PROJECT 3

Market Analysis in Banking Domain

PROBLEM STATEMENT :

Background and Objective:

Your client, a Portuguese banking institution, ran a marketing campaign to convince potential customers to invest in a bank term deposit scheme. The marketing campaigns were based onphone calls. Often, the same customer was contacted more than once through phone, in order to assess if they would wantto subscribe to the bank term deposit or not. You have to perform the marketing analysis of the data generated by this campaign.

Domain: Banking (Market Analysis)

Analysis tasks to be done-:

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

- Load data and create a Spark data frame.
- Give marketing success rate (No. of people subscribed / totalno. of entries).
- Give marketing failure rate

- Give the maximum, mean, and minimum age of the averagetargeted customer.
- Check the quality of customers by checking average balance,median balance of customers.
- Check if age matters in marketing subscription for deposit.
- Check if marital status mattered for a subscription todeposit.
- Check if age and marital status together mattered for asubscription to deposit scheme.
- Do feature engineering for the bank and find the right ageeffect on the campaign.

WRITEUP :

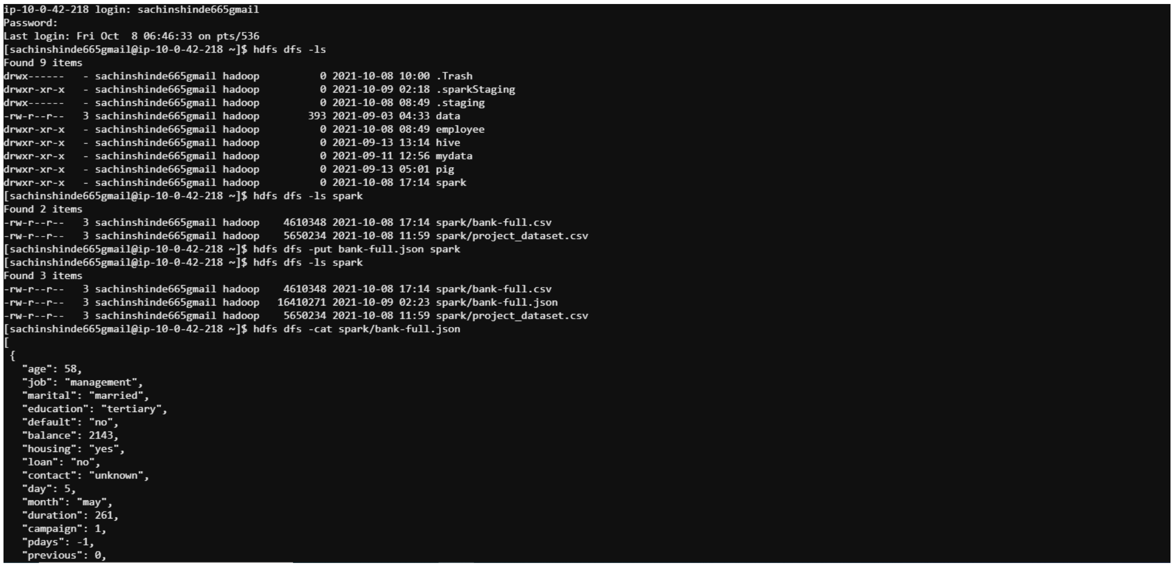
The project is related to the marketing analysis of the Portuguese Banking Institution. The goal of the project is to perform the analysis of the data generated by the marketing campaign.

The banking institution ran a marketing campaign to convince potential customers to invest in a bank 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 thebank term deposit or not.

The project uses Scala as a programming language, Hive as a data storage unit and Spark SQL functions to achieve the results.

SOURCE CODE WITH OUTPUT

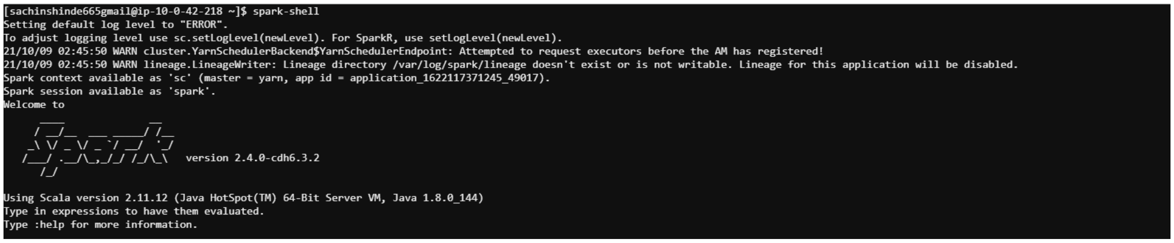
- **Load data and create a Spark data frame**



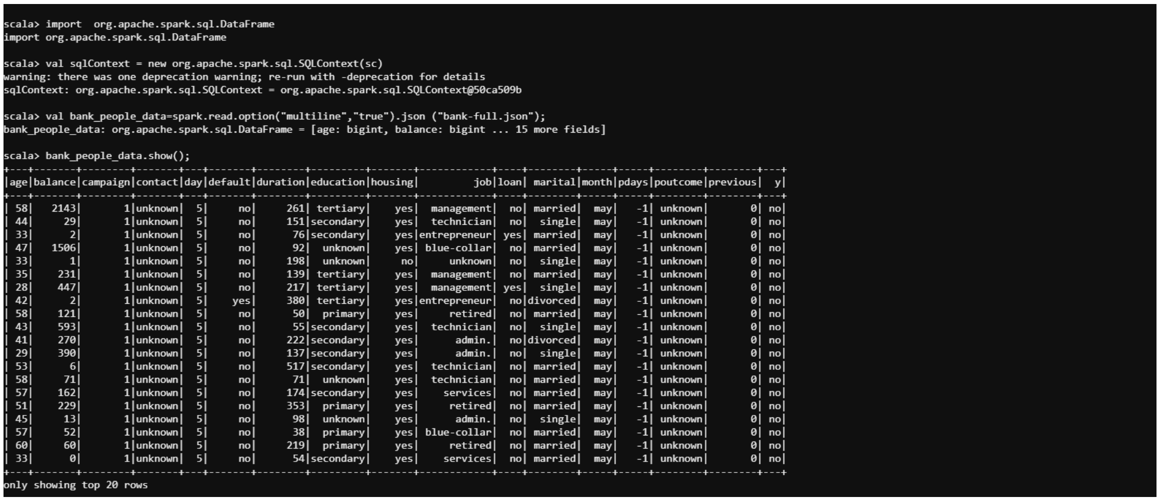
```
ip-10-0-42-218 login: sachinshinde665gmail
Password:
Last login: Fri Oct  8 06:46:33 on pts/536
[sachinshinde665gmail@ip-10-0-42-218 ~]$ hdfs dfs -ls
Found 9 items
drwx------   - sachinshinde665gmail hadoop          0 2021-10-08 10:00 .Trash
drwxr-xr-x   - sachinshinde665gmail hadoop          0 2021-10-09 02:18 .sparkStaging
drwx------   - sachinshinde665gmail hadoop          0 2021-10-08 08:49 .staging
-rw-r--r--   3 sachinshinde665gmail hadoop        393 2021-09-03 04:33 data
drwxr-xr-x   - sachinshinde665gmail hadoop          0 2021-10-08 08:49 employee
drwxr-xr-x   - sachinshinde665gmail hadoop          0 2021-09-13 13:14 hive
drwxr-xr-x   - sachinshinde665gmail hadoop          0 2021-09-11 12:56 mydata
drwxr-xr-x   - sachinshinde665gmail hadoop          0 2021-09-13 05:01 pig
drwxr-xr-x   - sachinshinde665gmail hadoop          0 2021-10-08 17:14 spark
[sachinshinde665gmail@ip-10-0-42-218 ~]$ hdfs dfs -ls spark
Found 2 items
-rw-r--r--   3 sachinshinde665gmail hadoop    4610348 2021-10-08 17:14 spark/bank-full.csv
-rw-r--r--   3 sachinshinde665gmail hadoop    5650234 2021-10-08 11:59 spark/project_dataset.csv
[sachinshinde665gmail@ip-10-0-42-218 ~]$ hdfs dfs -put bank-full.json spark
[sachinshinde665gmail@ip-10-0-42-218 ~]$ hdfs dfs -ls spark
Found 3 items
-rw-r--r--   3 sachinshinde665gmail hadoop    4610348 2021-10-08 17:14 spark/bank-full.csv
-rw-r--r--   3 sachinshinde665gmail hadoop   16410271 2021-10-09 02:23 spark/bank-full.json
-rw-r--r--   3 sachinshinde665gmail hadoop    5650234 2021-10-08 11:59 spark/project_dataset.csv
[sachinshinde665gmail@ip-10-0-42-218 ~]$ hdfs dfs -cat spark/bank-full.json
[
 {
   "age": 58,
   "job": "management",
   "marital": "married",
   "education": "tertiary",
   "default": "no",
   "balance": 2143,
   "housing": "yes",
   "loan": "no",
   "contact": "unknown",
   "day": 5,
   "month": "may",
   "duration": 261,
   "campaign": 1,
   "pdays": -1,
   "previous": 0,
```

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

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



```
[sachinshinde665gmail@ip-10-0-42-218 ~]$ spark-shell
Setting default log level to "ERROR".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
21/10/09 02:45:50 WARN cluster.YarnSchedulerBackend$YarnSchedulerEndpoint: Attempted to request executors before the AM has registered!
21/10/09 02:45:50 WARN lineage.LineageWriter: Lineage directory /var/log/spark/lineage doesn't exist or is not writable. Lineage for this application will be disabled.
Spark context available as 'sc' (master = yarn, app id = application_1622117371245_49017).
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> import  org.apache.spark.sql.DataFrame
import org.apache.spark.sql.DataFrame

scala> val sqlContext = new org.apache.spark.sql.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@50ca509b

scala> val bank_people_data=spark.read.option("multiline","true").json ("bank-full.json");
bank_people_data: org.apache.spark.sql.DataFrame = [age: bigint, balance: bigint ... 15 more fields]

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

```
scala> val bank_people_data=spark.read.option("multiline","true").json
("spark/bank-full.json");
```

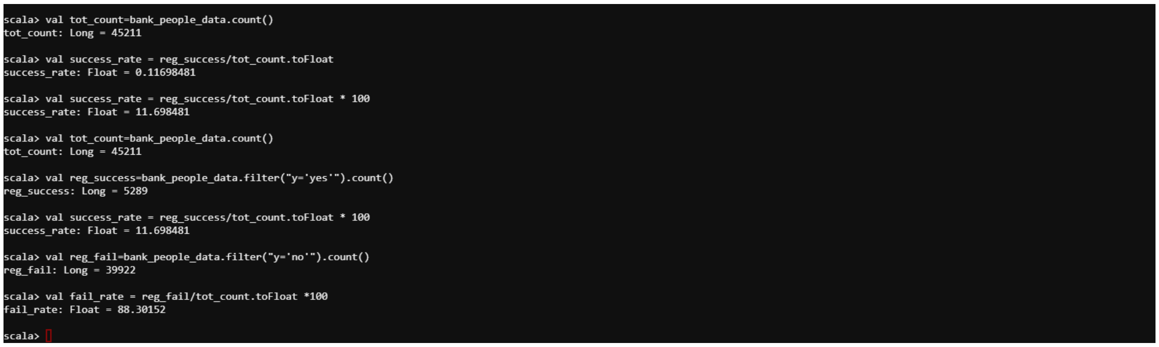
```
scala> bank_people_data.show();
```

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

```
scala> val tot_count=bank_people_data.count()
tot_count: Long = 45211

scala> val reg_success=bank_people_data.filter("y='yes'").count()
reg_success: Long = 5289

scala> val success_rate = reg_success/tot_count.toFloat * 100
success_rate: Float = 11.698481

scala> val reg_fail=bank_people_data.filter("y='no'").count()
```

```
reg_fail: Long = 39922
```

```
scala> val fail_rate = reg_fail/tot_count.toFloat *100
fail_rate: Float = 88.30152
```

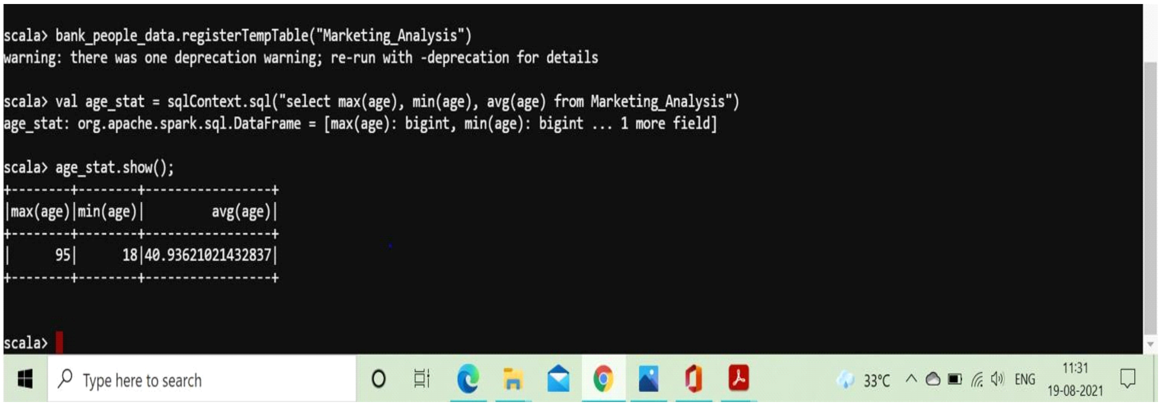


```
scala> val tot_count=bank_people_data.count()
tot_count: Long = 45211

scala> val success_rate = reg_success/tot_count.toFloat
success_rate: Float = 0.11698481

scala> val success_rate = reg_success/tot_count.toFloat * 100
success_rate: Float = 11.698481

scala> val tot_count=bank_people_data.count()
tot_count: Long = 45211

scala> val reg_success=bank_people_data.filter("y='yes'").count()
reg_success: Long = 5289

scala> val success_rate = reg_success/tot_count.toFloat * 100
success_rate: Float = 11.698481

scala> val reg_fail=bank_people_data.filter("y='no'").count()
reg_fail: Long = 39922

scala> val fail_rate = reg_fail/tot_count.toFloat *100
fail_rate: Float = 88.30152

scala>
```

- **Give the maximum, mean, and minimum age of the average targetedcustomer.**

```
scala> bank_people_data.registerTempTable("Marketing_Analysis")
```

```
scala> val age_stat = sqlContext.sql("select max(age), min(age), avg(age) from Marketing_Analysis")

age_stat: org.apache.spark.sql.DataFrame = [max(age): bigint, min(age): bigint ... 1more field]

scala> age_stat.show();
```

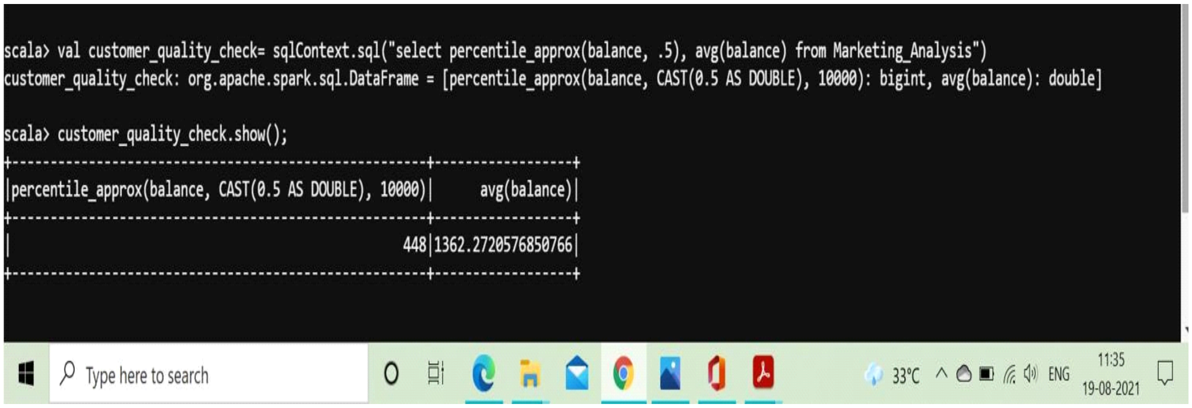


```
scala> bank_people_data.registerTempTable("Marketing_Analysis")
warning: there was one deprecation warning; re-run with -deprecation for details

scala> val age_stat = sqlContext.sql("select max(age), min(age), avg(age) from Marketing_Analysis")
age_stat: org.apache.spark.sql.DataFrame = [max(age): bigint, min(age): bigint ... 1 more field]

scala> age_stat.show();
+--------+--------+----------------+
|max(age)|min(age)|        avg(age)|
+--------+--------+----------------+
|      95|      18|40.93621021432837|
+--------+--------+----------------+


scala>
```

- **Check the quality of customers by checking average balance, medianbalance of customers.**

scala> val customer_quality_check= sqlContext.sql("select percentile_approx(balance, .5), avg(balance) from Marketing_Analysis")

scala> customer_quality_check.show();

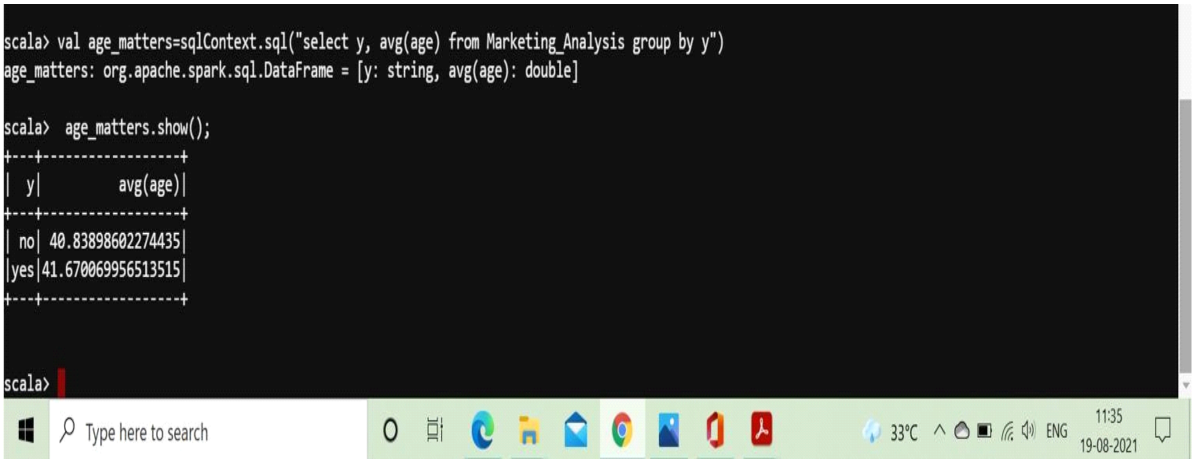


```
scala> val customer_quality_check= sqlContext.sql("select percentile_approx(balance, .5), avg(balance) from Marketing_Analysis")
customer_quality_check: org.apache.spark.sql.DataFrame = [percentile_approx(balance, CAST(0.5 AS DOUBLE), 10000): bigint, avg(balance): double]

scala> customer_quality_check.show();
+-----------------------------------------------------+-----------------+
|percentile_approx(balance, CAST(0.5 AS DOUBLE), 10000)|     avg(balance)|
+-----------------------------------------------------+-----------------+
|                                                  448|1362.2720576850766|
+-----------------------------------------------------+-----------------+
```

- **Check if age matters in marketing subscription for deposit.**

scala> val age_matters=sqlContext.sql("select y, avg(age) from Marketing_Analysis group by y")

scala> age_matters.show();

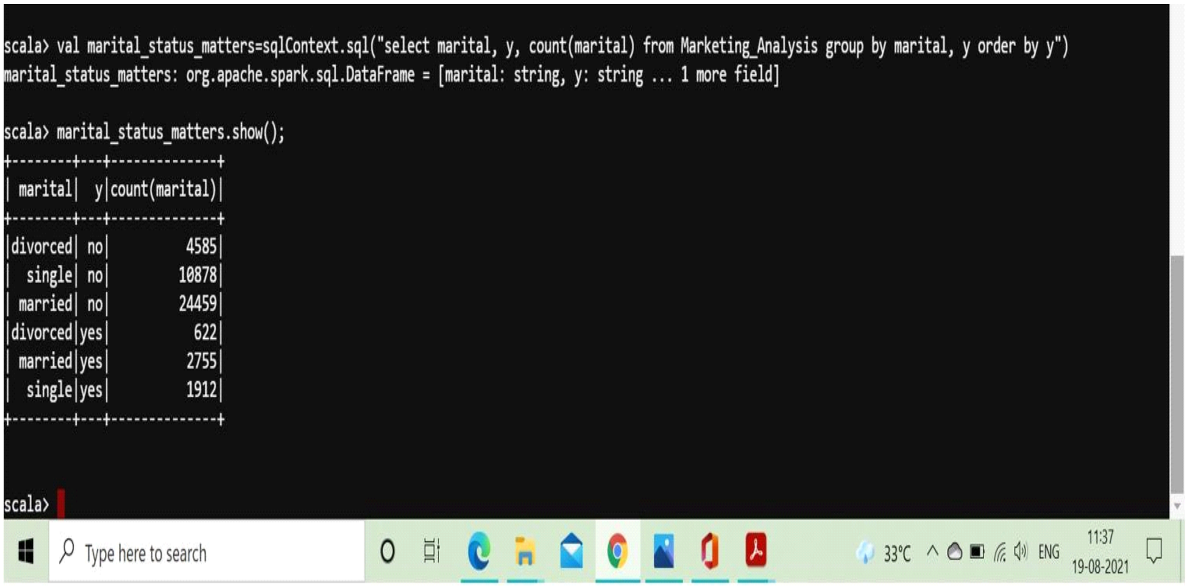


```
scala> val age_matters=sqlContext.sql("select y, avg(age) from Marketing_Analysis group by y")
age_matters: org.apache.spark.sql.DataFrame = [y: string, avg(age): double]

scala>  age_matters.show();
+---+-----------------+
|  y|         avg(age)|
+---+-----------------+
| no| 40.83898602274435|
|yes|41.670069956513515|
+---+-----------------+


scala>
```

- **Check if marital status mattered for a subscription to deposit.**

scala> val marital_status_matters=sqlContext.sql("select marital, y, count(marital) from Marketing_Analysis group by marital, y order by y")

scala> marital_status_matters.show();

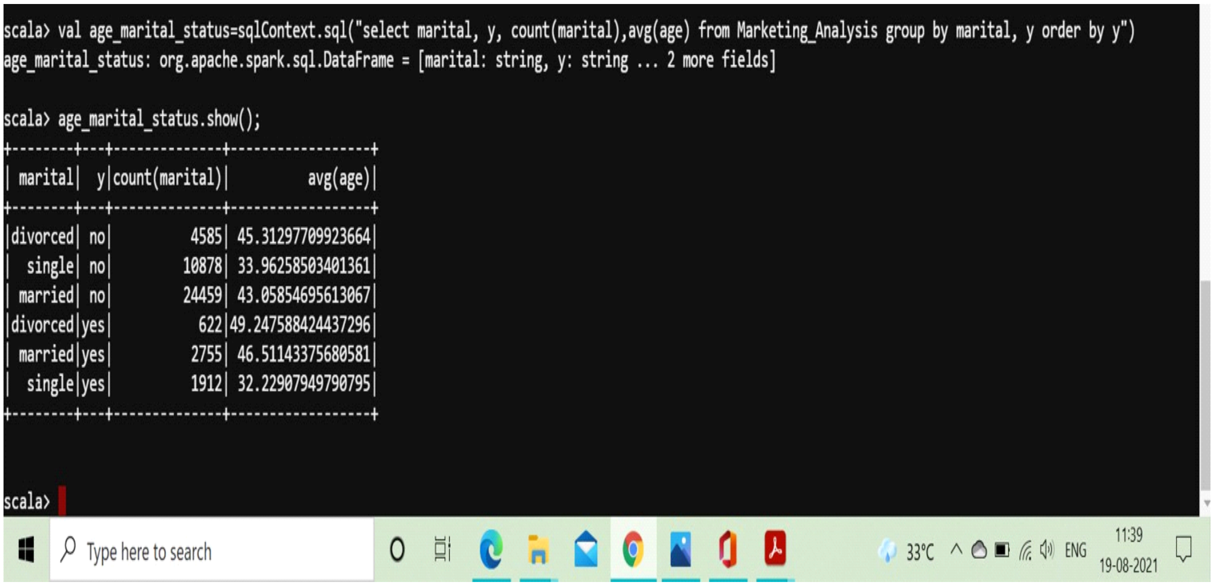


```
scala> val marital_status_matters=sqlContext.sql("select marital, y, count(marital) from Marketing_Analysis group by marital, y order by y")
marital_status_matters: org.apache.spark.sql.DataFrame = [marital: string, y: string ... 1 more field]

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


scala>
```

- **Check if age and marital status together mattered for a subscription todeposit scheme.**

scala> val age_marital_status=sqlContext.sql("select marital, y, count(marital),avg(age) from Marketing_Analysis group by marital, y order by y")

scala> age_marital_status.show();



```
scala> val age_marital_status=sqlContext.sql("select marital, y, count(marital),avg(age) from Marketing_Analysis group by marital, y order by y")
age_marital_status: org.apache.spark.sql.DataFrame = [marital: string, y: string ... 2 more fields]

scala> age_marital_status.show();
+--------+---+--------------+-----------------+
| marital|  y|count(marital)|         avg(age)|
+--------+---+--------------+-----------------+
|divorced| no|          4585| 45.31297709923664|
|  single| no|         10878| 33.96258503401361|
| married| no|         24459| 43.05854695613067|
|divorced|yes|           622|49.247588424437296|
| married|yes|          2755| 46.51143375680581|
|  single|yes|          1912| 32.22907949790795|
+--------+---+--------------+-----------------+


scala>
```

- **Do feature engineering for the bank and find the right age effect on thecampaign.**

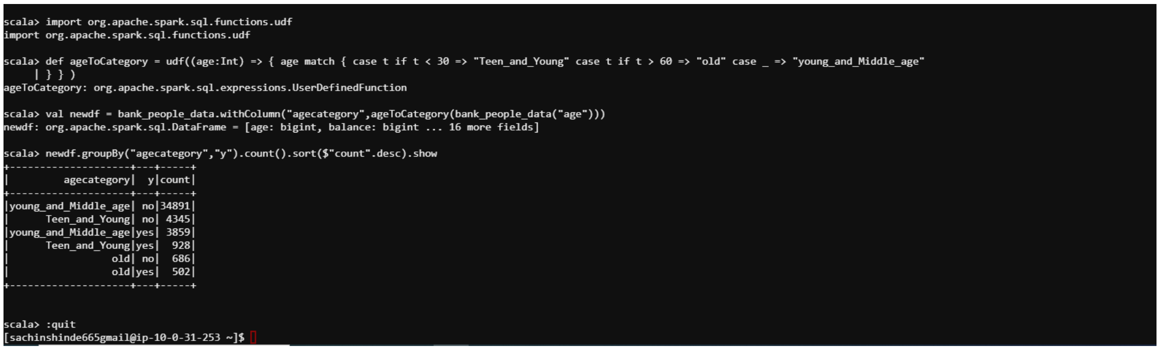
scala> import org.apache.spark.sql.functions.udf // import lib for UDF

scala> def ageToCategory = udf((age:Int) => { age match { case t if t < 30 => "Teen_and_Young"case t if t > 60 => "old" case _ => "young_and_Middle_age"

} }) //create UDF

scala> val newdf = bank_people_data.withColumn("agecategory",ageToCategory(bank_people_data ("age")))
//apply udf to data frame

scala> newdf.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 t if t < 30 => "Teen_and_Young" case t if t > 60 => "old" case _ => "young_and_Middle_age"
     | } } )
ageToCategory: org.apache.spark.sql.expressions.UserDefinedFunction

scala> val newdf = bank_people_data.withColumn("agecategory",ageToCategory(bank_people_data("age")))
newdf: org.apache.spark.sql.DataFrame = [age: bigint, balance: bigint ... 16 more fields]

scala> newdf.groupBy("agecategory","y").count().sort($"count".desc).show
+--------------------+---+-----+
|         agecategory|  y|count|
+--------------------+---+-----+
|young_and_Middle_age| no|34891|
|      Teen_and_Young| no| 4345|
|young_and_Middle_age|yes| 3859|
|      Teen_and_Young|yes|  928|
|                 old| no|  686|
|                 old|yes|  502|
+--------------------+---+-----+


scala> :quit
[sachinshinde665gmail@ip-10-0-31-253 ~]$ 
```