Loaded data and spark data frame created, completed the required task to for market analysis in bank also, completed the feature engineering for the finding the campaign results.

Project 3 - Market Analysis in Banking Domain

Rohit B Jondhale

Market Analysis in Banking Domain

The data size is huge, and the marketing team has asked you to perform the below analysis

1. Load data and create a Spark data frame

   Cd Project_1/Projectdata.csv

   mv ~/Project_1/Projectdata.csv ~/Project_1/Projectdata.txt

   cd ~/Project_1/

   head Projectdata.txt

   hdfs dfs -put Projectdata.txt project/

   var rdd = sc.textFile(“project/Projectdata.txt”)

   rdd.take(2)

   val nrdd = rdd.map(“x=> x.replace(“””””,””))

   nrdd.take(2)

   nrdd.coalesce(1).saveAsTextFile(“project/ndata”)

   var df =
spark.read.format("csv").option("header","true").option("delimiter",";").option("inferschema","true")
.load("project/ndata")

```
ip-10-0-31-42 login: rohitjondhalemsgmail
Password:
[rohitjondhalemsgmail@ip-10-0-31-42 ~]$ hdfs dfs -ls
Found 3 items
drwx------   - rohitjondhalemsgmail hadoop          0 2022-04-03 20:00 .Trash
drwxr-xr-x   - rohitjondhalemsgmail hadoop          0 2022-05-17 15:00 .sparkStaging
drwxr-xr-x   - rohitjondhalemsgmail hadoop          0 2022-06-11 10:47 project
[rohitjondhalemsgmail@ip-10-0-31-42 ~]$ cd Project_1
[rohitjondhalemsgmail@ip-10-0-31-42 Project_1]$ cd Project_1/Projectdata.csv
-bash: cd: Project_1/Projectdata.csv: No such file or directory
[rohitjondhalemsgmail@ip-10-0-31-42 Project_1]$ mv Project_1/Projectdata.csv ~/Projec
mv: cannot stat ‘Project_1/Projectdata.csv’: No such file or directory
[rohitjondhalemsgmail@ip-10-0-31-42 Project_1]$ mv ~/Project_1/Projectdata.csv ~/Proj
[rohitjondhalemsgmail@ip-10-0-31-42 Project_1]$ cd ~/Project_1/
[rohitjondhalemsgmail@ip-10-0-31-42 Project_1]$ head Projectdata.txt
"age;""job"";""marital"";""education"";""default"";""balance"";""housing"";""loan"";"
"58;""management"";""married"";""tertiary"";""no"";2143;""yes"";""no"";""unknown"";5;
"44;""technician"";""single"";""secondary"";""no"";29;""yes"";""no"";""unknown"";5;""
"33;""entrepreneur"";""married"";""secondary"";""no"";2;""yes"";""yes"";""unknown"";5
"47;""blue-collar"";""married"";""unknown"";""no"";1506;""yes"";""no"";""unknown"";5;
"33;""unknown"";""single"";""unknown"";""no"";1;""no"";""no"";""unknown"";5;""may"";1
"35;""management"";""married"";""tertiary"";""no"";231;""yes"";""no"";""unknown"";5;"
"28;""management"";""single"";""tertiary"";""no"";447;""yes"";""yes"";""unknown"";5;"
"42;""entrepreneur"";""divorced"";""tertiary"";""yes"";2;""yes"";""no"";""unknown"";5
"58;""retired"";""married"";""primary"";""no"";121;""yes"";""no"";""unknown"";5;""may
[rohitjondhalemsgmail@ip-10-0-31-42 Project_1]$ hdfs dfs -ls
Found 3 items
drwx------   - rohitjondhalemsgmail hadoop          0 2022-04-03 20:00 .Trash
drwxr-xr-x   - rohitjondhalemsgmail hadoop          0 2022-05-17 15:00 .sparkStaging
drwxr-xr-x   - rohitjondhalemsgmail hadoop          0 2022-06-11 10:47 project
[rohitjondhalemsgmail@ip-10-0-31-42 Project_1]$ hdfs dfs -put Projectdata.txt project
```

```
[rohitjondhalemsgmail@ip-10-0-31-42 Project_1]$ spark-shell
Setting default log level to "ERROR".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(new
22/06/11 11:05:18 WARN cluster.YarnSchedulerBackend$YarnSchedulerEndpoint: Attempted
22/06/11 11:05:18 WARN lineage.LineageWriter: Lineage directory /var/log/spark/lineag
Spark context available as 'sc' (master = yarn, app id = application_1640258093152_78
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.0-cdh6.3.2
      /_/

Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_144)
Type in expressions to have them evaluated.
Type :help for more information.

scala> var rdd = sc.textFile("project/Projectdata.txt")
rdd: org.apache.spark.rdd.RDD[String] = project/Projectdata.txt MapPartitionsRDD[1] a

scala> rdd.take (2)
res0: Array[String] = Array("age;""job"";""marital"";""education"";""default"";""bala
ious"";""poutcome"";""y""", "58;""management"";""married"";""tertiary"";""no"";2143;"


scala> val nrdd = rdd.map(x=> x.replace ("""""",""))
nrdd: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[3] at map at <console>:25

scala> nrdd.take(2)
res2: Array[String] = Array(age;job;marital;education;default;balance;housing;loan;co
;yes;no;unknown;5;may;261;1;-1;0;unknown;no)
```

```
scala> nrdd.coalesce(1).saveAsTextFile("project/ndata")

scala> var df = spark.read.format("csv").option("header", "true").option("delimiter",
22/06/11 12:15:02 WARN lineage.LineageWriter: Lineage directory /var/log/spark/lineag
df: org.apache.spark.sql.DataFrame = [age: int, job: string ... 15 more fields]

scala> df.show
+---+-----------+--------+---------+-------+-------+-------+----+-------+---+-----+-
|age|        job| marital|education|default|balance|housing|loan|contact|day|month|d
+---+-----------+--------+---------+-------+-------+-------+----+-------+---+-----+-
| 58|  management| married| tertiary|     no|   2143|    yes|  no|unknown|  5|  may|
| 44|  technician|  single|secondary|     no|     29|    yes|  no|unknown|  5|  may|
| 33|entrepreneur| married|secondary|     no|      2|    yes| yes|unknown|  5|  may|
| 47| blue-collar| married|  unknown|     no|   1506|    yes|  no|unknown|  5|  may|
| 33|     unknown|  single|  unknown|     no|      1|     no|  no|unknown|  5|  may|
| 35|  management| married| tertiary|     no|    231|    yes|  no|unknown|  5|  may|
| 28|  management|  single| tertiary|     no|    447|    yes| yes|unknown|  5|  may|
| 42|entrepreneur|divorced| tertiary|    yes|      2|    yes|  no|unknown|  5|  may|
| 58|     retired| married|  primary|     no|    121|    yes|  no|unknown|  5|  may|
| 43|  technician|  single|secondary|     no|    593|    yes|  no|unknown|  5|  may|
| 41|      admin.|divorced|secondary|     no|    270|    yes|  no|unknown|  5|  may|
| 29|      admin.|  single|secondary|     no|    390|    yes|  no|unknown|  5|  may|
| 53|  technician| married|secondary|     no|      6|    yes|  no|unknown|  5|  may|
| 58|  technician| married|  unknown|     no|     71|    yes|  no|unknown|  5|  may|
| 57|    services| married|secondary|     no|    162|    yes|  no|unknown|  5|  may|
| 51|     retired| married|  primary|     no|    229|    yes|  no|unknown|  5|  may|
| 45|      admin.|  single|  unknown|     no|     13|    yes|  no|unknown|  5|  may|
| 57| blue-collar| married|  primary|     no|     52|    yes|  no|unknown|  5|  may|
| 60|     retired| married|  primary|     no|     60|    yes|  no|unknown|  5|  may|
| 33|    services| married|secondary|     no|      0|    yes|  no|unknown|  5|  may|
+---+-----------+--------+---------+-------+-------+-------+----+-------+---+-----+-
only showing top 20 rows
```

```
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
```

```
scala> val sqlContext = new org.apache.spark.sql.SQLContext(sc)
warning: there was one deprecation warning; re-run with -deprecation for detail
sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@3
```

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

```
val success_rate = df.filter($"y" === "yes").count.toFloat / df.count.toFloat *100

val failure_rate = df.filter($"y" === "no").count.toFloat / df.count.toFloat *100
```

```
scala> val success_rate = df.filter($"y" === "yes").count.toFloat / df.count.to
success_rate: Float = 11.698481

scala>

scala> val failure_rate = df.filter($"y" === "no").count.toFloat / df.count.toF
failure_rate: Float = 88.30152
```

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

   df.agg(max($"age"),min($"age"), avg($"age")).show()

```
scala> df.agg(max($"age"),min($"age"), avg($"age")).show()
+--------+--------+-----------------+
|max(age)|min(age)|         avg(age)|
+--------+--------+-----------------+
|      95|      18|40.93621021432837|
+--------+--------+-----------------+
```

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

   import org.apache.commons.math3.stat.descriptive

   df.createOrReplaceTempView("bankdata")

   val medBal = sql("SELECT max(balance) as max, min(balance) as min, avg(balance) as average, percentile_approx(balance, 0.5) as median FROM bankdata");

   medBal.show()

```
scala> val medBal = sql("SELECT max(balance) as max, min(balance) as min, avg(b
medBal: org.apache.spark.sql.DataFrame = [max: int, min: int ... 2 more fields]

scala> medBal.show()
+------+-----+------------------+------+
|   max|  min|           average|median|
+------+-----+------------------+------+
|102127|-8019|1362.2720576850766|   448|
+------+-----+------------------+------+
```

3. Check if age matters in marketing subscription for deposit

val age = sqlContext.sql("SELECT age, count(*) as number from bankdata where y='yes' group by age order by number desc ").show()

```
scala> val age = sqlContext.sql("SELECT age, count(*) as number from bankdata w
+---+------+
|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 = ()
```

Shows that, age matters. The age range between (30-36) shows most promising while age 32 people are most subscribed.

4. Check if marital status mattered for a subscription to deposit

df.groupBy($"y".alias("Did the customer Subscribed?")).agg(count($"marital").alias("Marital Count")).show

```
scala> df.groupBy($"y".alias("Did the customer Subscribed?")).agg(count($"marit
+--------------------------+-------------+
|Did the customer Subscribed?|Marital Count|
+--------------------------+-------------+
|                        no|        39922|
|                       yes|         5289|
+--------------------------+-------------+
```

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

df.groupBy("marital","y").count.sort($"count").show

```
scala> df.groupBy("marital","y").count.sort($"count").show
+--------+---+-----+
| marital|  y|count|
+--------+---+-----+
|divorced|yes|  622|
|  single|yes| 1912|
| married|yes| 2755|
|divorced| no| 4585|
|  single| no|10878|
| married| no|24459|
+--------+---+-----+
```

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

import org.apache.spark.sql.functions.udf

```
scala> def ageToCategory = udf((age:Int) => {
     |       age match {
     |       case a if a < 30 => "young"
     |       case a if a > 65 => "Old"
     |       case _ => "mid"
     |       }
     |       }
     |     )
```

val new_df = df.withColumn("agecategory",ageToCategory(df("age")))

new_df.groupBy("agecategory","y").count().sort($"count".desc).show

```
scala> import org.apache.spark.sql.functions.udf
import org.apache.spark.sql.functions.udf

scala> def ageToCategory = udf((age:Int) => {
     |       age match {
     |       case a if a < 30 => "young"
     |       case a if a > 65 => "Old"
     |       case _ => "mid"
     |       }
     |       }
     |      )
ageToCategory: org.apache.spark.sql.expressions.UserDefinedFunction

scala> val new_df = df.withColumn("agecategory",ageToCategory(df("age")))
new_df: org.apache.spark.sql.DataFrame = [age: int, job: string ... 16 more fie

scala> new_df.groupBy("agecategory","y").count().sort($"count".desc).show
+-----------+---+-----+
|agecategory|  y|count|
+-----------+---+-----+
|        mid| no|35146|
|      young| no| 4345|
|        mid|yes| 4041|
|      young|yes|  928|
|        Old| no|  431|
|        Old|yes|  320|
+-----------+---+-----+
```

Correlation Analysis

import org.apache.spark.sql.functions._

val new_DF = df.withColumn("sub", when($"y" === "yes" , 1).otherwise(0))

new_DF.show(5)

```
scala> import org.apache.spark.sql.functions._
import org.apache.spark.sql.functions._
```

```
scala> val new_DF = df.withColumn("sub", when($"y" === "yes" , 1).otherwise(0))
new_DF: org.apache.spark.sql.DataFrame = [age: int, job: string ... 16 more fie

scala>

scala> new_DF.show(5)
+---+-----------+-------+---------+-------+-------+-------+----+-------+---+--
|age|        job|marital|education|default|balance|housing|loan|contact|day|mo
+---+-----------+-------+---------+-------+-------+-------+----+-------+---+--
| 58|  management|married| tertiary|     no|   2143|    yes|  no|unknown|  5|
| 44|  technician| single|secondary|     no|     29|    yes|  no|unknown|  5|
| 33|entrepreneur|married|secondary|     no|      2|    yes| yes|unknown|  5|
| 47| blue-collar|married|  unknown|     no|   1506|    yes|  no|unknown|  5|
| 33|     unknown| single|  unknown|     no|      1|     no|  no|unknown|  5|
+---+-----------+-------+---------+-------+-------+-------+----+-------+---+--
only showing top 5 rows
```

```
import org.apache.spark.ml.linalg.{Matrix, Vectors}
import org.apache.spark.ml.stat.Correlation
import org.apache.spark.sql.Row
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.linalg.Vectors
```

```
scala> import org.apache.spark.ml.linalg.{Matrix, Vectors}
import org.apache.spark.ml.linalg.{Matrix, Vectors}

scala> import org.apache.spark.ml.stat.Correlation
import org.apache.spark.ml.stat.Correlation

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

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

scala> import org.apache.spark.ml.linalg.Vectors
import org.apache.spark.ml.linalg.Vectors
```

val corr_DF = new_DF.select($"age",$"sub")

```
scala> val corr_DF = new_DF.select($"age",$"sub")
corr_DF: org.apache.spark.sql.DataFrame = [age: int, sub: int]

scala> corr_DF.show
+---+---+
|age|sub|
+---+---+
| 58|  0|
| 44|  0|
| 33|  0|
| 47|  0|
| 33|  0|
| 35|  0|
| 28|  0|
| 42|  0|
| 58|  0|
| 43|  0|
| 41|  0|
| 29|  0|
| 53|  0|
| 58|  0|
| 57|  0|
| 51|  0|
| 45|  0|
| 57|  0|
| 60|  0|
| 33|  0|
+---+---+
only showing top 20 rows
```

corr_DF.show

Pearson Correlation

println("Pearson Correlation: " + corr_DF.stat.corr("age","sub"))

```
scala> println("Pearson Correlation: " + corr_DF.stat.corr("age","sub"))
Pearson Correlation: 0.025155017088387376
```

Pearson Correlation is 0.02, depicts week correlation or no correlation.

We can conclude that, from the Feature Engineering, It is the ‘Middle Aged’ people between aged people should be the targeted customers as they subscribe the most.