Session 20: SPARK SQL 1 Assignment 1

- Prachi Mohite

Spark SQL. It provides a programming abstraction called DataFrame and can act as distributed SQL query engine.

Features of Spark SQL

- **Integrated** − Seamlessly mix SQL queries with Spark programs. Spark SQL lets you query structured data as a distributed dataset (RDD) in Spark, with integrated APIs in Python, Scala and Java. This tight integration makes it easy to run SQL queries alongside complex analytic algorithms.

- **Unified Data Access** − Load and query data from a variety of sources. Schema-RDDs provide a single interface for efficiently working with structured data, including Apache Hive tables, parquet files and JSON files.

- **Hive Compatibility** − Run unmodified Hive queries on existing warehouses. Spark SQL reuses the Hive frontend and MetaStore, giving you full compatibility with existing Hive data, queries, and UDFs. Simply install it alongside Hive.

- **Standard Connectivity** − Connect through JDBC or ODBC. Spark SQL includes a server mode with industry standard JDBC and ODBC connectivity.

- **Scalability** − Use the same engine for both interactive and long queries. Spark SQL takes advantage of the RDD model to support mid-query fault tolerance, letting it scale to large jobs too. Do not worry about using a different engine for historical data.

- A DataFrame is a distributed collection of data, which is organized into named columns. Conceptually, it is equivalent to relational tables with good optimization techniques.
- A DataFrame can be constructed from an array of different sources such as Hive tables, Structured Data files, external databases, or existing RDDs.

Features of DataFrame

- Ability to process the data in the size of Kilobytes to Petabytes on a single node cluster to large cluster.

- Supports different data formats (Avro, csv, elastic search, and Cassandra) and storage systems (HDFS, HIVE tables, mysql, etc).

- State of art optimization and code generation through the Spark SQL Catalyst optimizer (tree transformation framework).

- Can be easily integrated with all Big Data tools and frameworks via Spark-Core.

- Provides API for Python, Java, Scala, and R Programming.

SQLContext

SQLContext is a class and is used for initializing the functionalities of Spark SQL. SparkContext class object (sc) is required for initializing SQLContext class object.

The following command is used for initializing the SparkContext through spark-shell.

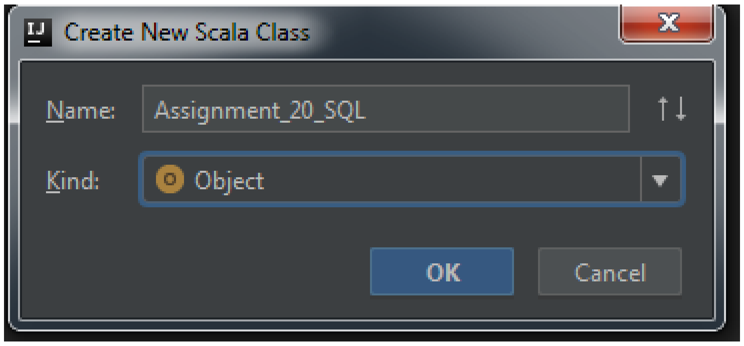
By default, the SparkContext object is initialized with the name sc when the spark-shell starts.

Use the following command to create SQLContext.

```
scala> val sqlcontext = new org.apache.spark.sql.SQLContext(sc)
```

In this Assignment we will be using IDEA IntelliJ to Complete the given Task

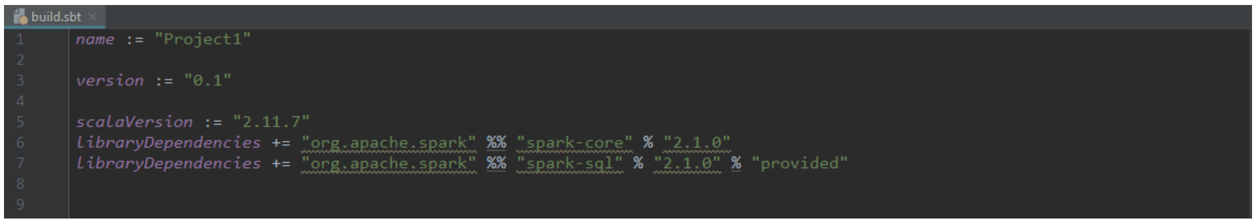
1. Created new Project and added scala object named as Assignment_20 as below



Name: Assignment_20_SQL

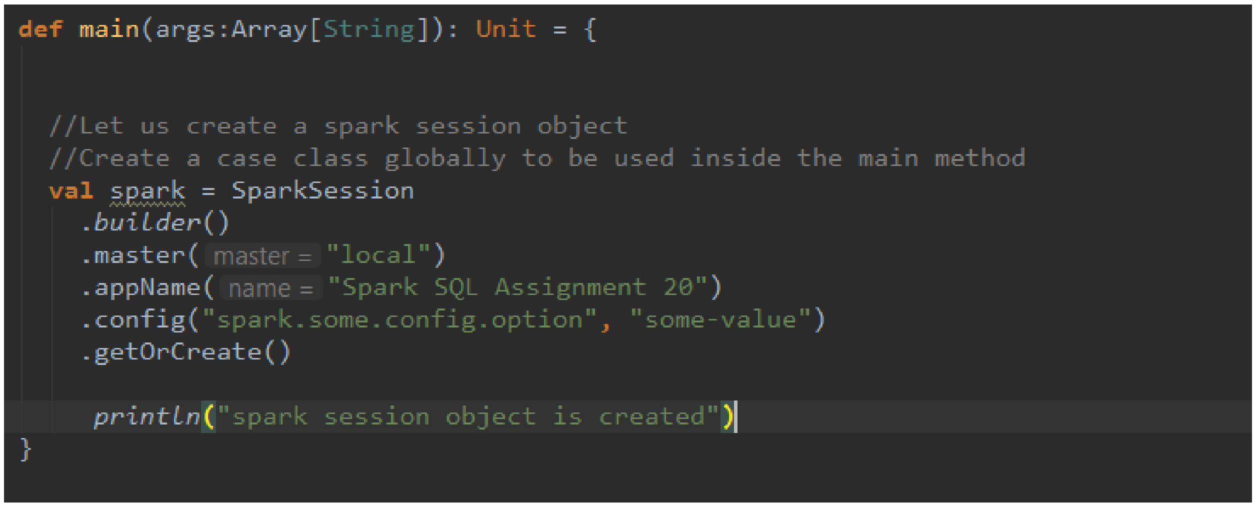
Kind: Object

2. To add the required dependencies we have created scala sbt project in IDEA and added library dependency from maven repository as below



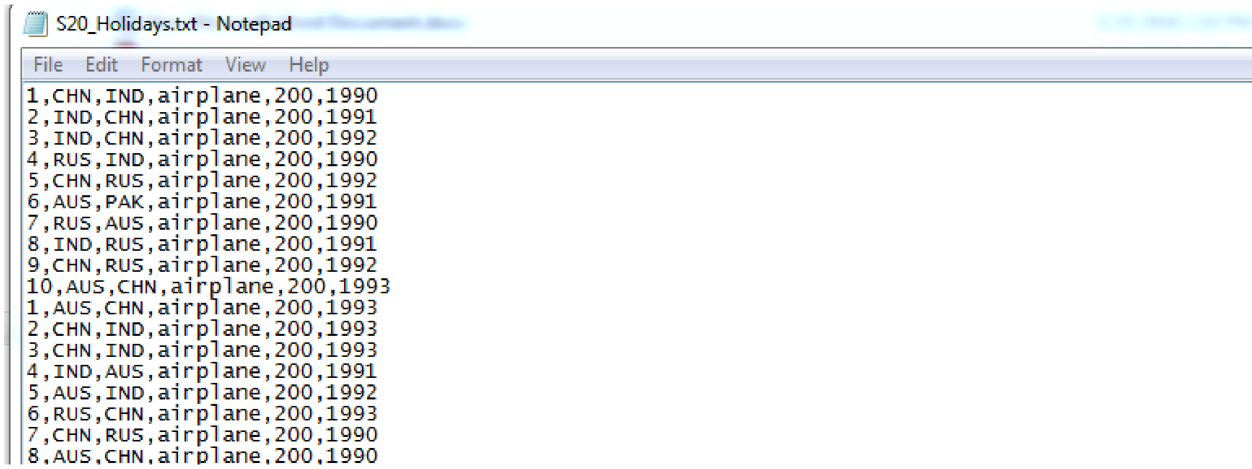
```
name := "Project1"

version := "0.1"

scalaVersion := "2.11.7"
libraryDependencies += "org.apache.spark" %% "spark-core" % "2.1.0"
libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.1.0" % "provided"
```

3. Added main function and created the spark object as below



```
def main(args:Array[String]): Unit = {


  //Let us create a spark session object
  //Create a case class globally to be used inside the main method
  val spark = SparkSession
    .builder()
    .master( master = "local")
    .appName( name = "Spark SQL Assignment 20")
    .config("spark.some.config.option", "some-value")
    .getOrCreate()

     println("spark session object is created")
}
```

4. We will be using below datasets for this assignment
   a. Holiday Details

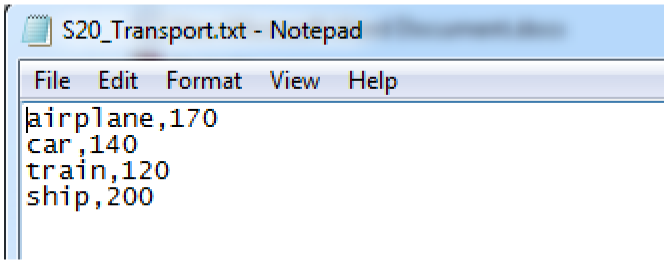


S20_Holidays.txt - Notepad

File Edit Format View Help

```
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
```

   b. Columns are - UserID , Country departure from, Country where User is arriving, mode of travel , distance and year
   c. Transport Details

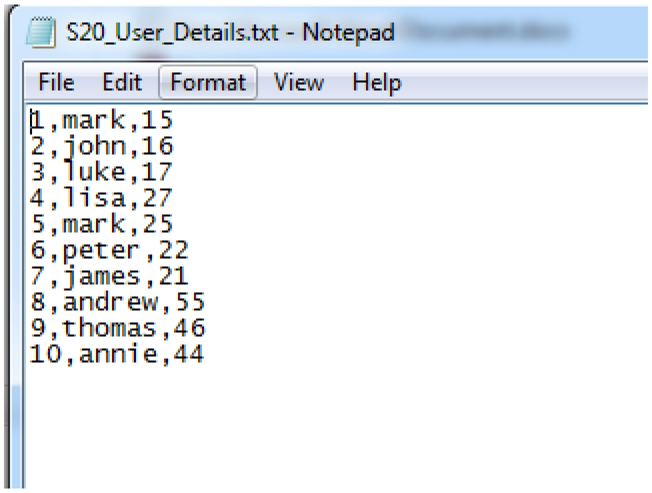


S20_Transport.txt - Notepad

File Edit Format View Help

```
airplane,170
car,140
train,120
ship,200
```

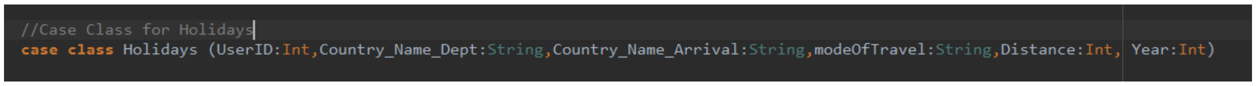
   d. Columns are – Mode of Travel, Travel Expenses
   e. User Details



```
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
```

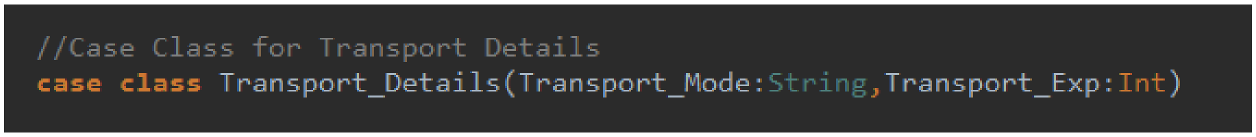
f. Columns are – UserID, Name, age

5. To Complete the assignment first we have to load the data from these local files to Dataframes in Spark SQL as below
   a. Created the case class to map the details in Dataframe from text files
      i. Case Class for Holiday Details



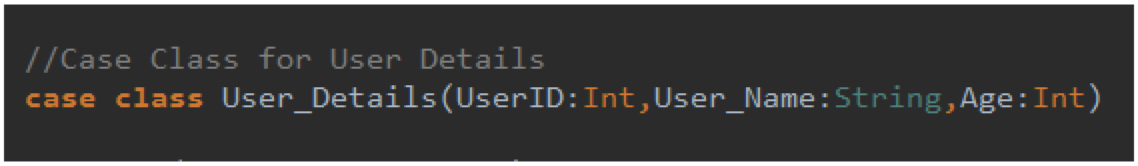
```scala
//Case Class for Holidays
case class Holidays (UserID:Int,Country_Name_Dept:String,Country_Name_Arrival:String,modeOfTravel:String,Distance:Int, Year:Int)
```

      ii. Case Class for Transport Details



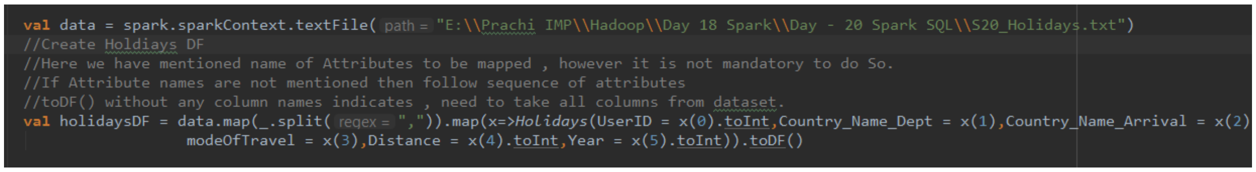
```scala
//Case Class for Transport Details
case class Transport_Details(Transport_Mode:String,Transport_Exp:Int)
```

      iii. Case Class for User Details

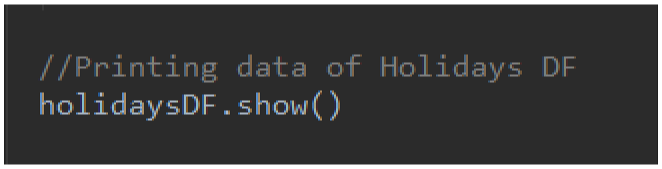


```scala
//Case Class for User Details
case class User_Details(UserID:Int,User_Name:String,Age:Int)
```

   b. Loaded data in frames
      i. Loaded Holiday details

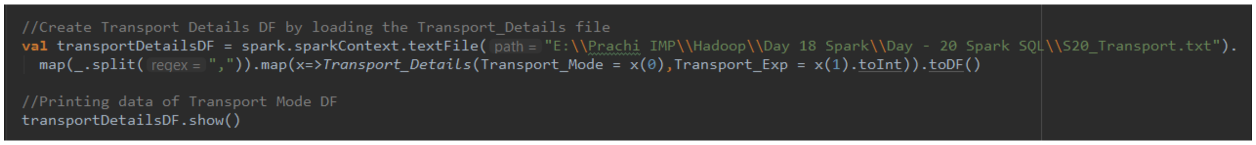


```scala
val data = spark.sparkContext.textFile( path = "E:\\Prachi IMP\\Hadoop\\Day 18 Spark\\Day - 20 Spark SQL\\S20_Holidays.txt")
//Create Holdiays DF
//Here we have mentioned name of Attributes to be mapped , however it is not mandatory to do So.
//If Attribute names are not mentioned then follow sequence of attributes
//toDF() without any column names indicates , need to take all columns from dataset.
val holidaysDF = data.map(_.split( regex = ",")).map(x=>Holidays(UserID = x(0).toInt,Country_Name_Dept = x(1),Country_Name_Arrival = x(2)
               modeOfTravel = x(3),Distance = x(4).toInt,Year = x(5).toInt)).toDF()
```



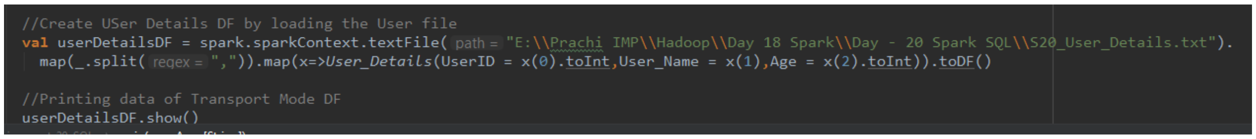
```
//Printing data of Holidays DF
holidaysDF.show()
```

ii. Loaded Transport Details



```
//Create Transport Details DF by loading the Transport_Details file
val transportDetailsDF = spark.sparkContext.textFile( path = "E:\\Prachi IMP\\Hadoop\\Day 18 Spark\\Day - 20 Spark SQL\\S20_Transport.txt").
  map(_.split( regex = ",")).map(x=>Transport_Details(Transport_Mode = x(0),Transport_Exp = x(1).toInt)).toDF()

//Printing data of Transport Mode DF
transportDetailsDF.show()
```

iii. Loaded User Details



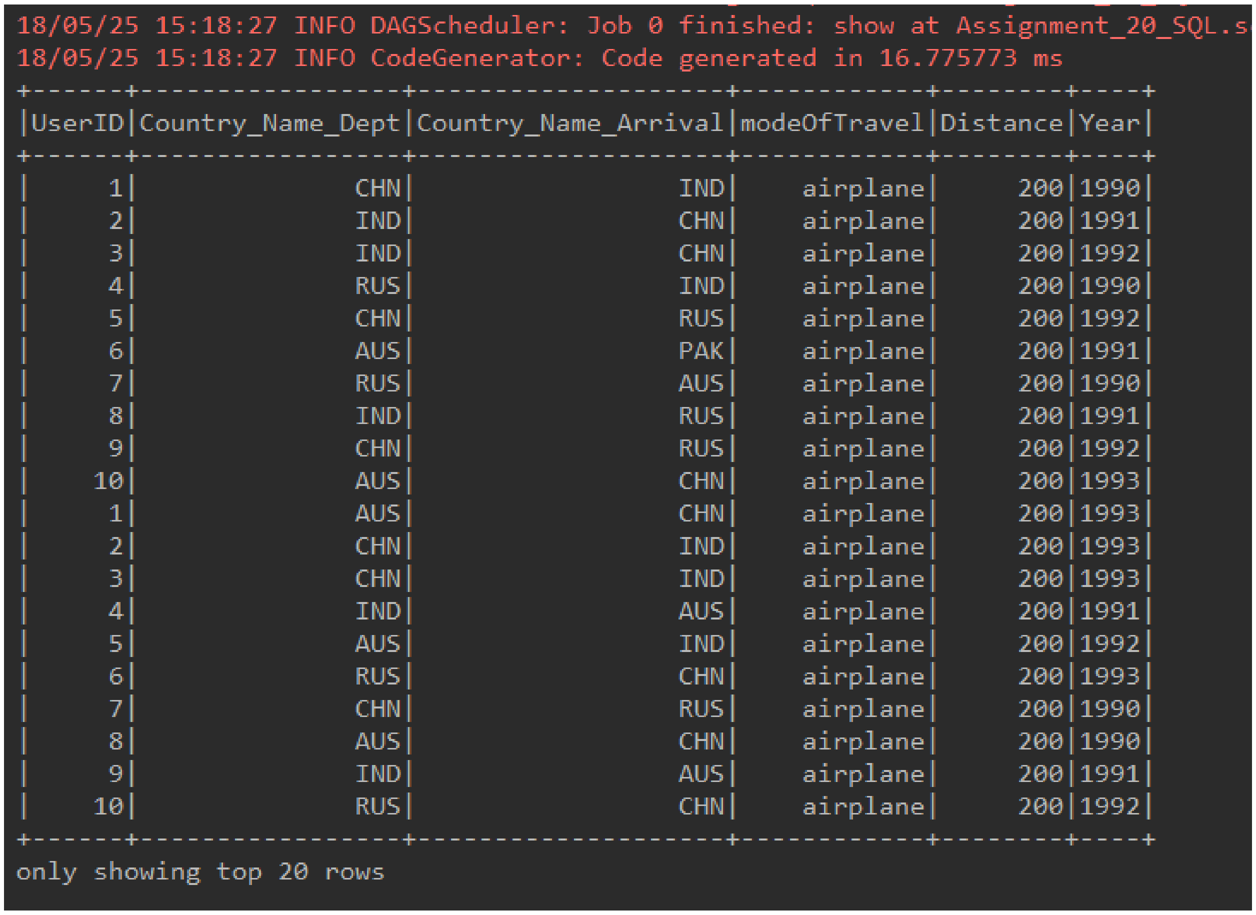
```
//Create USer Details DF by loading the User file
val userDetailsDF = spark.sparkContext.textFile( path = "E:\\Prachi IMP\\Hadoop\\Day 18 Spark\\Day - 20 Spark SQL\\S20_User_Details.txt").
  map(_.split( regex = ",")).map(x=>User_Details(UserID = x(0).toInt,User_Name = x(1),Age = x(2).toInt)).toDF()

//Printing data of Transport Mode DF
userDetailsDF.show()
```

Explanation Of Above Codes :

1. Read the text file from local path by using spark context *textfile()* method which created RDD.
2. Mapped those created RDD to Dataframe by split function (separator is ',') and mapped function and further mapped to case class.
3. While mapping to case class, we can mention the attribute name to be mapped, if we are not mentioning it then we have to maintain the sequence of the same.
4. Data type casting is required if datatype from RDD is not matching what has been defined in case class. e.g. Age, Year , Distance etc.

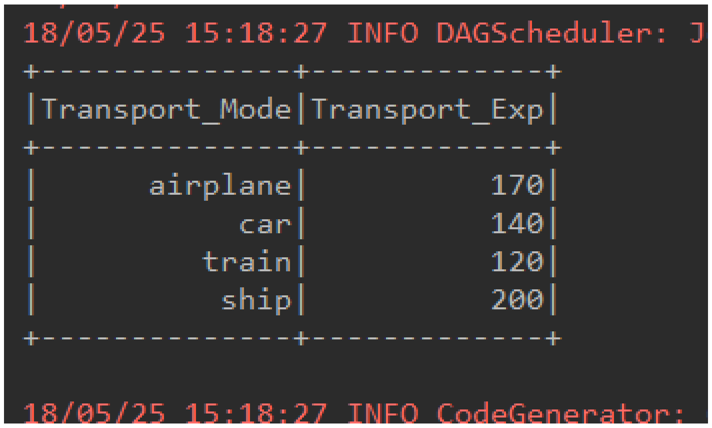
Below is the out of the Dataframes created above

1. Holiday Details



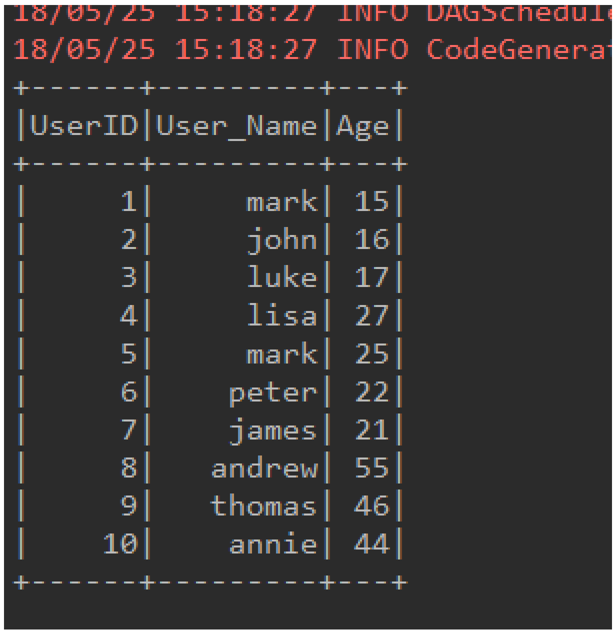
```
18/05/25 15:18:27 INFO DAGScheduler: Job 0 finished: show at Assignment_20_SQL.s
18/05/25 15:18:27 INFO CodeGenerator: Code generated in 16.775773 ms
+------+-----------------+--------------------+------------+--------+----+
|UserID|Country_Name_Dept|Country_Name_Arrival|modeOfTravel|Distance|Year|
+------+-----------------+--------------------+------------+--------+----+
|     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|
+------+-----------------+--------------------+------------+--------+----+
only showing top 20 rows
```

2. Transport Details



```
18/05/25 15:18:27 INFO DAGScheduler: J
+--------------+-------------+
|Transport_Mode|Transport_Exp|
+--------------+-------------+
|      airplane|          170|
|           car|          140|
|         train|          120|
|          ship|          200|
+--------------+-------------+

18/05/25 15:18:27 INFO CodeGenerator:
```

3. User Details



```
18/05/25 15:18:27 INFO DAGSchedule
18/05/25 15:18:27 INFO CodeGenera
+------+---------+---+
|UserID|User_Name|Age|
+------+---------+---+
|     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|
+------+---------+---+
```

Task 1.1 What is the distribution of the total number of air-travelers per year

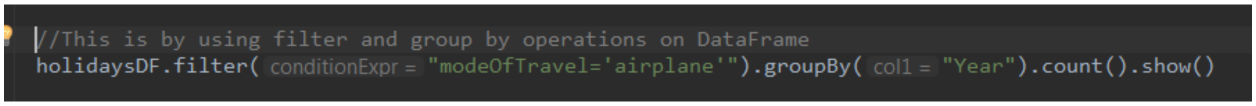
Solution Approach –

1. We have to query holiday details dataframe where mode of travel should be 'airplane' and need to group by 'Year'

This can be achieved by

1. Using SPARK SQL Operations

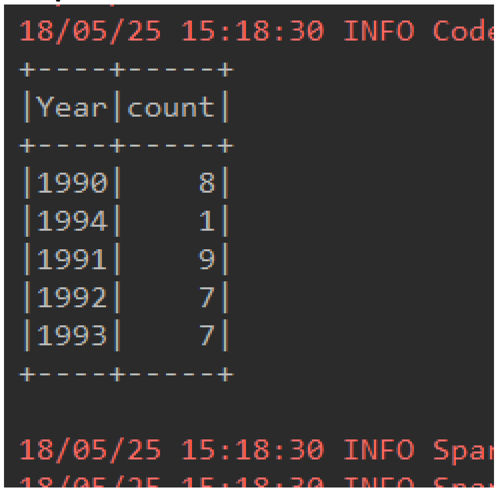
Code



```
//This is by using filter and group by operations on DataFrame
holidaysDF.filter( conditionExpr = "modeOfTravel='airplane'").groupBy( col1 = "Year").count().show()
```

Used filter and groupby operations

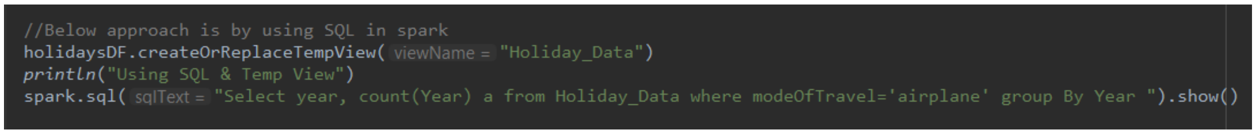
Output



```
18/05/25 15:18:30 INFO Code
+----+-----+
|Year|count|
+----+-----+
|1990|    8|
|1994|    1|
|1991|    9|
|1992|    7|
|1993|    7|
+----+-----+

18/05/25 15:18:30 INFO Spar
```

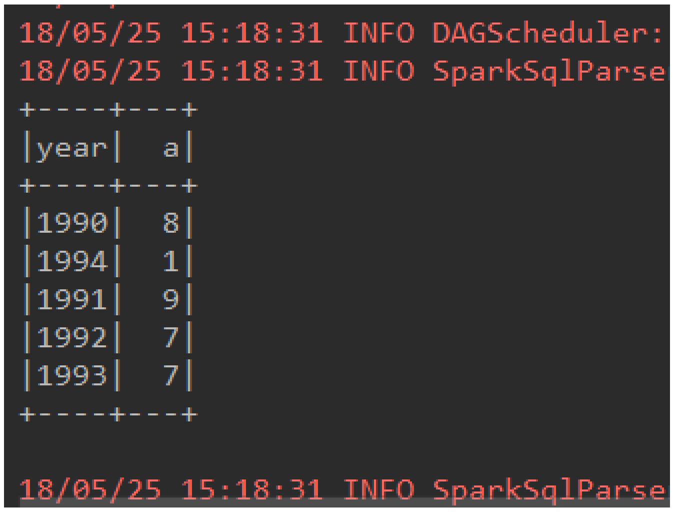
2. Using SQL queries in spark.sqp operation
    a. We have create view for this dataframe which can be achieved by ‘createOrReplaceTempView ‘
    b. Used spark object to create sql query as below.

Code



```
//Below approach is by using SQL in spark
holidaysDF.createOrReplaceTempView( viewName = "Holiday_Data")
println("Using SQL & Temp View")
spark.sql( sqlText = "Select year, count(Year) a from Holiday_Data where modeOfTravel='airplane' group By Year ").show()
```

Output



```
18/05/25 15:18:31 INFO DAGScheduler:
18/05/25 15:18:31 INFO SparkSqlParse
+----+---+
|year|  a|
+----+---+
|1990|  8|
|1994|  1|
|1991|  9|
|1992|  7|
|1993|  7|
+----+---+

18/05/25 15:18:31 INFO SparkSqlParse
```

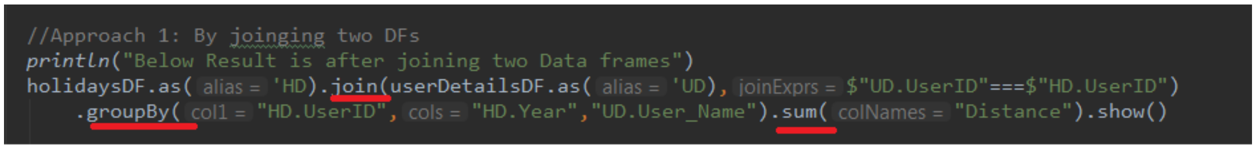
Task 1.2 What is the total air distance covered by each user per year

Solution Approach –

1. We need group on holiday details on user id and year to get total sum of air distance covered
2. Here we have to join two DFs to get user name as well

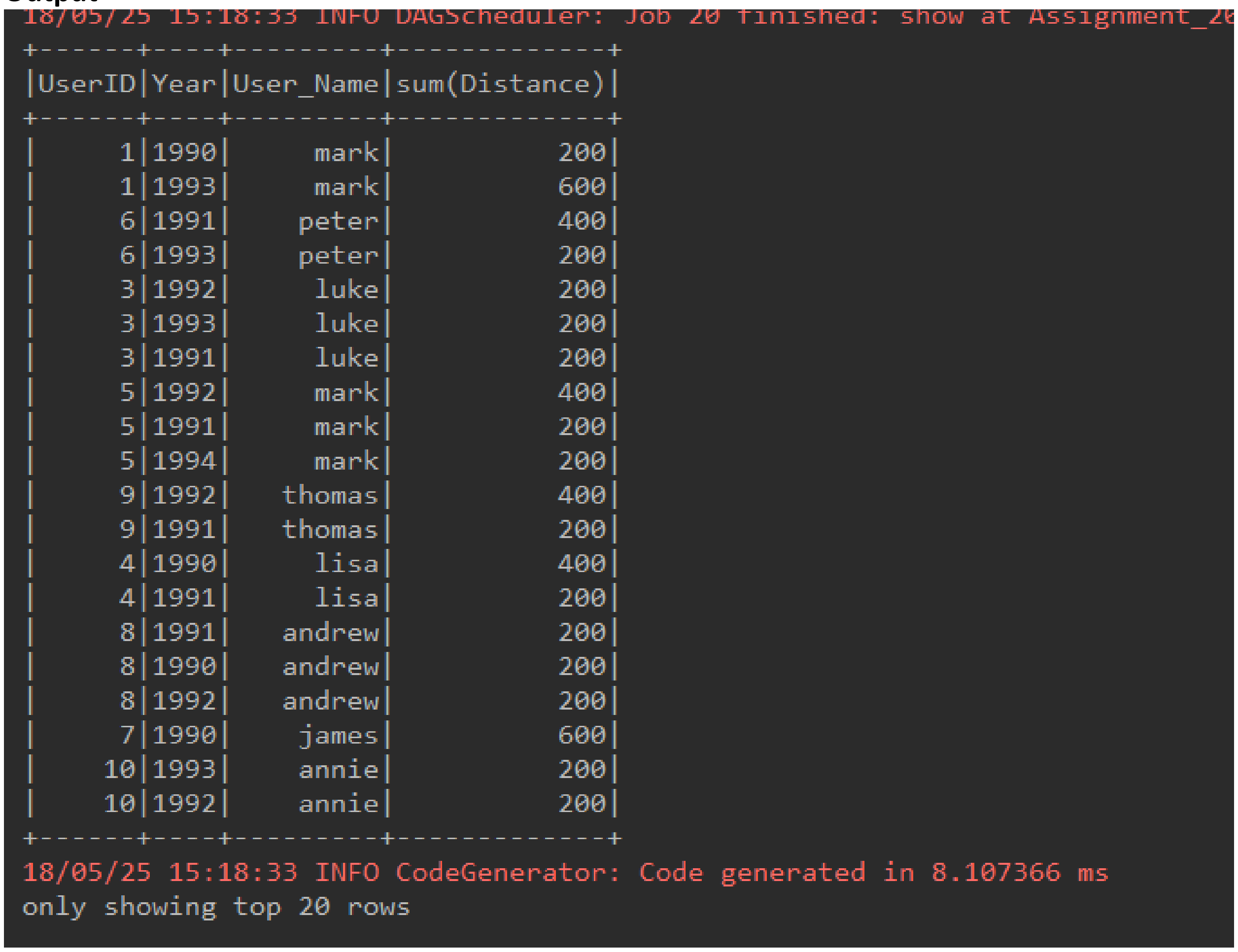
Approach 1 : Using SPARK SQL Operations → GroupBy , sum and join for joining two DFs

Code



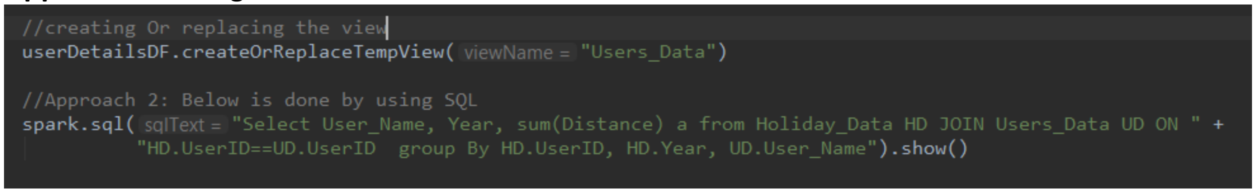
```
//Approach 1: By joinging two DFs
println("Below Result is after joining two Data frames")
holidaysDF.as( alias = 'HD).join(userDetailsDF.as( alias = 'UD), joinExprs = $"UD.UserID"===$"HD.UserID")
    .groupBy( col1 = "HD.UserID", cols = "HD.Year","UD.User_Name").sum( colNames = "Distance").show()
```

Output



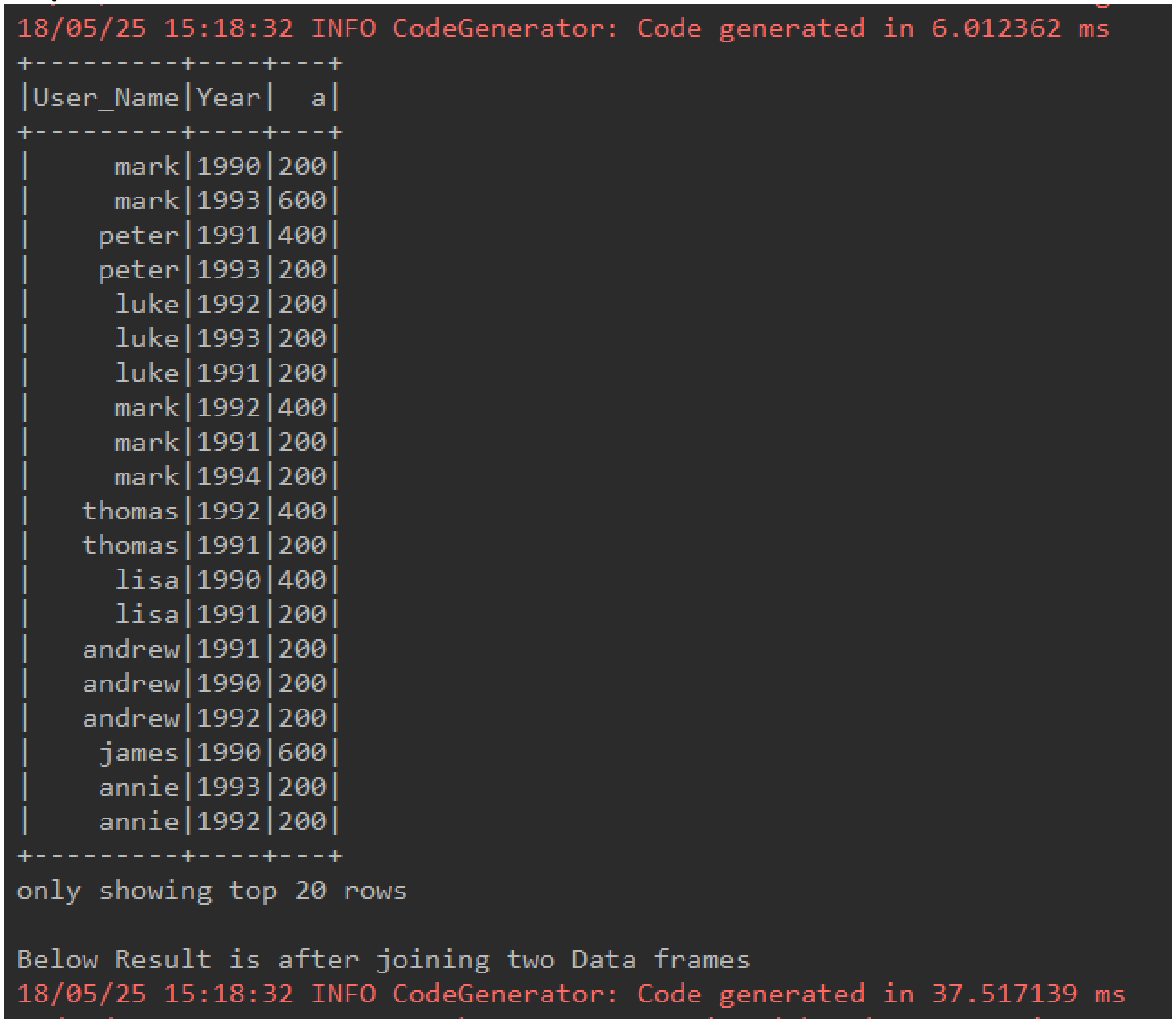
```
18/05/25 15:18:33 INFO DAGScheduler: Job 20 finished: show at Assignment_20
+------+----+---------+-------------+
|UserID|Year|User_Name|sum(Distance)|
+------+----+---------+-------------+
|     1|1990|     mark|          200|
|     1|1993|     mark|          600|
|     6|1991|    peter|          400|
|     6|1993|    peter|          200|
|     3|1992|     luke|          200|
|     3|1993|     luke|          200|
|     3|1991|     luke|          200|
|     5|1992|     mark|          400|
|     5|1991|     mark|          200|
|     5|1994|     mark|          200|
|     9|1992|   thomas|          400|
|     9|1991|   thomas|          200|
|     4|1990|     lisa|          400|
|     4|1991|     lisa|          200|
|     8|1991|   andrew|          200|
|     8|1990|   andrew|          200|
|     8|1992|   andrew|          200|
|     7|1990|    james|          600|
|    10|1993|    annie|          200|
|    10|1992|    annie|          200|
+------+----+---------+-------------+
18/05/25 15:18:33 INFO CodeGenerator: Code generated in 8.107366 ms
only showing top 20 rows
```

Approach 2: Using SQL Queries



```
//creating Or replacing the view
userDetailsDF.createOrReplaceTempView( viewName = "Users_Data")

//Approach 2: Below is done by using SQL
spark.sql( sqlText = "Select User_Name, Year, sum(Distance) a from Holiday_Data HD JOIN Users_Data UD ON " +
          "HD.UserID==UD.UserID  group By HD.UserID, HD.Year, UD.User_Name").show()
```

Output



```
18/05/25 15:18:32 INFO CodeGenerator: Code generated in 6.012362 ms
+---------+----+---+
|User_Name|Year|  a|
+---------+----+---+
|     mark|1990|200|
|     mark|1993|600|
|    peter|1991|400|
|    peter|1993|200|
|     luke|1992|200|
|     luke|1993|200|
|     luke|1991|200|
|     mark|1992|400|
|     mark|1991|200|
|     mark|1994|200|
|   thomas|1992|400|
|   thomas|1991|200|
|     lisa|1990|400|
|     lisa|1991|200|
|   andrew|1991|200|
|   andrew|1990|200|
|   andrew|1992|200|
|    james|1990|600|
|    annie|1993|200|
|    annie|1992|200|
+---------+----+---+
only showing top 20 rows

Below Result is after joining two Data frames
18/05/25 15:18:32 INFO CodeGenerator: Code generated in 37.517139 ms
```

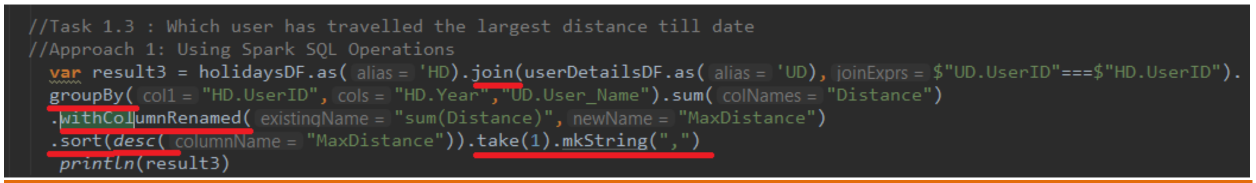
Task 1.3 Which user has travelled the largest distance till date

Solution Approach –

1. We need group on holiday details on user id and year to get total sum of air distance covered
2. Here we have to join two DFs to get user name as well
3. Then we have order above data in descending order to take first row as max distance travelled

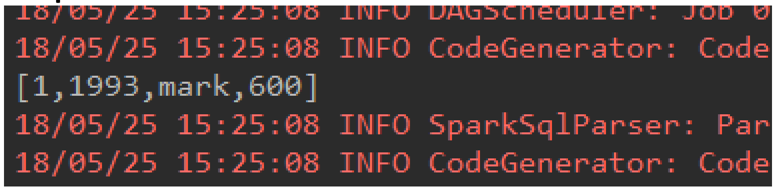
Approach 1 : Using SPARK SQL Operations → GroupBy , sum , join, withCoulmnRenamed, ort, take for joining two DFs
withCoulmnRenamed → this is to rename the any existing column. In this case we are renaming the column generated after using sum function.
Sort→ this sorts the result Dataframe on given column and given order.
Take→ this takes the no. of rows from dataframe as mentioned in parameter

Code



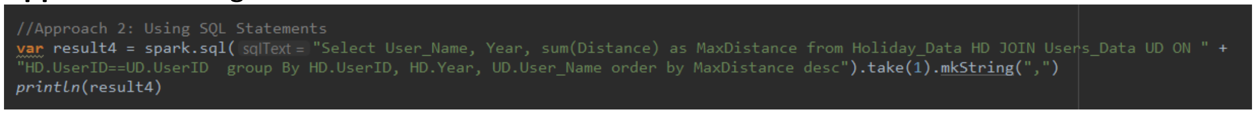
```
//Task 1.3 : Which user has travelled the largest distance till date
//Approach 1: Using Spark SQL Operations
  var result3 = holidaysDF.as( alias = 'HD).join(userDetailsDF.as( alias = 'UD), joinExprs = $"UD.UserID"===$"HD.UserID").
  groupBy( col1 = "HD.UserID", cols = "HD.Year","UD.User_Name").sum( colNames = "Distance")
  .withColumnRenamed( existingName = "sum(Distance)", newName = "MaxDistance")
  .sort(desc( columnName = "MaxDistance")).take(1).mkString(",")
   println(result3)
```

Output



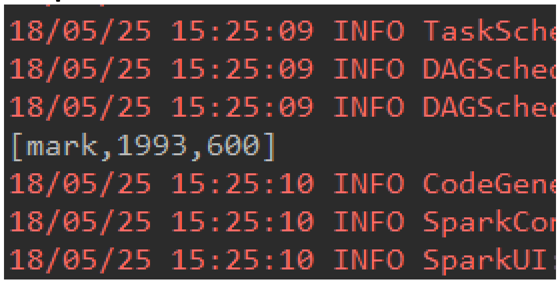
```
18/05/25 15:25:08 INFO DAGScheduler: Job 0
18/05/25 15:25:08 INFO CodeGenerator: Code
[1,1993,mark,600]
18/05/25 15:25:08 INFO SparkSqlParser: Par
18/05/25 15:25:08 INFO CodeGenerator: Code
```

Approach 2: Using SQL Queries



```
//Approach 2: Using SQL Statements
var result4 = spark.sql( sqlText = "Select User_Name, Year, sum(Distance) as MaxDistance from Holiday_Data HD JOIN Users_Data UD ON " +
"HD.UserID==UD.UserID  group By HD.UserID, HD.Year, UD.User_Name order by MaxDistance desc").take(1).mkString(",")
println(result4)
```

Output



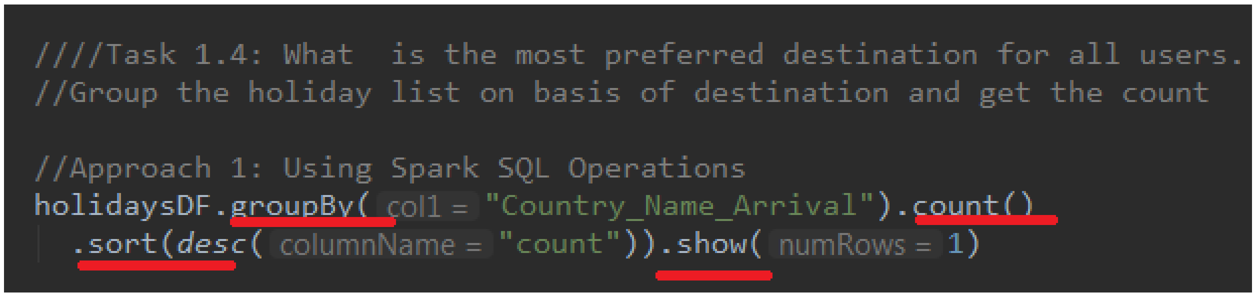
```
18/05/25 15:25:09 INFO TaskSche
18/05/25 15:25:09 INFO DAGSched
18/05/25 15:25:09 INFO DAGSched
[mark,1993,600]
18/05/25 15:25:10 INFO CodeGene
18/05/25 15:25:10 INFO SparkCon
18/05/25 15:25:10 INFO SparkUI:
```

Task 1.4 What is the most preferred destination for all users.

Solution Approach –

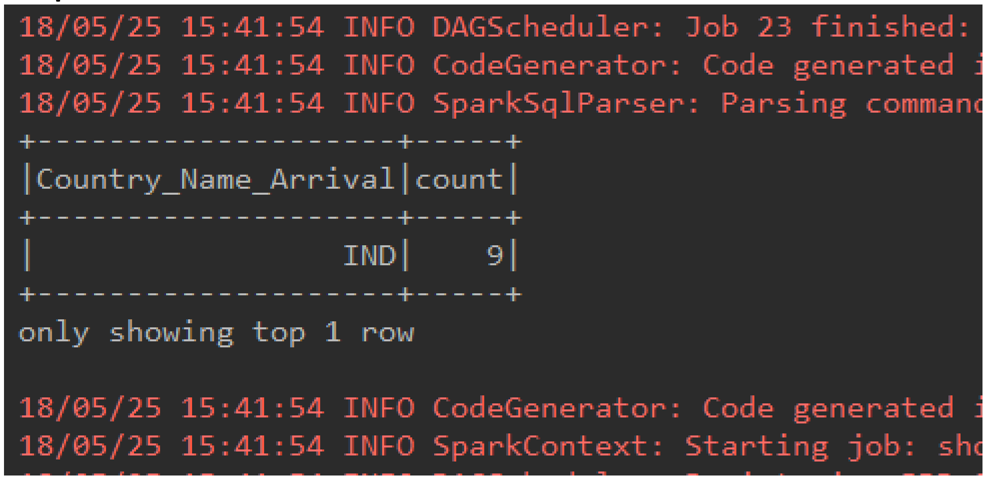
1. We have to group holiday details on Country of Arrival and take count of the same.
2. Order above dataframe in descending order and get the top most row which has maximum count and so the preferred localtion

Approach 1 : Using SPARK SQL Operations → GroupBy ,count,sort,show
Show(number)→is used to display top 'n' rows.



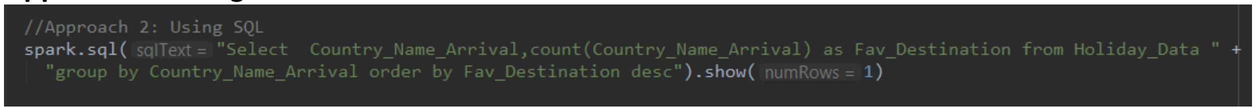
```
////Task 1.4: What  is the most preferred destination for all users.
//Group the holiday list on basis of destination and get the count

//Approach 1: Using Spark SQL Operations
holidaysDF.groupBy( col1 = "Country_Name_Arrival").count()
  .sort(desc( columnName = "count")).show( numRows = 1)
```

Output



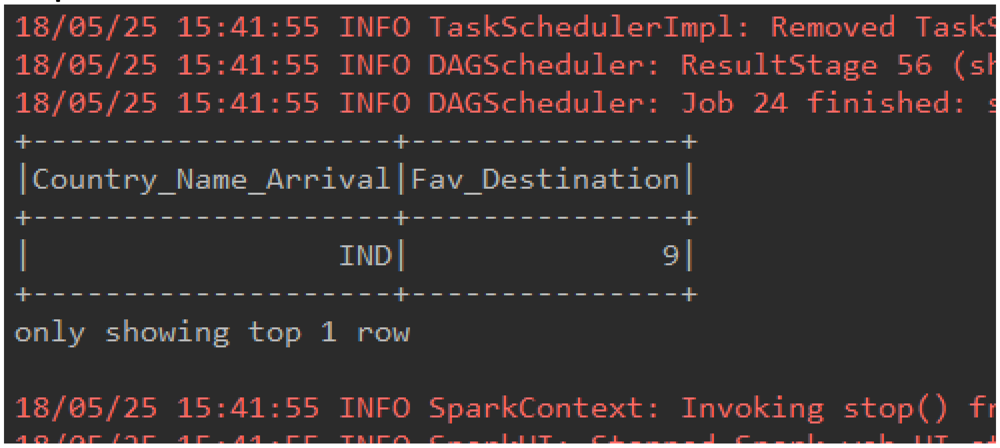
```
18/05/25 15:41:54 INFO DAGScheduler: Job 23 finished:
18/05/25 15:41:54 INFO CodeGenerator: Code generated i
18/05/25 15:41:54 INFO SparkSqlParser: Parsing command
+--------------------+-----+
|Country_Name_Arrival|count|
+--------------------+-----+
|                 IND|    9|
+--------------------+-----+
only showing top 1 row

18/05/25 15:41:54 INFO CodeGenerator: Code generated i
18/05/25 15:41:54 INFO SparkContext: Starting job: sho
```

Approach 2: Using SQL Queries



```
//Approach 2: Using SQL
spark.sql( sqlText = "Select  Country_Name_Arrival,count(Country_Name_Arrival) as Fav_Destination from Holiday_Data " +
  "group by Country_Name_Arrival order by Fav_Destination desc").show( numRows = 1)
```

Output



```
18/05/25 15:41:55 INFO TaskSchedulerImpl: Removed TaskS
18/05/25 15:41:55 INFO DAGScheduler: ResultStage 56 (sh
18/05/25 15:41:55 INFO DAGScheduler: Job 24 finished: s
+--------------------+---------------+
|Country_Name_Arrival|Fav_Destination|
+--------------------+---------------+
|                 IND|              9|
+--------------------+---------------+
only showing top 1 row

18/05/25 15:41:55 INFO SparkContext: Invoking stop() fr
18/05/25 15:41:55 INFO SparkUI: Stopped Spark web UI at
```

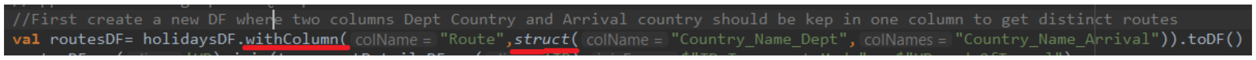
Task 1. 5 Which route is generating the most revenue per year

Solution Approach –

1. In this case we have different columns for Country Arrival and Country Departure. So group on route we must combine these in one column and group on that column.
2. Get the count of above grouping per year
3. To get expenses join to Transport Details on Mode Of Travel

Created new DF with new column using withColumn and combined two columns using struct

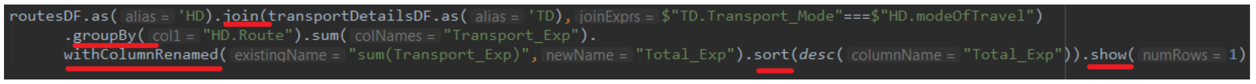
Code



```
//First create a new DF where two columns Dept Country and Arrival country should be kep in one column to get distinct routes
val routesDF= holidaysDF.withColumn( colName = "Route",struct( colName = "Country_Name_Dept", colNames = "Country_Name_Arrival")).toDF()
```

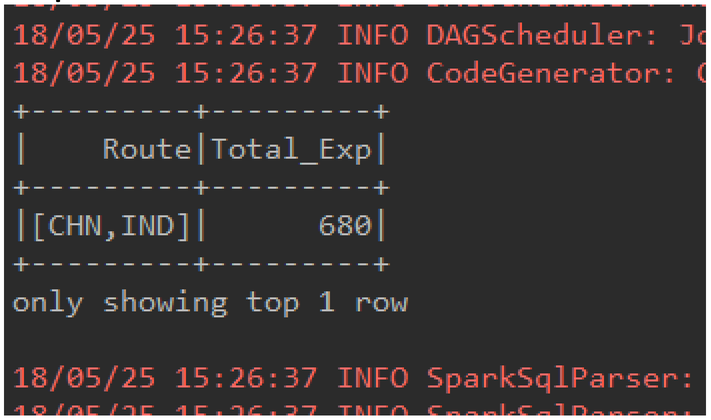
Approach 1 : Using SPARK SQL Operations →join , groupby,WithColumnRenamed,sort,show

Code



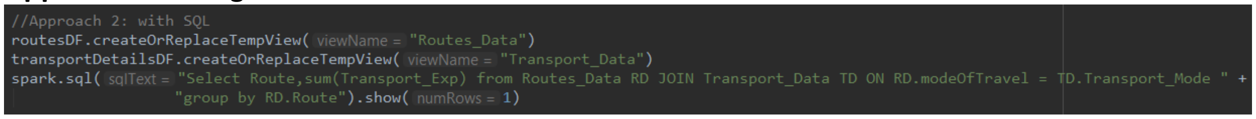
```
routesDF.as( alias = 'HD).join(transportDetailsDF.as( alias = 'TD), joinExprs = $"TD.Transport_Mode"===$"HD.modeOfTravel")
      .groupBy( col1 = "HD.Route").sum( colNames = "Transport_Exp").
      withColumnRenamed( existingName = "sum(Transport_Exp)", newName = "Total_Exp").sort(desc( columnName = "Total_Exp")).show( numRows = 1)
```

Output



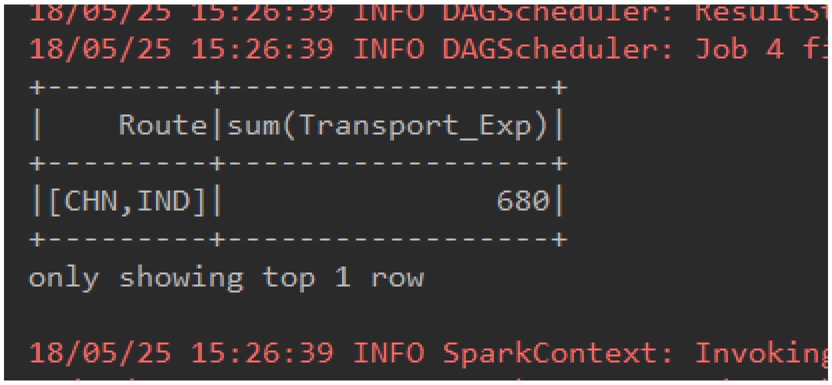
```
18/05/25 15:26:37 INFO DAGScheduler: Jo
18/05/25 15:26:37 INFO CodeGenerator: C
+---------+---------+
|    Route|Total_Exp|
+---------+---------+
|[CHN,IND]|      680|
+---------+---------+
only showing top 1 row

18/05/25 15:26:37 INFO SparkSqlParser:
```

Approach 2: Using SQL Queries



```
//Approach 2: with SQL
routesDF.createOrReplaceTempView( viewName = "Routes_Data")
transportDetailsDF.createOrReplaceTempView( viewName = "Transport_Data")
spark.sql( sqlText = "Select Route,sum(Transport_Exp) from Routes_Data RD JOIN Transport_Data TD ON RD.modeOfTravel = TD.Transport_Mode " +
                "group by RD.Route").show( numRows = 1)
```

Output



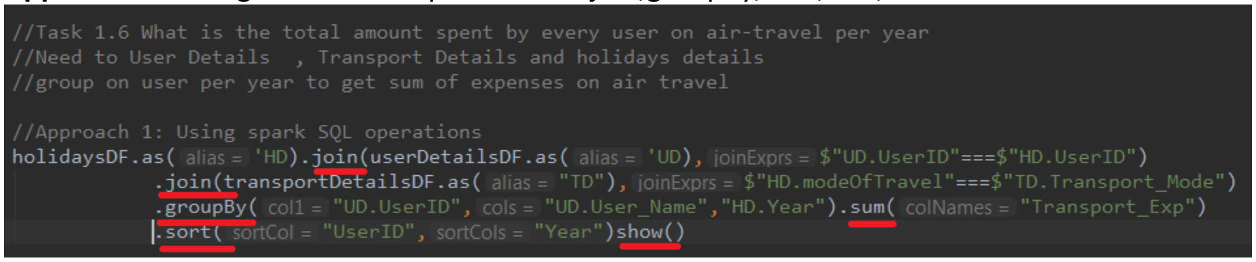
```
18/05/25 15:26:39 INFO DAGScheduler: ResultS
18/05/25 15:26:39 INFO DAGScheduler: Job 4 f
+---------+------------------+
|    Route|sum(Transport_Exp)|
+---------+------------------+
|[CHN,IND]|               680|
+---------+------------------+
only showing top 1 row

18/05/25 15:26:39 INFO SparkContext: Invokin
```

Task 1. 6 What is the total amount spent by every user on air-travel per year

Solution Approach –

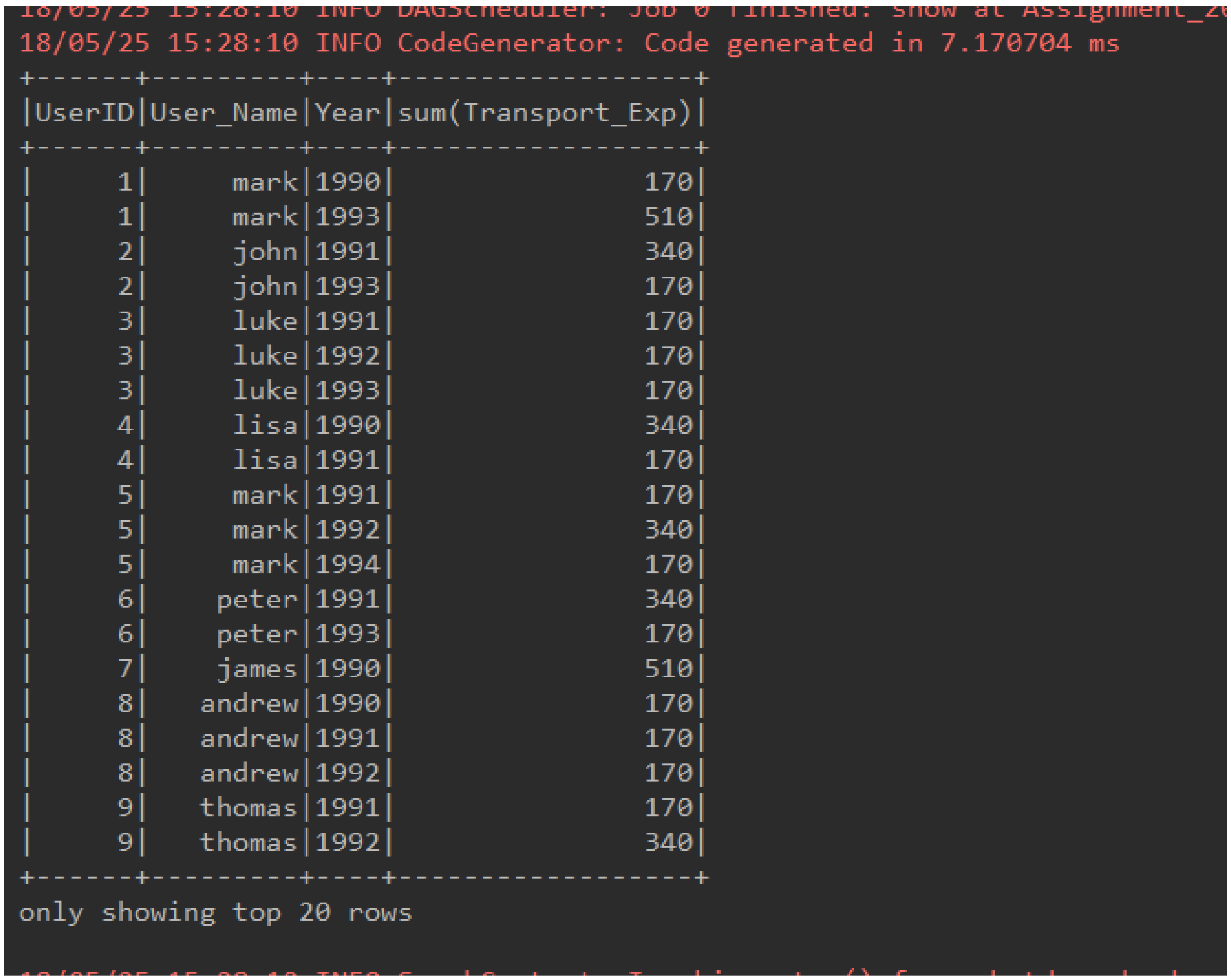
1. Join all the dataframes and group on user and year and get the count of air travel
2. Sum the travel expendes from Transport details and get amount spent every year.

Approach 1 : Using SPARK SQL Operations →join,groupby,sum,sort,show



```
//Task 1.6 What is the total amount spent by every user on air-travel per year
//Need to User Details  , Transport Details and holidays details
//group on user per year to get sum of expenses on air travel

//Approach 1: Using spark SQL operations
holidaysDF.as( alias = 'HD).join(userDetailsDF.as( alias = 'UD), joinExprs = $"UD.UserID"===$"HD.UserID")
            .join(transportDetailsDF.as( alias = "TD"), joinExprs = $"HD.modeOfTravel"===$"TD.Transport_Mode")
            .groupBy( col1 = "UD.UserID", cols = "UD.User_Name","HD.Year").sum( colNames = "Transport_Exp")
            .sort( sortCol = "UserID", sortCols = "Year")show()
```

Output



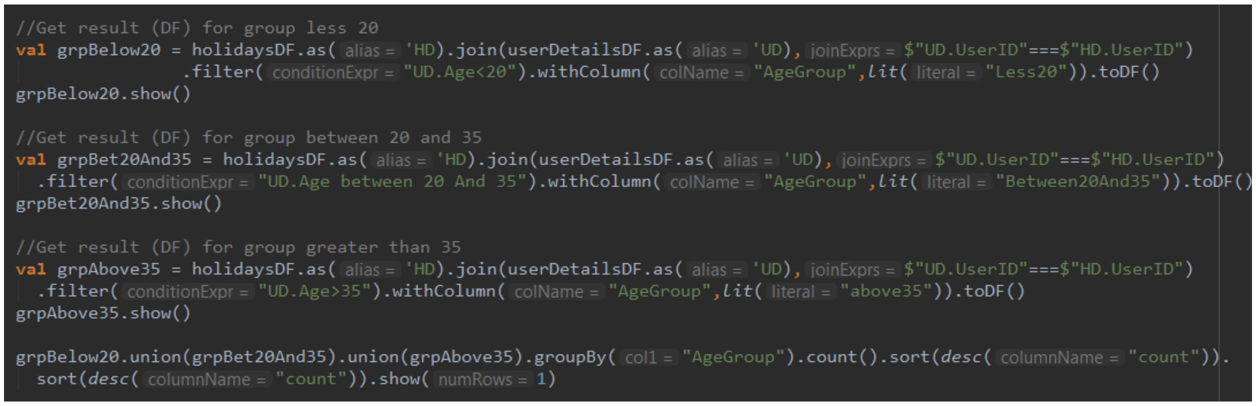
```
18/05/25 15:28:10 INFO CodeGenerator: Code generated in 7.170704 ms
+------+---------+----+------------------+
|UserID|User_Name|Year|sum(Transport_Exp)|
+------+---------+----+------------------+
|     1|     mark|1990|               170|
|     1|     mark|1993|               510|
|     2|     john|1991|               340|
|     2|     john|1993|               170|
|     3|     luke|1991|               170|
|     3|     luke|1992|               170|
|     3|     luke|1993|               170|
|     4|     lisa|1990|               340|
|     4|     lisa|1991|               170|
|     5|     mark|1991|               170|
|     5|     mark|1992|               340|
|     5|     mark|1994|               170|
|     6|    peter|1991|               340|
|     6|    peter|1993|               170|
|     7|    james|1990|               510|
|     8|   andrew|1990|               170|
|     8|   andrew|1991|               170|
|     8|   andrew|1992|               170|
|     9|   thomas|1991|               170|
|     9|   thomas|1992|               340|
+------+---------+----+------------------+
only showing top 20 rows
```

Task 1. 7 Considering age groups of < 20 , 20-35, 35 > ,Which age group is travelling the most every year.

Solution Approach –

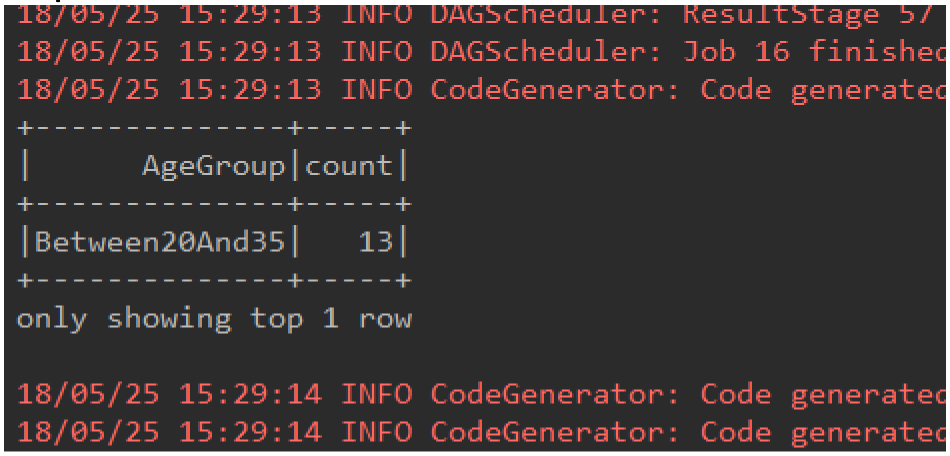
1. We have to join user details and holiday details to get age and count of travel based / grouped on user age group.
2. However we don't have any column which has groups categorized inside it. There are two approaches to resolve this

Approach 1:
Write three different queries based on age criteria as mentioned in given task and get the **three different dataframes with newly added column as 'AgeGroup' and union those datasets and then do grouping on newly added column 'AgeGroup'** and get the desired result
Code



```
//Get result (DF) for group less 20
val grpBelow20 = holidaysDF.as( alias = 'HD).join(userDetailsDF.as( alias = 'UD), joinExprs = $"UD.UserID"===$"HD.UserID")
                .filter( conditionExpr = "UD.Age<20").withColumn( colName = "AgeGroup",lit( literal = "Less20")).toDF()
grpBelow20.show()

//Get result (DF) for group between 20 and 35
val grpBet20And35 = holidaysDF.as( alias = 'HD).join(userDetailsDF.as( alias = 'UD), joinExprs = $"UD.UserID"===$"HD.UserID")
  .filter( conditionExpr = "UD.Age between 20 And 35").withColumn( colName = "AgeGroup",lit( literal = "Between20And35")).toDF()
grpBet20And35.show()

//Get result (DF) for group greater than 35
val grpAbove35 = holidaysDF.as( alias = 'HD).join(userDetailsDF.as( alias = 'UD), joinExprs = $"UD.UserID"===$"HD.UserID")
  .filter( conditionExpr = "UD.Age>35").withColumn( colName = "AgeGroup",lit( literal = "above35")).toDF()
grpAbove35.show()

grpBelow20.union(grpBet20And35).union(grpAbove35).groupBy( col1 = "AgeGroup").count().sort(desc( columnName = "count")).
  sort(desc( columnName = "count")).show( numRows = 1)
```

Output

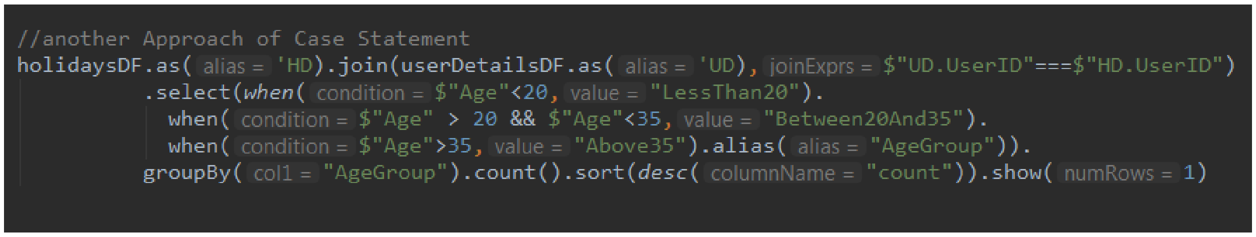


```
18/05/25 15:29:13 INFO DAGScheduler: ResultStage 57
18/05/25 15:29:13 INFO DAGScheduler: Job 16 finishec
18/05/25 15:29:13 INFO CodeGenerator: Code generatec
+--------------+-----+
|      AgeGroup|count|
+--------------+-----+
|Between20And35|   13|
+--------------+-----+
only showing top 1 row

18/05/25 15:29:14 INFO CodeGenerator: Code generatec
18/05/25 15:29:14 INFO CodeGenerator: Code generatec
```

Approach 2:

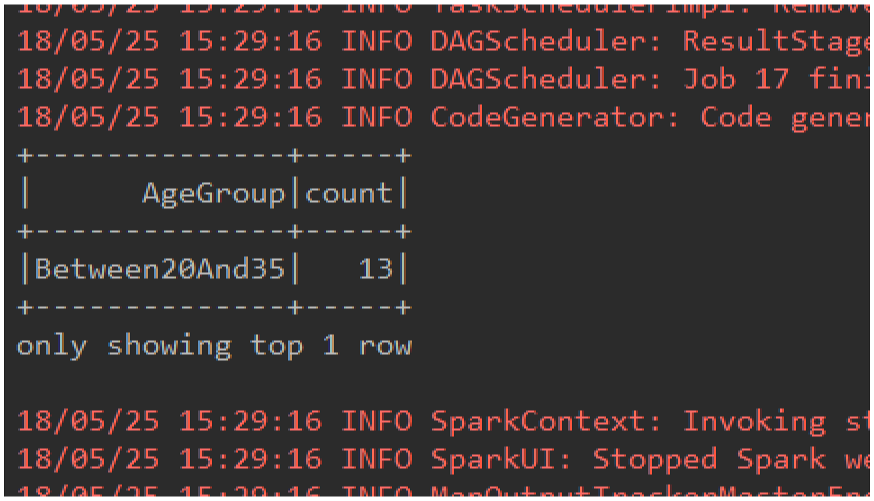
Another approach is write case statement to add new column in existing Dataframe and do the grouping on newly added column.

Code



```
//another Approach of Case Statement
holidaysDF.as( alias = 'HD).join(userDetailsDF.as( alias = 'UD), joinExprs = $"UD.UserID"===$"HD.UserID")
          .select(when( condition = $"Age"<20, value = "LessThan20").
           when( condition = $"Age" > 20 && $"Age"<35, value = "Between20And35").
           when( condition = $"Age">35, value = "Above35").alias( alias = "AgeGroup")).
          groupBy( col1 = "AgeGroup").count().sort(desc( columnName = "count")).show( numRows = 1)
```

Output – Age group between 20 and 35 travels most



```
18/05/25 15:29:16 INFO DAGScheduler: ResultStage
18/05/25 15:29:16 INFO DAGScheduler: Job 17 fin
18/05/25 15:29:16 INFO CodeGenerator: Code gene
+-------------+-----+
|     AgeGroup|count|
+-------------+-----+
|Between20And35|   13|
+-------------+-----+
only showing top 1 row

18/05/25 15:29:16 INFO SparkContext: Invoking s
18/05/25 15:29:16 INFO SparkUI: Stopped Spark w
```