Assignment 18.2

Downloaded the 3 datasets **S18_DatasetHolidays.txt**, **S18_Dataset_Transport.txt** and **S18_Dataset_user_details.txt**

And loaded the datasets into the location /home/acadgild/sumona

```
scala> val baseRDD1 = sc.textFile("/home/acadgild/sumona/S18_Dataset_Holidays.txt")
baseRDD1: org.apache.spark.rdd.RDD[String] = /home/acadgild/sumona/S18_Dataset_Holidays.txt MapPartitionsRDD[27] at textFile at <console>:25

scala> val baseRDD2 = sc.textFile("/home/acadgild/sumona/S18_Dataset_Transport.txt")
baseRDD2: org.apache.spark.rdd.RDD[String] = /home/acadgild/sumona/S18_Dataset_Transport.txt MapPartitionsRDD[29] at textFile at <console>:25

scala> val baseRDD3 = sc.textFile("/home/acadgild/sumona/S18_Dataset_User_details.txt")
baseRDD3: org.apache.spark.rdd.RDD[String] = /home/acadgild/sumona/S18_Dataset_User_details.txt MapPartitionsRDD[31] at textFile at <console>:25

scala>
```

Then we perform a map of the above RDD's

```
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[32] at map at <console>:28

scala> val transport = baseRDD2.map(x => (x.split(",")(0),x.split(",")(1).toInt))
transport: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[33] at map at <console>:28

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[34] at map at <console>:28

scala>
```

1. Which route is generating the most revenue per year

First we will select the columns from the above RDD's and then perform a join.

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)

```
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[34] at map at <console>:28

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[35] at map at <console>:30

scala> travelmap.collect().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))
```

```
scala> val transportmap = transport.map(x=> x._1 -> x._2)
transportmap: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[36] at map at <console>:30

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

scala>
```

```
scala> val join1 = travelmap.join(transportmap)
join1: org.apache.spark.rdd.RDD[(String, ((String, Int, Int), Int))] = MapPartitionsRDD[39] at join at <console>:38

scala> join1.collect().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))

scala>
```

Now we will perform the following steps to get the route that generates more revenue per year

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

```
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[40] at map at <console>:40

scala> routeMap.collect().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)

scala>
```

val costsum = routeMap.groupByKey().map(x => x._2.sum -> x._1)

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

scala> costsum.collect().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))

scala>
```

val sortRevenue = costsum.sortByKey(false).first()

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

scala>
```

As you can see through the output, IND generates more revenue per year.

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

To calculate the above query, lets us first create a RDD userMap from the travel RDD and then join it with transportMap RDD

val userMap = travel.map(x => x._4 -> (x._1,x._5,x._6))

val amtMap = userMap.join(transportmap)

```
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[44] at map at <console>:30

scala> userMap.collect().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))

scala>
```

```
scala> val amtMap = userMap.join(transportmap)
amtMap: org.apache.spark.rdd.RDD[(String, ((Int, Int, Int), Int))] = MapPartitionsRDD[47] at join at <console>:38

scala> amtMap.collect().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>
```

After the join, we will perform another map on the joined RDD and select the desired columns

val spendMap = amtMap.map(x => (x._2._1._1, x._2._1._3) -> (x._2._1._2 * x._2._2))

Post which, using the following command we will get the total amount spent by user's in air-travel per year

val total = spendMap.groupByKey().map(x => x._1 -> x._2.sum)

```
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[48] at map at <console>:40

scala> spendMap.collect
res21: Array[((Int, Int), Int)] = Array(((1,1990),34000), ((2,1991),34000), ((3,1992),34000), ((4,1990),34000), ((5,1992),34000), ((6,1991),34000), ((7,1990),340
00), ((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),3
4000), ((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))

scala>
```

```
scala> val total = spendMap.groupByKey().map(x => x._1 -> x._2.sum)
total: org.apache.spark.rdd.RDD[((Int, Int), Int)] = MapPartitionsRDD[50] at map at <console>:42

scala> total.collect().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)

scala>
```

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

First we shall perform a map of the RDD's and select the columns and then perform a join

```
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)
```

```
scala> val UIDMap = travel.map(x => x._1 -> 1)
UIDMap: org.apache.spark.rdd.RDD[(Int, Int)] = MapPartitionsRDD[52] at map at <console>:30

scala> UIDMap.collect().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)
```

```
scala> val AgeMap = user.map(x => x._1 ->
     | {
     | if(x._3<20)
     | "20"
     | else if(x._3>35)
     | "35"
     |  else "20-35"
     | })
```

```
scala> val joinMap = AgeMap.join(UIDMap)
joinMap: org.apache.spark.rdd.RDD[(Int, (String, Int))] = MapPartitionsRDD[63] at join at <console>:39

scala> joinMap.collect().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))

scala>
```

Post these, the following commands will help us get desired output

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()

```
scala> val joinMap2 = joinMap.map(x => x._2._1 -> x._2._2)
joinMap2: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[64] at map at <console>:41

scala> val groupKey = joinMap2.groupByKey.map(x => x._1 -> x._2.sum)
groupKey: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[66] at map at <console>:43

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

scala>
```

As you see in the output the age 13 and 20-35 travels the most.