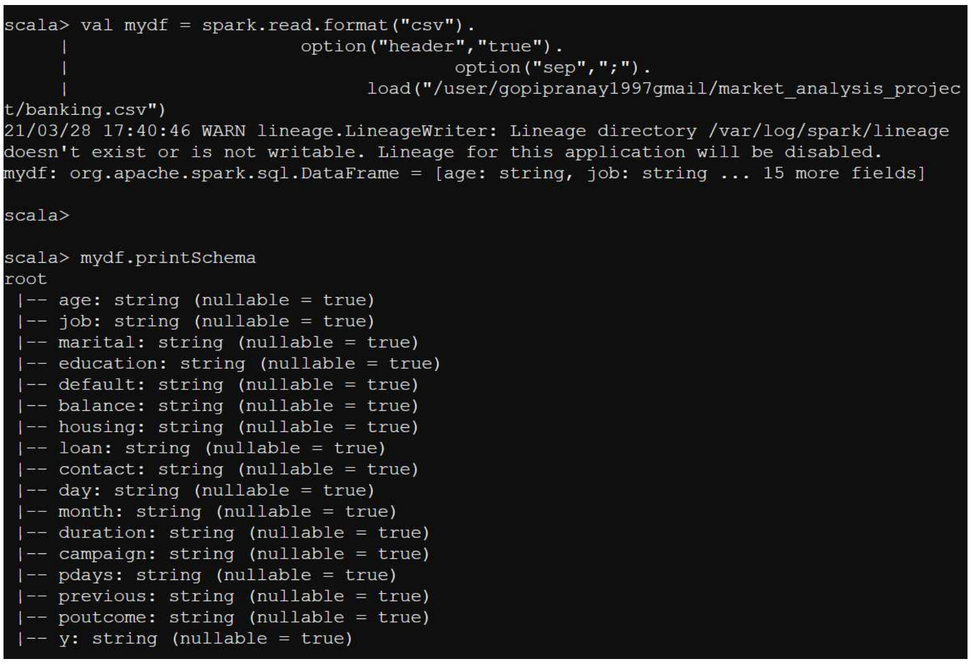
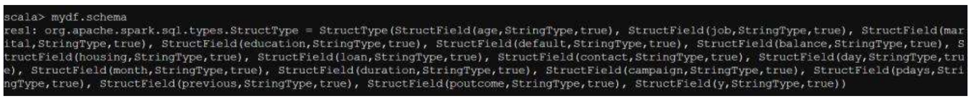
Market Analysis in Banking Domain (Screenshots)

1.Load data and create a Spark data frame

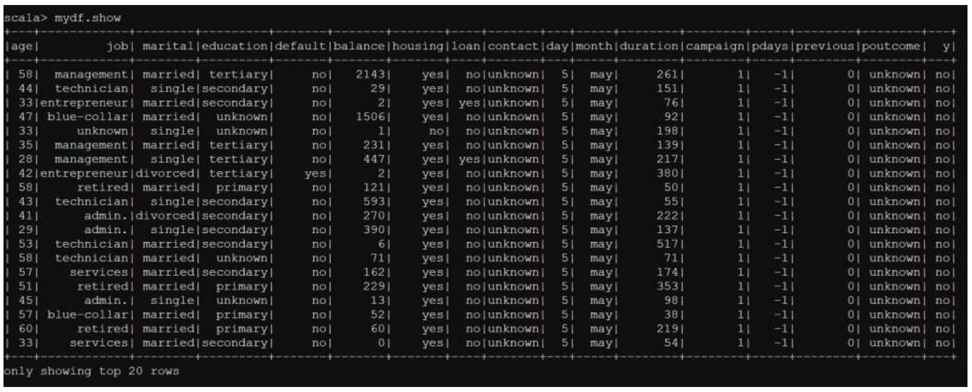
ANS:



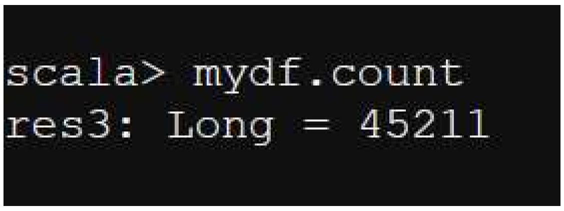
```
scala> val mydf = spark.read.format("csv").
     |                         option("header","true").
     |                                      option("sep",";").
     |                               load("/user/gopipranay1997gmail/market_analysis_projec
t/banking.csv")
21/03/28 17:40:46 WARN lineage.LineageWriter: Lineage directory /var/log/spark/lineage
doesn't exist or is not writable. Lineage for this application will be disabled.
mydf: org.apache.spark.sql.DataFrame = [age: string, job: string ... 15 more fields]

scala>

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)
```



```
scala> mydf.schema
res1: org.apache.spark.sql.types.StructType = StructType(StructField(age,StringType,true), StructField(job,StringType,true), StructField(mar
ital,StringType,true), StructField(education,StringType,true), StructField(default,StringType,true), StructField(balance,StringType,true), S
tructField(housing,StringType,true), StructField(loan,StringType,true), StructField(contact,StringType,true), StructField(day,StringType,tru
e), StructField(month,StringType,true), StructField(duration,StringType,true), StructField(campaign,StringType,true), StructField(pdays,Stri
ngType,true), StructField(previous,StringType,true), StructField(poutcome,StringType,true), StructField(y,StringType,true))
```

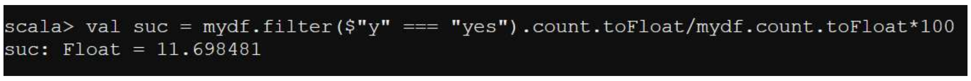


```
scala> mydf.show
+---+------------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+---+
|age|         job| marital|education|default|balance|housing|loan|contact|day|month|duration|campaign|pdays|previous|poutcome|  y|
+---+------------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+---+
| 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
```



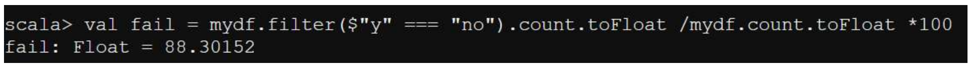
```
scala> mydf.count
res3: Long = 45211
```

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

ANS: 

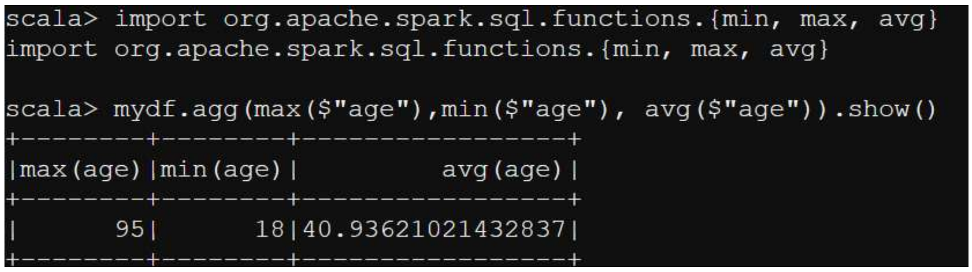
```
scala> val suc = mydf.filter($"y" === "yes").count.toFloat/mydf.count.toFloat*100
suc: Float = 11.698481
```

- Give marketing failure rate

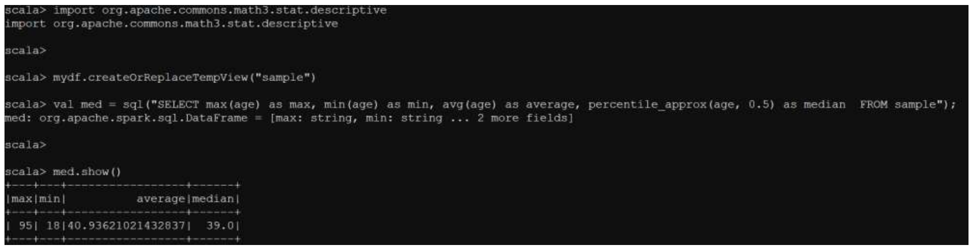


```
scala> val fail = mydf.filter($"y" === "no").count.toFloat /mydf.count.toFloat *100
fail: Float = 88.30152
```

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

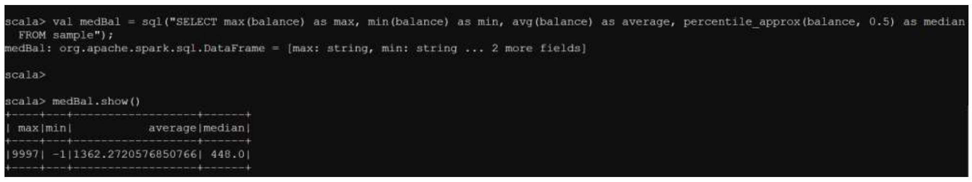
ANS: 

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

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



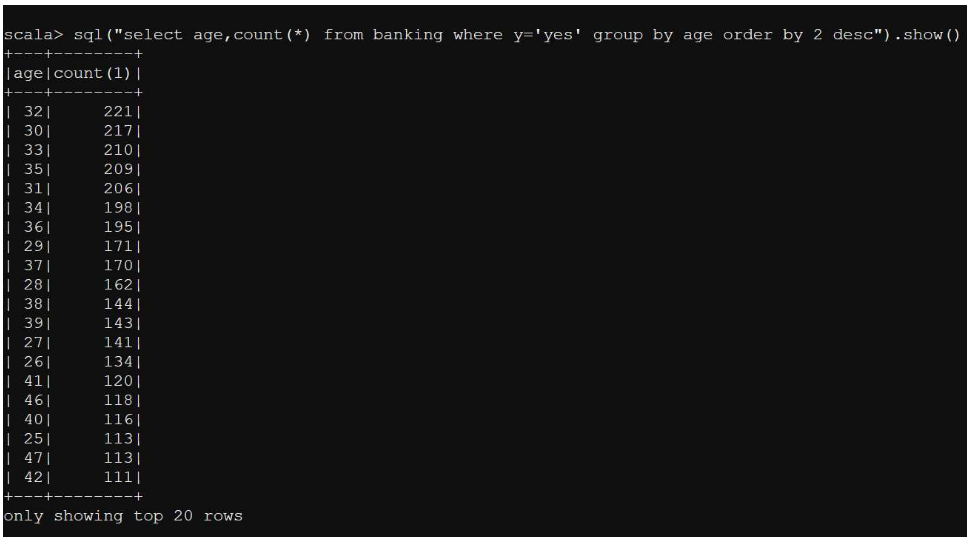
```
scala> import org.apache.commons.math3.stat.descriptive
import org.apache.commons.math3.stat.descriptive

scala>

scala> mydf.createOrReplaceTempView("sample")

scala> val med = sql("SELECT max(age) as max, min(age) as min, avg(age) as average, percentile_approx(age, 0.5) as median  FROM sample");
med: org.apache.spark.sql.DataFrame = [max: string, min: string ... 2 more fields]

scala>

scala> med.show()
+---+---+-----------------+------+
|max|min|          average|median|
+---+---+-----------------+------+
| 95| 18|40.93621021432837|  39.0|
+---+---+-----------------+------+
```

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

ANS: 

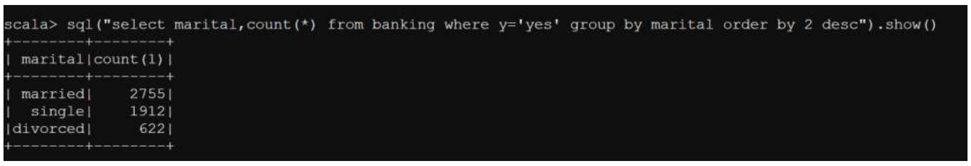
```
scala> val medBal = sql("SELECT max(balance) as max, min(balance) as min, avg(balance) as average, percentile_approx(balance, 0.5) as median
  FROM sample");
medBal: org.apache.spark.sql.DataFrame = [max: string, min: string ... 2 more fields]

scala>

scala> medBal.show()
+----+---+------------------+------+
| max|min|           average|median|
+----+---+------------------+------+
|9997| -1|1362.2720576850766| 448.0|
+----+---+------------------+------+
```

4. Check if age matters in marketing subscription for deposit

ANS: 

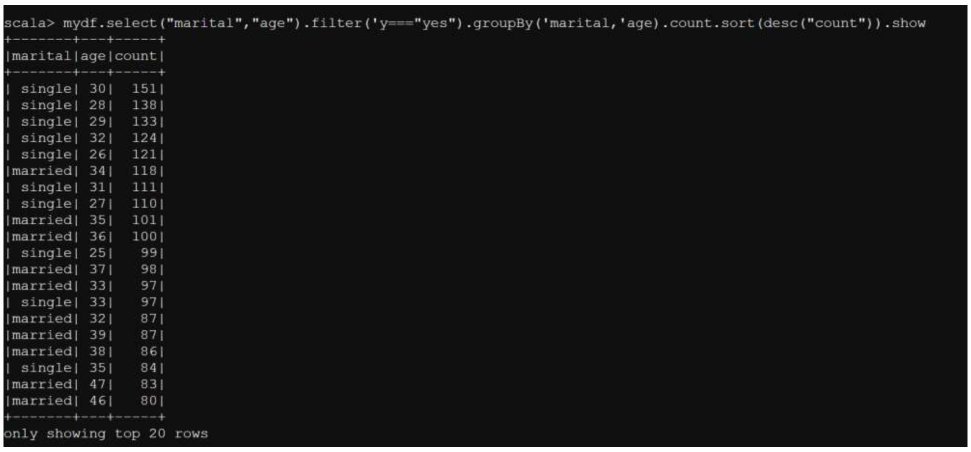
```
scala> sql("select age,count(*) from banking where y='yes' group by age order by 2 desc").show()
+---+--------+
|age|count(1)|
+---+--------+
| 32|     221|
| 30|     217|
| 33|     210|
| 35|     209|
| 31|     206|
| 34|     198|
| 36|     195|
| 29|     171|
| 37|     170|
| 28|     162|
| 38|     144|
| 39|     143|
| 27|     141|
| 26|     134|
| 41|     120|
| 46|     118|
| 40|     116|
| 25|     113|
| 47|     113|
| 42|     111|
+---+--------+
only showing top 20 rows
```

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

ANS: 

```
scala> sql("select marital,count(*) from banking where y='yes' group by marital order by 2 desc").show()
+--------+--------+
| marital|count(1)|
+--------+--------+
| married|    2755|
|  single|    1912|
|divorced|     622|
+--------+--------+
```

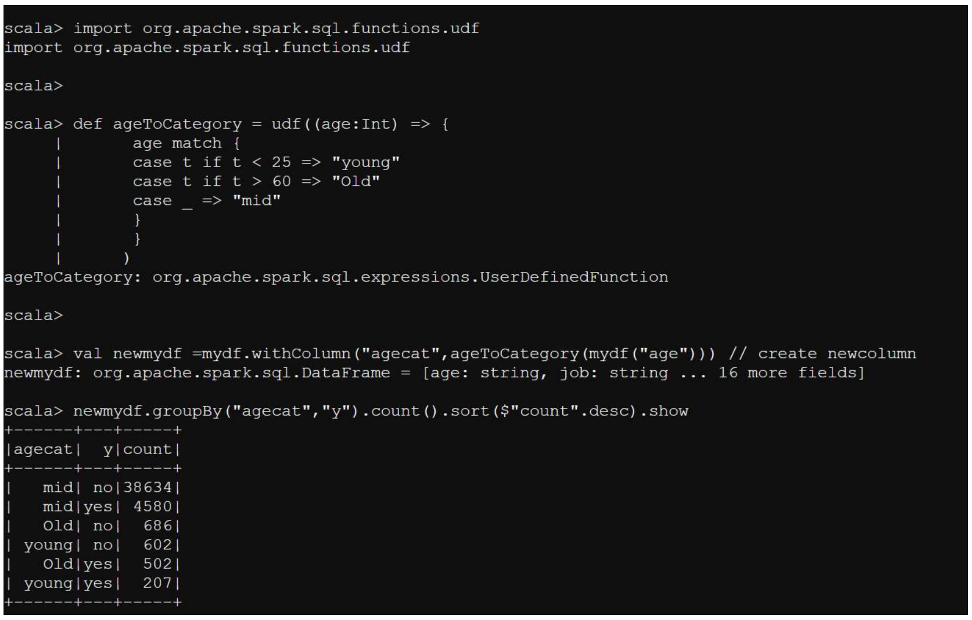
6. Check if age and marital status together mattered for a subscription to deposit scheme

ANS: 

```
scala> mydf.select("marital","age").filter('y==="yes").groupBy('marital,'age).count.sort(desc("count")).show
+-------+---+-----+
|marital|age|count|
+-------+---+-----+
| 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| 32|   87|
|married| 39|   87|
|married| 38|   86|
| single| 35|   84|
|married| 47|   83|
|married| 46|   80|
+-------+---+-----+
only showing top 20 rows
```

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

ANS:



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

scala>

scala> def ageToCategory = udf((age:Int) => {
     |        age match {
     |        case t if t < 25 => "young"
     |        case t if t > 60 => "Old"
     |        case _ => "mid"
     |        }
     |        }
     |       )
ageToCategory: org.apache.spark.sql.expressions.UserDefinedFunction

scala>

scala> val newmydf =mydf.withColumn("agecat",ageToCategory(mydf("age"))) // create newcolumn
newmydf: org.apache.spark.sql.DataFrame = [age: string, job: string ... 16 more fields]

scala> newmydf.groupBy("agecat","y").count().sort($"count".desc).show
+------+---+-----+
|agecat|  y|count|
+------+---+-----+
|   mid| no|38634|
|   mid|yes| 4580|
|   Old| no|  686|
| young| no|  602|
|   Old|yes|  502|
| young|yes|  207|
+------+---+-----+
```