

Session 20:  
SPARK SQL 1  
Assignment 1

Task 1

- 1) What is the distribution of the total number of air-travelers per year  
Below is the entire code given:

```
package core

import org.apache.spark.sql.SparkSession

object SparkObject2DF {
  //Create a case class globally to be used inside the main method
  //Inferring the Schema Using Reflection. Automatically converting an RDD containing case
  classes to a DataFrame.
  // The case class defines the schema of the table. The names of the arguments to the case
  class are read using reflection
  // and become the names of the columns.

  case class User(id:Long,name:String,age:Int)
  case class Transport(transport_mode:String,cost_per_unit:Long)
  // Main method - The execution entry point for the program
  case class
  Holidays(id:Int,source:String,destination:String,transport_mode:String,distance:Int,year:In
  t)

  def main(args: Array[String]): Unit = {
    val spark = SparkSession
      .builder()
      .master("local")
      .appName("Spark file2DF basic example")
      .config("spark.some.config.option", "some-value")
      .getOrCreate()
    //Set the log level as warning
    spark.sparkContext.setLogLevel("WARN")
    //// For implicit conversions like converting RDDs and sequences to DataFrames

    // Reading a file for the User schema created above & splitting on comma character (,)
    // trim is used to eliminate leading.trailing whitespace

    import spark.implicits._
    val user_schema_DF = spark.sparkContext

    .textFile("C:/Users/myipc/Desktop/User_details_schema.txt").map(_._split(",")).map(x=>Us
    er(x(0).trim.toInt,x(1),x(2).trim.toInt)).toDF
```

```

// Reading a file for the Transport schema created above & splitting on comma
character (,)
// trim is used to eliminate leading.trailing whitespace
val transport_DF =spark.sparkContext

.textFile("C:/Users/myipc/Desktop/Dataset_Transport_Schema.txt").map(_split(",")).map
(x=>Transport(x(0),x(1).trim.toLong)).toDF
// Reading a file for the Holidays schema created above & splitting on comma character
(,)
// trim is used to eliminate leading.trailing whitespace
val holidays_DF =spark.sparkContext

.textFile("C:/Users/myipc/Desktop/Dataset_Holidays.txt").map(_split(",")).map(x=>Holid
ys(x(0).trim.toInt,x(1),x(2),x(3),x(4).trim.toInt,x(5).trim.toInt)).toDF

println("User Data Frame:")
user_schema_DF.show()
println("Transport Data Frame:")
transport_DF.show()
println("Holidays Data Frame:")
holidays_DF.show()
//Converting each of the above created schemas into an SQL view
user_schema_DF.createOrReplaceTempView("user")
transport_DF.createOrReplaceTempView("transport")
holidays_DF.createOrReplaceTempView("holidays")
// To find distribution of Air travellers/year we need to find count of id or year for each
using a group by method.
val x=spark.sql("select year, count(year) as num_of_travelers from holidays group by
year")
val y =spark.sql("select year, count(id) as num_of_travelers from holidays group by
year")
x.show
y.show
//selecting each id for each user, year for grouping each year & adding total distance to
find the total air by selecting
//transport_mode as 'airplane' only.
val x2=spark.sql("select id,year,sum(distance) as total_distance from holidays where
transport_mode='airplane' group by id, year order by id ,year ASC")
x2.show
//Selecting id of all the users & adding up the total distance from holiday view
val a3_top = spark.sql("select id, sum(distance) as largest_distance from holidays group
by id order by largest_distance desc")
//Temp view created for saving the result of the query above
a3_top.createOrReplaceTempView("a3_top")
a3_top.show
//selecting only those id's from from the view (a3_top) created above to select the user
id's who have travelled max distance till date

```

```
val large_dist=spark.sql("select id,largest_distance as Largest_Dist_Travelled from  
a3_top where largest_distance = (select Max(largest_distance)as total_distance_covered  
from a3_top)")
```

```
large_dist.show  
//Selecting destination & count of destination & providing an alias Dest_Visited from  
holiday view & creating an Alias  
val a4_top_dest = spark.sql("select destination, count(destination) as Dest_Visited from  
holidays group by destination")  
////Temp view created for saving the result of the query above  
a4_top_dest.createOrReplaceTempView("a4_top_dest")  
a4_top_dest.show  
//Selecting the maximum of all the destination grouped above in the view.  
val a5 = spark.sql("select destination as `Most Visited Destination` from a4_top_dest  
where Dest_Visited=(select Max(Dest_Visited) as most_visited_dest from a4_top_dest)")
```

```
a5.show  
// Joining two views holiday & transport to fetch cost_per_unit from transport & all  
rows from holiday.  
// Joining two views holiday & transport to fetch cost_per_unit from transport & all  
rows from holiday.  
val holi_trans_mode = spark.sql("select h.*, t.cost_per_unit from holidays h join  
transport t on h.transport_mode = t.transport_mode")  
////Temp view created for saving the result of the query above  
holi_trans_mode.createOrReplaceTempView("holi_trans_mode")  
holi_trans_mode.show  
//From the holi_trans view above selecting the route & mutiplying the count of  
trnsport_mode with it's respective cost/unit  
//to fetch the revenue generated by a particular route each year.  
val max_revenue_transport = spark.sql("select source,destination,year,  
(count(transport_mode) * cost_per_unit) as Revenue from holi_trans_mode group by  
source,destination,year,transport_mode,cost_per_unit order by Revenue desc")  
max_revenue_transport.createOrReplaceTempView("max_revenue_transport")  
max_revenue_transport.show  
//selecting the route that genarates maximum revenue each year from  
max_revenue_transport view & storing the result in a6.  
val a6 = spark.sql("select source,destination,year,Revenue from max_revenue_transport  
group by source ,destination,year,Revenue Having Revenue IN (select Max(Revenue) as  
most_revenue_generated_year from max_revenue_transport)")  
a6.show  
// Joining two views holiday & transport to fetch cost_per_unit from transport for  
Air_travellers only & all rows from holiday.  
val holi_trans_mode_airplane = spark.sql("select h.*, t.cost_per_unit from holidays h  
join transport t on h.transport_mode = t.transport_mode where  
t.transport_mode='airplane'")  
holi_trans_mode_airplane.createOrReplaceTempView("holi_trans_mode_airplane")  
holi_trans_mode_airplane.show
```

```

//To fetch the total amount spent by a particular user each year grouping by id,year &
counting by each id,year
// & multiplying it by cost.
val a7 = spark.sql("select id,year,(count(id,year) * cost_per_unit) as `Total Amount
Spent` from holi_trans_mode_airplane group by id,year ,cost_per_unit order by id,year
asc")

a7.show
// Create an RDD of from a text file User_details.txt.
val user_schema_DF1 =
spark.sparkContext.textFile("C:/Users/mypc/Desktop/User_details_schema.txt").map(_s
plit(",")).map(x=>(x(0).trim.toLong,x(1),x(2).trim.toLong))
// Create an RDD of from a text file Dataset Holidays.txt.
val holidays_DF1
=spark.sparkContext.textFile("C:/Users/mypc/Desktop/Dataset_Holidays.txt").map(_split
(",")).map(x=>(x(0).trim.toLong,x(1),x(2),x(3),x(4).trim.toLong,x(5).trim.toLong))
val transport_DF1 =spark.sparkContext

.textFile("C:/Users/mypc/Desktop/Dataset_Transport_Schema.txt").map(_split(",")).map
(x=>Transport(x(0),x(1).trim.toInt))
// create an RDD AgeGroup from user to get different age-groups from age column.
val age_group = user_schema_DF1.map(x => x._1 ->{if(x._3<20)"20" else if (x._3>35)"35"
else "25-35"})
// create an RDD year_holiday_travel from travel to map id as key and (distance and
year) as value
val year_holiday_travel = holidays_DF1.map(x => (x._1 -> (x._6,x._5)))
// create an RDD travelMap to join age_Group and year-holiday_travel
val travelMap= age_group.join(year_holiday_travel)
val ageTravelMap= travelMap.map(x => (x._2._1 ,x._2._2._1) -> x._2._2._2)
// create an RDD to aggregate the keys year and age-groups
val ageTravelReduce = ageTravelMap.reduceByKey((x,y)=> x+y).sortByKey()
//convert the RDD yearGroupSort to a Dataframe
val yearGroupSort = ageTravelReduce.map( x => (x._1._2,x._1._1,x._2)).toDF
//Now we use spark-sql to get the output....

val newName = Seq("year","ageGroup","Distance")
//Schema of yearGroupSort is (.1,.2,.3),convert it into (year,ageGroup,Distance) in
yearGroupSortNew
val yearGroupSortNew = yearGroupSort.toDF(newName: _*)
// to check the new shema of yearGroupSortNew Data Frame
yearGroupSortNew.printSchema()
// Register the DataFrame as a temporary view AGEGROUP
yearGroupSortNew.createOrReplaceTempView("AGEGROUP")
val max_distance_per_year = spark.sql("SELECT a.*FROM AGEGROUP a " +
"INNER JOIN " +
"(SELECT year, MAX(distance) AS max FROM AGEGROUP " +
"GROUP BY year) b " +
"ON a.year = b.year " +

```

```

"AND a.distance = b.max ").show()

println("Above results shows the age group travelling the most every year")

}

}

```

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

// To find distribution of Air travellers/year we need to find count of id or year for each using a group by method.

```

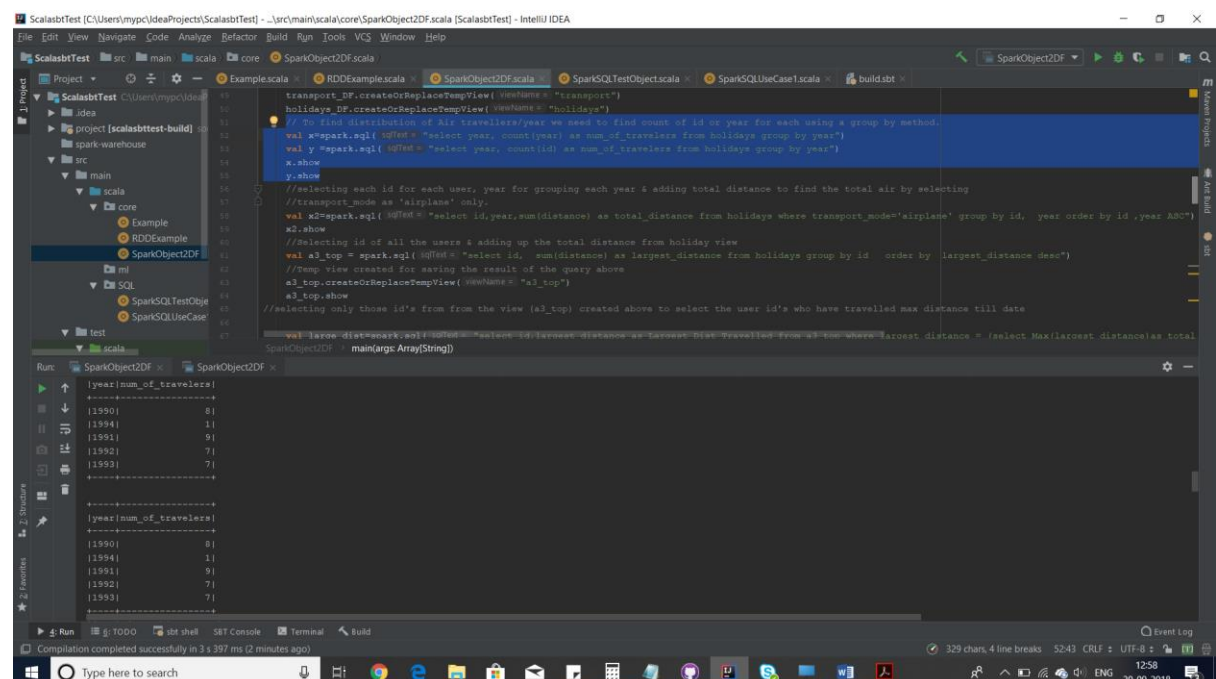
val x=spark.sql("select year, count(year) as num_of_travelers from holidays group by year")

val y=spark.sql("select year, count(id) as num_of_travelers from holidays group by year")

x.show

y.show

```

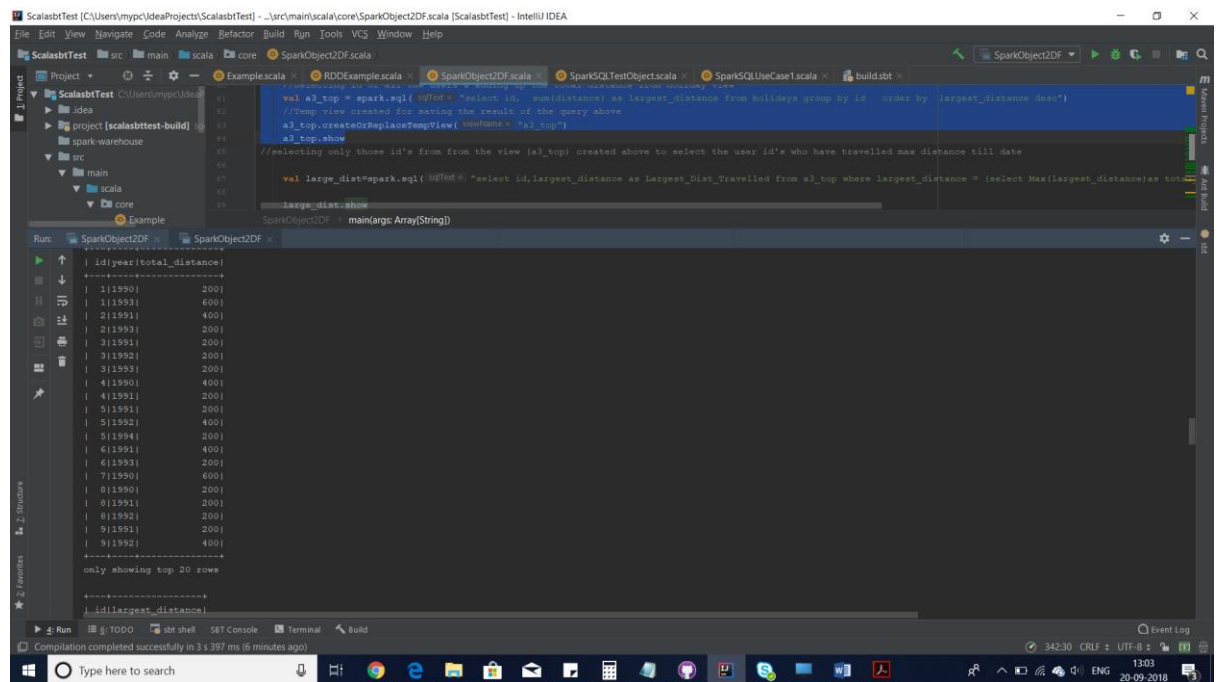


- 2) What is the total air distance covered by each user per year  
//selecting each id for each user, year for grouping each year & adding total distance to find the total air by selecting

```

//transport_mode as 'airplane' only.
val x2=spark.sql("select id,year,sum(distance) as total_distance from holidays where
transport_mode='airplane' group by id, year order by id ,year ASC")
x2.show
//Selecting id of all the users & adding up the total distance from holiday view
val a3_top = spark.sql("select id, sum(distance) as largest_distance from holidays group
by id order by largest_distance desc")
//Temp view created for saving the result of the query above
a3_top.createOrReplaceTempView("a3_top")
a3_top.show

```



### 3) Which user has travelled the largest distance till date

```

4) //selecting only those id's from from the view (a3_top) created above to select the user id's who
have travelled max distance till date

val large_dist=spark.sql("select id,largest_distance as Largest_Dist_Travelled from a3_top
where largest_distance = (select Max(largest_distance)as total_distance_covered from a3_top)")

large_dist.show
//Selecting destination & count of destination & providing an alias Dest_Visited from holiday
view & creating an Alias
val a4_top_dest = spark.sql("select destination, count(destination) as Dest_Visited from
holidays group by destination")
////Temp view created for saving the result of the query above
a4_top_dest.createOrReplaceTempView("a4_top_dest")
a4_top_dest.show

```

//selecting only those id's from from the view (a3\_top) created above to select the user id's who have travelled max distance till date

```

val large_dist=spark.sql("select id,largest_distance as Largest_Dist_Travelled from
a3_top where largest_distance = (select Max(largest_distance)as total_distance_covered
from a3_top)")

```

large\_dist.show

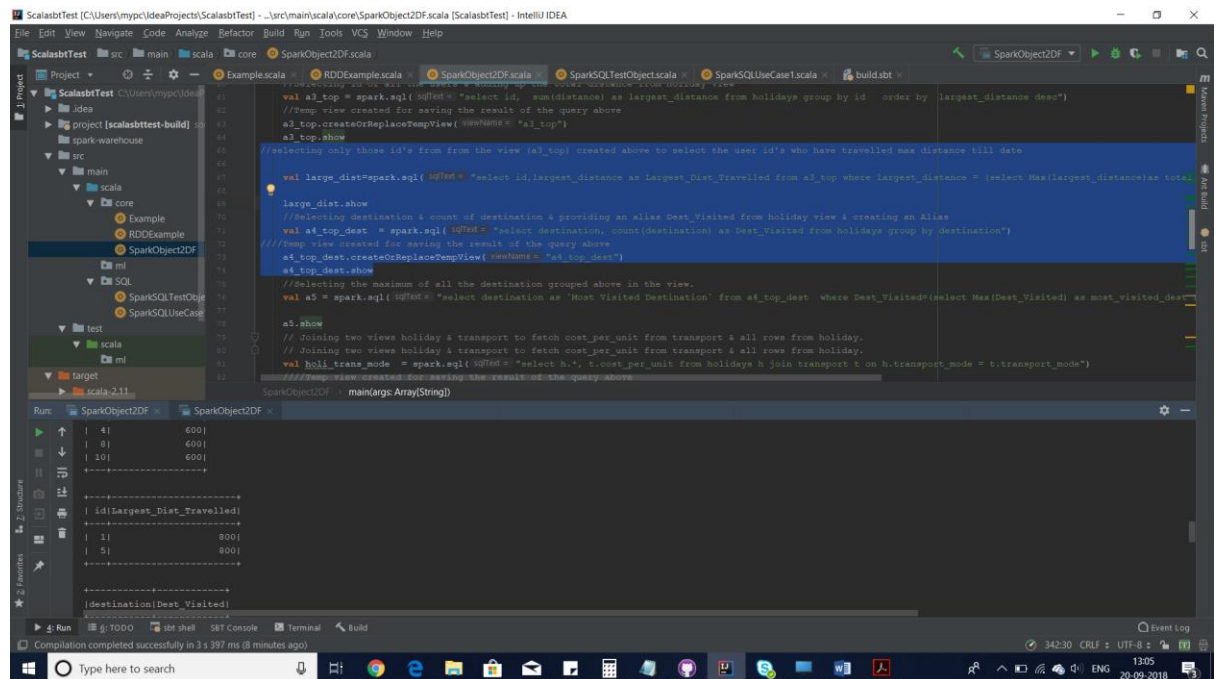
//Selecting destination & count of destination & providing an alias Dest\_Visited from holiday view & creating an Alias

val a4\_top\_dest = spark.sql("select destination, count(destination) as Dest\_Visited from holidays group by destination")

////Temp view created for saving the result of the query above

a4\_top\_dest.createOrReplaceTempView("a4\_top\_dest")

a4\_top\_dest.show



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

//Selecting destination & count of destination & providing an alias Dest\_Visited from holiday view & creating an Alias

val a4\_top\_dest = spark.sql("select destination, count(destination) as Dest\_Visited from holidays group by destination")

////Temp view created for saving the result of the query above

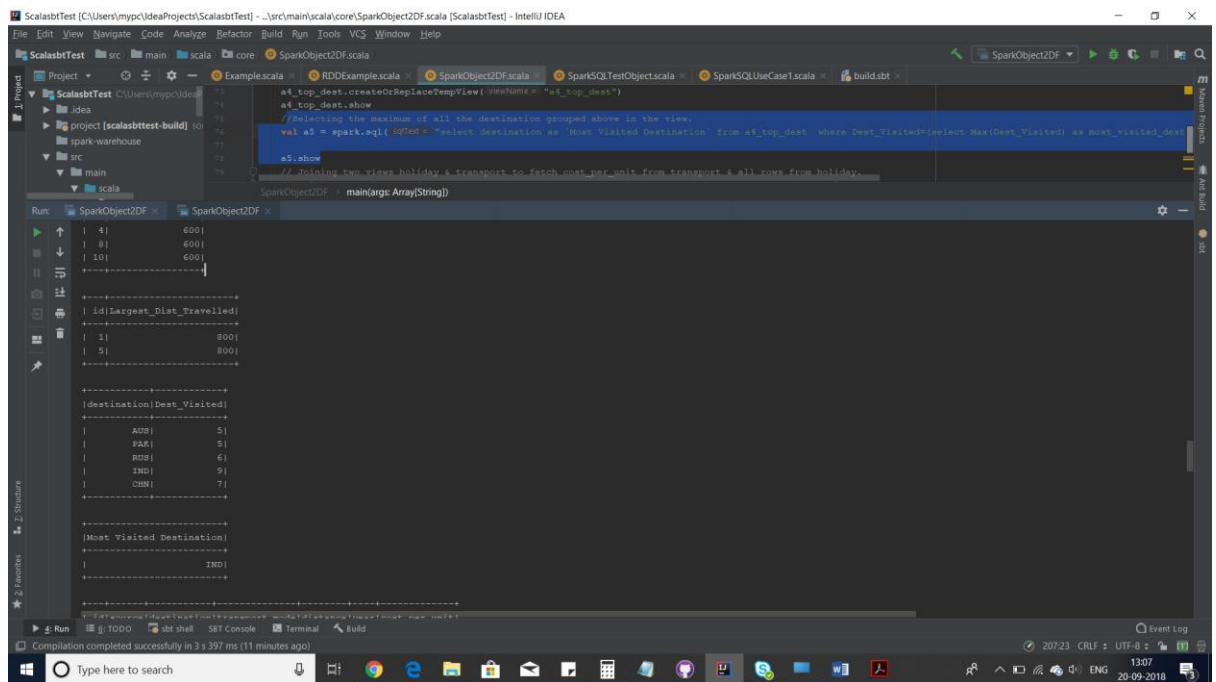
a4\_top\_dest.createOrReplaceTempView("a4\_top\_dest")

a4\_top\_dest.show

//Selecting the maximum of all the destination grouped above in the view.

val a5 = spark.sql("select destination as `Most Visited Destination` from a4\_top\_dest where Dest\_Visited=(select Max(Dest\_Visited) as most\_visited\_dest from a4\_top\_dest)")

a5.show



5) Which route is generating the most revenue per year

// Joining two views holiday & transport to fetch cost\_per\_unit from transport & all rows from holiday.

// Joining two views holiday & transport to fetch cost\_per\_unit from transport & all rows from holiday.

```
val holi_trans_mode = spark.sql("select h.*, t.cost_per_unit from holidays h join transport t on h.transport_mode = t.transport_mode")
```

//// Temp view created for saving the result of the query above

```
holi_trans_mode.createOrReplaceTempView("holi_trans_mode")
```

```
holi_trans_mode.show
```

// From the holi\_trans view above selecting the route & multiplying the count of transport\_mode with its respective cost/unit

// to fetch the revenue generated by a particular route each year.

```
val max_revenue_transport = spark.sql("select source, destination, year, (count(transport_mode) * cost_per_unit) as Revenue from holi_trans_mode group by source, destination, year, transport_mode, cost_per_unit order by Revenue desc")
```

```
max_revenue_transport.createOrReplaceTempView("max_revenue_transport")
```

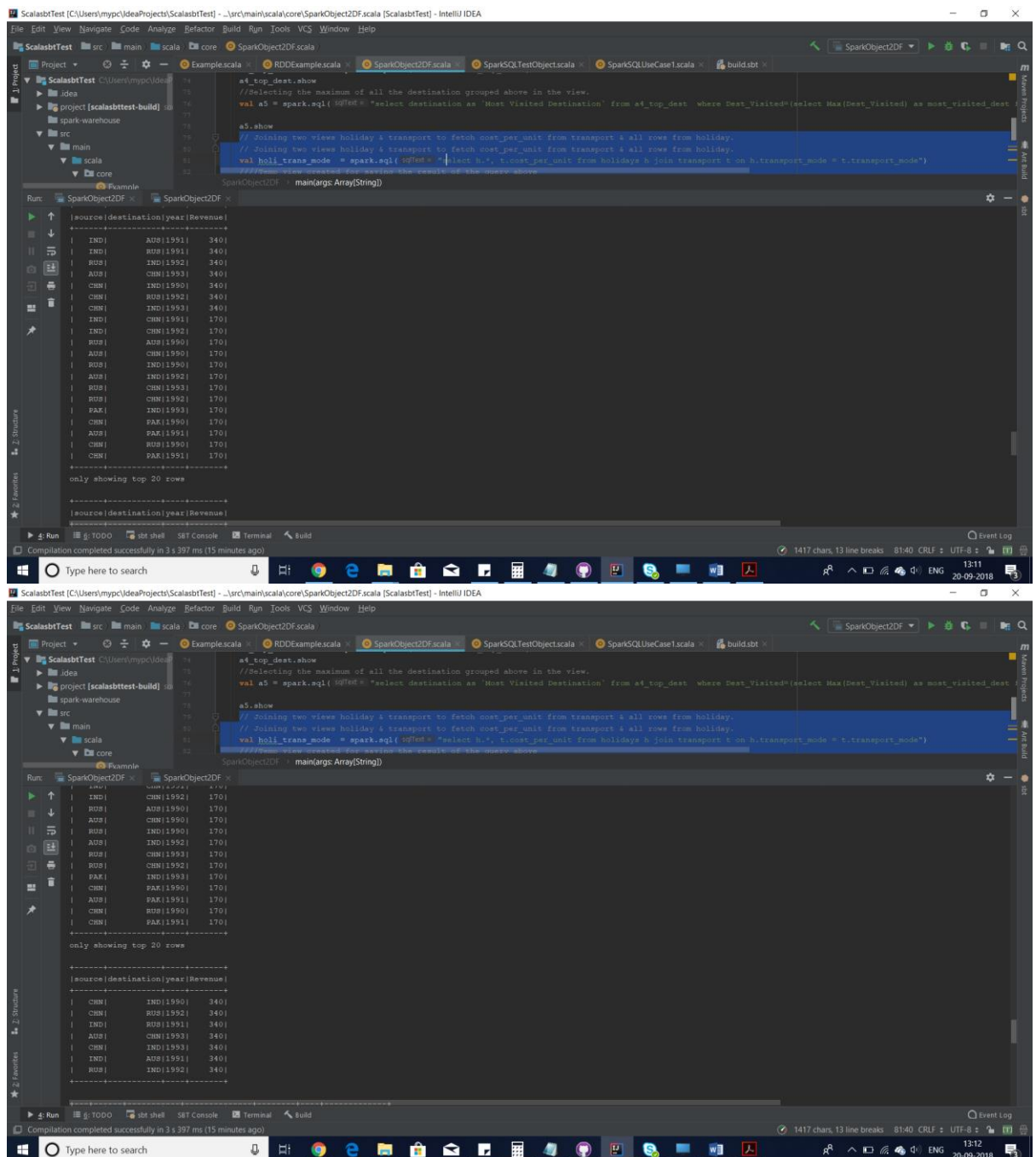
```
max_revenue_transport.show
```

// selecting the route that generates maximum revenue each year from max\_revenue\_transport view & storing the result in a6.

```
val a6 = spark.sql("select source, destination, year, Revenue from max_revenue_transport group by source, destination, year, Revenue Having Revenue IN (select Max(Revenue) as most_revenue_generated_year from max_revenue_transport)")
```

```
a6.show
```





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

//To fetch the total amount spent by a particular user each year grouping by id,year & counting by each id,year

// & multiplying it by cost.

```

val holi_trans_mode_airplane = spark.sql("select h.*, t.cost_per_unit from holidays h join transport t on h.transport_mode = t.transport_mode where t.transport_mode='airplane'")
holi_trans_mode_airplane.createOrReplaceTempView("holi_trans_mode_airplane")
holi_trans_mode_airplane.show

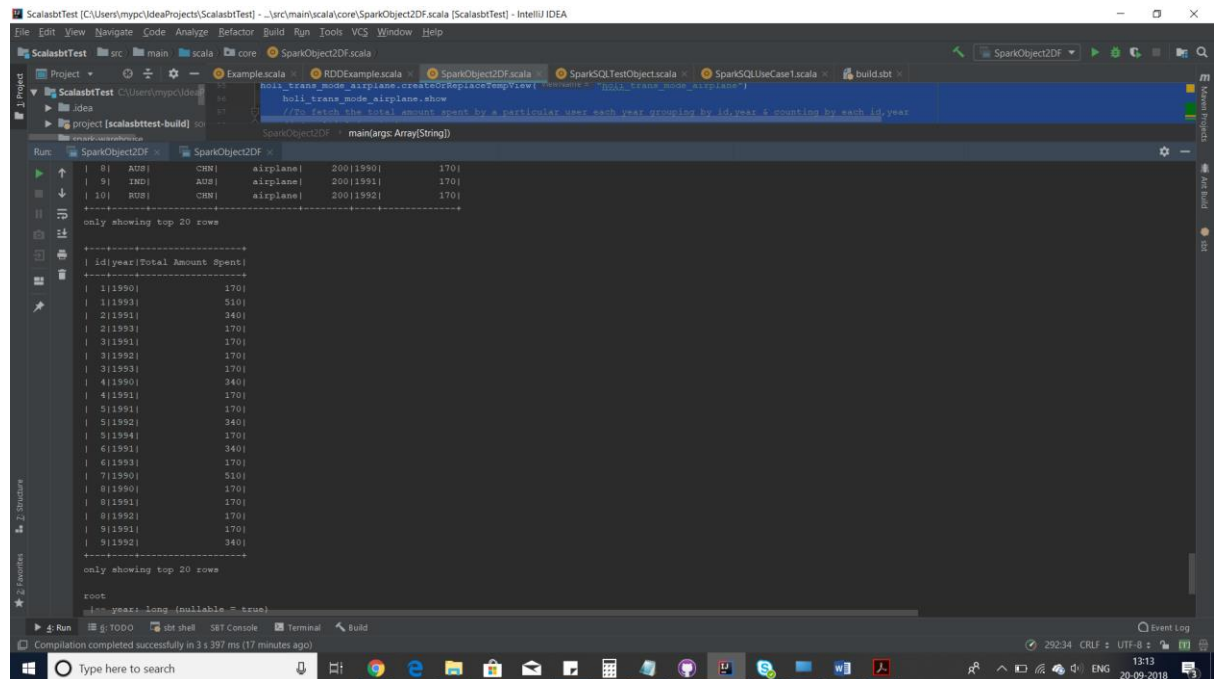
```

```
//To fetch the total amount spent by a particular user each year grouping by id,year &
counting by each id,year
```

```
// & multiplying it by cost.
```

```
val a7 = spark.sql("select id,year,(count(id,year) * cost_per_unit) as `Total Amount
Spent` from holi_trans_mode_airplane group by id,year ,cost_per_unit order by id,year
asc")
```

```
a7.show
```



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

```
// Create an RDD of from a text file User_details.txt.
```

```
val user_schema_DF1 =
```

```
spark.sparkContext.textFile("C:/Users/mypc/Desktop/User_details_schema.txt").map(_split(",")).map(x=>(x(0).trim.toLong,x(1),x(2).trim.toLong))
```

```
// Create an RDD of from a text file Dataset Holidays.txt.
```

```
val holidays_DF1
```

```
=spark.sparkContext.textFile("C:/Users/mypc/Desktop/Dataset_Holidays.txt").map(_split(",")).map(x=>(x(0).trim.toLong,x(1),x(2),x(3),x(4).trim.toLong,x(5).trim.toLong))
```

```
val transport_DF1 =spark.sparkContext
```

```
.textFile("C:/Users/mypc/Desktop/Dataset_Transport_Schema.txt").map(_split(",")).map(x=>Transport(x(0),x(1).trim.toInt))
```

```
// create an RDD AgeGroup from user to get different age-groups from age column.
```

```
val age_group = user_schema_DF1.map(x => x._1 -> {if(x._3<20)"20" else if (x._3>35)"35" else "25-35"})
```

```

    // create an RDD year_holiday_travel from travel to map id as key and (distance and
year) as value
    val year_holiday_travel = holidays_DF1.map(x => (x._1 -> (x._6,x._5)))
    // create an RDD travelMap to join age_Group and year-holiday_travel
val travelMap= age_group.join(year_holiday_travel)
    val ageTravelMap= travelMap.map(x => (x._2._1 ,x._2._2._1) -> x._2._2._2)
    // create an RDD to aggregate the keys year and age-groups
    val ageTravelReduce = ageTravelMap.reduceByKey((x,y)=> x+y).sortByKey()
    //convert the RDD yearGroupSort to a Dataframe
    val yearGroupSort = ageTravelReduce.map( x => (x._1._2,x._1._1,x._2)).toDF
    //Now we use spark-sql to get the output....

    val newName = Seq("year","ageGroup","Distance")
    //Schema of yearGroupSort is (. _1,._2,._3),convert it into (year,ageGroup,Distance) in
yearGroupSortNew
    val yearGroupSortNew = yearGroupSort.toDF(newName: _*)
    // to check the new shema of yearGroupSortNew Data Frame
    yearGroupSortNew.printSchema()
    // Register the DataFrame as a temporary view AGEGROUP
    yearGroupSortNew.createOrReplaceTempView("AGEGROUP")
    val max_distance_per_year = spark.sql("SELECT a.*FROM AGEGROUP a " +
    "INNER JOIN " +
    "(SELECT year, MAX(distance) AS max FROM AGEGROUP " +
    "GROUP BY year) b " +
    "ON a.year = b.year " +
    "AND a.distance = b.max ").show()

    println("Above results shows the age group travelling the most every year")

}
}

```

```
ScalaTest [C:\Users\myopc\IdeaProjects\ScalaTest] - ...src\main\scala\core\SparkObject2DF.scala [ScalaTest] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
ScalaTest [C:\Users\myopc\IdeaProjects\ScalaTest]
Project src main scala core SparkObject2DF.scala
SparkObject2DF.scala SparkSQLTestObject.scala SparkSQLUseCase1.scala build.sbt
val transportDF1 = spark.sparkContext
SparkObject2DF main(args: Array[String])
Run SparkObject2DF SparkObject2DF
411990 340
41991 170
51991 170
51992 340
51994 170
61991 340
61993 170
71990 510
81990 170
81991 170
81992 170
91991 170
91992 340
-----
only showing top 20 rows
root
|-- year: long (nullable = true)
|-- ageGroup: string (nullable = true)
|-- Distance: long (nullable = true)
-----
(year|ageGroup|Distance)
-----
(1993) 20 1000
(1992) 35 800
(1991) 25-35 800
(1994) 25-35 200
(1990) 25-35 1000
-----
Above results shows the age group travelling the most every year
Process finished with exit code 0
Run g:1000 SBT shell SBT Console Terminal Build
Compilation completed successfully in 3 s 397 ms (18 minutes ago)
Type here to search 366.1 CRLF : UTF-8 13:14 20-09-2018
```