|  |  |
| --- | --- |
|  | Market Analysis in Banking System |
|  |  |
|  | Margil Shah  Big Data Hadoop and Spark Developer  4/19/21 |

Source Code – Full Project

Contents

[DESCRIPTION 2](#_Toc69757939)

[Background and Objective 2](#_Toc69757940)

[Domain 2](#_Toc69757941)

[Dataset Description 2](#_Toc69757942)

[Analysis tasks to be done 3](#_Toc69757943)

[Analysis 1 3](#_Toc69757944)

[Analysis 2 4](#_Toc69757945)

[Analysis 3 4](#_Toc69757946)

[Analysis 4 5](#_Toc69757947)

[Analysis 5 5](#_Toc69757948)

[Analysis 6 6](#_Toc69757949)

[Analysis 7 6](#_Toc69757950)

[Analysis 8 7](#_Toc69757951)

# 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.

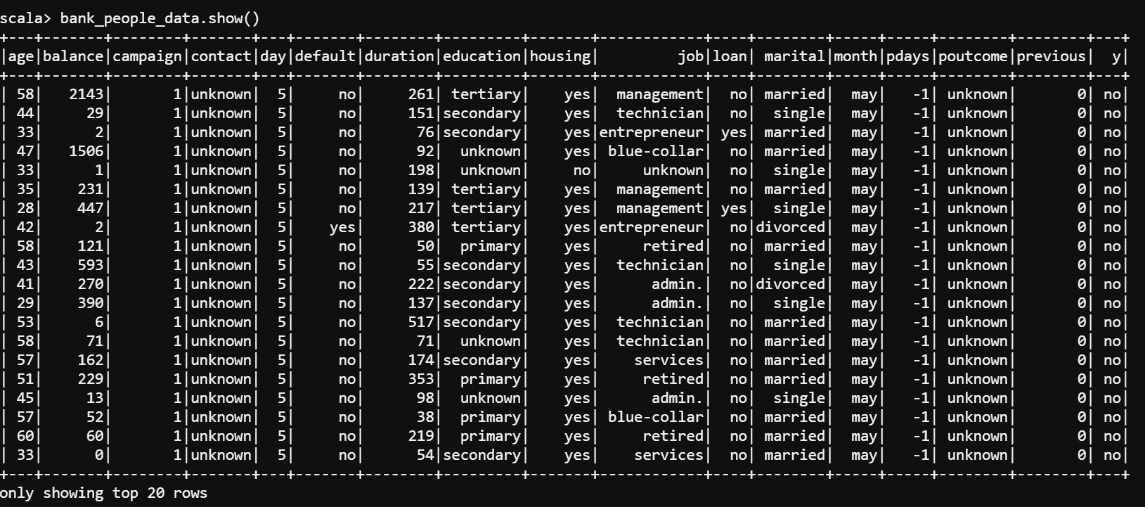
Let’s Launch the PRACTICE LABS

Step1: Upload the bank\_edited.json using HUE under **/user/shahmargilgmail/**

Analysis 1: Load data and create a Spark data frame

Solution:

1. Upload the **bank\_edited.json** using HUE under **/user/shahmargilgmail/**
2. Login to **Webconsole** using credentials
3. Run spark using **spark-shell**
4. import org.apache.spark.ml.feature.StringIndexer
5. val bank\_people\_data = spark.read.option("multiline","true").json("/user/shahmargilgmail/bank\_edited.json");
6. bank\_people\_data.show()



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

* Give marketing failure rate

**Success Rate**

val subscribe = spark.sql("select count(\*) as number from datanewtable where y='yes'")

subscribe: org.apache.spark.sql.DataFrame = [number: bigint]

val total = spark.sql("select count(\*) as number from datanewtable")

total: org.apache.spark.sql.DataFrame = [number: bigint]

subscribe.show()

+------+

|number|

+------+

| 5289|

+------+

total.show()

+------+

|number|

+------+

| 45211|

+------+

**Success Rate =** 5289 / 45211 = 11.6984 %

**Failure Rate =** 100 – 11.6984 = 88.3015%

Analysis 3: Give the maximum, mean, and minimum age of the average targeted customer

bank\_people\_data.select(max($"age")).show()

+--------+

|max(age)|

+--------+

| 95|

+--------+

bank\_people\_data.select(mean($"age")).show()

+-----------------+

| avg(age)|

+-----------------+

|40.93621021432837|

+-----------------+

bank\_people\_data.select(min($"age")).show()

+--------+

|min(age)|

+--------+

| 18|

+--------+

Analysis 4: Check the quality of customers by checking average balance, median balance of customers

bank\_people\_data.select(avg($"balance")).show()

+------------------+

| avg(balance)|

+------------------+

|1362.2720576850766|

+------------------+

bank\_people\_data.createOrReplaceTempView("datanewtable")

val median = spark.sql("SELECT percentile\_approx(balance,0.5) FROM datanewtable").show()

+------------------------------------------------------+

|percentile\_approx(balance, CAST(0.5 AS DOUBLE), 10000)|

+------------------------------------------------------+

| 448|

+------------------------------------------------------+

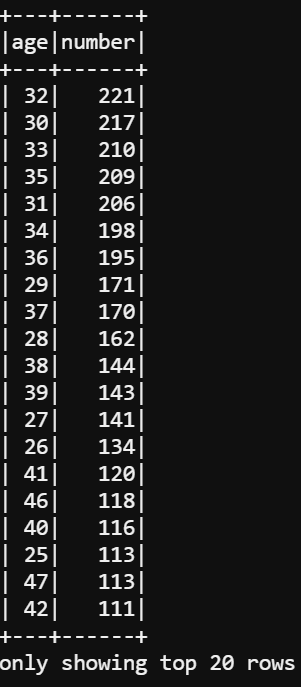
median: Unit = ()

Analysis 5: Check if age matters in marketing subscription for deposit

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

agedata: org.apache.spark.sql.DataFrame = [age: bigint, number: bigint]

agedata.show()



Analysis 6: Check if marital status mattered for a subscription to deposit

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

maritaldata: org.apache.spark.sql.DataFrame = [marital: string, number: bigint]

maritaldata.show()

+--------+------+

| marital|number|

+--------+------+

| married| 2755|

| single| 1912|

|divorced| 622|

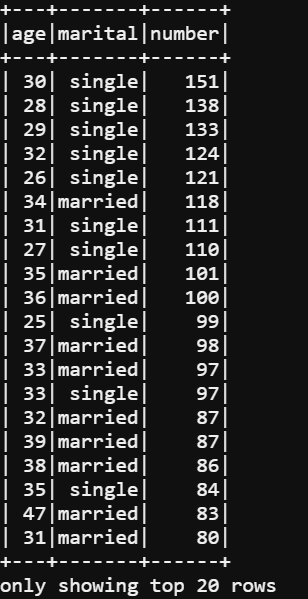
+--------+------+

Analysis 7: Check if age and marital status together mattered for a subscription to deposit scheme

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

ageandmaritaldata: org.apache.spark.sql.DataFrame = [age: bigint, marital: string ... 1 more field]

ageandmaritaldata.show()



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

val agedata = spark.udf.register("agedata",(age:Int) => {

if (age<20)

"Teen"

else if (age>20 && age<=32)

"Young"

else if (age>33 && age <=55)

"Middle Aged"

else

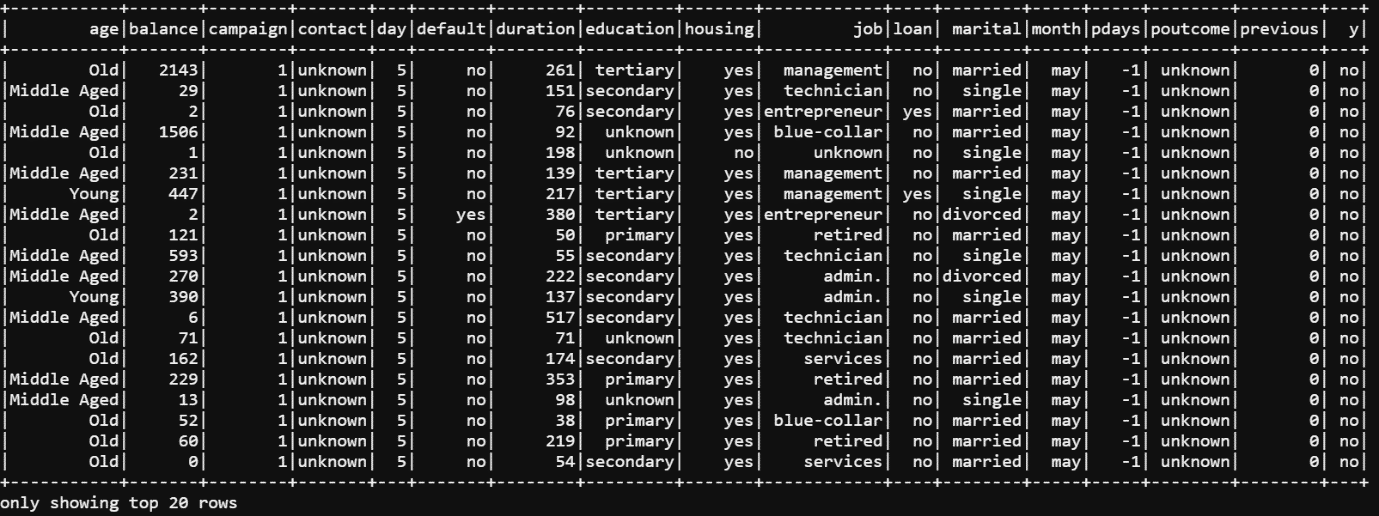
"Old"

})

val banknewDF = bank\_people\_data.withColumn("age",agedata(bank\_people\_data("age")))

banknewDF: org.apache.spark.sql.DataFrame = [age: string, balance: bigint … 15 more fields]

banknewDF.show()



banknewDF.registerTempTable("banknewtable")

warning: there was one deprecation warning; re-run with -deprecation for details

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

targetage: org.apache.spark.sql.DataFrame = [age: string, number: bigint]

targetage.show()

+-----------+------+

| age|number|

+-----------+------+

|Middle Aged| 2601|

| Young| 1539|

| Old| 1131|

| Teen| 18|

+-----------+------+

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

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

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

agedata2: org.apache.spark.ml.feature.StringIndexer = strIdx\_fde4682da0fa

var strindModel = agedata2.fit(banknewDF)

strindModel: org.apache.spark.ml.feature.StringIndexerModel = strIdx\_fde4682da0fa

strindModel.transform(banknewDF).select(“age”,”ageIndex”).show(5)

+---------------+------------+

| age|ageIndex|

+---------------+------------+

| old| 2.0 |

|Middle Aged| 0.0|

| old| 2.0 |

|Middle Aged| 0.0|

| old| 2.0 |

+---------------+------------+

Only showing top 5 rows