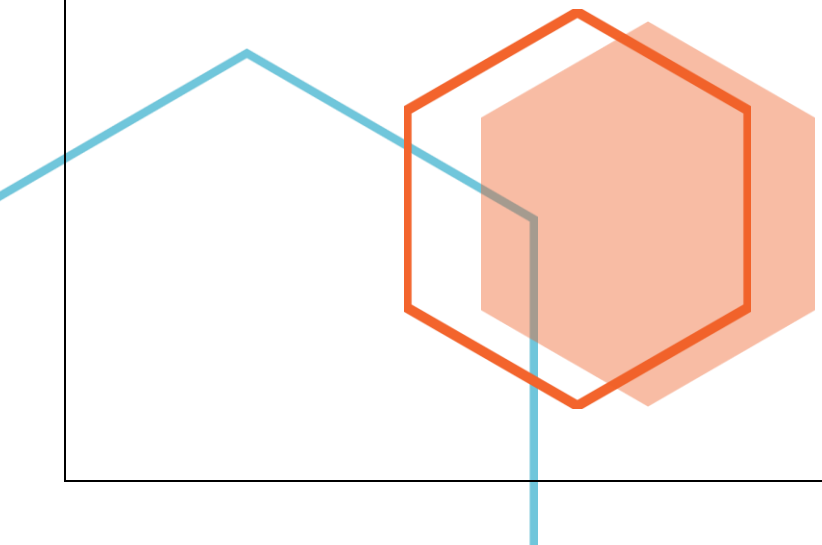
Big Data Hadoop and Spark Developer

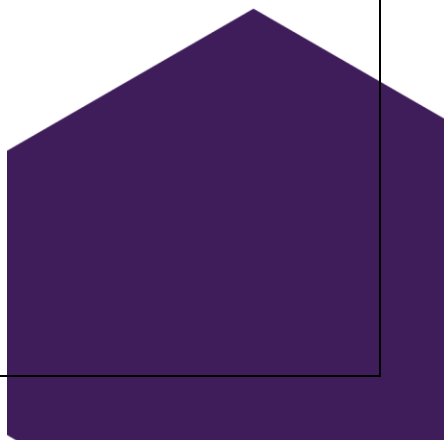
Project 3: Market Analysis in Banking Domain

Name: Ms. Shamali V. Sawant

Instructor: Mr. Deepak Garg

A Portuguese banking institution ran a marketing campaign to convince potential customers to invest in a 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. This problem is addressed by conducting the market 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').

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:

First, we load the data in spark environment and look into its structure:

```
val data = sc.textFile("/user/shamalisawantgmail/Project-3/Project 1_dataset_bank-full (2).csv")

data.collect.take(5).foreach(println)
```

```
scala> val data = sc.textFile("/user/shamalisawantgmail/Project-3/Project 1_dataset_bank-full (2).csv")
data: org.apache.spark.rdd.RDD[String] = /user/shamalisawantgmail/Project-3/Project 1_dataset_bank-full (2).csv MapPartitionsRDD[1] at textFile at <console>:24
```

```
scala> data.collect.take(5).foreach(println)
"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"""
```

The data is not in proper csv format. Therefore, to convert into a dataframe, first we should fit this data into a proper schema by defining a case class that will extract the schema from the existing dataset.

```
val header = data.first

val no_header = data.filter(_!=header)

case class BankData(age : Int, job : String, marital : String, education : String, defaulter : 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)
```

```
scala> val header = data.first
header: String = "age;""job"";""marital"";""education"";""default"";""balance"";""housing"";""loan"";""contact"";""day"";""month"";""duration"";""campaign"";""pdays"";""previous"";""po
utcome"";""y"""

scala> val no_header = data.filter(_!=header)
no_header: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at filter at <console>:27

scala> case class BankData(age : Int, job : String, marital : String, education : String, defaulter : String, balance : Int, housing : String, loan : String, contact : String, day : In
t, month : String, duration : Int, campaign : Int, pdays : Int, previous : Int, poutcome : String, y : String)
defined class BankData
```

Before applying the schema, it is important to get rid of the unnecessary delimiters.

```
val clean_data = no_header.map(x => x.split(";").map(x => x.replaceAll("\"","")))

clean_data.take(10).foreach(x => println(x.mkString(",")))
```

```
scala> val clean_data = no_header.map(x => x.split(";").map(x => x.replaceAll("\"","")))
clean_data: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[3] at map at <console>:25

scala> clean_data.take(10).foreach(x => println(x.mkString(",")))
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
```

Now, the data is ready to put into a proper schema and to convert into a spark dataframe.

```
val bank_data = clean_data.map(attributes =>
BankData(attributes(0).trim.toInt,attributes(1),attributes(2),attributes(3),attributes(4),attribute
s(5).trim.toInt,attributes(6),attributes(7),attributes(8),attributes(9).trim.toInt,attributes(10),attri
butes(11).trim.toInt,attributes(12).trim.toInt,attributes(13).trim.toInt,attributes(14).trim.toInt,att
ributes(15),attributes(16))).toDF()

bank_data.show
```

```
scala> val bank_data = clean_data.map(attributes => BankData(attributes(0).trim.toInt,attributes(1),attributes(2),attributes(3),attributes(4),attributes(5).trim.toInt,attributes(6),att
ributes(7),attributes(8),attributes(9).trim.toInt,attributes(10),attributes(11).trim.toInt,attributes(12).trim.toInt,attributes(13).trim.toInt,attributes(14).trim.toInt,attributes(15),
attributes(16))).toDF()
21/05/06 04:15:47 WARN lineage.LineageWriter: Lineage directory /var/log/spark/lineage doesn't exist or is not writable. Lineage for this application will be disabled.
bank_data: org.apache.spark.sql.DataFrame = [age: int, job: string ... 15 more fields]

scala> bank_data.show
+---+-----------+--------+---------+---------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+---+
|age|        job| marital|education|defaulter|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
```

Analyzing the impact of the campaign, one should look into the success and failure rate of the campaign that help company format the plans accordingly.

```
val success_rate:Double = (bank_data.filter(col("y").equalTo ("yes")).count) *100 /bank_data.count

println("Success Rate is: " + success_rate +"%")

val failure_rate :Double = (bank_data.filter(col("y").equalTo("no")).count) *100 / bank_data.count

print("Failure Rate is :" + failure_rate + "%")
```

```
scala> val success_rate:Double = (bank_data.filter(col("y").equalTo ("yes")).count) *100 /bank_data.count
success_rate: Double = 11.0

scala> println("Success Rate is: " + success_rate +"%")
Success Rate is: 11.0%

scala> val failure_rate :Double = (bank_data.filter(col("y").equalTo("no")).count) *100 / bank_data.count
failure_rate: Double = 88.0

scala> print("Failure Rate is :" + failure_rate + "%")
Failure Rate is :88.0%
```

So many features available in the dataset that might have neglected while contacting the customers and have affected the effect of the campaign resulting into a poor success rate of 11%, but will surely help target the customers that are extremely likely to accept the term deposit.

Moving forward, we will look into the age category of the targeted customers.

```
bank_data.createOrReplaceTempView("bank_data_table")

val max_age = bank_data.select(max($"age"))

print("Maximum age of average targetd customers is :" + max_age.first)

val min_age = bank_data.select(min($"age"))

println("Minimum age of average targeted customers is :" + min_age.first)

val mean_age = bank_data.select(round(mean($"age")))

print("Average age of targetd customers is :" + mean_age.first)
```

```
scala> bank_data.createOrReplaceTempView("bank_data_table")

scala> val max_age = bank_data.select(max($"age"))
max_age: org.apache.spark.sql.DataFrame = [max(age): int]

scala> print("Maximum age of average targetd customers is :" + max_age.first)
Maximum age of average targetd customers is :[95]
scala> val min_age = bank_data.select(min($"age"))
min_age: org.apache.spark.sql.DataFrame = [min(age): int]

scala> println("Minimum age of average targeted customers is :" + min_age.first)
Minimum age of average targeted customers is :[18]

scala> val mean_age = bank_data.select(round(mean($"age")))
mean_age: org.apache.spark.sql.DataFrame = [round(avg(age), 0): double]

scala> print("Average age of targetd customers is :" + mean_age.first)
Average age of targetd customers is :[41.0]
```

Thus, the customers within age 18-95 were targeted for the term deposit campaign.

We can also get to know the quality of the targeted customers by looking into their mean and median balance.

```
val avg_balance = bank_data.select(round(mean($"balance")))

println("Average Balance of customers is :" + avg_balance.first)

val medianArray = bank_data.stat.approxQuantile("balance", Array(0.5),0)

print("Median balance of customers : " + medianArray(0))
```

```
scala> val avg_balance = bank_data.select(round(mean($"balance")))
avg_balance: org.apache.spark.sql.DataFrame = [round(avg(balance), 0): double]

scala> println("Average Balance of customers is :" + avg_balance.first)
Average Balance of customers is :[1362.0]

scala> val medianArray = bank_data.stat.approxQuantile("balance", Array(0.5),0)
medianArray: Array[Double] = Array(448.0)

scala> print("Median balance of customers : " + medianArray(0))
Median balance of customers : 448.0
```

As we have already seen that, the success rate of the campaign was very poor and that might be because of the wrongly targeted customers. To select correct targets, we should dive deep into the features of the customers that accepted the term deposit.

```
val age_depend = bank_data.filter(col("y").equalTo("yes")).groupBy("age").count.orderBy(col("count").desc)

age_depend.show(10)

val marital_status = bank_data.filter(col("y").equalTo("yes")).groupBy("marital").count.orderBy(col("count").desc)

marital_status.show(10)

val age_and_marital1 = spark.sql("select age,marital,count(*) as number from bank_data_table where y='yes'  group by age,marital order by number desc")
```

```
scala> val age_depend = bank_data.filter(col("y").equalTo("yes")).groupBy("age").count.orderBy(col("count").desc)
age_depend: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [age: int, count: bigint]

scala> age_depend.show(10)
+---+-----+
|age|count|
+---+-----+
| 32|  221|
| 30|  217|
| 33|  210|
| 35|  209|
| 31|  206|
| 34|  198|
| 36|  195|
| 29|  171|
| 37|  170|
| 28|  162|
+---+-----+
only showing top 10 rows


scala> val marital_status = bank_data.filter(col("y").equalTo("yes")).groupBy("marital").count.orderBy(col("count").desc)
marital_status: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [marital: string, count: bigint]

scala> marital_status.show(10)
+--------+-----+
| marital|count|
+--------+-----+
| married| 2755|
|  single| 1912|
|divorced|  622|
+--------+-----+
```

```
scala> val age_and_marital1 = spark.sql("select age,marital,count(*) as number from bank_data_table where y='yes' group by age,marital order by number desc")
age_and_marital1: org.apache.spark.sql.DataFrame = [age: int, marital: string ... 1 more field]

scala> age_and_marital1.show(10)
+---+-------+------+
|age|marital|number|
+---+-------+------+
| 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|
+---+-------+------+
only showing top 10 rows
```

We can see that, out of all the targeted customers who accepted the term deposit were either single or married in the age group of 26 to 40. Divorced customers were less likely to go for a term deposit. Thus, the bank should target the customers within this age group.

To generalize the age criterion, we will divide the age into 4 categories → “Teen”, “Young”, “Middle Aged”, “Old”

This will make the picture more clear while targeting the customers.

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

if (age< 20) "Teen"

else if (age >= 20 && age <35) "Young"

else if (age >=35 && age <55) "Middle Aged"

else "Old"

})

val bank_data_newDF =
bank_data.withColumn("Age_Group",age_group(bank_data("age")))

bank_data_newDF.createOrReplaceTempView("bank_data_new_table")

val age_group_data = spark.sql("select Age_Group,count(*) as number from
bank_data_new_table where

y='yes' group by Age_Group order by number desc")

age_group_data.show
```

```
scala> val age_group = spark.udf.register("age_group" ,(age:Int) =>{
     | if (age< 20) "Teen"
     | else if (age >= 20 && age <35) "Young"
     | else if (age >=35 && age <55) "Middle Aged"
     | else "Old"
     | })
age_group: org.apache.spark.sql.expressions.UserDefinedFunction = UserDefinedFunction(<function1>,StringType,Some(List(IntegerType)))

scala> val bank_data_newDF = bank_data.withColumn("Age_Group",age_group(bank_data("age")))
bank_data_newDF: org.apache.spark.sql.DataFrame = [age: int, job: string ... 16 more fields]

scala> bank_data_newDF.createOrReplaceTempView("bank_data_new_table")

scala> val age_group_data = spark.sql("select Age_Group,count(*) as number from bank_data_new_table where y='yes' group by Age_Group order by number desc")
age_group_data: org.apache.spark.sql.DataFrame = [Age_Group: string, number: bigint]

scala> age_group_data.show
+-----------+------+
|  Age_Group|number|
+-----------+------+
|Middle Aged|  2327|
|      Young|  1962|
|        Old|   982|
|       Teen|    18|
+-----------+------+
```

With this, we can say that the Middle aged and young people are extremely likely to get a term deposit. Old people may or may not go for a term deposit. So, based on the priorities and the risk that the bank can afford, it should decide whether to target old people or not.

Conclusion:

In conclusion, we have analyzed the campaign data of a banking institution and studied the features like age, marital status and balance of the customer that will help the institution target a customer that will most likely accept the term deposit.

*****