# Market Analysis Spark Shell Source Code

**Analysis tasks to be done:**

1. Load data and create a Spark data frame
2. Give marketing success rate (No. of people subscribed / total no. of entries)
3. Give marketing failure rate
4. Give the maximum, mean, and minimum age of the average targeted customer
5. Check the quality of customers by checking average balance, median balance of customers
6. Check if age matters in marketing subscription for deposit
7. Check if marital status mattered for a subscription to deposit
8. Check if age and marital status together mattered for a subscription to deposit scheme
9. Do feature engineering for the bank and find the right age effect on the campaign.

## Command to clean the data on cloud lab

```
awk 'BEGIN { FS=";"; OFS="," } { gsub("\"", "") } { $1=$1 } 1' Project\ 1_dataset_bank-full\ \(2\).csv  > project1_outputfile.csv
```

## Commands to load the clean data on HDFS in project directory on HDFS

```
Ls -l
hadoop fs -mkdir project
hadoop fs -put project1_outputfile.csv project/

hadoop fs -ls project //this command should show you the clean file

hadoop fs -cat project/project1_outputfile.csv //this command should show you the content of the clean file
```

## Use Case 1 - Load the dataframe

```
//Open spark shell and run the below command

val bankDF = spark.read.option("inferSchema", "True").option("header", "True").csv("project/project1_outputfile.csv")

bankDF.printSchema

bankDF.show()

import org.apache.spark.sql.DataFrame

case class bank(age:Int, job:String, marital:String, education:String, defaultn:String, balance:Int, housing:String, loan:String, contact:String, day:Int, month: String, duration:Int, campaign:Int, pdays:Int, previous:Int, poutcome:String, y:String)

bankDF.createOrReplaceTempView("bank")
```

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

```
import org.apache.spark.sql.SQLContext

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

val marketing_success_rate=spark.sql("select count(case when y= 'yes' then 1 end)/count(*)*100  as marketing_success_rate from bank").show()
```

### Use Case 3: Give marketing failure rate

```
val marketing_fail_rate=spark.sql("select count(case when y='no' then 1 end)/count(*)*100  as marketing_fail_rate from bank").show()
```

### Use Case 4: Give the maximum, mean, and minimum age of the average targeted customer

```
val maximum_mean_min_age=spark.sql("select max(age) as maximum_age,avg(age) as mean_avg, min(age) as min_age from bank").show()
```

### Use Case 5: Check the quality of customers by checking average balance, median balance of customers

```
val average_median_balance =spark.sql("SELECT AVG(balance) as average,  percentile_approx(balance, 0.5) as median  FROM bank").show()
```

### Use Case 6: 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()
```

### Use Case 7: 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()
```

## Use Case 8: 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()
```

## Use Case 9: Do feature engineering for the bank and find the right age effect on the campaign.

```
val ssb=new org.apache.spark.sql.SparkSession.Builder()

val sparkSession=ssb.getOrCreate()

val sqlCtx=sparkSession.sqlContext;

val ageRDD = sqlCtx.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"})

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

banknewDF.createOrReplaceTempView("bank_new")

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

val ageInd = new org.apache.spark.ml.feature.StringIndexer().setInputCol("age").setOutputCol("ageIndex")

var strIndModel = ageInd.fit(banknewDF)

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