Apache Spark—Real Time Project—Marketing Analysis

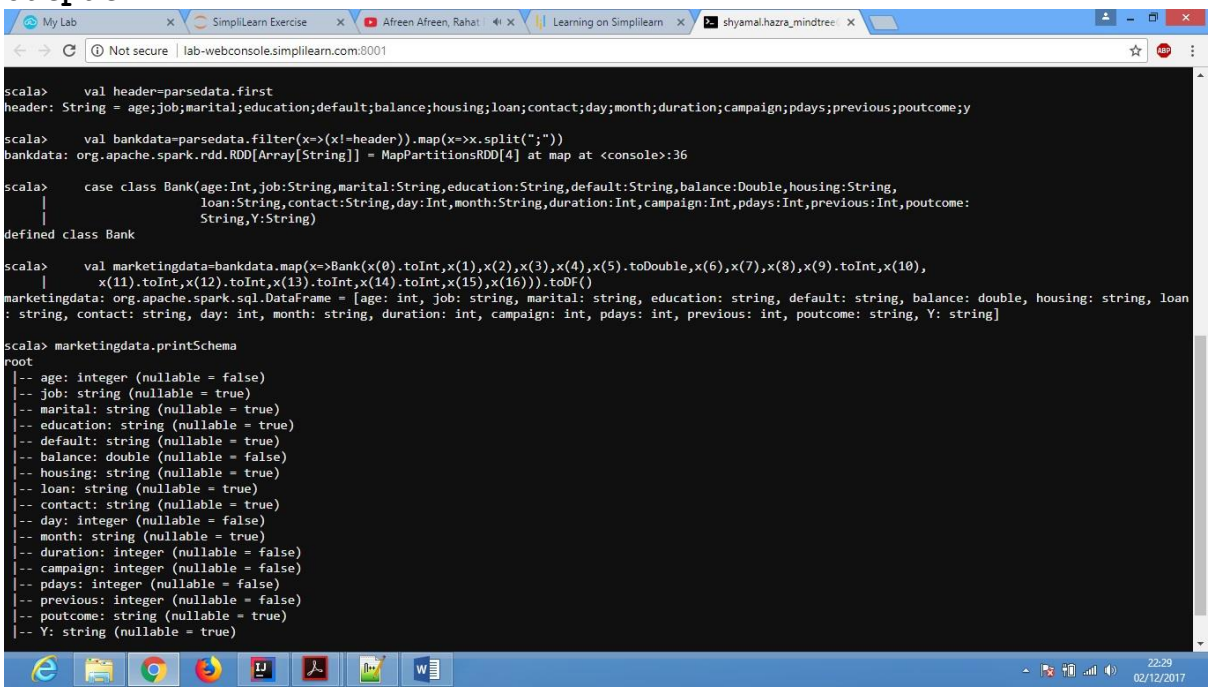
Analyze marketing data for call campaign by bank

SHYAMAL HAZRA

1. Load data and create Spark data frame

```
val dataset = sc.textFile("/user/shyamal.hazra_mindtree/Labdata/Project
1_dataset_bank-full (2).csv")

val parsedata=dataset.map(x=>x.replaceAll("\"",""))

val bankdata=parsedata.filter(x=>(x!=header)).map(x=>x.split(";"))

case class
Bank(age:Int,job:String,marital:String,education:String,default:String,balance:D
ouble,housing:String,

loan:String,contact:String,day:Int,month:String,duration:Int,campaign:Int,pdays:
Int,previous:Int,poutcome: String,Y:String)

val marketingdata=bankdata.map(x=>Bank(x(0).toInt,x(1),x(2),x(3),x(4),
x(5).toDouble,x(6),x(7),x(8),x(9).toInt,x(10),
  x(11).toInt,x(12).toInt,x(13).toInt,x(14).toInt,x(15),x(16))).toDF()
```

Output:



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

```
val successRecors=marketingdata.filter($"Y"=="yes").count()

val totalrecord=marketingdata.count()

val successRecors=marketingdata.filter($"Y"=="yes").count()

val successRate=(successRecors/totalrecord.toFloat)*100
```

Output:

```
: string, contact: string, day: int, month: string, duration: int, campaign: int, pdays: int, previous: int, poutcome: string, Y: string]

scala> marketingdata.printSchema
root
 |-- age: integer (nullable = false)
 |-- job: string (nullable = true)
 |-- marital: string (nullable = true)
 |-- education: string (nullable = true)
 |-- default: string (nullable = true)
 |-- balance: double (nullable = false)
 |-- housing: string (nullable = true)
 |-- loan: string (nullable = true)
 |-- contact: string (nullable = true)
 |-- day: integer (nullable = false)
 |-- month: string (nullable = true)
 |-- duration: integer (nullable = false)
 |-- campaign: integer (nullable = false)
 |-- pdays: integer (nullable = false)
 |-- previous: integer (nullable = false)
 |-- poutcome: string (nullable = true)
 |-- Y: string (nullable = true)


scala> val successRecors=marketingdata.filter($"Y"=="yes").count()
successRecors: Long = 5289

scala>     val totalrecord=marketingdata.count()
totalrecord: Long = 45211

scala>     val successRecors=marketingdata.filter($"Y"=="yes").count()//5289
successRecors: Long = 5289

scala>     val successRate=(successRecors/totalrecord.toFloat)*100
successRate: Float = 11.698481

scala>
```

2a Give marketing failure rate

```
val failrecord=marketingdata.filter($"Y"=="no").count()

val failrate=(failrecord/(totalrecord.toFloat))*100
```

Output:

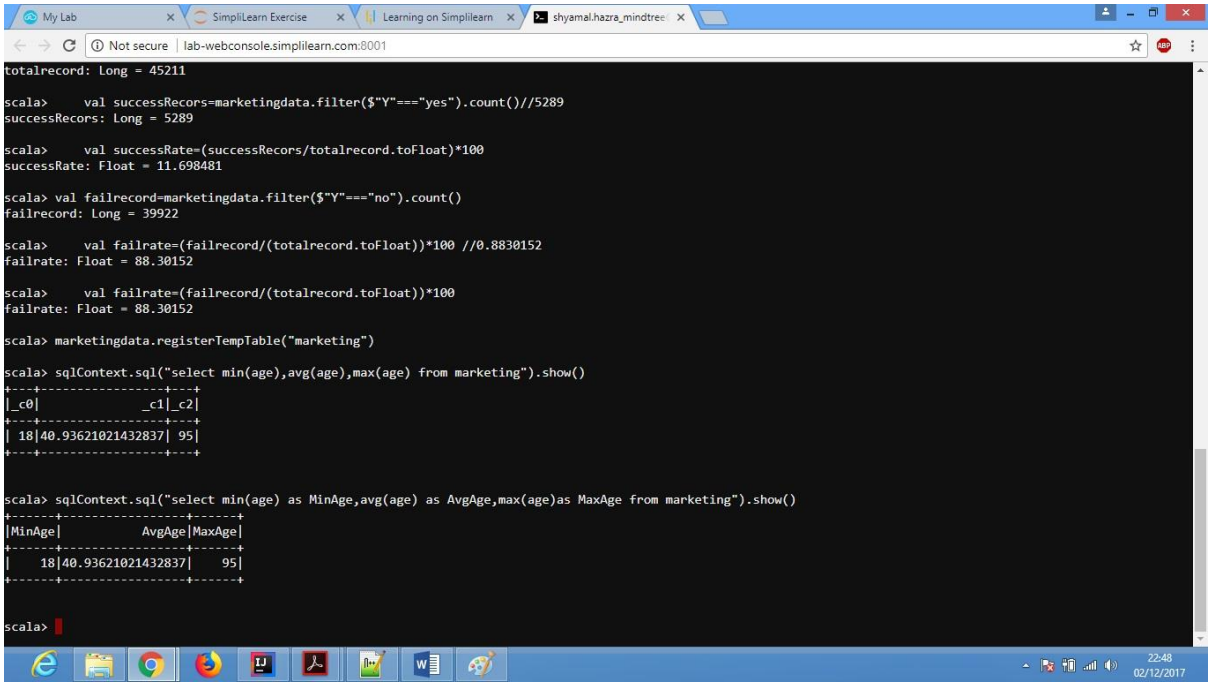
```
 |-- balance: double (nullable = false)
 |-- housing: string (nullable = true)
 |-- loan: string (nullable = true)
 |-- contact: string (nullable = true)
 |-- day: integer (nullable = false)
 |-- month: string (nullable = true)
 |-- duration: integer (nullable = false)
 |-- campaign: integer (nullable = false)
 |-- pdays: integer (nullable = false)
 |-- previous: integer (nullable = false)
 |-- poutcome: string (nullable = true)
 |-- Y: string (nullable = true)


scala> val successRecors=marketingdata.filter($"Y"=="yes").count()
successRecors: Long = 5289

scala>     val totalrecord=marketingdata.count()
totalrecord: Long = 45211

scala>     val successRecors=marketingdata.filter($"Y"=="yes").count()//5289
successRecors: Long = 5289

scala>     val successRate=(successRecors/totalrecord.toFloat)*100
successRate: Float = 11.698481

scala> val failrecord=marketingdata.filter($"Y"=="no").count()
failrecord: Long = 39922

scala>     val failrate=(failrecord/(totalrecord.toFloat))*100 //0.8830152
failrate: Float = 88.30152

scala>     val failrate=(failrecord/(totalrecord.toFloat))*100
failrate: Float = 88.30152

scala>
```

2. Maximum, Mean, and Minimum age of average targeted customer

```
marketingdata.registerTempTable("marketing")
```

```
sqlContext.sql("select min(age) as MinAge,avg(age) as AvgAge,max(age)as MaxAge from marketing").show()
```

Output:



4.Check quality of customers by checking average balance, median balance of customers

```
sqlContext.sql("select avg(balance)as AvergaeBalance,percentile_approx(balance, 0.5)as MedianBalance from marketing").show()
```

Output:

```
scala> val failrecord=marketingdata.filter($"Y"=="no").count()
failrecord: Long = 39922

scala>     val failrate=(failrecord/(totalrecord.toFloat))*100 //0.8830152
failrate: Float = 88.30152

scala>     val failrate=(failrecord/(totalrecord.toFloat))*100
failrate: Float = 88.30152

scala> marketingdata.registerTempTable("marketing")

scala> sqlContext.sql("select min(age),avg(age),max(age) from marketing").show()
+---+-----------------+---+
|_c0|              _c1|_c2|
+---+-----------------+---+
| 18|40.93621021432837| 95|
+---+-----------------+---+


scala> sqlContext.sql("select min(age) as MinAge,avg(age) as AvgAge,max(age)as MaxAge from marketing").show()
+------+-----------------+------+
|MinAge|           AvgAge|MaxAge|
+------+-----------------+------+
|    18|40.93621021432837|    95|
+------+-----------------+------+


scala> sqlContext.sql("select avg(balance)as AvergaeBalance,percentile_approx(balance, 0.5)as MedianBalance from marketing").show()
+------------------+-------------+
|    AvergaeBalance|MedianBalance|
+------------------+-------------+
|1362.272057685076 6|    447.84375|
+------------------+-------------+


scala>
```

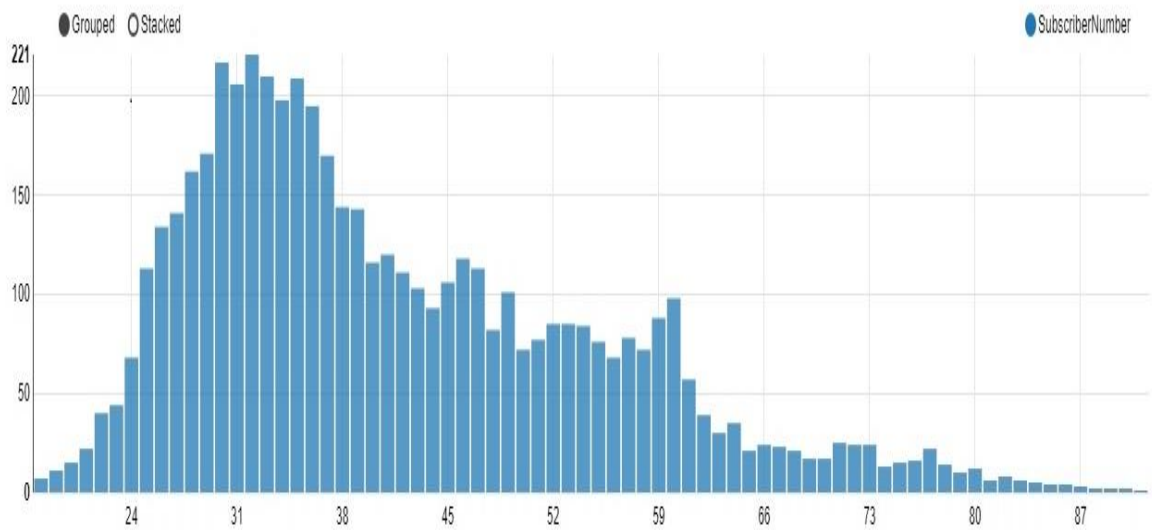
5. Check if age matters in marketing subscription for deposit

```
sqlContext.sql("select age,count(*) as SubscriberNumber from marketing where
Y='yes' group by age order by SubscriberNumber desc").show()
```

Output: Yes age matters as we can see in figure below that major

Number of subscriber are in age range of 27-38.

(N.B::Using Zeppelin notebook for better visualization)

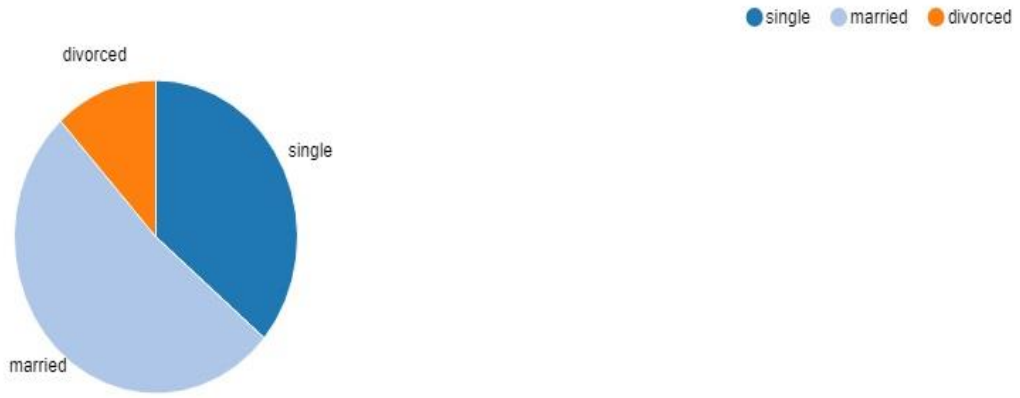


6. Check if marital status mattered for subscription to deposit.

```
sqlContext.sql("select marital,count(*) as SubscriberNumber from marketing where
Y='yes' group by marital order by SubscriberNumber desc").show()
```

Output: Yes martial status matters as we can see from below result

(N.B::Using Zeppelin notebook for better visualization)

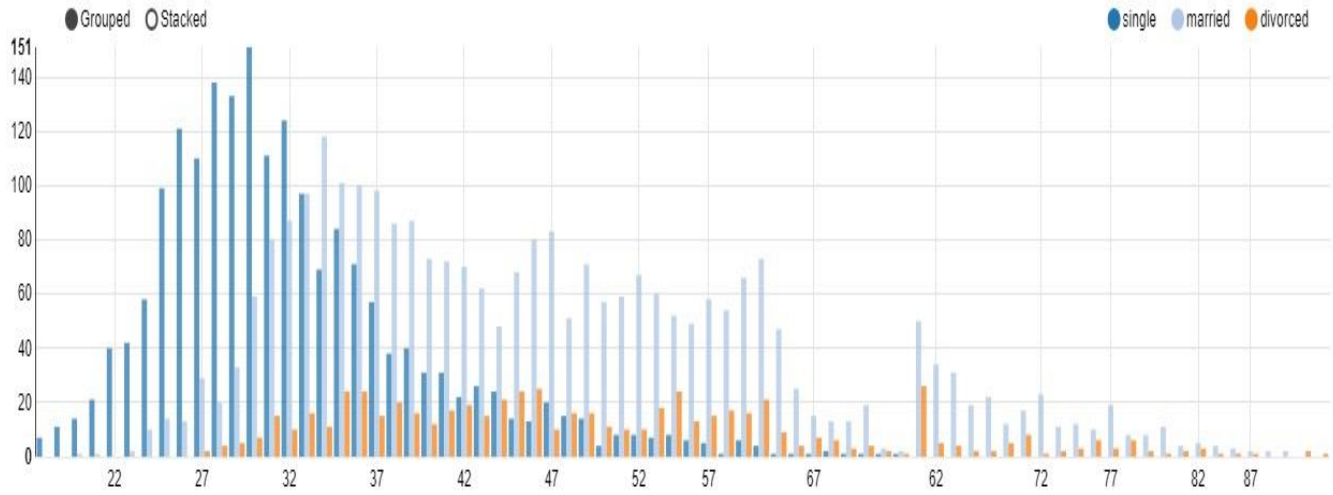


7. Check if age and marital status together mattered for subscription to deposit scheme

```
sqlContext.sql("select age,marital,count(*) as SubscriberNumber from marketing
where Y='yes' group by age,marital order by SubscriberNumber desc").show()
```

Output: Yes age and martial status mattered for subscription, as we can see from output below

(N.B::Using Zeppelin notebook for better visualization)



8. Do feature engineering for column—age and find right age effect on campaign

```
val ageIndexer = sqlContext.udf.register("ageIndexer",(age:Int) => {
  if(age<20){
    "Teen"
  }
  else if(age>=20 && age<=35){
    "Youth"
  }
  else if(age>=36 && age<=60)
    "Mid age"
  else {
    "Senior citizen"
  }
})

val ageIndexedDf=marketingdata.filter(marketingdata("Y")=="yes")withColumn("age",
ageIndexer(marketingdata("age")))

ageIndexedDf.registerTempTable("marketingIndexed")

val ageindexer=new StringIndexer().setInputCol("age").setOutputCol("AgeIndex")

val model=ageindexer.fit(ageIndexedDf)

model.transform(ageIndexedDf).select("age","AgeIndex").groupBy("AgeIndex").count().
show()
```

Output: By feture engineering we can see that age group 36-60 has highest effect of campaign.

```
scala>     val model=ageindexer.fit(ageIndexedDf)
model: org.apache.spark.ml.feature.StringIndexerModel = strIdx_41e38c5b5b69

scala>  val targetedGroup=sqlContext.sql("select age,count(*) as CustomerAgerGroupNumber from marketingIndexed where Y='yes' group by age order by CustomerAgerGrou
pNumber desc").show()
+--------------+----------------------+
|           age|CustomerAgerGroupNumber|
+--------------+----------------------+
|       Mid age|                  2598|
|         Youth|                  2171|
|Senior citizen|                   502|
|          Teen|                    18|
+--------------+----------------------+

targetedGroup: Unit = ()

scala>     val ageindexer=new StringIndexer().setInputCol("age").setOutputCol("AgeIndex")
ageindexer: org.apache.spark.ml.feature.StringIndexer = strIdx_8b51e8bbbeaa

scala>     //ageindexer.explainParams()

scala>     val model=ageindexer.fit(ageIndexedDf)
model: org.apache.spark.ml.feature.StringIndexerModel = strIdx_8b51e8bbbeaa

scala>     model.transform(ageIndexedDf).select("age","AgeIndex").groupBy("AgeIndex").count().show()
+--------+-----+
|AgeIndex|count|
+--------+-----+
|     1.0| 2171|
|     3.0|   18|
|     0.0| 2598|
|     2.0|  502|
+--------+-----+


scala>
```