Project 1- Apache Spark—Real Time Project—Marketing Analysis

Pre-requisites:

The csv data file was cleaned and loaded as comma separated text file P1_Bank_DataSet.txt.

Then the file was uploaded to cloudlab using FTP service.

Then they were uploaded to Hadoop FS using the command:

```
hadoop fs -put P1_Bank_DataSet.txt .
```

The spark shell is then launched and the data processing starts.

1. Load data and create Spark data frame

```
scala> val lines = sc.textFile("P1_Bank_DataSet.txt")
lines: org.apache.spark.rdd.RDD[String] = P1_Bank_DataSet.txt MapPartitionsRDD[1] at textFile at <console>:27

scala> val bank = lines.map(x => x.split(","))
bank: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[2] at map at <console>:29

scala> case class Bank(age:Int, job:String, marital:String, education:String, default:String, balance:Int, housing:String, loan:String, contact:Strin
g, day:Int, month:String, duration:Int, campaign:Int, pdays:Int, previous:Int, poutcome:String, y:String)
defined class Bank

scala> val bankrdd = bank.map( x => Bank(x(0).toInt,
     | x(1).replaceAll("\"",""),
     | x(2).replaceAll("\"",""),
     | x(3).replaceAll("\"",""),
     | x(4).replaceAll("\"",""),
     | x(5).toInt,
     | x(6).replaceAll("\"",""),
     | x(7).replaceAll("\"",""),
     | x(8).replaceAll("\"",""),
     | x(9).toInt,
     | x(10).replaceAll("\"",""),
     | x(11).toInt,
     | x(12).toInt,
     | x(13).toInt,
     | x(14).toInt,
     | x(15).replaceAll("\"",""),
     | x(16).replaceAll("\"","")
     | )
     | )
bankrdd: org.apache.spark.rdd.RDD[Bank] = MapPartitionsRDD[3] at map at <console>:33
```

```
scala> val bankDF = bankrdd.toDF()
bankDF: org.apache.spark.sql.DataFrame = [age: int, job: string, marital: string, education: string, default: 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]

scala> bankDF.printSchema()
root
 |-- age: integer (nullable = false)
 |-- job: string (nullable = true)
 |-- marital: string (nullable = true)
 |-- education: string (nullable = true)
 |-- default: string (nullable = true)
 |-- balance: integer (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> bankDF.show()
+---+-----------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+---+
|age|        job| marital|education|default|balance|housing|loan|contact|day|month|duration|campaign|pdays|previous|poutcome|  y|
+---+-----------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+---+
| 58|  management| married| tertiary|     no|   2143|    yes|  no|unknown|  5|  may|     261|       1|   -1|       0| unknown| no|
| 44|  technician|  single|secondary|     no|     29|    yes|  no|unknown|  5|  may|     151|       1|   -1|       0| unknown| no|
| 33|entrepreneur| married|secondary|     no|      2|    yes| yes|unknown|  5|  may|      76|       1|   -1|       0| unknown| no|
| 47| blue-collar| married|  unknown|     no|   1506|    yes|  no|unknown|  5|  may|      92|       1|   -1|       0| unknown| no|
| 33|     unknown|  single|  unknown|     no|      1|     no|  no|unknown|  5|  may|     198|       1|   -1|       0| unknown| no|
| 35|  management| married| tertiary|     no|    231|    yes|  no|unknown|  5|  may|     139|       1|   -1|       0| unknown| no|
| 28|  management|  single| tertiary|     no|    447|    yes| yes|unknown|  5|  may|     217|       1|   -1|       0| unknown| no|
| 42|entrepreneur|divorced| tertiary|    yes|      2|    yes|  no|unknown|  5|  may|     380|       1|   -1|       0| unknown| no|
| 58|     retired| married|  primary|     no|    121|    yes|  no|unknown|  5|  may|      50|       1|   -1|       0| unknown| no|
| 43|  technician|  single|secondary|     no|    593|    yes|  no|unknown|  5|  may|      55|       1|   -1|       0| unknown| no|
| 41|      admin.|divorced|secondary|     no|    270|    yes|  no|unknown|  5|  may|     222|       1|   -1|       0| unknown| no|
| 29|      admin.|  single|secondary|     no|    390|    yes|  no|unknown|  5|  may|     137|       1|   -1|       0| unknown| no|
| 53|  technician| married|secondary|     no|      6|    yes|  no|unknown|  5|  may|     517|       1|   -1|       0| unknown| no|
| 58|  technician| married|  unknown|     no|     71|    yes|  no|unknown|  5|  may|      71|       1|   -1|       0| unknown| no|
| 57|    services| married|secondary|     no|    162|    yes|  no|unknown|  5|  may|     174|       1|   -1|       0| unknown| no|
| 51|     retired| married|  primary|     no|    229|    yes|  no|unknown|  5|  may|     353|       1|   -1|       0| unknown| no|
| 45|      admin.|  single|  unknown|     no|     13|    yes|  no|unknown|  5|  may|      98|       1|   -1|       0| unknown| no|
| 57| blue-collar| married|  primary|     no|     52|    yes|  no|unknown|  5|  may|      38|       1|   -1|       0| unknown| no|
| 60|     retired| married|  primary|     no|     60|    yes|  no|unknown|  5|  may|     219|       1|   -1|       0| unknown| no|
| 33|    services| married|secondary|     no|      0|    yes|  no|unknown|  5|  may|      54|       1|   -1|       0| unknown| no|
+---+-----------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+---+
only showing top 20 rows
```

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

```
scala> val success = sqlContext.sql("select (a.subscribed/b.total)*100 as success_percent from (select count(*) as subscribed from bankAVS where y='y
es') a,(select count(*) as total from bankAVS) b").show()
+------------------+
|   success_percent|
+------------------+
|11.698480458295547|
+------------------+

success: Unit = ()
```

Marketing Success Rate : 11.698 %

2a.Give marketing failure rate

```
scala> val failure = sqlContext.sql("select (a.not_subscribed/b.total)*100 as failure_percent from (select count(*) as not_subscribed from bankAVS wh
ere y='no') a,(select count(*) as total from bankAVS) b").show()
+-----------------+
|  failure_percent|
+-----------------+
|88.30151954170445|
+-----------------+

failure: Unit = ()
```

Marketing Failure Rate : 88.301 %

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

```
scala> bankDF.select(max($"age")).show()
+--------+
|max(age)|
+--------+
|      95|
+--------+


scala> bankDF.select(min($"age")).show()
+--------+
|min(age)|
+--------+
|      18|
+--------+


scala> bankDF.select(avg($"age")).show()
+------------------+
|          avg(age)|
+------------------+
|40.93621021432837|
+------------------+
```

Max Age of Targeted Customer: 95
Min Age of Targeted Customer: 18
Average Age of Targeted Customer: 40.936

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

```
scala> bankDF.select(avg($"balance")).show()
+------------------+
|      avg(balance)|
+------------------+
|1362.2720576850766|
+------------------+


scala> val median = sqlContext.sql("SELECT percentile_approx(balance, 0.5) from bankAVS").show()
+---------+
|      _c0|
+---------+
|447.84375|
+---------+

median: Unit = ()
```

Average Balance of customers: 1362.272
Median Balance of customers: 447.843

5. Check if age matters in marketing subscription for deposit

```
scala> val age = sqlContext.sql("select age, count(*) as number from bankAVS 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|
| 47|   113|
| 25|   113|
| 42|   111|
+---+------+
only showing top 20 rows

age: Unit = ()
```

Conclusion:

Age matters. The age range of 30-36 is quite strong here.

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

```
scala> val marital = sqlContext.sql("select marital, count(*) as number from bankAVS where y='yes' group by marital order by number desc ").show()
+--------+------+
| marital|number|
+--------+------+
| married|  2755|
|  single|  1912|
|divorced|   622|
+--------+------+
```

Conclusion:
Marital Status also matters. Married people tend to do it more.

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

```
scala> val age_marital = sqlContext.sql("select age, marital, count(*) as number from bankAVS 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|married|    97|
| 33| single|    97|
| 32|married|    87|
| 39|married|    87|
| 38|married|    86|
| 35| single|    84|
| 47|married|    83|
| 46|married|    80|
+---+-------+------+
only showing top 20 rows

age_marital: Unit = ()
```

Conclusion:
Single people in the age 30-35 dominate the subscriptions.

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

```
scala> 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"
     | })
ageRDD: org.apache.spark.sql.UserDefinedFunction = UserDefinedFunction(<function1>,StringType,List(IntegerType))

scala>

scala> val banknewDF = bankDF.withColumn("age",ageRDD(bankDF("age")))
banknewDF: org.apache.spark.sql.DataFrame = [age: string, job: string, marital: string, education: string, default: string, balance: int, housing: st
ring, loan: string, contact: string, day: int, month: string, duration: int, campaign: int, pdays: int, previous: int, poutcome: string, y: string]

scala> banknewDF.registerTempTable("bank_new")

scala> val age_target = sqlContext.sql("select age, count(*) as number from bank_new where y='yes' group by age order by number desc ").show()
+-----------+------+
|        age|number|
+-----------+------+
|Middle Aged|  2601|
|      Young|  1539|
|        Old|  1131|
|       Teen|    18|
+-----------+------+

age_target: Unit = ()
```

Conclusion:

Age < 20 → Teen
Age in between 21-32 → Young
Age in between 33-55 → Middle Aged
Age > 56 → Old

We can conclude here that the 'Middle Aged' people between age 33 and 55 is the right age for the campaign.