Spark Assignment 18.2

Given the below 3 dataset

1) S18_Dataset_Holidays.txt as a text file (user_id, src, dest, travel_mode, distance, year_of_travel)

2) S18_Dataset_Transport.txt as a text file (travel_mode, cost_per_unit)

3) S18_Dataset_User_details.txt as a text file (user_id, src, dest, travel_mode, distance, year_of_travel)

Solve the below mentioned problem statement in spark rdd.

This assignment is done in the spark shell within the acadgildVM.

Steps Followed:

To Solve the problems first I've combined the datas from the above 3 files to one, using the common keys. Used travel_mode as a key to combine transport and holidays data. Used user_id to combine user_details and holidays.

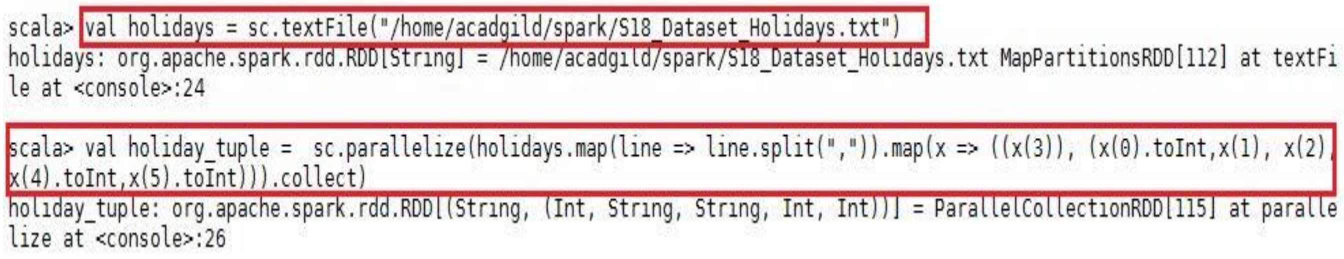
1) Copied the dataset file in the path /home/acadgild/spark/S18_Dataset_Holidays.txt Then read the text file by using sc.textfile as below.

   ```
   val holidays = sc.textFile("/home/acadgild/spark/S18_Dataset_Holidays.txt")
   ```

2) Then created a holiday_tuple rdd by using map function over the holidays rdd with travel_mode as key. Then created a parallelized collection by using parallelize method over the mapped tuple.

   ```
   val holiday_tuple =  sc.parallelize(holidays.map(line => line.split(",")).map(x =>
   ((x(3)), (x(0).toInt,x(1), x(2),x(4).toInt,x(5).toInt))).collect)
   ```

Spark-shell output



```
scala> val holidays = sc.textFile("/home/acadgild/spark/S18_Dataset_Holidays.txt")
holidays: org.apache.spark.rdd.RDD[String] = /home/acadgild/spark/S18_Dataset_Holidays.txt MapPartitionsRDD[112] at textFi
le at <console>:24

scala> val holiday_tuple =  sc.parallelize(holidays.map(line => line.split(",")).map(x => ((x(3)), (x(0).toInt,x(1), x(2),
x(4).toInt,x(5).toInt))).collect)
holiday_tuple: org.apache.spark.rdd.RDD[(String, (Int, String, String, Int, Int))] = ParallelCollectionRDD[115] at paralle
lize at <console>:26
```

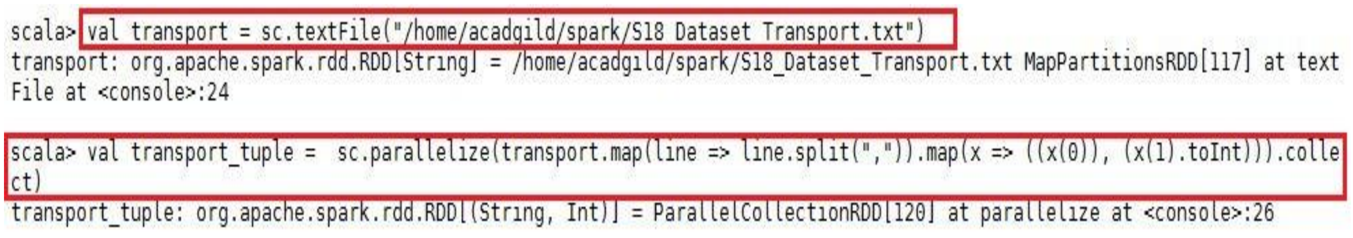
3) Copied the dataset file in the path /home/acadgild/spark/S18_Dataset_ Transport.txt Then read the text file by using sc.textfile as below.

   val transport = sc.textFile("/home/acadgild/spark/S18_Dataset_Transport.txt")

4) Then created a transport_tuple rdd by using map function over the transport rdd with travel_mode as key. Then created a parallelized collection by using parallelize method over the mapped tuple.

   val transport_tuple = sc.parallelize(transport.map(line => line.split(",")).map(x => ((x(0)), (x(1).toInt))).collect)

Spark-shell output



```
scala> val transport = sc.textFile("/home/acadgild/spark/S18 Dataset Transport.txt")
transport: org.apache.spark.rdd.RDD[String] = /home/acadgild/spark/S18_Dataset_Transport.txt MapPartitionsRDD[117] at text
File at <console>:24

scala> val transport_tuple =  sc.parallelize(transport.map(line => line.split(",")).map(x => ((x(0)), (x(1).toInt))).colle
ct)
transport_tuple: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[120] at parallelize at <console>:26
```

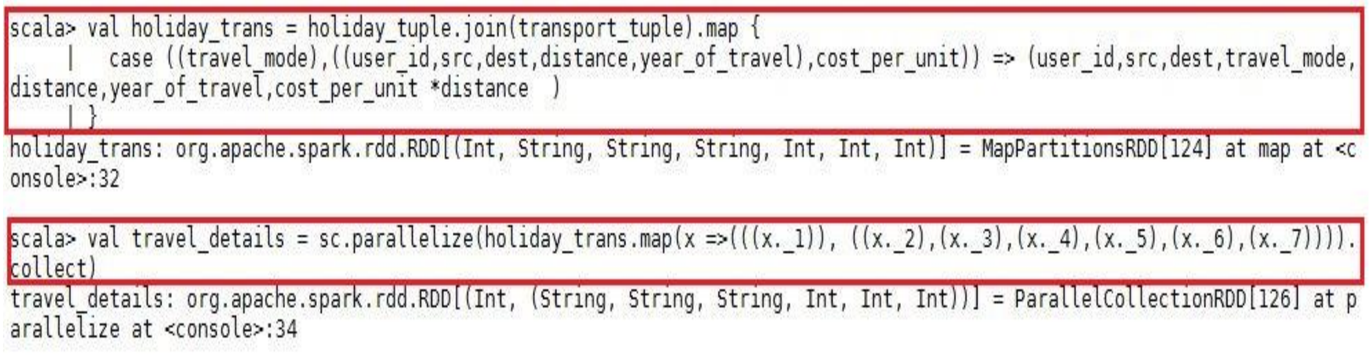
5) Then using join function joined the holiday_tuple and transport_tuple as below as both had travel_mode as key.From the transport_tuple taken the cost_per unit and multiplied with distance, to calculate the expenses for each of the travel.

   val holiday_trans = holiday_tuple.join(transport_tuple).map {

   case ((travel_mode),((user_id,src,dest,distance,year_of_travel),cost_per_unit)) => (user_id,src,dest,travel_mode,distance,year_of_travel,cost_per_unit *distance)

   }

Spark-shell output



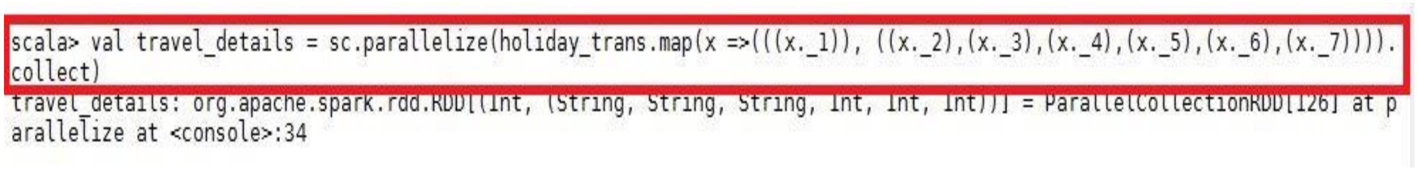
```
scala> val holiday_trans = holiday_tuple.join(transport_tuple).map {
     |   case ((travel_mode),((user_id,src,dest,distance,year_of_travel),cost_per_unit)) => (user_id,src,dest,travel_mode,
distance,year_of_travel,cost_per_unit *distance  )
     | }
holiday_trans: org.apache.spark.rdd.RDD[(Int, String, String, String, Int, Int, Int)] = MapPartitionsRDD[124] at map at <c
onsole>:32

scala> val travel_details = sc.parallelize(holiday_trans.map(x =>(((x._1)), ((x._2),(x._3),(x._4),(x._5),(x._6),(x._7)))).
collect)
travel_details: org.apache.spark.rdd.RDD[(Int, (String, String, String, Int, Int, Int))] = ParallelCollectionRDD[126] at p
arallelize at <console>:34
```

6) Next,To combine the holiday_trans with user data,I've modified the holiday_trans and kept the key as user_id

```
val travel_details = sc.parallelize(holiday_trans.map(x =>(((x._1)),
((x._2),(x._3),(x._4),(x._5),(x._6),(x._7)))).collect)
```

Spark-shell output



```
scala> val travel_details = sc.parallelize(holiday_trans.map(x =>(((x._1)), ((x._2),(x._3),(x._4),(x._5),(x._6),(x._7)))).
collect)
travel_details: org.apache.spark.rdd.RDD[(Int, (String, String, String, Int, Int, Int))] = ParallelCollectionRDD[126] at p
arallelize at <console>:34
```

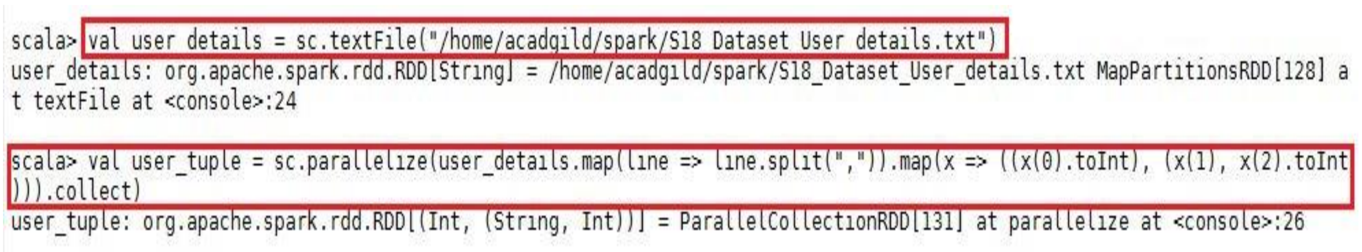
7) Copied the dataset file in the path /home/acadgild/spark/ S18_Dataset_User_details .Then read the text file by using sc.textfile as below.

```
val user_details = sc.textFile("/home/acadgild/spark/S18_Dataset_User_details.txt")
```

8) Then created a user_tuple rdd by using map function over the user_details rdd with user_id as key. Then created a parallelized collection by using parallelize method over the mapped tuple.

```
val user_tuple = sc.parallelize(user_details.map(line => line.split(",")).map(x =>
((x(0).toInt), (x(1), x(2).toInt))).collect)
```

Spark-shell output



```
scala> val user_details = sc.textFile("/home/acadgild/spark/S18_Dataset_User_details.txt")
user_details: org.apache.spark.rdd.RDD[String] = /home/acadgild/spark/S18_Dataset_User_details.txt MapPartitionsRDD[128] a
t textFile at <console>:24

scala> val user_tuple = sc.parallelize(user_details.map(line => line.split(",")).map(x => ((x(0).toInt), (x(1), x(2).toInt
))).collect)
user_tuple: org.apache.spark.rdd.RDD[(Int, (String, Int))] = ParallelCollectionRDD[131] at parallelize at <console>:26
```

9) Then using join function joined the user_tuple and travel_details as below as both had user_id as key.

```
val user_travel = user_tuple.join(travel_details).map {

  case (user_id,((name,age),(src,dest,travel_mode,distance,year_of_travel,charges)))
=> (user_id,name,age,src,dest,travel_mode,distance,year_of_travel,charges )

}
```

The user_travel tuple, will have items in the following order user_id,name,age,src,dest,travel_mode,distance,year_of_travel,charges

user_id ->1,

name -> 2,

age ->3,

src ->4,

dest -> 5,

travel_mode -> 6,

distance -> 7,

year_of_travel -> 8,

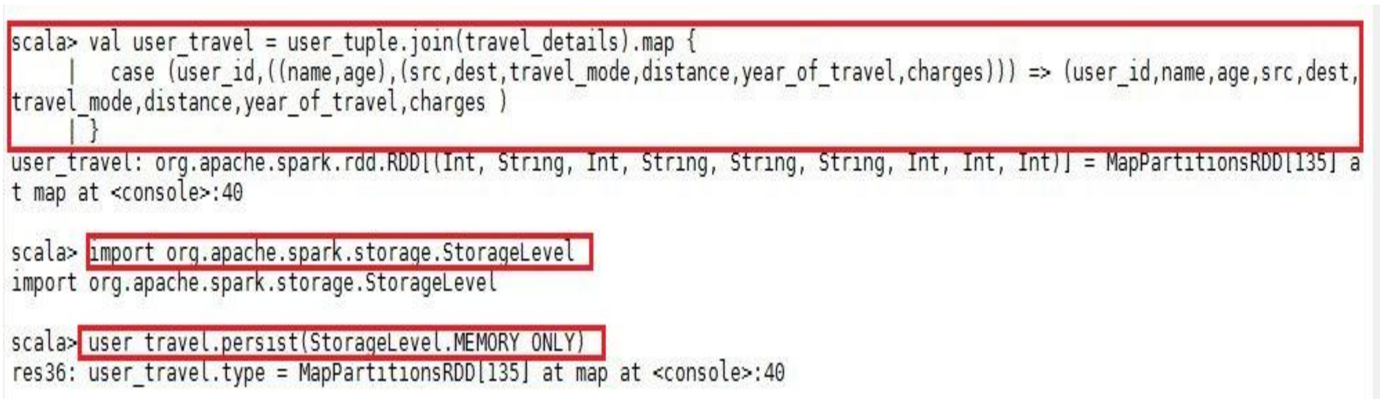
charges -> 9

10) The above Rdd, user_travel is the final tuple, which is a combination of user_details, transport and holidays. It will be used to solve multiple problems, so we are storing it in the memory.

import org.apache.spark.storage.StorageLevel

user_travel.persist(StorageLevel.MEMORY_ONLY)

Spark-shell output



```
scala> val user_travel = user_tuple.join(travel_details).map {
     |   case (user_id,((name,age),(src,dest,travel_mode,distance,year_of_travel,charges))) => (user_id,name,age,src,dest,
travel_mode,distance,year_of_travel,charges )
     | }
user_travel: org.apache.spark.rdd.RDD[(Int, String, Int, String, String, String, Int, Int, Int)] = MapPartitionsRDD[135] a
t map at <console>:40

scala> import org.apache.spark.storage.StorageLevel
import org.apache.spark.storage.StorageLevel

scala> user_travel.persist(StorageLevel.MEMORY_ONLY)
res36: user_travel.type = MapPartitionsRDD[135] at map at <console>:40
```

Problem Statement:

1) ***Which route is generating the most revenue per year.***

To find which route has generated most revenue., we are calling the map function over the user_travel, creating a key with user_travel rdds 8th element which refers to the travel year and setting user_travel rdds 4th ,5th and 9th element, src, dest, charges as value for each item.

Then calling the reduceBykey, where the values for each key are aggregated using the given reduce function revenueCalculate. Here we are calculating the maximum of the 3rd item of the values for each key and finally calling the collect action.

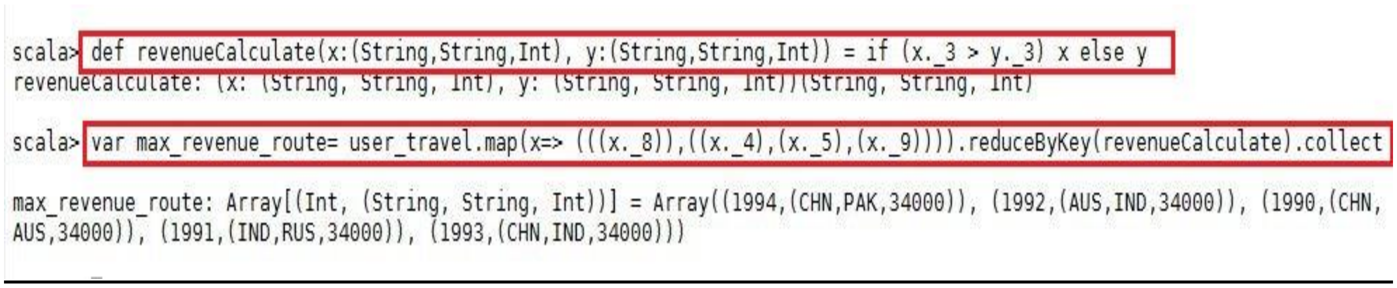
```
def revenueCalculate(x:(String,String,Int), y:(String,String,Int)) = if (x._3 > y._3) x else y
```

```
var max_revenue_route= user_travel.map(x=>
(((x._8)),((x._4),(x._5),(x._9)))).reduceByKey(revenueCalculate).collect
```

Output

max_revenue_route: Array[(Int, (String, String, Int))] = Array((1994,(CHN,PAK,34000)), (1992,(AUS,IND,34000)), (1990,(CHN,AUS,34000)), (1991,(IND,RUS,34000)), (1993,(CHN,IND,34000)))

Spark-shell ouput



```
scala> def revenueCalculate(x:(String,String,Int), y:(String,String,Int)) = if (x._3 > y._3) x else y
revenueCalculate: (x: (String, String, Int), y: (String, String, Int))(String, String, Int)

scala> var max_revenue_route= user_travel.map(x=> (((x._8)),((x._4),(x._5),(x._9)))).reduceByKey(revenueCalculate).collect

max_revenue_route: Array[(Int, (String, String, Int))] = Array((1994,(CHN,PAK,34000)), (1992,(AUS,IND,34000)), (1990,(CHN,
AUS,34000)), (1991,(IND,RUS,34000)), (1993,(CHN,IND,34000)))
```

2) ***What is the total amount spent by every user on air-travel per year.***

To find the the total number of air-travellers per year, we are calling the map function over the user_travel, creating a key with user_travel rdds 1st, 2nd and 8th element which refers to the, id, name, year_of travel, and setting user_travel rdds 9th element, charges as value for each item.

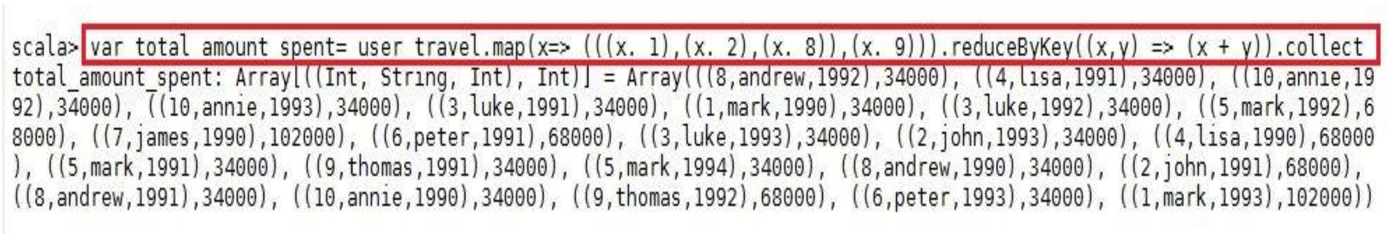
Then calling the reduceBykey, where the values for each key are aggregated using the given reduce function. Here we are adding the values for each key and finally calling the collect action.

```
var total_amount_spent= user_travel.map(x=>
(((x._1),(x._2),(x._8)),(x._9))).reduceByKey((x,y) => (x + y)).collect
```

<u>Output</u>

```
total_amount_spent: Array[((Int, String, Int), Int)] = Array(((8,andrew,1992),34000),
((4,lisa,1991),34000), ((10,annie,1992),34000), ((10,annie,1993),34000),
((3,luke,1991),34000), ((1,mark,1990),34000), ((3,luke,1992),34000),
((5,mark,1992),68000), ((7,james,1990),102000), ((6,peter,1991),68000),
((3,luke,1993),34000), ((2,john,1993),34000), ((4,lisa,1990),68000),
((5,mark,1991),34000), ((9,thomas,1991),34000), ((5,mark,1994),34000),
((8,andrew,1990),34000), ((2,john,1991),68000), ((8,andrew,1991),34000),
((10,annie,1990),34000), ((9,thomas,1992),68000), ((6,peter,1993),34000),
((1,mark,1993),102000))
```

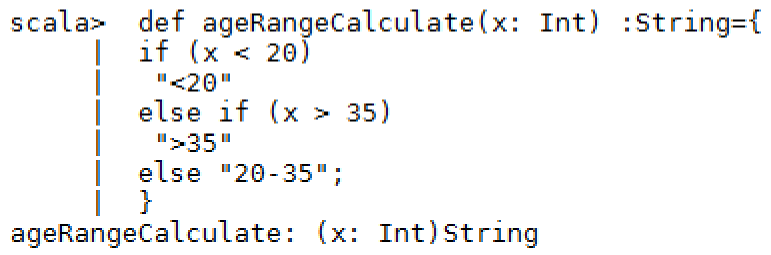
<u>Spark-shell ouput</u>



```
scala> var total_amount_spent= user_travel.map(x=> (((x._1),(x._2),(x._8)),(x._9))).reduceByKey((x,y) => (x + y)).collect
total_amount_spent: Array[((Int, String, Int), Int)] = Array(((8,andrew,1992),34000), ((4,lisa,1991),34000), ((10,annie,19
92),34000), ((10,annie,1993),34000), ((3,luke,1991),34000), ((1,mark,1990),34000), ((3,luke,1992),34000), ((5,mark,1992),6
8000), ((7,james,1990),102000), ((6,peter,1991),68000), ((3,luke,1993),34000), ((2,john,1993),34000), ((4,lisa,1990),68000
), ((5,mark,1991),34000), ((9,thomas,1991),34000), ((5,mark,1994),34000), ((8,andrew,1990),34000), ((2,john,1991),68000),
((8,andrew,1991),34000), ((10,annie,1990),34000), ((9,thomas,1992),68000), ((6,peter,1993),34000), ((1,mark,1993),102000))
```

3) ***Considering age groups of < 20 , 20-35, 35 > ,Which age group is travelling the most every year.***

For this problem, first I've calculated the age group, using the below function.



```
scala>  def ageRangeCalculate(x: Int) :String={
     |  if (x < 20)
     |   "<20"
     |  else if (x > 35)
     |   ">35"
     |  else "20-35";
     |  }
ageRangeCalculate: (x: Int)String
```

```
def ageRangeCalculate(x: Int) :String={
 if (x < 20)
  "<20"
 else if (x > 35)
 ">35"
 else "20-35";
}
```

To find which group has travelled the most every year., we are calling the map function over the user_travel rdds 8th and 3rd element which refers to the travel_year, age . We are passing the age value to the function ageRangeCalculate, calculating the age group , then setting the year and agegroup as key and setting user_travel rdds 7th element, distance as value for each item.

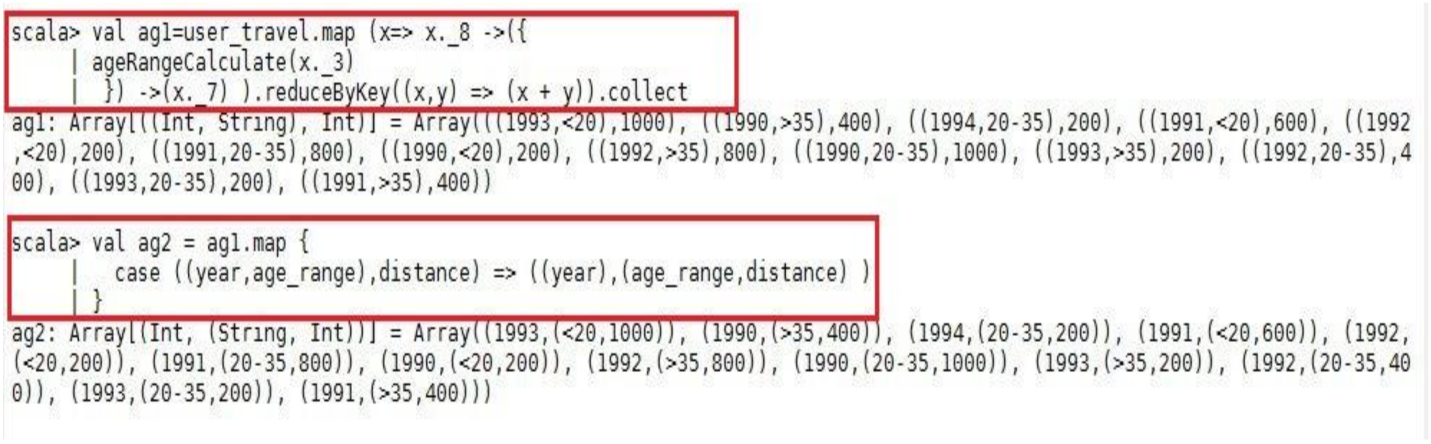
Then calling the reduceBykey, where the values for each key are aggregated using the given reduce function. Here we are adding the values for each key and finally calling the collect action.

```
val ag1=user_travel.map (x=> x._8 ->({
ageRangeCalculate(x._3)
 }) ->(x._7) ).reduceByKey((x,y) => (x + y)).collect
```

```
val ag2 = ag1.map {
  case ((year,age_range),distance) => ((year),(age_range,distance) )
}
```

In ag2, we are calling the map function over the ag1, and creating a key with year and age_range and distance as value for the key, since we need to find the travel happened for each year.

Spark-shell ouput



```
scala> val ag1=user_travel.map (x=> x._8 ->({
     | ageRangeCalculate(x._3)
     |  }) ->(x._7) ).reduceByKey((x,y) => (x + y)).collect
ag1: Array[((Int, String), Int)] = Array(((1993,<20),1000), ((1990,>35),400), ((1994,20-35),200), ((1991,<20),600), ((1992
,<20),200), ((1991,20-35),800), ((1990,<20),200), ((1992,>35),800), ((1990,20-35),1000), ((1993,>35),200), ((1992,20-35),4
00), ((1993,20-35),200), ((1991,>35),400))

scala> val ag2 = ag1.map {
     |   case ((year,age_range),distance) => ((year),(age_range,distance) )
     | }
ag2: Array[(Int, (String, Int))] = Array((1993,(<20,1000)), (1990,(>35,400)), (1994,(20-35,200)), (1991,(<20,600)), (1992,
(<20,200)), (1991,(20-35,800)), (1990,(<20,200)), (1992,(>35,800)), (1990,(20-35,1000)), (1993,(>35,200)), (1992,(20-35,40
0)), (1993,(20-35,200)), (1991,(>35,400)))
```

To calculate the most travelled age group, we are calling the reduceBykey over the ag2, where the values for each key are aggregated using the given reduce function group_most_travelled. Here we are calculating the maximum of the 2nd item of the values for each key and finally calling the collect action.

```
def group_most_travelled(x:(String,Int), y:(String,Int)) = if (x._2 > y._2) x else y
```

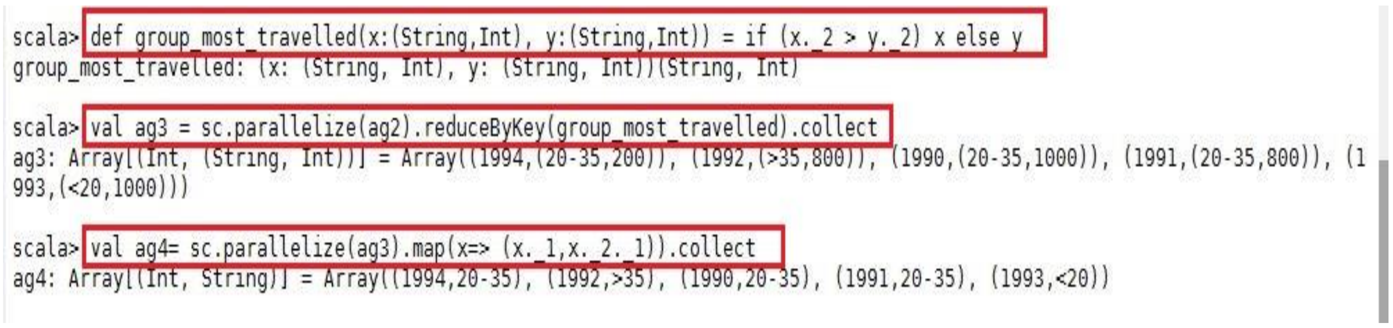
val ag3 = sc.parallelize(ag2).reduceByKey(group_most_travelled).collect

val ag4= sc.parallelize(ag3).map(x=> (x._1,x._2._1)).collect

Output

ag4: Array[(Int, String)] = Array((1994,20-35), (1992,>35), (1990,20-35), (1991,20-35), (1993,<20))

Spark-shell ouput



```
scala> def group_most_travelled(x:(String,Int), y:(String,Int)) = if (x._2 > y._2) x else y
group_most_travelled: (x: (String, Int), y: (String, Int))(String, Int)

scala> val ag3 = sc.parallelize(ag2).reduceByKey(group_most_travelled).collect
ag3: Array[(Int, (String, Int))] = Array((1994,(20-35,200)), (1992,(>35,800)), (1990,(20-35,1000)), (1991,(20-35,800)), (1
993,(<20,1000)))

scala> val ag4= sc.parallelize(ag3).map(x=> (x._1,x._2._1)).collect
ag4: Array[(Int, String)] = Array((1994,20-35), (1992,>35), (1990,20-35), (1991,20-35), (1993,<20))
```