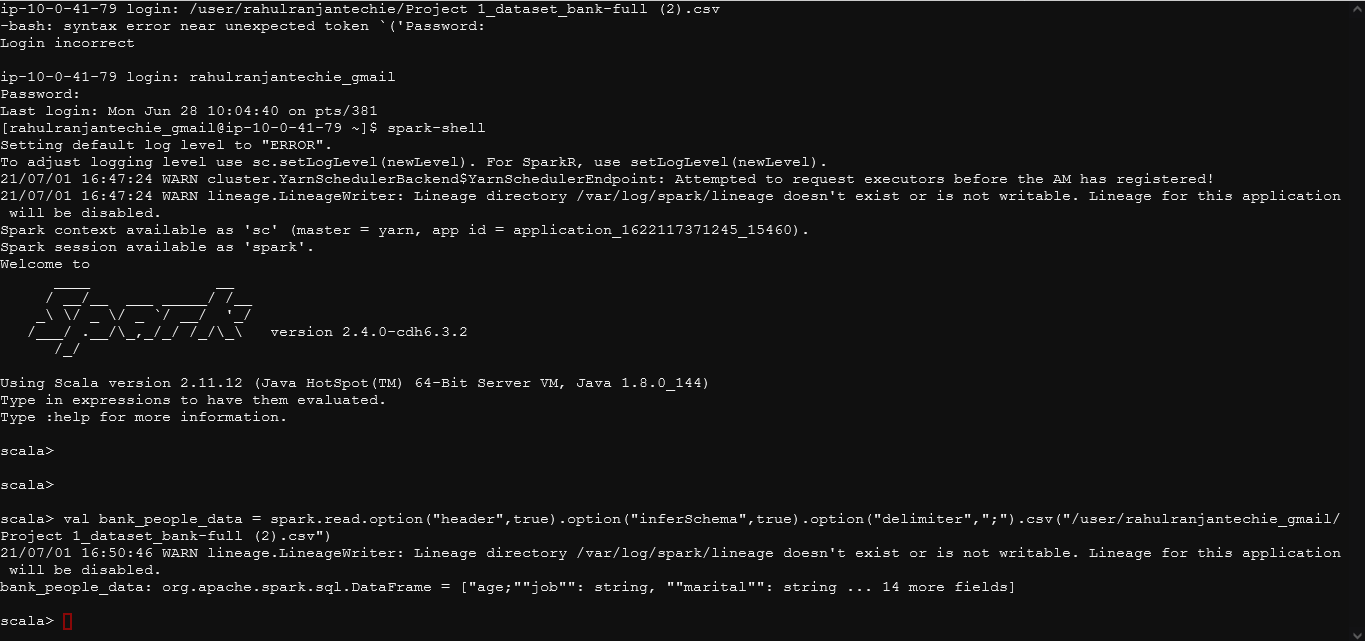
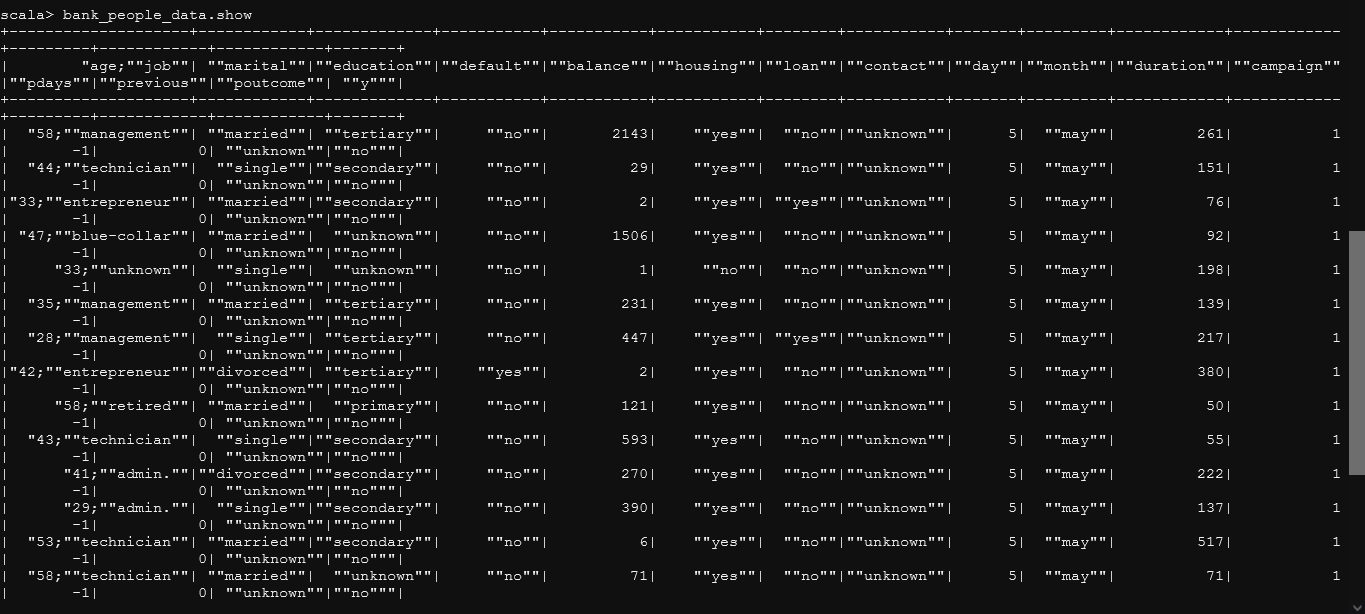
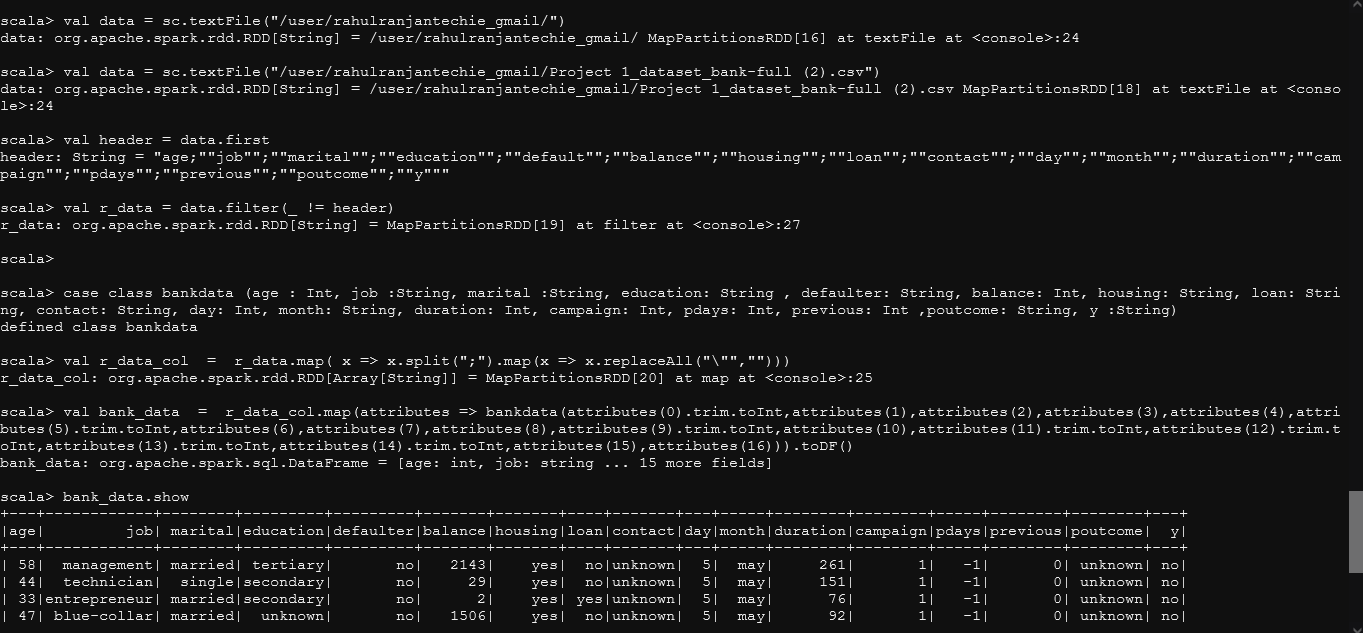
**PROJECT 3: MARKET ANALYSIS IN BANKING DOMAIN**





Here I tried to create a DataFrame directly using DataFrame API with the data provided but due to improper format proper cleaning was required to load the data into DataFrame.

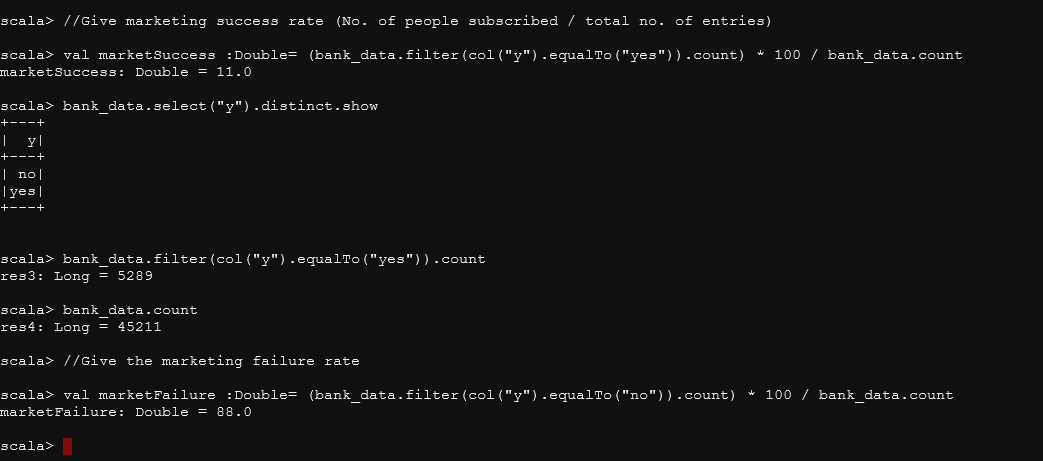
Thus in the further steps I am loading data in the RDD so that cleaning the data and creating DataFrame from it becomes easy, so that further analysis could be carried out smoothly.

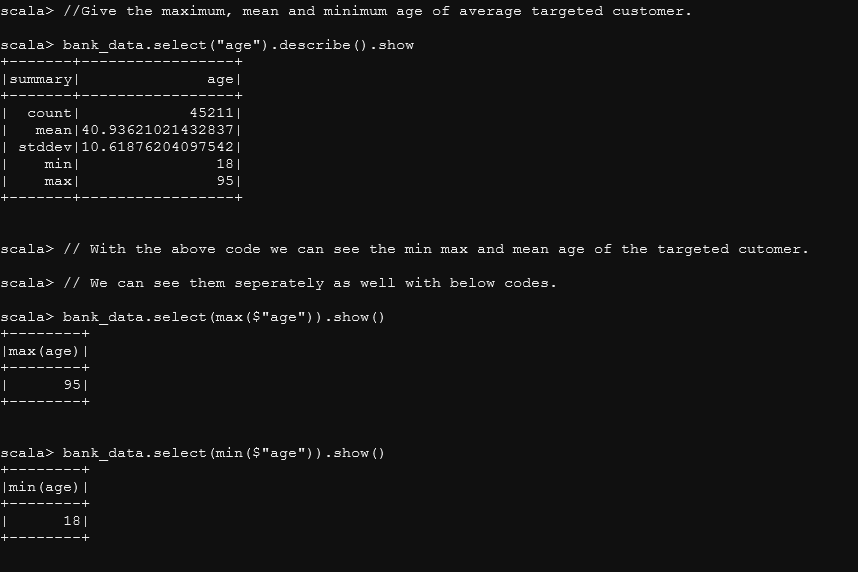


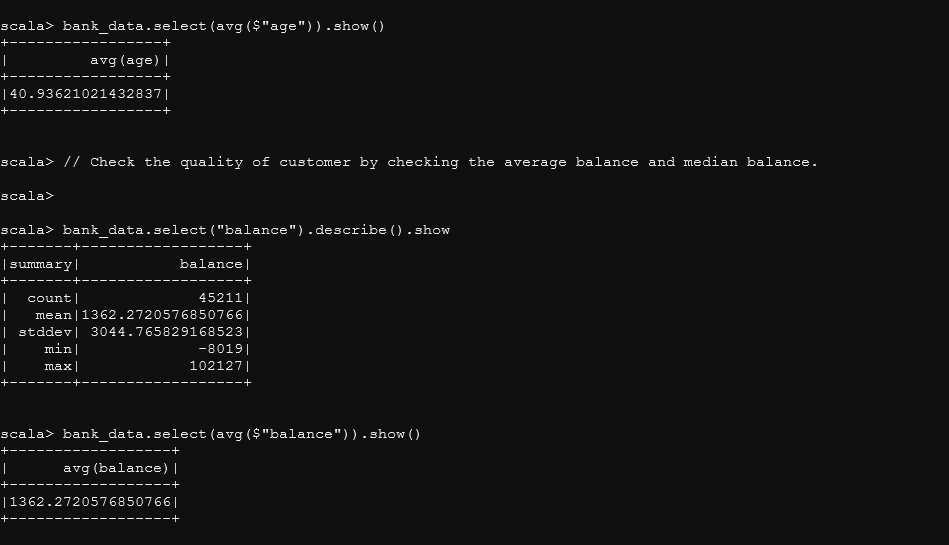
Now in the above table we can see the data in proper tabular format as in DataFrames.

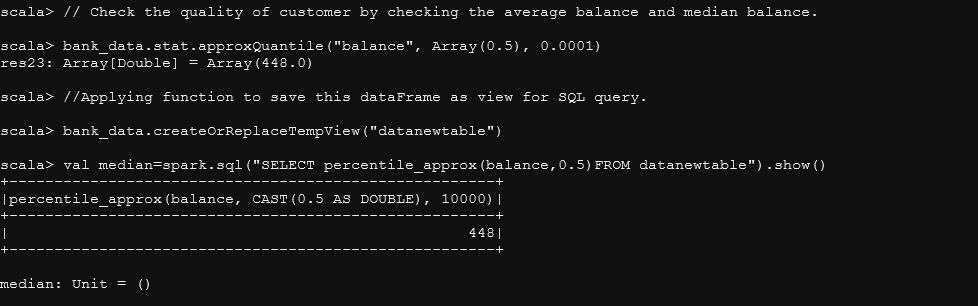
Solving all the problem statement as demanded in the project.

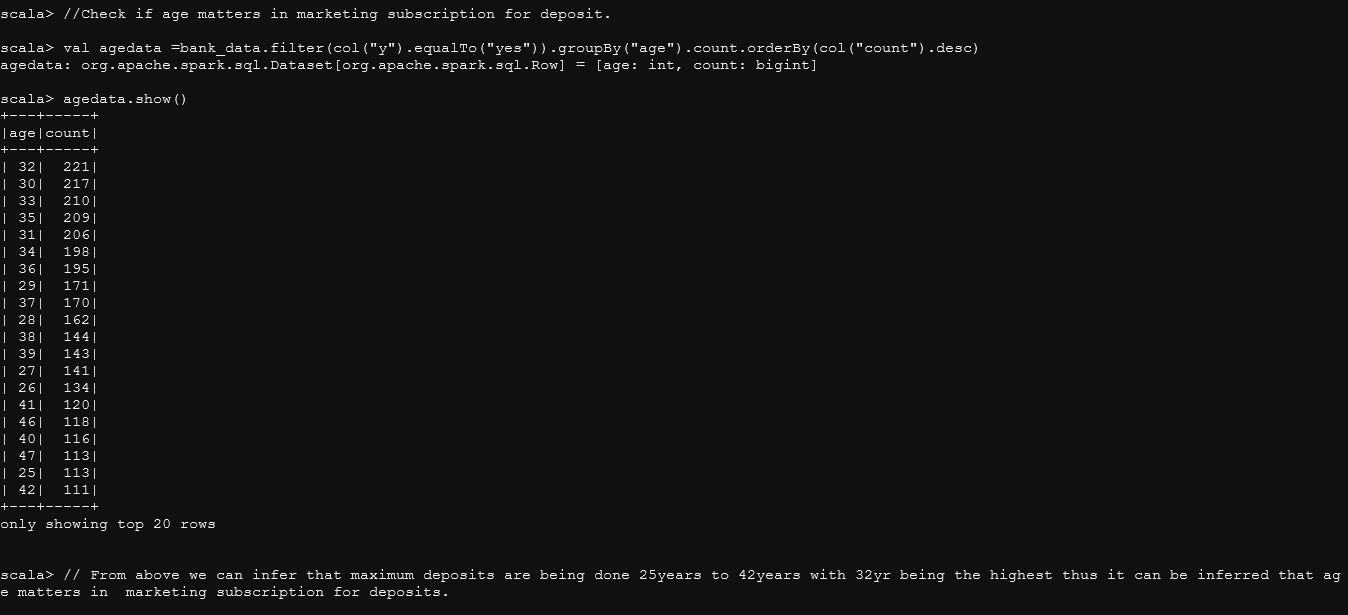
From here onwards all the problem statement, comments, inference and insights has been written/ commented out in console itself below and above the code output.

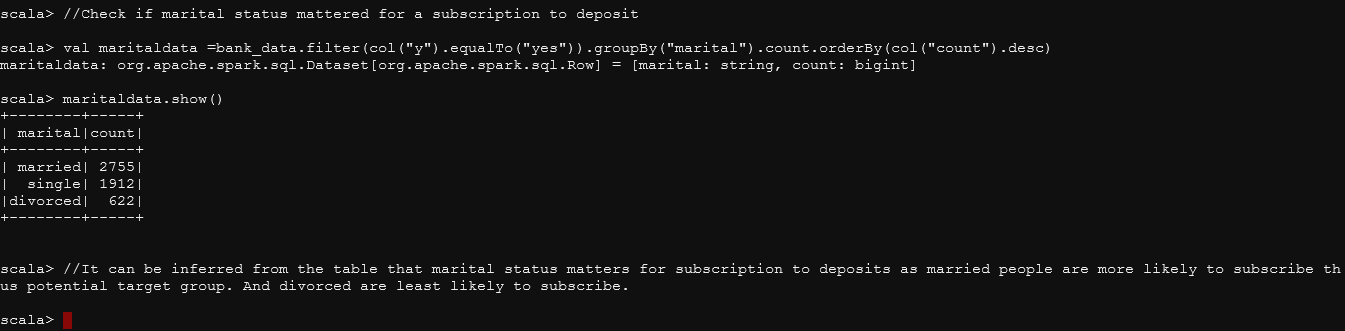


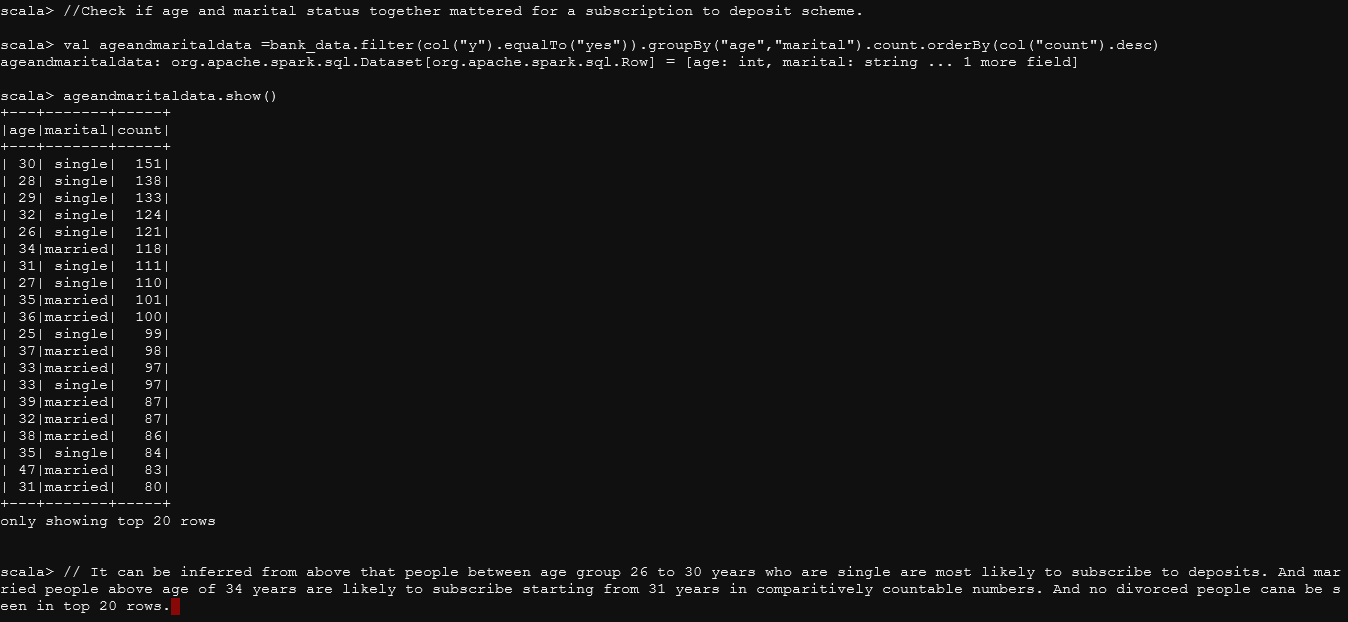


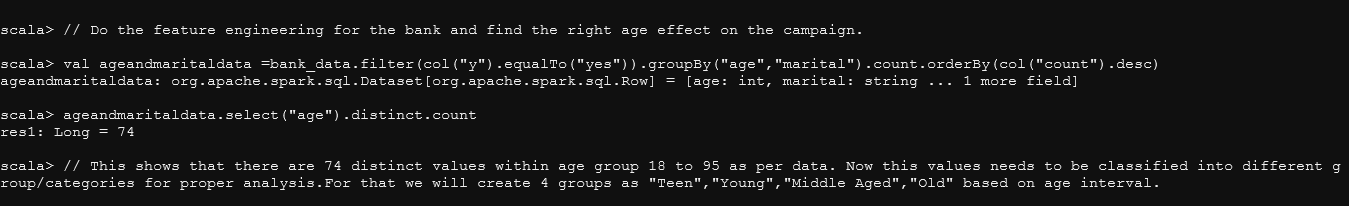


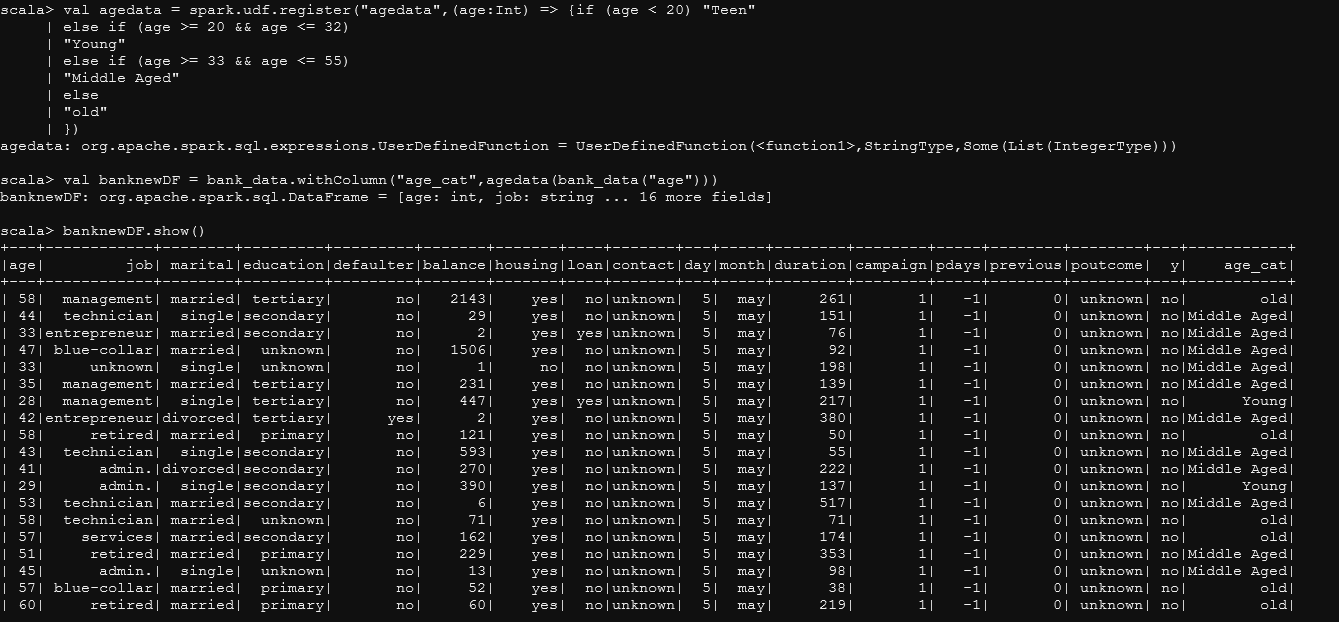


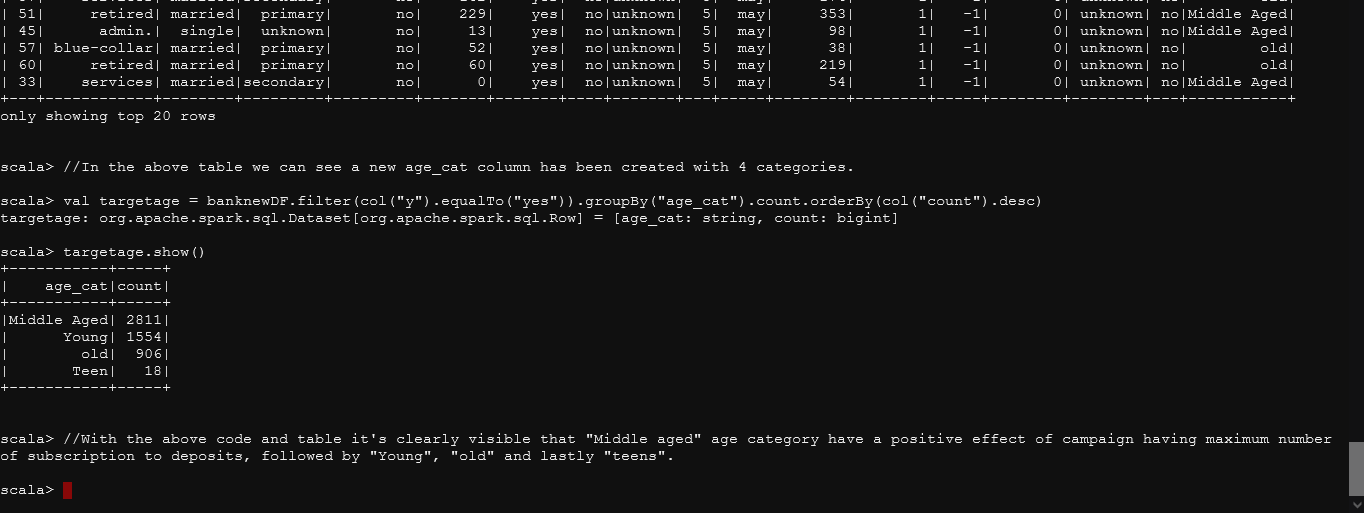












All the problem statement has been solved successfully with screenshot of all codes and their outputs attached with inference.

**SOURCE CODE**

val bank\_people\_data = spark.read.option("header",true).option("inferSchema",true).option("delimiter",";").csv("/user/rahulranjantechie\_gmail/

Project 1\_dataset\_bank-full (2).csv")

val data = sc.textFile("/user/rahulranjantechie\_gmail/Project 1\_dataset\_bank-full (2).csv")

val header = data.first

val r\_data = data.filter(\_ != header)

val r\_data\_col = r\_data.map( x => x.split(";").map(x => x.replaceAll("\"","")))

val bank\_data = r\_data\_col.map(attributes => bankdata(attributes(0).trim.toInt,attributes(1),attributes(2),attributes(3),attributes(4),attributes(5).trim.toInt,attributes(6),attributes(7),attributes(8),attributes(9).trim.toInt,attributes(10),attributes(11).trim.toInt,attributes(12).trim.toInt,attributes(13).trim.toInt,attributes(14).trim.toInt,attributes(15),attributes(16))).toDF()

//Give marketing success rate (No. of people subscribed / total no. of entries)

val marketSuccess :Double= (bank\_data.filter(col("y").equalTo("yes")).count) \* 100 / bank\_data.count

bank\_data.select("y").distinct.show

bank\_data.filter(col("y").equalTo("yes")).count

bank\_data.count

//Give the marketing failure rate

val marketFailure :Double= (bank\_data.filter(col("y").equalTo("no")).count) \* 100 / bank\_data.count

//Give the maximum, mean and minimum age of average targeted customer.

bank\_data.select("age").describe().show

// With the above code we can see the min max and mean age of the targeted cutomer.

// We can see them seperately as well with below codes.

bank\_data.select(max($"age")).show()

bank\_data.select(min($"age")).show()

bank\_data.select(avg($"age")).show()

// Check the quality of customer by checking the average balance and median balance.

bank\_data.select("balance").describe().show

bank\_data.select(avg($"balance")).show()

// Check the quality of customer by checking the median balance.

bank\_data.stat.approxQuantile("balance", Array(0.5), 0.0001)

//Applying function to save this datframe as view of sql query.

bank\_data.createOrReplaceTempView("datanewtable")

val median= spark.sql("SELECT percentile\_approx(balance,0.5) FROM datanewtable").show()

//Check if age matters in marketing subscription for deposit.

val agedata =bank\_data.filter(col("y").equalTo("yes")).groupBy("age").count.orderBy(col("count").desc)

agedata.show()

//Check if marital status mattered for a subscription to deposit.

val maritaldata =bank\_data.filter(col("y").equalTo("yes")).groupBy("marital").count.orderBy(col("count").desc)

maritaldata.show()

//Check if age and marital status together mattered for a subscription to deposit scheme.

val ageandmaritaldata =bank\_data.filter(col("y").equalTo("yes")).groupBy("age","marital").count.orderBy(col("count").desc)

ageandmaritaldata.show()

// Do the feature engineering for the bank and find the right age effect on the campaign.

ageandmaritaldata.select("age").distinct.count

val agedata = spark.udf.register("agedata",(age:Int) => {if (age < 20) "Teen"

else if (age >= 20 && age <= 32)

"Young"

else if (age >= 33 && age <= 55)

"Middle Aged"

else

"old"

})

val banknewDF = bank\_data.withColumn("age\_cat",agedata(bank\_data("age")))

banknewDF.show()

val targetage = banknewDF.filter(col("y").equalTo("yes")).groupBy("age\_cat").count.orderBy(col("count").desc)

targetage.show()

All the code used in the project have been attached above in this word document.

**Thank you**

**Submitted by:-**

**Rahul Ranjan**

**Email:- rahulranjantechie@gmail.com**