**Évaluation formative**

# PARTIE 1 – Activité Spark SQL

## Lecture du fichier AAPL.CSV

### Importer les librairies :

import org.apache.spark.\_;

import org.apache.spark.rdd.RDD;

import org.apache.spark.util.IntParam;

import org.apache.spark.sql.SQLContext;

import org.apache.spark.sql.functions.\_;

import org.apache.spark.sql.types.\_;

import org.apache.spark.sql.\_;

import org.apache.spark.mllib.stat.Statistics;

import org.apache.spark.sql.SparkSession;

val spark = SparkSession.builder().appName("Spark SQL basic example").config("spark.some.config.option", "some-value").getOrCreate()

import sqlContext.implicits.\_

import spark.\_

import sqlContext.\_

### Créer des classes :

case class Stock(dt: String, openprice: Double, highprice: Double, lowprice: Double, closeprice:Double, volume: Double, adjcloseprice: Double)

### Définir des fonctions:

def parseStock(str: String): Stock = {

val line = str.split(",")

Stock(line(0), line(1).toDouble, line(2).toDouble, line(3).toDouble, line(4).toDouble,line(5).toDouble,line(6).toDouble)

}

def parseRDD(rdd: RDD[String]): RDD[Stock] = {

val header = rdd.first

rdd.filter(\_(0) != header(0)).map(parseStock).cache()

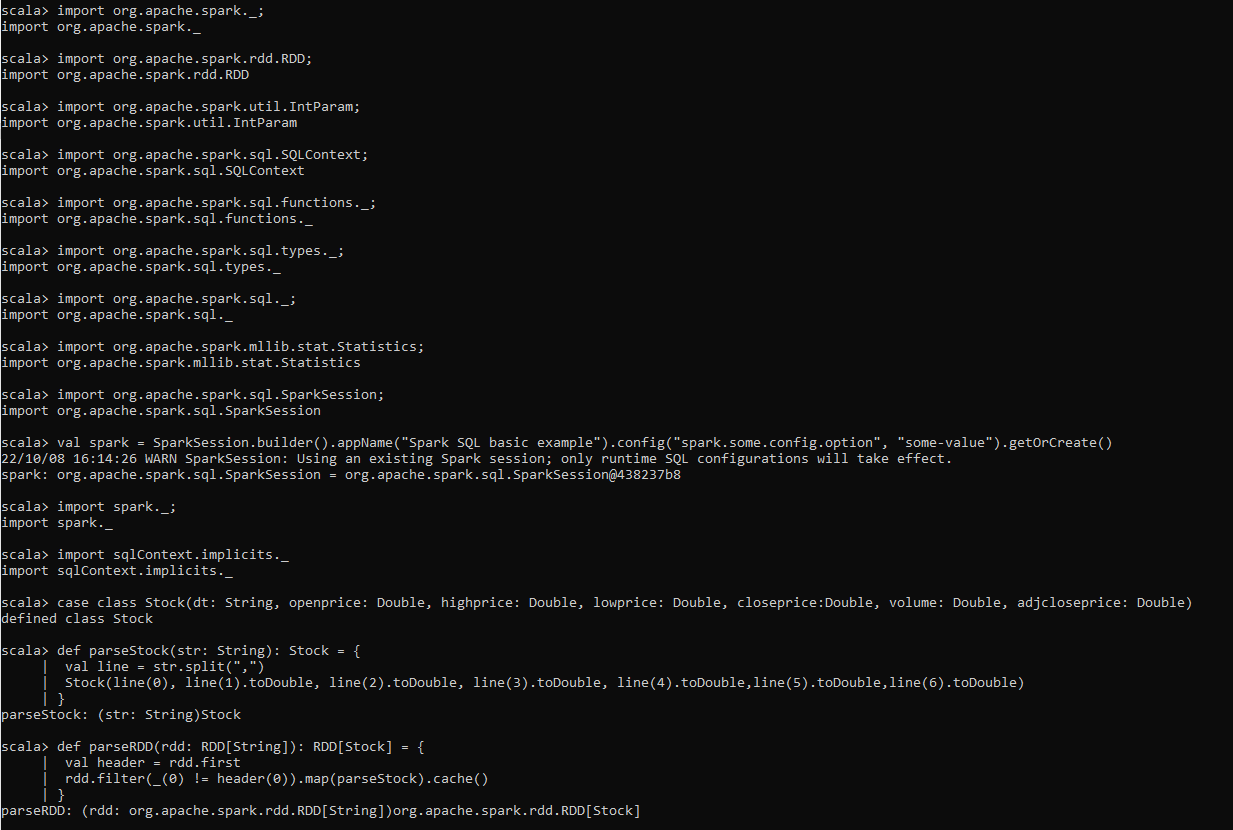
}

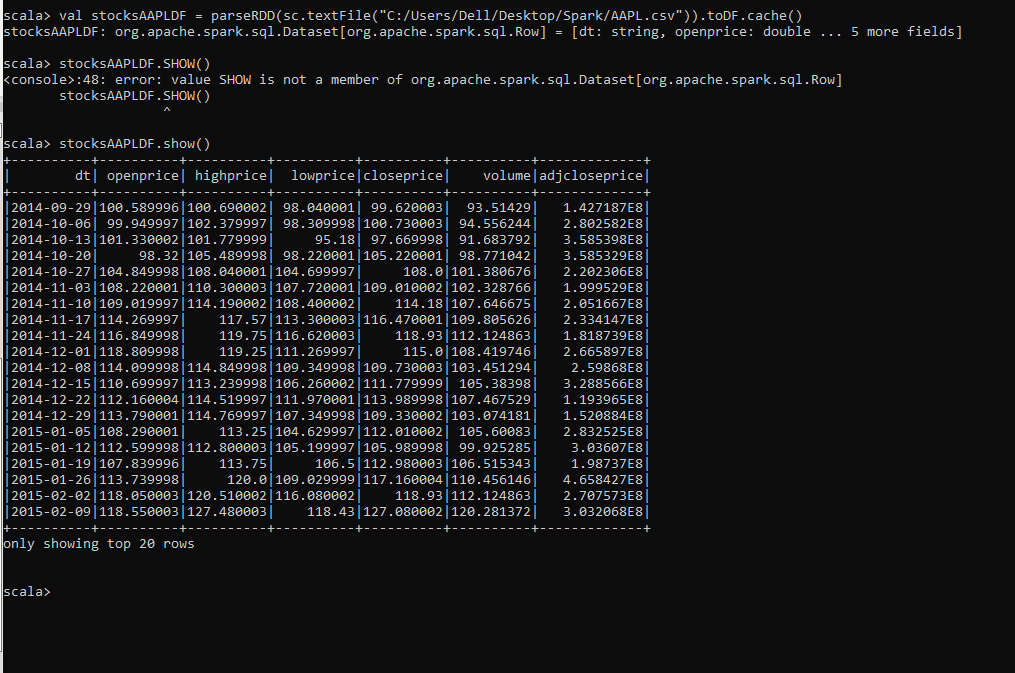
### Télécharger le fichier:

val stocksAAPLDF = parseRDD(sc.textFile("C:/Users/Dell/Desktop/Spark/AApl.csv")).toDF.cache()

stocksAAPLDF.show()

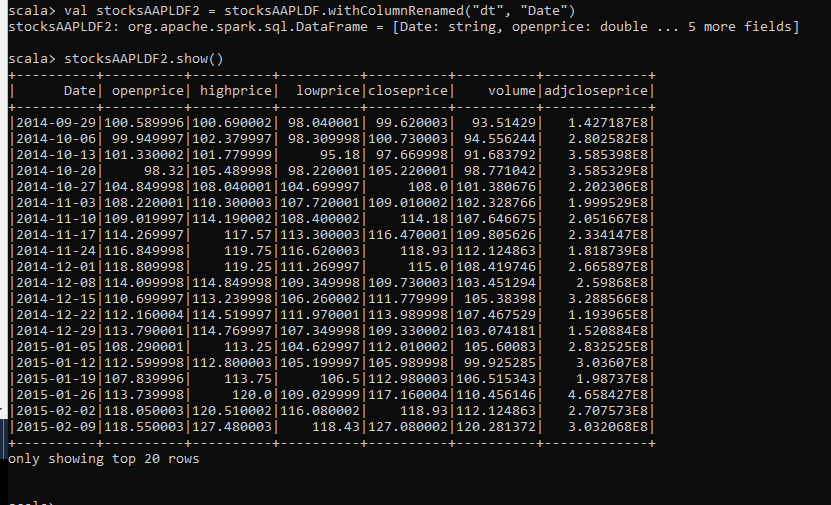
### Résultat du code :





### Renommer la colonne « dt » en « Date »

val stocksAAPLDF2= stocksAAPLDF.withColumnRenamed("dt","Date")

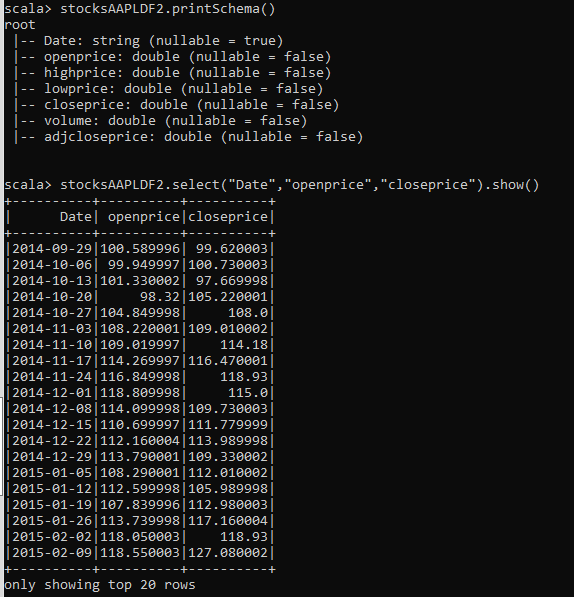


## Liste des dates de transaction, ouverture et fermeture de la valeur de l’action

## 1. Code :

1. stocksAAPLDF2.show()
2. stocksAAPLDF2.printSchema()
3. stocksAAPLDF2.select("Date","openprice","closeprice").show()

## Résultat



## 3. Liste des dates de transaction avec la différence entre fermeture et ouverture

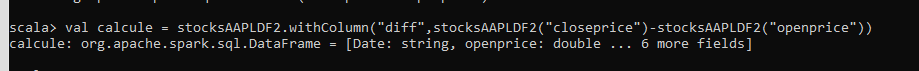
### création de la RDD QUI CALCULE LA DIFFERENCE

val diff = stocksAAPLDF2("closeprice")-stocksAAPLDF2("openprice")



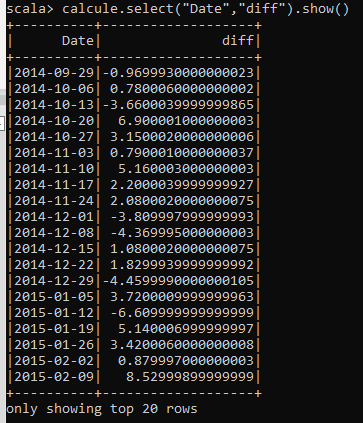
### utilisation de la fonction WithColumn

val calcule = stocksAAPLDF2.withColumn("diff",stocksAAPLDF2("closeprice")-stocksAAPLDF2("openprice"))



### utilisation de SQL select

1. calcule.select("Date","diff").show()

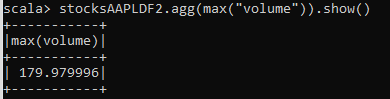


## 4. Max et Min des volumes

### Maximum des volumes

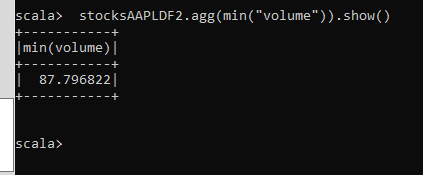
stocksAAPLDF2.groupBy("Date").max("volume").show()

stocksAAPLDF2.agg(max("volume")).show()



### Minimum des volumes

stocksAAPLDF2.agg(min("volume")).show()



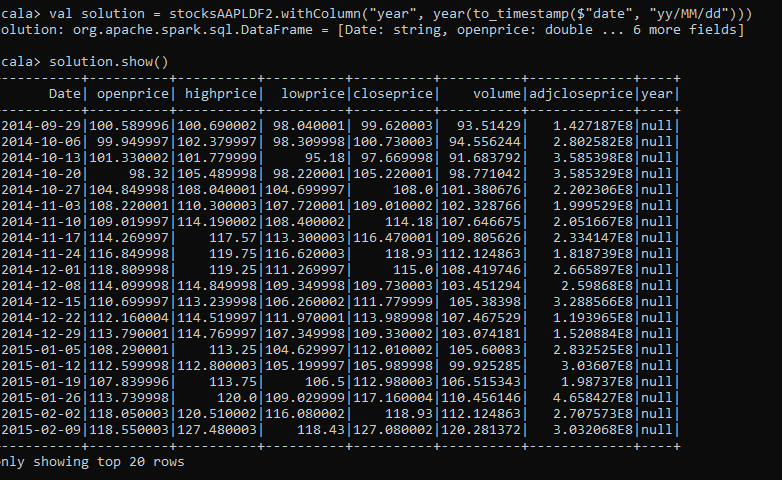
## 5. Moyenne des volumes d’ouverture par années

### Extraire l’année de la colonne date

val stocksAAPLDF3 = stocksAAPLDF2.withColumn("Date",col("Date")cast(DateType))

val solution = stocksAAPLDF3.withColumn("year", year(to\_timestamp($"Date", "yyyy/MM/dd")))

solution.show()

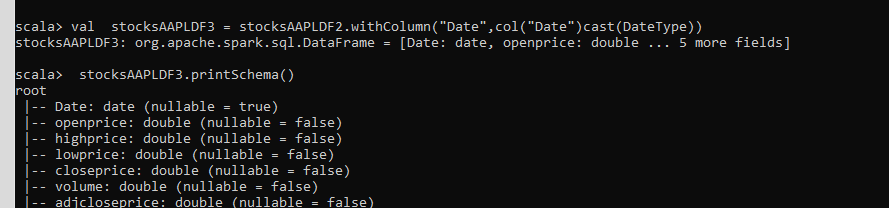


### Moyenne des volumes d’ouverture par années

Il faut d’abord extraire l’année de la colonne date, et pour ce faire il faut changer le type de la colonne Date en type Date

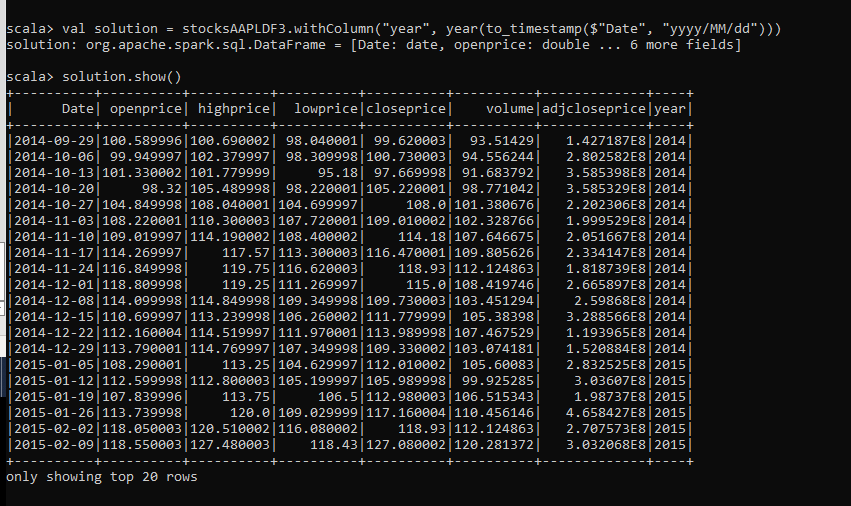
val stocksAAPLDF3 = stocksAAPLDF2.withColumn("Date",col("Date")cast(DateType))

stocksAAPLDF3.printSchema()



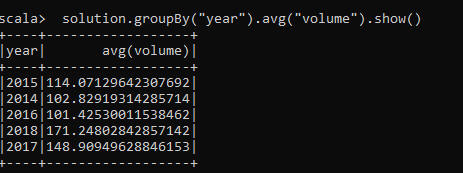
### Ensuite extraire l’année comme suit

1. val solution = stocksAAPLDF3.withColumn("year", year(to\_timestamp($"Date", "yyyy/MM/dd")))
2. solution.show()



## Puis calculer la moyenne des volumes par année

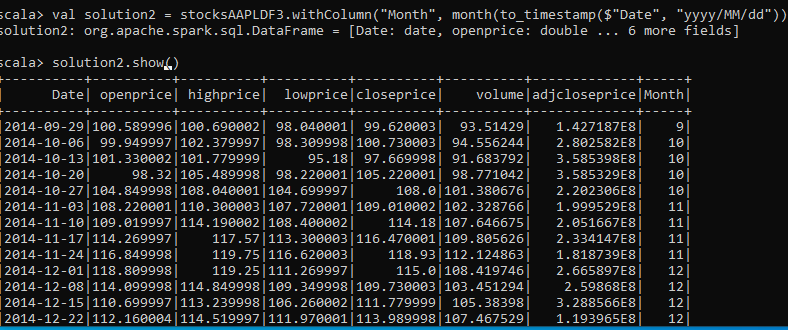
solution.groupBy("year").avg("volume").show()



## 6. Somme des volumes par mois

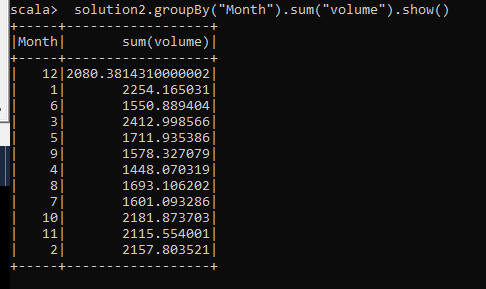
### Ensuite extraire le mois de la colonne Date comme suit

1. val solution2 = stocksAAPLDF3.withColumn("Month", month(to\_timestamp($"Date", "yyyy/MM/dd
2. solution.show()



### Calculer la somme par mois

1. solution2.groupBy("Month").sum("volume").show()



# PARTIE 2 – Refaire le traitement que vous faites habituellement avec MapReduce en utilisant Spark

On avait calculé la moyenne d’age selon la categorie de Revenu

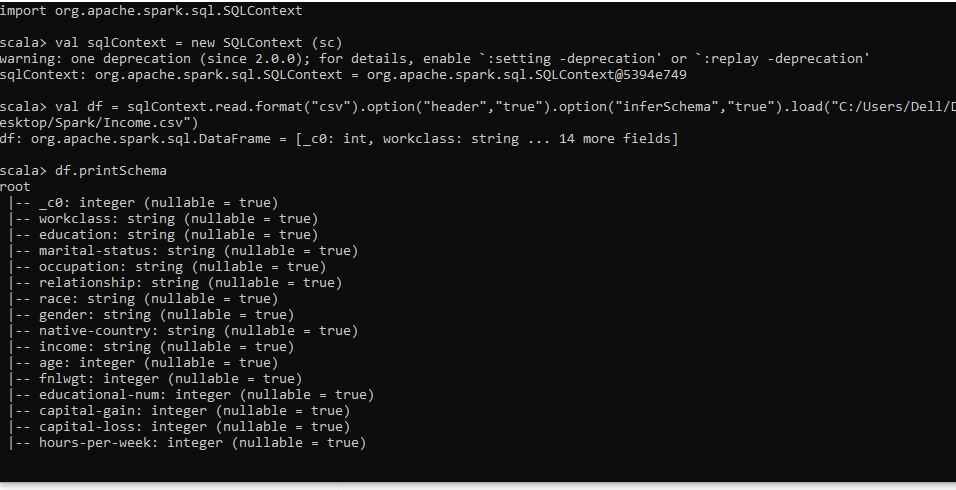
## 1. Lecture du fichier Income.csv

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

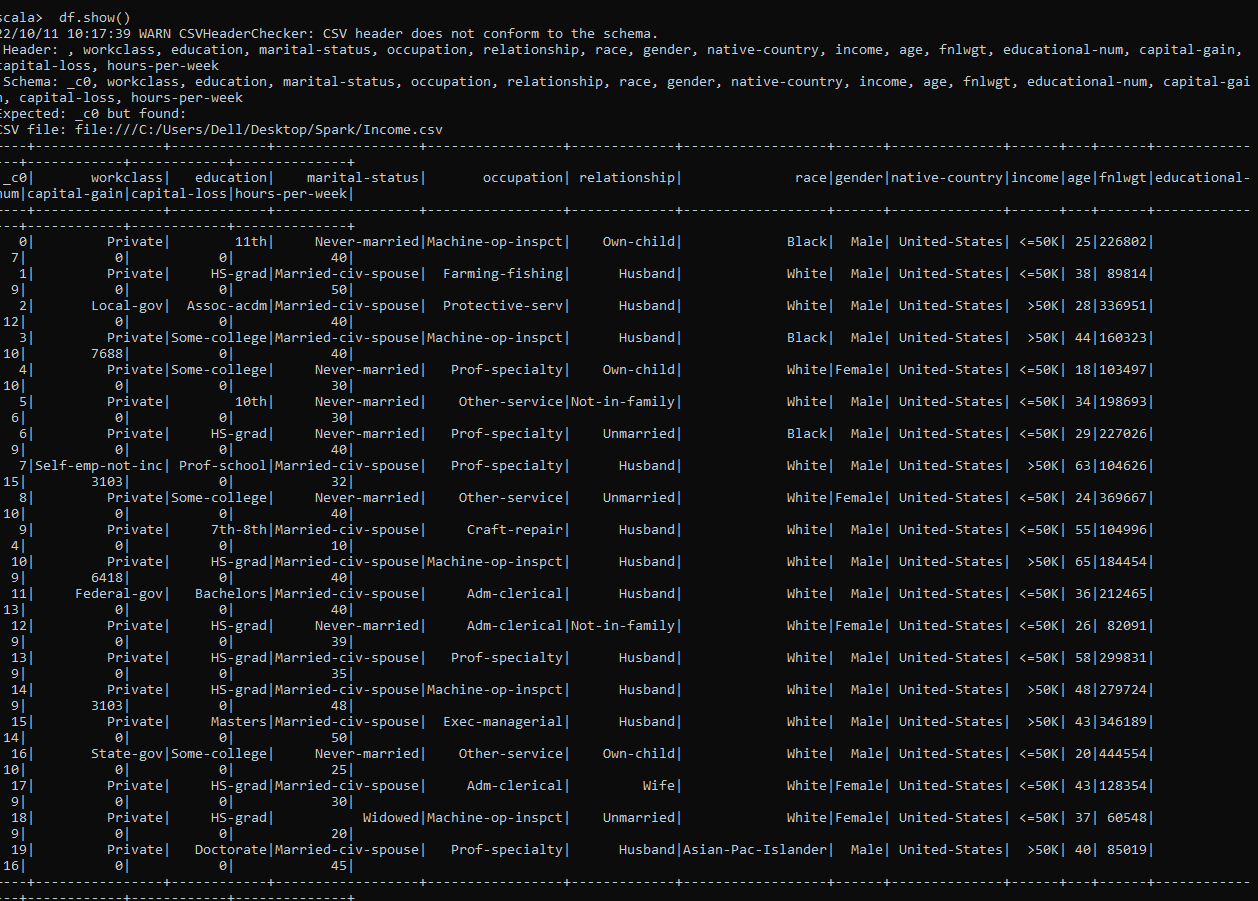
scala> val sqlContext = new SQLContext (sc)

scala> val df = sqlContext.read.format("csv").option("header","true").option("inferSchema","true").load("C:/Users/dell/Desktop/SPARK/Income.csv")

scala> df.printSchema

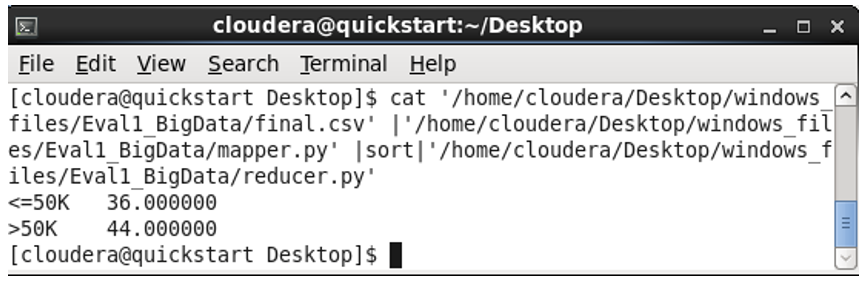


scala> df.show()



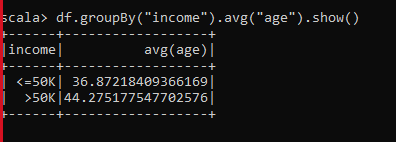
## 2.Calculer la moyenne d’Age selon les revenus avec Spark

### Les résultats obtenus en utilisant le code MapReduce de Cloudera (dans l’évaluation 1)



### En utilisant Spark Scala

1. df.groupBy("income").avg("age").show()



### ioAutre façon avec Spark :

import org.apache.spark.\_;

import org.apache.spark.rdd.RDD;

import org.apache.spark.util.IntParam;

import org.apache.spark.sql.SQLContext;

import org.apache.spark.sql.functions.\_;

import org.apache.spark.sql.types.\_;

import org.apache.spark.sql.\_;

import org.apache.spark.mllib.stat.Statistics;

import org.apache.spark.sql.SparkSession;

val spark = SparkSession.builder().appName("Spark SQL basic example").config("spark.some.config.option", "some-value").getOrCreate()

import spark.\_;

import sqlContext.implicits.\_;

case class Income(id: Double, workclass: String, education: String, maritalstatus: String, occupation: String, relationship: String, race: String, gender: String, nativecountry: String, income: String, age: Double, fnlwgt: Double, educationalnum: Double, capitalgain: Double, capitalloss: Double, hoursperweek: Double)

def parseIncome(str: String): Income = {

val line = str.split(",")

Income(line(0).toDouble, line(1).toString, line(2).toString, line(3).toString,line(4).toString, line(5).toString, line(6).toString, line(7).toString,line(8).toString,line(9).toString, line(10).toDouble, line(11).toDouble, line(12).toDouble,line(13).toDouble,line(14).toDouble, line(14).toDouble)

}

def parseRDD(rdd: RDD[String]): RDD[Income] = {

val header = rdd.first

rdd.filter(\_(0) != header(0)).map(parseIncome).cache()

}

val Income = parseRDD(sc.textFile("C:/Users/CST/Income.csv")).toDF.cache()

// Display Data Frame

Income.printSchema()

Income.show()

Income.groupBy("income").avg("age").show()

