

# Portafolio de CBD

Manuel José Martínez Baños

May 24, 2021

## Contents

<b>1</b>	<b>Introducción</b>	<b>2</b>
<b>2</b>	<b>Hadoop Streaming – Datos de Compras</b>	<b>2</b>
2.1	Ventas Agregadas por Categoría de Producto a través de Todas las Tiendas	3
2.1.1	Solución . . . . .	3
2.1.2	Resultados . . . . .	4
2.2	Valor de la Venta de Mayor Importe para Cada Tienda . . . . .	5
2.2.1	Solución . . . . .	5
2.2.2	Resultados . . . . .	5
<b>3</b>	<b>Hadoop Streaming – Logs de un Servidor Web</b>	<b>7</b>
3.1	Número Total de Accesos a un Recurso . . . . .	8
3.1.1	Solución . . . . .	8
3.1.2	Resultados . . . . .	8
3.2	Número Total de Accesos desde una Misma Dirección IP . . . . .	10
3.2.1	Solución . . . . .	10
3.2.2	Resultados . . . . .	10
3.3	Recurso Web Más Popular . . . . .	11
3.3.1	Solución . . . . .	11
3.3.2	Resultados . . . . .	11
<b>4</b>	<b>Spark – Compras y Logs</b>	<b>12</b>
4.1	Ejercicio 0 . . . . .	13
4.1.1	Solución . . . . .	13
4.2	Ejercicio 1 . . . . .	13
4.2.1	Solución . . . . .	13
4.3	Ejercicio 2 . . . . .	14
4.3.1	Solución . . . . .	14
4.4	Ejercicio 3 . . . . .	15
4.4.1	Solución . . . . .	15
4.5	Ejercicio 4 . . . . .	15
4.5.1	Solución . . . . .	15

**Abstract**

En el siguiente documento, se expone el trabajo realizado durante la práctica de MapReduce y Apache Hadoop, así como, los ejercicios evaluables de Apache Spark.

## 1 Introducción

Cuando la resolución a un problema implica tratar con grandes cantidades de datos (Datasets), llega a un punto donde no es viable trabajar sobre equipos locales o ciertos modelos de programación.

En unas de las prácticas realizadas en la asignatura de CBD, se presenta el modelo de programación MapReduce, que viene a resolver las limitaciones (en casos concretos) ante Datasets de gran tamaño. En dicha práctica (MapReduce y Apache Hadoop) es utilizado un cluster MapReduce levantado sobre una plataforma Cloud y como implementación del modelo de programación MapReduce es utilizado Apache Hadoop, junto a HDFS, permitiendo construir un sistema de ficheros distribuidos. Por último, se trabaja sobre Spark, instalado sobre el cluster Hadoop, reemplazando la implementación de MapReduce de Hadoop por la suya propia más eficiente.

En los siguientes apartados son mostrados y solucionados los evaluables propuestos de la práctica de MapReduce y Apache Hadoop. Para finalizar, se exponen los ejercicios evaluables de Apache Spark y sus correspondientes soluciones.

## 2 Hadoop Streaming – Datos de Compras

En los evaluables propuestos 4 y 5 de MapReduce y Apache Hadoop, se ha trabajado sobre un conjunto de datos que incluye un registro de las compras realizadas en las diferentes tiendas de una cadena con el siguiente formato:

date	time	store name	item description	cost	method of payment
2012-01-01	09:00	San	Diego Music	66.08	Cash
2012-01-01	09:00	Pittsburgh	Pet Supplies	493.51	Discover
2012-01-01	09:00	Omaha	Kid's Clothing	235.63	MasterCard

El Dataset de compras se encuentra copiado en el sistema de archivos HDFS del cluster Hadoop en `/datasets/purchases/purchases.txt` y es utilizado por los dos primeros evaluables, para los siguientes ejercicios se trabaja sobre un Dataset de logs de un Servidor Web, este será detallado más adelante en el apartado 1.2.

Para poder trabajar mejor con el Dataset anterior, primero se han realizado pruebas sobre fragmento del Dataset en local como se muestra a continuación:

```
$ cat /opt/datasets/purchases.txt | head -n 50 | purchases/mapper.py | sort -k 1,1 |
  purchases/reducer.py | head -n 20
```

Una vez verificado el funcionamiento, se comprueba el funcionamiento del trabajo realizado sobre el Dataset completo, con ayuda de un script para la invocación de un trabajo en Hadoop Streaming. Este script contiene el siguiente contenido:

```
$ more `which hs`  
/opt/hadoop/bin/hadoop jar /opt/hadoop/share/hadoop/tools/lib/hadoop-streaming-*.jar -  
mapper $1 -reducer $2 -file $1 -file $2 -input $3 -output $4
```

## 2.1 Ventas Agregadas por Categoría de Producto a través de Todas las Tiendas

En este primer evaluable se pide crear dos nuevos scripts mapper.py y reducer.py, para obtener la suma total de ventas para cada categoría de producto, independientemente de la tienda en la que se haya realizado la compra.

### 2.1.1 Solución

En este apartado se muestra el código implementado para la resolución del evaluable, en concreto, dos scripts mapper.py y reducer.py.

Código 1: mapper.py

```
#!/usr/bin/python  
  
# Format of each line is:  
# date\ttime\tstore name\titem description\tcost\tmethod of payment  
#  
# We want elements 2 (store name) and 4 (cost)  
# We need to write them out to standard output, separated by a tab  
  
import sys  
  
for line in sys.stdin:  
    data = line.strip().split("\t")  
    if len(data) == 6:  
        date, time, store, item, cost, payment = data  
        print "{0}\t{1}".format(item, cost)
```

Código 2: reducer.py

```
#!/usr/bin/python  
import sys  
  
salesTotal = 0  
oldKey = None  
  
for line in sys.stdin:  
    data_mapped = line.strip().split("\t")  
    if len(data_mapped) != 2:  
        # Something has gone wrong. Skip this line.  
        continue  
  
    thisKey, thisSale = data_mapped  
  
    if oldKey and oldKey != thisKey:  
        print oldKey, "\t", salesTotal  
        oldKey = thisKey;  
        salesTotal = 0  
    oldKey = thisKey  
    salesTotal += float(thisSale)  
  
if oldKey != None:  
    print oldKey, "\t", salesTotal
```

### 2.1.2 Resultados

Primero se comprueban los resultados sobre un fragmento del Dataset, sin hacer uso de hadoop:

```
$ cat /opt/datasets/purchases.txt | head -n 1000 | ./mapper.py | sort -k  
1,1 | ./reducer.py | head -n 10  
Baby      12412.82  
Books     13769.89  
Cameras   13138.99  
CDs       15183.28  
Children's Clothing 13310.85  
Computers 11632.52  
Consumer Electronics 14587.07  
Crafts    14329.82  
DVDs      13050.68  
Garden    13947.95
```

A continuación se muestra el resultado obtenido sobre el Dataset **purchases.txt**, este debe coincidir con el siguiente solución:

```
Baby 57491808.44  
Books 57450757.91  
CDs 57410753.04  
Cameras 57299046.64  
Children's Clothing 57624820.94  
Computers 57315406.32  
Consumer Electronics 57452374.13  
Crafts 57418154.5  
DVDs 57649212.14  
Garden 57539833.11  
Health and Beauty 57481589.56  
Men's Clothing 57621279.04  
Music 57495489.7  
Pet Supplies 57197250.24  
Sporting Goods 57599085.89  
Toys 57463477.11  
Video Games 57513165.58  
Women's Clothing 57434448.97
```

Lanzamiento del "evaluable 4" con Hadoop Streaming al cluster Hadoop y obtención del resultado:

```
$ ../hs mapper.py reducer.py /datasets/purchases purchases_task4/output  
$ hadoop fs -text purchases_task4/output/part-00000 | head -n 20  
Baby      57491808.44  
Books     57450757.91  
CDs       57410753.04  
Cameras   57299046.64  
Children's Clothing 57624820.94  
Computers 57315406.32  
Consumer Electronics 57452374.13  
Crafts    57418154.5  
DVDs      57649212.14  
Garden    57539833.11  
Health and Beauty 57481589.56  
Men's Clothing 57621279.04  
Music     57495489.7  
Pet Supplies 57197250.24  
Sporting Goods 57599085.89  
Toys      57463477.11  
Video Games 57513165.58  
Women's Clothing 57434448.97
```

## 2.2 Valor de la Venta de Mayor Importe para Cada Tienda

En el siguiente evaluable número 5 se pide crear dos nuevos scripts mapper.py y reducer.py, para obtener el valor de la venta de mayor importe para cada tienda.

### 2.2.1 Solución

En este apartado se muestra el código implementado para la resolución del evaluable 5, en concreto, dos scripts mapper.py y reducer.py.

Código 3: mapper.py

```
#!/usr/bin/python

import sys

for line in sys.stdin:
    data = line.strip().split("\t")
    if len(data) == 6:
        date, time, store, item, cost, payment = data
        print "{0}\t{1}".format(store, cost)
```

Código 4: reducer.py

```
#!/usr/bin/python
import sys

maxTotal = 0
oldKey = None

for line in sys.stdin:
    data_mapped = line.strip().split("\t")
    if len(data_mapped) != 2:
        # Something has gone wrong. Skip this line.
        continue

    thisKey, thisSale = data_mapped

    if oldKey and oldKey != thisKey:
        print oldKey, "\t", maxTotal
        oldKey = thisKey;
        maxTotal = 0

    oldKey = thisKey
    if float(thisSale) > float(maxTotal):
        maxTotal=float(thisSale)

if oldKey != None:
    print oldKey, "\t", maxTotal
```

### 2.2.2 Resultados

En este apartado, se comprueba el código implementado, con un lanzamiento en local que trata solo con fragmento del Dataset, este debe ofrecer un resultado como el siguiente:

```
cat /opt/datasets/purchases.txt | head -n 1000 | ./mapper.py | sort -k
1,1 | ./reducer.py | head -n 10
Albuquerque 484.24
Anaheim 456.53
Anchorage 450.6
Arlington 400.08
```

```
Atlanta 484.31
Aurora 495.97
Austin 483.1
Bakersfield 498.93
Baltimore 467.63
Baton Rouge 411.54
```

El resultado anterior debe coincidir con el siguiente resultado correcto:

```
task-5$ cat /opt/datasets/purchases.txt | head -n 1000 | ./mapper.py |
      sort -k 1,1 | ./reducer.py | head -n 10
Albuquerque      484.24
Anaheim          456.53
Anchorage        450.6
Arlington        400.08
Atlanta          484.31
Aurora   495.97
Austin   483.1
Bakersfield      498.93
Baltimore        467.63
Baton Rouge      411.54
```

Por último, es lanzado el código en hadoop para tratar con todo el Dataset de **purchases.txt**, el resultado obtenido debe coincidir con el siguiente resultado correcto:

```
Albuquerque 499.98
Anaheim 499.98
Anchorage 499.99
Arlington 499.95
Atlanta 499.96
Aurora 499.97
-----
Virginia Beach 499.98
Washington 499.98
Wichita 499.97
```

En el siguiente fragmento es mostrado el lanzamiento del "evaluable 5" con Hadoop Streaming al cluster Hadoop y la obtención del resultado:

```
task-5$ ../hs mapper.py reducer.py /datasets/purchases purchases_task5/
      output
task-5$ hadoop fs -text purchases_task5/output/part-00000 | head -n 20
Albuquerque      499.98
Anaheim          499.98
Anchorage        499.99
Arlington        499.95
Atlanta          499.96
Aurora   499.97
Austin   499.97
Bakersfield      499.97
Baltimore        499.99
Baton Rouge      499.98
Birmingham      499.99
Boise   499.98
Boston   499.99
Buffalo   499.99
Chandler   499.98
Charlotte   499.98
Chesapeake   499.98
Chicago     499.99
Chula Vista  499.99
Cincinnati   499.98
```

### 3 Hadoop Streaming – Logs de un Servidor Web

En los siguientes evaluables propuestos 6,7 y 8 de MapReduce y Apache Hadoop, se ha trabajado sobre con un conjunto de datos que incluye el log de acceso al servidor web Apache de una empresa para poder resolver varios problemas relacionados con el análisis de los archivos de registro (logs). El formato en el que los datos se estructuran en dicho fichero (Common Log Format) es el siguiente:

date	time	store name	item description	cost	method of payment
2012-01-01	09:00	San	Diego Music	66.08	Cash
2012-01-01	09:00	Pittsburgh	Pet Supplies	493.51	Discover
2012-01-01	09:00	Omaha	Kid's Clothing	235.63	MasterCard

Este dataset, al igual que purchases, se dispone de un conjunto de datos en */opt/datasets/access.log* en el entorno de prácticas y ya ha sido copiado a HDFS y está disponible en */datasets/accesslog/access.log*. El Dataset ocupa 482 MBytes y consta de 4.477.843 líneas de texto en formato Common Log Format.

### 3.1 Número Total de Accesos a un Recurso

En este evaluable número 6 se pide crear dos nuevos scripts mapper.py y reducer.py, para obtener el número total de accesos a cada uno de los recursos de la web de la que se disponen los logs.

#### 3.1.1 Solución

Ahora se muestra el contenido de los dos scripts mapper.py y reducer.py para la obtención de la solución.

Código 5: mapper.py

```
#!/usr/bin/python
import sys

for line in sys.stdin:
    data = line.strip().split(" ") #Split by single space
    if len(data) == 10:
        ip, idclient, iduser, date, date2, typereq, item, protocol,
        codestate, size = data
        print "{0}\t{1}".format(item, typereq)
```

Código 6: reducer.py

```
#!/usr/bin/python

import sys

getTotal = 0
oldKey = None

for line in sys.stdin:
    data_mapped = line.strip().split("\t")
    if len(data_mapped) != 2:
        # Something has gone wrong. Skip this line.
        continue

    thisKey, thisReq = data_mapped
    if oldKey and oldKey != thisKey:
        print oldKey, "\t", getTotal
        oldKey = thisKey;
        getTotal = 0

    oldKey = thisKey
    getTotal += 1

if oldKey != None:
    print oldKey, "\t", getTotal
```

#### 3.1.2 Resultados

Ahora se comprueba el código implementado. En primer lugar, con un lanzamiento en local, tratando solo con fragmento del Dataset, este debe ofrecer un resultado como el siguiente:

```
cat /opt/datasets/access_log | head -n 1000 | ./mapper.py | sort -k 1,1 | ./reducer.py
| head -n 15
/ 37
/about-us/ 6
/about-us 6
```



```

/about-us/campaigns/ 1
/about-us/campaigns 1
/about-us/people/ 2
/about-us/people 2
/assets/css/960.css 54
/assets/css/reset.css 54
/assets/css/the-associates.css 54
/assets/flv/dummy/home-media-block/district-13.flv 2

```

El resultado anterior debe coincidir con el siguiente resultado generado por el lanzamiento de los dos scripts MapReduce creados:

```

task-6$ cat /opt/datasets/access_log | head -n 1000 | ./mapper.py |
      sort -k 1,1 | ./reducer.py | head -n 15
/
37
/about-us      6
/about-us/    6
/about-us/campaigns      1
/about-us/campaigns/    1
/about-us/people      2
/about-us/people/      2
/assets/css/960.css      54
/assets/css/reset.css    54
/assets/css/the-associates.css    54
/assets/flv/dummy/home-media-block/district-13.flv      2

```

Por último, es lanzado el código en Hadoop para tratar con todo el Dataset de logs completo, el resultado obtenido debe coincidir con el resultado de estos tres recursos, para verificar su correcto funcionamiento:

```

/assets/img/home-logo.png 98744
/assets/css/reset.css 2382
/assets/js/lowpro.js 293

```

Seguidamente, se muestra el lanzamiento del "evaluable 6" con Hadoop Streaming al cluster Hadoop y el resultado obtenido:

```

task-6$ ../hs mapper.py reducer.py /datasets/accesslog/access_log
common_logs/output
task-6$ hadoop fs -text common_logs/output/part-00000 | grep -E '^/
assets/img/home-logo.png '
/assets/img/home-logo.png      98744

task-6$ hadoop fs -text common_logs/output/part-00000 | grep -E '^/
assets/css/reset.css '
/assets/css/reset.css    2382

task-6$ hadoop fs -text common_logs/output/part-00000 | grep -E '^/
assets/js/lowpro.js '
/assets/js/lowpro.js    293

```

## 3.2 Número Total de Accesos desde una Misma Dirección IP

En el siguiente evaluable número 7 se pide crear dos nuevos scripts mapper.py y reducer.py, para obtener el número total de accesos desde la IP 10.223.157.186 a cualquiera de los recursos de la web de la que se disponen los logs.

### 3.2.1 Solución

A continuación, el contenido de los dos scripts mapper.py y reducer.py para la obtención de los accesos a la IP 10.223.157.186.

Código 7: mapper.py

```
#!/usr/bin/python
import sys

for line in sys.stdin:
    data = line.strip().split(" ") #Split by single space
    if len(data) == 10:
        ip, idclient, iduser, date, date2, typereq, item, protocol,
        codestate, size = data
        print "{0}\t{1}".format(ip, typereq)
```

Código 8: reducer.py

```
#!/usr/bin/python

import sys

getTotal = 0
findKey = '10.223.157.186'

for line in sys.stdin:
    data_mapped = line.strip().split("\t")
    if len(data_mapped) != 2:
        # Something has gone wrong. Skip this line.
        continue

    thisKey, thisGet = data_mapped

    if thisKey == findKey:
        getTotal += 1

print findKey, "\t", getTotal
```

### 3.2.2 Resultados

En este apartado, se comprueba el código implementado. Es posible comprobar mediante comandos del terminal el resultado. Esta opción no es óptima porque solo es utilizado un nodo al ejecutarse.

```
cat /opt/datasets/access_log | grep 10.223.157.186 | wc -l
115
```

El resultado anterior debe coincidir con el siguiente resultado utilizando Hadoop Streaming mediante el script hs:

```
task-7$ ../hs mapper.py reducer.py /datasets/accesslog/access_log
common_logs/output
task-7$ hadoop fs -text common_logs/output/part-00000
10.223.157.186 115
```

### 3.3 Recurso Web Más Popular

En el siguiente evaluable número 8 se pide crear dos nuevos scripts mapper.py y reducer.py, para obtener el recurso web más popular, es decir, aquel que recibió el mayor número de accesos.

#### 3.3.1 Solución

A continuación, los dos scripts mapper.py y reducer.py para la obtención de la solución.

Código 9: mapper.py

```
#!/usr/bin/python
import sys

for line in sys.stdin:
    data = line.strip().split(" ") #Split by single space
    if len(data) == 10:
        ip, idclient, iduser, date, date2, typereq, item, protocol,
        codestate, size = data
        print "{0}\t{1}".format(item, typereq)
```

Código 10: reducer.py

```
#!/usr/bin/python
import sys

getTotal = 0
oldKey = None
maxTotal=0;
maxItem=None;

for line in sys.stdin:
    data_mapped = line.strip().split("\t")
    if len(data_mapped) != 2:
        # Something has gone wrong. Skip this line.
        continue
    #Get typeReq if we want filter by type request like GET
    thisKey, thisTypeReq = data_mapped

    if oldKey and oldKey != thisKey:
        if getTotal > maxTotal:
            maxTotal=getTotal;
            maxItem=oldKey;
        oldKey = thisKey;
        getTotal = 0

    oldKey = thisKey
    getTotal += 1

if maxItem != None:
    print maxItem, "\t", maxTotal
```

#### 3.3.2 Resultados

Se comprueba el resultado con un lanzamiento en local, tratando solo con fragmento del dataset, este debe ofrecer un resultado como el siguiente:

```
cat /opt/datasets/access_log | head -n 1000 | ./mapper.py | sort -k 1,1
| ./reducer.py
/assets/css/960.css 54
```

Este resultado debe coincidir con el resultado del lanzamiento de los dos scripts mostrados anteriormente:

```
task-8$ cat /opt/datasets/access_log | head -n 1000 | ./mapper.py |
      sort -k 1,1 | ./reducer.py
/assets/css/960.css      54
```

Por último, es lanzado el código con Hadoop Streaming al cluster para tratar con todo el Dataset de logs, el recurso más popular debe ser **/assets/css/combined.css con 117348 hits**:

```
task-8$ hadoop fs -rm -r common_logs/output

task-8$ ../hs mapper.py reducer.py /datasets/accesslog/access_log
      common_logs/output

task-8$ hadoop fs -text common_logs/output/part-00000
/assets/css/combined.css      117326
```

## 4 Spark – Compras y Logs

En los anteriores evaluables se ha tratado con Hadoop Streaming, en los siguientes ejercicios se trabaja con el framework para el procesamiento de datos de gran escala **Apache Spark**.

Spark implementa de forma distinta el MapReduce de Hadoop, para paliar algunas de sus flaquezas. Por ejemplo, la penalización en prestaciones cuando hay múltiples escrituras en HDFS seguidas de lecturas de HDFS para la siguiente iteración de MapReduce. Estos casos se dan, por ejemplo, cuando se desea realizar una página de ranking o procesamiento de grafos, donde se requiere la ejecución de procesos MapReduce de forma iterativa. Spark mantiene los datos intermedios en memoria y optimiza el trabajo MapReduce en función las operaciones realizadas gracias al concepto de RDD (Resilient Distributed Dataset) para minimizar penalizaciones.

En esta práctica, Spark está instalado sobre el mismo cluster Hadoop donde se han realizado los evaluables anteriores y son utilizados los mismos Datasets previamente detallados. En el apartado que viene a continuación, se muestra los ejercicios realizados sobre Spark, con el mismo enunciado, en su mayoría, a los anteriormente creados con Hadoop Streaming.


## 4.1 Ejercicio 0

En este ejercicio es igual al evaluable 5 donde se pide obtener para cada tienda, la transacción de máximo importe. Este ejercicio y todos los posteriores se ha utilizado la sesión interactiva PySpark en el cluster. El Dataset utilizado es `/datasets/purchases/purchases.txt`.

### 4.1.1 Solución

A continuación una imagen con la resolución de este primer ejercicio número 0 con PySpark.

```

Welcome to
 version 3.0.1

Using Python version 2.7.6 (default, Nov 13 2018 12:45:42)
SparkSession available as 'spark'.

>>> purchases = sc.textFile("/datasets/purchases/purchases.txt")
>>> splits = purchases.map(lambda s: s.split("\t"))
>>> costByStore = splits.map(lambda rec: (rec[2], float(rec[4])))
>>> maxCostByStore = costByStore.reduceByKey(max)
>>> print maxCostByStore.take(10)
[(u'Portland', 499.96), (u'Rochester', 499.99), (u'Baltimore', 499.99), (u'Denver', 499.97), (u'Newark', 499.99), (u'Glendale', 499.98), (u'Louisville', 499.98), (u'Reno', 499.99), (u'Santa Ana', 499.97), (u'Spokane', 499.99)]
>>>

```

Figure 1: Ejercicio 0 en PySpark

Como se aprecia en la imagen, los resultados son parejos al evaluable correspondiente, en este caso, el resultado no esta ordenado alfabéticamente por nombre de tienda. El ejercicio 0 se resume en las siguientes lineas de código:

Código 11: Ejercicio 0

```
>>> purchases = sc.textFile("/datasets/purchases/purchases.txt")
>>> splits = purchases.map(lambda s: s.split("\t"))
>>> costByStore = splits.map(lambda rec: (rec[2], float(rec[4])))
>>> maxCostByStore = costByStore.reduceByKey(max)
>>> print maxCostByStore
```

## 4.2 Ejercicio 1

En este ejercicio se solicita la suma total de ventas para cada categoría de producto, al igual que el evaluable 4. Por tanto se trabaja sobre el Dataset `/datasets/purchases/purchases.txt`.

### 4.2.1 Solución

Código 12: Ejercicio 1

```
>>> purchases = sc.textFile("/datasets/purchases/purchases.txt")
>>> splits = purchases.map(lambda s: s.split("\t"))
>>> StoreTypes = splits.map(lambda rec: (rec[3], float(rec[4])))
>>> group = StoreTypes.groupByKey()
>>> total = group.mapValues(lambda counts: sum(counts))
>>> print total.take(10)
[(u"Men's Clothing", 57621279.04000138),
 (u'Pet Supplies', 57197250.0000008),
 (u"Women's Clothing", 57434448.969999881),
```

```
(u'Music', 57495489.700000465),
(u'Sporting Goods', 57599085.890000574),
(u'Baby', 57491808.43999965),
(u'Children's Clothing', 57624820.94000126),
(u'Health and Beauty', 57481589.560001),
(u'Consumer Electronics', 57452374.12999909),
(u'CDs', 57410753.04000111)]
```

Como se puede observar, con resultado generado por "print total.take(10)" se obtienen los 10 primeros valores. Los resultados de cada categoría son iguales al evaluable 4, con la diferencia de que no están ordenados por alfabéticamente por nombre de categoría:

```
Baby 57491808.44
Books 57450757.91
CDs 57410753.04
Cameras 57299046.64
Children's Clothing 57624820.94
Computers 57315406.32
Consumer Electronics 57452374.13
Crafts 57418154.5
DVDs 57649212.14
Garden 57539833.11
Health and Beauty 57481589.56
Men's Clothing 57621279.04
Music 57495489.7
Pet Supplies 57197250.24
```

## 4.3 Ejercicio 2

En el siguiente ejercicio solicita obtener número total de accesos al recurso `/assets/img/home-logo.png`. A partir de los siguientes ejercicios, se trata sobre el Dataset de Logs de un servidor localizado en `"/datasets/accesslog/access_log"`, al ser similar al evaluable 6, se sabe con certeza que el resultado debe ser **98744**

### 4.3.1 Solución

Código 13: Ejercicio 2

```
>>> logFile = sc.textFile("/datasets/accesslog/access_log")
>>> splits = logFile.map(lambda s: s.split(" "))
>>> byFile = splits.filter(lambda x: "/assets/img/home-logo.png" in x)
>>> total = byFile.count()
>>> print total
98744
```

## 4.4 Ejercicio 3

En el siguiente ejercicio se debe de obtener número total de accesos desde la misma dirección **IP: 10.223.157.186**, al igual que el evaluable 7 debe de dar como resultado un número total de accesos de **115**.

### 4.4.1 Solución

Código 14: Ejercicio 3

```
>>> logFile = sc.textFile("/datasets/accesslog/access_log")
>>> splits = logFile.map(lambda s: s.split(" "))
>>> byFile = splits.filter(lambda x: "10.223.157.186" in x)
>>> total = byFile.count()
>>> print total
115
```

## 4.5 Ejercicio 4

En este último ejercicio, consiste en obtener el recurso web con mayor número de accesos. Como en el evaluable 8, el recurso más popular debe ser **/assets/css/combined.css** con **117348 hits**:

### 4.5.1 Solución

Código 15: Ejercicio 4

```
>>> logFile = sc.textFile("/datasets/accesslog/access_log")
>>> splits = logFile.map(lambda s: s.split(" "))
>>> ByResource = splits.map(lambda rec: (rec[6], 1))
>>> group = ByResource.groupByKey()
>>> totalsByResource = group.mapValues(lambda counts: sum(counts))
>>> max = totalsByResource.max(lambda x:x[1])
>>> print max
(u'/assets/css/combined.css', 117348)
```