**PROJECT 3**

**Market Analysis in Banking Domain**

**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)**

**Following are the steps:**

**Open spark-shell**

**Importing required spark libraries**

import scala.reflect.runtime.universe

import org.apache.spark.SparkConf

import org.apache.spark.SparkContext

import org.apache.spark.ml.Pipeline

import org.apache.spark.ml.classification.LogisticRegression

import org.apache.spark.ml.feature.Bucketizer

import org.apache.spark.ml.feature.Normalizer

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

import org.apache.spark.ml.feature.VectorAssembler

import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics

import org.apache.spark.sql.DataFrame

import org.apache.spark.sql.SQLContext

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

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

val bank\_people\_data = spark.read.option("multiline","true").json("/user/mailsanjaykumargmail/bank\_edited.json");

* **show the loaded data**

bank\_people\_data.show()

* **convert to dataframe**

val bankdf=bank\_people\_data.toDF()

bankdf.registerTempTable("bank")

spark.conf.set("spark.sql.crossJoin.enabled", true)

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

val success = spark.sql("select (a.subscribed/b.total)\*100 as success\_percent from (select count(\*) as subscribed from bank where y='yes') a,(select count(\*) as total from bank) b").show()

* **Give marketing failure rate**

val failure = spark.sql("select (a.not\_subscribed/b.total)\*100 as failure\_percent from (select count(\*) as not\_subscribed from bank where y='no') a,(select count(\*) as total from bank) b").show()

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

bankDF.select(max($"age")).show()

bankDF.select(min($"age")).show()

bankDF.select(avg($"age")).show()

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

bankDF.select(avg($"balance")).show()

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

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

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

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

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

marital order by number desc ").show()

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

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

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

--Defining a new UDF with which we will generate new features.We divide the age groups into 4 categories.

val ageRDD = spark.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"

})

--Replacing old “age” column with new “age” column

val banknewDF = bankdf.withColumn("age",ageRDD(bankdf("age")))

banknewDF.registerTempTable("bank\_new")

--Running a query to see the age group which subscribed the most. We see it’s ‘Middle-Aged’

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

--Pipeline

val ageInd = new StringIndexer().setInputCol("age").setOutputCol("ageIndex")

--Fitting the model

var strIndModel = ageInd.fit(banknewDF)

--StringIndexerModel.transform() assigns the generated index to each value of the column in the

given DataFrame.

--Middle aged is the most frequent word in this data, so it is given index 0

strIndModel.transform(banknewDF).select("age","ageIndex").show(5)

**So we can conclude from the Feature Engineering that It is the ‘Middle Aged’ people between age 33 and 55 who should be the targeted customers as they subscribe the most**