ANALYZE MARKETING DATA FOR CALL CAMPAIGN BY BANK

Project submission by, Radhika Srinivasan

A Portuguese banking institution ran a marketing campaign to convince potential customers to invest in bank term deposit. The marketing campaigns were based on phone calls. Often, the same customer was contacted more than once through phone, in order to assess if they would want to subscribe to the bank term deposit or not.

Problem statement

Since data size is huge marketing team has asked to use Spark to help them get the following below.

1. Load data and create Spark data frame
2. Give marketing success rate
3. Give marketing failure rate
4. Maximum, Mean, Minimum age of average targeted customer
5. To check quality of customers by checking average balance, median balance of customers
6. To check if age matters in marketing subscription for deposit
7. To check if marital status mattered for subscription to deposit
8. To check if age and marital status together mattered for subscription to deposit scheme
9. To do feature engineering for column age and find right age effect on campaign

```
ip-10-0-1-10 login: radsrinivasan_gmail_com
Password:
Last failed login: Wed May 30 06:07:34 UTC 2018 from 183.82.22.222 on pts/16
There was 1 failed login attempt since the last successful login.
Last login: Tue May 29 13:16:31 from 183.82.22.222
[radsrinivasan_gmail_com@ip-10-0-1-10 ~]$ spark-shell --conf spark.ui.port=4040
--packages com.databri
cks:spark-csv_2.11:1.5.0
Ivy Default Cache set to: /home/radsrinivasan_gmail_com/.ivy2/cache
The jars for the packages stored in: /home/radsrinivasan_gmail_com/.ivy2/jars
:: loading settings :: url =
jar:file:/opt/cloudera/parcels/CDH-5.14.0-1.cdh5.14.0.p0.24/jars/spark-as
sembly-1.6.0-cdh5.14.0-hadoop2.6.0-cdh5.14.0.jar!/org/apache/ivy/core/settings/ivysetting
s.xml
com.databricks#spark-csv_2.11 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent;1.0
       confs: [default]
       found com.databricks#spark-csv_2.11;1.5.0 in central
       found org.apache.commons#commons-csv;1.1 in central
       found com.univocity#univocity-parsers;1.5.1 in central
:: resolution report :: resolve 289ms :: artifacts dl 4ms
       :: modules in use:
       com.databricks#spark-csv_2.11;1.5.0 from central in [default]
       com.univocity#univocity-parsers;1.5.1 from central in [default]
       org.apache.commons#commons-csv;1.1 from central in [default]
       ---------------------------------------------------------------------
```

```
       |                 |           modules            ||   artifacts   |
       |       conf      | number| search|dwnlded|evicted|| number|dwnlded|
       ---------------------------------------------------------------------
       |      default    |   3   |   0   |   0   |   0   ||   3   |   0   |
       ---------------------------------------------------------------------
:: retrieving :: org.apache.spark#spark-submit-parent
       confs: [default]
       0 artifacts copied, 3 already retrieved (0kB/6ms)
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/
Using Scala version 2.10.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_144)
Type in expressions to have them evaluated.
Type :help for more information.
Spark context available as sc (master = yarn-client, app id =
application_1525768906538_3695).
SQL context available as sqlContext.
scala>
```

1. Load data and create Spark data frame

Create dataframe bankingdf. Load bankmarketingdata file

```
scala> val bankingdf =
sqlContext.read.format("com.databricks.spark.csv").option("header","true").option("inferS
chema","true").option("delimiter",";").load("/user/radsrinivasan_gmail_com/bankmarketingd
ata.csv")
bankingdf: 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> bankingdf.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
```

We need Total count to calculate number of subscribers and non subscriber

```
scala> val totalCount = bankingdf.count().toDouble
totalCount: Double = 45211.0
```

Data above is collected by a Portuguese bank's contact-center to do direct marketing campaigns which were based on phone calls. The response variable 'y' contains 'yes', and 'no', representing if the client subscribes a bank term deposit.

```
scala> val subscriptionCount = bankingdf.filter($"y" === "yes").count().toDouble
subscriptionCount: Double = 5289.0
```

We get a count of 5289.0 subscriber

2. Give marketing success rate

```
scala> val successRate = subscriptionCount/totalCount
successRate: Double = 0.11698480458295547

scala> val nonSubscriberCount = bankingdf.filter($"y" === "no").count().toDouble
nonSubscriberCount: Double = 39922.0
```

We get a count of 39922.0 non-subscriber

3. Give marketing failure rate

```
scala> val failureRate = nonSubscriberCount/totalCount
failureRate: Double = 0.8830151954170445
```

4. Maximum, Mean, Minimum age of average targeted customer

```
scala> bankingdf.select(max($"age"), avg($"age"), min($"age")).show
+--------+-----------------+--------+
|max(age)|         avg(age)|min(age)|
+--------+-----------------+--------+
|      95|40.93621021432837|      18|
+--------+-----------------+--------+
```

Register bankingdf dataframe as a banking table

```
scala> bankingdf.registerTempTable("banking")
```

5. To check quality of customers by checking average balance, median balance of customers

```
scala> sqlContext.sql("select percentile(balance, 0.5) as median, avg(balance) as average
from banking").show
+------+------------------+
|median|           average|
+------+------------------+
| 448.0|1362.2720576850766|
+------+------------------+
```

6. To check if age matters in marketing subscription for deposit

```
scala> bankingdf.groupBy("y").agg(avg($"age")).show
+---+------------------+
|  y|          avg(age)|
+---+------------------+
| no| 40.83898602274435|
|yes|41.670069956513515|
+---+------------------+
```

From the output average age for marketing subscription is 41.67

```
scala> bankingdf.filter($"y" === "yes").groupBy("y","age").count().sort($"count".
desc).show
+---+---+-----+
|  y|age|count|
+---+---+-----+
|yes| 32|  221|
|yes| 30|  217|
|yes| 33|  210|
|yes| 35|  209|
|yes| 31|  206|
|yes| 34|  198|
|yes| 36|  195|
|yes| 29|  171|
|yes| 37|  170|
|yes| 28|  162|
|yes| 38|  144|
|yes| 39|  143|
|yes| 27|  141|
|yes| 26|  134|
|yes| 41|  120|
|yes| 46|  118|
|yes| 40|  116|
|yes| 25|  113|
|yes| 47|  113|
```

```
|yes| 42|  111|
+---+---+-----+
only showing top 20 rows
```

From the above data, we can conclude that age matters in marketing subscription for deposit. We can interpret that age range from **30-36** years shows most subscriptions. While the average age of subscription is **41.67** where **32** years of Age group has the highest subscription count of **221**.

7. To check if marital status mattered for subscription to deposit

```
scala> bankingdf.groupBy("y", "marital").count().show
+---+--------+-----+
|  y| marital|count|
+---+--------+-----+
|yes| married| 2755|
| no| married|24459|
|yes|divorced|  622|
|yes|  single| 1912|
| no|divorced| 4585|
| no|  single|10878|
+---+--------+-----+

scala> val ageMaritalSub = bankingdf.filter($"y" ===
"yes").groupBy("marital","y").count().sort($"coun
t".desc).show
+--------+---+-----+
| marital|  y|count|
+--------+---+-----+
| married|yes| 2755|
|  single|yes| 1912|
|divorced|yes|  622|
+--------+---+-----+
ageMaritalSub: Unit = ()
```

From the above data, we can interpret that married people have the highest subscription with a count of **2755;** while the lowest being the divorced group at **622** and single at 1912.

8. To check if age and marital status together mattered for subscription to deposit scheme

```
scala> val ageMaritalSub = bankingdf.filter($"y" ===
"yes").groupBy("marital","y","age").count().sort(
$"count".desc).show
+-------+---+---+-----+
|marital|  y|age|count|
+-------+---+---+-----+
| single|yes| 30|  151|
```

```
| single|yes| 28|  138|
| single|yes| 29|  133|
| single|yes| 32|  124|
| single|yes| 26|  121|
|married|yes| 34|  118|
| single|yes| 31|  111|
| single|yes| 27|  110|
|married|yes| 35|  101|
|married|yes| 36|  100|
| single|yes| 25|   99|
|married|yes| 37|   98|
|married|yes| 33|   97|
| single|yes| 33|   97|
|married|yes| 32|   87|
|married|yes| 39|   87|
|married|yes| 38|   86|
| single|yes| 35|   84|
|married|yes| 47|   83|
|married|yes| 46|   80|
+-------+---+---+-----+
only showing top 20 rows
ageMaritalSub: Unit = ()
```

From the above data, we can interpret that '**Single'** group of people have the highest count of **151** at the age of **30.** Most of the subscribers are in the age group of **28-32** with a range of **151 to 124.**

9. To do feature engineering for column age and find right age effect on campaign

Import libraries

```
scala> import org.apache.spark.ml.feature.{StringIndexer}
import org.apache.spark.ml.feature.StringIndexer
scala> import org.apache.spark.SparkContext
import org.apache.spark.SparkContext
scala> import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.DataFrame
scala> import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.SQLContext
scala> import org.apache.spark.sql.functions.mean
import org.apache.spark.sql.functions.mean
scala> import org.apache.spark.SparkConf
import org.apache.spark.SparkConf
scala> import scala.reflect.runtime.universe
import scala.reflect.runtime.universe
```

We need to divide age groups into 4 categories. Defining UDF to add new features.

```
scala> val ageRDD = sqlContext.udf.register("ageRDD",(age:Int) => {
     |     if (age < 20)
     |        "Teenager"
     |     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(Int
egerType))
```

Creating new “Age” column

```
scala> val newbankingdf = bankingdf.withColumn("age",ageRDD(bankingdf("age")))
newbankingdf: org.apache.spark.sql.DataFrame = [age: string, 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]
```

Register table newbanking

```
scala> newbankingdf.registerTempTable("newbanking")

scala> val targetAge = newbankingdf.filter($"y" ===
"yes").groupBy("age","y").count().sort($"count".desc).show
+-----------+---+-----+
|        age|  y|count|
+-----------+---+-----+
|Middle Aged|yes| 2601|
|      Young|yes| 1539|
|        Old|yes| 1131|
|   Teenager|yes|   18|
+-----------+---+-----+

targetAge: Unit = ()
```

“Middle Aged” group has the highest subscriptions with a count of **2601** and “Teenager” subscriber’s is as low as 18.

Pipeline processes data inside dataframe

```
scala> val ageInd = new StringIndexer().setInputCol("age").setOutputCol("ageIndex")
ageInd: org.apache.spark.ml.feature.StringIndexer = strIdx_5bc48f93eba0
```

Fit the model

```
scala> var strIndModel = ageInd.fit(newbankingdf)
strIndModel: org.apache.spark.ml.feature.StringIndexerModel = strIdx_5bc48f93eba0
```

StringIndexerModel.transform assigns generated index to each value of the column age in newbanking dataframe.

```
scala> val strIndModel1 =
strIndModel.transform(newbankingdf).select("age","ageIndex").show(7)
+-----------+--------+
|        age|ageIndex|
+-----------+--------+
|        Old|     2.0|
|Middle Aged|     0.0|
|        Old|     2.0|
|Middle Aged|     0.0|
|        Old|     2.0|
|Middle Aged|     0.0|
|      Young|     1.0|
+-----------+--------+
only showing top 7 rows

strIndModel1: Unit = ()
```

"**Middle Aged**" group is most frequent word in this data, so it is given at index "**0**"

```
scala> val strIndModel1 =
strIndModel.transform(newbankingdf).select("age","ageIndex").sort($"ageIndex".desc).show(
7)
+--------+--------+
|     age|ageIndex|
+--------+--------+
|Teenager|     3.0|
|Teenager|     3.0|
|Teenager|     3.0|
|Teenager|     3.0|
|Teenager|     3.0|
|Teenager|     3.0|
|Teenager|     3.0|
+--------+--------+
only showing top 7 rows

strIndModel1: Unit = ()
```

```
scala> val strIndModel1 =
strIndModel.transform(newbankingdf).select("age","ageIndex").sort($"ageIndex".asc).show(2
5)
+-----------+--------+
|        age|ageIndex|
+-----------+--------+
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
+-----------+--------+
only showing top 25 rows

strIndModel1: Unit = ()
```

From the Feature Engineering, we can conclude that **"Middle Aged"** group that is between **33 and 55** should be the "Target Audience" as we can infer from the data they subscribe the most than the rest of the age groups.

Screenshots for reference

```
[radsrinivasan_gmail_com@ip-10-0-1-10 ~]$ spark-shell --conf spark.ui.port=4040 --packages com.databricks:spark-csv_2.11:1.5.0
Ivy Default Cache set to: /home/radsrinivasan_gmail_com/.ivy2/cache
The jars for the packages stored in: /home/radsrinivasan_gmail_com/.ivy2/jars
:: loading settings :: url = jar:file:/opt/cloudera/parcels/CDH-5.14.0-1.cdh5.14.0.p0.24/jars/spark-assembly-1.6.0-cdh5.14.0-ha
doop2.6.0-cdh5.14.0.jar!/org/apache/ivy/core/settings/ivysettings.xml
com.databricks#spark-csv_2.11 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent;1.0
        confs: [default]
        found com.databricks#spark-csv_2.11;1.5.0 in central
        found org.apache.commons#commons-csv;1.1 in central
        found com.univocity#univocity-parsers;1.5.1 in central
:: resolution report :: resolve 287ms :: artifacts dl 4ms
        :: modules in use:
        com.databricks#spark-csv_2.11;1.5.0 from central in [default]
        com.univocity#univocity-parsers;1.5.1 from central in [default]
        org.apache.commons#commons-csv;1.1 from central in [default]
        ---------------------------------------------------------------------
        |                  |            modules            ||   artifacts   |
        |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
        ---------------------------------------------------------------------
        |      default     |   3   |   0   |   0   |   0   ||   3   |   0   |
        ---------------------------------------------------------------------
:: retrieving :: org.apache.spark#spark-submit-parent
        confs: [default]
        0 artifacts copied, 3 already retrieved (0kB/7ms)
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Scala version 2.10.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_144)
Type in expressions to have them evaluated.
Type :help for more information.
Spark context available as sc (master = yarn-client, app id = application_1525768906538_3478).
```

```
scala> val bankingdf = sqlContext.read.format("com.databricks.spark.csv").option("header","true").option("inferSchema","true").
option("delimiter",";").load("/user/radsrinivasan_gmail_com/bankmarketingdata.csv")
bankingdf: org.apache.spark.sql.DataFrame = [age: int, job: string, marital: string, education: string, default: string, balanc
e: int, housing: string, loan: string, contact: string, day: int, month: string, duration: int, campaign: int, pdays: int, prev
ious: int, poutcome: string, y: string]

scala> bankingdf.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|
```

```
scala> val totalCount = bankingdf.count().toDouble
totalCount: Double = 45211.0

scala> val subscriptionCount = bankingdf.filter($"y" === "yes").count().toDouble
subscriptionCount: Double = 5289.0

scala> val successRate = subscriptionCount/totalCount
successRate: Double = 0.11698480458295547

scala> val nonSubscriberCount = bankingdf.filter($"y" === "no").count().toDouble
nonSubscriberCount: Double = 39922.0

scala> val failureRate = nonSubscriberCount/totalCount
failureRate: Double = 0.8830151954170445

scala> bankingdf.select(max($"age"), avg($"age"), min($"age")).show
+--------+-----------------+--------+
|max(age)|         avg(age)|min(age)|
+--------+-----------------+--------+
|      95|40.93621021432837|      18|
+--------+-----------------+--------+


scala> bankingdf.registerTempTable("banking")

scala> sqlContext.sql("select percentile(balance, 0.5) as median, avg(balance) as average from banking").show
+------+------------------+
|median|           average|
+------+------------------+
| 448.0|1362.2720576850766|
+------+------------------+
```

```
scala> bankingdf.groupBy("y", "marital").count().show
+---+--------+-----+
|  y| marital|count|
+---+--------+-----+
|yes| married| 2755|
| no| married|24459|
|yes|divorced|  622|
|yes|  single| 1912|
| no|divorced| 4585|
| no|  single|10878|
+---+--------+-----+
```

```
scala> bankingdf.groupBy("y").agg(avg($"age")).show
+---+------------------+
|  y|          avg(age)|
+---+------------------+
| no| 40.83898602274435|
|yes|41.670069956513515|
+---+------------------+


scala> bankingdf.filter($"y" === "yes").groupBy("y","age").count().sort($"count". desc).show
+---+---+-----+
|  y|age|count|
+---+---+-----+
|yes| 32|  221|
|yes| 30|  217|
|yes| 33|  210|
|yes| 35|  209|
|yes| 31|  206|
|yes| 34|  198|
|yes| 36|  195|
|yes| 29|  171|
|yes| 37|  170|
|yes| 28|  162|
|yes| 38|  144|
|yes| 39|  143|
|yes| 27|  141|
|yes| 26|  134|
|yes| 41|  120|
|yes| 46|  118|
|yes| 40|  116|
|yes| 25|  113|
|yes| 47|  113|
|yes| 42|  111|
+---+---+-----+
only showing top 20 rows

scala> val ageMaritalSub = bankingdf.filter($"y" === "yes").groupBy("marital","y").count().sort($"count".desc).show
+--------+---+-----+
| marital|  y|count|
+--------+---+-----+
| married|yes| 2755|
|  single|yes| 1912|
|divorced|yes|  622|
+--------+---+-----+

ageMaritalSub: Unit = ()

scala> val ageMaritalSub = bankingdf.filter($"y" === "yes").groupBy("marital","y","age").count().sort($"count".desc).show
+-------+---+---+-----+
|marital|  y|age|count|
+-------+---+---+-----+
| single|yes| 30|  151|
| single|yes| 28|  138|
| single|yes| 29|  133|
| single|yes| 32|  124|
| single|yes| 26|  121|
|married|yes| 34|  118|
| single|yes| 31|  111|
| single|yes| 27|  110|
|married|yes| 35|  101|
|married|yes| 36|  100|
| single|yes| 25|   99|
|married|yes| 37|   98|
|married|yes| 33|   97|
| single|yes| 33|   97|
|married|yes| 32|   87|
|married|yes| 39|   87|
|married|yes| 38|   86|
| single|yes| 35|   84|
|married|yes| 47|   83|
|married|yes| 46|   80|
+-------+---+---+-----+
only showing top 20 rows
```

```
scala> import org.apache.spark.ml.feature.{StringIndexer}
import org.apache.spark.ml.feature.StringIndexer

scala> import org.apache.spark.SparkContext
import org.apache.spark.SparkContext

scala> import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.DataFrame

scala> import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.SQLContext

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

scala> import org.apache.spark.SparkConf
import org.apache.spark.SparkConf

scala> import scala.reflect.runtime.universe
import scala.reflect.runtime.universe


scala> import scala.reflect.runtime.universe
import scala.reflect.runtime.universe

scala> val ageRDD = sqlContext.udf.register("ageRDD",(age:Int) => {
     |     if (age < 20)
     |        "Teenager"
     |     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(Int
egerType))

scala> val newbankingdf = bankingdf.withColumn("age",ageRDD(bankingdf("age")))
newbankingdf: org.apache.spark.sql.DataFrame = [age: string, 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> val targetAge = newbankingdf.filter($"y" === "yes").groupBy("age","y").count().sort($"count".desc).show
+-----------+---+-----+
|        age|  y|count|
+-----------+---+-----+
|Middle Aged|yes| 2601|
|      Young|yes| 1539|
|        Old|yes| 1131|
|   Teenager|yes|   18|
+-----------+---+-----+

targetAge: Unit = ()

scala> val ageInd = new StringIndexer().setInputCol("age").setOutputCol("ageIndex")
ageInd: org.apache.spark.ml.feature.StringIndexer = strIdx_5bc48f93eba0

scala> var strIndModel = ageInd.fit(newbankingdf)
strIndModel: org.apache.spark.ml.feature.StringIndexerModel = strIdx_5bc48f93eba0

scala> val strIndModel1 = strIndModel.transform(newbankingdf).select("age","ageIndex").show(7)
+-----------+--------+
|        age|ageIndex|
+-----------+--------+
|        Old|     2.0|
|Middle Aged|     0.0|
|        Old|     2.0|
|Middle Aged|     0.0|
|        Old|     2.0|
|Middle Aged|     0.0|
|      Young|     1.0|
+-----------+--------+
only showing top 7 rows

strIndModel1: Unit = ()
```

```
scala> val strIndModel1 = strIndModel.transform(newbankingdf).select("age","ageIndex").sort($"ageIndex".asc).show(25)
+-----------+--------+
|        age|ageIndex|
+-----------+--------+
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
+-----------+--------+
only showing top 25 rows

strIndModel1: Unit = ()
```