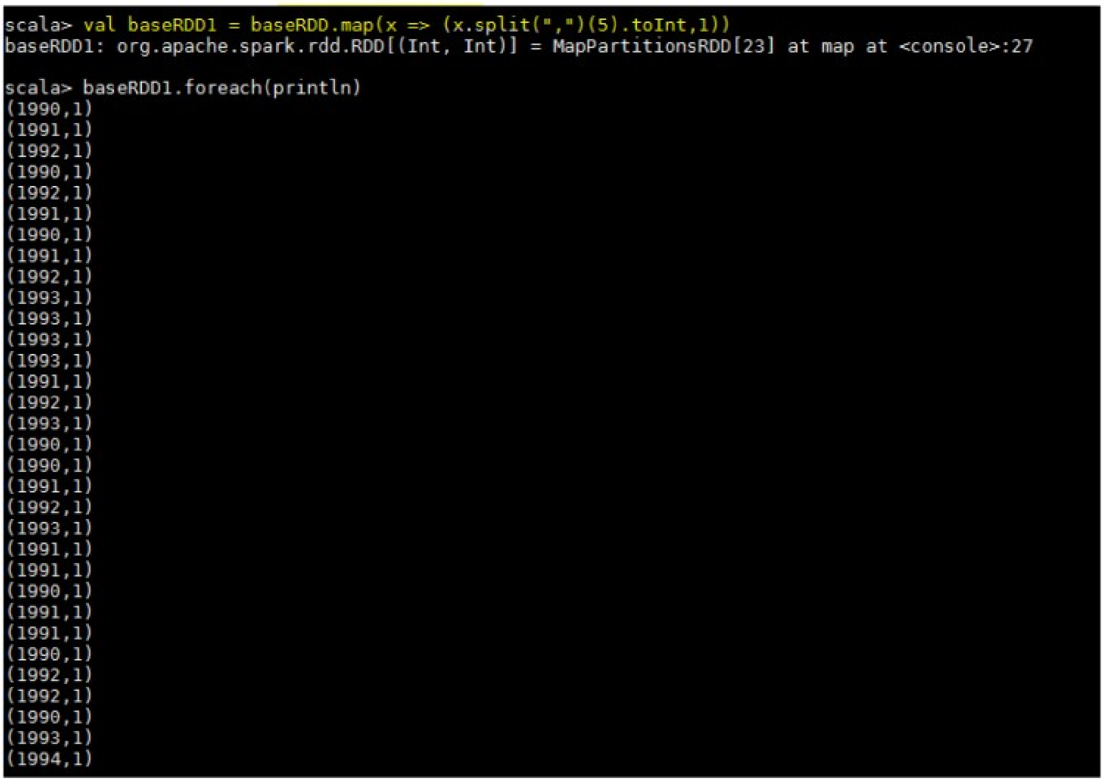
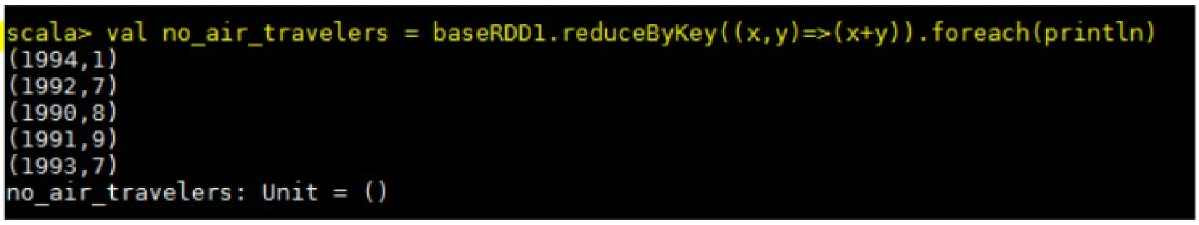
1) What is the distribution of the total number of air-travelers per year

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.



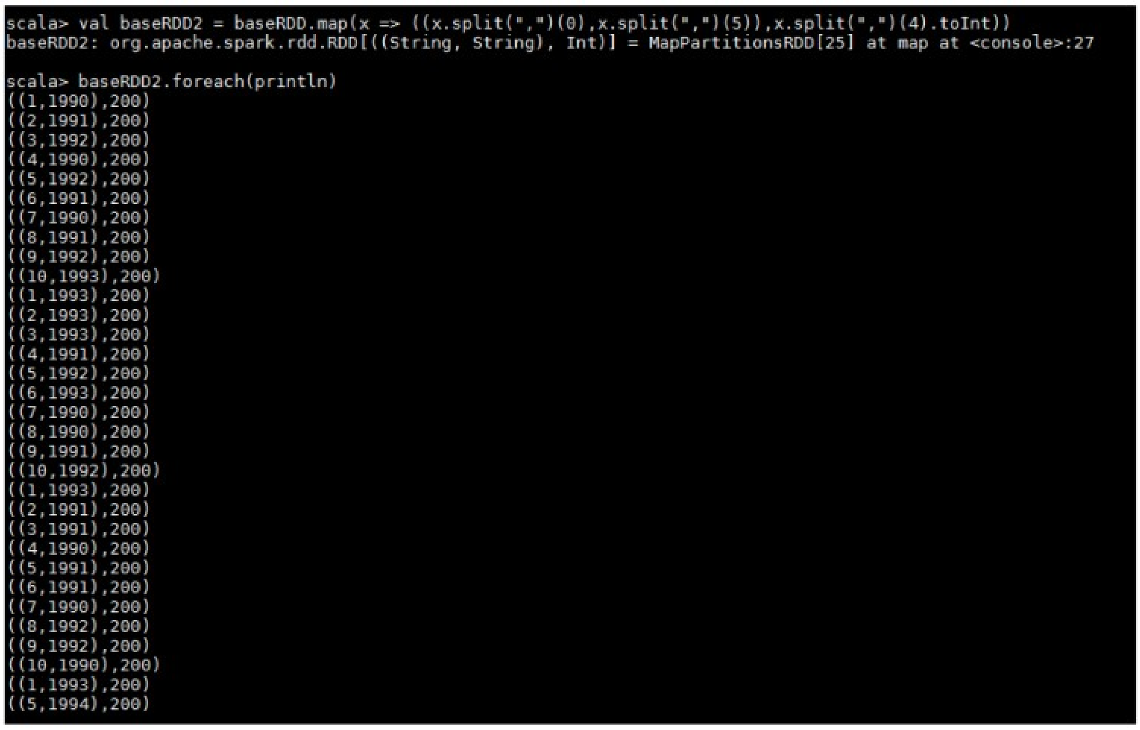
```
scala> val baseRDD1 = baseRDD.map(x => (x.split(",")(5).toInt,1))
baseRDD1: org.apache.spark.rdd.RDD[(Int, Int)] = MapPartitionsRDD[23] at map at <console>:27

scala> baseRDD1.foreach(println)
(1990,1)
(1991,1)
(1992,1)
(1990,1)
(1992,1)
(1991,1)
(1990,1)
(1991,1)
(1992,1)
(1993,1)
(1993,1)
(1993,1)
(1993,1)
(1991,1)
(1992,1)
(1993,1)
(1990,1)
(1990,1)
(1991,1)
(1992,1)
(1993,1)
(1991,1)
(1991,1)
(1990,1)
(1991,1)
(1991,1)
(1990,1)
(1992,1)
(1992,1)
(1990,1)
(1993,1)
(1994,1)
```



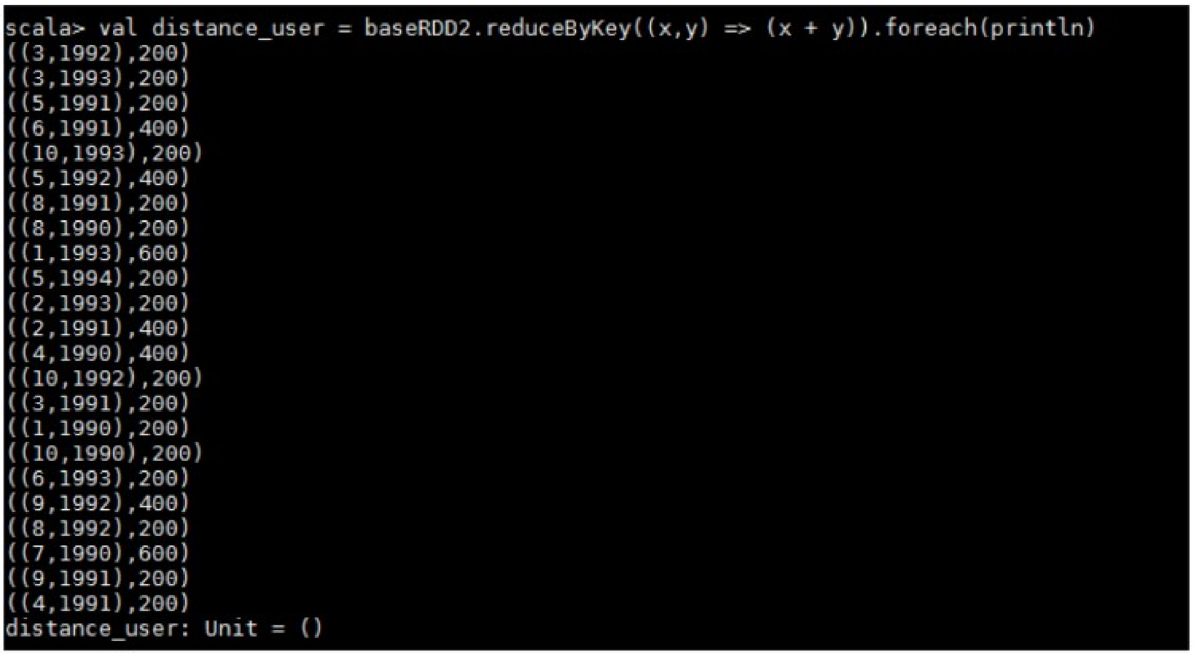
```
scala> val no_air_travelers = baseRDD1.reduceByKey((x,y)=>(x+y)).foreach(println)
(1994,1)
(1992,7)
(1990,8)
(1991,9)
(1993,7)
no_air_travelers: Unit = ()
```

2) What is the total air distance covered by each user per year

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)***



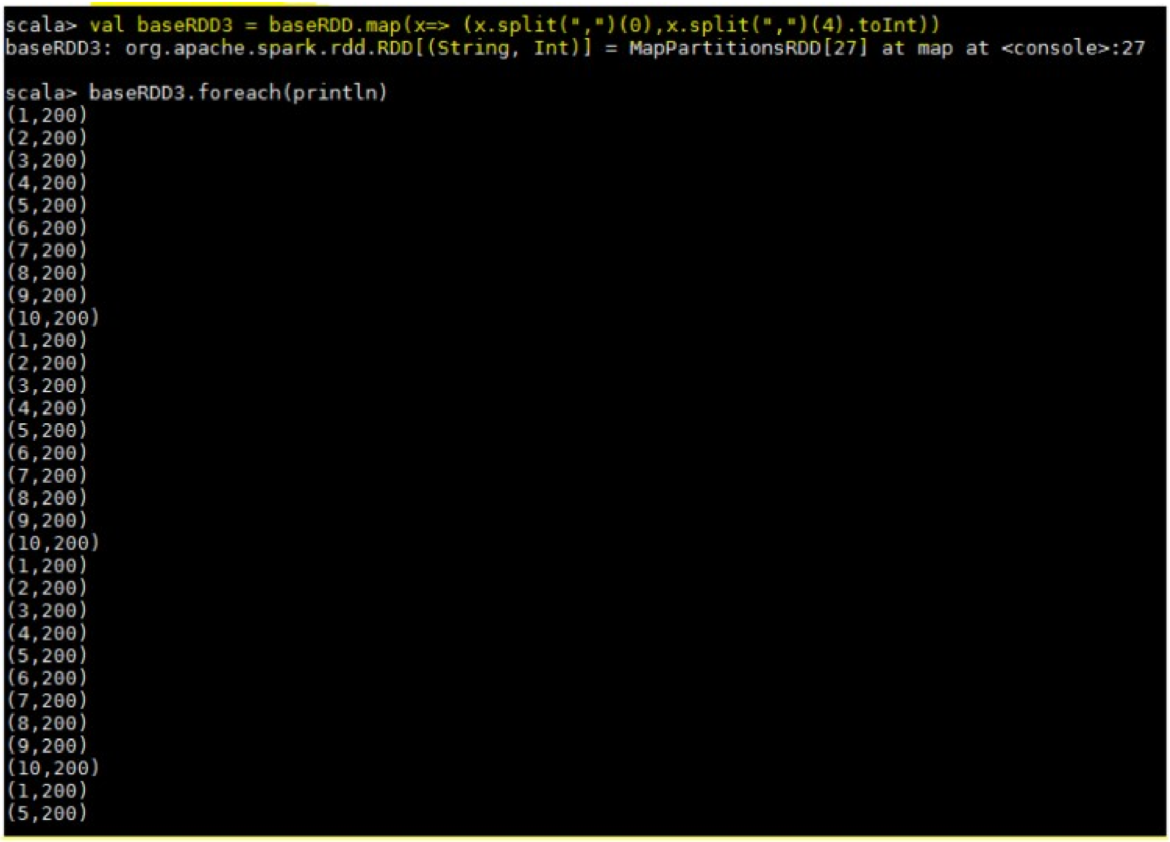
```
scala> val baseRDD2 = baseRDD.map(x => ((x.split(",")(0),x.split(",")(5)),x.split(",")(4).toInt))
baseRDD2: org.apache.spark.rdd.RDD[((String, String), Int)] = MapPartitionsRDD[25] at map at <console>:27

scala> baseRDD2.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)
((1,1993),200)
((5,1994),200)
```



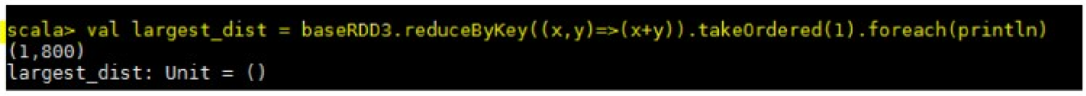
```
scala> val distance_user = baseRDD2.reduceByKey((x,y) => (x + y)).foreach(println)
((3,1992),200)
((3,1993),200)
((5,1991),200)
((6,1991),400)
((10,1993),200)
((5,1992),400)
((8,1991),200)
((8,1990),200)
((1,1993),600)
((5,1994),200)
((2,1993),200)
((2,1991),400)
((4,1990),400)
((10,1992),200)
((3,1991),200)
((1,1990),200)
((10,1990),200)
((6,1993),200)
((9,1992),400)
((8,1992),200)
((7,1990),600)
((9,1991),200)
((4,1991),200)
distance_user: Unit = ()
```

3) Which user has travelled the largest distance till date

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)***

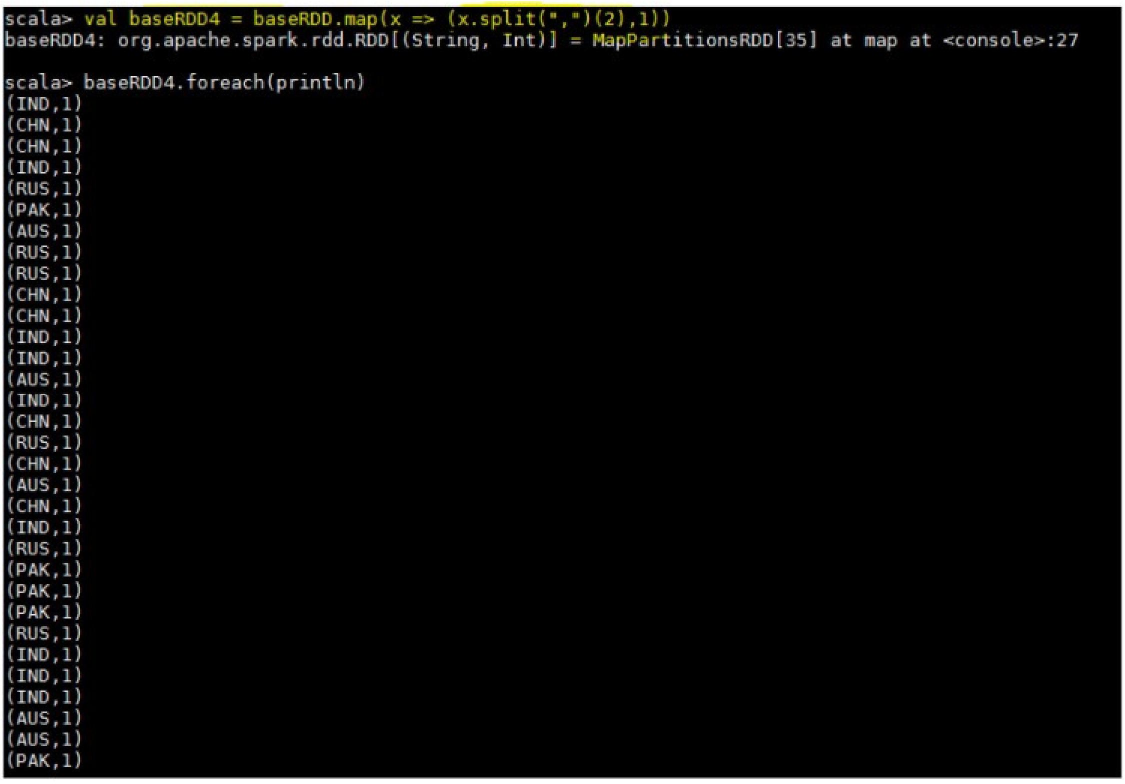

```
scala> val baseRDD3 = baseRDD.map(x=> (x.split(",")(0),x.split(",")(4).toInt))
baseRDD3: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[27] at map at <console>:27

scala> baseRDD3.foreach(println)
(1,200)
(2,200)
(3,200)
(4,200)
(5,200)
(6,200)
(7,200)
(8,200)
(9,200)
(10,200)
(1,200)
(2,200)
(3,200)
(4,200)
(5,200)
(6,200)
(7,200)
(8,200)
(9,200)
(10,200)
(1,200)
(2,200)
(3,200)
(4,200)
(5,200)
(6,200)
(7,200)
(8,200)
(9,200)
(10,200)
(1,200)
(5,200)
```

The required output,

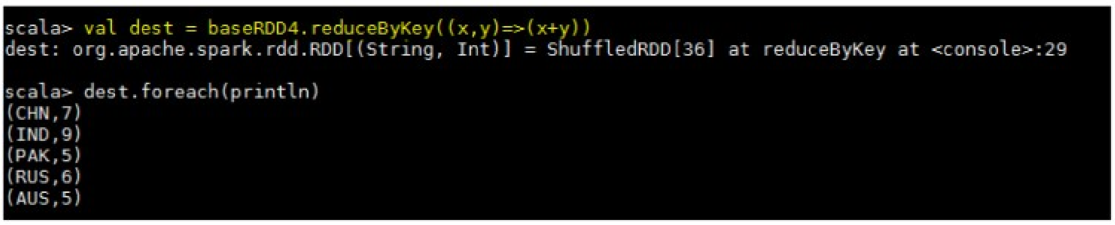

```
scala> val largest_dist = baseRDD3.reduceByKey((x,y)=>(x+y)).takeOrdered(1).foreach(println)
(1,800)
largest_dist: Unit = ()
```

4) What is the most preferred destination for all users.

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))***

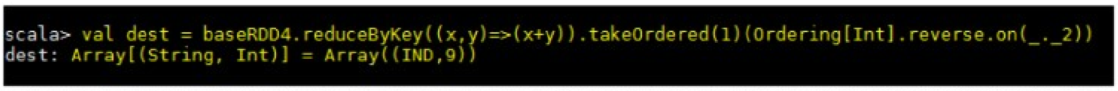


```
scala> val baseRDD4 = baseRDD.map(x => (x.split(",")(2),1))
baseRDD4: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[35] at map at <console>:27

scala> baseRDD4.foreach(println)
(IND,1)
(CHN,1)
(CHN,1)
(IND,1)
(RUS,1)
(PAK,1)
(AUS,1)
(RUS,1)
(RUS,1)
(CHN,1)
(CHN,1)
(IND,1)
(IND,1)
(AUS,1)
(IND,1)
(CHN,1)
(RUS,1)
(CHN,1)
(AUS,1)
(CHN,1)
(IND,1)
(RUS,1)
(PAK,1)
(PAK,1)
(PAK,1)
(RUS,1)
(IND,1)
(IND,1)
(IND,1)
(AUS,1)
(AUS,1)
(PAK,1)
```



```
scala> val dest = baseRDD4.reduceByKey((x,y)=>(x+y))
dest: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[36] at reduceByKey at <console>:29

scala> dest.foreach(println)
(CHN,7)
(IND,9)
(PAK,5)
(RUS,6)
(AUS,5)
```

The required output,



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

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

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

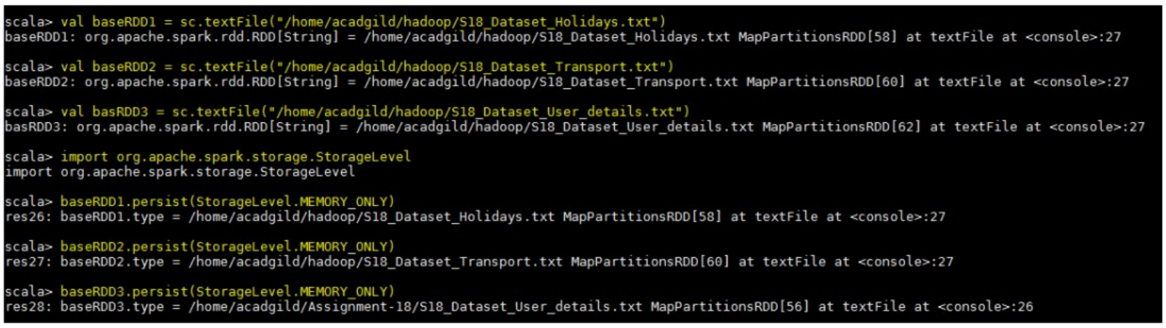
1. Considering age groups of **< 20 , 20-35, 35 >** ,Which age group spends the most Amount of money travelling.
2. What is the amount spent by each age-group, every year in travelling?

Before that, we are loading the dataset into the spark context,

1. ***val baseRDD1 = sc.textFile("/home/acadgild/hadoop/S18_Dataset_Holidays.txt")***
2. ***val baseRDD2 = sc.textFile("/home/acadgild/hadoop/S18_Dataset_Transport.txt")***
3. ***val baseRDD3 = sc.textFile("/home/acadgild/hadoop/S18_Dataset_User_details.txt")***

Importing the singleton object for controlling the storage of an RDD,

4. ***import org.apache.spark.storage.StorageLevel***
5. ***baseRDD1.persist(StorageLevel.MEMORY_ONLY)***
6. ***baseRDD2.persist(StorageLevel.MEMORY_ONLY)***
7. ***baseRDD3.persist(StorageLevel.MEMORY_ONLY)***



```
scala> val baseRDD1 = sc.textFile("/home/acadgild/hadoop/S18_Dataset_Holidays.txt")
baseRDD1: org.apache.spark.rdd.RDD[String] = /home/acadgild/hadoop/S18_Dataset_Holidays.txt MapPartitionsRDD[58] at textFile at <console>:27

scala> val baseRDD2 = sc.textFile("/home/acadgild/hadoop/S18_Dataset_Transport.txt")
baseRDD2: org.apache.spark.rdd.RDD[String] = /home/acadgild/hadoop/S18_Dataset_Transport.txt MapPartitionsRDD[60] at textFile at <console>:27

scala> val basRDD3 = sc.textFile("/home/acadgild/hadoop/S18_Dataset_User_details.txt")
basRDD3: org.apache.spark.rdd.RDD[String] = /home/acadgild/hadoop/S18_Dataset_User_details.txt MapPartitionsRDD[62] at textFile at <console>:27

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

scala> baseRDD1.persist(StorageLevel.MEMORY_ONLY)
res26: baseRDD1.type = /home/acadgild/hadoop/S18_Dataset_Holidays.txt MapPartitionsRDD[58] at textFile at <console>:27

scala> baseRDD2.persist(StorageLevel.MEMORY_ONLY)
res27: baseRDD2.type = /home/acadgild/hadoop/S18_Dataset_Transport.txt MapPartitionsRDD[60] at textFile at <console>:27

scala> baseRDD3.persist(StorageLevel.MEMORY_ONLY)
res28: baseRDD3.type = /home/acadgild/Assignment-18/S18_Dataset_User_details.txt MapPartitionsRDD[56] at textFile at <console>:26
```

val AgeMap = user.map(x=>x._1->

| {

| if(x._3<20)

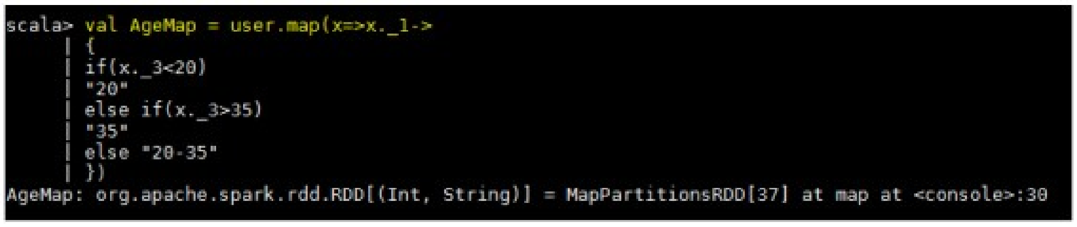
| "20"

| else if(x._3>35)

| "35"

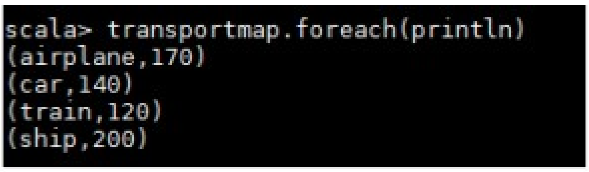
| else "20-35"

| })



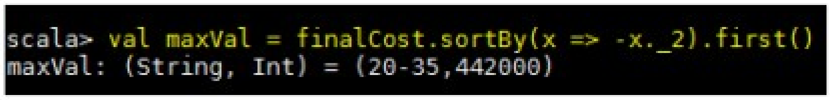
```
scala> val AgeMap = user.map(x=>x._1->
     | {
     | if(x._3<20)
     | "20"
     | else if(x._3>35)
     | "35"
     | else "20-35"
     | })
AgeMap: org.apache.spark.rdd.RDD[(Int, String)] = MapPartitionsRDD[37] at map at <console>:30
```

1. val userMap = holidays.map(x => x._4 -> (x._1,x._5))
2. val transportmap = transport.map(x=> x._1 -> x._2)
3. val joinCost = userMap.join(transportmap)
4. val calCost = joinCost.map(x => x._2._1._1 -> x._2._1._2 * x._2._2)
5. val groupCost = calCost.groupByKey().map(x => x._1 -> x._2.sum)
6. val groupAgeCost = AgeMap.join(groupCost).map(x => x._2._1 -> x._2._2)
7. val finalCost = groupAgeCost.groupByKey().map(x => x._1 -> x._2.sum)
8. val maxVal = finalCost.sortBy(x => -x._2).first()



```
scala> transportmap.foreach(println)
(airplane,170)
(car,140)
(train,120)
(ship,200)
```

Expected output



```
scala> val maxVal = finalCost.sortBy(x => -x._2).first()
maxVal: (String, Int) = (20-35,442000)
```

The codes used,

1. **val UserHolidays = holidays.map(x => x._4 -> (x._1,x._5,x._6))**
2. **val usertransport = transport.map(x=>x._1->x._2)**
3. **val join1 = UserHolidays.join(usertransport)**
4. **val cost_dist_amt = join1.map(x=>x._2._1._1->(x._2._1._3,x._2._1._2*x._2._2))**
5. **val join2 = AgeMap.join(cost_dist_amt).map(x=>(x._2._1,x._2._2._1)->x._2._2._2)**
6. **val ExpEachAgeGroup = join2.groupByKey().map(x=>x._1->x._2.sum)**

The codes used,

1. **val UserHolidays = holidays.map(x => x._4 -> (x._1,x._5,x._6))**
2. **val usertransport = transport.map(x=>x._1->x._2)**
3. **val join1 = UserHolidays.join(usertransport)**
4. **val cost_dist_amt = join1.map(x=>x._2._1._1->(x._2._1._3,x._2._1._2*x._2._2))**
5. **val join2 = AgeMap.join(cost_dist_amt).map(x=>(x._2._1,x._2._2._1)->x._2._2._2)**
6. **val ExpEachAgeGroup = join2.groupByKey().map(x=>x._1->x._2.sum)**