**ACD\_BDDOF\_Session\_6\_Assignment\_5\_Main**

**Problem Statement:**

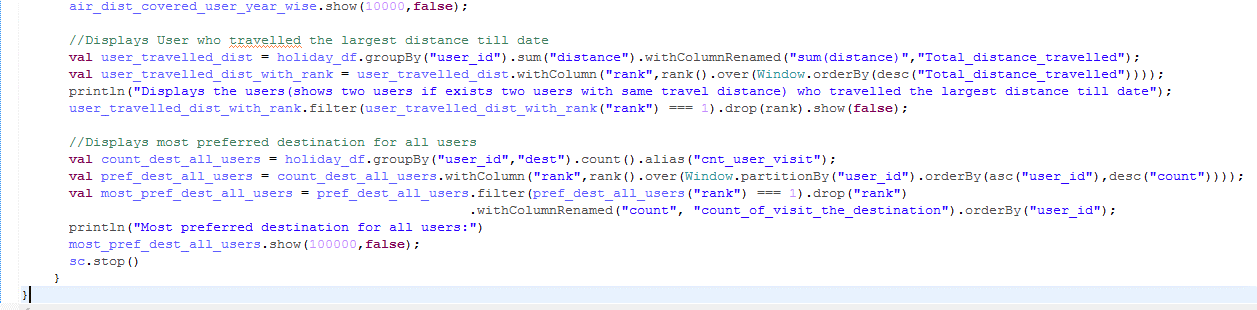
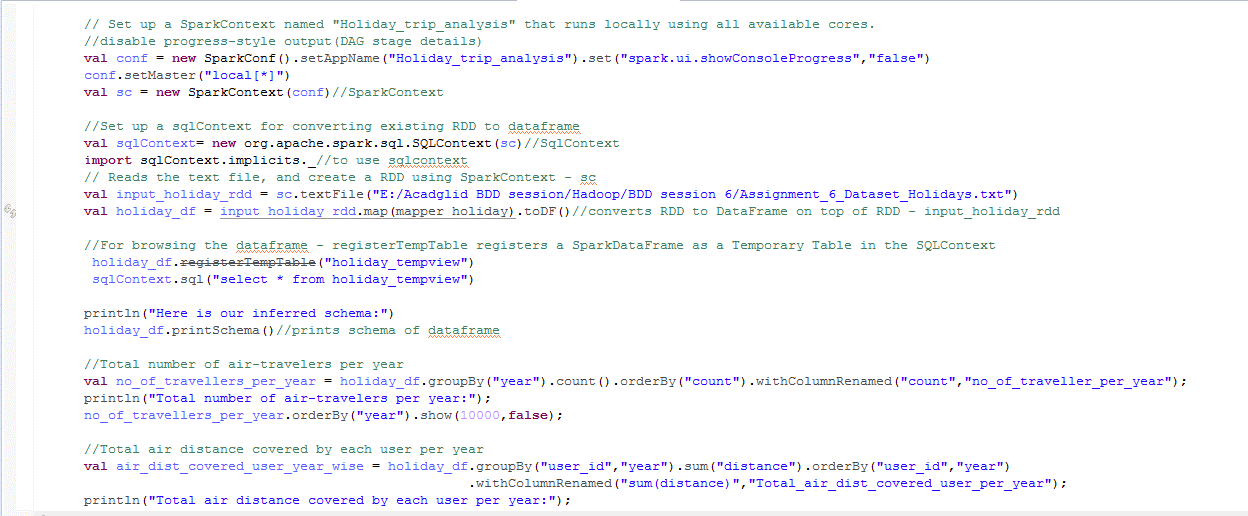
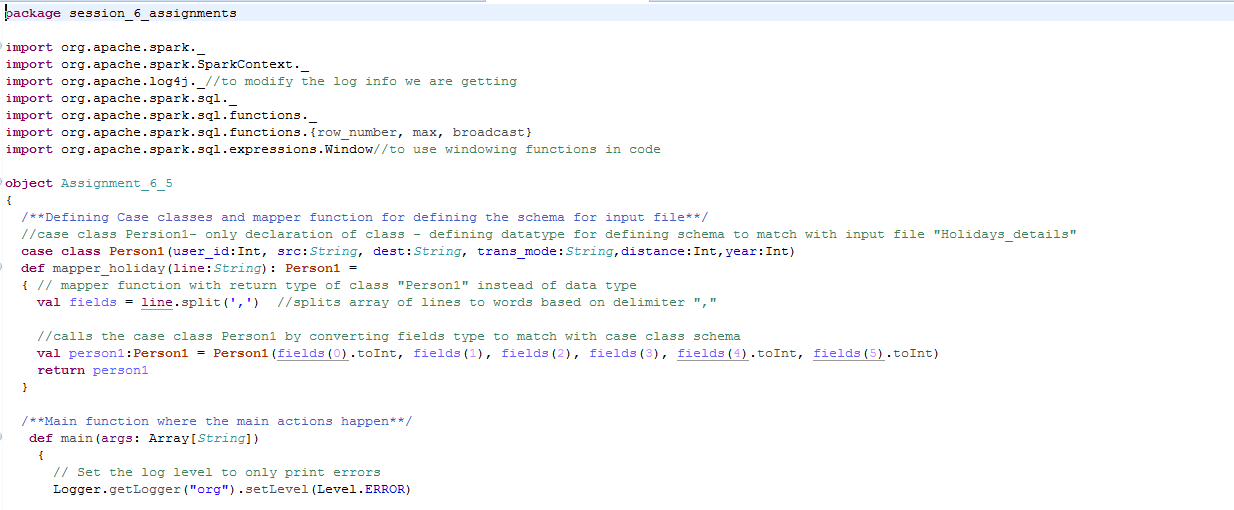
1) What is the distribution of the total number of air-travelers per year

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.

**Scala code:**

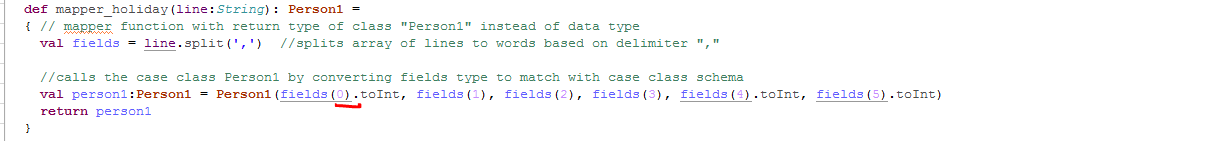


**Explanation:**

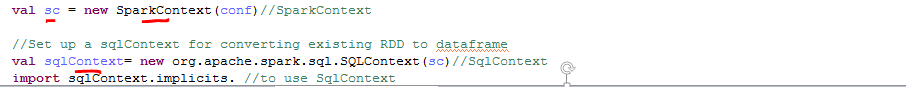
* Created 1 **case class** called “Person1” for defining the schema and name of the fields in input file – holidays file

Person1(Holidays file) 

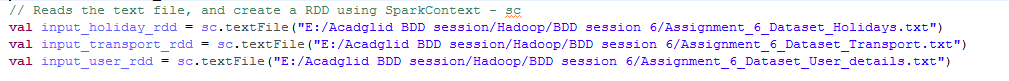
* Created a **mapper function** for holidays files – which will convert the datatype of the fields to match with corresponding Person class schema and returns “Person” type



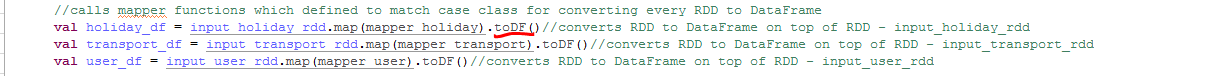
* Set the **log level to “Error”** to display only error messages - Logger.getLogger("org").setLevel(Level.ERROR) **and disabled progress-style** output(DAG stage details) - set("spark.ui.showConsoleProgress","false")
* Set up a **SparkContext** named – sc – for application "Holiday\_trip\_analysis" that runs locally using all available cores.
* Set up a **sqlContext** for converting existing RDD to **dataframe**



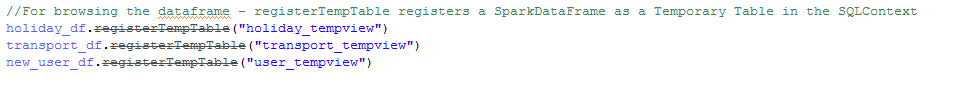
* Reads the text file, and create a RDD using SparkContext – sc



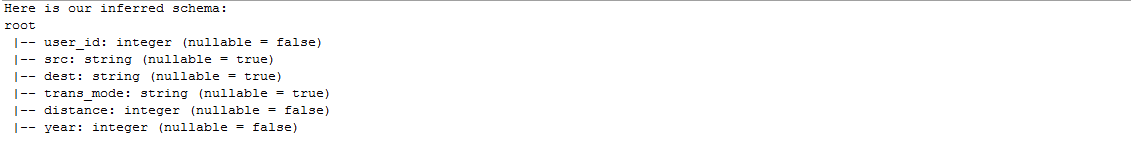
* Calls mapper functions which defined to match case class for converting every RDD to DataFrame
* Calls mapper functions by passing mapper function with RDD and retunrs Dataframe as toDF() is used



* For browsing the dataframe **- registerTempTable registers a SparkDataFrame** as a Temporary Table in the SQLContext and Sqlcontext holds the date of data frame

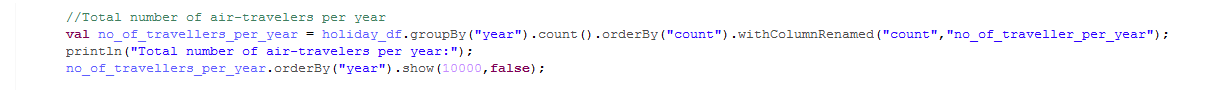


* Now as the data from RDD is converted to Dataframe,it can be queried with “SparkContext”.
* Can query the created view and can get the schema of the stored data in dataframe with printSchema()

**Prints Schema of dataframe – holiday\_df – which holds data from holidays file:**

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

* Grouped data in dataframe - holiday\_df **based on “year” and taken count of rows which is count of travelers in that year.**
* Displayed the result ordered by year with show(10000,false) – to print first 10000 lines and disable truncating column values in output console



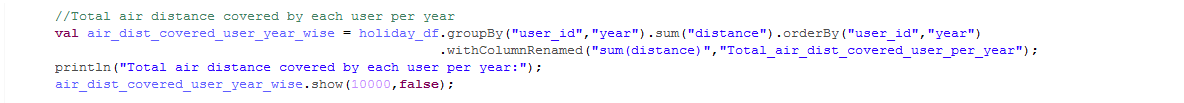
**Output:**

Displays the year and total number of travelers in that year.



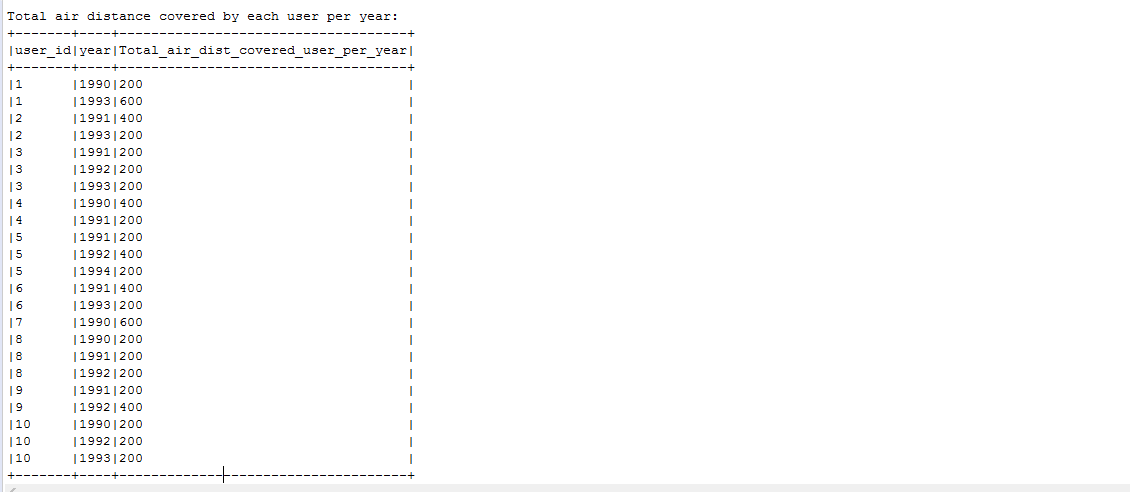
**2) What is the total air distance covered by each user per year**

* Grouped data in dataframe - holiday\_df **based on “user\_id”,“year” and taken sum(distance)** from same file - which is **total air distance** covered by each user in each year
* Ordered the above data which is grouped based on “user\_id”,”year” displayed the result ordered based on user\_id,year with show(10000,false) – so as to print first 10000 lined and disable truncating column values in output console



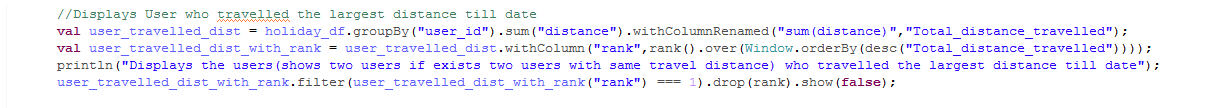
**Output:**

Displays the user\_id,year(ordered with each user\_id) along with total air distance covered by user in that year ordered based on user\_id.



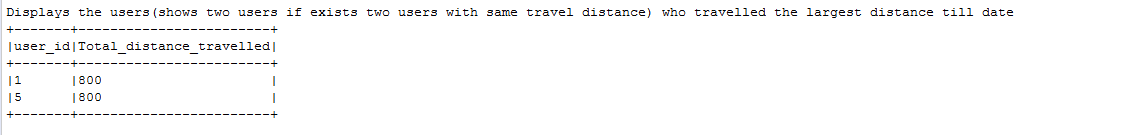
**3) Which user has travelled the largest distance till date**

* Grouped dataframe – holiday\_df based on **“user\_id”** and get the sum of distance - will give the sum(distance) travelled by user in all year till data
* Using windowing function ordered data by **desc(sum(distance))** and assigned rankand **get the first row** – will **get one row of a user or more than one user(if both user holds same amount of total distance)** which is having max(distance) till date.
* Displayed the result with show(false) – for disabling truncating column values in output console



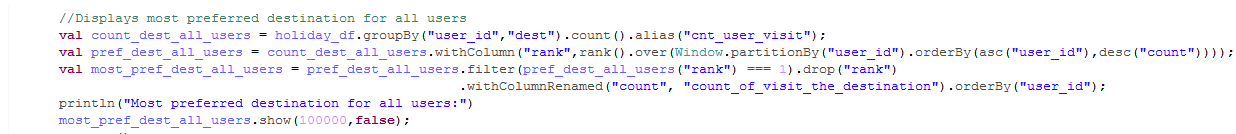
**Output:**

Displays the user or users(if both user holds same amount of total distance) who travelled largest distance in all years till date. Displays user\_id 1 & 5 as both has total distance traveled as “800”.(which is maximum of all)



**4) What is the most preferred destination for all users.**

* Grouped data frame in “holiday\_df” based on **“user\_id”,”dest”** and get **the count of rows** which of count of visits of a user to that destination - will give the count of visits according to each user and destination
* **Most preferred destination – user should have gone to that destination more times compared to other destinations**
* Using windowing function **assigned rank** to dataset – wherein data is **partitioned by “user” and ordered based on “count of visits” in descending order and get the first row** – this will get one row or more rows (if both destination is having same count of visits) for each user which is having more count
* Displayed the user\_id,dest with show(10000,false) – so as to print first 10000 lined and disable truncating column values in output console



**Output:**

Displays the user\_id,destination(which is preferred destination for that user as it is been visited by user more times) along with count of visits to that destination.

This will get one row or more rows (if both destination is having same count of visits) for each user(like for user 2 both RUS and CHN has one visit).

