HARISH GANESAN

PROJECT CODE

Market Analysis in Banking Domain

 # connect to spark2 shell

 # The --master option specifies the master URL for a distributed cluster, or local to run locally with one thread, or local[N] to run locally with N threads. You should start by using local for testing. Since many users are using lab at this time, we are using local so that connection interruptions don't happen

```
spark2-shell --master local
```

 # call/reference packages that deal with csv or xml and help us to smoothly work with dataframes

```
spark2-shell --master local --packages com.databricks:spark-csv_2.10:1.4.0,com.databricks:spark-xml_2.10:0.4.1
```

 # importing data into a dataframe df

```
val df = spark.read.format("csv").option("inferSchema","true").option("header","true").option("delimiter",",").load("/user/ajaykuma24_gmail/common4all/resources/Bank_full.csv")
```

#get the metadata by printing schema of the dataframe

df.printSchema

#check the top 5 rows of the dataframe to check contents of it and see if import has happened seamlessly from raw data to this spark interface

df.show(5)

#total count of number of records of dataframe

val totCount = df.count()

#Getting the total count as double format

val totCount = df.count().toDouble

#success rate count

 # Converting long format to double format as we will deal with ratios

val successC = df.filter($"y" === "yes").count().toDouble

#success rate as percentage

val successP = successC/totCount *100

#Fail rate count

val failC = df.filter($"y" === "no").count().toDouble

#fail rate as percentage

```
val failP = 100- successP
```

#optionally using group by checking counts

```
df.groupBy("y").count().show()
```

#give the max min and mean of average targeted customer

```
df.select(max("age") as "max",min("age") as "min",mean("age") as "mean").show()
```

#explore balance column

```
df.select("balance").show()
```

#getting the average balance

```
df.select(avg("balance")).show()
```

#creating temporary view

```
df.createOrReplaceTempView("bankdata")
```

 # retrieving 1 line from view

```
spark.sql("select * from bankdata limit 1").show()
```

 # getting median of the balance

```
spark.sql("select percentile(balance,0.5) as median from bankdata").show()
```

#By age grouping the response

```
df.groupBy("age","y").count().show()
```

#sorting the above by descending order of count

```
df.groupBy("age","y").count().sort($"count".desc).show()
```

 # Checking just by the response variable counts

```
df.groupBy("y").count().show()
```

#Aggregating the average of age

```
df.groupBy("y").agg(avg("age")).show()
```

#Exploring marital groups counts

```
df.groupBy("marital").count().show()
```

#sorting the above by descending order of count

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

#getting the marital and age group and response counts sorted in descending order

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

#calling the udf package below to create user defined function

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

#creating user defined function

```
def ageToCategory = udf((age: Int) => {
    age match {
    case n if n <= 30 => "young"
    case n if n >= 65 => "old"
    case n if n >30 && n <65 => "mid"
    }
    }
```

 # We are comparing the newly created dataframe with the past. The age bin is added

```
df.show(2)
newdf.show(2)
```

 # getting the summary stats of agegroup and sorted in descending fashion

```
newdf.groupBy("ageGroup","y").count().sort($"count".desc).show(25)
```