import scala.reflect.runtime.universe

import org.apache.spark.SparkConf

import org.apache.spark.SparkContext

import org.apache.spark.ml.Pipeline

import org.apache.spark.ml.classification.LogisticRegression

import org.apache.spark.ml.feature.Bucketizer

import org.apache.spark.ml.feature.Normalizer

import org.apache.spark.ml.feature.StringIndexer

import org.apache.spark.ml.feature.VectorAssembler

import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics

import org.apache.spark.sql.DataFrame

import org.apache.spark.sql.SQLContext

import org.apache.spark.sql.functions.mean

val bank\_people\_data = spark.read.option("multiline","true").json("/user/aang6160hgmail/bank\_edited.json");

bank\_people\_data.show()

bank\_people\_data.createOrReplaceTempView("datanewtable")

bank\_people\_data.select(max($"age")).show()

bank\_people\_data.select(min($"age")).show()

bank\_people\_data.select(avg($"age")).show()

bank\_people\_data.select(avg($"balance")).show()

val median = spark.sql("SELECT percentile\_approx(balance, 0.5) FROM datanewtable").show()

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

agedata.show()

val maritaldata = spark.sql("select marital, count(\*) as number from datanewtable where y='yes' group by marital order by number desc")

maritaldata.show()

val ageandmaritaldata = spark.sql("select age, marital, count(\*) as number from datanewtable where y='yes' group by age,marital order by number desc")

ageandmaritaldata.show()

val agedata = spark.udf.register("agedata",(age:Int) => {

if (age < 20)

"Teen"

else if (age > 20 && age <= 32)

"Young"

else if (age > 33 && age <= 55)

"Middle Aged"

else

"old"

})

## Replacing the old age column with the new age column

val banknewDF = bank\_people\_data.withColumn("age",agedata(bank\_people\_data("age")))

banknewDF.show()

banknewDF.createOrReplaceTempView("banknewtable")

## Finding out the age group that has the most subscriptions

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

targetage.show()

## String Indexer Pipelining

val agedata2 = new StringIndexer().setInputCol("age").setOutputCol("ageindex")

## Model Fitting

var strindModel = agedata2.fit(banknewDF)

## Assignment of generated value of index of the column by feature engineering

strindModel.transform(banknewDF).select("age","ageIndex").show(5)