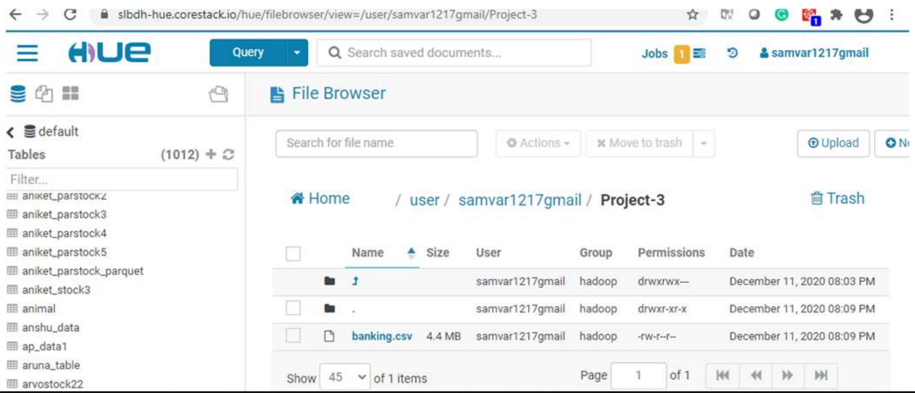
Big Data Hadoop and Spark Developer

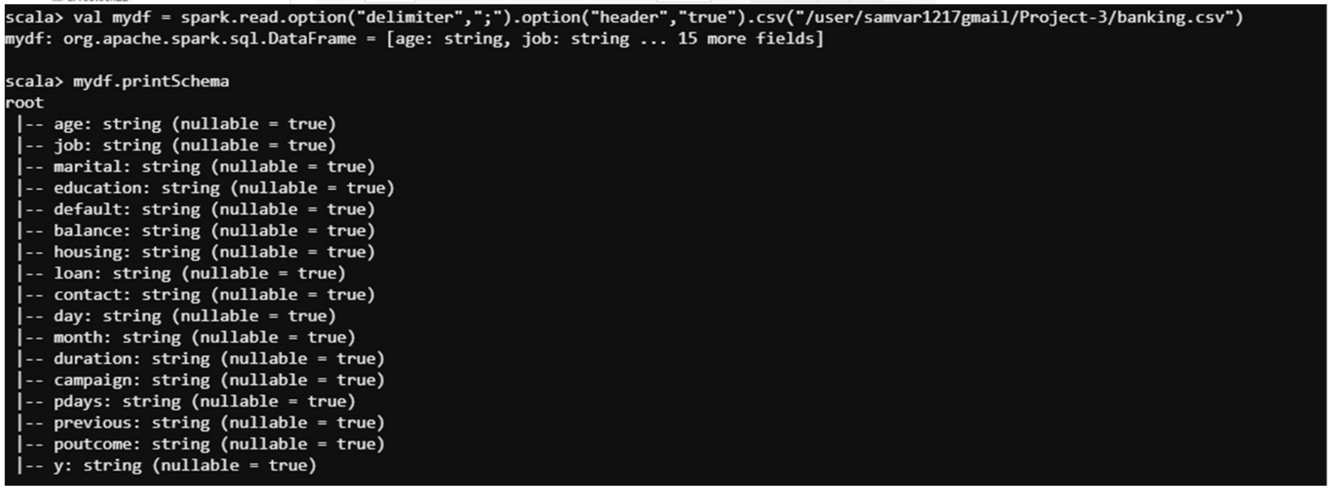
Project-3: Market Analysis in Banking Domain

Submitted by: Sagar Samaria
Date: Dec-12-2020

1.Load data and create a Spark data frame



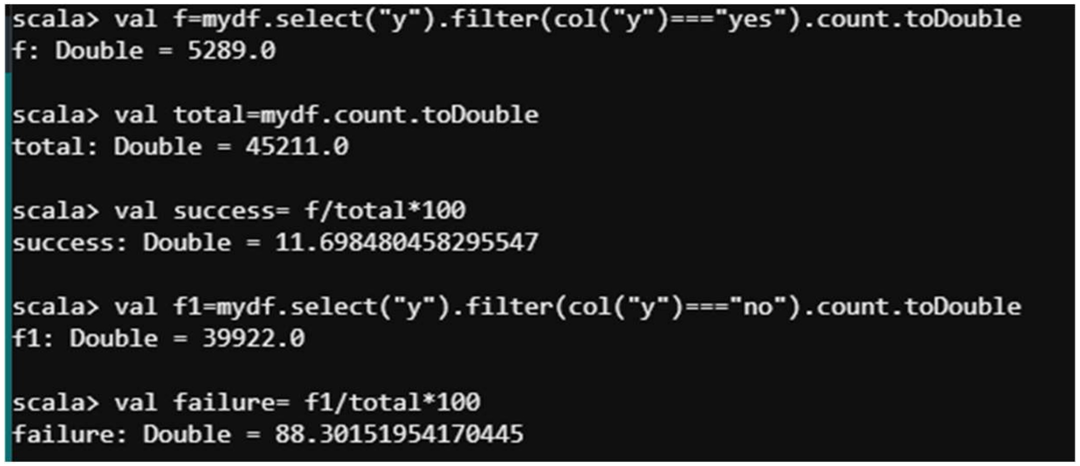
Name	Size	User	Group	Permissions	Date
↥		samvar1217gmail	hadoop	drwxrwx---	December 11, 2020 08:03 PM
.		samvar1217gmail	hadoop	drwxr-xr-x	December 11, 2020 08:09 PM
banking.csv	4.4 MB	samvar1217gmail	hadoop	-rw-r--r--	December 11, 2020 08:09 PM



```
scala> val mydf = spark.read.option("delimiter",";").option("header","true").csv("/user/samvar1217gmail/Project-3/banking.csv")
mydf: org.apache.spark.sql.DataFrame = [age: string, job: string ... 15 more fields]

scala> mydf.printSchema
root
 |-- age: string (nullable = true)
 |-- job: string (nullable = true)
 |-- marital: string (nullable = true)
 |-- education: string (nullable = true)
 |-- default: string (nullable = true)
 |-- balance: string (nullable = true)
 |-- housing: string (nullable = true)
 |-- loan: string (nullable = true)
 |-- contact: string (nullable = true)
 |-- day: string (nullable = true)
 |-- month: string (nullable = true)
 |-- duration: string (nullable = true)
 |-- campaign: string (nullable = true)
 |-- pdays: string (nullable = true)
 |-- previous: string (nullable = true)
 |-- poutcome: string (nullable = true)
 |-- y: string (nullable = true)
```

Query-1: val mydf = spark.read.option("delimiter",";").option("header","true").csv("/user/samvar1217gmail/Project-3/banking.csv")
Query-2: mydf.printSchema

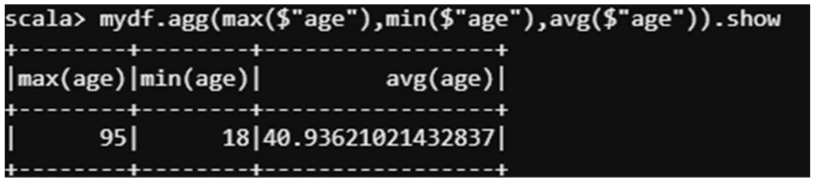
2.Give marketing success rate (No. of people subscribed / total no. of entries)
•Give marketing failure rate



```
scala> val f=mydf.select("y").filter(col("y")==="yes").count.toDouble
f: Double = 5289.0

scala> val total=mydf.count.toDouble
total: Double = 45211.0

scala> val success= f/total*100
success: Double = 11.698480458295547

scala> val f1=mydf.select("y").filter(col("y")==="no").count.toDouble
f1: Double = 39922.0

scala> val failure= f1/total*100
failure: Double = 88.30151954170445
```

<u>Query-1:</u> val f=mydf.select("y").filter(col("y")==="yes").count.toDouble
<u>Query-2:</u> val total=mydf.count.toDouble
<u>Query-3:</u> val success= f/total*100
<u>Query-4:</u> val f1=mydf.select("y").filter(col("y")==="no").count.toDouble
<u>Query-5:</u> val failure= f1/total*100

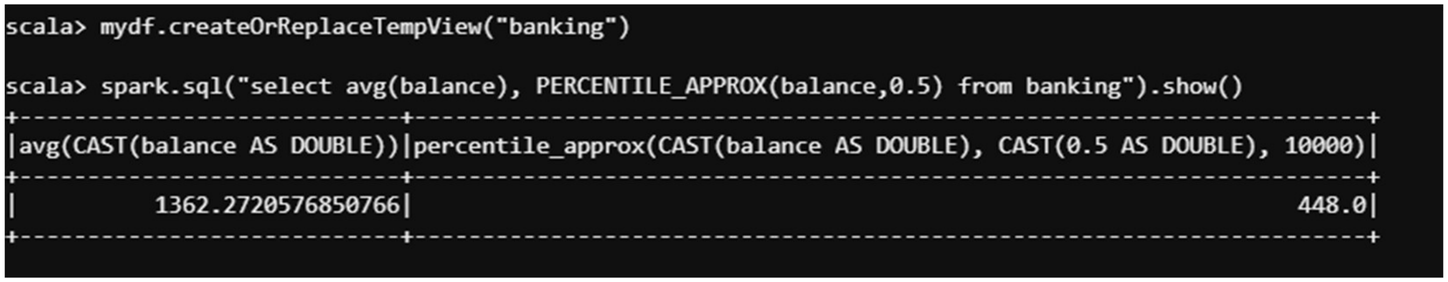
1.Give the maximum, mean, and minimum age of the average targeted customer



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

Query-1: mydf.createOrReplaceTempView("banking")
Query-2: mydf.agg(max($"age"),min($"age"),avg($"age")).show

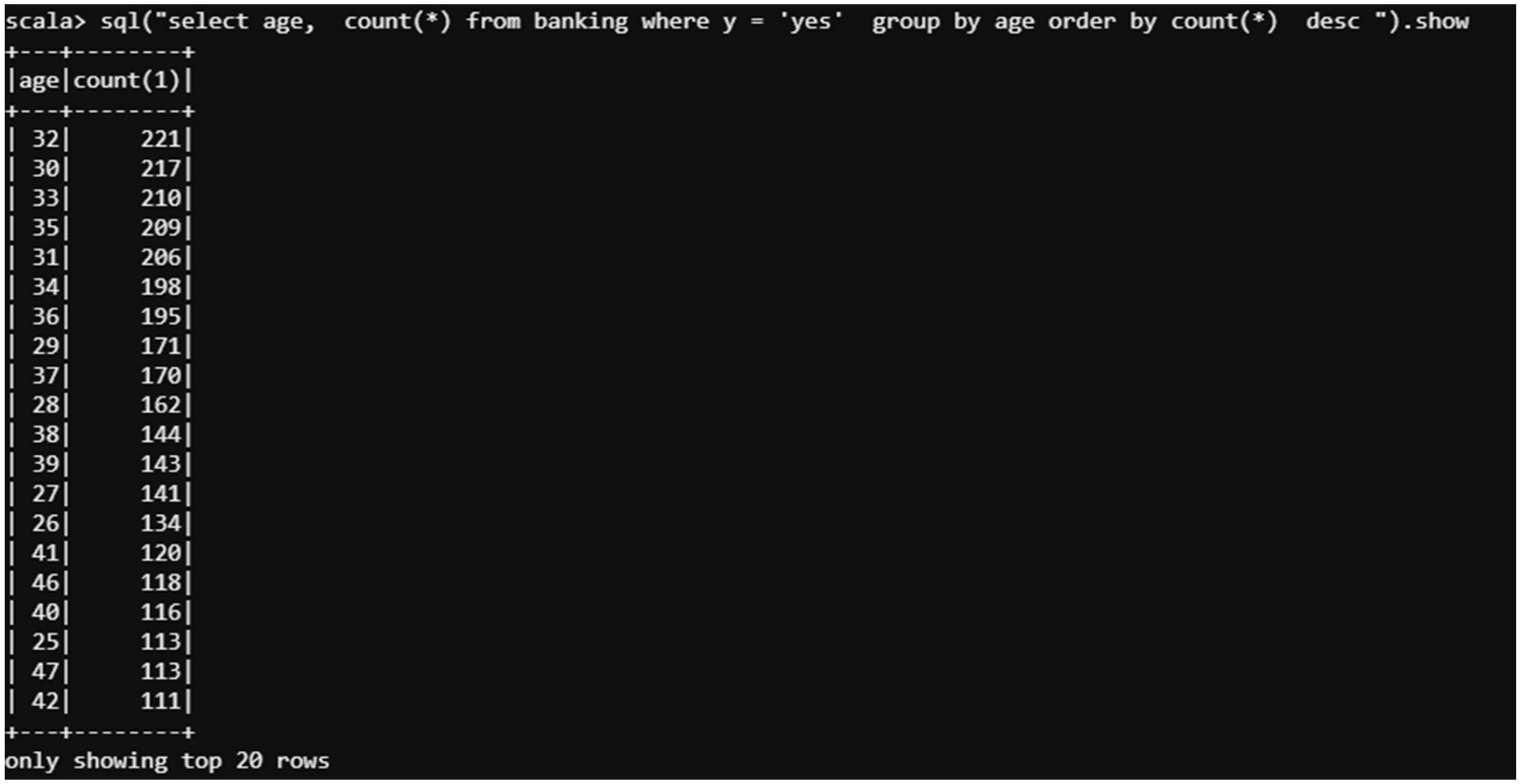
2. Check the quality of customers by checking average balance, median balance of customers



```
scala> mydf.createOrReplaceTempView("banking")

scala> spark.sql("select avg(balance), PERCENTILE_APPROX(balance,0.5) from banking").show()
+--------------------------+------------------------------------------------------------------+
|avg(CAST(balance AS DOUBLE))|percentile_approx(CAST(balance AS DOUBLE), CAST(0.5 AS DOUBLE), 10000)|
+--------------------------+------------------------------------------------------------------+
|         1362.2720576850766|                                                             448.0|
+--------------------------+------------------------------------------------------------------+
```

Query-1: mydf.createOrReplaceTempView("banking")
Query-2: spark.sql("select avg(balance), PERCENTILE_APPROX(balance,0.5) from banking").show()

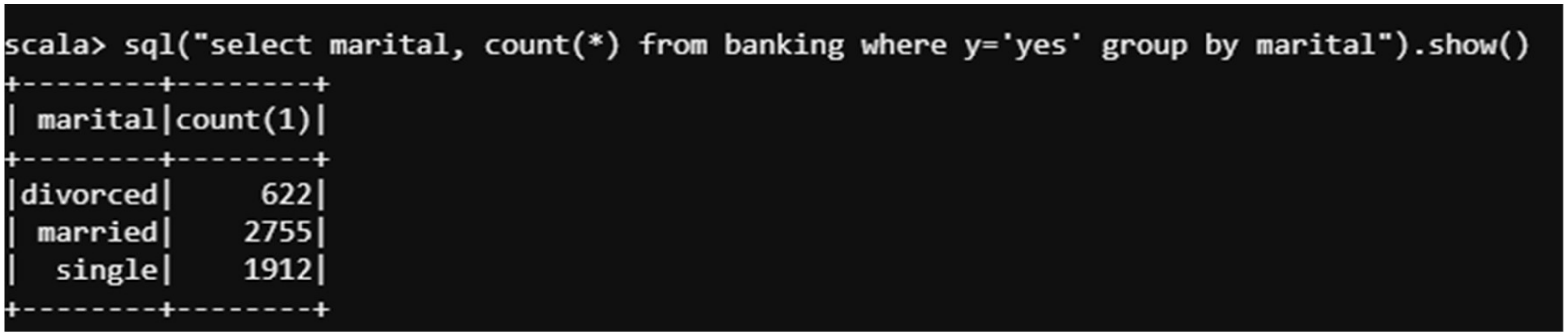
3.Check if age matters in marketing subscription for deposit



```
scala> sql("select age,  count(*) from banking where y = 'yes'  group by age order by count(*)  desc ").show
+---+--------+
|age|count(1)|
+---+--------+
| 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|
| 25|     113|
| 47|     113|
| 42|     111|
+---+--------+
only showing top 20 rows
```

Query-1: sql("select age, count(*) from banking where y = 'yes' group by age order by count(*) desc ").show

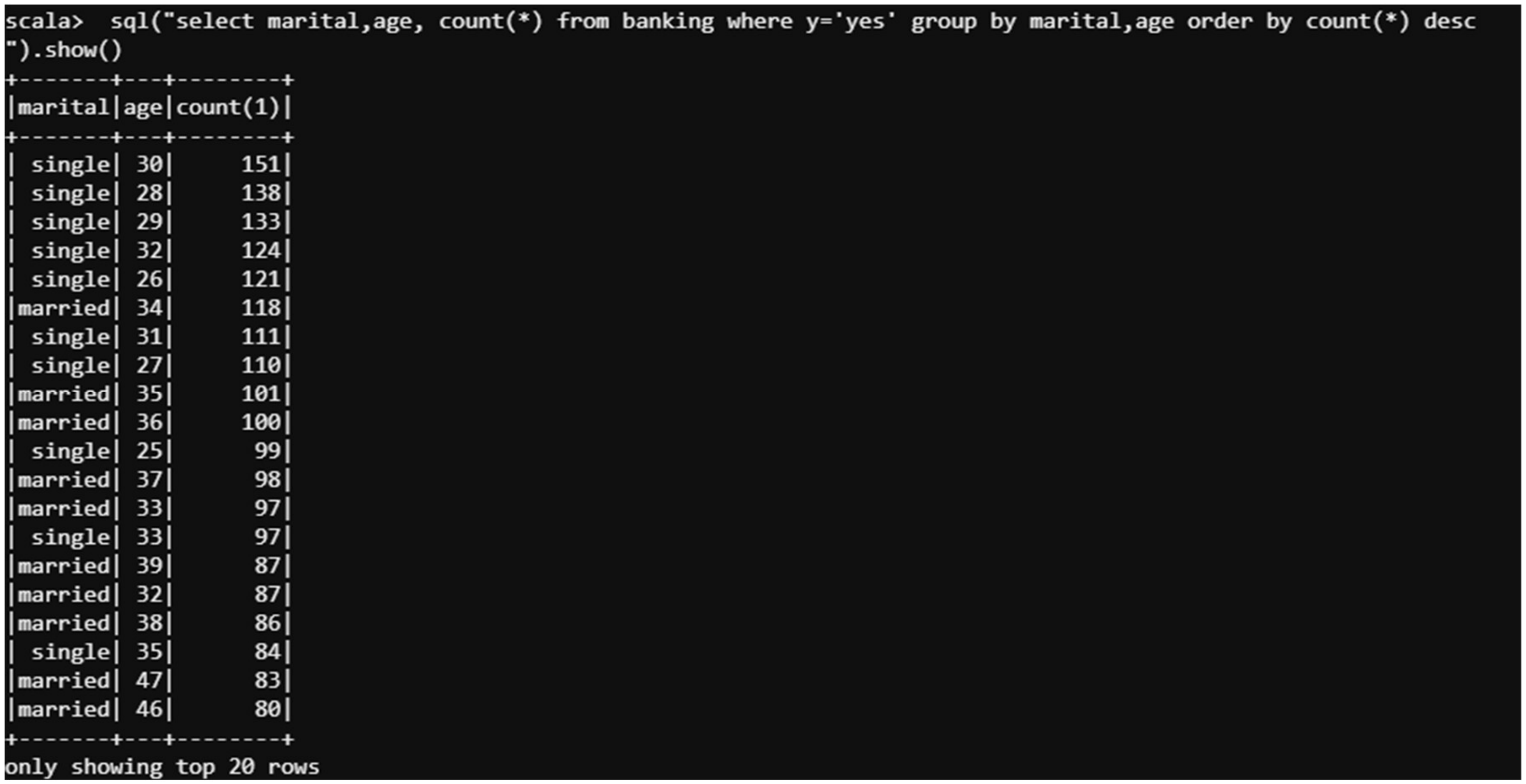
4. Check if marital status mattered for a subscription to deposit



```
scala> sql("select marital, count(*) from banking where y='yes' group by marital").show()
+--------+--------+
| marital|count(1)|
+--------+--------+
|divorced|     622|
| married|    2755|
|  single|    1912|
+--------+--------+
```

<u>Query-1:</u> sql("select marital, count(*) from banking where y='yes' group by marital").show()

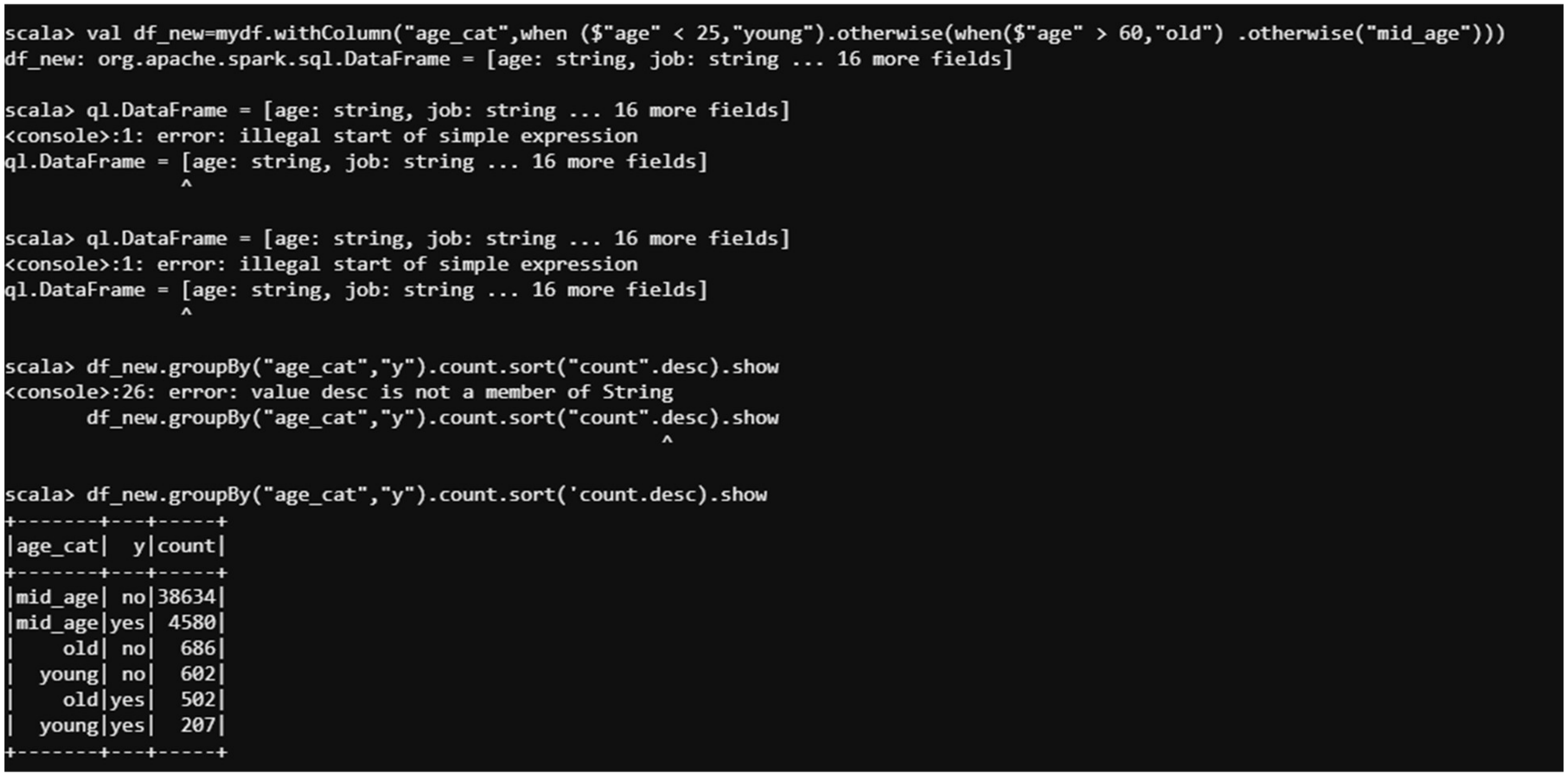
5. Check if age and marital status together mattered for a subscription to deposit scheme



```
scala>  sql("select marital,age, count(*) from banking where y='yes' group by marital,age order by count(*) desc
").show()
+-------+---+--------+
|marital|age|count(1)|
+-------+---+--------+
| single| 30|     151|
| single| 28|     138|
| single| 29|     133|
| single| 32|     124|
| single| 26|     121|
|married| 34|     118|
| single| 31|     111|
| single| 27|     110|
|married| 35|     101|
|married| 36|     100|
| single| 25|      99|
|married| 37|      98|
|married| 33|      97|
| single| 33|      97|
|married| 39|      87|
|married| 32|      87|
|married| 38|      86|
| single| 35|      84|
|married| 47|      83|
|married| 46|      80|
+-------+---+--------+
only showing top 20 rows
```

<u>Query-1:</u> sql("select age, marital, count(*) from banking where y = 'yes' group by age, marital order by count(*) desc ").show

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



```
scala> val df_new=mydf.withColumn("age_cat",when ($"age" < 25,"young").otherwise(when($"age" > 60,"old") .otherwise("mid_age")))
df_new: org.apache.spark.sql.DataFrame = [age: string, job: string ... 16 more fields]

scala> ql.DataFrame = [age: string, job: string ... 16 more fields]
<console>:1: error: illegal start of simple expression
ql.DataFrame = [age: string, job: string ... 16 more fields]
               ^

scala> ql.DataFrame = [age: string, job: string ... 16 more fields]
<console>:1: error: illegal start of simple expression
ql.DataFrame = [age: string, job: string ... 16 more fields]
               ^

scala> df_new.groupBy("age_cat","y").count.sort("count".desc).show
<console>:26: error: value desc is not a member of String
       df_new.groupBy("age_cat","y").count.sort("count".desc).show
                                                        ^

scala> df_new.groupBy("age_cat","y").count.sort('count.desc).show
+-------+---+-----+
|age_cat|  y|count|
+-------+---+-----+
|mid_age| no|38634|
|mid_age|yes| 4580|
|    old| no|  686|
|  young| no|  602|
|    old|yes|  502|
|  young|yes|  207|
+-------+---+-----+
```

Query-1: val df_new=mydf.withColumn("age_cat",when ($"age" < 25,"young").otherwise(when($"age" > 60,"old") .otherwise("mid_age")))
Query-2: df_new.groupBy("age_cat","y").count.sort('count.desc).show