**1)Create Dataframe**

scala> val lines = sc.textFile("/user/jayeshrasikkumaranz/simplilearn/bankmarketingdata.csv")

lines: org.apache.spark.rdd.RDD[String] = /user/jayeshrasikkumaranz/simplilearn/bankmarketingdata.csv MapPartitionsRDD[1] at textFile at <console>:23

val bank = lines.map(x => x.split(";")) –

**-Drop header**

val bankf = bank.mapPartitionsWithIndex { (idx, iter) => if (idx == 0) iter.drop(1) else iter }

bankrdd.registerTempTable("bank")

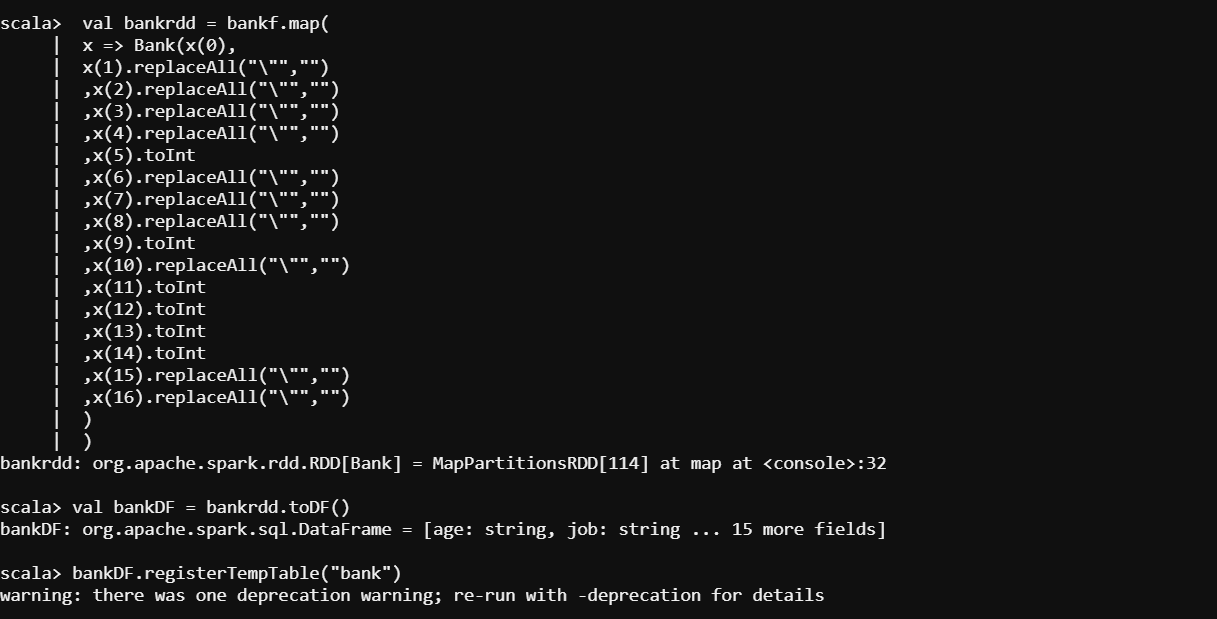
**Define Class**

scala> case class Bank(age:Int, job:String, marital:String, education:String, deffield:String, balance:Int,

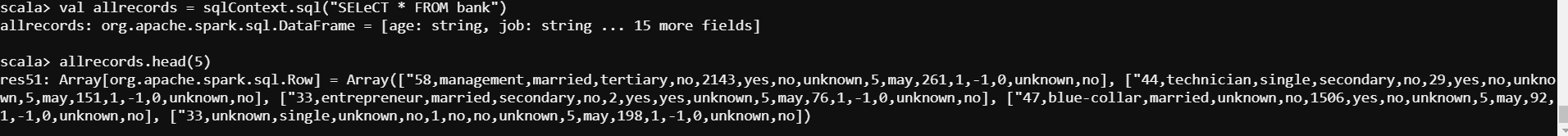
| housing:String, loan:String, contact:String, day:Int, month: String, duration:Int, campaign:Int, pdays:Int,

| previous:Int, poutcome:String, y:String)

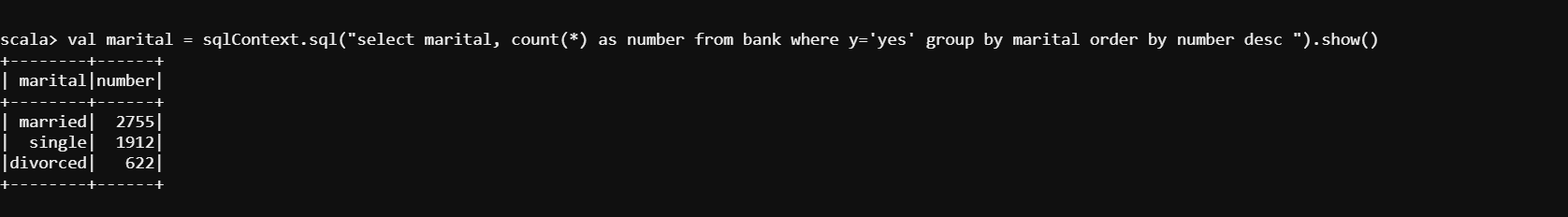
**defined class Bank**



**Check records are read correctly**



**Check if marital status matters**



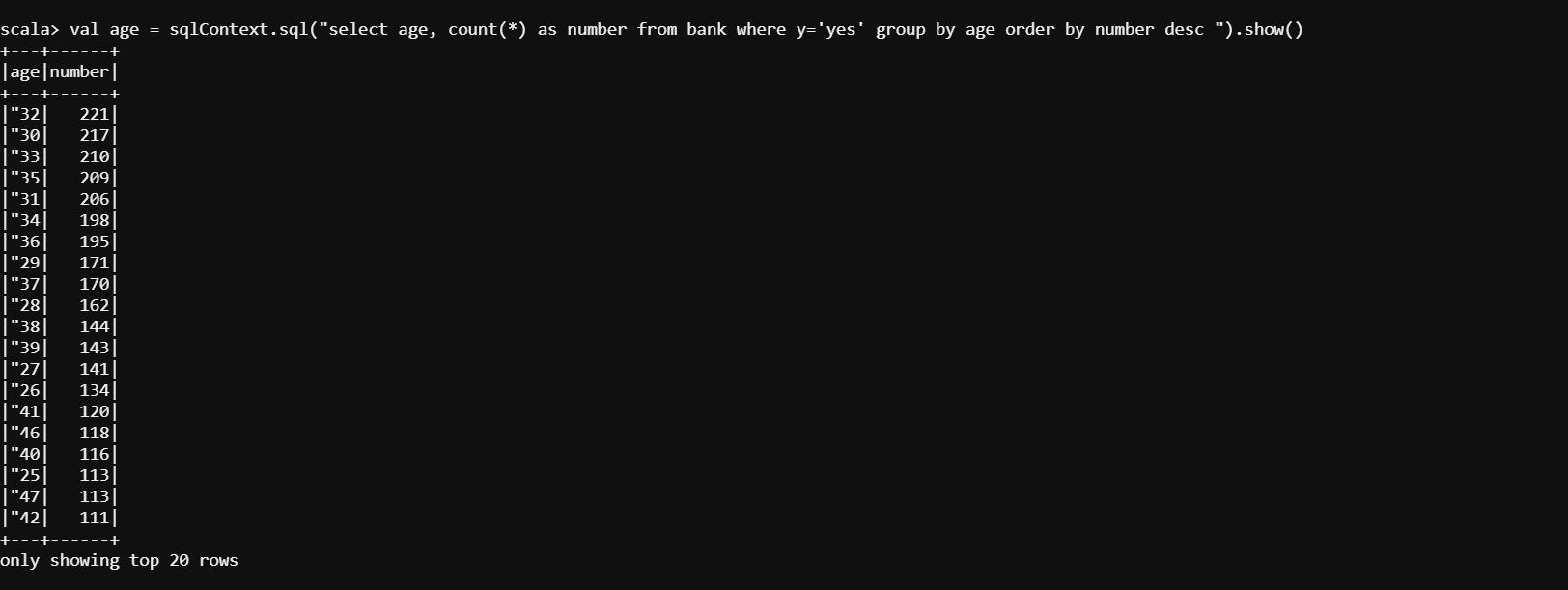
**3)Max,Min, Mean age of targeted customer**



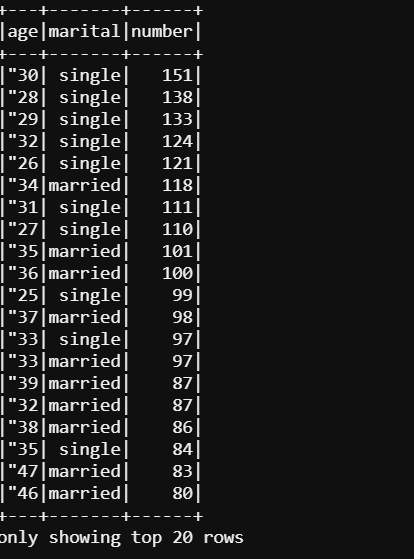
**Check if age matters in the marketing subscription for deposit**

val age = sqlContext.sql("select age, count(\*) as number from bank where y='yes' group by age order by number desc ").show()

**The age range between (30-36) shows most promise.**



val age\_marital = sqlContext.sql("select age, marital, count(\*) as number from bank where y='yes' group by age,marital order by number desc ").show()

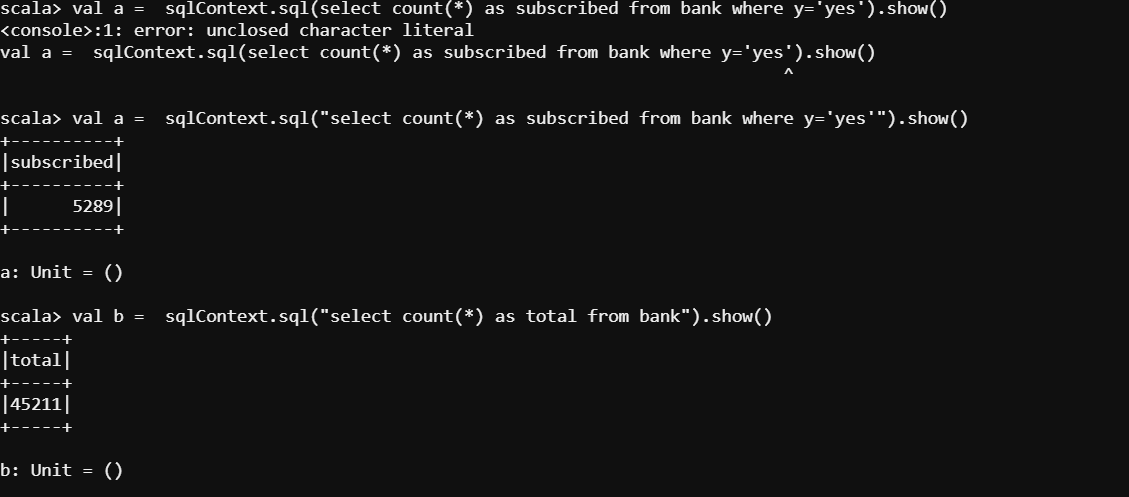


**No Marketing Success Rate**

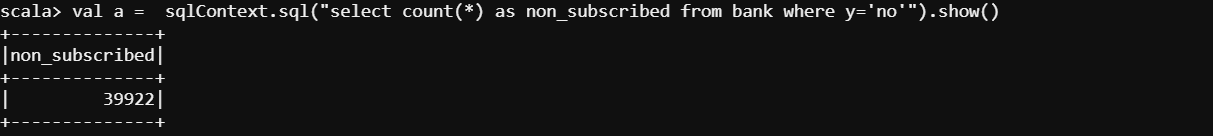
--2) Marketing Success Rate val success = sqlContext.sql("select (a.subscribed/b.total)\*100 as success\_percent from (select count(\*) as subscribed from bank where y='yes') a,(select count(\*) as total from bank) b").show() –

**Marketing Failure Rate**

val failure = sqlContext.sql("select (a.not\_subscribed/b.total)\*100 as failure\_percent from (select count(\*) as not\_subscribed from bank where y='no') a,(select count(\*) as total from bank) b").show()



**No subscriber**



8) Feature Engineering for cloumn "age" --Import necessary libraries

import scala.reflect.runtime.universe

import org.apache.spark.SparkConf

import org.apache.spark.SparkContext

import org.apache.spark.sql.DataFrame

import org.apache.spark.sql.SQLContext

import org.apache.spark.sql.functions.mean

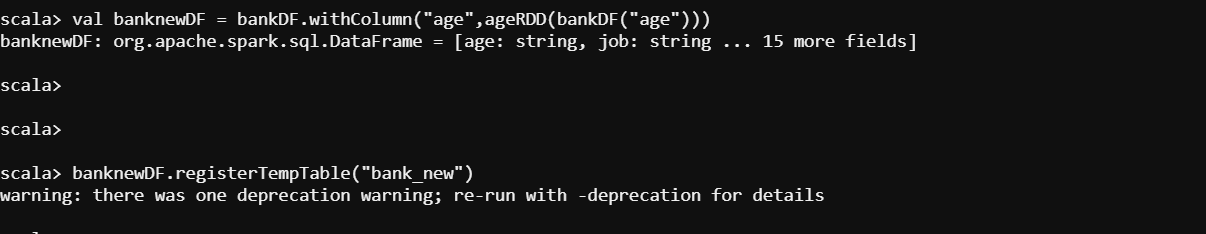
--Defining a new UDF with which we will generate new features.

**We divide the age groups into 4 categories.**

val ageRDD = sqlContext.udf.register("ageRDD",(age:Int) => { if (age < 20) "Teen" else if (age > 20 && age <= 32) "Young" else if (age > 33 && age <= 55) "Middle Aged" else "Old" })

scala> val banknewDF = bankDF.withColumn("age",ageRDD(bankDF("age")))banknewDF: org.apache.spark.sql.DataFrame = [age: string, job: string ... 15 more fields]

scala> banknewDF.registerTempTable("bank\_new") warning: there was one deprecation warning; re-run with -deprecation for details



val age\_target = sqlContext.sql("select age, count(\*) as number from bank\_new where y='yes' group by age order by number desc ").show()