

2024 - 2025

**Máster en Big Data y Data Science**

**03MBID – Procesamiento de Datos Masivos**

**Nombre: Gonzalo Antonio Delgado Rubio**

**Fecha: 22/06/2024**

**Curso 2022 – Ed. Abril**

**ACTIVIDAD 2**

# Problema 1

Dado un dataset que contenga entradas con la forma “persona;método\_pago;dinero\_gastado”, crea un programa llamado personaGastosConTarjetaCredito que para cada persona indique la suma del dinero gastado con tarjeta de crédito, con el formato persona;gastoconTDC. Ejemplo:

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Notar que Alice gasta en total 450 euros, pero sólo 250 son con tarjeta de crédito (100 + 150). Se valorará positivamente la eficiencia del programa, por ejemplo, no usar transformaciones innecesarias.

## Solución

* Se Inicializa creando el contexto spark y seteando ando variables para el input y output de la data. (Se debe cambiar según ejecución)

from pyspark import SparkContext, SparkConf

# Crear un contexto de Spark

conf = SparkConf().setAppName("problema1")

sc = SparkContext(conf=conf)

# Constantes -- cambiar según requerimiento

PATH\_INPUT = './problema1/casoDePrueba1.txt'

PATH\_OUT = './problema1/output/personaGastosConTarjetaCredito'

* Se inicializa programa con función main, generando una lectura de los datos mediante un textFile
* Aplicando una función map separamos la información según el separador de archivo ‘;’
* Seleccionamos columnas y damos formato a columna de gasto
* Empleamos una función map para seleccionar la persona y gasto con TDC
* Finalmente empleamos función reduceByKey para agrupar datos.

# 1. lectura de datos

    input\_rdd = sc.textFile(PATH\_INPUT)

    # 2. Procesamiento de datos

    # 2.1 Convirtiendo datos en tupla

    tuple\_rdd = (input\_rdd

        .map(lambda line: line.split(";"))

        .map(lambda x: (x[0], x[1], float(x[2])))

    )

    # 2.2 Seleccionamos persona y gasto en caso este sea con TDC en otro caso se coloca 0

    process\_rdd = tuple\_rdd.map(lambda x: (x[0], x[2] if x[1] == 'Tarjeta de crédito' else 0))

    # 2.3 Sumarizamos los gastos por persona

    reduce\_rdd = process\_rdd.reduceByKey(lambda x, y: x + y)

* Por ultimo se genera función de escritura para almacenar datos en output path.

def writeRddAsText(rrd ,file\_path:str):

    """Escribe un rdd en un archivo de texto según una ruta data"""

    import shutil

    # Eliminamos directorio si existe

    shutil.rmtree(file\_path, ignore\_errors=True)

    # Generamos nuevo rdd con separador

    rdd\_save = rrd.map(lambda x: f"{x[0]};{x[1]}")

    # almacenamos data en path

    rdd\_save.saveAsTextFile(file\_path)

    pass

## Ejecución de Programa

Se ejempla comando de ejecución desde el repositorio

spark-submit problema1/personaGastosConTarjetaCredito.py

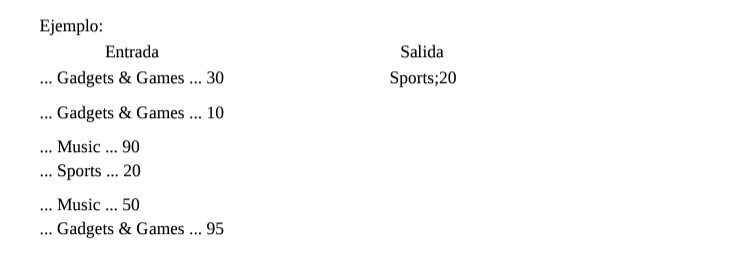
## 1.2 Evidencia Ejecución

Captura de pantalla de computadora

Descripción generada automáticamente

# Problema 2

Dado un dataset que contiene información sobre los videos de Youtube (https://netsg.cs.sfu.ca/youtubedata/), crear un programa llamado CategoriaDeVideosMenosVista que obtenga cuál es la categoría de videos menos vista de la plataforma Youtube y el número total de visualizaciones que hay en esa categoría. El programa debe recibir dos parámetros de entrada: la carpeta en la que está el dataset y la carpeta en la que se guardará el resultado. En la carpeta donde está el dataset se tienen que descomprimir UNO de los archivos 0222.zip, 0301.zip, etc., que se encuentran en el enlace anterior. Importante: si la persona que hace la actividad dispone de pocos recursos computaciones, entonces se recomienda que únicamente descomprima algún .zip pequeño para que pueda desarrollar el programa. La carpeta de datos de entrada debería quedar como se ve en la Figura 1. Los datos de entrada están en los archivos 0.txt, 1.txt, etc y cada fila contiene la información de un video tabulada con el siguiente formato: id del video de youtube, usuario que subió el video, número de días desde que se subió el video y la fecha en la que obtuvieron los datos, categoría del video, longitud del video, número de visitas del video, puntuación del video, número de puntuaciones del video, número de comentarios del video, y una lista de ids de videos relacionados. Se valorará positivamente la eficiencia del programa, por ejemplo no usar transformaciones innecesarias.



Notar que la categoría “Sports” es la que menos visitas tiene: 20 en un único vídeo. “Music” es la que más visitas tiene: 90 en un video + 50 en otro video, es decir, en total 140 visitas, y la categoría “Gadgets & Games” tiene en total 135 visitas obtenidas de 30 + 10 + 95. El programa debe funcionar independientemente del número de categorías, para cualquier cantidad de filas que se pueda llegar a tener, para cualquier cantidad de ficheros e ignorar el log.txt.

## 2.1 Solución

* Se inicializa programa generando contexto spark
* Se genera programa de lectura de datos a partir de una ruta data, se descromprime archivo zipeado
* Se emplea método wholeTextFiles para lectura de los archivos .txt
* Se ignora el archivo ‘log.txt’
* Se procede a realizar un flapMap para separar los datos en líneas por cada uno de los archivos leídos.

from pyspark import SparkContext, SparkConf

from zipfile import ZipFile

import os

import sys

import shutil

# Crear un contexto de Spark

conf = SparkConf().setAppName("problema2")

sc = SparkContext(conf=conf)

def read\_rdd\_from\_dataset(dataset\_path:str):

    """Lectura de un rdd desde un archivo zip"""

    folder\_path = os.path.dirname(dataset\_path)

    filename = os.path.basename(dataset\_path).split('.')[0]

    # 1. Eliminamos la carpeta data si existe

    shutil.rmtree(os.path.join(folder\_path, 'data', filename))

    # descomprimiendo el archivo

    with ZipFile(dataset\_path, 'r') as zip\_ref:

        zip\_ref.extractall(os.path.join(folder\_path, 'data'))

    # lectura de los archivos, eliminando el archivo log.txt

    rdd = sc.wholeTextFiles(os.path.join(folder\_path,'data', filename))

    rdd = rdd.filter(lambda x: "log.txt" not in x[0])

    # separando por lineas

    rdd\_lines = rdd.flatMap(lambda x: x[1].splitlines())

    return rdd\_lines

* Se separa la información por el separador del archivo ‘tabulador’
* Se eligen las columnas de interés, según posición: categoría y nro de visitas
* # 4. Lectura manteniendo columnas de interes: categoria y nro de visitas
* select\_rdd = (
* input\_rdd
* .map(lambda line: line.split("\t"))
* .filter(lambda fields: len(fields) > 5) # quito nulos
* .map(lambda fields: (fields[3], int(fields[5])))
* )
* Agrupo los datos para sumarizar el nro de visitas
* Finalmente ordeno y elijo el elemento con menor nro de visitas

# 5. Agrupo por categoria y sumo las visitas

    grouped\_rdd = select\_rdd.reduceByKey(lambda a, b: a + b)

    # 6. Encontrar la categoria con menos visitas -> retorna una lista con la categoria y el nro de visitas

    min\_category = grouped\_rdd.takeOrdered(1, key=lambda x: x[1])

* Escribimos data en ruta especificada

 # 7. Escribir el resultado en un archivo de texto

    writeRddAsText(sc.parallelize(min\_category) ,output\_path)

## 2.2 Ejecución Programa

Se ejempla comando de ejecución desde el repositorio

spark-submit problema2/categoriaDeVideosMenosVista.py ./problema2/dataset/0301.zip ./problema2/output

## 2.3 Evidencia Ejecución

Captura de pantalla de computadora

Descripción generada automáticamente

# Problema 3

Dado un dataset que contenga entradas con la forma “persona;método\_pago;dinero\_gastado”, crea un programa llamado personaYMetodosDePago que:

1. Por cada persona indique en cuántas compras pagó más de 1500 euros con un medio de pago diferente a tarjeta de crédito. La solución se tiene que guardar en un archivo llamado comprasSinTDCMayorDe1500.
2. Por cada persona indique en cuántas compras pagó menos o igual a 1500 euros con un medio de pago diferente a tarjeta de crédito. La solución se tiene que guardar en un archivo llamado comprasSinTDCMenoroIgualDe1500.

Se valorará positivamente la eficiencia del programa, por ejemplo, no usar transformaciones innecesarias.

Texto, Carta

Descripción generada automáticamente

## 3.1 Solución

* Creamos nuestro context spark
* Agregamos las rutas del archivo input y output

from pyspark import SparkContext, SparkConf

# Crear un contexto de Spark

conf = SparkConf().setAppName("problema3")

sc = SparkContext(conf=conf)

# Constantes

PATH\_INPUT = './problema3/casoDePrueba3.txt'

PATH\_OUT = './problema3/output'

* Leemos la data con método textFile
* Separamos la información por separador ‘;’ y generamos lista de tuplas de rdd
* # 1. lectura de datos
* input\_rdd = sc.textFile(PATH\_INPUT)
* # 2. Procesamiento de datos
* # 2.1 Convirtiendo datos en tupla
* tuple\_rdd = (input\_rdd
* .map(lambda line: line.split(";"))
* .map(lambda x: (x[0], x[1], float(x[2])))
* )
* Para las compras sin TDC mayores a 1500 generamos una función map que se encargue de realizar la tarea
* Finalmente agrupamos la información sumarizando los datos

def processComprasSinTDCMayorDe1500(rdd):

    """Procesa las compras sin TDC mayores a 1500"""

    # aplico funcion de mapeo

    process\_rdd = rdd.map(lambda x: (x[0], 1 if x[1]!='Tarjeta de crédito' and x[2]>1500 else 0))

    # agrupamiento

    group\_rdd = process\_rdd.reduceByKey(lambda x,y: x+y)

    return group\_rdd

* Para procesar las compras sin TDC con montos menores e iguales a 1500 se emplea función maper
* Agrupamos y sumarizamos los datos
* def processcomprasSinTDCMenoroIgualDe1500(rdd):
* """Procesa las compras sin TDC menores o iguales a 1500"""
* # aplico funcion de mapeo
* process\_rdd = rdd.map(lambda x: (x[0], 1 if x[1]!='Tarjeta de crédito' and x[2]<=1500 else 0))
* # agrupamiento
* group\_rdd = process\_rdd.reduceByKey(lambda x,y: x+y)
* return group\_rdd
* Se recupera rdd de las funciones y por último escribimos resultados en rutas definidas

 # 2.2 Procesando compras sin TDC mayores a 1500

    compras\_rdd = processComprasSinTDCMayorDe1500(tuple\_rdd)

    # 2.3 Procesando compras sin TDC menores o iguales a 1500

    reduce\_rdd = processcomprasSinTDCMenoroIgualDe1500(tuple\_rdd)

    # 3. Escritura de datos

    writeRddAsText(compras\_rdd, f"{PATH\_OUT}/comprasSinTDCMayorDe1500")

    writeRddAsText(reduce\_rdd, f"{PATH\_OUT}/comprasSinTDCMenoroIgualDe1500")

## 3.2 Ejecución Programa

Se ejempla comando de ejecución desde el repositorio

spark-submit problema3/ personaYMetodosDePago.py

## 3.3 Evidencia Ejecución

Texto

Descripción generada automáticamente