TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

NOMBRE DE LOS ALUMNOS:

GALAVIZ LONA OSCAR EDUARDO (N.CONTROL: 17212993)

MARQUEZ MILLAN SEASHELL VANESSA (N.CONTROL:)

Carrera: Ingeniería Informática

Semestre: 9no

MATERIA: Datos Masivos

PROFESOR: JOSE CHRISTIAN ROMERO HERNANDEZ

Practica 1

Unidad 3

Developement

Well he first thing is to import all library we will go to use

```
//import de logisticRegression y otros
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.sql.SparkSession

import org.apache.log4j._
Logger.getLogger("org").setLevel(Level.ERROR)
```

```
scala> import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.ml.classification.LogisticRegression

scala> import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SparkSession

scala>

scala> import org.apache.log4j._
import org.apache.log4j._

scala> Logger.getLogger("org").setLevel(Level.ERROR)
```

After import all, we go to start a sesion in spark

```
//iniciar una sesion de spark y cargar los datos e imprimir estructura
val spark = SparkSession.builder().getOrCreate()

val data  = spark.read.option("header","true").option("inferSchema",
"true").format("csv").load("advertising.csv")

data.printSchema()
```

```
scala> data.printSchema()
root
 |-- Daily Time Spent on Site: double (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Area Income: double (nullable = true)
 |-- Daily Internet Usage: double (nullable = true)
 |-- Ad Topic Line: string (nullable = true)
 |-- City: string (nullable = true)
 |-- Male: integer (nullable = true)
 |-- Country: string (nullable = true)
 |-- Timestamp: timestamp (nullable = true)
 |-- Clicked on Ad: integer (nullable = true)
```

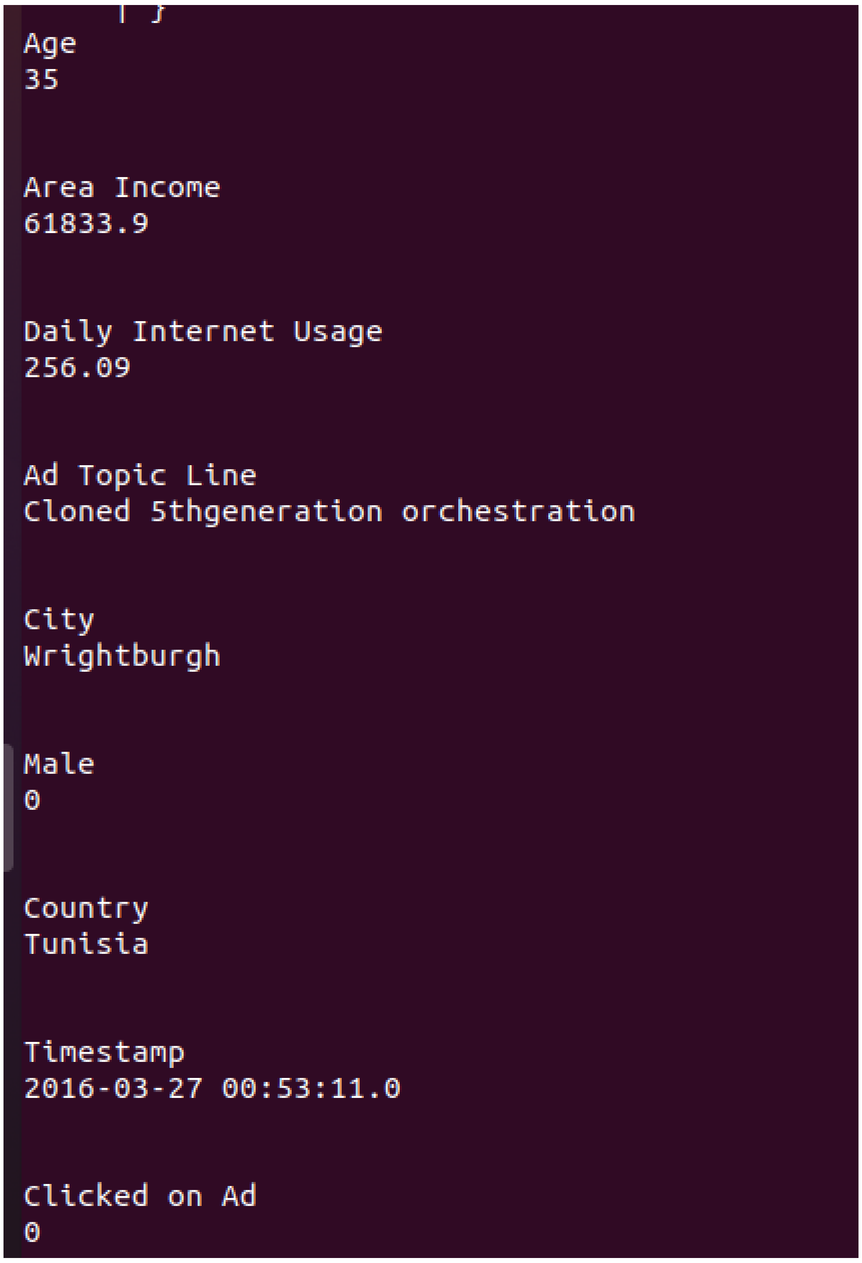
This comand there are for get information about data set

```
data.head(1)

val colnames = data.columns
val firstrow = data.head(1)(0)
```

```
scala> data.head(1)
res2: Array[org.apache.spark.sql.Row] =
Array([68.95,35,61833.9,256.09,Cloned 5thgeneration
orchestration,Wrightburgh,0,Tunisia,2016-03-27 00:53:11.0,0])

scala> val colnames = data.columns
colnames: Array[String] = Array(Daily Time Spent on Site, Age, Area Income,
Daily Internet Usage, Ad Topic Line, City, Male, Country, Timestamp,
Clicked on Ad)

scala> val firstrow = data.head(1)(0)
firstrow: org.apache.spark.sql.Row = [68.95,35,61833.9,256.09,Cloned
5thgeneration orchestration,Wrightburgh,0,Tunisia,2016-03-27 00:53:11.0,0]
```

Here only print a little example

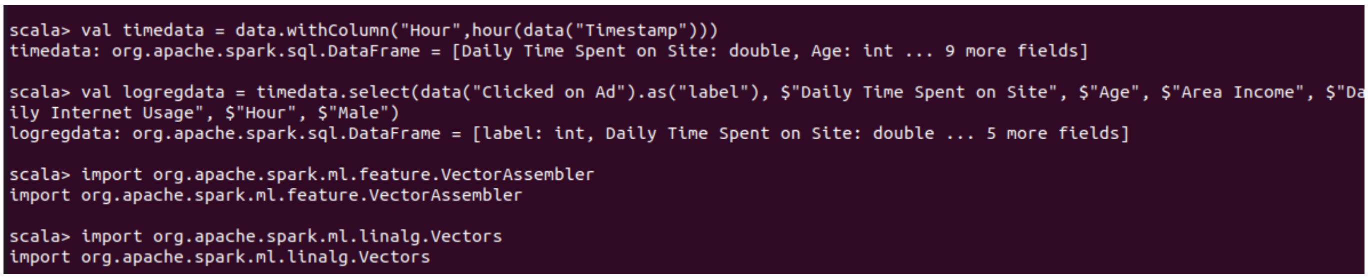
```
// Imprima un renglon de ejemplo

println("\n")
println("Example data row")
for(ind <- Range(1, colnames.length)){
    println(colnames(ind))
    println(firstrow(ind))
    println("\n")
}
```



```
Age
35


Area Income
61833.9


Daily Internet Usage
256.09


Ad Topic Line
Cloned 5thgeneration orchestration


City
Wrightburgh


Male
0


Country
Tunisia


Timestamp
2016-03-27 00:53:11.0


Clicked on Ad
0
```

The next thing is to prepare the data, like normaly we prepare the data

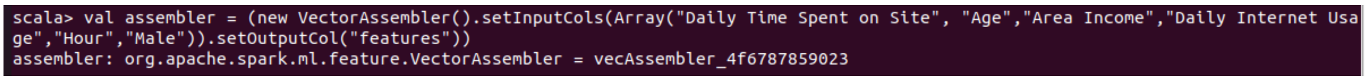
```
//// Preparar el DataFrame para Machine Learning ////
//creacion de una nueva columna llamada Timestamp
val timedata = data.withColumn("Hour",hour(data("Timestamp")))

//renombrar "clicked on Ad" a "label" y tomar las columnas como features
val logregdata = timedata.select(data("Clicked on Ad").as("label"), $"Daily
Time Spent on Site", $"Age", $"Area Income", $"Daily Internet Usage",
$"Hour", $"Male")
// Importe VectorAssembler y Vectors

// Cree un nuevo objecto VectorAssembler llamado assembler para los feature
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.linalg.Vectors
```



```
scala> val timedata = data.withColumn("Hour",hour(data("Timestamp")))
timedata: org.apache.spark.sql.DataFrame = [Daily Time Spent on Site: double, Age: int ... 9 more fields]

scala> val logregdata = timedata.select(data("Clicked on Ad").as("label"), $"Daily Time Spent on Site", $"Age", $"Area Income", $"Da
ily Internet Usage", $"Hour", $"Male")
logregdata: org.apache.spark.sql.DataFrame = [label: int, Daily Time Spent on Site: double ... 5 more fields]

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

scala> import org.apache.spark.ml.linalg.Vectors
import org.apache.spark.ml.linalg.Vectors
```

We genere the vector assembler

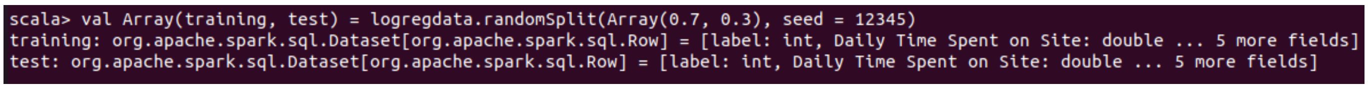
```
val assembler = (new VectorAssembler().setInputCols(Array("Daily Time Spent
on Site", "Age","Area Income","Daily Internet
Usage","Hour","Male")).setOutputCol("features"))
```



```
scala> val assembler = (new VectorAssembler().setInputCols(Array("Daily Time Spent on Site", "Age","Area Income","Daily Internet Usa
ge","Hour","Male")).setOutputCol("features"))
assembler: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_4f6787859023
```

And here split the data, we use training and test

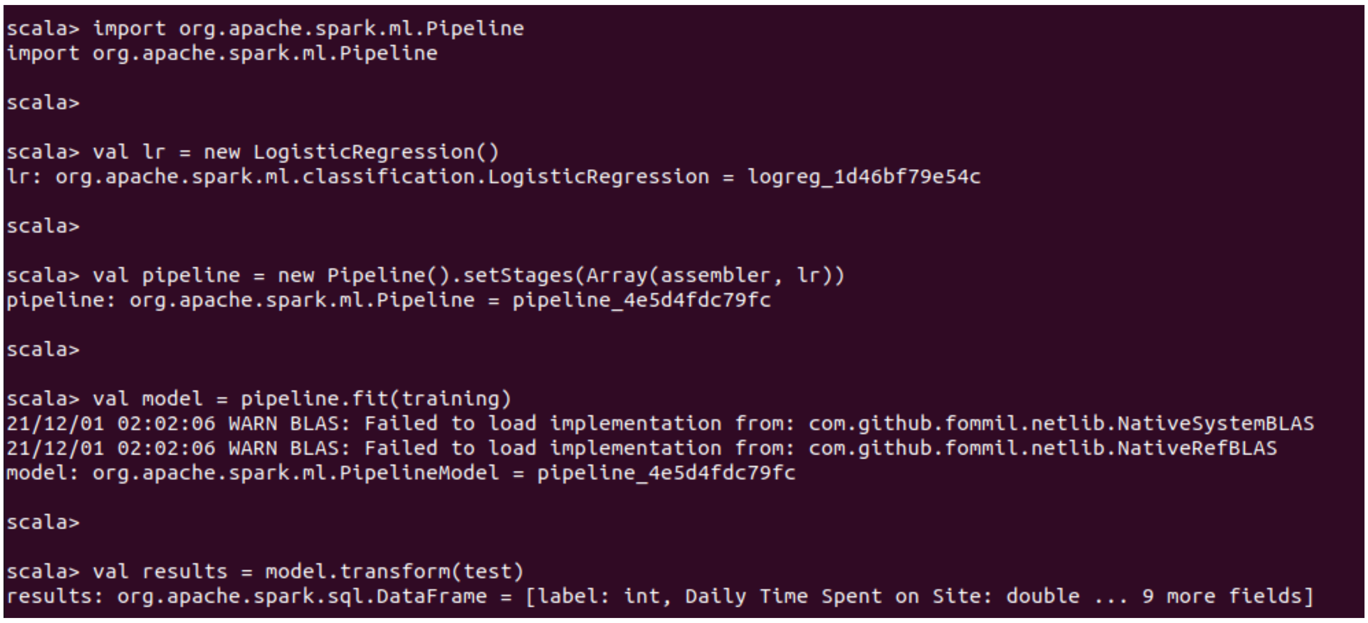
```
// Utilice randomSplit para crear datos de train y test divididos en 70/30
val Array(training, test) = logregdata.randomSplit(Array(0.7, 0.3), seed =
12345)
```



```
scala> val Array(training, test) = logregdata.randomSplit(Array(0.7, 0.3), seed = 12345)
training: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [label: int, Daily Time Spent on Site: double ... 5 more fields]
test: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [label: int, Daily Time Spent on Site: double ... 5 more fields]
```

Now we need to do the pipiline configuration, and start with the model

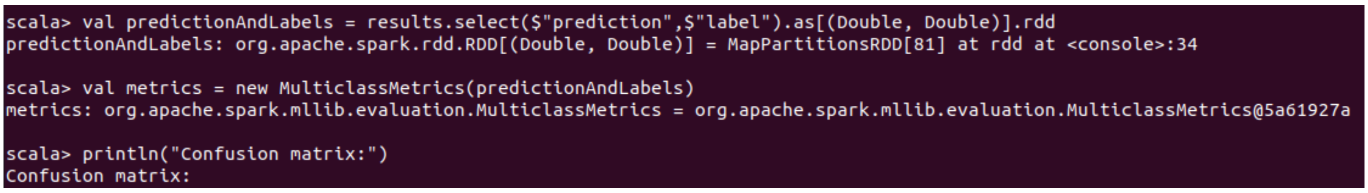
```


import org.apache.spark.ml.Pipeline

val lr = new LogisticRegression()

val pipeline = new Pipeline().setStages(Array(assembler, lr))

val model = pipeline.fit(training)

val results = model.transform(test)
```



```
scala> import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.Pipeline

scala>

scala> val lr = new LogisticRegression()
lr: org.apache.spark.ml.classification.LogisticRegression = logreg_1d46bf79e54c

scala>

scala> val pipeline = new Pipeline().setStages(Array(assembler, lr))
pipeline: org.apache.spark.ml.Pipeline = pipeline_4e5d4fdc79fc

scala>

scala> val model = pipeline.fit(training)
21/12/01 02:02:06 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
21/12/01 02:02:06 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
model: org.apache.spark.ml.PipelineModel = pipeline_4e5d4fdc79fc

scala>

scala> val results = model.transform(test)
results: org.apache.spark.sql.DataFrame = [label: int, Daily Time Spent on Site: double ... 9 more fields]
```

Some times we need more librarys like this case, here to do the prediction

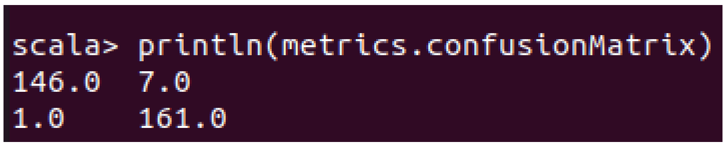
```
//// Evaluacion del modelo /////////////

// Para Metrics y Evaluation importe MulticlassMetrics

import org.apache.spark.mllib.evaluation.MulticlassMetrics

val predictionAndLabels =
results.select($"prediction",$"label").as[(Double, Double)].rdd
val metrics = new MulticlassMetrics(predictionAndLabels)
```



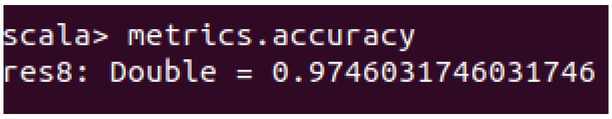
```
scala> val predictionAndLabels = results.select($"prediction",$"label").as[(Double, Double)].rdd
predictionAndLabels: org.apache.spark.rdd.RDD[(Double, Double)] = MapPartitionsRDD[81] at rdd at <console>:34

scala> val metrics = new MulticlassMetrics(predictionAndLabels)
metrics: org.apache.spark.mllib.evaluation.MulticlassMetrics = org.apache.spark.mllib.evaluation.MulticlassMetrics@5a61927a

scala> println("Confusion matrix:")
Confusion matrix:
```

And the last thing is to do the confusion matrix, is so important because to do the data more real results

```
println("Confusion matrix:")
println(metrics.confusionMatrix)

metrics.accuracy
```



```
scala> println(metrics.confusionMatrix)
146.0  7.0
1.0    161.0
```



```
scala> metrics.accuracy
res8: Double = 0.9746031746031746
```