SPARK ASSIGNMENT 18.1

S18_Dataset_User_details.txt -> userid,name,age

S18_Dataset_Transport.txt -> travel_mode, cost_per_unit

S18_Dataset_Holidays.txt -> userid,source, destination,travel_mode, distance,year_of_travel

Problem Statement

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

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

18.1.3) Which user has travelled the largest distance till date

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

==========

Input Commands:

//putting the input files to HDFS

[acadgild@localhost]$cd Downloads

[acadgild@localhost Downloads]$ hadoop fs -put S18_Dataset_User_details.txt /user/acadgild/spark/

[acadgild@localhost Downloads]$ hadoop fs -put S18_Dataset_Transport.txt /user/acadgild/spark/

[acadgild@localhost Downloads]$ hadoop fs -put S18_Dataset_Holidays.txt /user/acadgild/spark/

//Initiating the spark-shell to run spark

[acadgild@localhost Downloads]$ spark-shell

```
//importing the spark packages

import org.apache.spark.sql.Row;

import
org.apache.spark.sql.types.{StructType,StructField,StringType,NumericTyp
e,IntegerType};

import org.apache.spark.sql._

import sqlContext.implicits._

//Loading the input files to respective RDDs

val userdetailsRDD =
sc.textFile("hdfs://localhost:9000//user/acadgild/spark/S18_Dataset_User_d
etails.txt")

val transportRDD =
sc.textFile("hdfs://localhost:9000//user/acadgild/spark/S18_Dataset_Transp
ort.txt")

val holidaysRDD =
sc.textFile("hdfs://localhost:9000//user/acadgild/spark/S18_Dataset_Holiday
s.txt")

//Defining the schemas for each of the 3 files mentioned above respectively

val schemaStringu = "userid:integer,name:string,age:integer"

val schemaStringt = "travel_mode:string,cost_per_unit:integer"

val schemaStringh =
"userid:integer,source:string,destination:string,travel_mode:string,distanc
e:integer,year_of_travel:integer"

//Defining the Structtype and StructField for each DB2 Schema

val schemau = StructType(schemaStringu.split(",").map(fieldInfo =>
StructField(fieldInfo.split(":")(0),

 if (fieldInfo.split(":")(1).equals("integer")) IntegerType else
StringType,true)))
```

```
val schemat = StructType(schemaStringt.split(",").map(fieldInfo =>
StructField(fieldInfo.split(":")(0), if (fieldInfo.split(":")(1).equals("string"))
StringType else IntegerType, true)))

val schemah = StructType(schemaStringh.split(",").map(fieldInfo =>
StructField(fieldInfo.split(":")(0), if (fieldInfo.split(":")(1).equals("string"))
StringType else IntegerType, true)))

//Mapping the data present in TEXT files at HDFS

val RDDu = userdetailsRDD.map(_.split(",")).map(r => Row(r(0).toInt, r(1),
r(2).toInt ))

val RDDt = transportRDD.map(_.split(",")).map(r => Row(r(0), r(1).toInt ))

val RDDh = holidaysRDD.map(_.split(",")).map(r => Row(r(0).toInt, r(1) ,
r(2), r(3) , r(4).toInt, r(5).toInt ))

//Defining the SQLCONTEXT object with the help of Spark Context object

val sqlContext = new org.apache.spark.sql.SQLContext(sc);

//Creating the Dataframe with the help of schema and data in text files

val uDF =sqlContext.createDataFrame(RDDu, schemau)

val tDF =sqlContext.createDataFrame(RDDt, schemat)

val hDF =sqlContext.createDataFrame(RDDh, schemah)

//Defining the temporary tables  with the newly created dataframes

uDF.registerTempTable("userdetails")

tDF.registerTempTable("transport")

hDF.registerTempTable("holiday")
```

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

```
sqlContext.sql("SELECT year_of_travel,COUNT(userid) FROM holiday
GROUP BY year_of_travel").show()
```

Output:

```
scala> sqlContext.sql("SELECT year_of_travel,COUNT(userid) FROM holiday GROUP BY year_of_travel").sh
ow()
+--------------+---+
|year_of_travel|_c1|
+--------------+---+
|          1990|  8|
|          1991|  9|
|          1992|  7|
|          1993|  7|
|          1994|  1|
+--------------+---+


scala>
```

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

sqlContext.sql("SELECT userid, year_of_travel,SUM(distance) FROM holiday GROUP BY userid,year_of_travel ORDER BY userid,year_of_travel").show()

Output:

```
scala> sqlContext.sql("SELECT userid, year_of_travel,SUM(distance) FROM holiday GROUP BY userid,year
_of_travel ORDER BY userid,year_of_travel").show()
+------+--------------+---+
|userid|year_of_travel|_c2|
+------+--------------+---+
|     1|          1990|200|
|     1|          1993|600|
|     2|          1991|400|
|     2|          1993|200|
|     3|          1991|200|
|     3|          1992|200|
|     3|          1993|200|
|     4|          1990|400|
|     4|          1991|200|
|     5|          1991|200|
|     5|          1992|400|
|     5|          1994|200|
|     6|          1991|400|
|     6|          1993|200|
|     7|          1990|600|
|     8|          1990|200|
|     8|          1991|200|
|     8|          1992|200|
|     9|          1991|200|
|     9|          1992|400|
+------+--------------+---+
only showing top 20 rows
```

18.1.3) Which user has travelled the largest distance till date

val resultdf = sqlContext.sql("SELECT h.userid, u.name , SUM(h.distance) FROM holiday h ,userdetails u WHERE h.userid = u.userid GROUP BY h.userid ,u.name ORDER BY SUM(h.distance) DESC ").take(2)

resultdf.foreach(println)

Output:

```
scala> val resultdf = sqlContext.sql("SELECT h.userid, u.name , SUM(h.distance) FROM holiday h ,user
details u WHERE h.userid = u.userid GROUP BY h.userid ,u.name ORDER BY SUM(h.distance) DESC ").take(
2)
resultdf: Array[org.apache.spark.sql.Row] = Array([1,mark,800], [5,mark,800])

scala> resultdf.foreach(println)
[1,mark,800]
[5,mark,800]

scala>
```

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

val resdf = sqlContext.sql("select destination from holiday group by destination order by count(*) desc").take(1)

resdf.foreach(println)

Output:

```
scala> val resdf = sqlContext.sql("select destination from holiday group by destination order by cou
nt(*) desc").take(1)
resdf: Array[org.apache.spark.sql.Row] = Array([IND])

scala> resdf.foreach(println)
[IND]

scala>
```