

# Session 20:

## SPARK SQL 1

### Assignment 1

#### Task 1

- 1) What is the distribution of the total number of air-travelers per year
- 2) What is the total air distance covered by each user per year
- 3) Which user has travelled the largest distance till date
- 4) What is the most preferred destination for all users.
- 5) Which route is generating the most revenue per year
- 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.

Dataset used,

S18\_Dataset\_Holidays.txt,

```
[acadgild@localhost hadoop]$ cat S18_Dataset_Holidays.txt
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
```

The above dataset has the data column wise, userID, Destination, arrival, travel mode, travel distance and the year,

We are loading the dataset into the spark context,

```
val baseRDD = sc.textFile("/home/acadgild/hadoop/S18_Dataset_Holidays.txt")  
import org.apache.spark.storage.StorageLevel
```

We are caching the “baseRDD” using the persist(StorageLevel.MEMORY\_ONLY)

```
baseRDD.persist(StorageLevel.MEMORY_ONLY)
```

```
scala> val baseRDD = sc.textFile("/home/acadgild/hadoop/S18_Dataset_Holidays.txt")  
baseRDD: org.apache.spark.rdd.RDD[String] = /home/acadgild/hadoop/S18_Dataset_Holidays.txt MapPartitionsRDD[22] at textFile at <console>:24  
  
scala> import org.apache.spark.storage.StorageLevel  
import org.apache.spark.storage.StorageLevel  
  
scala> baseRDD.persist(StorageLevel.MEMORY_ONLY)  
res11: baseRDD.type = /home/acadgild/hadoop/S18_Dataset_Holidays.txt MapPartitionsRDD[22] at textFile at <console>:24
```

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

Codes

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

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)
```

We are reducing the number of occurrences using **reduceByKey** and printing the result. Therefore, total no of air travelers per year is,

```
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

Codes

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

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 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)
```

Reducing the number of occurrences using **reduceByKey** and printing the result, therefore the total air distance covered by each user per year is show below in the screenshot,

```
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

Codes

```
val baseRDD3 = baseRDD.map(x=> (x.split(",")(0),x.split(",")(4).toInt))
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 2<sup>nd</sup> 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[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)
```

```
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.

Codes,

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

The output of the each step is shown below,

```
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)
```

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

## 5) Which route is generating the most revenue per year

S18\_Dataset\_User\_details.txt,

```
[acadgild@localhost hadoop]$ cat S18_Dataset_User_details.txt
1,mark,15
2,john,16
3,luke,17
4,lisa,27
5,mark,25
6,peter,22
7,james,21
8,andrew,55
9,thomas,46
[acadgild@localhost hadoop]$
```

S18\_Dataset\_Transport.txt,

```
[acadgild@localhost hadoop]$ cat S18_Dataset_Transport.txt
airplane,170
car,140
train,120
ship,200[acadgild@localhost hadoop]$ ls -l
total 69060
```

Loading the dataset into the spark context,

```
val baseRDD1 = sc.textFile("/home/acadgild/hadoop/S18_Dataset_Holidays.txt")
val baseRDD2 = sc.textFile("/home/acadgild/hadoop/S18_Dataset_Transport.txt")
val baseRDD3 = sc.textFile("/home/acadgild/hadoop/S18_Dataset_User_details.txt")
```

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

```
import org.apache.spark.storage.StorageLevel
baseRDD1.persist(StorageLevel.MEMORY_ONLY)
baseRDD2.persist(StorageLevel.MEMORY_ONLY)
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 baseRDD3 = sc.textFile("/home/acadgild/hadoop/S18_Dataset_User_details.txt")
baseRDD3: 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[62] at textFile at <console>:26
```

The given dataset's have been loaded and creating the tuple RDD columns wise in the spark context,

Loading the dataset's in the name of holidays, transport and user RDD's.

**val holidays =**

**baseRDD1.map(x=>(x.split(",")(0).toInt,x.split(",")(1),x.split(",")(2),x.split(",")(3),x.split(",")(4).toInt,x.split(",")(5).toInt))**

```
scala> val Holidays = baseRDD1.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[63] at map at <console>:30

scala> Holidays.foreach(println)
(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)
```

**val Transport = baseRDD2.map(x=> (x.split(",")(0),x.split(",")(1).toInt))**

```
scala> val Transport = baseRDD2.map(x=> (x.split(",")(0),x.split(",")(1).toInt))
Transport: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[64] at map at <console>:30

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

**val user = baseRDD3.map(x=>(x.split(",")(0).toInt,x.split(",")(1),x.split(",")(2).toInt))**

```
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[8] at map at <console>:27

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

Codes,

**val holidaysmap = holidays.map(x=>x.\_4->(x.\_2,x.\_5,x.\_6))**

**val transportmap = transport.map(x=>x.\_1->x.\_2)**

**val join1 = holidaysmap.join(transportmap)**

**val route = join1.map(x=>(x.\_2.\_1.\_1->x.\_2.\_1.\_3)->(x.\_2.\_1.\_2\*x.\_2.\_2))**

**val revenue = route.groupByKey().map(x=>x.\_2.sum->x.\_1)**

**val routemostrevenue = revenue.sortByKey(false).first()**



**result: routemostrevenue: (Int, (String, Int)) = (204000,(IND,1991))**

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

```
scala> holidaysmap.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))
```

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[11] at map at <console>:29

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

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[14] at join at <console>:37

scala> join1.foreach(println)
(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,((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))
```

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)] = MapPartitionsRDD[15] at map at <console>:39

scala> route.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)
```

Using groupByKey function, 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[24] at map at <console>:41

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))
```

Output

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

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

Codes,

```
val userMap = holidays.map(x => x._4 -> (x._1,x._5,x._6))
val amount = userMap.join(transportmap)
val spend = amount.map(x => (x._2._1._1, x._2._1._3) -> (x._2._1._2 * x._2._2))
val total = spend.groupByKey().map(x => x._1 -> x._2.sum)
```

Creating a tuple rdd from a baseRDD **holidays** making the travel mode as Key and the userID, distance & year as values.

```
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))
(airplane,(1,200,1993))
(airplane,(5,200,1994))
```

Joining the created tuple RDD **userMap** with the already created tuple RDD **transportMap** using the join function and calculating the expenditure for each user by multiplying the distance and the amount spent for the travel mode airplane

```
scala> amount.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))
(airplane,((1,200,1993),170))
(airplane,((5,200,1994),170))
```

```
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)
((10,1990),34000)
((1,1993),34000)
((5,1994),34000)
```

Summing the total value for each user yearly wise, please see the expected result in the below screen shot.

```
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)
((8,1991),34000)
((8,1992),34000)
```

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

Code,

```
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
```

Actual codes,

```
val UserID = holidays.map(x => x._1 -> 1)
val joinMap1 = AgeMap.join(UserID)
val joinMap2 = joinMap1.map(x => x._2._1 -> x._2._2)
val groupKey = joinMap2.groupByKey.map(x => x._1 -> x._2.sum)
val mostGroup = groupKey.sortBy(x => -x._2).first()
```

Mapping the user ID from the RDD **holidays** with the numerical 1 and joining the 2 RDD's **UserID** and the **AgeMap**, So we are getting the below tuple RDD

```
scala> val UserID = holidays.map(x => x._1 -> 1)
UserID: org.apache.spark.rdd.RDD[(Int, Int)] = MapPartitionsRDD[38] at map at <console>:30

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

```
scala> val joinMap1 = AgeMap.join(UserID)
joinMap1: org.apache.spark.rdd.RDD[(Int, (String, Int))] = MapPartitionsRDD[41] at join at <console>:38

scala> joinMap1.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))
(2,(20,1))
(2,(20,1))
```

Eliminating the user ID in this step

```
scala> val joinMap2 = joinMap1.map(x => x._2._1 -> x._2._2)
joinMap2: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[42] at map at <console>:40

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

Sum the total value by grouping the Age Group,

```
scala> val groupKey = joinMap2.groupByKey.map(x => x._1 -> x._2.sum)
groupKey: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[44] at map at <console>:42

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

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