Market Analysis Screenshots

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.

- **Use Case 1 - Load data and create a Spark data frame**

  //Load data

```
scala> val bankDF = spark.read.option("inferSchema", "True").option("header", "True").csv("project/project1_outputfile.csv")
21/08/31 01:09:59 WARN lineage.LineageWriter: Lineage directory /var/log/spark/lineage doesn't exist or is not writable. Lineage for this application will be disabled.
bankDF: org.apache.spark.sql.DataFrame = [age: int, job: string ... 15 more fields]

scala> bankDF.printSchema
root
 |-- age: integer (nullable = true)
 |-- job: string (nullable = true)
 |-- marital: string (nullable = true)
 |-- education: string (nullable = true)
 |-- default: string (nullable = true)
 |-- balance: integer (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)


scala> bankDF.show()
+---+-----------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+---+
|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|
| 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
```

// create a Spark data frame

```
scala> import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.DataFrame

scala> 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
Int, previous:Int, poutcome:String, y:String)
defined class bank

scala> bankDF.createOrReplaceTempView("bank")
```

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

```
scala> val marketing_success_rate=spark.sql("select count(case when y= 'yes' then 1 end)/count(*)*100  as marketing_success_rate from bank").show()
+----------------------+
|marketing_success_rate|
+----------------------+
|     11.698480458295547|
+----------------------+
```

- **Use Case 3:** Give marketing failure rate

```
marketing_success_rate: Unit = ()

scala> val marketing_fail_rate=spark.sql("select count(case when y='no' then 1 end)/count(*)*100  as marketing_fail_rate from bank").show()
+--------------------+
|marketing_fail_rate|
+--------------------+
|   88.30151954170445|
+--------------------+
```

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

```
scala> val max_mean_min_age=spark.sql("select max(age) as maxmium_age, avg(age) as mean_age, min(age) as minimum_age from bank").show()
+-----------+-----------------+-----------+
|maxmium_age|         mean_age|minimum_age|
+-----------+-----------------+-----------+
|         95|40.93621021432837|         18|
+-----------+-----------------+-----------+

max_mean_min_age: Unit = ()
```

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

```
scala> //Check the quality of customers by checking average balance, median balance of customers

scala> val average_median_balance=spark.sql("select  avg(balance) as average_balance, percentile_approx(balance,0.5) as median_balance from bank").show()
+---------------+--------------+
|average_balance|median_balance|
+---------------+--------------+
|       102812.5|         40000|
+---------------+--------------+

average_median_balance: Unit = ()
```

- **Use Case 6: Check if age matters in marketing subscription for deposit**

```
scala> //Check if age matters in marketing subscription for deposit

scala> val age = spark.sql("select age, count(*) as number from bank where y='yes' group by age order by number desc ").show()
+---+------+
|age|number|
+---+------+
| 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: Unit = ()
```

- **Use Case 7: Check if marital status mattered for a subscription to deposit**

```
scala> //Check if marital status mattered for a subscription to deposit

scala> val marital = spark.sql("select marital,count(*) as number from bank where y='yes' group by marital order by number desc ").show()
+--------+------+
| marital|number|
+--------+------+
| married|  2755|
|  single|  1912|
|divorced|   622|
+--------+------+

marital: Unit = ()
```

- **Use Case 8: Check if age and marital status together mattered for a subscription to deposit scheme**

```
scala> //Check if age and marital status together mattered for a subscription to deposit scheme

scala> 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()
+---+-------+------+
|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|
| 25| single|    99|
| 37|married|    98|
| 33|married|    97|
| 33| single|    97|
| 39|married|    87|
| 32|married|    87|
| 38|married|    86|
| 35| single|    84|
| 47|married|    83|
| 31|married|    80|
+---+-------+------+
only showing top 20 rows

age_marital: Unit = ()
```

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

```
scala> banknewDF.createOrReplaceTempView("bank_new")

scala> import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.SQLContext

scala> import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.SQLContext

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

scala> val ssb=new org.apache.spark.sql.SparkSession.Builder()
ssb: org.apache.spark.sql.SparkSession.Builder = org.apache.spark.sql.SparkSession$Builder@2e897323

scala> val sparkSession=ssb.getOrCreate()
sparkSession: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@385abc08

scala> val sqlCtx=sparkSession.sqlContext;
sqlCtx: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@63cb06df

scala> 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"})
21/08/30 07:19:03 WARN analysis.SimpleFunctionRegistry: The function agerdd replaced a previously registered function.
ageRDD: org.apache.spark.sql.expressions.UserDefinedFunction = UserDefinedFunction(<function1>,StringType,Some(List(IntegerType)))

scala> val banknewDF = bankDF.withColumn("age",ageRDD(bankDF("age")))
banknewDF: org.apache.spark.sql.DataFrame = [age: string, job: string ... 15 more fields]

scala> banknewDF.createOrReplaceTempView("bank_new")

scala> val age_target = spark.sql("select age, count(*) as number from bank_new where y='yes' group by age order by number desc ").show( )
+-----------+------+
|        age|number|
+-----------+------+
|Middle Aged|  2601|
|      Young|  1539|
|        Old|  1131|
|       Teen|    18|
+-----------+------+

age_target: Unit = ()
```

As can be seen from the above screenshot, we can conclude from the feature engineering that it is the “Middle Aged” people between 33 and 55 who should be the targeted customers as they subscribe the most.

```
scala> val ageInd = new org.apache.spark.ml.feature.StringIndexer().setInputCol("age").setOutputCol("ageIndex")
ageInd: org.apache.spark.ml.feature.StringIndexer = strIdx_5ef0842e102c

scala>  var strIndModel = ageInd.fit(banknewDF)
strIndModel: org.apache.spark.ml.feature.StringIndexerModel = strIdx_5ef0842e102c

scala> 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
```