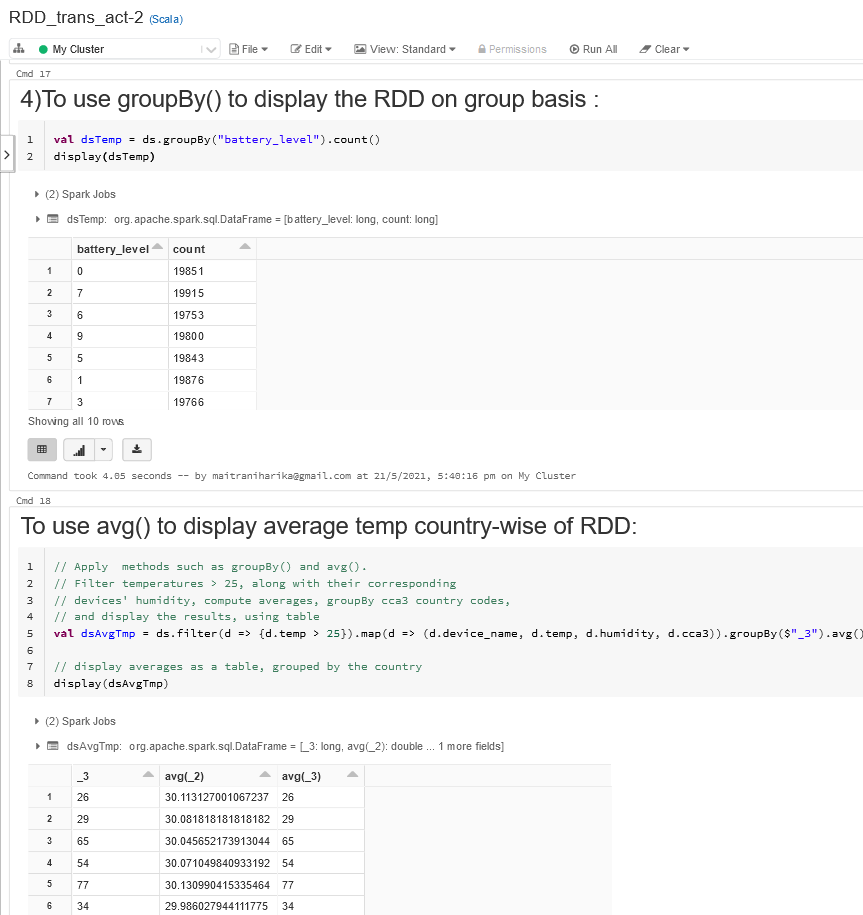
****

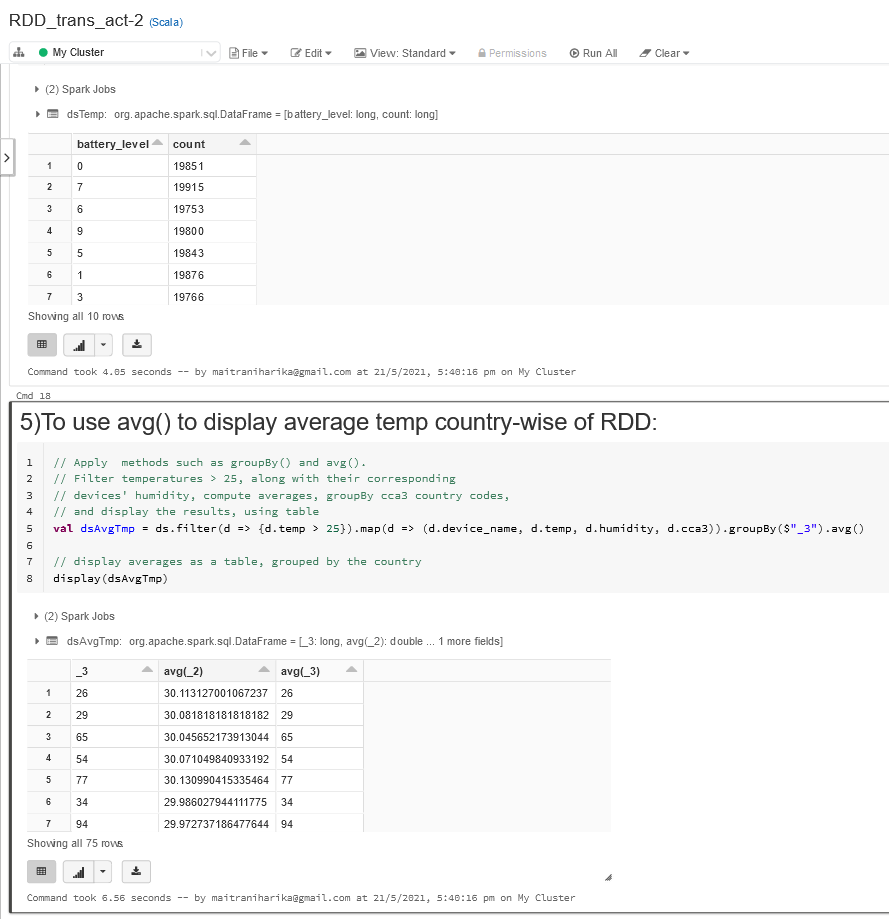
****

****

****

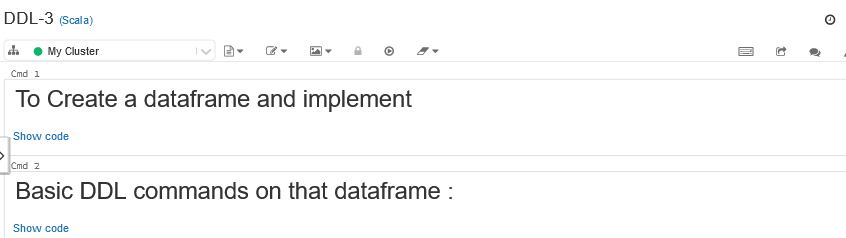
****

****

****

**Activity - 3: Create a dataframe and implement basic SQL(DDL) commands on that dataframe. Write your observation about "Can we do any updation in dataframe table"**

**Scala Program Codes and the corresponding Outputs obtained on implementing : Create a dataframe and implement basic SQL(DDL) commands on that dataframe :**

****

**// Create the case classes for our domain**

**case class Department(id: String, name: String)**

**case class Employee(firstName: String, lastName: String, email: String, salary: Int)**

**case class DepartmentWithEmployees(department: Department, employees: Seq[Employee])**

**// Create the Departments**

**val department1 = new Department("123456", "Computer Science")**

**val department2 = new Department("789012", "Mechanical Engineering")**

**val department3 = new Department("345678", "Theater and Drama")**

**val department4 = new Department("901234", "Indoor Recreation")**

**// Create the Employees**

**val employee1 = new Employee("michael", "armbrust", "no-reply@berkeley.edu", 100000)**

**val employee2 = new Employee("xiangrui", "meng", "no-reply@stanford.edu", 120000)**

**val employee3 = new Employee("matei", null, "no-reply@waterloo.edu", 140000)**

**val employee4 = new Employee(null, "wendell", "no-reply@princeton.edu", 160000)**

**val employee5 = new Employee("michael", "jackson", "no-reply@neverla.nd", 80000)**

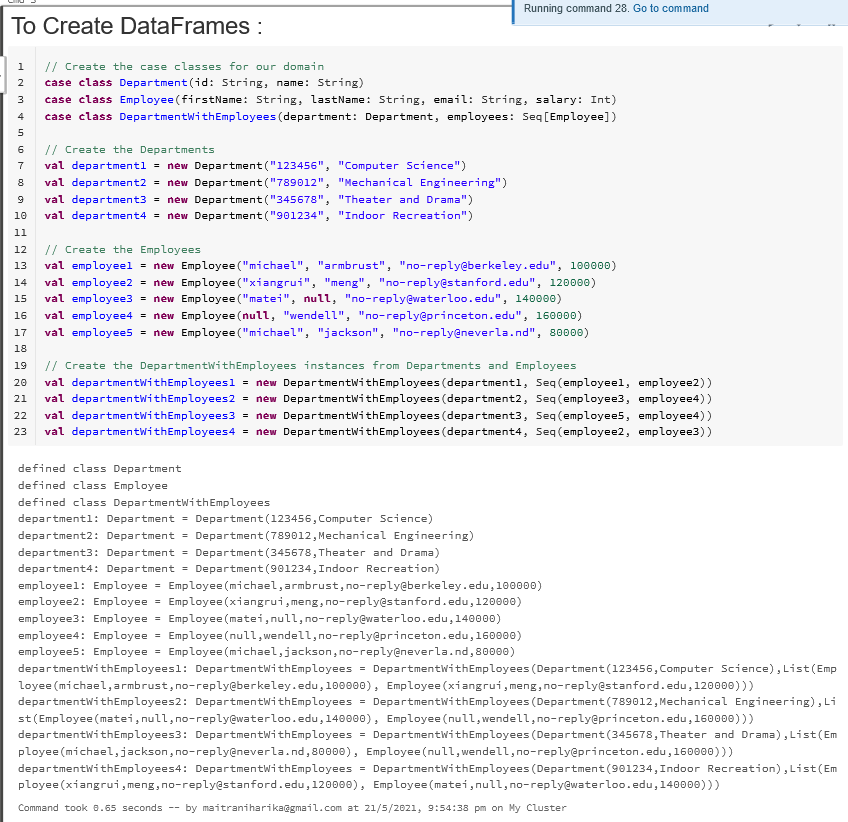
**// Create the DepartmentWithEmployees instances from Departments and Employees**

**val departmentWithEmployees1 = new DepartmentWithEmployees(department1, Seq(employee1, employee2))**

**val departmentWithEmployees2 = new DepartmentWithEmployees(department2, Seq(employee3, employee4))**

**val departmentWithEmployees3 = new DepartmentWithEmployees(department3, Seq(employee5, employee4))**

**val departmentWithEmployees4 = new DepartmentWithEmployees(department4, Seq(employee2, employee3))**

****

**//Create DataFrames from a list of the case classes :**

**val departmentsWithEmployeesSeq1 = Seq(departmentWithEmployees1, departmentWithEmployees2)**

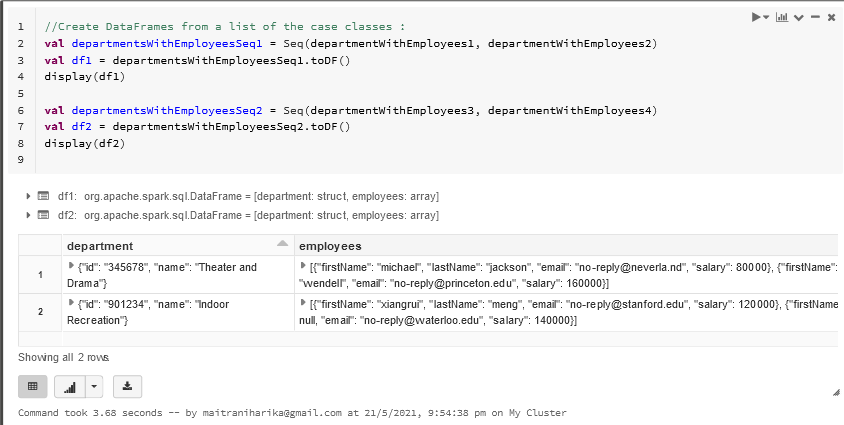
**val df1 = departmentsWithEmployeesSeq1.toDF()**

**display(df1)**

**val departmentsWithEmployeesSeq2 = Seq(departmentWithEmployees3, departmentWithEmployees4)**

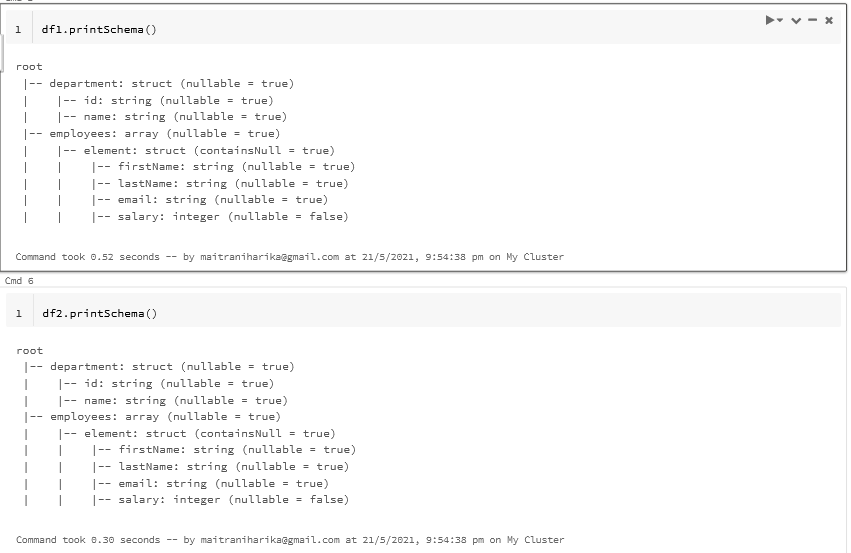
**val df2 = departmentsWithEmployeesSeq2.toDF()**

**display(df2)**

****

**df1.printSchema()**

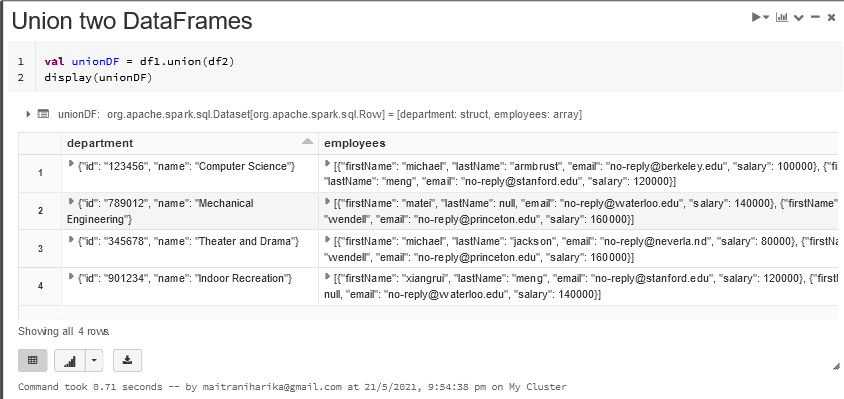
**df2.printSchema()**

****

****

**val unionDF = df1.union(df2)**

**display(unionDF)**

****

**// Remove the file if it exists**

**dbutils.fs.rm("/tmp/databricks-df-example.parquet", true)**

**unionDF.write.parquet("/tmp/databricks-df-example.parquet")**

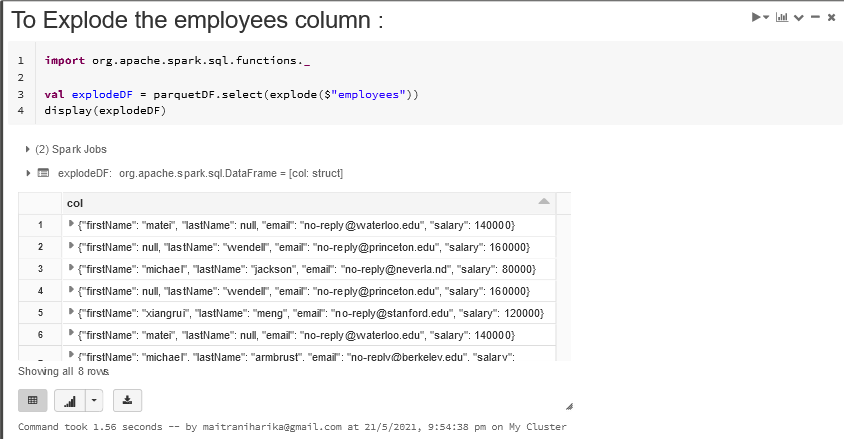
**val parquetDF = spark.read.parquet("/tmp/databricks-df-example.parquet")**

****

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

**val explodeDF = parquetDF.select(explode($"employees"))**

**display(explodeDF)**

****

**val flattenDF = explodeDF.select($"col.\*")**

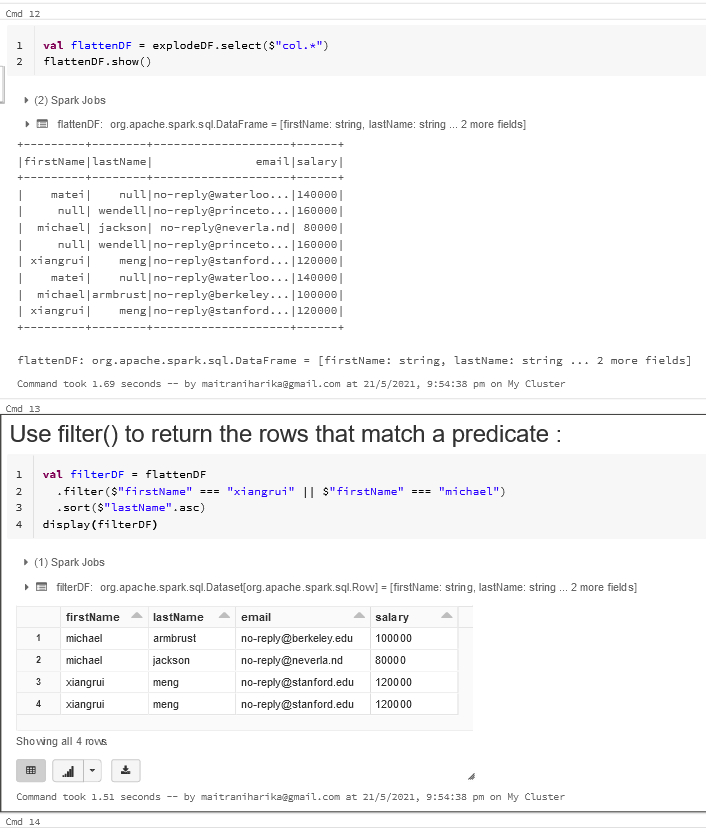
**flattenDF.show()**

**val filterDF = flattenDF**

**.filter($"firstName" === "xiangrui" || $"firstName" === "michael")**

**.sort($"lastName".asc)**

**display(filterDF)**

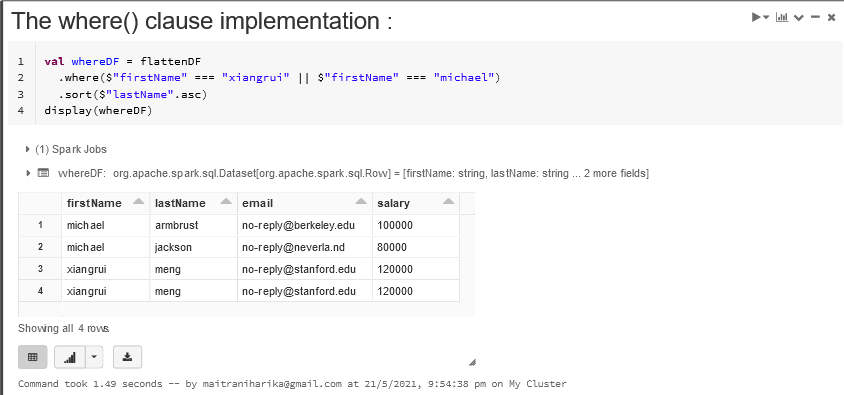
****

**val whereDF = flattenDF**

**.where($"firstName" === "xiangrui" || $"firstName" === "michael")**

**.sort($"lastName".asc)**

**display(whereDF)**

****

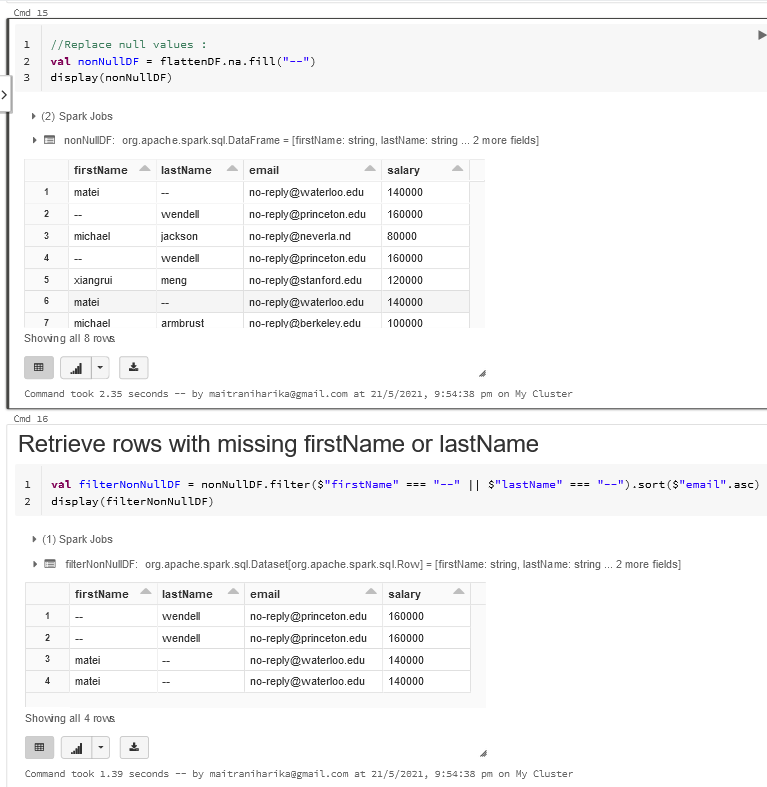
**//Replace null values :**

**val nonNullDF = flattenDF.na.fill("--")**

**display(nonNullDF)**

**val filterNonNullDF = nonNullDF.filter($"firstName" === "--" || $"lastName" === "--").sort($"email".asc)**

**display(filterNonNullDF)**

****

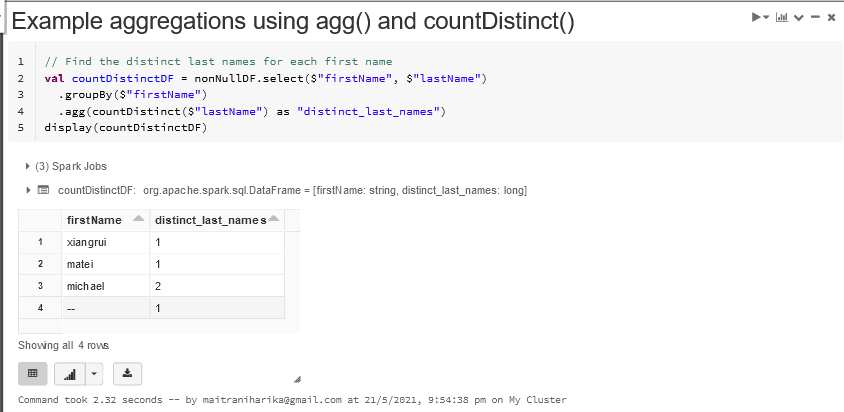
**// Find the distinct last names for each first name**

**val countDistinctDF = nonNullDF.select($"firstName", $"lastName")**

**.groupBy($"firstName")**

**.agg(countDistinct($"lastName") as "distinct\_last\_names")**

**display(countDistinctDF)**

****

**// register the DataFrame as a temp view so that we can query it using SQL**

**nonNullDF.createOrReplaceTempView("databricks\_df\_example")**

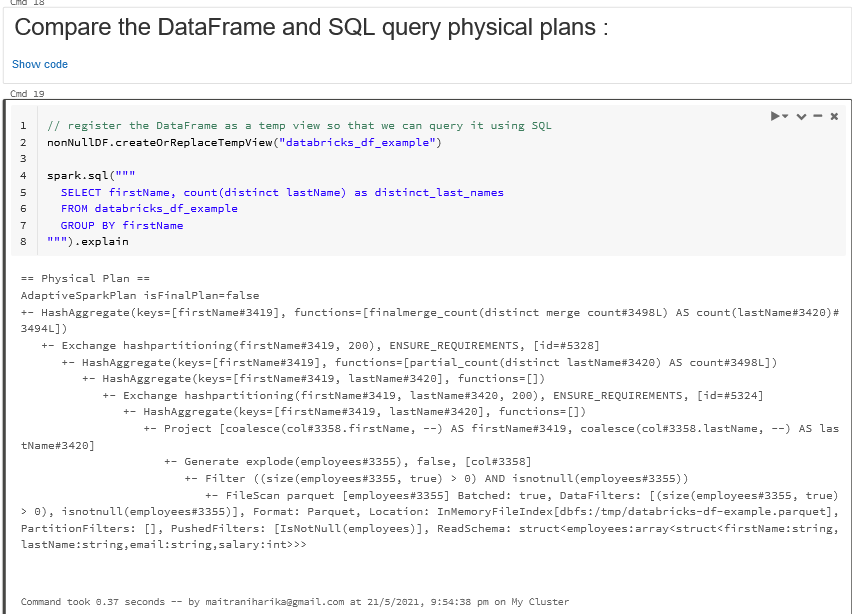
**spark.sql("""**

**SELECT firstName, count(distinct lastName) as distinct\_last\_names**

**FROM databricks\_df\_example**

**GROUP BY firstName**

**""").explain**

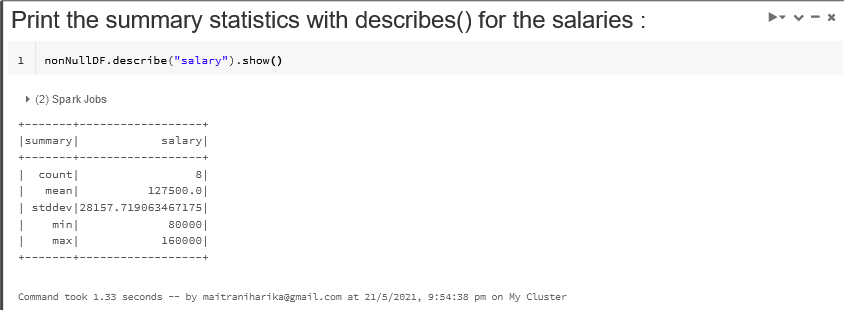
****

**val salarySumDF = nonNullDF.agg("salary" -> "sum")**

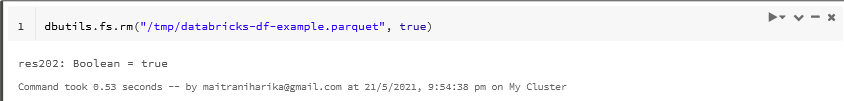
**display(salarySumDF)**

****

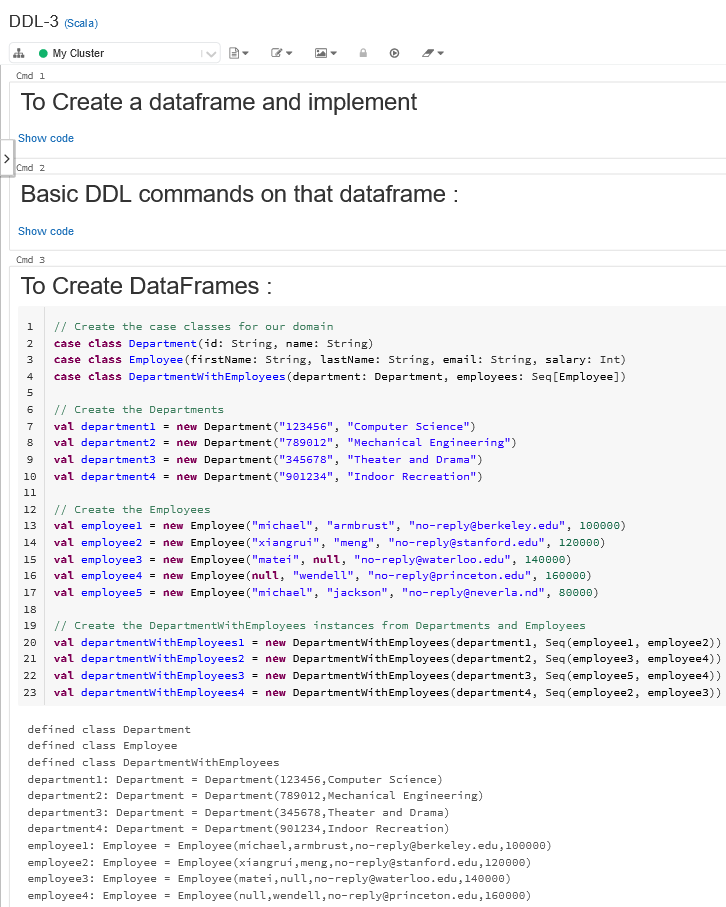
**nonNullDF.describe("salary").show()**

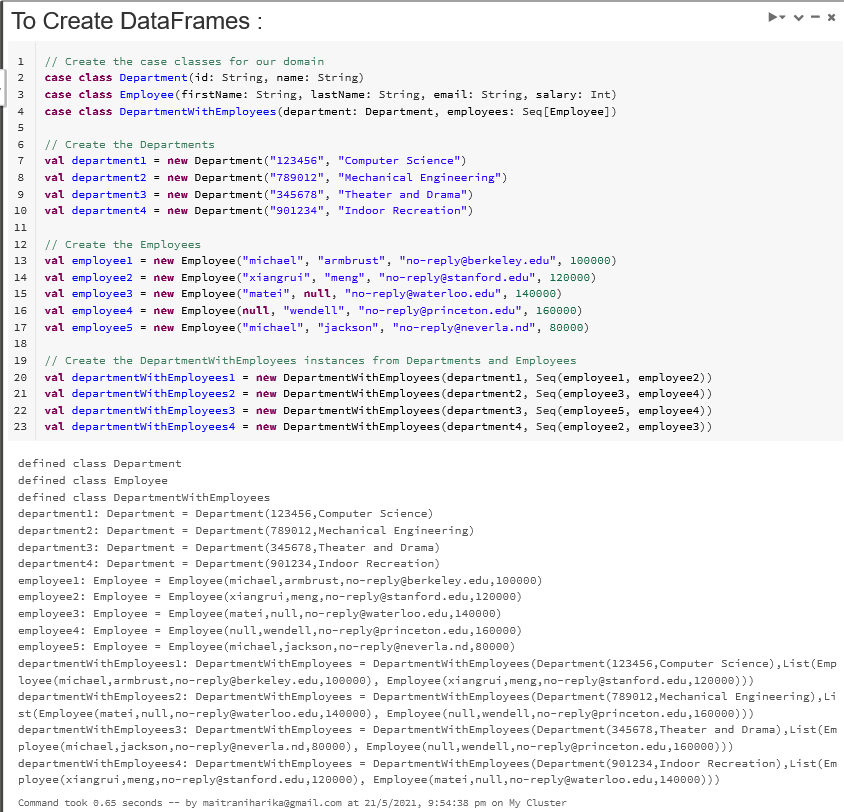
****

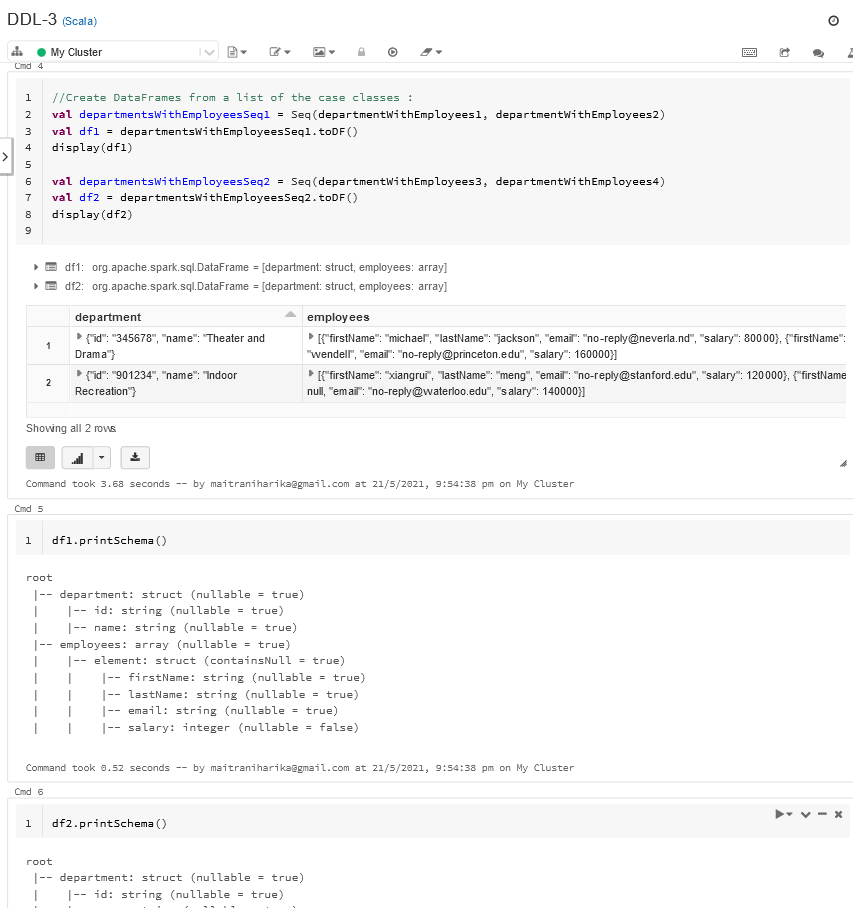
**dbutils.fs.rm("/tmp/databricks-df-example.parquet", true)**

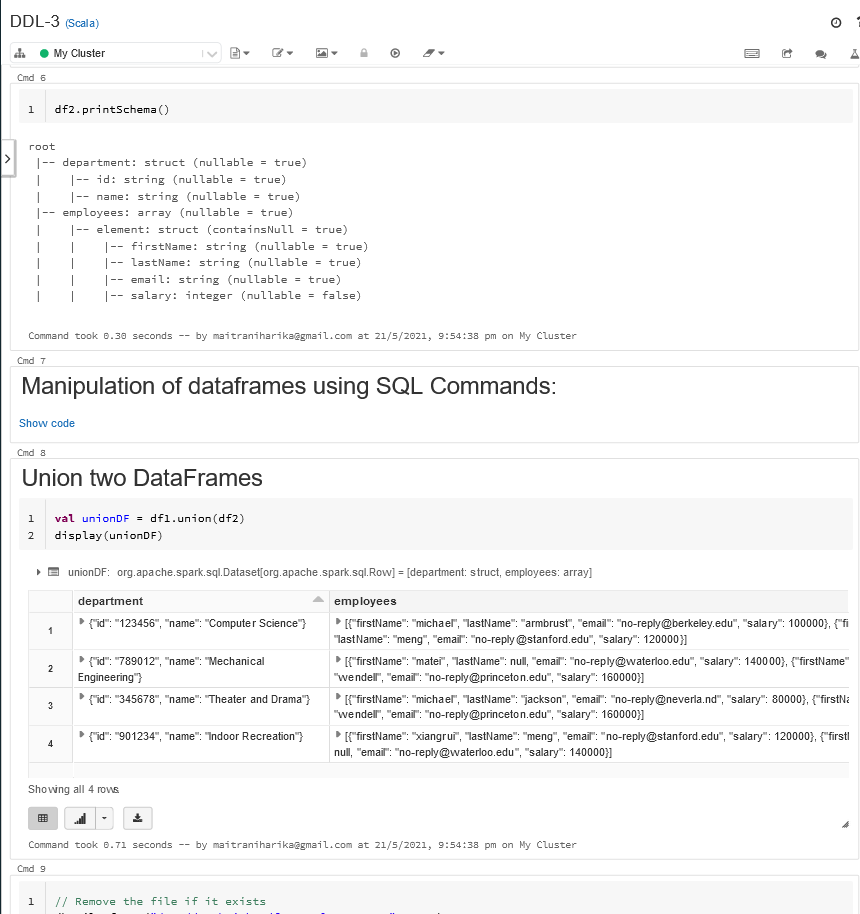
****

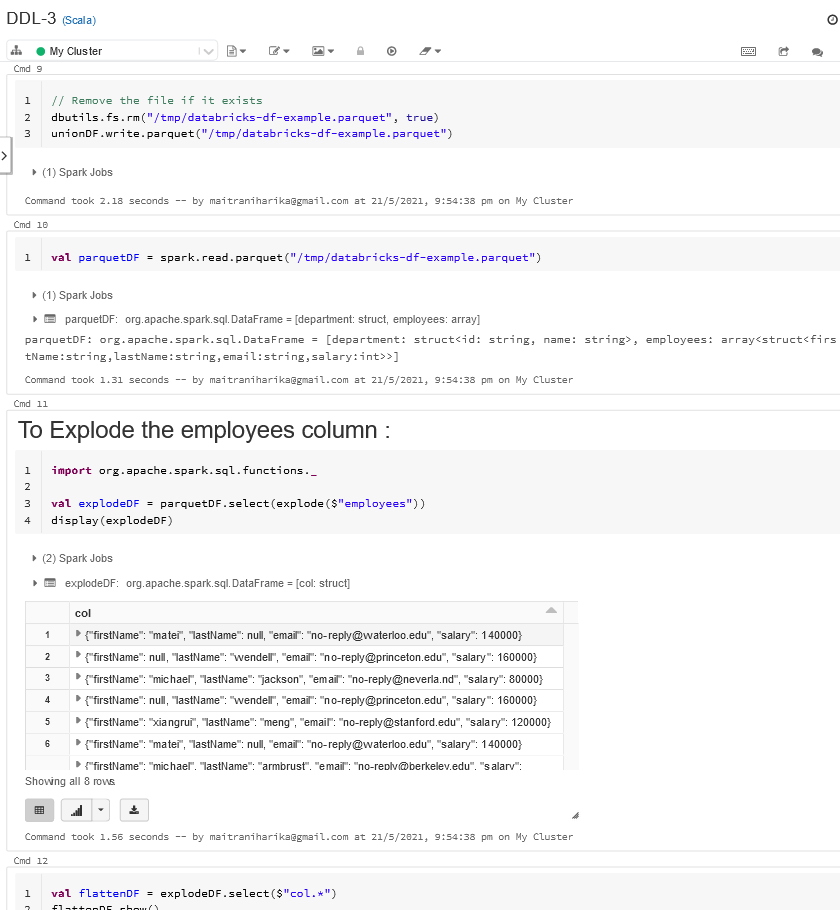
**Screenshots of the Scala Program Codes and the corresponding Outputs obtained on implementing : Create a dataframe and implement basic SQL(DDL) commands on that dataframe :**

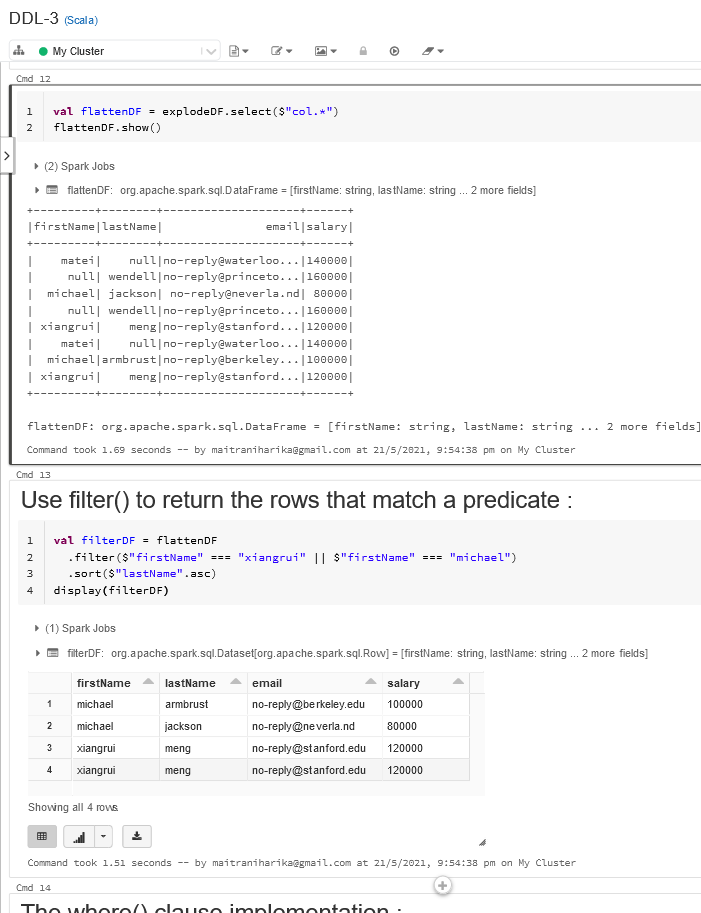
****

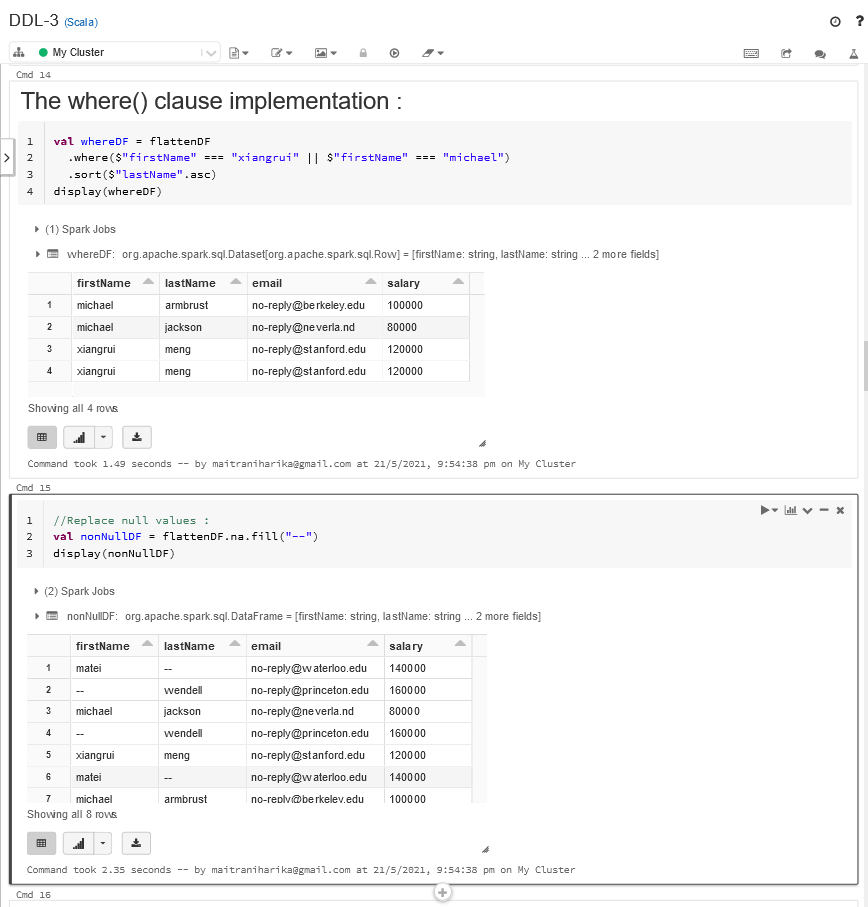
****

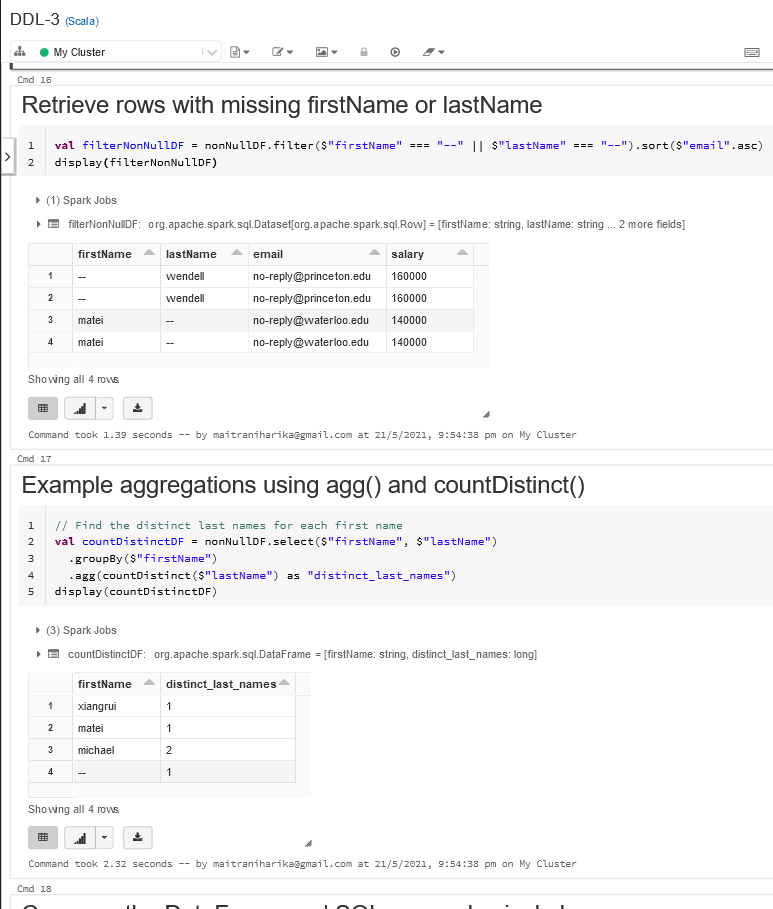
****

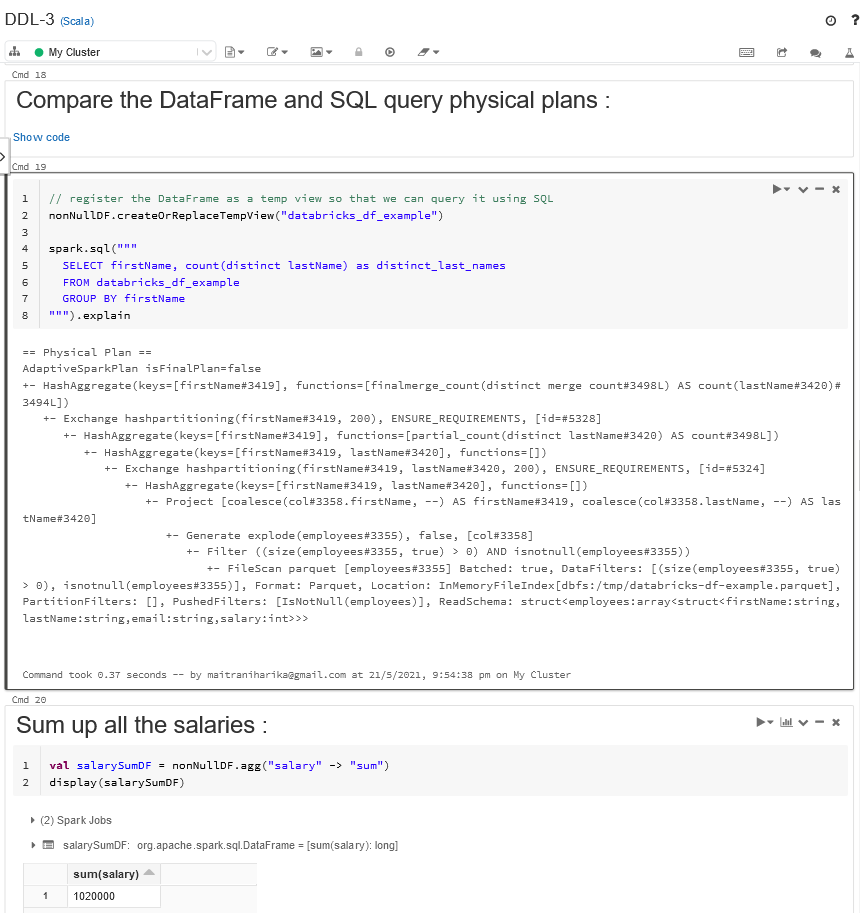
****

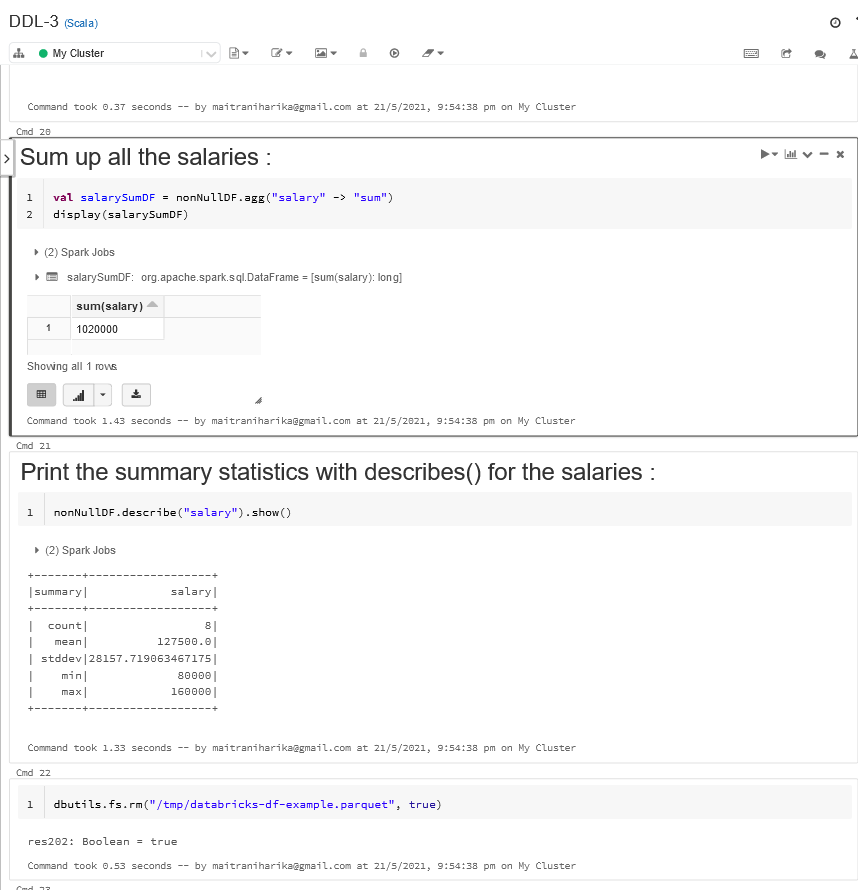
****

****

****

****

****

****

**A brief note on the Observation that "if we can do any updation in dataframe table:**

**Apache Spark has multiple ways to read data from different sources like files, databases etc.**

**But when it comes to loading data into RDBMS(relational database management system),**

**Spark supports only Append and Overlay of the data using dataframes.**

**Spark dataframes does not support Updating of data into a database.**

**DataFrame is based on RDD,it is not possible to update values in it.**

**But the same could be done with Column for updating values by adding new column.**