DESCRIPTION

Background and Objective:

Your client, a Portuguese banking institution, ran a marketing campaign to convince potential customers to invest in a bank term deposit scheme. 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. You have to perform the marketing analysis of the data generated by this campaign.

Domain: Banking (Market Analysis)

Dataset Description

The data fields are as follows:

1.	age	numeric
2.	job	type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')
3.	marital	marital status (categorical: 'divorced', 'married', 'single', 'unknown'; note: 'divorced' means divorced or widowed)
4.	education	(categorical: 'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','unknown')
5.	default	has credit in default? (categorical: 'no', 'yes', 'unknown')
6.	housing:	has housing loan? (categorical: 'no', 'yes', 'unknown')
7.	loan	has a personal loan? (categorical: 'no', 'yes', 'unknown')

related to the last contact of the current campaign:

8.	contact	contact communication type (categorical: 'cellular', 'telephone')
9.	month	Month of last contact (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
10.	day_of_week	last contact day of the week (categorical: 'mon','tue','wed','thu','fri')
11.	duration	last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (example, if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call “y” is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

other attributes:

12.	campaign	number of times a customer was contacted during the campaign (numeric, includes last contact)
13.	pdays:	number of days passed after the customer was last contacted from a previous campaign (numeric; 999 means customer was not previously contacted)
14.	previous	number of times the customer was contacted prior to (or before) this campaign (numeric)
15.	poutcome	outcome of the previous marketing campaign (categorical: 'failure', 'nonexistent', 'success')

#Output variable (desired target):

16	y	has the customer subscribed a term deposit? (binary: 'yes', 'no')

Download the Sample Dataset.

Analysis tasks to be done-:

The data size is huge and the marketing team has asked you to perform the below analysis-

1. Load data and create a Spark data frame
2. Give marketing success rate (No. of people subscribed / total no. of entries)

- Give marketing failure rate

1. Give the maximum, mean, and minimum age of the average targeted customer
2. Check the quality of customers by checking average balance, median balance of customers
3. Check if age matters in marketing subscription for deposit
4. Check if marital status mattered for a subscription to deposit
5. Check if age and marital status together mattered for a subscription to deposit scheme
6. Do feature engineering for the bank and find the right age effect on the campaign.

```
 [cloudera@quickstart ~]$ hdfs dfs -copyFromLocal "/home/cloudera/Proyecto1-Spark/Project1_dataset_bank-full.csv" "/Project1-Spark/Project1_dataset_bank-full.csv"


Cloudera@quickstart ~]$ cat Project3.scala | spark-shell
19/04/24 13:45:13 WARN Utils: Your hostname, quickstart.cloudera resolves to a loopback address: 127.0.0.1; using 192.168.71.130 instead (on interface eth1)
19/04/24 13:45:13 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
19/04/24 13:45:15 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://192.168.71.130:4040
Spark context available as 'sc' (master = local[*], app id = local-1556138738716).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.1
      /_/

Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_201)
Type in expressions to have them evaluated.
Type :help for more information.

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

scala> import spark.implicits._
import spark.implicits._

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

scala> import org.apache.spark.sql.types.{StructType, StructField, StringType, IntegerType}
import org.apache.spark.sql.types.{StructType, StructField, StringType, IntegerType}

scala> // 1.     Load data and create a Spark data frame ************************************************************

scala> val spark = SparkSession.builder().appName("Project 1").config("spark.some.config.option", "some-value").getOrCreate()
19/04/24 13:45:54 WARN SparkSession$Builder: Using an existing SparkSession; some configuration may not take effect.
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@c50503e

scala> val customSchema = StructType(Array(
     |     StructField("age", IntegerType, true),
     |     StructField("job", StringType, true),
     |     StructField("marital", StringType, true),
     |     StructField("education", StringType, true),
     |     StructField("default", StringType, true),
     |     StructField("balance", IntegerType, true),
     |     StructField("housing", StringType, true),
     |     StructField("loan", StringType, true),
     |     StructField("contact", StringType, true),
     |     StructField("day", IntegerType, true),
     |     StructField("month", StringType, true),
     |     StructField("duration", IntegerType, true),
     |     StructField("campaign", IntegerType, true),
     |     StructField("pdays", IntegerType, true),
     |     StructField("previous", IntegerType, true),
     |     StructField("poutcome", StringType, true),
     |     StructField("y", StringType, true)
     |     )
     | )

customSchema: org.apache.spark.sql.types.StructType = StructType(StructField(age,IntegerType,true), StructField(job,StringType,true), StructField(marital,StringType,true), StructField(education,StringType,true), StructField(default,StringType,true), StructField(balance,IntegerType,true), StructField(housing,StringType,true), StructField(loan,StringType,true), StructField(contact,StringType,true), StructField(day,IntegerType,true), StructField(month,StringType,true), StructField(duration,IntegerType,true), StructField(campaign,IntegerType,true), StructField(pdays,IntegerType,true), StructField(previous,IntegerType,true), StructField(poutcome,StringType,true), StructField(y,StringType,true))


scala> val df = spark.read.schema(customSchema).option("header", "true").option("delimiter", ";").csv("/home/cloudera/Project1-Spark/Project1_dataset_bank-full2.csv")
df: org.apache.spark.sql.DataFrame = [age: int, job: string ... 15 more fields]

scala> df.createOrReplaceTempView("Bank_table")
```

```
scala> // The data is collected by the contact center of the Portuguese bank to do direct marketing.

scala> // It is a campaign based on phone calls.

scala> // The variable "and" contains "yes" and "no",

scala> // It means if the client subscribes a bank term deposit.

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

scala> val valquestion2 = df.filter($"y" === "yes").count.toFloat / df.count.toFloat *100
valquestion2: Float = 11.698481

scala> // 2a.  Give marketing failure rate ************************************************************************

scala> val failure_rate = df.filter($"y" === "no").count.toFloat / df.count.toFloat *100
failure_rate: Float = 88.30152

scala> // 3.   Maximum, Mean, and Minimum age of average targeted customer ************************************

scala> val question3 = df.select(max($"age"), min($"age"), avg($"age")).show()
+-----------+-----------+-------------------------+
|max(age)|min(age)|       avg(age)           |
+-----------+-----------+-------------------------+
|  95       |   18      |40.93621021432837|
+-----------+-----------+-------------------------+

question3: Unit = ()

scala> // 4.   Check quality of customers by checking average balance, median balance of customers *********************

scala> val question4 = df.sqlContext.sql("select percentile(balance, 0.5) as median, avg(balance) as average from Bank_table").show
19/04/24 13:46:32 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
+----------+---------------------------+
|median|         average           |
+----------+---------------------------+
| 448.0   |1362.2720576850766|
+----------+---------------------------+

question4: Unit = ()

scala> // 5.   Check if age matters in marketing subscription for deposit ***********************************************

scala> val question5 = df.groupBy("y").agg(avg($"age")).show
+----+---------------------------+
|  y|         avg(age)             |
+----+---------------------------+
| no| 40.83898602274435-|
|yes|41.670069956513515|
+----+---------------------------+

question5: Unit = ()

From the output average age for marketing subscription is 41.67

scala> // 6.   Check if marital status mattered for subscription to deposit. ***********************************************

scala> val question6 = df.groupBy("y", "marital").count().show
+----+-----------+--------+
|  y  | marital |count |
+----+-----------+--------+
| no| married|24459|
| no|  single   |10878|
|yes|  single   | 1912 |
|yes| married| 2755 |
| no|divorced| 4585 |
|yes|divorced|  622  |
+----+-----------+--------+

question6: Unit = ()

From this it can be interpreted that married people have the highest subscription with 2755, while the lowest is the group divorced in 622 and single in
1912.


scala> // 7.   Check if age and marital status together mattered for subscription to deposit scheme ************************
```

```
scala> val question7 = df.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| 39|   87   |
|married|yes| 32|   87   |
|married|yes| 38|   86   |
| single   |yes| 35|   84   |
|married|yes| 47|   83   |
|married|yes| 31|   80   |
+----------+-----+----+--------+
only showing top 20 rows

question7: Unit = ()
```

It can be interpreted that the group of people "Single" has the highest count of 151 at the age of 30 years. The majority of subscribers are in the age group of 28-32 with a range of 151
to 124.

```
scala> // 8.   Do feature engineering for column—age and find right age effect on campaign ********************************

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


scala> // To generate the new features we define a UDF, we divide the age groups into 4 categories

scala> val ageRDD = df.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.expressions.UserDefinedFunction = UserDefinedFunction(<function1>,StringType,Some(List(IntegerType)))

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

scala> banknewDF.registerTempTable("Bank_table2")
warning: there was one deprecation warning; re-run with -deprecation for details

scala> // Running a query to see the age group which subscribed the most. We see it's 'Middle-Aged'

scala> val age_target = df.sqlContext.sql("select age, count(*) as number from Bank_table2 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 = ()
```

The group "Middle Aged" has the highest subscriptions with 2601 and "Teen" is the lowest with 18.

SCREENSHOT

cloudera-quickstart-vm-5.13.0-0-vmware - VMware Workstation 15 Player (Non-commercial use only)

Player | Applications Places System | Wed Apr 24, 5:25 PM cloudera

cloudera@quickstart:~

File Edit View Search Terminal Help

```
bash: /usr/lib/jvm/jre-1.8.0-openjdk.x86_64/bin: is a directory
[cloudera@quickstart ~]$ cat Project3.scala | spark-shell
19/04/24 16:50:17 WARN Utils: Your hostname, quickstart.cloudera resolves to a l
oopback address: 127.0.0.1; using 192.168.71.130 instead (on interface eth1)
19/04/24 16:50:17 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another
address
19/04/24 16:50:28 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLeve
l(newLevel).
Spark context Web UI available at http://192.168.71.130:4040
Spark context available as 'sc' (master = local[*], app id = local-1556149854782
).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.1
      /_/

Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_201)
Type in expressions to have them evaluated.
Type :help for more information.

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

scala> import spark.implicits._
import spark.implicits._

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

scala> import org.apache.spark.sql.types.{StructType, StructField, StringType, I
ntegerType}
import org.apache.spark.sql.types.{StructType, StructField, StringType, IntegerT
ype}

scala>


scala>

scala> // 1.     Load data and create a Spark data frame

scala>
```

cloudera@quickstart:~ | cloudera - File Browser | Project3.scala (~) - ge... | cloudera@quickstart:~

Escritorio | ESP ES | 20:25 24/4/2019

cloudera-quickstart-vm-5.13.0-0-vmware - VMware Workstation 15 Player (Non-commercial use only)

Player | Applications Places System | Wed Apr 24, 5:26 PM cloudera

cloudera@quickstart:~

File Edit View Search Terminal Help

```
scala> val spark = SparkSession.builder().appName("Project 1").config("spark.som
e.config.option", "some-value").getOrCreate()
19/04/24 16:51:10 WARN SparkSession$Builder: Using an existing SparkSession; som
e configuration may not take effect.
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@326
489c4

scala>

scala> val customSchema = StructType(Array(
     |     StructField("age", IntegerType, true),
     |     StructField("job", StringType, true),
     |     StructField("marital", StringType, true),
     |     StructField("education", StringType, true),
     |     StructField("default", StringType, true),
     |     StructField("balance", IntegerType, true),
     |     StructField("housing", StringType, true),
     |     StructField("loan", StringType, true),
     |     StructField("contact", StringType, true),
     |     StructField("day", IntegerType, true),
     |     StructField("month", StringType, true),
     |     StructField("duration", IntegerType, true),
     |     StructField("campaign", IntegerType, true),
     |     StructField("pdays", IntegerType, true),
     |     StructField("previous", IntegerType, true),
     |     StructField("poutcome", StringType, true),
     |     StructField("y", StringType, true)
     |     )
     | )
customSchema: org.apache.spark.sql.types.StructType = StructType(StructField(age
,IntegerType,true), StructField(job,StringType,true), StructField(marital,String
Type,true), StructField(education,StringType,true), StructField(default,StringTy
pe,true), StructField(balance,IntegerType,true), StructField(housing,StringType,
true), StructField(loan,StringType,true), StructField(contact,StringType,true),
StructField(day,IntegerType,true), StructField(month,StringType,true), StructFie
ld(duration,IntegerType,true), StructField(campaign,IntegerType,true), StructFie
ld(pdays,IntegerType,true), StructField(previous,IntegerType,true), StructField(
poutcome,StringType,true), StructField(y,StringType,true))

scala>

scala> val df = spark.read.schema(customSchema).option("header", "true").option(
"delimiter", ";").csv("/home/cloudera/Project1-Spark/Project1_dataset_bank-full2
.csv")
df: org.apache.spark.sql.DataFrame = [age: int, job: string ... 15 more fields]

scala>
```

cloudera@quickstart:~ | cloudera - File Browser | Project3.scala (~) - ge... | cloudera@quickstart:~

Escritorio | ESP ES | 20:26 24/4/2019

cloudera-quickstart-vm-5.13.0-0-vmware - VMware Workstation 15 Player (Non-commercial use only)

Player | Applications Places System | Wed Apr 24, 5:27 PM cloudera

cloudera@quickstart:~

File Edit View Search Terminal Help

```
scala> df.createOrReplaceTempView("Bank_table")

scala>


scala>

scala> // The data is collected by the contact center of the Portuguese bank to
do direct marketing.

scala> // It is a campaign based on phone calls.

scala> // The variable "and" contains "yes" and "no",

scala> // It means if the client subscribes a bank term deposit.

scala>


scala>


scala>


scala> // 2.     Give marketing success rate (No. of people subscribed / total n
o. of entries)

scala>

scala> val valquestion2 = df.filter($"y" === "yes").count.toFloat / df.count.toF
loat *100
valquestion2: Float = 11.698481

scala>


scala>

scala> // 2a.  Give marketing failure rate

scala>

scala> val failure_rate = df.filter($"y" === "no").count.toFloat / df.count.toFl
oat *100
failure_rate: Float = 88.30152

scala>


scala>

scala> // 3.   Maximum, Mean, and Minimum age of average targeted customer
```

cloudera@quickstart:~ | cloudera - File Browser | Project3.scala (~) - ge... | cloudera@quickstart:~

Escritorio | ESP ES | 20:27 24/4/2019

cloudera-quickstart-vm-5.13.0-0-vmware - VMware Workstation 15 Player (Non-commercial use only)

Player | Applications Places System | Wed Apr 24, 5:29 PM cloudera

cloudera@quickstart:~

File Edit View Search Terminal Help

```
scala> val question3 = df.select(max($"age"), min($"age"), avg($"age")).show()
+--------+--------+-----------------+
|max(age)|min(age)|         avg(age)|
+--------+--------+-----------------+
|      95|      18|40.93621021432837|
+--------+--------+-----------------+

question3: Unit = ()

scala>


scala>

scala> // 4.   Check quality of customers by checking average balance, median ba
lance of customers

scala>

scala> val question4 = df.sqlContext.sql("select percentile(balance, 0.5) as med
ian, avg(balance) as average from Bank_table").show
19/04/24 16:51:49 WARN ObjectStore: Failed to get database global_temp, returnin
g NoSuchObjectException
+------+------------------+
|median|           average|
+------+------------------+
| 448.0|1362.2720576850766|
+------+------------------+

question4: Unit = ()

scala>


scala>

scala> // 5.   Check if age matters in marketing subscription for deposit

scala> val question5 = df.groupBy("y").agg(avg($"age")).show
+---+------------------+
|  y|          avg(age)|
+---+------------------+
| no| 40.83898602274435|
|yes|41.670069956513515|
+---+------------------+

question5: Unit = ()

scala>
```

cloudera@quickstart:~ | cloudera - File Browser | Project3.scala (~) - ge... | cloudera@quickstart:~

Escritorio | ESP ES | 20:29 24/4/2019

cloudera-quickstart-vm-5.13.0-0-vmware - VMware Workstation 15 Player (Non-commercial use only)

Player | Applications Places System | Wed Apr 24, 5:30 PM cloudera

cloudera@quickstart:~

File Edit View Search Terminal Help

```
scala> // 6.   Check if marital status mattered for subscription to deposit.

scala> val question6 = df.groupBy("y", "marital").count().show
+---+--------+-----+
|  y| marital|count|
+---+--------+-----+
| no| married|24459|
| no|  single|10878|
|yes|  single| 1912|
|yes| married| 2755|
| no|divorced| 4585|
|yes|divorced|  622|
+---+--------+-----+

question6: Unit = ()

scala>

scala> // 7.   Check if age and marital status together mattered for subscriptio
n to deposit scheme

scala> val question7 = df.filter($"y" === "yes").groupBy("marital","y").count().
sort($"count".desc).show
+--------+---+-----+
| marital|  y|count|
+--------+---+-----+
| married|yes| 2755|
|  single|yes| 1912|
|divorced|yes|  622|
+--------+---+-----+

question7: Unit = ()

scala>

scala>

scala> // 8.   Do feature engineering for column—age and find right age effect o
n campaign

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

scala>

scala> // To generate the new features we define a UDF, we divide the age groups
 into 4 categories
```

cloudera@quickstart:~ | cloudera - File Browser | Project3.scala (~) - ge... | cloudera@quickstart:~

Escritorio | ESP ES | 20:30 24/4/2019

cloudera-quickstart-vm-5.13.0-0-vmware - VMware Workstation 15 Player (Non-commercial use only)

Player | Applications Places System | Wed Apr 24, 5:31 PM cloudera

cloudera@quickstart:~

File Edit View Search Terminal Help

Click to view your appointments and tasks

```
scala>

scala> val ageRDD = df.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.expressions.UserDefinedFunction = UserDefinedFuncti
on(<function1>,StringType,Some(List(IntegerType)))

scala>

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

scala>

scala> banknewDF.registerTempTable("Bank_table2")
warning: there was one deprecation warning; re-run with -deprecation for details

scala>

scala> // Running a query to see the age group which subscribed the most. We see
 it's 'Middle-Aged'

scala> val age_target = df.sqlContext.sql("select age, count(*) as number from B
ank_table2 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 = ()

scala>

scala>
```

cloudera@quickstart:~ | cloudera - File Browser | Project3.scala (~) - ge... | cloudera@quickstart:~

Escritorio | ESP ES | 20:31 24/4/2019