Problem Statement

1. Which route is generating the most revenue per year

Before proceeding we are creating RDDs to read the input files and then we are using Caching because all the files are being read again and again. Below are the screenshots for it-

Reading file1 for Holidays-

```
scala> val baseRDD1 = sc.textFile("/home/acadgild/Assignment-18/S18_Dataset_Holidays.txt")
baseRDD1: org.apache.spark.rdd.RDD[String] = /home/acadgild/Assignment-18/S18_Dataset_Holidays.txt MapPartitionsRDD[1] at textFile at <console>:24

scala>
```

Reading File 2 for Transport

```
scala> val baseRDD2 = sc.textFile("/home/acadgild/Assignment-18/S18_Dataset_Transport.txt")
baseRDD2: org.apache.spark.rdd.RDD[String] = /home/acadgild/Assignment-18/S18_Dataset_Transport.txt MapPartitionsRDD[3] at textFile at <console>:24

scala>
```

Reading File 3 for User-

```
scala> val baseRDD3 = sc.textFile("/home/acadgild/Assignment-18/S18_Dataset_User_details.txt")
baseRDD3: org.apache.spark.rdd.RDD[String] = /home/acadgild/Assignment-18/S18_Dataset_User_details.txt MapPartitionsRDD[5] at textFile at <console>:24

scala>
```

Caching all above created RDDs-

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

scala> baseRDD1.persist(StorageLevel.MEMORY_ONLY)
res0: baseRDD1.type = /home/acadgild/Assignment-18/S18_Dataset_Holidays.txt MapPartitionsRDD[1] at textFile at <console>:24

scala> baseRDD2.persist(StorageLevel.MEMORY_ONLY)
res1: baseRDD2.type = /home/acadgild/Assignment-18/S18_Dataset_Transport.txt MapPartitionsRDD[3] at textFile at <console>:24

scala> baseRDD3.persist(StorageLevel.MEMORY_ONLY)
res2: baseRDD3.type = /home/acadgild/Assignment-18/S18_Dataset_User_details.txt MapPartitionsRDD[5] at textFile at <console>:24

scala>
```

Below is the command used to find the result-

- val travel = baseRDD1.map(x => (x.split(",")(0).toInt,x.split(",")(1),x.split(",")(2),x.split(",")(3),x.split(",")(4).toInt,x.split(",")(5).toInt))
- val transport = baseRDD2.map(x => (x.split(",")(0),x.split(",")(1).toInt))
- val travelmap = travel.map(x=> x._4 -> (x._2,x._5,x._6))
- val transportmap = transport.map(x=> x._1 -> x._2)
- val join1 = travelmap.join(transportmap)
- val routeMap = join1.map(x => (x._2._1._1 -> x._2._1._3) -> (x._2._1._2 * x._2._2))
- val costsum = routeMap.groupByKey().map(x => x._2.sum -> x._1)
- val sortRevenue = costsum.sortByKey(false).first()

Now we will try to understand each command one by one-

Here we are extracting each and every column of Holidays dataset-

```
scala> val travel = baseRDD1.map(x => (x.split(",")(0).toInt,x.split(",")(1),x.split(",")(2),x.split(",")(3),x.split(",")(4).toInt,x.split(",")(5).toInt))
travel: org.apache.spark.rdd.RDD[(Int, String, String, String, Int, Int)] = MapPartitionsRDD[6] at map at <console>:27

scala> travel.foreach(println)
[Stage 0:>                                                          (0 + 0) / 2](1,CHN,IND,airplane,200,1990)
(2,IND,CHN,airplane,200,1991)
(3,IND,CHN,airplane,200,1992)
(4,RUS,IND,airplane,200,1990)
(5,CHN,RUS,airplane,200,1992)
(6,AUS,PAK,airplane,200,1991)
(7,RUS,AUS,airplane,200,1990)
(8,IND,RUS,airplane,200,1991)
(9,CHN,RUS,airplane,200,1992)
(10,AUS,CHN,airplane,200,1993)
(1,AUS,CHN,airplane,200,1993)
(2,CHN,IND,airplane,200,1993)
(3,CHN,IND,airplane,200,1993)
(4,IND,AUS,airplane,200,1991)
(5,AUS,IND,airplane,200,1992)
(6,RUS,CHN,airplane,200,1993)
(7,CHN,RUS,airplane,200,1990)
(8,AUS,CHN,airplane,200,1990)
(9,IND,AUS,airplane,200,1991)
(10,RUS,CHN,airplane,200,1992)
(1,PAK,IND,airplane,200,1993)
(2,IND,RUS,airplane,200,1991)
(3,CHN,PAK,airplane,200,1991)
(4,CHN,PAK,airplane,200,1990)
(5,IND,PAK,airplane,200,1991)
(6,PAK,RUS,airplane,200,1991)
(7,CHN,IND,airplane,200,1990)
(8,RUS,IND,airplane,200,1992)
(9,RUS,IND,airplane,200,1992)
(10,CHN,AUS,airplane,200,1990)
(1,PAK,AUS,airplane,200,1993)
(5,CHN,PAK,airplane,200,1994)
```

Here we are extracting each and every column of Transport dataset-

```
scala> val transport = baseRDD2.map(x => (x.split(",")(0),x.split(",")(1).toInt))
transport: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[8] at map at <console>:27

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

scala>
```

Here we are extracting each and every column of User dataset-

```
scala> val user = baseRDD3.map(x => (x.split(",")(0).toInt,x.split(",")(1),x.split(",")(2).toInt))
user: org.apache.spark.rdd.RDD[(Int, String, Int)] = MapPartitionsRDD[9] at map at <console>:27

scala> user.foreach(println)
(7,james,21)
(8,andrew,55)
(9,thomas,46)
(10,annie,44)
(1,mark,15)
(2,john,16)
(3,luke,17)
(4,lisa,27)
(5,mark,25)
(6,peter,22)
```

We will use above RDDs in further problems also.

For this problem we are mapping the Holidays dataset's 4th column (mode of travel) to source, distance and year of same file. Below is the screenshot for same-

```
scala> val travelmap = travel.map(x=> x._4 -> (x._2,x._5,x._6))
travelmap: org.apache.spark.rdd.RDD[(String, (String, Int, Int))] = MapPartitionsRDD[10] at map at <console>:29

scala> travelmap.foreach(println)
(airplane,(CHN,200,1990))
(airplane,(IND,200,1991))
(airplane,(IND,200,1992))
(airplane,(RUS,200,1990))
(airplane,(CHN,200,1992))
(airplane,(AUS,200,1991))
(airplane,(RUS,200,1990))
(airplane,(IND,200,1991))
(airplane,(CHN,200,1992))
(airplane,(AUS,200,1993))
(airplane,(AUS,200,1993))
(airplane,(CHN,200,1993))
(airplane,(CHN,200,1993))
(airplane,(IND,200,1991))
(airplane,(AUS,200,1992))
(airplane,(RUS,200,1993))
(airplane,(CHN,200,1990))
(airplane,(AUS,200,1990))
(airplane,(IND,200,1991))
(airplane,(RUS,200,1992))
(airplane,(PAK,200,1993))
(airplane,(IND,200,1991))
(airplane,(CHN,200,1991))
(airplane,(CHN,200,1990))
(airplane,(IND,200,1991))
(airplane,(PAK,200,1991))
(airplane,(CHN,200,1990))
(airplane,(RUS,200,1992))
(airplane,(RUS,200,1992))
(airplane,(CHN,200,1990))
(airplane,(PAK,200,1993))
(airplane,(CHN,200,1994))
```

Again we are mapping the transport dataset's first column to its second column(cost)-

```
scala> val transportmap = transport.map(x=> x._1 -> x._2)
transportmap: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[11] at map at <console>:29

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

Now we are joining above RDDs using the column mode of travel which is "airplane" here-

```
scala> val join1 = travelmap.join(transportmap)
join1: org.apache.spark.rdd.RDD[(String, ((String, Int, Int), Int))] = MapPartitionsRDD[14] at join at <console>:37

scala> join1.foreach(println)
[Stage 8:>                                                          (0 + 0) / 2](airplane,((CHN,200,1990),170))
(airplane,((IND,200,1991),170))
(airplane,((IND,200,1992),170))
(airplane,((RUS,200,1990),170))
(airplane,((CHN,200,1992),170))
(airplane,((AUS,200,1991),170))
(airplane,((RUS,200,1990),170))
(airplane,((IND,200,1991),170))
(airplane,((CHN,200,1992),170))
(airplane,((IND,200,1991),170))
(airplane,((CHN,200,1992),170))
(airplane,((AUS,200,1993),170))
(airplane,((AUS,200,1993),170))
(airplane,((CHN,200,1993),170))
(airplane,((CHN,200,1993),170))
(airplane,((IND,200,1991),170))
(airplane,((AUS,200,1992),170))
(airplane,((RUS,200,1993),170))
(airplane,((CHN,200,1990),170))
(airplane,((AUS,200,1990),170))
(airplane,((IND,200,1991),170))
(airplane,((RUS,200,1992),170))
(airplane,((PAK,200,1993),170))
(airplane,((IND,200,1991),170))
(airplane,((CHN,200,1991),170))
(airplane,((CHN,200,1990),170))
(airplane,((IND,200,1991),170))
(airplane,((PAK,200,1991),170))
(airplane,((CHN,200,1990),170))
(airplane,((RUS,200,1992),170))
(airplane,((RUS,200,1992),170))
(airplane,((CHN,200,1990),170))
(airplane,((PAK,200,1993),170))
(airplane,((CHN,200,1994),170))
```

Here we are restructuring above join's result and making a key value pair in which source and year is key and distance multiplied by cost is the value-

```
scala> val routeMap = join1.map(x => (x._2._1._1 -> x._2._1._3) -> (x._2._1._2 * x._2._2))
routeMap: org.apache.spark.rdd.RDD[((String, Int), Int)] = MapPartitionsRDD[16] at map at <console>:39

scala> routeMap.foreach(println)
((CHN,1990),34000)
((IND,1991),34000)
((IND,1992),34000)
((RUS,1990),34000)
((CHN,1992),34000)
((AUS,1991),34000)
((RUS,1990),34000)
((IND,1991),34000)
((CHN,1992),34000)
((AUS,1993),34000)
((AUS,1993),34000)
((CHN,1993),34000)
((CHN,1993),34000)
((IND,1991),34000)
((AUS,1992),34000)
((RUS,1993),34000)
((CHN,1990),34000)
((AUS,1990),34000)
((IND,1991),34000)
((RUS,1992),34000)
((PAK,1993),34000)
((IND,1991),34000)
((CHN,1991),34000)
((CHN,1990),34000)
((IND,1991),34000)
((PAK,1991),34000)
((CHN,1990),34000)
((RUS,1992),34000)
((RUS,1992),34000)
((CHN,1990),34000)
((PAK,1993),34000)
((CHN,1994),34000)
```

Now we are using GroupByKey to group the results based on key and then mapping each revenue with source and year-

```
scala> val costsum = routeMap.groupByKey().map(x => x._2.sum -> x._1)
costsum: org.apache.spark.rdd.RDD[(Int, (String, Int))] = MapPartitionsRDD[18] at map at <console>:41

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

Now we are using sortByKey to find the highest revenue -

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

scala>
```

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

Here also we have used Caching as shown above and will be using same RDDs to read the files. Below is the code used to find the result-

- val travel = baseRDD1.map(x => (x.split(",")(0).toInt,x.split(",")(1),x.split(",")(2),x.split(",")(3),x.split(",")(4).toInt,x.split(",")(5).toInt))
- val userMap = travel.map(x => x._4 -> (x._1,x._5,x._6))
- val transport = baseRDD2.map(x => (x.split(",")(0),x.split(",")(1).toInt))
- val transportmap = transport.map(x=> x._1 -> x._2)
- val amtMap = userMap.join(transportmap)
- val spendMap = amtMap.map(x => (x._2._1._1, x._2._1._3) -> (x._2._1._2 * x._2._2))
- val total = spendMap.groupByKey().map(x => x._1 -> x._2.sum)

Here we are extracting each and every column of Holidays dataset-

```
scala> val travel = baseRDD1.map(x => (x.split(",")(0).toInt,x.split(",")(1),x.split(",")(2),x.split(",")(3),x.split(",")(4).toInt,x.split(",")(5).toInt))
travel: org.apache.spark.rdd.RDD[(Int, String, String, String, Int, Int)] = MapPartitionsRDD[6] at map at <console>:27

scala> travel.foreach(println)
[Stage 0:>                                                          (0 + 0) / 2](1,CHN,IND,airplane,200,1990)
(2,IND,CHN,airplane,200,1991)
(3,IND,CHN,airplane,200,1992)
(4,RUS,IND,airplane,200,1990)
(5,CHN,RUS,airplane,200,1992)
(6,AUS,PAK,airplane,200,1991)
(7,RUS,AUS,airplane,200,1990)
(8,IND,RUS,airplane,200,1991)
(9,CHN,RUS,airplane,200,1992)
(10,AUS,CHN,airplane,200,1993)
(1,AUS,CHN,airplane,200,1993)
(2,CHN,IND,airplane,200,1993)
(3,CHN,IND,airplane,200,1993)
(4,IND,AUS,airplane,200,1991)
(5,AUS,IND,airplane,200,1992)
(6,RUS,CHN,airplane,200,1993)
(7,CHN,RUS,airplane,200,1990)
(8,AUS,CHN,airplane,200,1990)
(9,IND,AUS,airplane,200,1991)
(10,RUS,CHN,airplane,200,1992)
(1,PAK,IND,airplane,200,1993)
(2,IND,RUS,airplane,200,1991)
(3,CHN,PAK,airplane,200,1991)
(4,CHN,PAK,airplane,200,1990)
(5,IND,PAK,airplane,200,1991)
(6,PAK,RUS,airplane,200,1991)
(7,CHN,IND,airplane,200,1990)
(8,RUS,IND,airplane,200,1992)
(9,RUS,IND,airplane,200,1992)
(10,CHN,AUS,airplane,200,1990)
(1,PAK,AUS,airplane,200,1993)
(5,CHN,PAK,airplane,200,1994)
```

Now we are mapping the mode of transport column (key) to User ID, distance and year(values)-

```
scala> val userMap = travel.map(x => x._4 -> (x._1,x._5,x._6))
userMap: org.apache.spark.rdd.RDD[(String, (Int, Int, Int))] = MapPartitionsRDD[22] at map at <console>:29

scala> userMap.foreach(println)
[Stage 28:>                                                         (0 + 0) / 2](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))
(airplane,(1,200,1993))
(airplane,(5,200,1994))
```

Here we are extracting each and every column of Transport dataset-

```
scala> val transport = baseRDD2.map(x => (x.split(",")(0),x.split(",")(1).toInt))
transport: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[8] at map at <console>:27

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

scala>
```

Again we are mapping the transport dataset's first column to its second column(cost)-

```
scala> val transportmap = transport.map(x=> x._1 -> x._2)
transportmap: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[11] at map at <console>:29

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

Now we are joining above RDDs with mode of transport as key-

```
scala> val amtMap = userMap.join(transportmap)
amtMap: org.apache.spark.rdd.RDD[(String, ((Int, Int, Int), Int))] = MapPartitionsRDD[25] at join at <console>:37

scala> amtMap.foreach(println)
[Stage 31:>                                                         (0 + 0) / 2](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))
(airplane,((1,200,1993),170))
(airplane,((5,200,1994),170))
```

Now we are calculating the amount spent by each user per year by multiplying the distance covered with cost as shown below-

```
scala> val spendMap = amtMap.map(x => (x._2._1._1, x._2._1._3) -> (x._2._1._2 * x._2._2))
spendMap: org.apache.spark.rdd.RDD[((Int, Int), Int)] = MapPartitionsRDD[26] at map at <console>:39

scala> spendMap.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)
((10,1990),34000)
((1,1993),34000)
((5,1994),34000)
```

Now we are using groupByKey and adding each travel cost per year per user-

```
scala> val total = spendMap.groupByKey().map(x => x._1 -> x._2.sum)
total: org.apache.spark.rdd.RDD[((Int, Int), Int)] = MapPartitionsRDD[28] at map at <console>:41

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

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

Below is the code used to find the solution -

- val UIDMap = travel.map(x => x._1 -> 1)
- val AgeMap = user.map(x => x._1 ->
  | {
  | if(x._3<20)
  | "20"
  | else if(x._3>35)
  | "35"
  | else "20-35"
  | })
- val joinMap = AgeMap.join(UIDMap)
- val joinMap2 = joinMap.map(x => x._2._1 -> x._2._2)
- val groupKey = joinMap2.groupByKey.map(x => x._1 -> x._2.sum)
- val maxVal = groupKey.sortBy(x => -x._2).first()

Here in order to proceed we are defining age group by using below map function-

```
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[29] at map at <console>:29

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

We have already defined RDD “travel”, here we are mapping the User ID with 1

```
scala> val UIDMap = travel.map(x => x._1 -> 1)
UIDMap: org.apache.spark.rdd.RDD[(Int, Int)] = MapPartitionsRDD[30] at map at <console>:29

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

Now we are joining the RDD AgeMap and UIDMap with User-ID as shown below-

```
scala> val joinMap = AgeMap.join(UIDMap)
joinMap: org.apache.spark.rdd.RDD[(Int, (String, Int))] = MapPartitionsRDD[33] at join at <console>:37

scala> joinMap.foreach(println)
[Stage 43:>                                                         (0 + 0) / 2](1,(20,1))
(1,(20,1))
(1,(20,1))
(1,(20,1))
(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))
(5,(20-35,1))
(5,(20-35,1))
(5,(20-35,1))
(5,(20-35,1))
(4,(20-35,1))
(4,(20-35,1))
(4,(20-35,1))
(6,(20-35,1))
(6,(20-35,1))
(6,(20-35,1))
(8,(35,1))
(8,(35,1))
(8,(35,1))
(10,(35,1))
(10,(35,1))
(10,(35,1))
(2,(20,1))
(2,(20,1))
(2,(20,1))
```

Here we are mapping each age group with column containing 1-

```
scala> val joinMap2 = joinMap.map(x => x._2._1 -> x._2._2)
joinMap2: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[34] at map at <console>:39

scala> joinMap2.foreach(println)
(20-35,1)
(20-35,1)
(20-35,1)
(20-35,1)
(20-35,1)
(20-35,1)
(35,1)
(35,1)
(35,1)
(35,1)
(35,1)
(35,1)
(20,1)
(20,1)
(20,1)
(20,1)
(20,1)
(20,1)
(20,1)
(20,1)
(20,1)
(20,1)
(20-35,1)
(20-35,1)
(20-35,1)
(35,1)
(35,1)
(35,1)
(20-35,1)
(20-35,1)
(20-35,1)
(20-35,1)
```

Here using groupByKey we are calculating sum of every occurrence of age group-

```
scala> val groupKey = joinMap2.groupByKey.map(x => x._1 -> x._2.sum)
groupKey: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[36] at map at <console>:41

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

Finally we are using sortBy function to find the maximum among above RDD which shows that age group 20-35 has travelled the most every year.

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

scala>
```