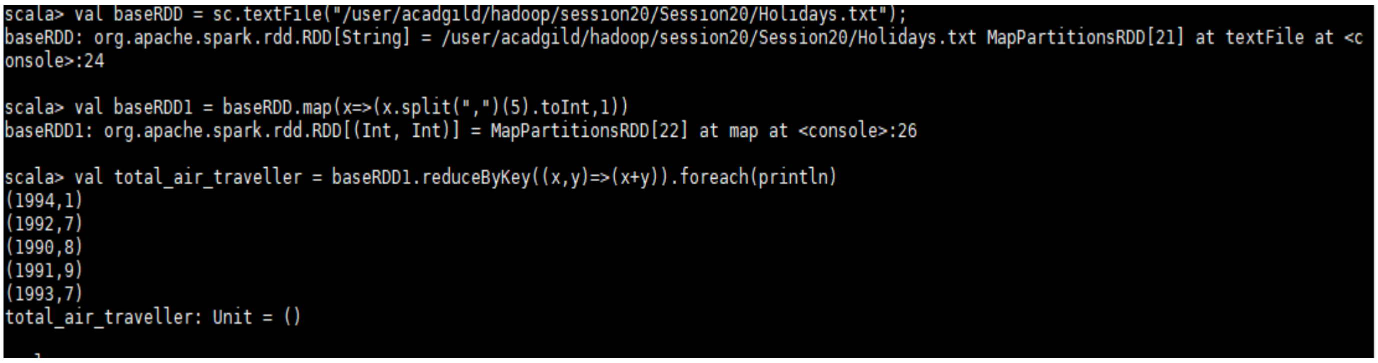
Task 1 - What is the distribution of the total number of air-travelers per year?

Codes used to achieve the above,

1. ***val baseRDD1 = baseRDD.map(x => (x.split(",")(5).toInt,1))***
2. ***val no_air_travelers = baseRDD1.reduceByKey((x,y)=>(x+y)).foreach(println)***

we are creating a tuple RDD baseRDD1 and mapping the key with numerical value 1. After which we are reducing the number of occurrences using reduceByKey and printing the result. Therefore, Total no of air travellers per year is,



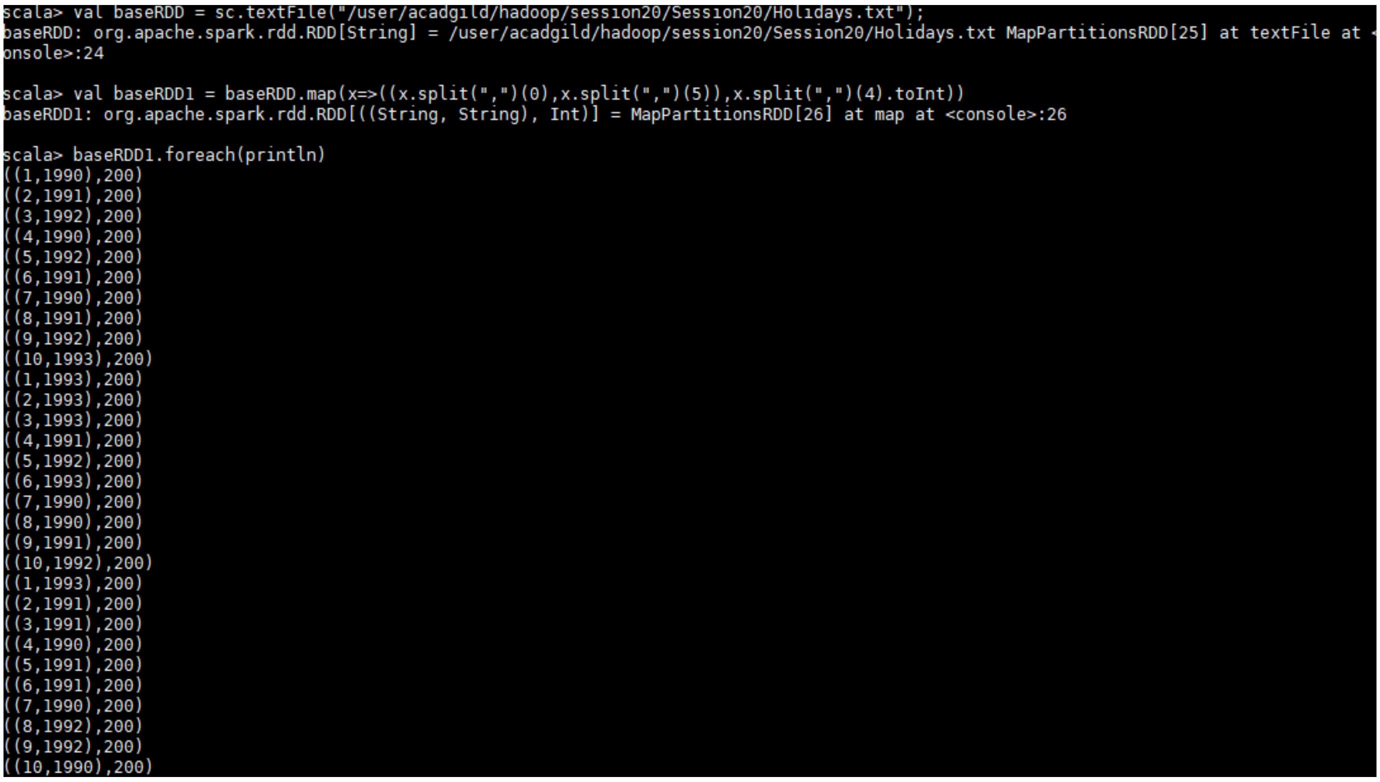
```
scala> val baseRDD = sc.textFile("/user/acadgild/hadoop/session20/Session20/Holidays.txt");
baseRDD: org.apache.spark.rdd.RDD[String] = /user/acadgild/hadoop/session20/Session20/Holidays.txt MapPartitionsRDD[21] at textFile at <c
onsole>:24

scala> val baseRDD1 = baseRDD.map(x=>(x.split(",")(5).toInt,1))
baseRDD1: org.apache.spark.rdd.RDD[(Int, Int)] = MapPartitionsRDD[22] at map at <console>:26

scala> val total_air_traveller = baseRDD1.reduceByKey((x,y)=>(x+y)).foreach(println)
(1994,1)
(1992,7)
(1990,8)
(1991,9)
(1993,7)
total_air_traveller: Unit = ()
```

Task 2 - What is the total air distance covered by each user per year?

Codes used to achieve the above,

1. ***val baseRDD2 = baseRDD.map(x => ((x.split(",")(0),x.split(",")(5)),x.split(",")(4).toInt))***
2. ***val distance_user = baseRDD2.reduceByKey((x,y) => (x + y)).foreach(println)***

We are creating a tuple rdd "baseRDD2" and mapping the key and value. Here the userID, year acts as key and the travel distance is value.



```
scala> val baseRDD = sc.textFile("/user/acadgild/hadoop/session20/Session20/Holidays.txt");
baseRDD: org.apache.spark.rdd.RDD[String] = /user/acadgild/hadoop/session20/Session20/Holidays.txt MapPartitionsRDD[25] at textFile at <
onsole>:24

scala> val baseRDD1 = baseRDD.map(x=>((x.split(",")(0),x.split(",")(5)),x.split(",")(4).toInt))
baseRDD1: org.apache.spark.rdd.RDD[((String, String), Int)] = MapPartitionsRDD[26] at map at <console>:26

scala> baseRDD1.foreach(println)
((1,1990),200)
((2,1991),200)
((3,1992),200)
((4,1990),200)
((5,1992),200)
((6,1991),200)
((7,1990),200)
((8,1991),200)
((9,1992),200)
((10,1993),200)
((1,1993),200)
((2,1993),200)
((3,1993),200)
((4,1991),200)
((5,1992),200)
((6,1993),200)
((7,1990),200)
((8,1990),200)
((9,1991),200)
((10,1992),200)
((1,1993),200)
((2,1991),200)
((3,1991),200)
((4,1990),200)
((5,1991),200)
((6,1991),200)
((7,1990),200)
((8,1992),200)
((9,1992),200)
((10,1990),200)
```

Task 3 - Which user has travelled the largest distance till date?

Codes used below,

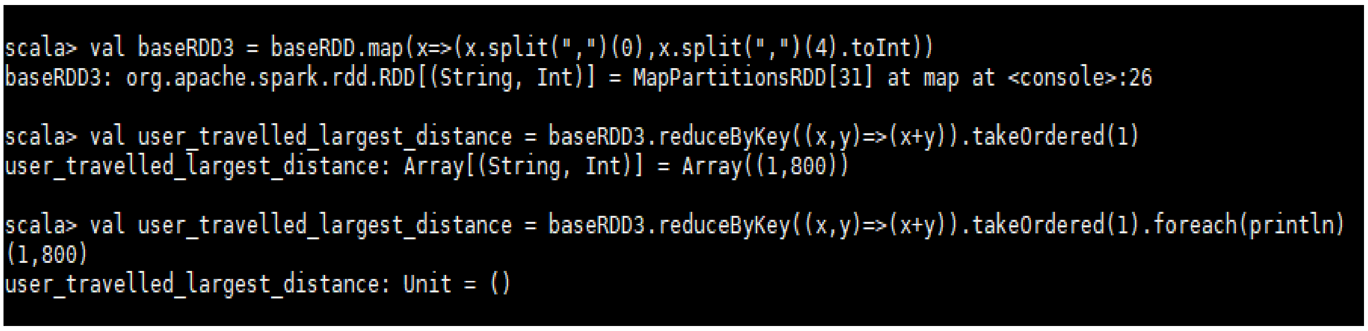
1. ***val baseRDD3 = baseRDD.map(x=> (x.split(",")(0),x.split(",")(4).toInt))***

2. ***val largest_dist = baseRDD3.reduceByKey((x,y)=>(x+y)).takeOrdered(1)***

The tuple rdd “baseRDD3” is created to map the key and value from the baseRDD. Here the userID and is key and the travel distance is value,

In the 2nd step, we are reducing the number of occurrences using reduceByKey and using the takeOrdered function to get the result,

largest_dist: Array[(String, Int)] = Array((1,800))



```
scala> val baseRDD3 = baseRDD.map(x=>(x.split(",")(0),x.split(",")(4).toInt))
baseRDD3: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[31] at map at <console>:26

scala> val user_travelled_largest_distance = baseRDD3.reduceByKey((x,y)=>(x+y)).takeOrdered(1)
user_travelled_largest_distance: Array[(String, Int)] = Array((1,800))

scala> val user_travelled_largest_distance = baseRDD3.reduceByKey((x,y)=>(x+y)).takeOrdered(1).foreach(println)
(1,800)
user_travelled_largest_distance: Unit = ()
```

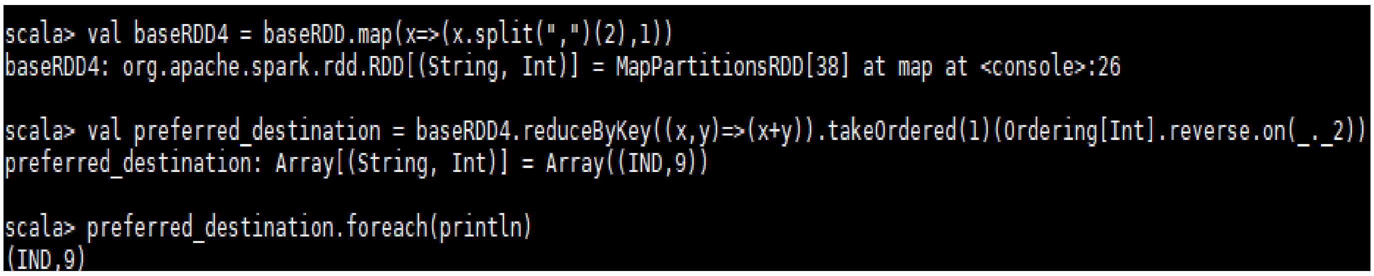
Task 4 – What is the most preferred destination for all users?

Codes used below,

1. ***val baseRDD4 = baseRDD.map(x => (x.split(",")(2),1))***
2. ***val dest = baseRDD4.reduceByKey((x,y)=>(x+y))***
3. ***val dest =***

   baseRDD4.reduceByKey((x,y)=>(x+y)).takeOrdered(1)(Ordering[Int].reverse.on(_._2))

A tuple rdd created with the destination as key and numerical 1 as value, and we are reducing the number of occurrences using the reduceByKey. Now, the most preferred destination is taken by using the function takeOrdered and ordering the values descending so that we can get the required output.



```
scala> val baseRDD4 = baseRDD.map(x=>(x.split(",")(2),1))
baseRDD4: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[38] at map at <console>:26

scala> val preferred_destination = baseRDD4.reduceByKey((x,y)=>(x+y)).takeOrdered(1)(Ordering[Int].reverse.on(_._2))
preferred_destination: Array[(String, Int)] = Array((IND,9))

scala> preferred_destination.foreach(println)
(IND,9)
```

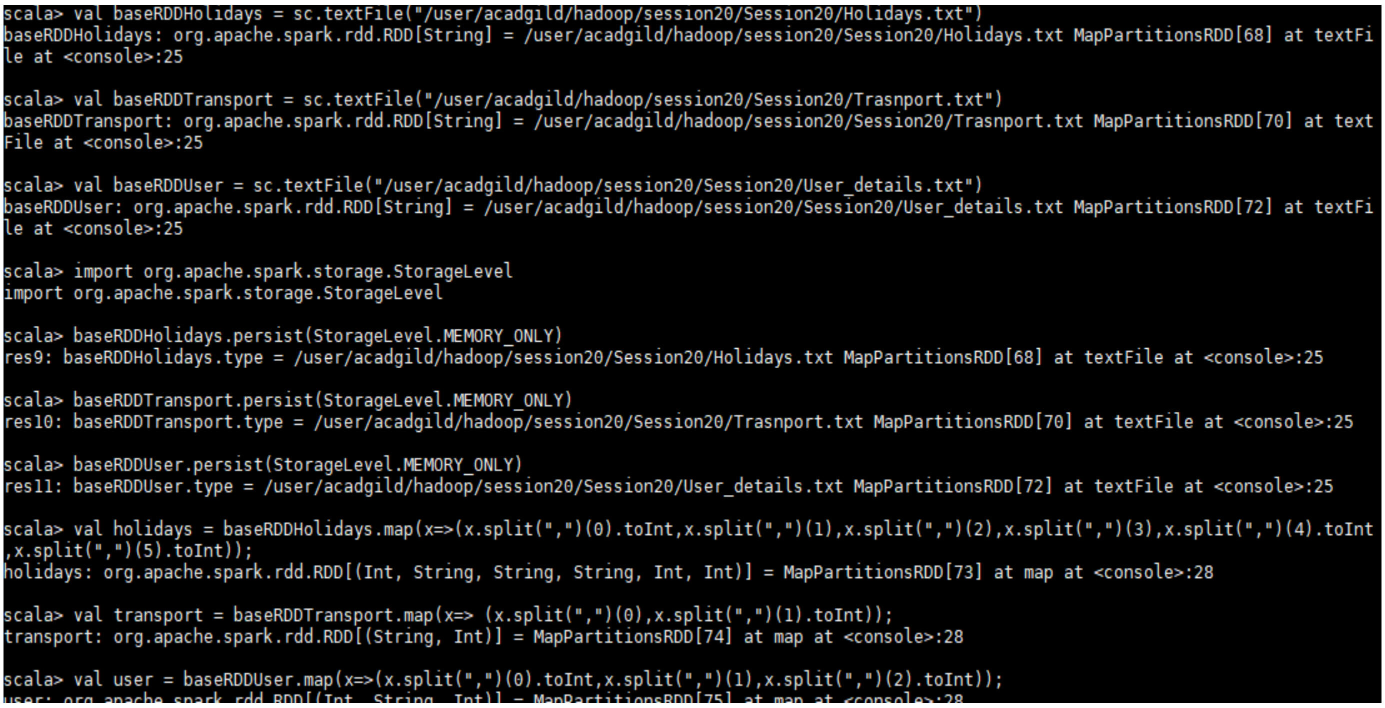
Task 5 - Which route is generating the most revenue per year?

We are loading the dataset’s in the name of **holidays**, **transport** and **user** RDD’s.

val holidays = baseRDDHolidays.map(x=>(x.split(",")(0).toInt,x.split(",")(1),x.split(",")(2),x.split(",")(3),x.split(",")(4).toInt,x.s plit(",")(5).toInt))

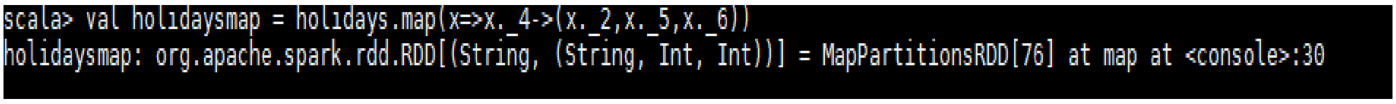
val transport = baseRDDTransport.map(x=> (x.split(",")(0),x.split(",")(1).toInt))

val user = baseRDDUser.map(x=>(x.split(",")(0).toInt,x.split(",")(1),x.split(",")(2).toInt))



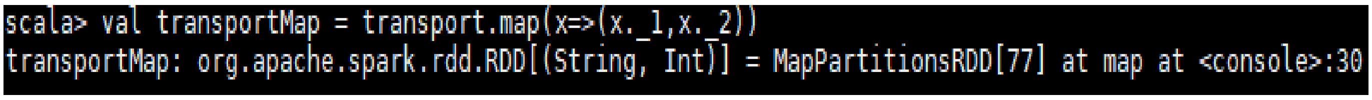
```
scala> val baseRDDHolidays = sc.textFile("/user/acadgild/hadoop/session20/Session20/Holidays.txt")
baseRDDHolidays: org.apache.spark.rdd.RDD[String] = /user/acadgild/hadoop/session20/Session20/Holidays.txt MapPartitionsRDD[68] at textFi
le at <console>:25

scala> val baseRDDTransport = sc.textFile("/user/acadgild/hadoop/session20/Session20/Trasnport.txt")
baseRDDTransport: org.apache.spark.rdd.RDD[String] = /user/acadgild/hadoop/session20/Session20/Trasnport.txt MapPartitionsRDD[70] at text
File at <console>:25

scala> val baseRDDUser = sc.textFile("/user/acadgild/hadoop/session20/Session20/User_details.txt")
baseRDDUser: org.apache.spark.rdd.RDD[String] = /user/acadgild/hadoop/session20/Session20/User_details.txt MapPartitionsRDD[72] at textFi
le at <console>:25

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

scala> baseRDDHolidays.persist(StorageLevel.MEMORY_ONLY)
res9: baseRDDHolidays.type = /user/acadgild/hadoop/session20/Session20/Holidays.txt MapPartitionsRDD[68] at textFile at <console>:25

scala> baseRDDTransport.persist(StorageLevel.MEMORY_ONLY)
res10: baseRDDTransport.type = /user/acadgild/hadoop/session20/Session20/Trasnport.txt MapPartitionsRDD[70] at textFile at <console>:25

scala> baseRDDUser.persist(StorageLevel.MEMORY_ONLY)
res11: baseRDDUser.type = /user/acadgild/hadoop/session20/Session20/User_details.txt MapPartitionsRDD[72] at textFile at <console>:25

scala> val holidays = baseRDDHolidays.map(x=>(x.split(",")(0).toInt,x.split(",")(1),x.split(",")(2),x.split(",")(3),x.split(",")(4).toInt
,x.split(",")(5).toInt));
holidays: org.apache.spark.rdd.RDD[(Int, String, String, String, Int, Int)] = MapPartitionsRDD[73] at map at <console>:28

scala> val transport = baseRDDTransport.map(x=> (x.split(",")(0),x.split(",")(1).toInt));
transport: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[74] at map at <console>:28

scala> val user = baseRDDUser.map(x=>(x.split(",")(0).toInt,x.split(",")(1),x.split(",")(2).toInt));
user: org.apache.spark.rdd.RDD[(Int, String, Int)] = MapPartitionsRDD[75] at map at <console>:28
```

Step 1 – we are mapping the key and value from the base RDD holidays as travel mode as key and the destination, distance and the year as value.



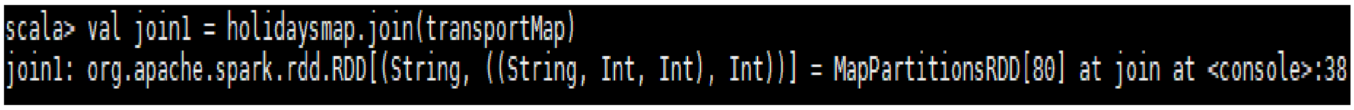
```
scala> val holidaysmap = holidays.map(x=>x._4->(x._2,x._5,x._6))
holidaysmap: org.apache.spark.rdd.RDD[(String, (String, Int, Int))] = MapPartitionsRDD[76] at map at <console>:30
```

Step -2 – same as step 1, we are creating a tuple RDD as travel mode as key and the rate as values, shown below.



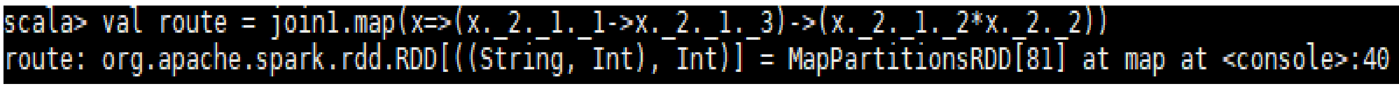
```
scala> val transportMap = transport.map(x=>(x._1,x._2))
transportMap: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[77] at map at <console>:30
```

Step 3 – we are joining the 2 RDD’s holidaysmap and the transport using join function



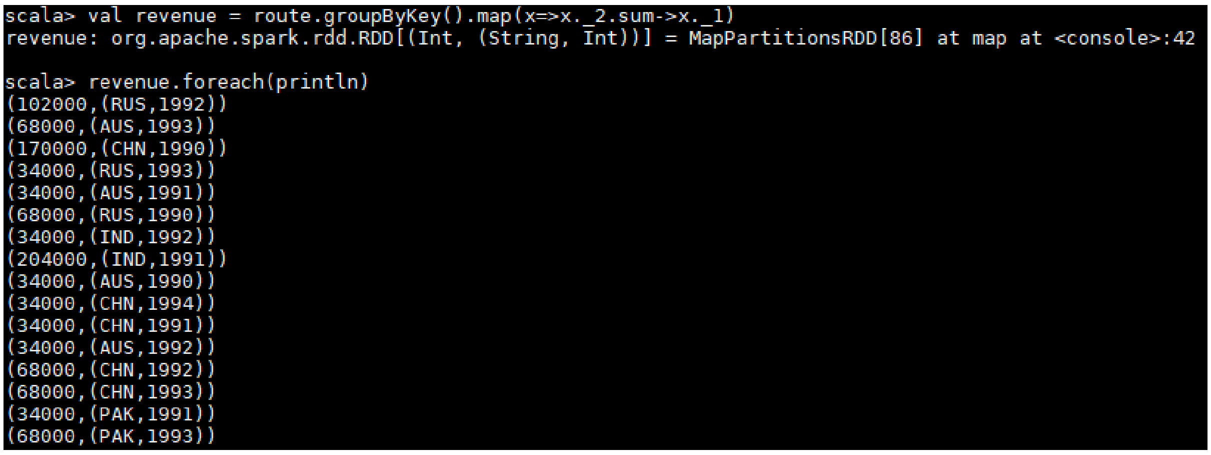
```
scala> val join1 = holidaysmap.join(transportMap)
join1: org.apache.spark.rdd.RDD[(String, ((String, Int, Int), Int))] = MapPartitionsRDD[80] at join at <console>:38
```

Step 4 – we are mapping the new RDD join1 as below, destination & year as key and the values as multiplication of the cost and the distance.



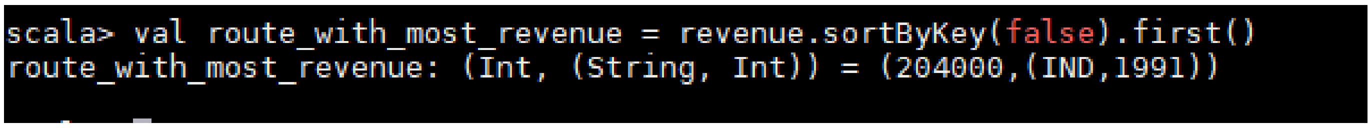
```
scala> val route = join1.map(x=>(x._2._1._1->x._2._1._3)->(x._2._1._2*x._2._2))
route: org.apache.spark.rdd.RDD[((String, Int), Int)] = MapPartitionsRDD[81] at map at <console>:40
```

Step 5 – using groupByKey function, we are grouping the destination & year with the sum of the costs.



```
scala> val revenue = route.groupByKey().map(x=>x._2.sum->x._1)
revenue: org.apache.spark.rdd.RDD[(Int, (String, Int))] = MapPartitionsRDD[86] at map at <console>:42

scala> revenue.foreach(println)
(102000,(RUS,1992))
(68000,(AUS,1993))
(170000,(CHN,1990))
(34000,(RUS,1993))
(34000,(AUS,1991))
(68000,(RUS,1990))
(34000,(IND,1992))
(204000,(IND,1991))
(34000,(AUS,1990))
(34000,(CHN,1994))
(34000,(CHN,1991))
(34000,(AUS,1992))
(68000,(CHN,1992))
(68000,(CHN,1993))
(34000,(PAK,1991))
(68000,(PAK,1993))
```

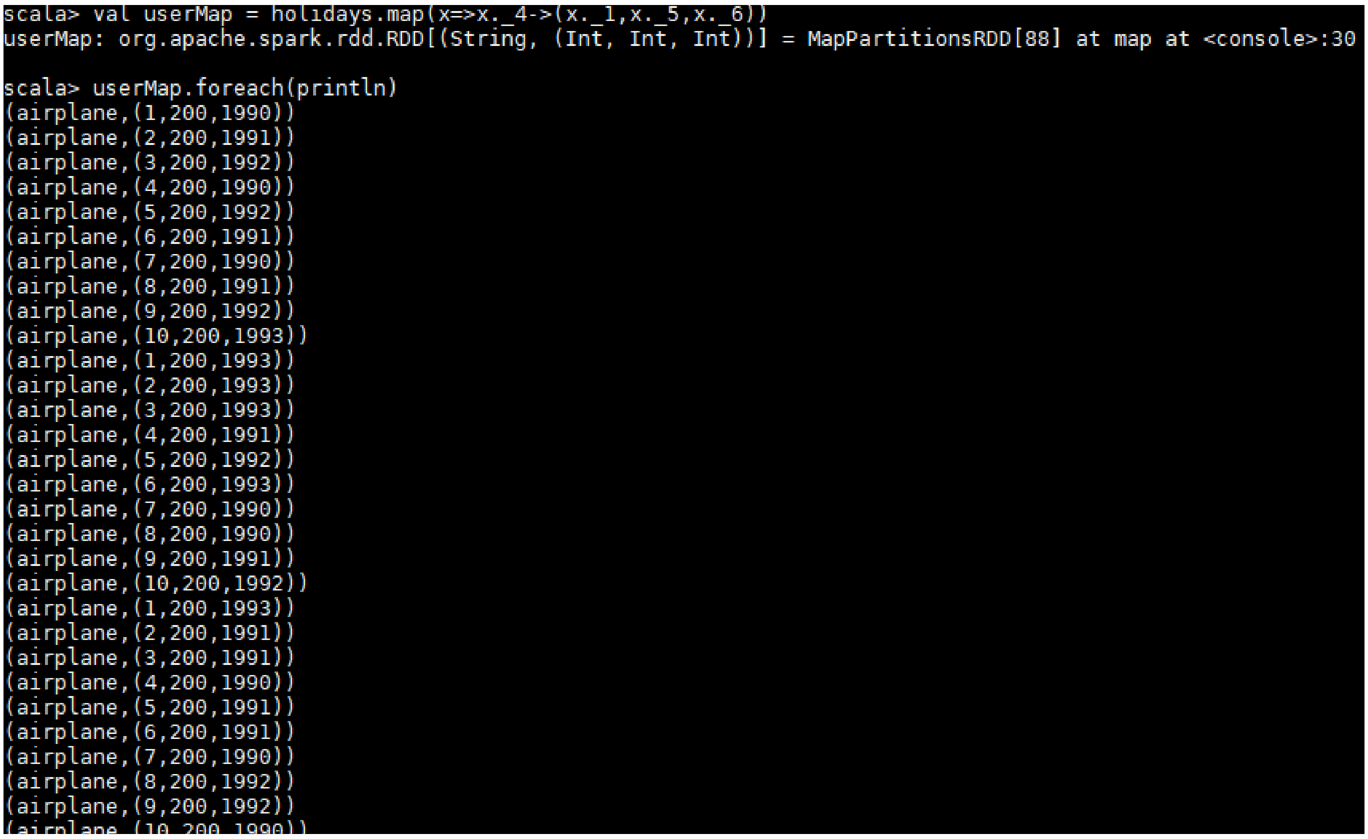
Desired output,



```
scala> val route_with_most_revenue = revenue.sortByKey(false).first()
route_with_most_revenue: (Int, (String, Int)) = (204000,(IND,1991))
```

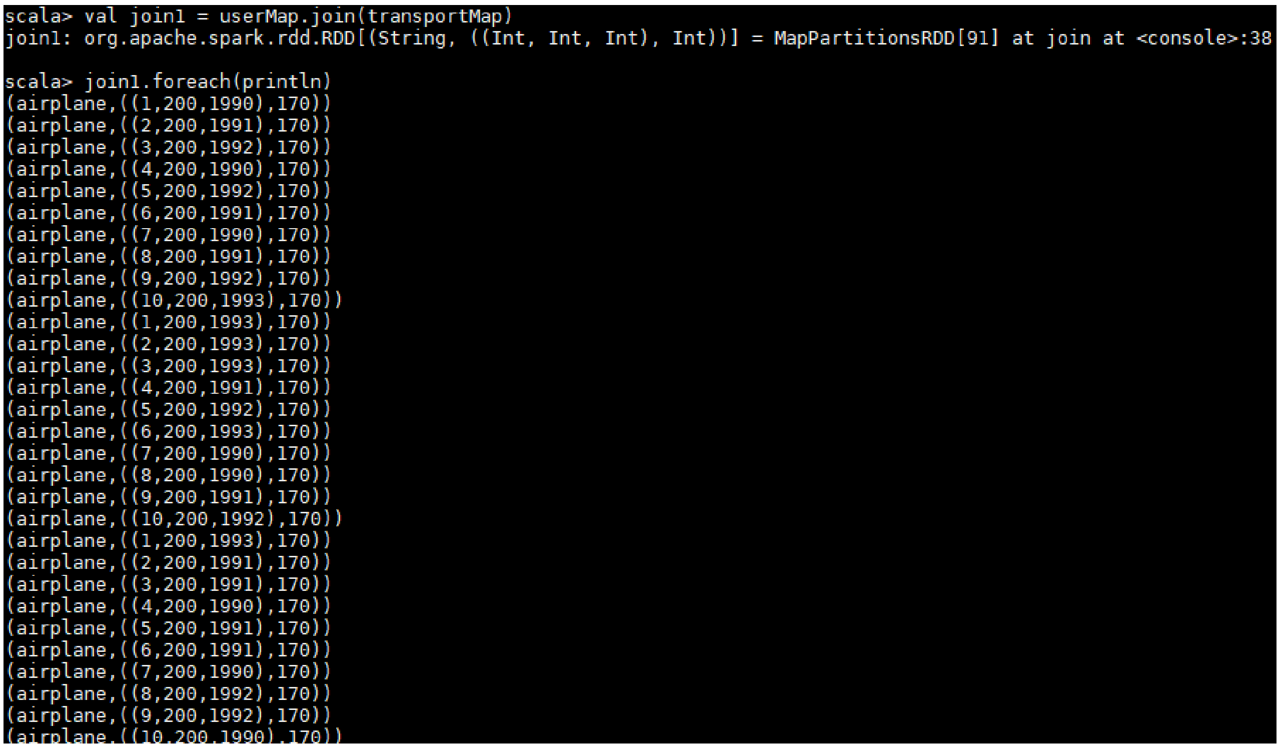
Task 6 - What is the total amount spent by every user on air-travel per year?

Step -1 – we are creating a tuple rdd from a baseRDD ***holidays*** making the travel mode as Key and the userID, distance & year as values.



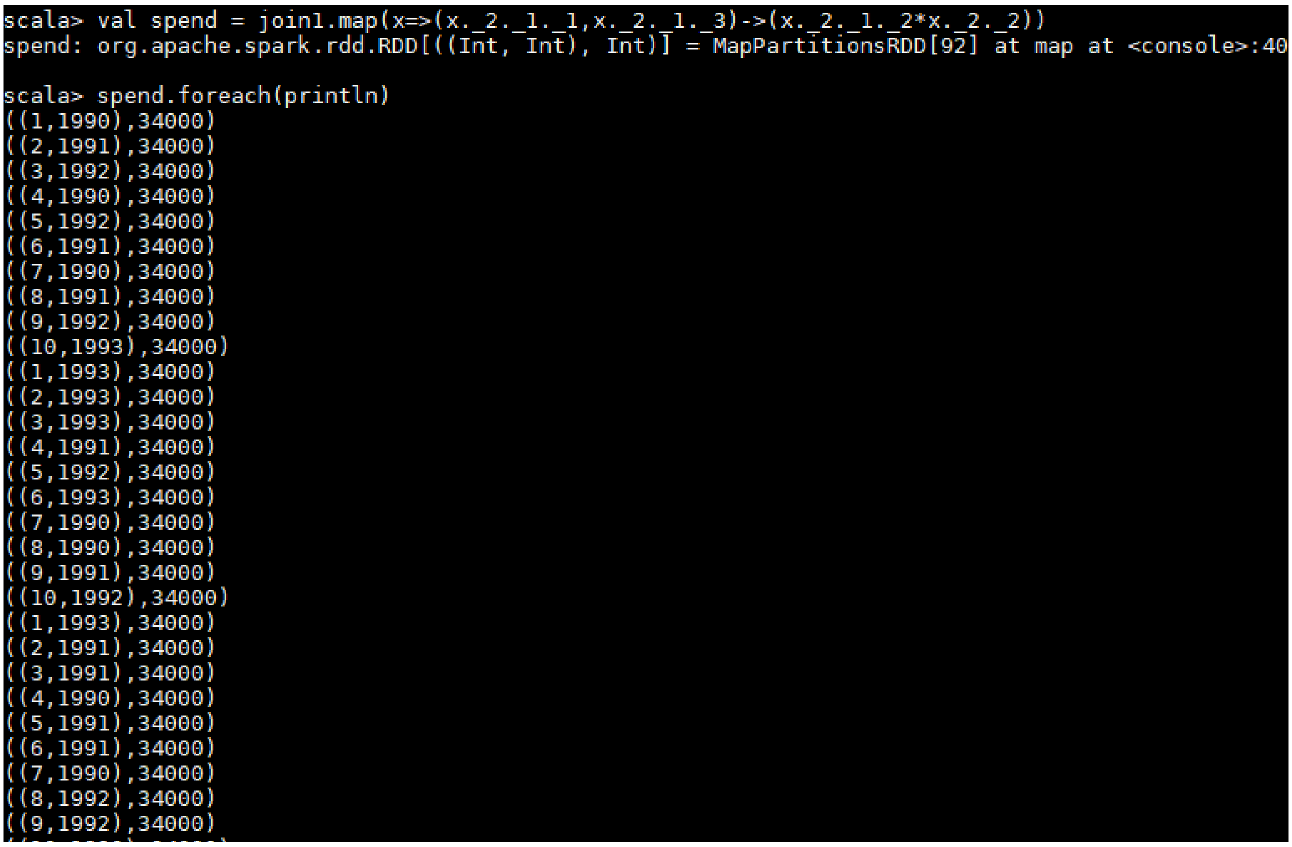
```
scala> val userMap = holidays.map(x=>x._4->(x._1,x._5,x._6))
userMap: org.apache.spark.rdd.RDD[(String, (Int, Int, Int))] = MapPartitionsRDD[88] at map at <console>:30

scala> userMap.foreach(println)
(airplane,(1,200,1990))
(airplane,(2,200,1991))
(airplane,(3,200,1992))
(airplane,(4,200,1990))
(airplane,(5,200,1992))
(airplane,(6,200,1991))
(airplane,(7,200,1990))
(airplane,(8,200,1991))
(airplane,(9,200,1992))
(airplane,(10,200,1993))
(airplane,(1,200,1993))
(airplane,(2,200,1993))
(airplane,(3,200,1993))
(airplane,(4,200,1991))
(airplane,(5,200,1992))
(airplane,(6,200,1993))
(airplane,(7,200,1990))
(airplane,(8,200,1990))
(airplane,(9,200,1991))
(airplane,(10,200,1992))
(airplane,(1,200,1993))
(airplane,(2,200,1991))
(airplane,(3,200,1991))
(airplane,(4,200,1990))
(airplane,(5,200,1991))
(airplane,(6,200,1991))
(airplane,(7,200,1990))
(airplane,(8,200,1992))
(airplane,(9,200,1992))
(airplane,(10,200,1990))
```

Step -2 – we are joining the created tuple RDD **userMap** with the already created tuple RDD **transportMap** using the join function.



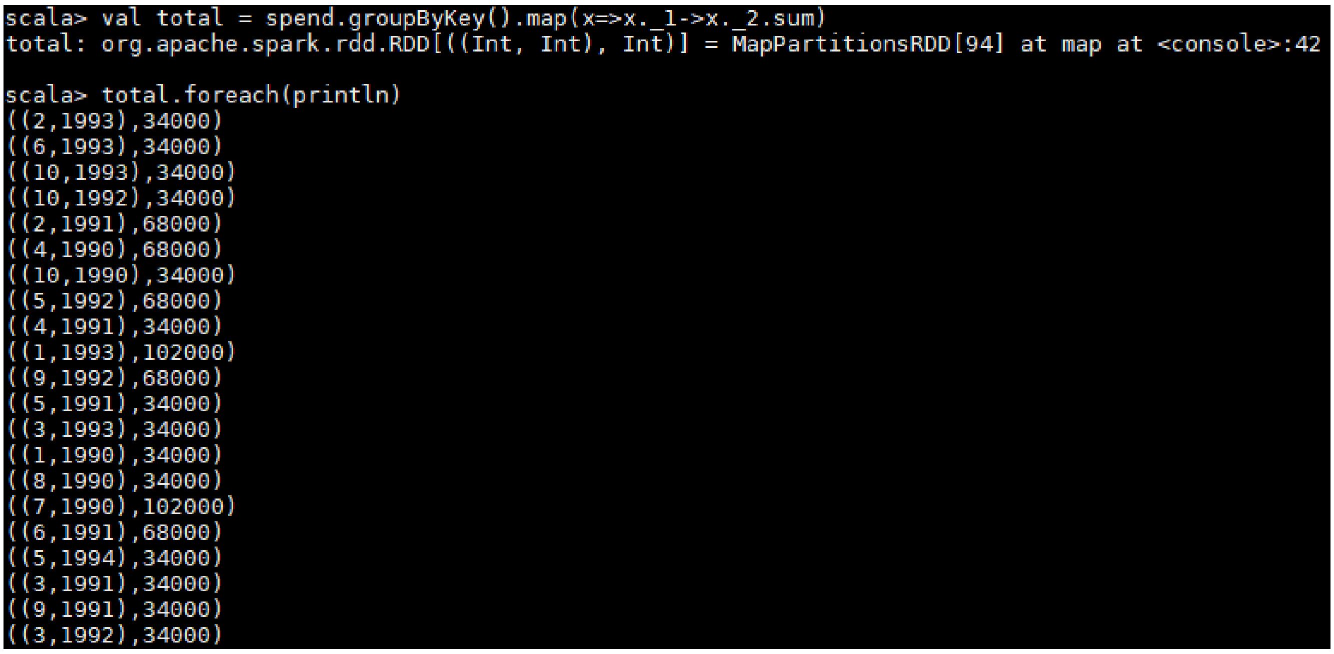
```
scala> val join1 = userMap.join(transportMap)
join1: org.apache.spark.rdd.RDD[(String, ((Int, Int, Int), Int))] = MapPartitionsRDD[91] at join at <console>:38

scala> join1.foreach(println)
(airplane,((1,200,1990),170))
(airplane,((2,200,1991),170))
(airplane,((3,200,1992),170))
(airplane,((4,200,1990),170))
(airplane,((5,200,1992),170))
(airplane,((6,200,1991),170))
(airplane,((7,200,1990),170))
(airplane,((8,200,1991),170))
(airplane,((9,200,1992),170))
(airplane,((10,200,1993),170))
(airplane,((1,200,1993),170))
(airplane,((2,200,1993),170))
(airplane,((3,200,1993),170))
(airplane,((4,200,1991),170))
(airplane,((5,200,1992),170))
(airplane,((6,200,1993),170))
(airplane,((7,200,1990),170))
(airplane,((8,200,1990),170))
(airplane,((9,200,1991),170))
(airplane,((10,200,1992),170))
(airplane,((1,200,1993),170))
(airplane,((2,200,1991),170))
(airplane,((3,200,1991),170))
(airplane,((4,200,1990),170))
(airplane,((5,200,1991),170))
(airplane,((6,200,1991),170))
(airplane,((7,200,1990),170))
(airplane,((8,200,1992),170))
(airplane,((9,200,1992),170))
(airplane,((10,200,1990),170))
```

Step – 3 – now, we are calculating the expenditure for each user by multiplying the distance and the amount spent for the travel mode airplane,



```
scala> val spend = join1.map(x=>(x._2._1._1,x._2._1._3)->(x._2._1._2*x._2._2))
spend: org.apache.spark.rdd.RDD[((Int, Int), Int)] = MapPartitionsRDD[92] at map at <console>:40

scala> spend.foreach(println)
((1,1990),34000)
((2,1991),34000)
((3,1992),34000)
((4,1990),34000)
((5,1992),34000)
((6,1991),34000)
((7,1990),34000)
((8,1991),34000)
((9,1992),34000)
((10,1993),34000)
((1,1993),34000)
((2,1993),34000)
((3,1993),34000)
((4,1991),34000)
((5,1992),34000)
((6,1993),34000)
((7,1990),34000)
((8,1990),34000)
((9,1991),34000)
((10,1992),34000)
((1,1993),34000)
((2,1991),34000)
((3,1991),34000)
((4,1990),34000)
((5,1991),34000)
((6,1991),34000)
((7,1990),34000)
((8,1992),34000)
((9,1992),34000)
```

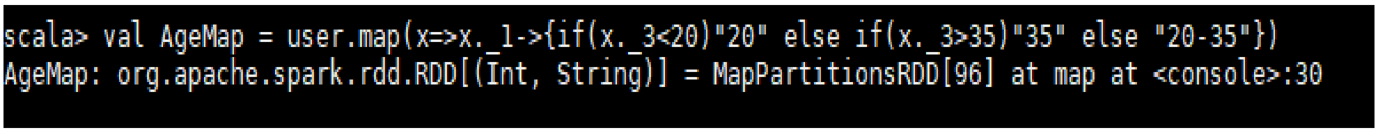
In the final step, we are summing the total value for each user yearly wise, please see the expected result in the below screen shot.



```
scala> val total = spend.groupByKey().map(x=>x._1->x._2.sum)
total: org.apache.spark.rdd.RDD[((Int, Int), Int)] = MapPartitionsRDD[94] at map at <console>:42

scala> total.foreach(println)
((2,1993),34000)
((6,1993),34000)
((10,1993),34000)
((10,1992),34000)
((2,1991),68000)
((4,1990),68000)
((10,1990),34000)
((5,1992),68000)
((4,1991),34000)
((1,1993),102000)
((9,1992),68000)
((5,1991),34000)
((3,1993),34000)
((1,1990),34000)
((8,1990),34000)
((7,1990),102000)
((6,1991),68000)
((5,1994),34000)
((3,1991),34000)
((9,1991),34000)
((3,1992),34000)
```

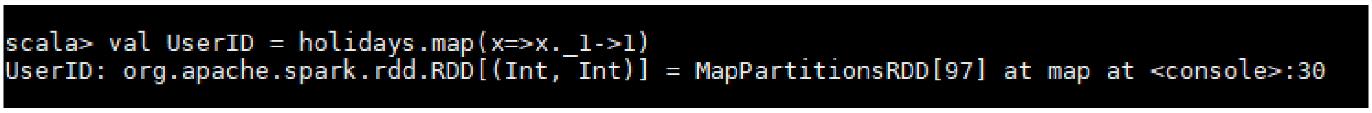
Task 7 - Considering age groups of < 20, 20-35, 35 >, which age group is travelling the most every year.

In Order to considering particular age groups, we are using a below if, else logic to define a RDD **AgeMap** which gives you a set of age groups,



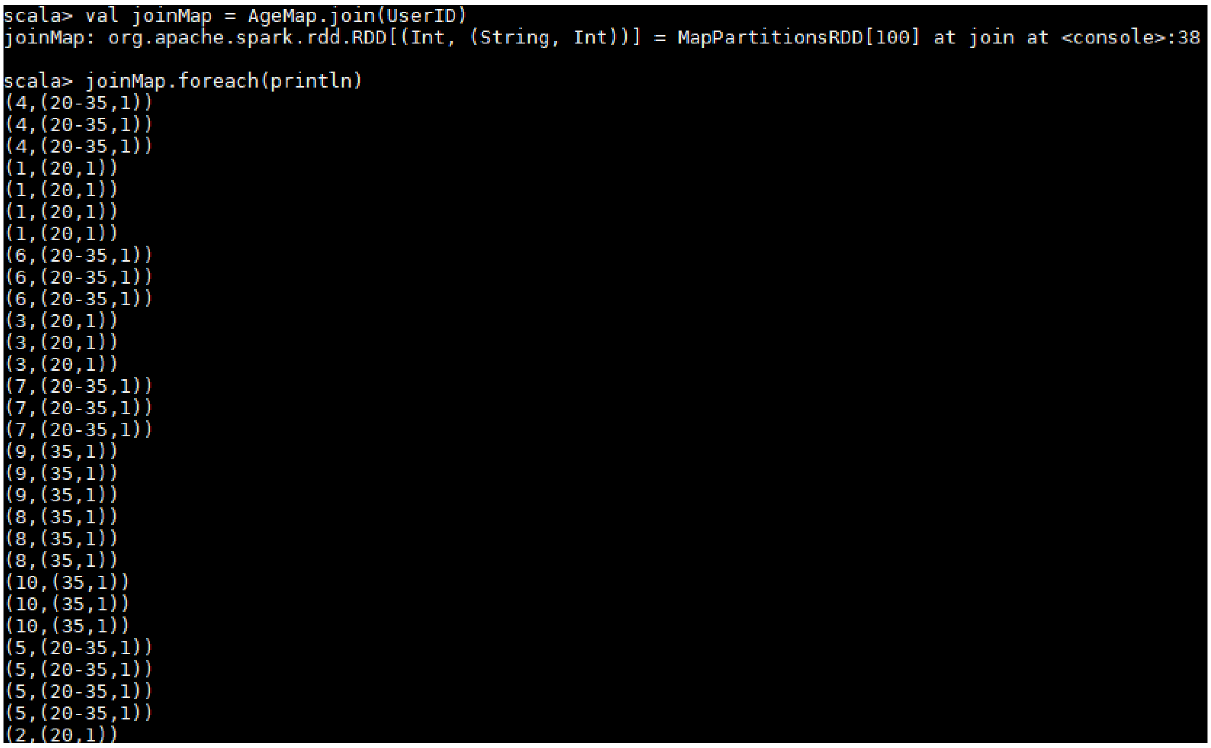
```
scala> val AgeMap = user.map(x=>x._1->{if(x._3<20)"20" else if(x._3>35)"35" else "20-35"})
AgeMap: org.apache.spark.rdd.RDD[(Int, String)] = MapPartitionsRDD[96] at map at <console>:30
```

Step – 1- we are just mapping the user ID from the RDD **holidays** with the numerical 1.



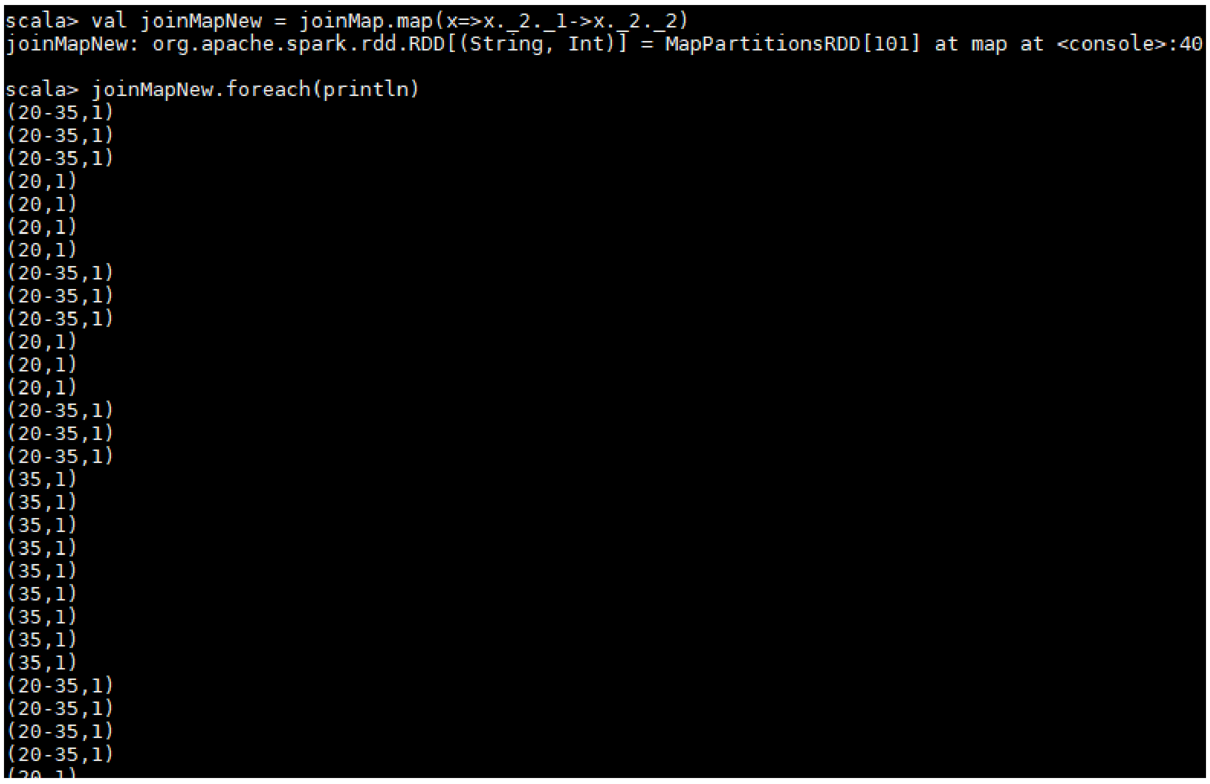
```
scala> val UserID = holidays.map(x=>x._1->1)
UserID: org.apache.spark.rdd.RDD[(Int, Int)] = MapPartitionsRDD[97] at map at <console>:30
```

Step -2 – In this step, we are joining the 2 RDD’s **UserID** and the **AgeMap,** So we are getting the below tuple RDD



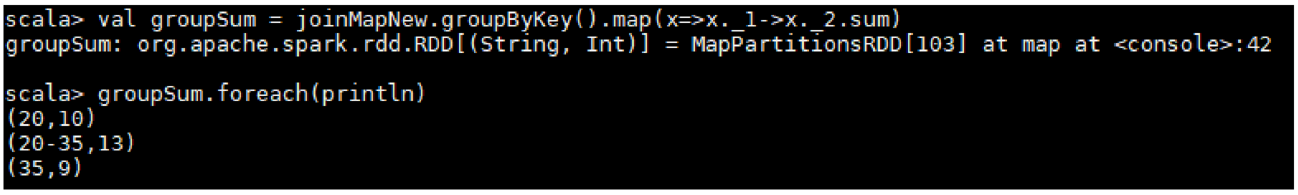
```
scala> val joinMap = AgeMap.join(UserID)
joinMap: org.apache.spark.rdd.RDD[(Int, (String, Int))] = MapPartitionsRDD[100] at join at <console>:38

scala> joinMap.foreach(println)
(4,(20-35,1))
(4,(20-35,1))
(4,(20-35,1))
(1,(20,1))
(1,(20,1))
(1,(20,1))
(1,(20,1))
(6,(20-35,1))
(6,(20-35,1))
(6,(20-35,1))
(3,(20,1))
(3,(20,1))
(3,(20,1))
(7,(20-35,1))
(7,(20-35,1))
(7,(20-35,1))
(9,(35,1))
(9,(35,1))
(9,(35,1))
(8,(35,1))
(8,(35,1))
(8,(35,1))
(10,(35,1))
(10,(35,1))
(10,(35,1))
(5,(20-35,1))
(5,(20-35,1))
(5,(20-35,1))
(5,(20-35,1))
(2,(20,1))
```

Step -3 – we are just eliminating the user ID in this step.



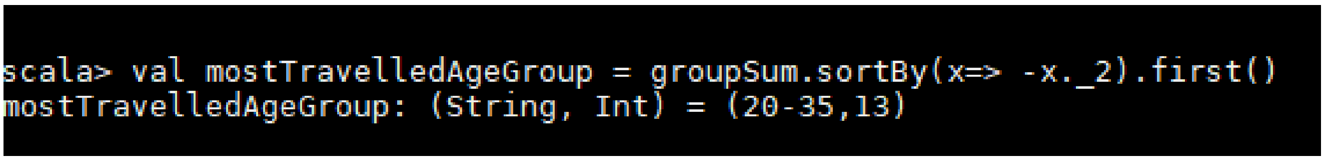
```
scala> val joinMapNew = joinMap.map(x=>x._2._1->x._2._2)
joinMapNew: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[101] at map at <console>:40

scala> joinMapNew.foreach(println)
(20-35,1)
(20-35,1)
(20-35,1)
(20,1)
(20,1)
(20,1)
(20,1)
(20-35,1)
(20-35,1)
(20-35,1)
(20,1)
(20,1)
(20,1)
(20-35,1)
(20-35,1)
(20-35,1)
(35,1)
(35,1)
(35,1)
(35,1)
(35,1)
(35,1)
(35,1)
(35,1)
(35,1)
(20-35,1)
(20-35,1)
(20-35,1)
(20-35,1)
```

Step – 4 – we just summed the total value by grouping the Age Group,



```
scala> val groupSum = joinMapNew.groupByKey().map(x=>x._1->x._2.sum)
groupSum: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[103] at map at <console>:42

scala> groupSum.foreach(println)
(20,10)
(20-35,13)
(35,9)
```

Step -5 - We use the function **first()** to find the age group who is travelling the most every year from the given dataset. The expected output shown below,



```
scala> val mostTravelledAgeGroup = groupSum.sortBy(x=> -x._2).first()
mostTravelledAgeGroup: (String, Int) = (20-35,13)
```