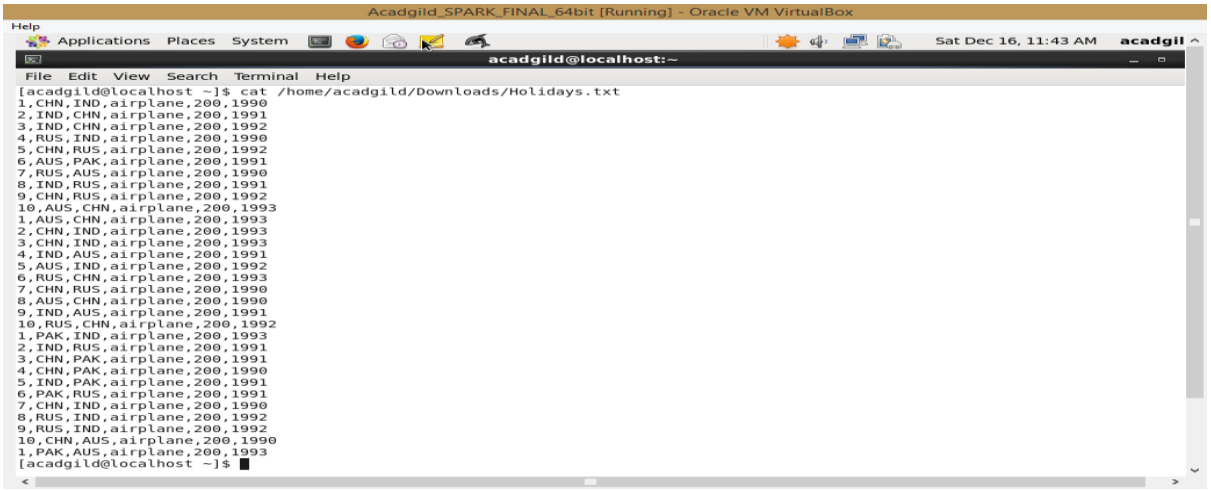
Problem Statement

- Which route is generating the most revenue per year
- What is the total amount spent by every user on air-travel per year
- Considering age groups of < 20 , 20-35, 35 > ,Which age group is travelling the most every year.

Dataset

https://drive.google.com/drive/folders/0B_P3pWagdIrrVThBaUdVSUtzbms

Dataset-Holidays:
The dataset is of holiday details of travelers with columns: **user_id**, **source**, **destination**, **travel_mode**, **distance**, **year_of_travel**:



Dataset-Transport:
The dataset is of transport details with columns: **travel_mode**, **cost_per_unit**:

```
[acadgild@localhost ~]$ cat /home/acadgild/Downloads/Transport.txt
airplane,170
car,140
train,120
ship,200[acadgild@localhost ~]$
```

Dataset-User_details:
The dataset is of user details of travelers with columns: **user_id**, **name,age**:

```
ship,200[acadgild@localhost ~]$ cat /home/acadgild/Downloads/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
10,annie,44[acadgild@localhost ~]$
```

Intialization Spark-Shell:

```
Acadgild_SPARK_FINAL_64bit [Running] - Oracle VM VirtualBox
Help
Applications  Places  System                                   Sat Dec 16, 12:49 PM   acadgil
acadgild@localhost:~
File  Edit  View  Search  Terminal  Help
[acadgild@localhost ~]$ spark-shell
log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.lib.MutableMetricsFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Using Spark's repl log4j profile: org/apache/spark/log4j-defaults-repl.properties
To adjust logging level use sc.setLogLevel("INFO")
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Scala version 2.10.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_65)
Type in expressions to have them evaluated.
Type :help for more information.
17/12/16 12:35:35 WARN Utils: Your hostname, localhost.localdomain resolves to a loopback address: 127.0.0.1; using 10.0.2.15
 instead (on interface eth6)
17/12/16 12:35:35 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Spark context available as sc.
17/12/16 12:35:53 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
17/12/16 12:35:56 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
17/12/16 12:36:14 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.verification is not ena
bled so recording the schema version 1.2.0
17/12/16 12:36:15 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException
17/12/16 12:36:25 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
17/12/16 12:36:26 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
SQL context available as sqlContext.
```

- Creating tupleRdd Holidays from dataset Holidays.txt

```
scala> var Holidays = sc.textFile("/home/acadgild/Downloads/Holidays.txt").map(x=>
     | {
     | val row = x.split(",")
     | (row.apply(0).toInt,row.apply(1),row.apply(2),row.apply(3),row.apply(4).toInt,row.apply(5).toInt)
     | })
Holidays: org.apache.spark.rdd.RDD[(Int, String, String, String, Int, Int)] = MapPartitionsRDD[2] at map at <console>:27

scala>
```

- Displaying all data tupleRdd Holidays

```
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)

scala>
acadgild@localhost:~
```

- Creating tupleRdd Transport from dataset Transport.txt

```
scala> var Transport = sc.textFile("/home/acadgild/Downloads/Transport.txt").map(x =>
     | {
     | val row = x.split(",")
     | (row.apply(0),row.apply(1).toInt)})
Transport: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[31] at map at <console>:27

scala>
```

- Displaying all data tupleRdd Transport

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

scala>
```

- Creating tupleRdd Users from dataset User_details.txt

```
scala> var Users = sc.textFile("/home/acadgild/Downloads/User_details.txt").map(x =>
     | {
     | val row = x.split(",")
     | (row.apply(0),row.apply(1),row.apply(2).toInt)})
Users: org.apache.spark.rdd.RDD[(String, String, Int)] = MapPartitionsRDD[34] at map at <console>:27

scala>
```

- Displaying all data tupleRdd Users

```
scala> Users.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)

scala>
```

- **Which route is generating the most revenue per year**

Using RDD Holidays and Transport created above

Map the Holidays to only the 4th column with the 2nd and 3rd column (travel_mode, (source, destination)). This will be RDD modeRoute. Then, join modeRoute with Transport map the result such that 1st column (source, destination and travel_mode) from Holidays and the cost_per_unit from transport .Then, group the data by the key. Here the key is the (source, destination and travel_mode) as the tuple is ((source, destination and travel_mode), cost_per_unit). Then, map the data such that (source, destination and mode) and sum of the cost for every row that contains the key. Then sort the data by cost descending and get the 1st record and display the result. (mostRevenue)

Code:

Finding route generating the most revenue per year

```
scala> val modeRoute = Holidays.map(x => (x._4,(x._2,x._3)))
modeRoute: org.apache.spark.rdd.RDD[(String, (String, String))] = MapPartitionsRDD[35] at map at <console>:29

scala>

scala> val route = modeRoute.join(Transport).map(x => (( x._2._1 -> x._1),x._2._2))
route: org.apache.spark.rdd.RDD[(((String, String), String), Int)] = MapPartitionsRDD[39] at map at <console>:33

scala>
```

```
scala> route.foreach(println)
(((CHN,IND),airplane),170)
(((IND,CHN),airplane),170)
(((IND,CHN),airplane),170)
(((RUS,IND),airplane),170)
(((CHN,RUS),airplane),170)
(((AUS,PAK),airplane),170)
(((RUS,AUS),airplane),170)
(((IND,RUS),airplane),170)
(((CHN,RUS),airplane),170)
(((AUS,CHN),airplane),170)
(((AUS,CHN),airplane),170)
(((CHN,IND),airplane),170)
(((CHN,IND),airplane),170)
(((IND,AUS),airplane),170)
(((AUS,IND),airplane),170)
(((RUS,CHN),airplane),170)
(((CHN,RUS),airplane),170)
(((AUS,CHN),airplane),170)
(((IND,AUS),airplane),170)
(((RUS,CHN),airplane),170)
(((PAK,IND),airplane),170)
(((IND,RUS),airplane),170)
(((CHN,PAK),airplane),170)
(((CHN,PAK),airplane),170)
(((IND,PAK),airplane),170)
(((PAK,RUS),airplane),170)
(((CHN,IND),airplane),170)
(((RUS,IND),airplane),170)
(((RUS,IND),airplane),170)
(((CHN,AUS),airplane),170)
(((PAK,AUS),airplane),170)
(((CHN,PAK),airplane),170)

scala>
```

Output:

Displaying route generating the most revenue per year

```
scala> val Most_Revenue = route.groupByKey().map(x => x._1 -> x._2.sum).sortBy(x => -x._2).first()
Most_Revenue: (((String, String), String), Int) = (((CHN,IND),airplane),680)

scala>
```

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

Using RDD Holidays and Transport created above

Map Holidays by grouping the data according to the 4th, 1st and 6th column (travel_mode, (user_id and year)). This is userYear. Then, join userYear with Transport and map the result such that 1st column (user_id and year) from Holidays and the cost_per_unit from Transport. This is userYearExpense. Then, group the data in userYearExpense by the key. Here the key is the (user_id, year) as the tuple is (user_id, year, cost_per_unit) Then, map the data such that the sum of the cost for every row that contains the (user_id, year) (distinct). For better readability, use sortByKey() that sorts the data according to the key (user_id, year) and display the result. (userTotExpense)

Code:

Finding total amount spent by every user on air-travel per year

```
scala> val Most_Revenue = route.groupByKey().map(x => x._1 -> x._2.sum).sortBy(x => -x._2).first()
Most_Revenue: (((String, String), String), Int) = (((CHN,IND),airplane),680)

scala>

scala> val userYear = Holidays.map(x=> (x._4,(x._1 -> x._6)))
userYear: org.apache.spark.rdd.RDD[(String, (Int, Int))] = MapPartitionsRDD[45] at map at <console>:29

scala>

scala> val UserYearExpense = userYear.join(Tranport).map(x => (x._2._1,x._2._2))
<console>:31: error: not found: value Tranport
         val UserYearExpense = userYear.join(Tranport).map(x => (x._2._1,x._2._2))
                                                ^

scala> val UserYearExpense = userYear.join(Transport).map(x => (x._2._1,x._2._2))
UserYearExpense: org.apache.spark.rdd.RDD[((Int, Int), Int)] = MapPartitionsRDD[49] at map at <console>:33

scala>

scala> val user_Total_Spent = UserYearExpense.groupByKey().map(x => (x._1,x._2.sum)).sortByKey()
user_Total_Spent: org.apache.spark.rdd.RDD[((Int, Int), Int)] = ShuffledRDD[52] at sortByKey at <console>:35
```

Output:

Displaying total amount spent by every user on air-travel per year

```
scala> user_Total_Spent.foreach(println)
((1,1990),170)
((1,1993),510)
((2,1991),340)
((2,1993),170)
((3,1991),170)
((3,1992),170)
((3,1993),170)
((4,1990),340)
((4,1991),170)
((5,1991),170)
((5,1992),340)
((5,1994),170)
((6,1991),340)
((6,1993),170)
((7,1990),510)
((8,1990),170)
((8,1991),170)
((8,1992),170)
((9,1991),170)
((9,1992),340)
((10,1990),170)
((10,1992),170)
((10,1993),170)

scala>
```

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

Using RDD Holidays and Users created above,
created an Object named AGTravellingMost that holds the implementation:
Map the data in Users by grouping the data according to the 1st column(user_id) and 2nd column formed from checking the condition in the problem statement:
If user age < 20 column data is "<20"
Else if user age > 35 column data is ">35"
Else column data is "20-35"
Now, filter Holidays so that we get the (year, 1) for every user_id and join this to the above
This is trans_Age. Then, map the data in trans_Age by new column to year and 1 for every row.
This data is then grouped by key. Here the key is the (new column, year) as the tuple is (new column, year, 1) .Then, map the data such that we get the sum of the 1's for every row that contains the (new column, year). Now, for every year, create a RDD and filter it by the year and sort it descending by 3rd column (sum of 1's) and get the 1st record and created a list for the results of each year named Most_Travelled_Age_Group_Every_year and display this as the result.

Code:

Finding age group travelled the most every year.

```
scala> object AGTravellingMost
defined module AGTravellingMost

scala> {
     |      val filterAge = Users.map(x => x._1 -> {
     |      if(x._3 < 20)
     |      "<20"
     |      else if (x._3 > 35)
     |      ">35"
     |      else "20-35"
     |      }).join(Holidays.map(x => x._1 -> (x._6,1)))
     |      val trans_Age = filterAge.map(x => (x._2._1 -> x._2._2._1,x._2._2._2)).groupByKey.map(x => (x._1._2,x._1._1,x._2.
sum))
     |      val Y1990 = trans_Age.filter(x => x._1 == 1990).sortBy(x => -x._3).first()
     |      val Y1991 = trans_Age.filter(x => x._1 == 1991).sortBy(x => -x._3).first()
     |      val Y1992 = trans_Age.filter(x => x._1 == 1992).sortBy(x => -x._3).first()
     |      val Y1993 = trans_Age.filter(x => x._1 == 1993).sortBy(x => -x._3).first()
     |      val Y1994 = trans_Age.filter(x => x._1 == 1994).sortBy(x => -x._3).first()
     |      val Most_Travelled_Age_Group_Every_Year = List(Y1990,Y1991,Y1992,Y1993,Y1994)
     |      Most_Travelled_Age_Group_Every_Year.foreach(println)
     | }
```

Output:

Displaying age group travelled the most every year.

```
scala> AGTravellingMost
(1990,20-35,5)
(1991,20-35,4)
(1992,>35,4)
(1993,<20,5)
(1994,20-35,1)
```