

## **PROJECT: MARKETING ANALYSIS IN BANKING DOMAIN**

**THARAKA RAM MADDINENI**

### **# UPLOAD THE DATASET INTO HDFS STORAGE**

```
hdfs -copyFromLocal Project_1_dataset_bank-full.csv /user/tharakarammadditigeranaly/Spark-Project/DataSets
```

### **# LOADING THE DATA**

```
lines = sc.textFile("/user/tharakarammadditigeranaly/Spark-Project/DataSets/Project_1_dataset_bank-full.csv")
```

```
header = lines.first()
```

### **# DATA PRE-PROCESSING**

```
bank = lines.filter(lambda line: line != header).map(lambda p: p.split(";")).map(lambda x: (int(x[0].strip("\n")),
```

```
x[1].strip("\n"),
```

```
x[2].strip("\n"),
```

```
x[3].strip("\n"),
```

```
x[4].strip("\n"),
```

```
float(x[5].strip("\n")),
```

```
x[6].strip("\n"),
```

```
x[7].strip("\n"),
```

```
x[8].strip("\n"),
```

```
int(x[9].strip("\n")),
```

```
x[10].strip("\n"),
```

```
int(x[11].strip("\n")),
```

```
int(x[12].strip("\n")),
```

```
int(x[13].strip("\n")),
```

```
int(x[14].strip("\n")),
```

```
x[15].strip("\\"),  
x[16].strip("\\")))
```

---

## # ADDING CUSTOM SCHEMA (STRUCT TYPE)

---

```
from pyspark.sql.types import *  
  
schemaForData = StructType([  
    StructField("age", IntegerType(), True),  
    StructField("job", StringType(), True),  
    StructField("marital", StringType(), True),  
    StructField("education", StringType(), True),  
    StructField("default", StringType(), True),  
    StructField("balance", DoubleType(), 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)])
```

---

## #1 CREATING DATA FRAME

---

```
bankDF = spark.createDataFrame( bank, schema = schemaForData)
```

```
>>> bankDF.show(10)
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
58	management	married	tertiary	no	2143.0	yes	no	unknown	5	may	261	1	-1	0	unknown	no	
44	technician	single	secondary	no	29.0	yes	no	unknown	5	may	151	1	-1	0	unknown	no	
33	entrepreneur	married	secondary	no	2.0	yes	yes	unknown	5	may	76	1	-1	0	unknown	no	
47	blue-collar	married	unknown	no	1506.0	yes	no	unknown	5	may	92	1	-1	0	unknown	no	
33	unknown	single	unknown	no	1.0	no	no	unknown	5	may	198	1	-1	0	unknown	no	
35	management	married	tertiary	no	231.0	yes	no	unknown	5	may	139	1	-1	0	unknown	no	
28	management	single	tertiary	no	447.0	yes	yes	unknown	5	may	217	1	-1	0	unknown	no	
42	entrepreneur	divorced	tertiary	yes	2.0	yes	no	unknown	5	may	380	1	-1	0	unknown	no	
58	retired	married	primary	no	121.0	yes	no	unknown	5	may	50	1	-1	0	unknown	no	
43	technician	single	secondary	no	593.0	yes	no	unknown	5	may	55	1	-1	0	unknown	no	

only showing top 10 rows

## # Data Frame Schema

```
>>> bankDF.printSchema()
root
|-- age: integer (nullable = true)
|-- job: string (nullable = true)
|-- marital: string (nullable = true)
|-- education: string (nullable = true)
|-- default: string (nullable = true)
|-- balance: double (nullable = true)
|-- housing: string (nullable = true)
|-- loan: string (nullable = true)
|-- contact: string (nullable = true)
|-- day: integer (nullable = true)
|-- month: string (nullable = true)
|-- duration: integer (nullable = true)
|-- campaign: integer (nullable = true)
|-- pdays: integer (nullable = true)
|-- previous: integer (nullable = true)
|-- poutcome: string (nullable = true)
|-- y: string (nullable = true)
```

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

```
success_rate = float(bankDF.filter(bankDF.y=="yes").count()) / float(bankDF.count())
```

```
>>> success_rate = float(bankDF.filter(bankDF.y=="yes").count()) / float(bankDF.count())
>>> success_rate
0.11698480458295547
```

### b. Give marketing failure rate

```
failure_rate = float(bankDF.filter(bankDF.y=="no").count()) / float(bankDF.count())
```

```
>>> failure_rate = float(bankDF.filter(bankDF.y=="no").count()) / float(bankDF.count())
>>> failure_rate
0.8830151954170445
>>> []
```

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

```
bankDF.select('age').summary(["min","max","mean"]).show()
```

```
>>> bankDF.select('age').summary(["min","max","mean"]).show()
+-----+-----+
|summary|          age|
+-----+-----+
|    min|           18|
|    max|           95|
|   mean|40.93621021432837|
+-----+-----+
```

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

```
bankDF.select("balance").summary(["mean","50%"]).show()
```

```
>>> bankDF.select("balance").summary(["mean","50%"]).show()
+-----+-----+
|summary|          balance|
+-----+-----+
|   mean|1362.2720576850766|
|   50%|           448.0|
+-----+-----+
```

#### #5 Check if age matters in marketing subscription for deposit

a. Using sparksql:

```
spark.sql("select age,count(*) as subscribers from bankTable where y='yes' group by age
order by subscribers desc").show()
```

b. Using Data Frame functions:

```
bankDF.filter(bankDF.y=='yes').groupBy('age').count().orderBy('age').show()
```

```
+---+-----+
|age|subscribers|
+---+-----+
| 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|count|
+---+-----+
| 18|     7|
| 19|    11|
| 20|    15|
| 21|    22|
| 22|    40|
| 23|    44|
| 24|    68|
| 25|   113|
| 26|   134|
| 27|   141|
| 28|   162|
| 29|   171|
| 30|   217|
| 31|   206|
| 32|   221|
| 33|   210|
| 34|   198|
| 35|   209|
| 36|   195|
| 37|   170|
+---+-----+
only showing top 20 rows
```

**Note:** Yes, subscription count is more from the customer of age in **30s**.

## #6 Check if marital status mattered for a subscription to deposit

a. **Using sparksql:**

```
spark.sql("select marital,count(*) as subscribers from bankTable where y='yes' group by marital order by subscribers desc").show()
```

b. **Using Data Frame functions:**

```
bankDF.filter(bankDF.y=='yes').groupBy('marital').count().show()
```

```
>>> spark.sql("select marital,
sc").show()
+-----+-----+
| marital|subscribers|
+-----+-----+
| married|      2755|
| single|      1912|
| divorced|      622|
+-----+-----+
```

```
>>> bankDF.filter(bankD
+-----+-----+
| marital|count|
+-----+-----+
| divorced|    622|
| married|   2755|
| single|   1912|
+-----+-----+
```

**Note:** Most subscriptions came from “married” customers followed by “single” marital customers.

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

a. **Using sparksql:**

```
spark.sql("select marital,age,count(*) as subscribers from bankTable where y='yes' group by age,marital order by subscribers desc").show()
```

b. **Using Data Frame functions:**

```
bankDF.filter(bankDF.y=='yes').groupBy('marital','age').count().show()
```

```
+-----+-----+-----+
| marital|age|subscribers|
+-----+-----+-----+
| single| 30|      151|
| single| 28|      138|
| single| 29|      133|
| single| 32|      124|
| single| 26|      121|
| married| 34|      118|
| single| 31|      111|
| single| 27|      110|
| married| 35|      101|
| married| 36|      100|
| single| 25|       99|
| married| 37|       98|
| single| 33|       97|
| married| 33|       97|
| married| 32|       87|
| married| 39|       87|
| married| 38|       86|
| single| 35|       84|
| married| 47|       83|
| married| 31|       80|
+-----+-----+-----+
only showing top 20 rows
```

```
+-----+-----+-----+
| marital|age|count|
+-----+-----+-----+
| single| 30|    151|
| single| 28|    138|
| single| 29|    133|
| single| 32|    124|
| single| 26|    121|
| married| 34|    118|
| single| 31|    111|
| single| 27|    110|
| married| 35|    101|
| married| 36|    100|
| single| 25|     99|
| married| 37|     98|
| married| 33|     97|
| single| 33|     97|
| married| 32|     87|
| married| 39|     87|
| married| 38|     86|
| single| 35|     84|
| married| 47|     83|
| married| 31|     80|
+-----+-----+-----+
```

**Note:** Highest subscriptions are from customers of “single” marital status and in age range 28-35.

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

```
spark.sql("select CASE WHEN age > 0 and age < 13 THEN 'child' WHEN age > 12 and age < 20 THEN  
'teen' WHEN age > 19 and age < 40 THEN 'young' WHEN age > 39 and age < 60 THEN 'middle-age'  
ELSE 'senior' END as age_group,count(*) as deposits from bankTable where y='yes' group by  
age_group order by deposits desc").show()
```

age_group	deposits
young	2823
middle-age	1848
senior	600
teen	18

**Conclusion:** Based on age and age\_groups, most the deposits came from young customers which are in range of 20 to 40, followed by middle-age and senior people.