PROJECT

Copying Dataset to HDFS

```
[pnelanutgmail@ip-10-0-42-218 ~]$ hdfs dfs -copyFromLocal /mnt/home/pnelanutgmail/Project/bank_data.csv /user/pnelanutgm
```

1. **1. Loading Data into Data frame**

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

scala> val sqlContext = new SQLContext(sc)
warning: there was one deprecation warning; re-run with -deprecation for details
sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@33acec5e

scala> val df = sqlContext.read.format("com.databricks.spark.csv").option("header", "true").option
ta.csv")
21/05/06 16:32:18 WARN lineage.LineageWriter: Lineage directory /var/log/spark/lineage doesn't exi
led.
df: org.apache.spark.sql.DataFrame = [age: int, job: string ... 15 more fields]
```

Print Schema

```
scala> df.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)
```

Checking Data in Data frame

```
scala> df.show()
+---+-----------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+-----
|age|        job| marital|education|default|balance|housing|loan|contact|day|month|duration|campa
+---+-----------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+-----
| 58|  management| married| tertiary|     no|   2143|    yes|  no|unknown|  5|  may|     261|
| 44|  technician|  single|secondary|     no|     29|    yes|  no|unknown|  5|  may|     151|
| 33|entrepreneur| married|secondary|     no|      2|    yes| yes|unknown|  5|  may|      76|
| 47| blue-collar| married|  unknown|     no|   1506|    yes|  no|unknown|  5|  may|      92|
| 33|     unknown|  single|  unknown|     no|      1|     no|  no|unknown|  5|  may|     198|
| 35|  management| married| tertiary|     no|    231|    yes|  no|unknown|  5|  may|     139|
| 28|  management|  single| tertiary|     no|    447|    yes| yes|unknown|  5|  may|     217|
| 42|entrepreneur|divorced| tertiary|    yes|      2|    yes|  no|unknown|  5|  may|     380|
| 58|     retired| married|  primary|     no|    121|    yes|  no|unknown|  5|  may|      50|
| 43|  technician|  single|secondary|     no|    593|    yes|  no|unknown|  5|  may|      55|
| 41|      admin.|divorced|secondary|     no|    270|    yes|  no|unknown|  5|  may|     222|
| 29|      admin.|  single|secondary|     no|    390|    yes|  no|unknown|  5|  may|     137|
| 53|  technician| married|secondary|     no|      6|    yes|  no|unknown|  5|  may|     517|
| 58|  technician| married|  unknown|     no|     71|    yes|  no|unknown|  5|  may|      71|
| 57|    services| married|secondary|     no|    162|    yes|  no|unknown|  5|  may|     174|
| 51|     retired| married|  primary|     no|    229|    yes|  no|unknown|  5|  may|     353|
| 45|      admin.|  single|  unknown|     no|     13|    yes|  no|unknown|  5|  may|      98|
| 57| blue-collar| married|  primary|     no|     52|    yes|  no|unknown|  5|  may|      38|
| 60|     retired| married|  primary|     no|     60|    yes|  no|unknown|  5|  may|     219|
| 33|    services| married|secondary|     no|      0|    yes|  no|unknown|  5|  may|      54|
+---+-----------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+-----
only showing top 20 rows
```

2. 2. Marketing Success rate (in percentage)

```
scala> val suc = df.filter($"y" === "yes").count.toFloat / df.count.toFloat *100
suc: Float = 11.698481
```

Marketing Fail rate (in percentage)

```
scala> val fail_rate = 100 - suc
fail_rate: Float = 88.30152

scala> val fail = df.filter($"y" === "no").count.toFloat / df.count.toFloat *100
fail: Float = 88.30152
```

Since the calculation by count of no and 100 – success is same we can conclude there is no null value under ‘y’

3. 3. Maximum, Mean, and Minimum age of the average targeted customer

```
scala> import org.apache.spark.sql.functions.{min, max, avg}
import org.apache.spark.sql.functions.{min, max, avg}

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

4. 4. Quality of customers by checking Average balance, Median balance of customers

a. creating table for Spark SQL

```
scala> df.select("age","job","marital","education","default","balance","housing","loan","contac
,"y").show
+---+-----------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--
|age|        job| marital|education|default|balance|housing|loan|contact|day|month|duration|ca
+---+-----------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--
| 58|  management| married| tertiary|     no|   2143|    yes|  no|unknown|  5|  may|     261|
| 44|  technician|  single|secondary|     no|     29|    yes|  no|unknown|  5|  may|     151|
| 33|entrepreneur| married|secondary|     no|      2|    yes| yes|unknown|  5|  may|      76|
| 47| blue-collar| married|  unknown|     no|   1506|    yes|  no|unknown|  5|  may|      92|
| 33|     unknown|  single|  unknown|     no|      1|     no|  no|unknown|  5|  may|     198|
| 35|  management| married| tertiary|     no|    231|    yes|  no|unknown|  5|  may|     139|
| 28|  management|  single| tertiary|     no|    447|    yes| yes|unknown|  5|  may|     217|
| 42|entrepreneur|divorced| tertiary|    yes|      2|    yes|  no|unknown|  5|  may|     380|
| 58|     retired| married|  primary|     no|    121|    yes|  no|unknown|  5|  may|      50|
| 43|  technician|  single|secondary|     no|    593|    yes|  no|unknown|  5|  may|      55|
| 41|      admin.|divorced|secondary|     no|    270|    yes|  no|unknown|  5|  may|     222|
| 29|      admin.|  single|secondary|     no|    390|    yes|  no|unknown|  5|  may|     137|
| 53|  technician| married|secondary|     no|      6|    yes|  no|unknown|  5|  may|     517|
| 58|  technician| married|  unknown|     no|     71|    yes|  no|unknown|  5|  may|      71|
| 57|    services| married|secondary|     no|    162|    yes|  no|unknown|  5|  may|     174|
| 51|     retired| married|  primary|     no|    229|    yes|  no|unknown|  5|  may|     353|
| 45|      admin.|  single|  unknown|     no|     13|    yes|  no|unknown|  5|  may|      98|
| 57| blue-collar| married|  primary|     no|     52|    yes|  no|unknown|  5|  may|      38|
| 60|     retired| married|  primary|     no|     60|    yes|  no|unknown|  5|  may|     219|
| 33|    services| married|secondary|     no|      0|    yes|  no|unknown|  5|  may|      54|
+---+-----------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--
only showing top 20 rows
```

b.

```
scala> df.registerTempTable("bank_Analysis")
warning: there was one deprecation warning; re-run with -deprecation for details
```

Use sqlContext to calculate data

```
scala> sqlContext.sql("""select * from bank_Analysis limit 10""").show();
+---+-----------+--------+---------+-------+-------+-------+----+-------+---+-----+---
|age|        job| marital|education|default|balance|housing|loan|contact|day|month|dur
+---+-----------+--------+---------+-------+-------+-------+----+-------+---+-----+---
| 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|
+---+-----------+--------+---------+-------+-------+-------+----+-------+---+-----+---
```

```
scala> import org.apache.commons.math3.stat.descriptive
import org.apache.commons.math3.stat.descriptive

scala> val medBal = sql("SELECT max(balance) as max, min(balance) as min, avg(balance)

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|
+------+-----+------------------+------+
```

5. **5. Age matters in marketing subscription for deposit?**

```
scala> val age_mktg=sqlContext.sql("select  y, avg(age), percentile_approx(age, 0.5) as
age_mktg: org.apache.spark.sql.DataFrame = [y: string, avg(age): double ... 1 more fiel

scala> age_mktg.show()
+---+------------------+------+
|  y|          avg(age)|median|
+---+------------------+------+
| no| 40.83898602274435|    39|
|yes|41.670069956513515|    38|
+---+------------------+------+
```

As per the above data we can conclude age doesn't matter for subscription.

6. **6. Marital status mattered for a subscription to deposit?**

```
scala> val marital_mktg=sqlContext.sql("select marital, y, count(marital)  from  bank_A
marital_mktg: org.apache.spark.sql.DataFrame = [marital: string, y: string ... 1 more f

scala> marital_mktg.show()
+--------+---+--------------+
| marital|  y|count(marital)|
+--------+---+--------------+
|divorced| no|          4585|
|  single| no|         10878|
| married| no|         24459|
|divorced|yes|           622|
| married|yes|          2755|
|  single|yes|          1912|
+--------+---+--------------+
```

The data is ambiguous, conclusion about marital status a matter for subscription cannot be driven purely by this information.

7. **7. Age and marital status together mattered for a subscription to deposit scheme?**

```
scala> val age_mktg_subs=sqlContext.sql("select marital, y, count(marital), min(age), m
s  group by marital, y order by y")
age_mktg_subs: org.apache.spark.sql.DataFrame = [marital: string, y: string ... 5 more

scala> age_mktg_subs.show()
+--------+---+--------------+----------+----------+------------------+------+
| marital|  y|count(marital)|min(age)|max(age)|          avg(age)|median|
+--------+---+--------------+----------+----------+------------------+------+
|divorced| no|          4585|      24|      94| 45.31297709923664|    45|
|  single| no|         10878|      18|      83| 33.96258503401361|    32|
| married| no|         24459|      20|      95| 43.05854695613067|    42|
|divorced|yes|           622|      27|      95|49.247588424437296|    47|
| married|yes|          2755|      20|      93| 46.51143375680581|    45|
|  single|yes|          1912|      18|      86| 32.22907949790795|    31|
+--------+---+--------------+----------+----------+------------------+------+
```

Though parameters are closely indicating for subscription and no subscription, a minor observation can be made such as Married and divorced people who are elderly and single people who are younger are more inclined for subscription.

8. **8. Feature engineering for the bank and find the right age effect on the campaign**

```
scala> val bank_f = df.filter($"y" === "yes")
bank_f: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [age: int, job: string

scala> val age_cat = bank_f.withColumn("age_cat", when($"age" < 25, "young").otherwise(
age_cat: org.apache.spark.sql.DataFrame = [age: int, job: string ... 16 more fields]

scala> val result = age_cat.groupBy("age_cat").count()
result: org.apache.spark.sql.DataFrame = [age_cat: string, count: bigint]

scala> result.show()
+-------+-----+
|age_cat|count|
+-------+-----+
|    mid| 4580|
|    old|  502|
|  young|  207|
+-------+-----+
```

```
scala> val resultWithMarital= age_cat.groupBy("age_cat", "marital").count().sort($"coun
resultWithMarital: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [age_cat: s

scala> resultWithMarital.show()
+-------+--------+-----+
|age_cat| marital|count|
+-------+--------+-----+
|    mid| married| 2345|
|    mid|  single| 1710|
|    mid|divorced|  525|
|    old| married|  396|
|  young|  single|  193|
|    old|divorced|   97|
|  young| married|   14|
|    old|  single|    9|
+-------+--------+-----+
```