**SOLUCION TIPO PRACTICA 1: HADDOP**

**mapperMaxTemp.py**

#!/usr/bin/python

import sys

"""

Mapper de MaxTemp

"""

# Por cada medida calculamos los pares <anyo, temp>

for linea in sys.stdin:

linea = linea.strip()

anyo, mes, temp = linea.split("\t", 2)

print("%s\t%s" % (anyo, temp))

**reducerMaxTemp.py**

#!/usr/bin/python

import sys

"""

Reducer de MaxTemp

"""

subproblema = None

tempMaxima = None

for claveValor in sys.stdin:

anyo, temp = claveValor.split("\t", 1)

#convertimos la temp a float

temp = float(temp)

#El primer subproblema es el primer anyo de reducer (y la temp máxima de momento también)

if subproblema == None:

subproblema = anyo

tempMaxima = temp

#si el anyo es del subproblema actual, comprobamos si es la temperatura maxima

if subproblema == anyo:

if temp > tempMaxima:

tempMaxima = temp

else: #si ya acabamos con el subproblema, emitimos

print("%s\t%s" % (subproblema, tempMaxima))

#Pasamos al siguiente subproblema (de momento la temp es la máxima)

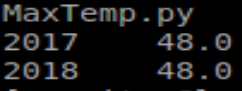
subproblema = anyo

tempMaxima = temp

#el anterior bucle no emite el último subproblema

print("%s\t%s" % (subproblema, tempMaxima))

hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-2.8.5.jar -files ./mapperMaxTemp.py -mapper ./mapperMaxTemp.py -file ./reducerMaxTemp.py -reducer ./reducerMaxTemp.py -input medidas.txt -output ./miSalidaMaxTemp1



**SOLUCION TIPO PRACTICA 2: PYSPARK**

Se puede calcular la media del dataframe df de varias maneras:

* average\_rating = df.selectExpr("avg(overall\_rating) as avg\_potencial").first()["avg\_potencial"]
* average\_rating = df.select(avg("overall\_rating")).collect()[0][0]
* average\_rating = df.select(avg(col("overall\_rating"))).first()[0]
* average\_rating = df.select(mean("overall\_rating")).collect()[0][0]

Lo que se podría haber hecho de otras maneras es:

* Filtrar los nulos que había en el campo
* Había jugadores repetidos. Os podíais haber quedado con el máximo de cada jugador, el primero o el último por fecha, …

La función a crear, podría ser del tipo:

def función\_potencial(campo):

if campo > average\_rating:

return "alto"

else:

return "bajo"

Y aplicarla como sigue:

from pyspark.sql.functions import udf

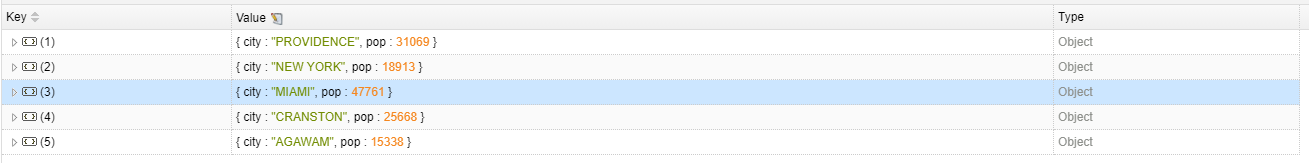
df = df.withColumn("potencial\_alto\_bajo ", función\_potencial("overall\_rating"))

**SOLUCION TIPO PRACTICA 3: MONGO**

El ‘truco’ en estas query es que aparecen ciudades en blanco y esas no deberían mostrarse.

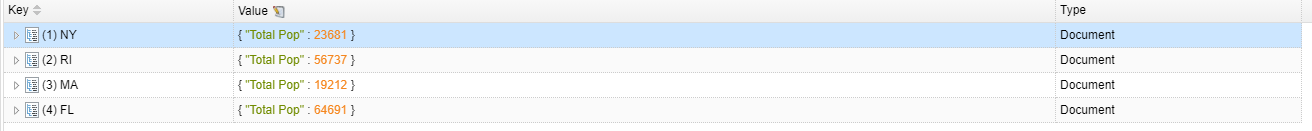
1. Mostrar el nombre y la población de las ciudades donde la población supera los 10000 habitantes

db.cities.find({ "pop": { $gt: 10000 }, "capital": { $exists: true } }, {"\_id": 0, "city": 1, "pop": 1 })



1. Mostrar el nombre y la población del estado según las ciudades mostradas anteriormente

db.cities.aggregate([{$match:{city:{$exists: true}}},{$group:{\_id:"$state", "Total Pop": {$sum:"$pop"}}}])



1. Muestra el \_id, ciudad, nombre de la ciudad capital de cada estado con una población mayor a 20,000 habitantes.

db.docs.find({city:{$exists:true}, pop:{$gt:20000}}, {city:1, "capital.name":1})

