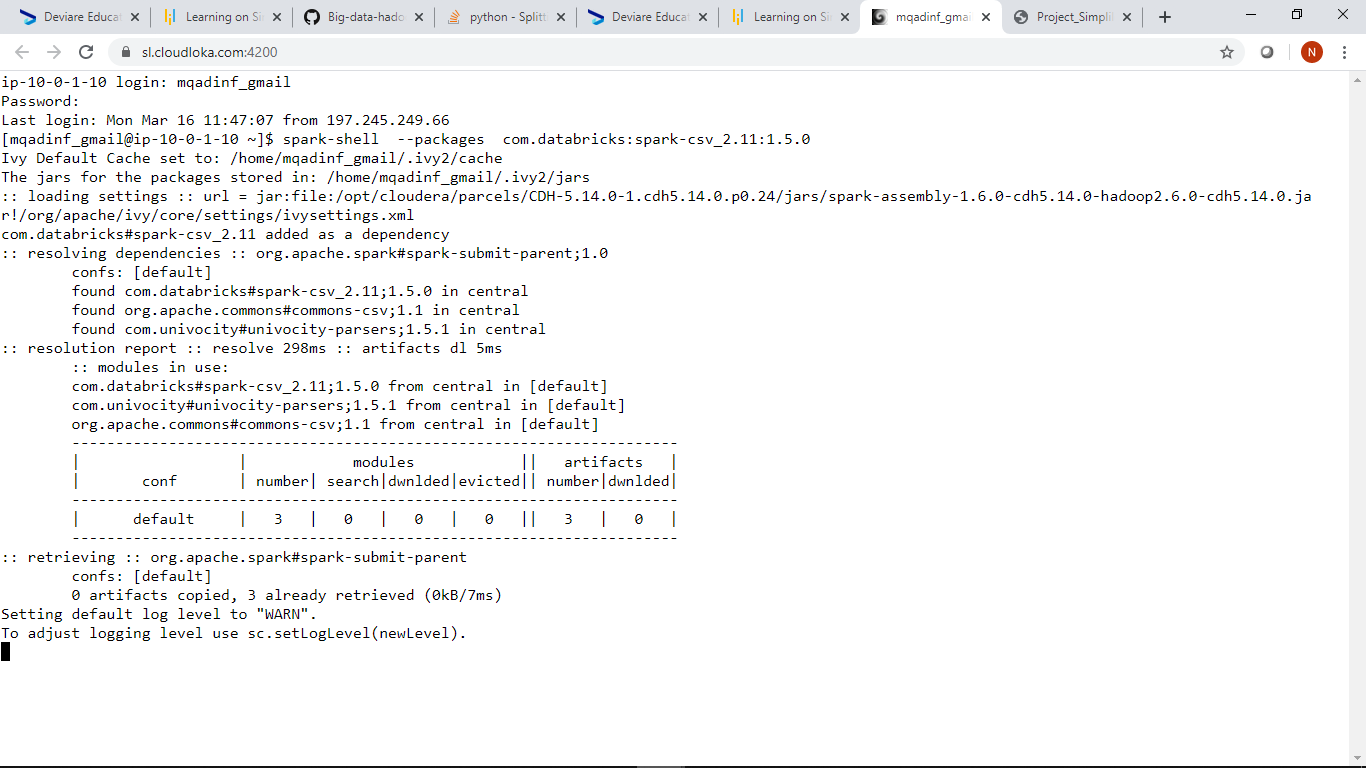
**Analysis tasks to be done-:**

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

1. **Load data and create a Spark data frame**

Before we use spark we use the following command :

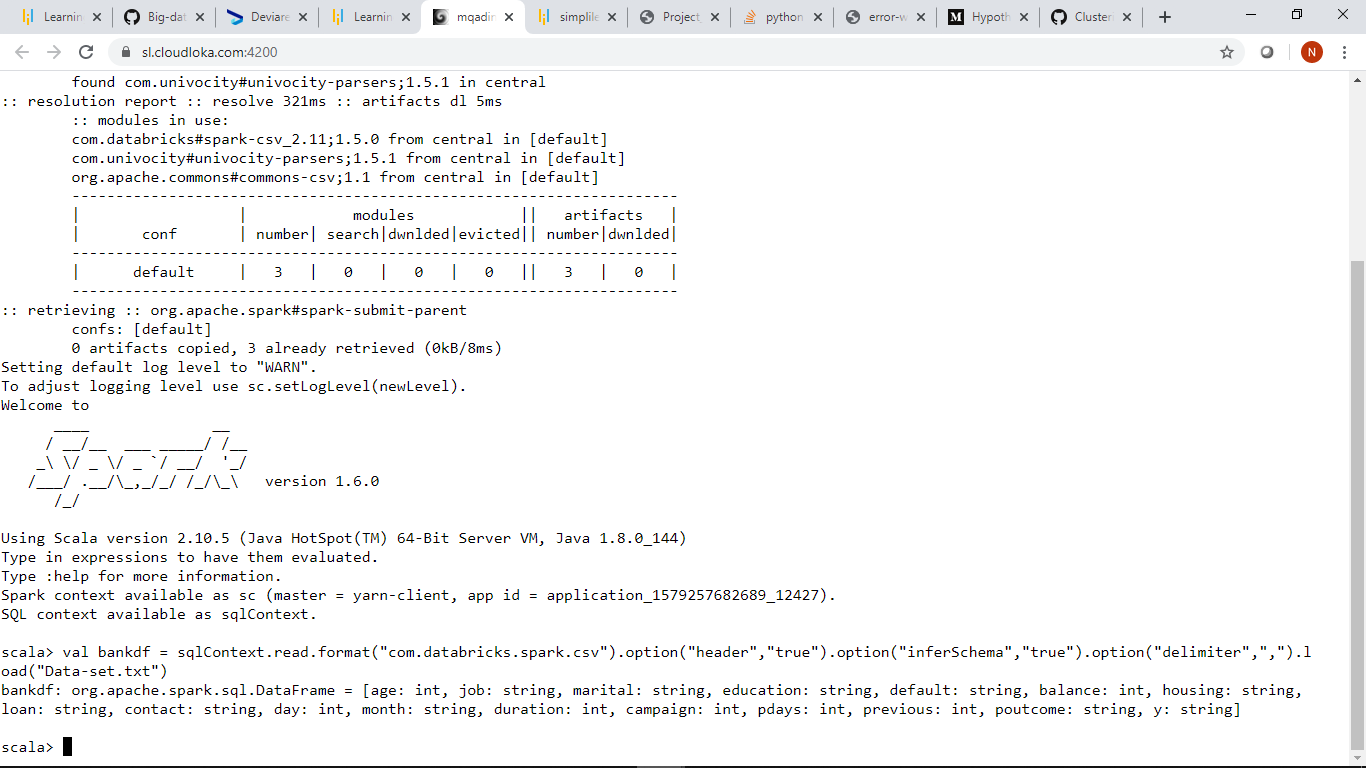
**spark-shell --packages com.databricks:spark-csv\_2.11:1.5.0**

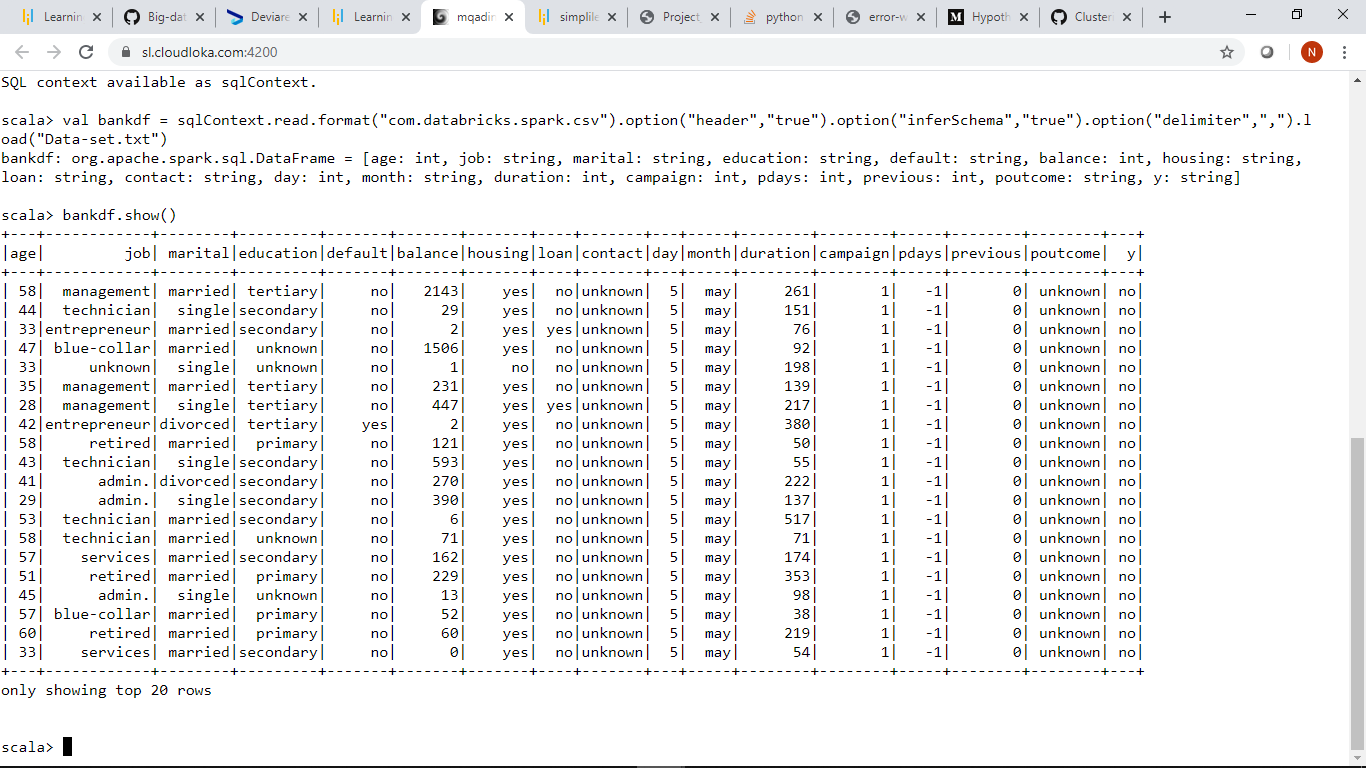


We then import the Dataset and we create a dataframe bankdf

**Code:**

**val df = sqlContext.read.format("com.databricks.spark.csv").option("header","true").option("inferSchema","true").option("delimiter",",").load("Data-set.txt")**





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

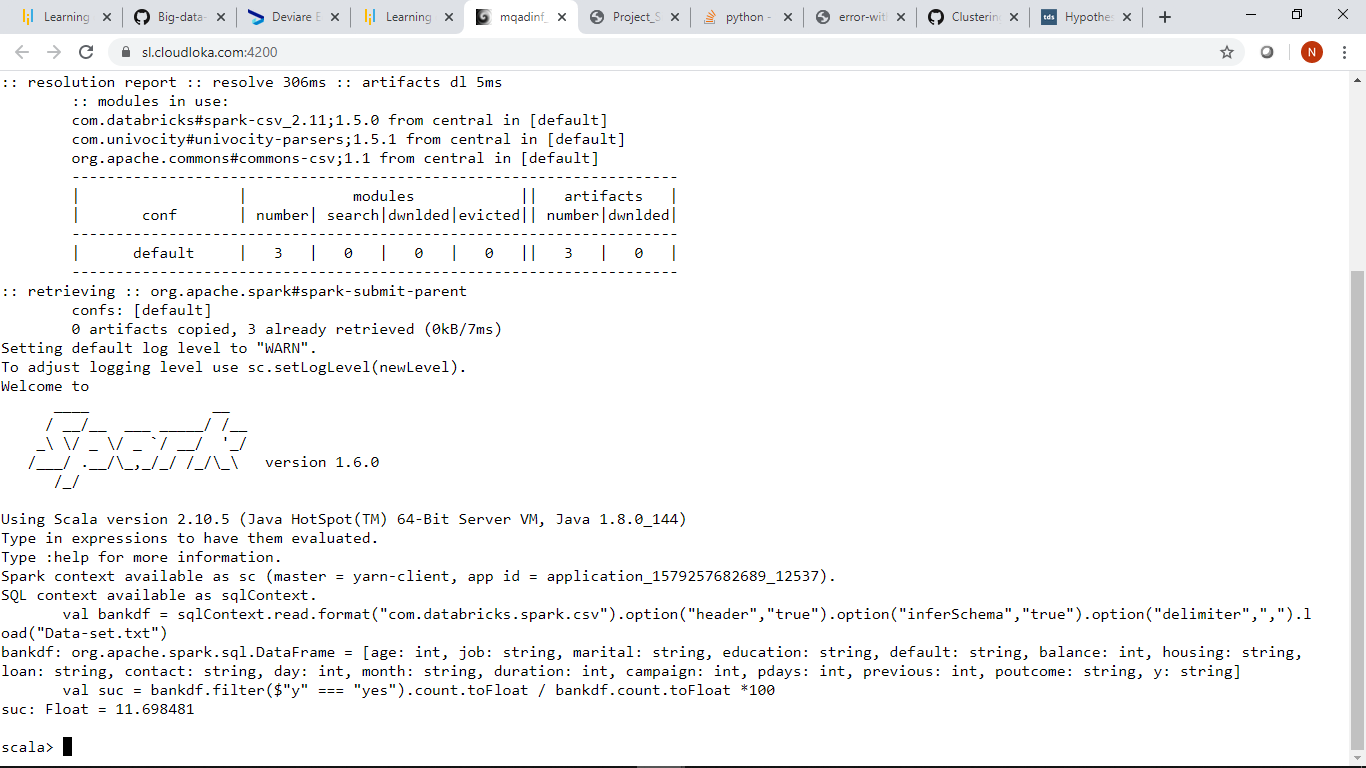
* **Give marketing failure rate**

The marketing success rate is 11.70

**Code:**

**Marketing success rate:**

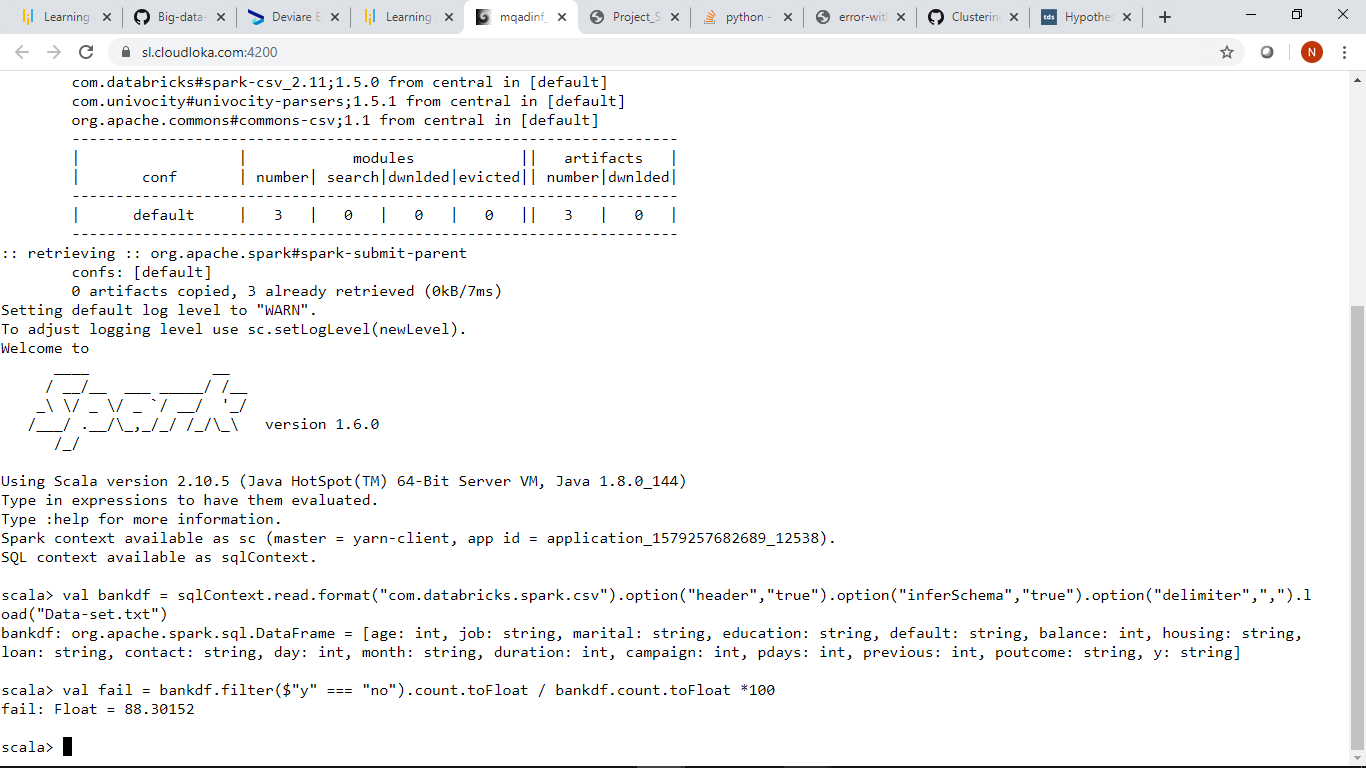
**val suc = bankdf.filter($"y" === "yes").count.toFloat / bankdf.count.toFloat \*100**



**Marketing fail rate**

**The marketing failure rate is 88.31**

**val fail = df.filter($"y" === "no").count.toFloat / df.count.toFloat \*100**



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

Before we can find these aggregate functions the maximum,mean and minimum age of the average targeted customer

We import the following library

**Import org.apache.spark.sql.functions.{min,max,avg}**

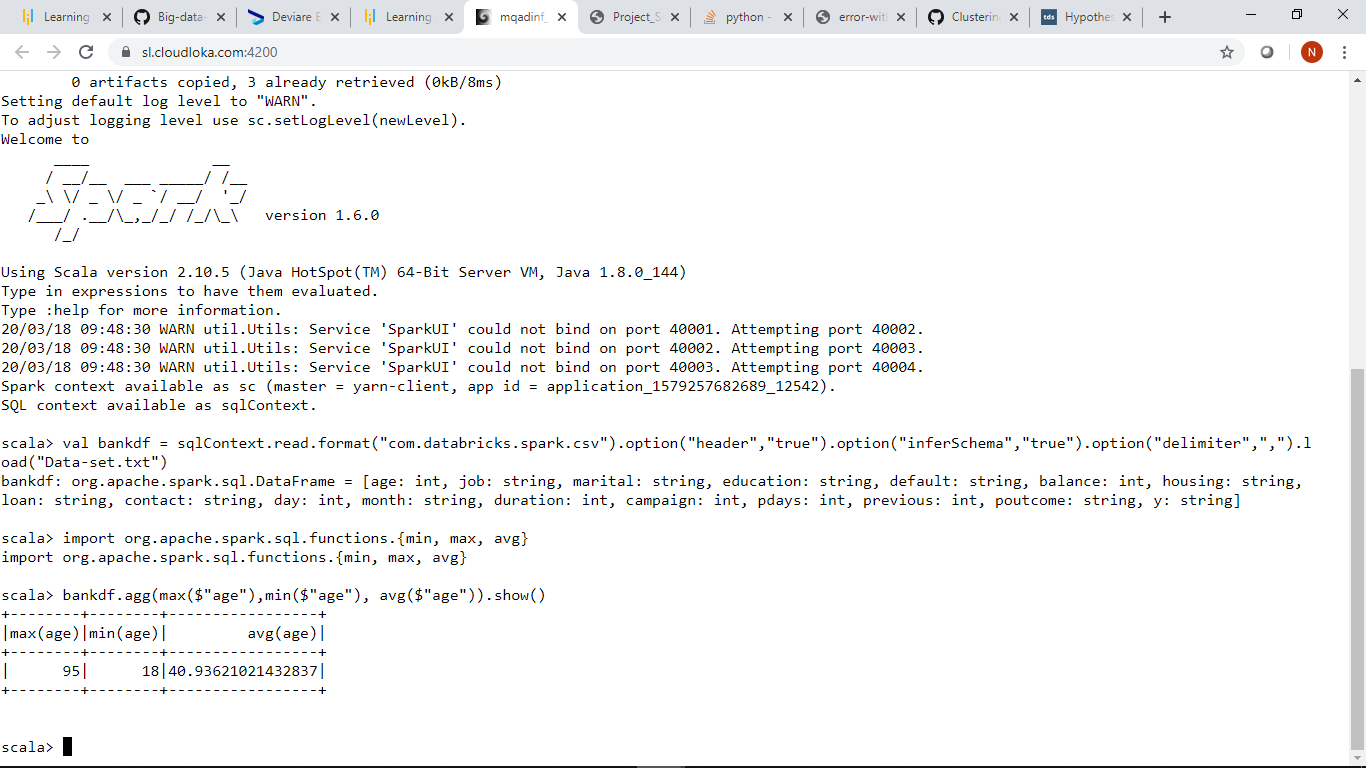
**The maximum age is 95**

**The minimum age is 18**

**The mean or average age is 40.97=41**

**Code for maximum, mean,and maximum age of the average targeted customer:**

**bankdf.agg(max($"age"),min($"age"), avg($"age")).show()**



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

We need to first registerTempTable using the command

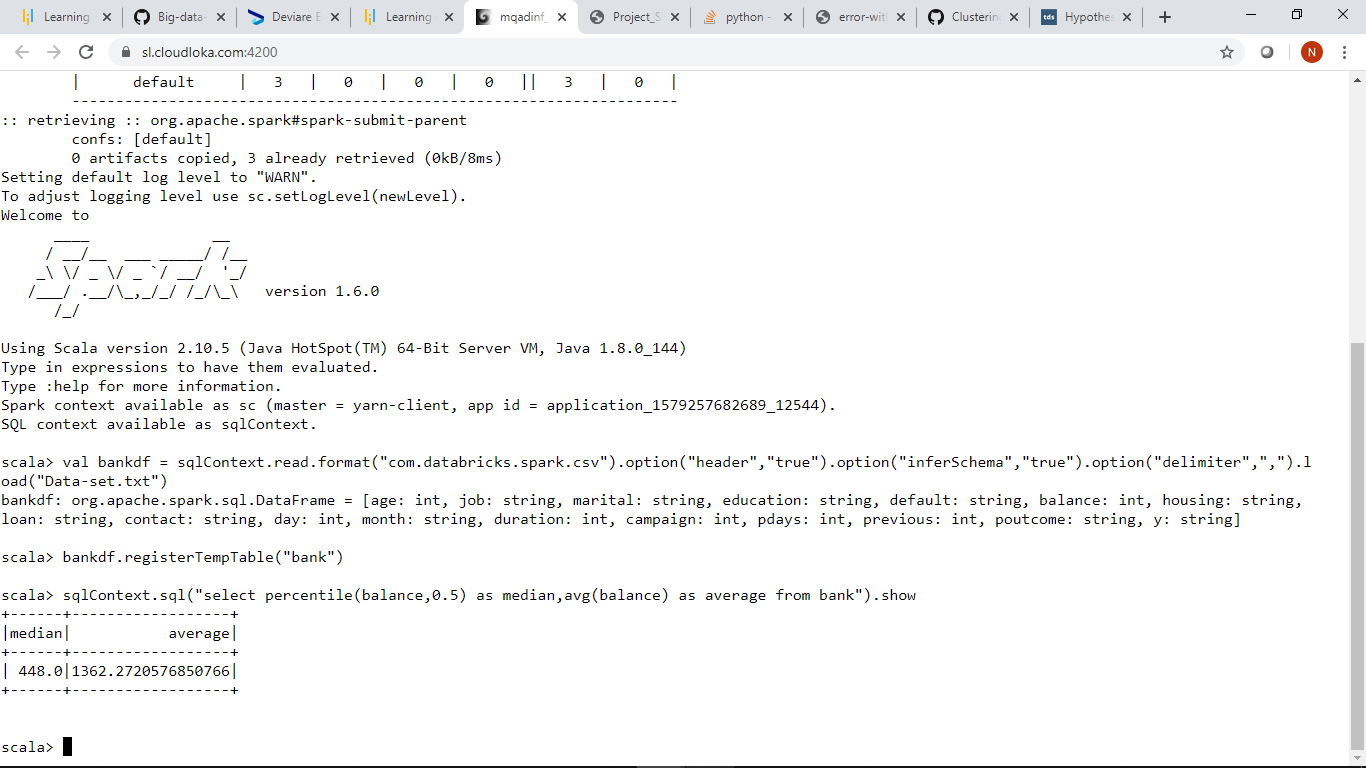
**bankdf.registerTempTable(“bank”)** to perform some sql functions.

The average balance is 1362.27

The median balance is 448

**Code for checking average balance and median balance of customers:**

**sqlContext.sql(“select percentile(balance,0.5) as median,avg(balance) as average from bank”).show**



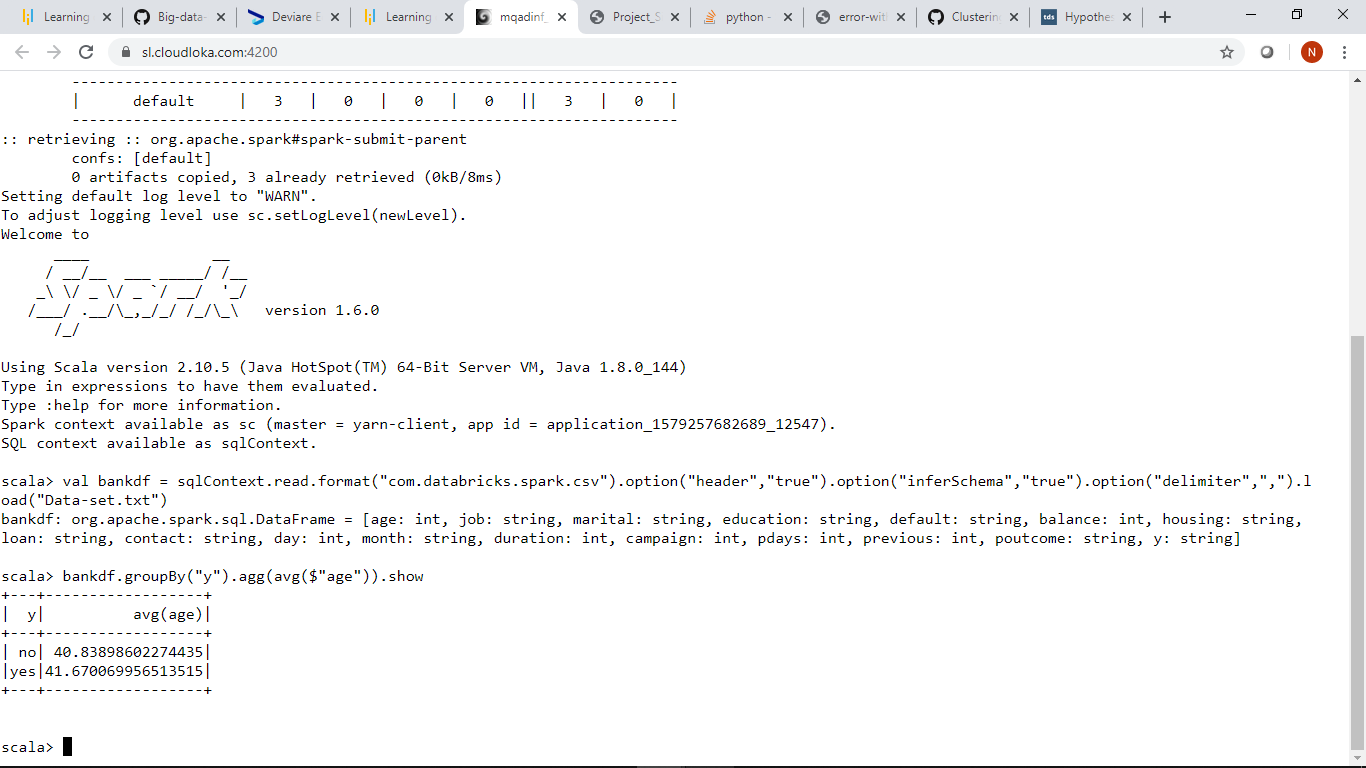
**3.Check if age matters in marketing subscription for deposit**

Below we can see that age really matters we have avg(age) of 41.67 for yes and avg(age) 40.84 for no.

The avg(age) for yes exceeds the avg(age) for no.

**Code:**

**bankdf.groupBy(“y”).agg(avg($”age”)).show**

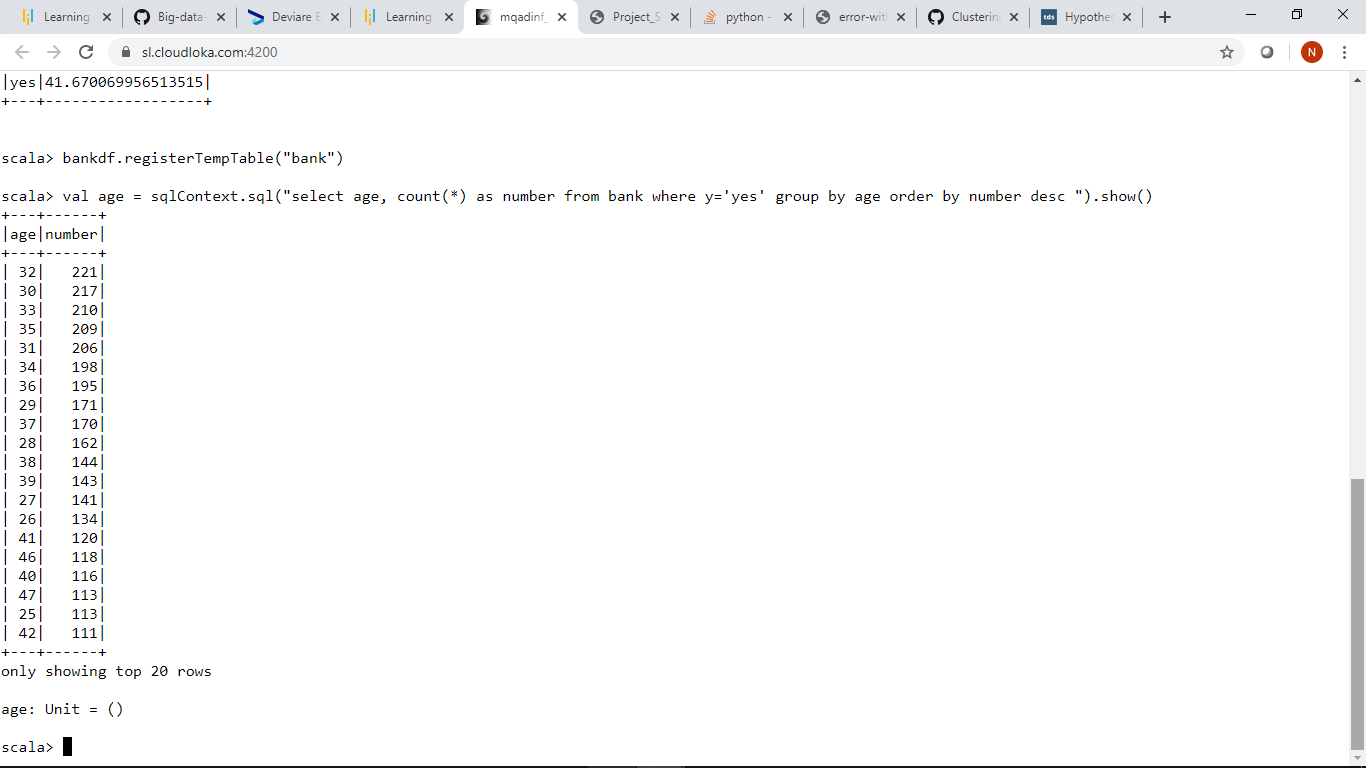


An alternate way

Below we can see that age really matters,the age range between 30-36 shows most promise.

**Code:**

**val age = sqlContext.sql("select age, count(\*) as number from bank where y='yes' group by age order by number desc ").show()**



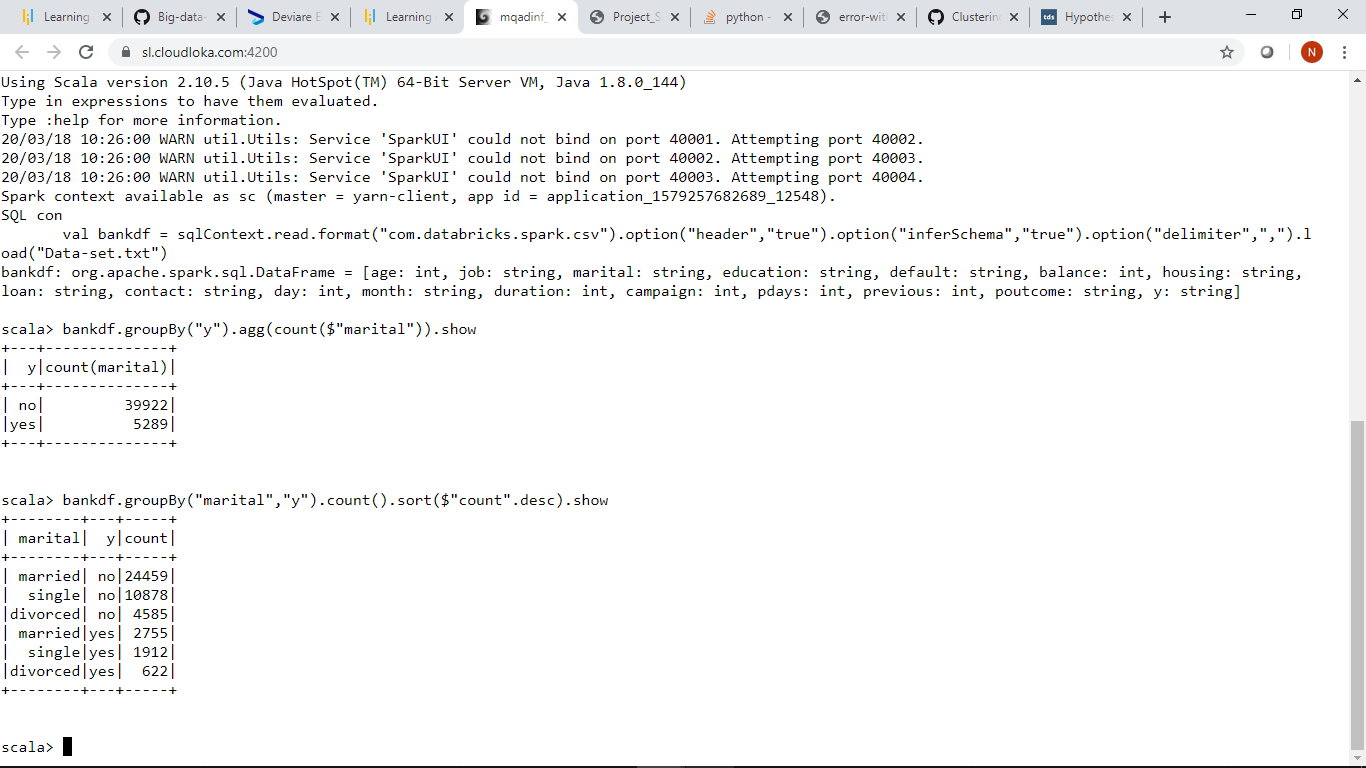
**4.Check if marital status mattered for a subscription to deposit**

We see that it is married couples who go for subscription the most.There are more than 24000 married couples who go for subscription.

**Code:**

**bankdf.groupBy(“y”).agg(count($”marital”)).show**

**bankdf.groupBy(“marital”,”y”).count().sort($”count”.desc).show**

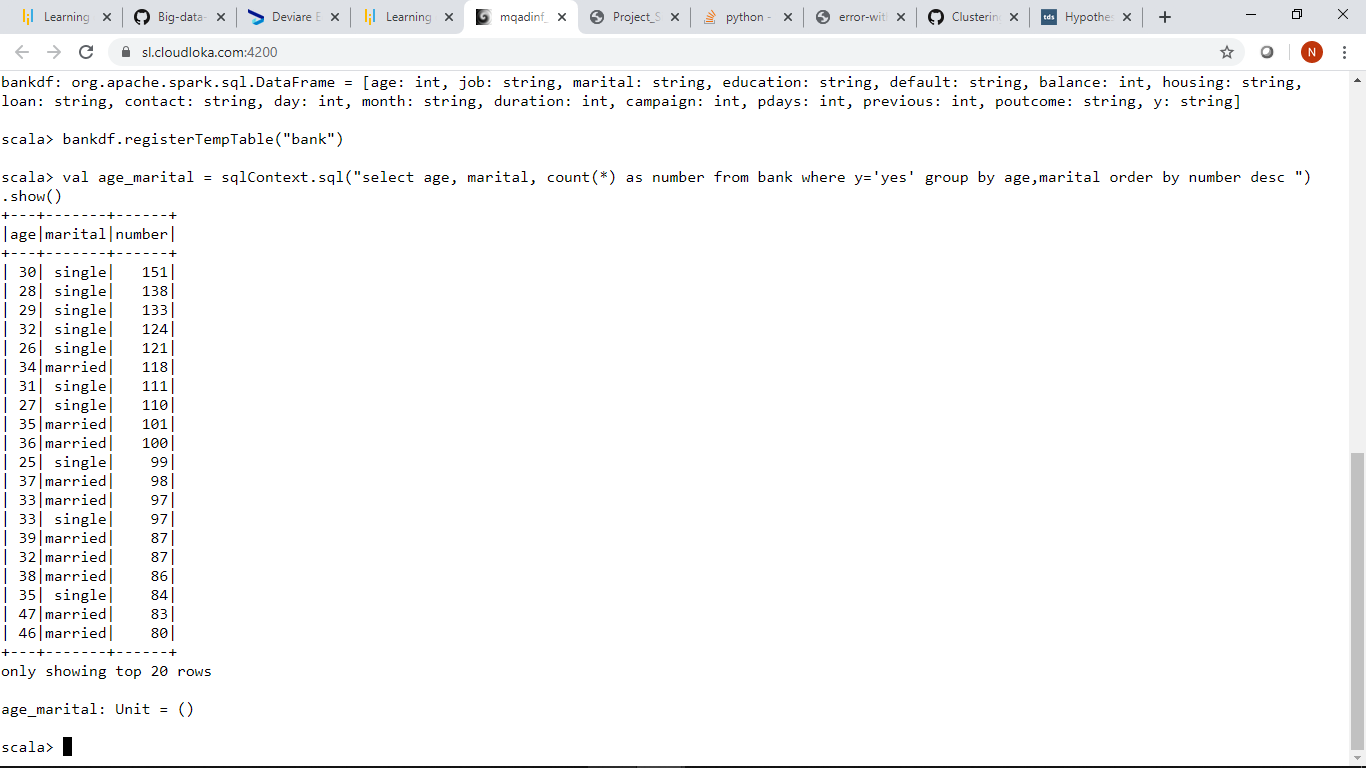


**5.Check if age and marital status together mattered for a subscription to deposit scheme**

Single people around the age of 30-35 show most subscriptions.

**Code:**

**val age\_marital = sqlContext.sql("select age, marital, count(\*) as number from bank where y='yes' group by age,marital order by number desc ").show()**



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

We must first import the following library :

**Import org.apache.spark.sql.functions.udf**

Then we create a method ageToCategory

**Code:**

**Def ageToCategory=udf((age:Int)=> {**

**Age match{**

**case t if t < 30 => “young”**

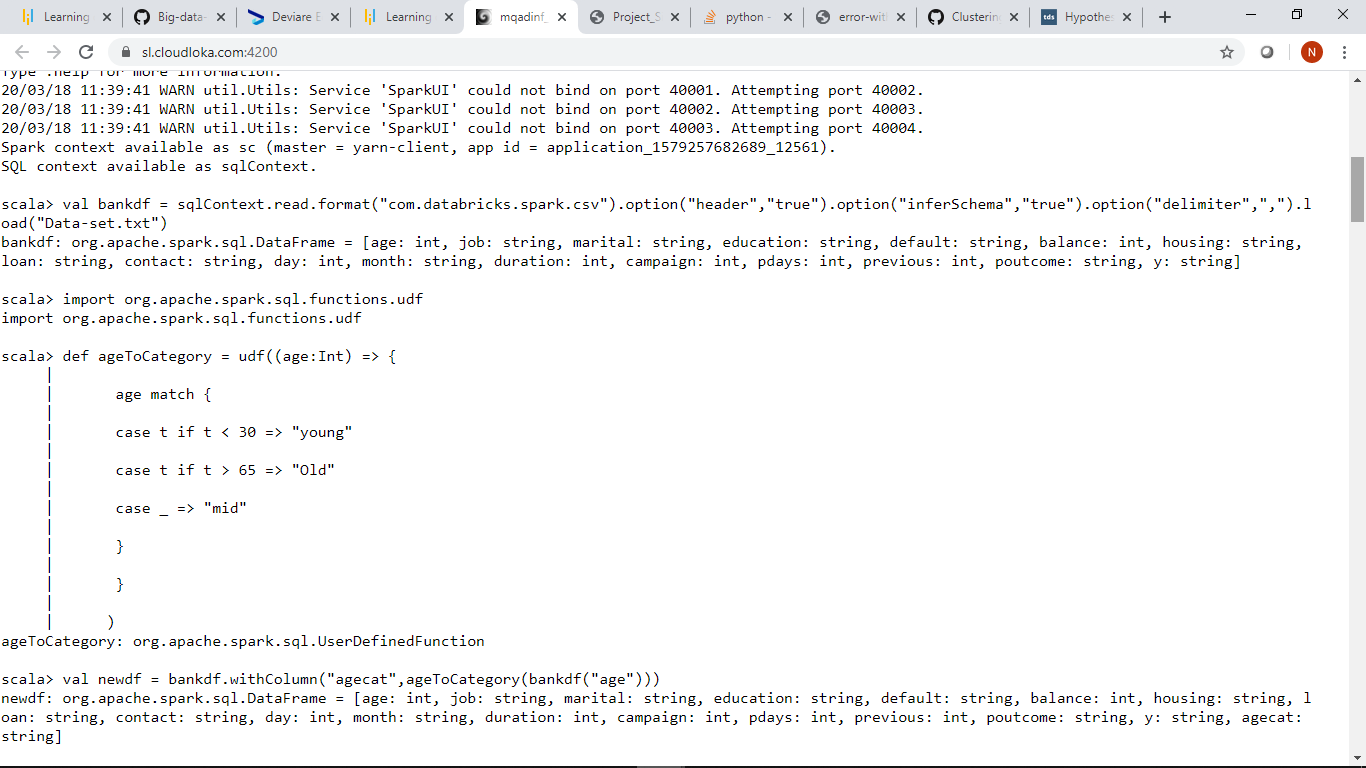
**case t if t >65 => “old”**

**case\_ => “mid”**

**}**

**}**

**)**



Then we create a new data frame newdf and then we create a groupby function to group by agecat and y and sort the count in descending order.

We then create a filter function to filter where y=yes.

Then we create an age category function where we say that when age is less than 25 it is classified as young,when age is greater than 60

It is classified as old or eitherwise as middle. Lastly we create a function result where we group by age category and a count of each of the age category.

Code:

**Val newdf=bankdf.withColumn(“agecat”,ageToCategory(bankdf(“age”)))**

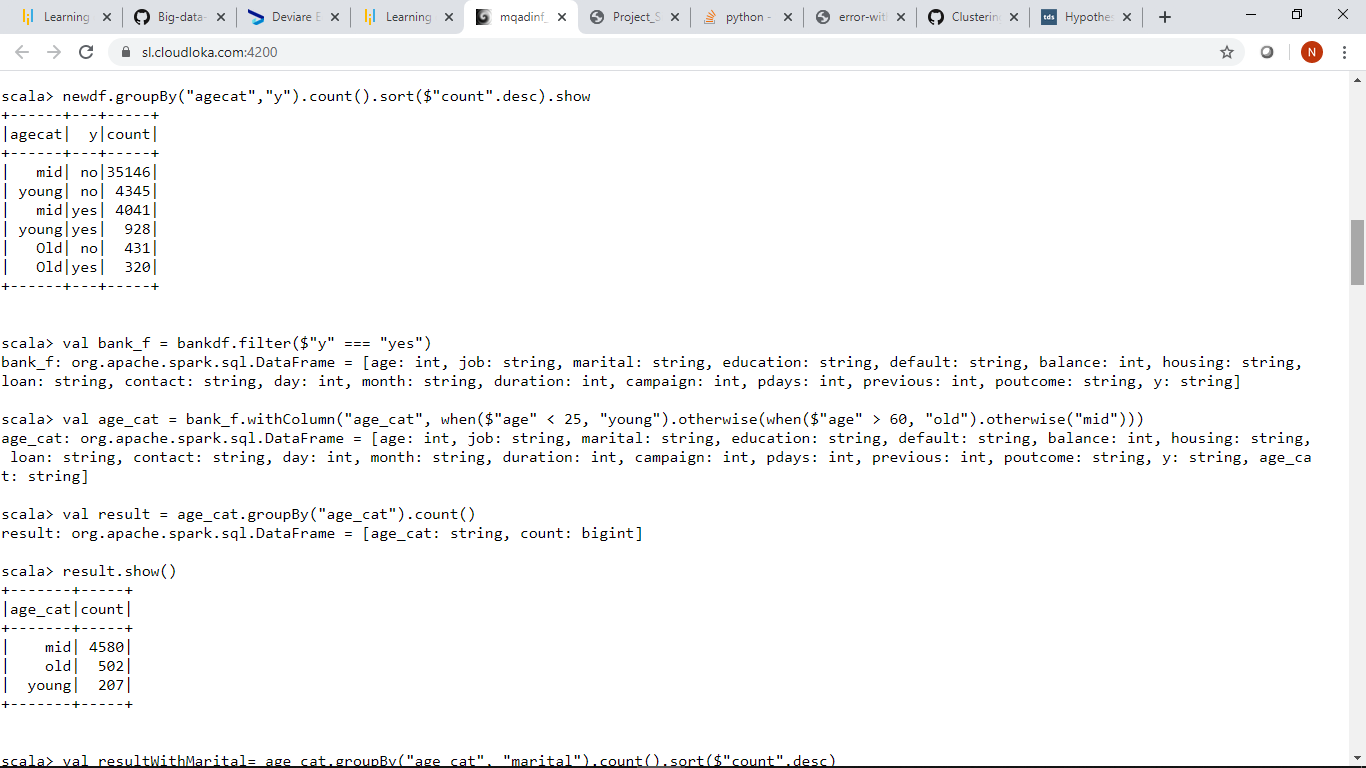
**newdf.groupBy(“agecat”,”y”).count().sort($”count”.desc).show**

**val bank\_f=bankdf.filter($”y”===”yes”)**

**val age\_cat=bank\_f.withColumn(“age\_cat”,when($”age” < 25,”young”).otherwise(when($”age”>60,”old”).otherwise(“mid”)))**

**val result=age\_cat.groupBy(“age\_cat”).count()**

**result.show()**



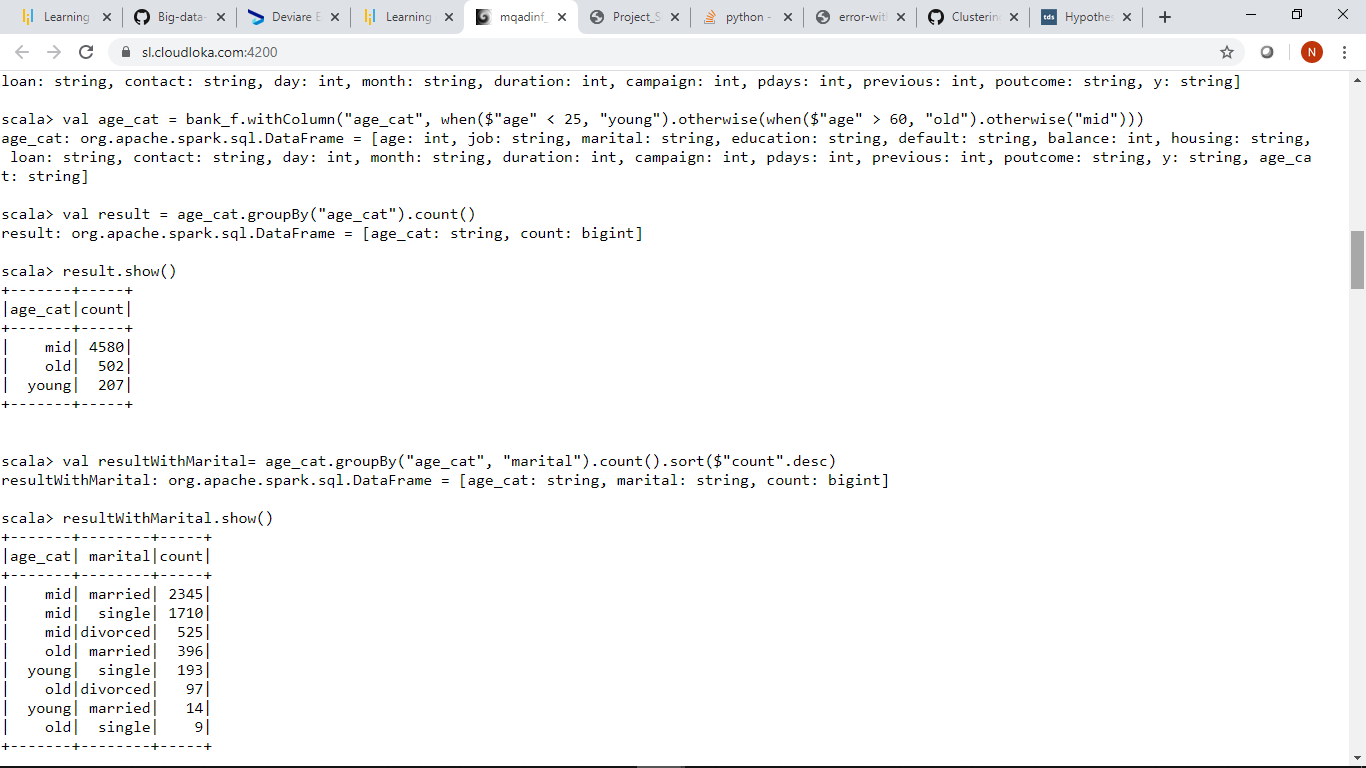
We create another function resultWithMarital where group the age category by marital status and create a count each of the age category and marital status

And then sorting in descending order.

**Code:**

**Val resultWithMarital=age\_cat.groupBy(“age\_cat”,”marital”).count().sort($”count”.desc)**

**resultWithMarital.show()**

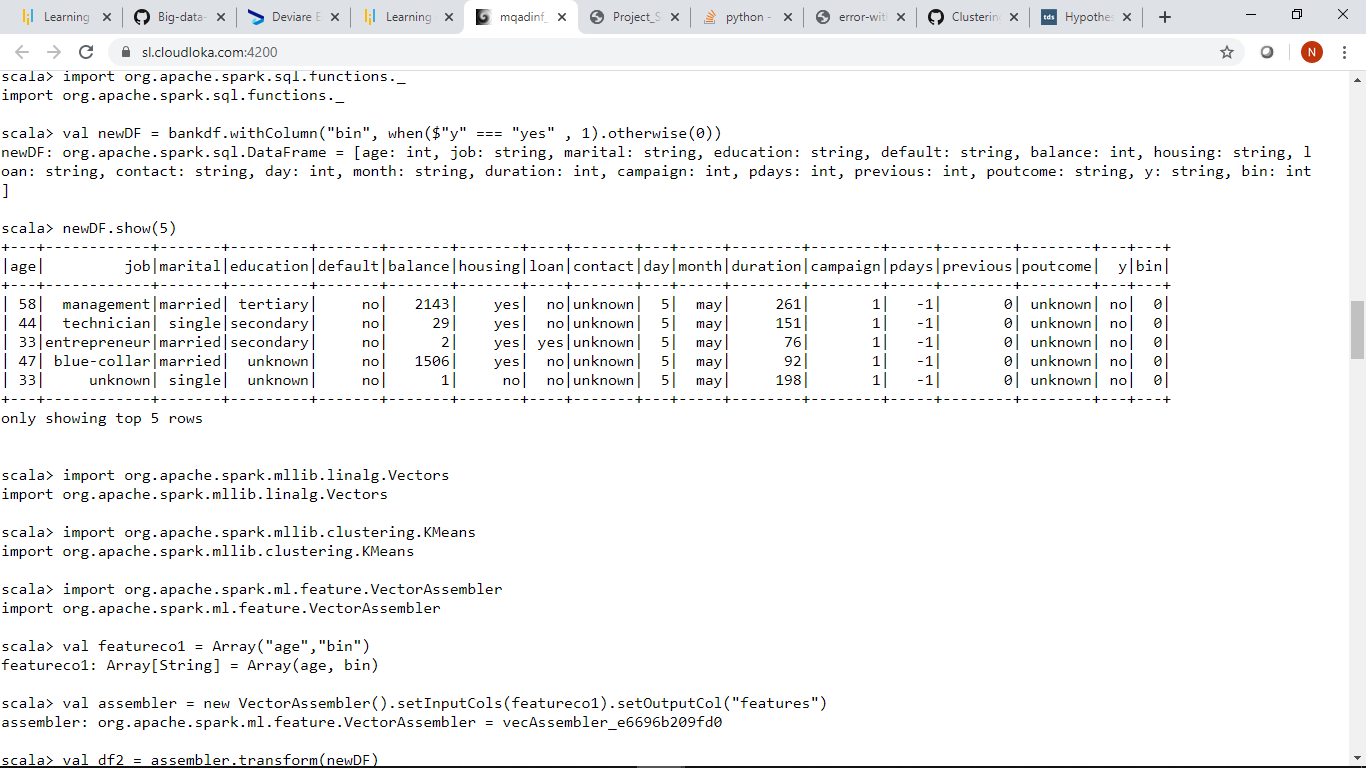


Here we will create a dummy variable where we will convert the variable “y” if the person subscribed to marketing strategy to dummy

**Code:**

**Val newDF=bankdf.withColumn(“bin”,when($”y”===”yes”,1).otherwise(0))**

**newDF.show(5)**



We will do a K-Means Analysis.

We will first import the following libraries:

**Import org.apache.spark.mllib.linag.Vectors**

**Import org.apache.spark.mllib.clustering.KMeans**

**Import org.spark.apache.ml.feature.VectorAssembler**

We will then implement a K-Means

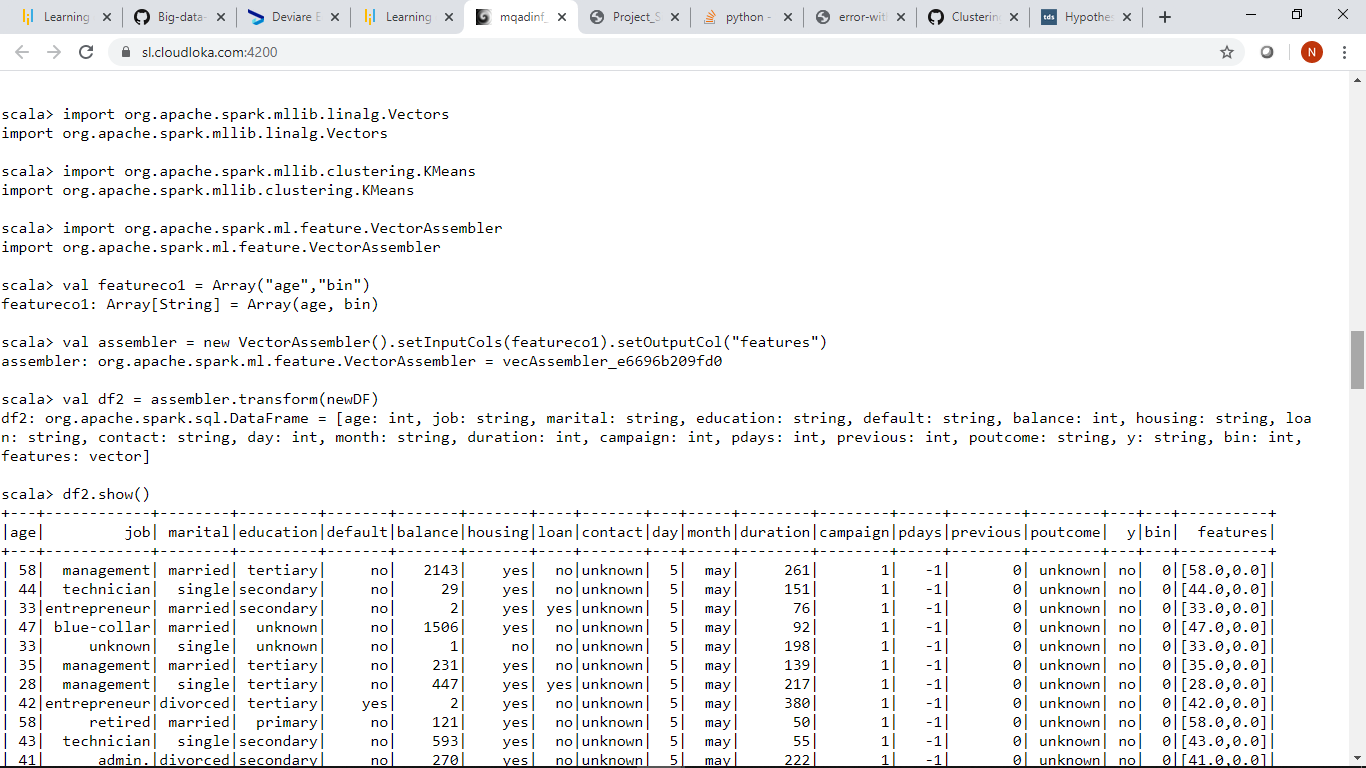
**Code:**

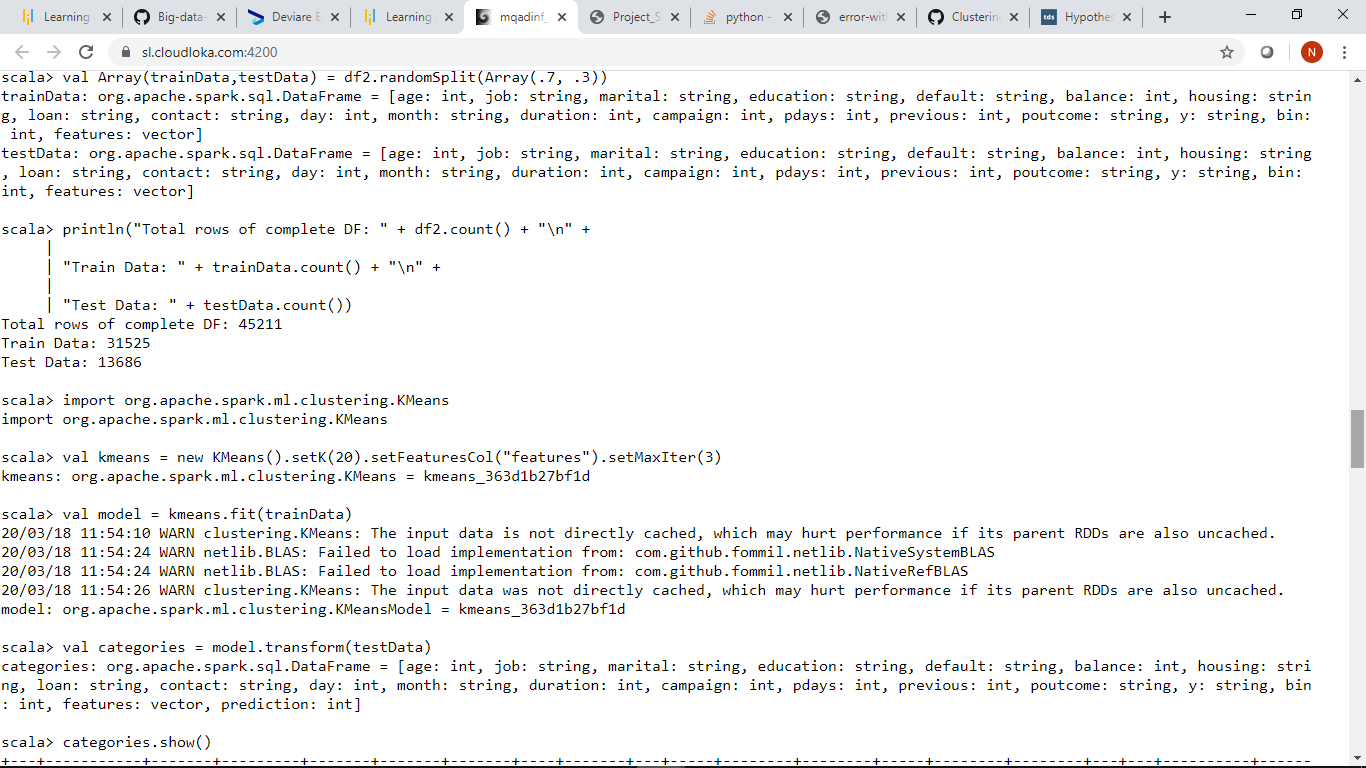
**val featurecol=Array(“age”,”bin”)**

**val assembler=new VectorAssembler().setInputCols(featurecol).setOutput(“features”)**

**val df2=assembler.transform(newDF)**

**df2.show()**





We will then do a Train Test split and do a description of a Train Test split.

**Code:**

**Val Array(trainData,testData)=df2.randomSplit(Array(.7,.3))**

**Println(“Total rows of complete DF:” + df2.count()+ “\n”+**

**“Train Data: “ + trainData.count()+”\n”+**

**“Test Data: “ + testData.count())**

**K-Means Algorithm**

**Code:**

**import org.apache.spark.ml.clustering.KMeans**

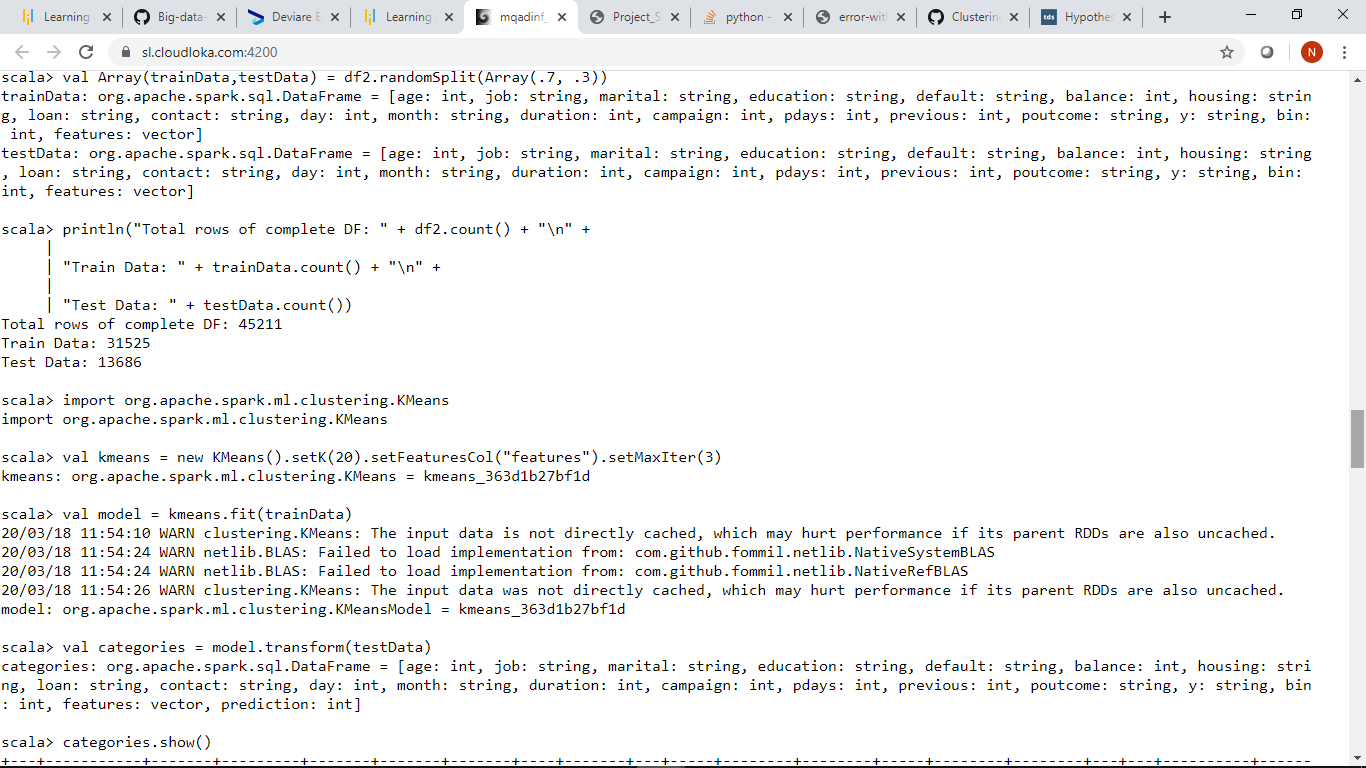
**val kmeans=new KMeans().setK(20).setFeaturesCol(“features”).setMaxIter(3)**

we will fit the KNN model

**val model=kmeans.fit(trainData)**

We then predict the test data.

**val categories=model.transform(testData)**



Now we need to visualize each age per cluster

**Code:**

**Import org.apache.spark.sql.functions.\_**

**categories.groupBy($”age”.alias(“age”)).agg(min$”prediction”).alias(“clusters”)).orderBy(asc(“age”)).show(100)**

