Market Analysis In Banking Domain

By Siddharth Sudhakar

```
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
```

1. Load Data And Create A Spark Dataframe

```
val bank_data = sc.textFile("C:/spark/banking_market_analysis/Project 1.csv")

val bank = bank_data.map(x => x.split(";"))
```

Remove the first row which are the names of the columns

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

Define a class for the schema

```
case class Bank(age:Int, job:String, marital:String, education:String, defaultn: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)
```

Map class data to RDD

```
val bank_clean = bank_r.map(
x => Bank(x(0).replaceAll("\"","").toInt,
x(1).replaceAll("\"",""),x(2).replaceAll("\"",""),x(3).replaceAll("\"",""),x(4).replaceAll("\"","")
,x(5).replaceAll("\"","").toInt,x(6).replaceAll("\"",""),x(7).replaceAll("\"",""),x(8).replaceAll("\"","")
,x(9).replaceAll("\"","").toInt,x(10).replaceAll("\"",""),x(11).replaceAll("\"","").toInt,x(12).replaceAll("
\"","").toInt,x(13).replaceAll("\"","").toInt,x(14).replaceAll("\"","").toInt
,x(15).replaceAll("\"",""),x(16).replaceAll("\"","")))
```

Convert RDD back to a DataFrame and then create a TempView instance

```
val bank_df = bank_clean.toDF()
bank_df.createOrReplaceTempView("bank")
```

2. Marketing Success And Failure Rate

```
val builder=new org.apache.spark.sql.SparkSession.Builder()
val sparkSession=builder.getOrCreate()
val sqlContext=sparkSession.sqlContext;
```

11.7 % of the people to whom the marketing campaign was initiated have subscribed to the company

```
val success = sqlContext.sql("select round((x.subscribed/y.total)*100,3) as success_rate from (se
lect count(*) as subscribed from bank where y='yes') x,(select count(*) as total from bank) y").show()
```

```
+------------+
|success_rate|
+------------+
|      11.698|
+------------+
```

```
val failure = sqlContext.sql("select round((x.not_subscribed/y.total)*100,3) as failure_rate from
 (select count(*) as not_subscribed from bank where y='no') x,(select count(*) as total from bank)
y").show()
```

```
+------------+
|failure_rate|
+------------+
|      88.302|
+------------+
```

3. Maximum, Mean, And Minimum Age Of The Average Targeted Customer

The maximum, minimum and mean age of the targeted customer is 95, 18 and 41 respectively.

```
sqlContext.sql("select max(age) as max_age,min(age) as min_age,round(avg(age),2) as avg_age from
bank").show()
```

```
+-------+-------+-------+
|max_age|min_age|avg_age|
+-------+-------+-------+
|     95|     18|  40.94|
+-------+-------+-------+
```

4. Average Balance, Median Balance Of Customers

```
val average_balance = sqlContext.sql("select round(avg(balance),2) as average_balance from
bank").show()
```

```
+---------------+
|average_balance|
+---------------+
|        1362.27|
+---------------+
```

val median_balance = sqlContext.sql(“SELECT percentile_approx(balance,0.5) as median_balance FROM bank”).show()

```
+--------------+
|median_balance|
+--------------+
|           448|
+--------------+
```

5. Role Of Age In Marketing Subscription For Deposit

Majority of the customers who had subscribed to the company were between the age of 25-35.

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

```
+---+------+
|age|number|
+---+------+
| 32|   221|
| 30|   217|
| 33|   210|
| 35|   209|
| 31|   206|
| 34|   198|
| 36|   195|
| 29|   171|
| 37|   170|
| 28|   162|
| 38|   144|
| 39|   143|
| 27|   141|
| 26|   134|
| 41|   120|
| 46|   118|
| 40|   116|
| 25|   113|
| 47|   113|
| 42|   111|
+---+------+
only showing top 20 rows
```

Below table shows the percentage of people who have subscribed to the company among the total number of people to whom the campaign was reached in each age category.

val response_by_age = sqlContext.sql("select x.age,round((x.subscribed/y.total)*100,2) as percent from (select age,count(age) as subscribed from bank where y='yes' group by age) x,(select age,count(age) as total from bank group by age) y where x.age = y.age order by percent desc").show()

```
+---+-------+
|age|percent|
+---+-------+
| 90|  100.0|
| 93|  100.0|
| 92|  100.0|
| 85|   80.0|
| 87|   75.0|
| 68|  58.33|
| 18|  58.33|
| 84|  55.56|
| 73|  54.55|
| 76|   50.0|
| 95|   50.0|
| 77|   50.0|
| 62|  48.75|
| 64|   47.3|
| 78|  46.67|
| 71|   46.3|
| 72|  46.15|
| 86|  44.44|
| 67|  42.59|
| 82|  42.11|
+---+-------+
only showing top 20 rows
```

Though the number of people who have subscribed to the company is large for people below the age of 45, but greater subscription rate was seen in people above the age of 60 (senior citizens).

Therefore, in the next marketing campaign it is advised to target senior citizens.

6. Role Of Marital Status For A Marketing Subscription To Deposit

Below table shows the number of people subscribed to the company segregated based on their marital status

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

```
+--------+------+
| marital|number|
+--------+------+
| married|  2755|
|  single|  1912|
|divorced|   622|
+--------+------+
```

Below table shows the percentage of people who have subscribed to the company among the people to whom the campaign was reached based on their marital status

val response_by_marital = sqlContext.sql("select x.marital,(x.subscribed/y.total)*100 as percent from (select marital,count(marital) as subscribed from bank where y='yes' group by marital) x,(select marital,count(marital) as total from bank group by marital) y where x.marital = y.marital order by percent desc").show()

```
+--------+-------+
| marital|percent|
+--------+-------+
|  single|  14.95|
|divorced|  11.95|
| married|  10.12|
+--------+-------+
```

From the above table we can infer that the subscription rate was relatively higher for people whose marital status was “single”.

7. Does Age And Marital Status Together Matter For A Subscription In The Deposit Scheme ?

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()

```
+---+-------+------+
|age|marital|number|
+---+-------+------+
| 30| single|   151|
| 28| single|   138|
| 29| single|   133|
| 32| single|   124|
| 26| single|   121|
| 34|married|   118|
| 31| single|   111|
| 27| single|   110|
| 35|married|   101|
| 36|married|   100|
| 25| single|    99|
| 37|married|    98|
| 33| single|    97|
| 33|married|    97|
| 32|married|    87|
| 39|married|    87|
| 38|married|    86|
| 35| single|    84|
| 47|married|    83|
| 46|married|    80|
+---+-------+------+
only showing top 20 rows
```

val response_by_marital_age = sqlContext.sql("selectx.marital,x.age, round((x.subscribed/y.total)*100,2) as percent from (select marital,age,count(marital) as subscribed from bank where y='yes' group by marital,age) x,(select marital,age,count(marital) as total from bank group by marital,age) y where x.marital = y.marital and x.age = y.age order by percent desc").show()

marital	age	percent
divorced	68	100.0
divorced	87	100.0
divorced	90	100.0
divorced	85	100.0
divorced	95	100.0
single	86	100.0
married	93	100.0
married	92	100.0
divorced	67	87.5
divorced	62	83.33
divorced	76	75.0
married	85	75.0
divorced	71	72.73
divorced	73	66.67
married	87	66.67
married	84	66.67
divorced	77	60.0
single	18	58.33
divorced	63	57.14
married	73	52.78

From the above table it can be inferred that the subscription rate was higher for those people whose marital status was "divorced" and they belong to the age category "above 60" or senior citizens.

8. Let Us Find The Effect Of The Marketing Campaign On Different Age Groups

❖ **Grouping By Age**

We define a UDF to divide the age into 4 categories

```
 val age_RDD = sqlContext.udf.register("age_RDD",(age:Int) => {
 if (age >= 18 && age <= 30)
"Young_Adult"
else if (age > 30 && age <= 45)
"Adult"
else if (age>45 && age <=60)
"Middle_aged"
else
"Old"
})
```

❖ **Pipeline Of Stringindexer Based On Age Group**

```
val bank_DF1 = bank_df.withColumn("age",age_RDD(bank_df("age")))

bank_DF1.createOrReplaceTempView("bank_DF1")

val age_index = new
org.apache.spark.ml.feature.StringIndexer().setInputCol("age").setOutputCol("ageIndex")
```

❖ **Fit And Transform**

```
var model_fit = age_index.fit(bank_DF1)
model_fit.transform(bank_DF1).select("age","ageIndex").show()
```

```
+-----------+--------+
|        age|ageIndex|
+-----------+--------+
|Middle_aged|     1.0|
|      Adult|     0.0|
|      Adult|     0.0|
|Middle_aged|     1.0|
|      Adult|     0.0|
|      Adult|     0.0|
|Young_Adult|     2.0|
|      Adult|     0.0|
|Middle_aged|     1.0|
|      Adult|     0.0|
|      Adult|     0.0|
|Young_Adult|     2.0|
|Middle_aged|     1.0|
|Middle_aged|     1.0|
|Middle_aged|     1.0|
|Middle_aged|     1.0|
|      Adult|     0.0|
|Middle_aged|     1.0|
|Middle_aged|     1.0|
|      Adult|     0.0|
+-----------+--------+
```

So we conclude from the Feature Engineering that the company's most potential customers follows the order Old>Young_Adult>Middle_aged>Adult.