Big Data Hadoop and Spark Developer

Project - Market Analysis in Banking Domain

Tools used: Simplilearn Lab (Webconsole & Spark Shell Scala, Hue)

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):

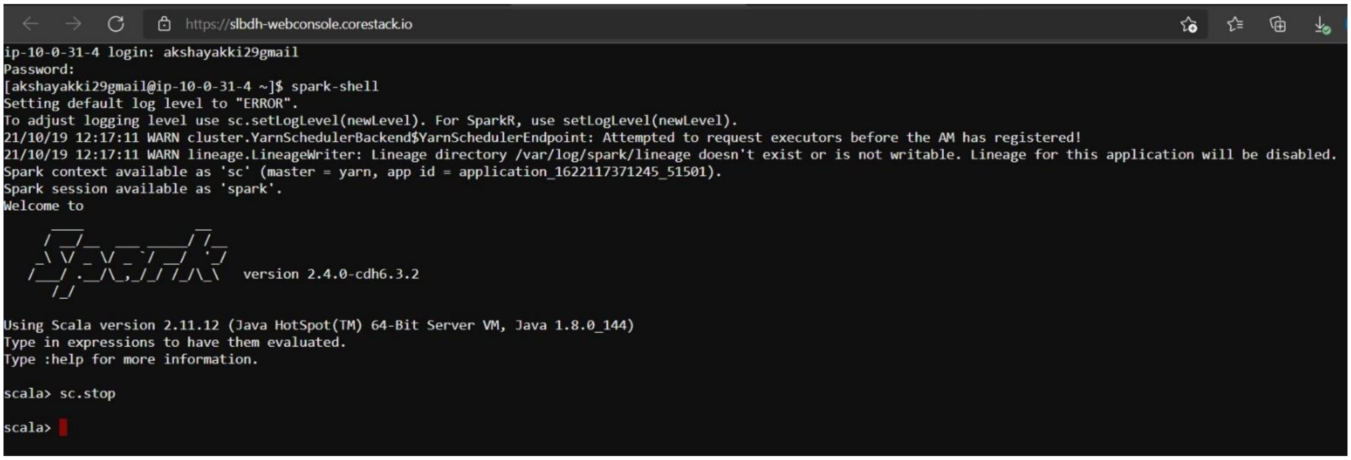
16	y	has the customer subscribed a term deposit? (binary: 'yes', 'no')

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**

Solution: First log into the Webconsole(open 2 tabs) with your given id and password. In one webconsole type spark-shell to open scala environment. Also use sc.stop to stop any sparkcontext running which will otherwise throw an error while execution. See below screenshot:



```
ip-10-0-31-4 login: akshayakki29gmail
Password:
[akshayakki29gmail@ip-10-0-31-4 ~]$ spark-shell
Setting default log level to "ERROR".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
21/10/19 12:17:11 WARN cluster.YarnSchedulerBackend$YarnSchedulerEndpoint: Attempted to request executors before the AM has registered!
21/10/19 12:17:11 WARN lineage.LineageWriter: Lineage directory /var/log/spark/lineage doesn't exist or is not writable. Lineage for this application will be disabled.
Spark context available as 'sc' (master = yarn, app id = application_1622117371245_51501).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.0-cdh6.3.2
      /_/

Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_144)
Type in expressions to have them evaluated.
Type :help for more information.

scala> sc.stop

scala>
```

Open Hue and create a folder BankData and upload the given csv file into it. Also, before uploading the file open it in excel and replace ‘ “ ‘ as it is not necessary and might throw error. Now in Webconsole type nano.BankProject.scala and hit enter. It will open a window where we can create our Spark Data frame and write the necessary commands to execute the project.

First import all the required packages and create an object BankData and load the csv file and show the data using following commands:

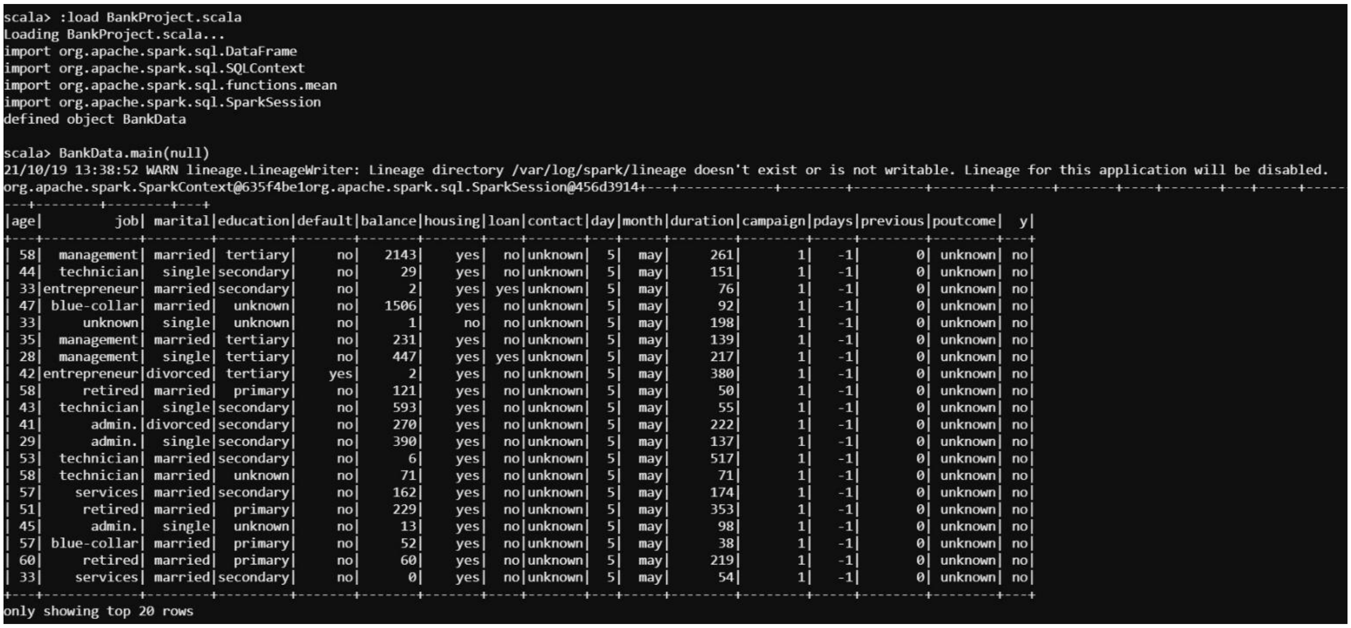
```
import org.apache.spark.sql.DataFrame

import org.apache.spark.sql.SQLContext

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

import org.apache.spark.sql.SparkSession

object BankData {

    def main(args: Array[String]): Unit = {

        val spark: SparkSession = SparkSession.builder().master("local[4]").appName("Spark SQL Session").getOrCreate()

        val sc = spark.sparkContext print(sc) print(spark)

        // Eliminate log unnecessary values while executing

        spark.sparkContext.setLogLevel("ERROR")

        // Analysis tasks to be done

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

        val bank_data = spark.read.option("header","true").option("delimiter",";").option("inferschema","true").csv("/user/akshayakki29gmail/BankData/ Project 1_dataset_bank-full.csv")

        bank_data.show()

        }

}
```

Now press Ctrl+x > y > Enter to save the scala file. Now in Spark Shell Scala write below commands and hit Enter after each one to see our output:

:load BankProject.scala

BankData.main(null)

See the below screenshot:



```
scala> :load BankProject.scala
Loading BankProject.scala...
import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.functions.mean
import org.apache.spark.sql.SparkSession
defined object BankData

scala> BankData.main(null)
21/10/19 13:38:52 WARN lineage.LineageWriter: Lineage directory /var/log/spark/lineage doesn't exist or is not writable. Lineage for this application will be disabled.
org.apache.spark.SparkContext@635f4be1org.apache.spark.sql.SparkSession@456d3914+---+------------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+--
---+--------+--------+---+
|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)**

   Give marketing failure rate

Solution: To answer this question we will use below commands:

val totalCount = bank_data.count().toDouble

println("Total entries are",totalCount)

val subscribed = bank_data.filter($"y" === "yes").count().toDouble

println("No. of people subscribed are",subscribed)

// Success Rate

val success_rate = (subscribed / totalCount) * 100

println("The success rate is",success_rate)

val not_subscribed = bank_data.filter($"y" === "no").count().toDouble

println("No. of people not subscribed are",not_subscribed)

// Failure Rate

val failure_rate = (not_subscribed / totalCount) * 100

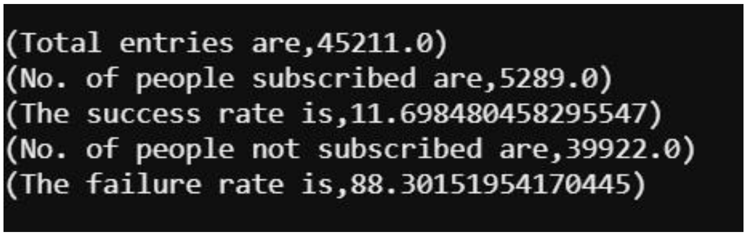
println("The failure rate is",failure_rate)

Now press Ctrl+x > y > Enter to save the scala file. Now in Spark Shell Scala write below commands and hit Enter after each one to see our output:

:load BankProject.scala

BankData.main(null)

See the below screenshot:



```
(Total entries are,45211.0)
(No. of people subscribed are,5289.0)
(The success rate is,11.698480458295547)
(No. of people not subscribed are,39922.0)
(The failure rate is,88.30151954170445)
```

3. **Give the maximum, mean, and minimum age of the average targeted customer**

Solution: To answer this question first we will create a temp view "banking" and then we will use sql statement to get min, avg and max age as given below:

bank_data.createOrReplaceTempView("banking")

println("The minimum, average and maximum age is given below :")

val MinAvgMaxAge = spark.sql("select min(age), avg(age), max(age) from banking")

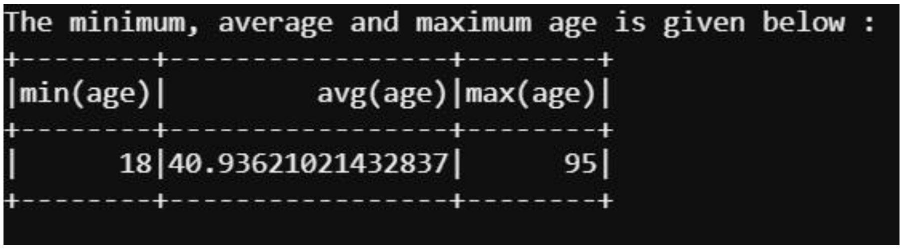
MinAvgMaxAge.show()

Now press Ctrl+x > y > Enter to save the scala file. Now in Spark Shell Scala write below commands and hit Enter after each one to see our output:

:load BankProject.scala

BankData.main(null)

See the below screenshot:



```
The minimum, average and maximum age is given below :
+--------+-----------------+--------+
|min(age)|         avg(age)|max(age)|
+--------+-----------------+--------+
|      18|40.93621021432837|      95|
+--------+-----------------+--------+
```

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

Solution: Use following sql statements to get answer to this question:

println("The Average and Median balance of customers is : ")

val AvgMedBal = spark.sql("select avg(balance), percentile_approx(balance, 0.5) from banking")

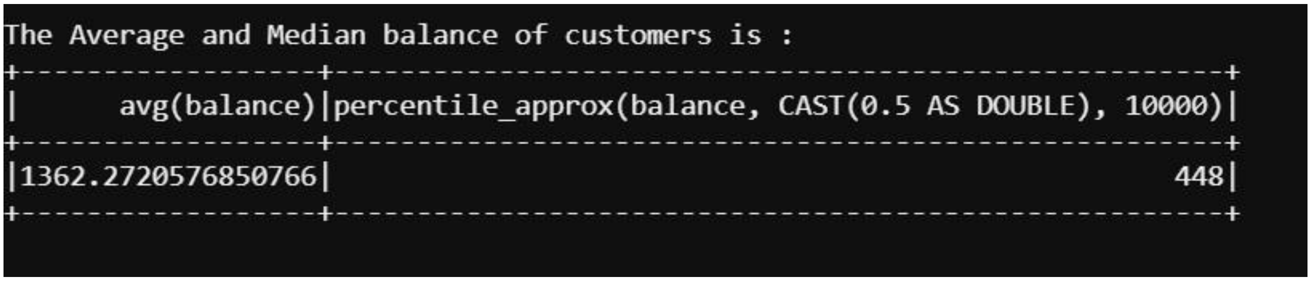
AvgMedBal.show()

Now press Ctrl+x > y > Enter to save the scala file. Now in Spark Shell Scala write below commands and hit Enter after each one to see our output:

:load BankProject.scala

BankData.main(null)

See the below screenshot:



```
The Average and Median balance of customers is :
+-----------------+-----------------------------------------------------+
|     avg(balance)|percentile_approx(balance, CAST(0.5 AS DOUBLE), 10000)|
+-----------------+-----------------------------------------------------+
|1362.2720576850766|                                                 448|
+-----------------+-----------------------------------------------------+
```

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

Solution: Use below sql statement to get answer for above question:

println("The number of people by age of customers who subscribed are given below :")

val agedata = spark.sql("select age, count(*) as count_age from banking where y = 'yes' group by age order by count_age desc")

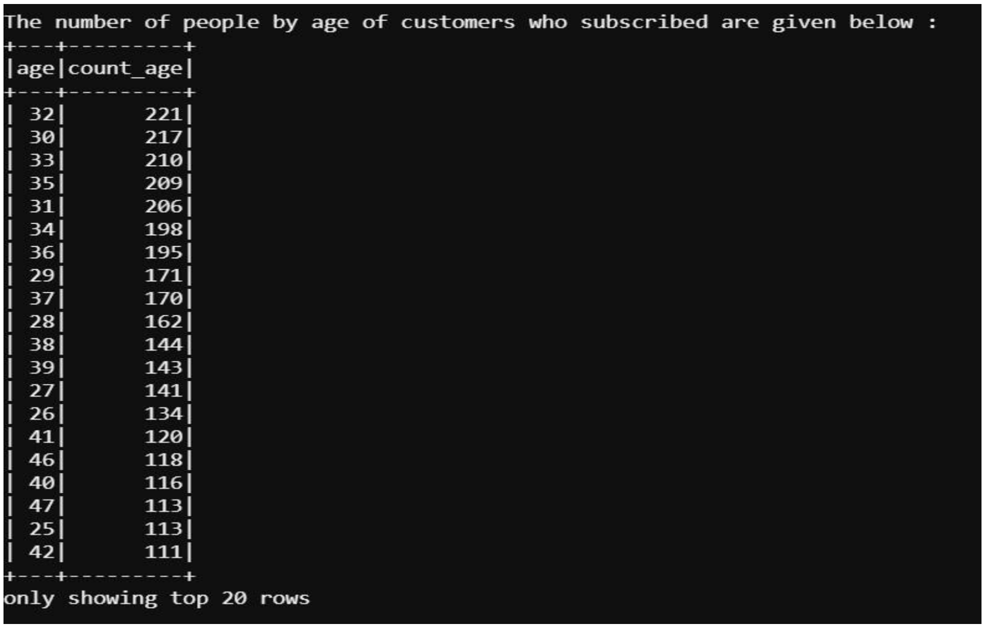
agedata.show()

Now press Ctrl+x > y > Enter to save the scala file. Now in Spark Shell Scala write below commands and hit Enter after each one to see our output:

:load BankProject.scala

BankData.main(null)

See the below screenshot:



```
The number of people by age of customers who subscribed are given below :
+---+---------+
|age|count_age|
+---+---------+
| 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
```

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

Solution: Use below sql statement to get answer for above question:

println("Marital status wise count is as below :")

val maritaldata = spark.sql("select marital, count(*) as count_marital from banking where y = 'yes' group by marital order by count_marital desc")

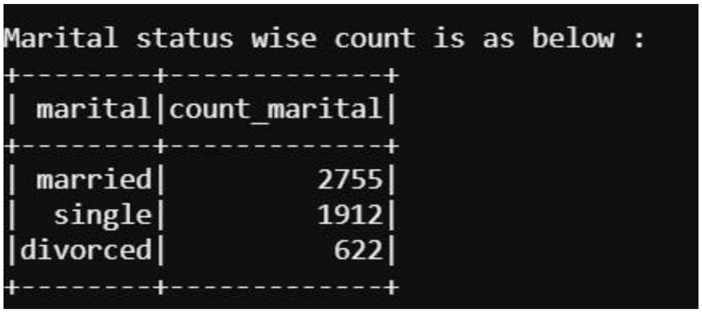
maritaldata.show()

Now press Ctrl+x > y > Enter to save the scala file. Now in Spark Shell Scala write below commands and hit Enter after each one to see our output:

:load BankProject.scala

BankData.main(null)

See the below screenshot:



```
Marital status wise count is as below :
+--------+-------------+
| marital|count_marital|
+--------+-------------+
| married|         2755|
|  single|         1912|
|divorced|          622|
+--------+-------------+
```

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

Solution: Use below sql statement to get answer for above question:

println("Age and Marital status wise count of people who subscribed :")

val ageMarital = spark.sql("select age, marital, count(*) as count from banking where y = 'yes' group by age, marital order by count desc")

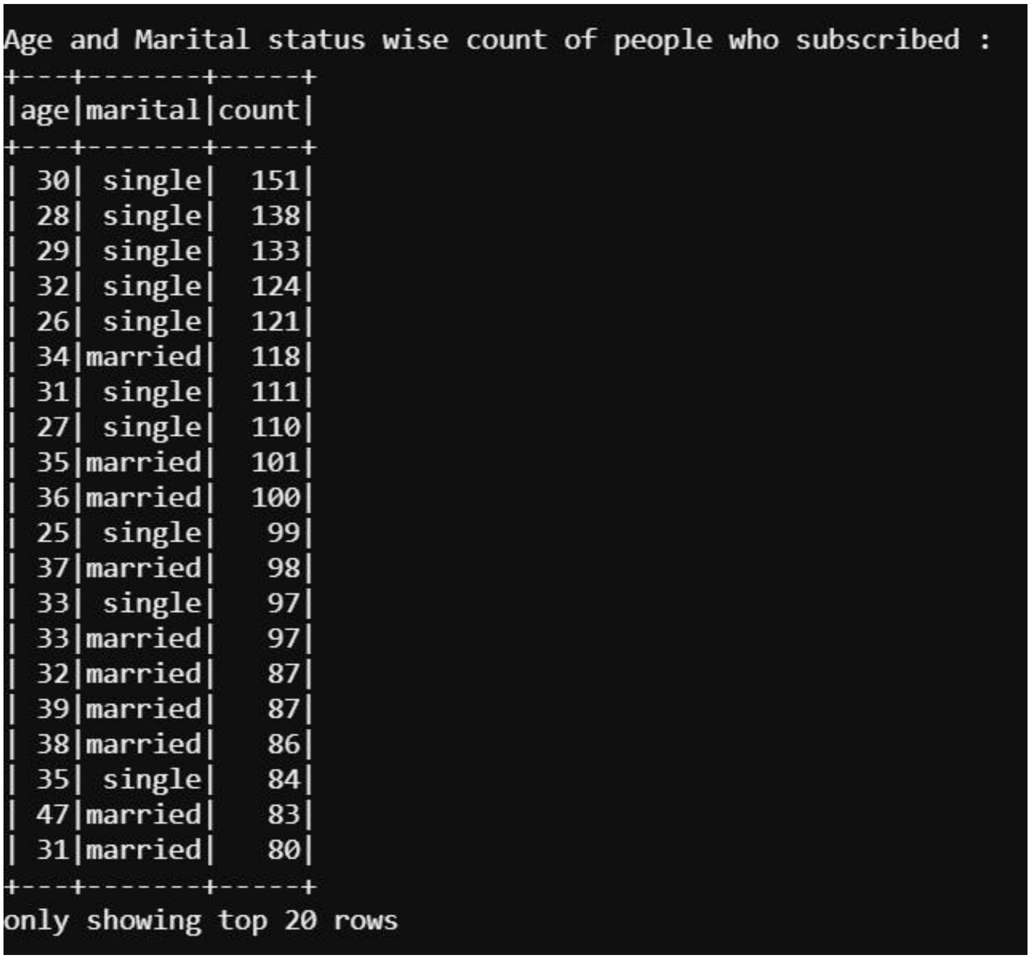
ageMarital.show()

Now press Ctrl+x > y > Enter to save the scala file. Now in Spark Shell Scala write below commands and hit Enter after each one to see our output:

:load BankProject.scala

BankData.main(null)

See the below screenshot:



```
Age and Marital status wise count of people who subscribed :
+---+-------+-----+
|age|marital|count|
+---+-------+-----+
| 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|
| 31|married|   80|
+---+-------+-----+
only showing top 20 rows
```

8. **Do feature engineering for the bank and find the right age effect on the campaign.**

Solution: Use below sql statement to get answer for above question:

```
val ageEffect = spark.udf.register("agedata",(age:Int) => {
if (age < 20)
"Teen"
else if (age >= 20 && age <= 32)
"Young"
else if (age >= 32 && age <= 55)
"Middle Aged"
else
"Old"
})
//Replacing the old age column with the new age column
val banknewDF = bank_data.withColumn("age",ageEffect(bank_data("age")))
```

banknewDF.show()

banknewDF.registerTempTable("banknewtable")

//which age group subscribed the most

val targetage = spark.sql("select age, count(*) as number from banknewtable where y='yes' group by age order by number desc")

targetage.show()

//Feature Engineering: This is to convert categorical age to Discrete Values

import org.apache.spark.ml.feature.StringIndexer

// Pipeline with string Indexer

val agedata2 = new StringIndexer().setInputCol("age").setOutputCol("ageindex")

// Fitting the model val stringModel = agedata2.fit(banknewDF)

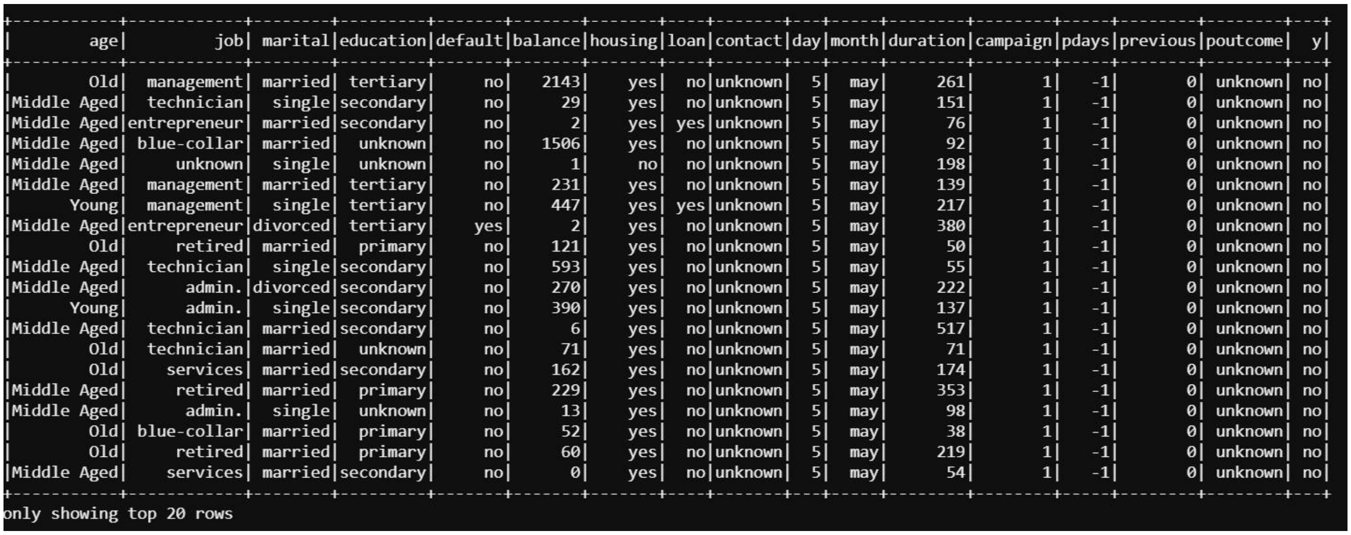
// assign generated values of label of the column by feature engineering stringModel.transform(banknewDF).select("age", "ageIndex").show(5)

Now press Ctrl+x > y > Enter to save the scala file. Now in Spark Shell Scala write below commands and hit Enter after each one to see our output:

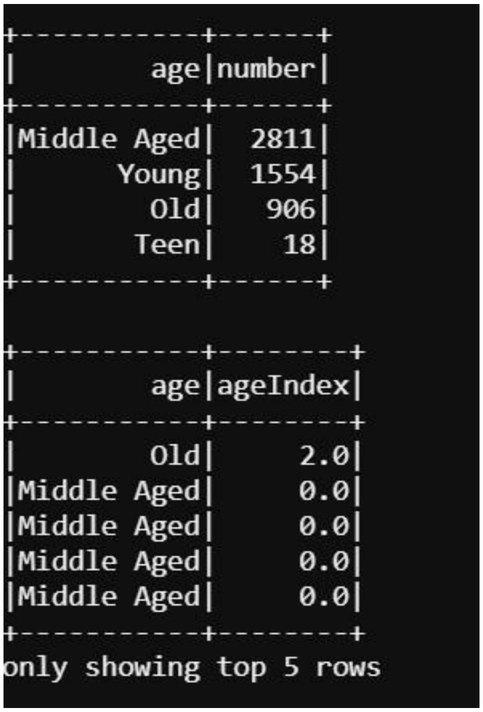
:load BankProject.scala

BankData.main(null)

See the below screenshot:



```
+-----------+------------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+---+
|        age|         job| marital|education|default|balance|housing|loan|contact|day|month|duration|campaign|pdays|previous|poutcome|  y|
+-----------+------------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+---+
|        Old|  management| married| tertiary|     no|   2143|    yes|  no|unknown|  5|  may|     261|       1|   -1|       0| unknown| no|
|Middle Aged|  technician|  single|secondary|     no|     29|    yes|  no|unknown|  5|  may|     151|       1|   -1|       0| unknown| no|
|Middle Aged|entrepreneur| married|secondary|     no|      2|    yes| yes|unknown|  5|  may|      76|       1|   -1|       0| unknown| no|
|Middle Aged| blue-collar| married|  unknown|     no|   1506|    yes|  no|unknown|  5|  may|      92|       1|   -1|       0| unknown| no|
|Middle Aged|     unknown|  single|  unknown|     no|      1|     no|  no|unknown|  5|  may|     198|       1|   -1|       0| unknown| no|
|Middle Aged|  management| married| tertiary|     no|    231|    yes|  no|unknown|  5|  may|     139|       1|   -1|       0| unknown| no|
|      Young|  management|  single| tertiary|     no|    447|    yes| yes|unknown|  5|  may|     217|       1|   -1|       0| unknown| no|
|Middle Aged|entrepreneur|divorced| tertiary|    yes|      2|    yes|  no|unknown|  5|  may|     380|       1|   -1|       0| unknown| no|
|        Old|     retired| married|  primary|     no|    121|    yes|  no|unknown|  5|  may|      50|       1|   -1|       0| unknown| no|
|Middle Aged|  technician|  single|secondary|     no|    593|    yes|  no|unknown|  5|  may|      55|       1|   -1|       0| unknown| no|
|Middle Aged|      admin.|divorced|secondary|     no|    270|    yes|  no|unknown|  5|  may|     222|       1|   -1|       0| unknown| no|
|      Young|      admin.|  single|secondary|     no|    390|    yes|  no|unknown|  5|  may|     137|       1|   -1|       0| unknown| no|
|Middle Aged|  technician| married|secondary|     no|      6|    yes|  no|unknown|  5|  may|     517|       1|   -1|       0| unknown| no|
|        Old|  technician| married|  unknown|     no|     71|    yes|  no|unknown|  5|  may|      71|       1|   -1|       0| unknown| no|
|        Old|    services| married|secondary|     no|    162|    yes|  no|unknown|  5|  may|     174|       1|   -1|       0| unknown| no|
|Middle Aged|     retired| married|  primary|     no|    229|    yes|  no|unknown|  5|  may|     353|       1|   -1|       0| unknown| no|
|Middle Aged|      admin.|  single|  unknown|     no|     13|    yes|  no|unknown|  5|  may|      98|       1|   -1|       0| unknown| no|
|        Old| blue-collar| married|  primary|     no|     52|    yes|  no|unknown|  5|  may|      38|       1|   -1|       0| unknown| no|
|        Old|     retired| married|  primary|     no|     60|    yes|  no|unknown|  5|  may|     219|       1|   -1|       0| unknown| no|
|Middle Aged|    services| married|secondary|     no|      0|    yes|  no|unknown|  5|  may|      54|       1|   -1|       0| unknown| no|
+-----------+------------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+---+
only showing top 20 rows
```



```
+-----------+------+
|        age|number|
+-----------+------+
|Middle Aged|  2811|
|      Young|  1554|
|        Old|   906|
|       Teen|    18|
+-----------+------+


+-----------+--------+
|        age|ageIndex|
+-----------+--------+
|        Old|     2.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
|Middle Aged|     0.0|
+-----------+--------+
only showing top 5 rows
```

Conclusion:

- We used the csv file given and created a Spark Dataframe.
- The marketing success rate is 11.69% and failure rate is 88.30%.
- The maximum, mean, and minimum age of the average targeted customer is: 95 years, 40.93 years and 18 years respectively.
- The average balance, median balance of customers is: 1362.27 and 448 respectively.
- The people between the age 26 and 39 are the highest subscribers considering age.
- The married (2755) and single (1912) are the highest subscribers considering marital status.
- The single between age 26 to 30 and married between age 34 to 37 are highest subscribers considering both age and marital status.
- The middle aged and young people are the highest subscribers than old and teen people. So the right people to target are middle aged and young.