

# Project #2

Filipa Rodrigues Santos Capela Nº 2018297335

Tiago Emanuel Pacheco caldeira Conceição Nº 2021167993

# Assignment #1 - Data Processing with Hadoop MapReduce

## Task #1 - Average Number of Friends by Age

- Classe *Friends\_Average(MRJob)* que contém as funções *steps(self)*, *mapper\_get\_friends(self, \_, line)* e *reducer\_avg\_friends(self, key, values)*.
- A função *steps(self)*, que está presente na imagem seguinte, informa o map reduce framework que funções são utilizadas para o mapper e o reduce, que neste caso são *mapper\_get\_friends* e *reducer\_avg\_friends*.

```
def steps(self):
    return [
        MRStep(mapper=self.mapper_get_friends, reducer=self.reducer_avg_friends)
    ]
```

- Seguidamente, fizemos a função *mapper\_get\_friends(self, \_, line)* em que fazemos um split dos valores das colunas e mapeamos os valores no formato (key, value), em que neste caso a key é a idade e o value o número de amigos, ficando assim: (age, friends)
- De seguida, implementámos a função *reducer\_avg\_friends(self, key, values)* que faz uma agregação dos values de acordo com a sua key. Neste caso a agregação feita é uma média dos values (número de amigos), para assim conseguirmos obter a média do número de amigos por idade.

```
def mapper_get_friends(self, _, line):
    (id, name, age, friends) = line.split(",")
    yield age, int(friends)
```

```
def reducer_avg_friends(self, key, values):
    values = list(values)
    new_value = sum(values)/len(values)
    yield key, new_value
```

- Output:

```
Labs#6 > Task1 > output.txt
1 "18" 343.375
2 "19" 213.27272727272728
3 "20" 165.0
4 "21" 350.875
5 "22" 206.42857142857142
6 "23" 246.3
7 "24" 233.8
8 "25" 187.45454545454547
9 "26" 242.05882352941177
10 "27" 228.125
11 "28" 209.1
12 "29" 215.91666666666666
13 "30" 235.8181818181818
14 "31" 267.25
15 "32" 207.9090909090909
16 "33" 325.3333333333333
17 "34" 245.5
18 "35" 211.625
19 "36" 246.6
20 "37" 249.33333333333334
21 "38" 193.53333333333333
22 "39" 169.28571428571428
23 "40" 250.8235294117647
```

```
24 "41" 268.55555555555554
25 "42" 303.5
26 "43" 230.57142857142858
27 "44" 282.16666666666667
28 "45" 309.53846153846155
29 "46" 223.69230769230768
30 "47" 233.22222222222223
31 "48" 281.4
32 "49" 184.66666666666666
33 "50" 254.6
34 "51" 302.14285714285717
35 "52" 340.6363636363636
36 "53" 222.85714285714286
37 "54" 278.0769230769231
38 "55" 295.53846153846155
39 "56" 306.66666666666667
40 "57" 258.8333333333333
41 "58" 116.54545454545455
42 "59" 220.0
43 "60" 202.71428571428572
44 "61" 256.22222222222223
45 "62" 220.76923076923077
46 "63" 384.0
47 "64" 281.3333333333333
48 "65" 298.2
49 "66" 276.44444444444446
50 "67" 214.625
51 "68" 269.6
52 "69" 235.2
```

## Task #2 - Minimum Temperature Per Capital

- Para esta task, o método utilizado é o mesmo que na task1, a diferença destaca-se no que é considerado como (key,value) e como é feita a agregação dos values de cada key. Como nesta task queremos saber a menor temperatura de cada capital, iremos guardar como (key,value) a weather station e a temperatura. A forma como agregamos é obtida através do mínimo valor da temperatura de cada capital. Seguidamente, observamos as funções *mapper\_get\_temperature(self, \_, line)* e *reducer\_min\_temperature(self, key, values)*

```
def mapper_get_temperature(self, _, line):
    (weather_station, date, observation_type, temperature, _, _, _) = line.split(",")
    yield weather_station, int(temperature)
```

```
def reducer_min_temperature(self, key, values):
    values = list(values)
    new_value = min(values)/10
    yield key, new_value
```

- Output:

```
"EZE00100082" -13.5
"G4000010962" 0.0
"ITE00100554" -14.8
```

### Task #3 - Sort the Word Frequency in a Book

- Para esta task, o método utilizado é o mesmo que na task1, a diferença destaca-se na forma de fazer o split, no que é considerado como (key,value) e como é feita a agregação dos values de cada key. Como nesta task, o dataset é um livro, a forma de fazer split nos dados é diferente: iremos utilizar uma expressão regular. Como queremos saber a quantidade de vezes que uma palavra aparece no dataset, o par (key,value) será (palavra,1). Na forma de agregação, iremos somar os values. Como o value é 1, assim conseguimos obter o número de vezes que a palavra aparece o dataset.

```
def mapper_count_words(self, __, line):
    for word in re.split(" |\s|/.\.\s|[^a-zA-Z0-9]+/g",line):
        word = word.lower()
        yield word,1
```

```
def reducer_count_words(self, key, values):
    values = list(values)
    new_value = sum(values)
    yield key, new_value
```

- Output: Como o output desta task é enorme, seguidamente está apenas apresentado uma pequena parte do output. Como podemos ver pelas imagens, as palavras contém algumas “,” e “.”, assim sendo, concluímos que a expressão regular não foi a melhor escolhida,

```
"been" 7
"before" 2
"before," 1
"before." 2
"began" 2
"began," 1
"began." 1
"behind" 5
"being" 6
"believe" 1
"below" 2
"besides" 1
"better" 1
"between" 1
"big" 2
"bit" 4
"bit"" 1
"bite" 1
"biting" 1
"black" 4
"black," 1
"blankly" 1
"bless" 1
"blind" 1
```

```
"bright," 1
"brilliant" 1
"brim" 1
"bring" 1
"brisked" 1
"briskly," 1
"brooked" 1
"brought" 1
"brown" 1
"burden" 1
"burning" 1
"bush" 1
"business," 1
"but" 18
"by" 11
"called" 1
"came" 6
"can" 1
"can't" 1
"carpet," 1
"carriage" 1
"carried" 3
"carried." 1
"carrier" 1
"carrying" 1
"certain" 6
"ertain," 1
```

## Assignment #2 - Data Processing with Hadoop MapReduce

### Task #1 - Minimum Temperature Per Capital

- Inicialmente é necessário configurar e dar um nome à app/job.
- Criámos o objeto *SparkContext* utilizando as configurações criadas anteriormente

```
conf = SparkConf().setMaster("local").setAppName("MinimumTemperaturePerCapital")
sc = SparkContext(conf=conf)
```

- De seguida, lemos o dataset, utilizando o *SparkContext*.

- Fazemos o split das linhas pela “,” e obtemos as colunas que necessitamos para esta task: *weather station*, *observation\_type* e a *temperature*
- Obtemos um RDD que resulta da função *parseLine(line)*, é um key-value RDD que neste caso é (*Station*, *ob\_type*,*temp*) (*weather station*, *observation\_type* e *temperature*)

```
def parseLine(line):
    fields = line.split(',')
    Station = str(fields[0])
    ob_type = str(fields[2])
    temp = float(fields[3]) / 10
    return (Station, ob_type,temp)

lines = sc.textFile("1800.csv")

rdd = lines.map(parseLine)
```

- Fazemos um filtro no key-value RDD(*Station*, *ob\_type*,*temp*), para obtermos apenas dados correspondentes a temperaturas mínimas

```
filter_tmin = rdd.filter(lambda x: x[1] == 'TMIN')
```

- Fazemos um mapeamento do RDD filtrado para obter apenas os valores (*Station*,*temp*), dado que o *ob\_type* já foi utilizado na filtragem.
- Seguidamente, usamos a função *reduceByKey* para uma agregação dos valores para cada key. Utilizamos a função *min()* porque queremos obter a temperatura mínima de cada key (*Station*)

```
reduction = filter_tmin.map(lambda x: (x[0],x[2]))

minbystation = reduction.reduceByKey(lambda x,y: min(x,y))
```

- Output:

```
('ITE00100554', -14.8)
('EZE00100082', -13.5)
```

## Task #2 - Obtain the Word Frequency in a Book

- Para esta task, o método de configuração é o mesmo que foi utilizado na task 1
- Seguidamente, aplicamos a função *flatMap()* para conseguirmos obter as palavras individuais, utilizando também um split com uma expressão regular
- Mapeamos os dados no formato (palavra,1) e implementámos um *reduceByKey()* que vai agrupar os values de cada key. Neste caso, é aplicada uma soma aos values para assim, obtermos o número de vezes que cada palavra se repete no dataset

```
count_words = lines.flatMap(lambda x: x.split(" "))\
    .map(lambda x : (x,1))\
    .reduceByKey(lambda x,y:x+y)
```

- Output: Como o output desta task é enorme, seguidamente está apenas apresentado uma pequena parte do output. Como podemos ver pelas imagens, as palavras contém algumas “,” e “.”, assim sendo, concluímos que a expressão regular não foi a melhor escolhida,

```
('weris', 2)
('This', 4)
('eBook', 2)
('is', 13)
('for', 20)
('the', 212)
('use', 1)
('anyone', 1)
('anywhere', 1)
('at', 40)
('no', 10)
('cost', 1)
('and', 140)
('with', 38)
('almost', 1)
('restrictions', 1)
('whatsoever', 1)
('You', 1)
('may', 3)
('copy', 1)
('it', 5)
('give', 5)
('it', 22)
('away', 4)
('on', 6)
('re-use', 1)
```

```
('updated', 1)
('May', 1)
('3', 1)
('2012', 1)
('Language', 1)
('English', 1)
('***', 2)
('START', 1)
('OF', 1)
('THIS', 1)
('PROJECT', 1)
('GUTENBERG', 1)
('EBOOK', 1)
('THE', 2)
('INVISIBLE', 1)
('MAN', 1)
('Produced', 1)
('Andrew', 1)
('Sly', 1)
('CHAPTER', 2)
('I', 19)
('STRANGE', 1)
('MAN'S', 1)
('ABBRIAL', 1)
```

### Task #3 - Sort the Word Frequency in a Book

- Para esta task foi utilizado o mesmo código que na task 2, a única diferença, é que acrescentou-se uma linha para ser possível mostrar os RDDs ordenados (utilizou-se a função `sortBy()`)

```
results = count_words.sortBy(lambda x: [x[1], x[0]]).collect()
```

- Output: Neste output os dados estão ordenados, em que as primeiras linhas apresentam as palavras que foram menos pesquisadas e as últimas linhas, as palavras mais pesquisadas

```
3 ('?', 1)
4 ('ARRIVAL', 1)
5 ('Accidents', 1)
6 ('After', 1)
7 ('Although', 1)
8 ('Altogether', 1)
9 ('Andrew', 1)
10 ('Author', 1)
11 ('Being', 1)
12 ('Bramblehurst.', 1)
13 ('Chirk', 1)
14 ('Date', 1)
15 ('EBOOK', 1)
16 ('EBOOK', 1)
17 ('English', 1)
18 ('Everything', 1)
19 ('Evidently', 1)
20 ('FIRST', 1)
21 ('February', 1)
22 ('GUTENBERG', 1)
23 ('Gleeson's', 1)
24 ('HENFREY'S', 1)
25 ('Hastings', 1)
26 ('He's', 1)
```

```
('on', 15)
('so', 16)
('up', 16)
('He', 18)
('it', 18)
('for', 20)
('Hall', 21)
('She', 21)
('as', 22)
('it', 22)
('Mrs.', 23)
('said', 36)
('her', 37)
('had', 38)
('with', 38)
('at', 40)
('that', 42)
('in', 53)
('she', 57)
('was', 66)
('he', 67)
('of', 76)
('his', 83)
('to', 90)
('a', 103)
('!', 116)
('and', 140)
('the', 212)
```

### Task #4 - Obtain the Total Amount Spent by Customer

- Nesta task foi utilizado o mesmo método que na task 1: foi feito um `parseLine(line)` das colunas mais importantes: (`customer_id`, `amount`)
- De seguida, fizemos um `reduceByKey`, para obtermos a quantidade de `amount` que cada `customer` gastou

```
def parseLine(line):
    fields = line.split(',')
    customer_id = str(fields[0])
    amount = float(fields[2])
    return (customer_id, amount)

lines = sc.textFile("customer-orders.csv")
rdd = lines.map(parseLine)
```

```
totalsbycustomer = rdd.reduceByKey(lambda x,y :x+y)
```

- Output: Como o output é muito extenso, apenas está representado uma parte do output

```
('44', 4756.8899999999985)
('35', 5155.4199999999999)
('2', 5994.59)
('47', 4316.2999999999999)
('29', 5032.5299999999999)
('91', 4642.2599999999999)
('70', 5368.2499999999999)
('85', 5503.43)
('53', 4945.2999999999999)
('14', 4735.0300000000001)
('51', 4975.22)
('42', 5696.8400000000003)
('79', 3790.5700000000001)
('50', 4517.27)
('20', 4836.8599999999999)
('15', 5413.5100000000001)
('5', 4561.0699999999999)
('48', 4384.33)
('31', 4765.05)
('4', 4815.0500000000002)
('36', 4278.0499999999997)
('57', 4628.4)
('12', 4664.5899999999998)
('22', 5019.4499999999999)
('54', 6065.3899999999999)
('0', 5524.9499999999998)
('88', 4830.5499999999999)
```

## Task #5 - Sort the Total Amount Spent by Customer

- Para esta task foi utilizado o código da task 4, acrescentou-se apenas uma linha de código para fazer o sort dos values

```
results = totalsbycustomer.sortBy(lambda x: [x[1], x[0]]).collect()
```

- Output: Neste output os dados estão ordenados, em que as primeiras linhas apresentam os costumers com menos amount e nas últimas linhas os costumers com mais amount. O output como é extenso não está todo apresentado

```
('44', 4756.8899999999985)
('35', 5155.4199999999999)
('2', 5994.59)
('47', 4316.2999999999999)
('29', 5032.5299999999999)
('91', 4642.2599999999999)
('70', 5368.2499999999999)
('85', 5503.43)
('53', 4945.2999999999999)
('14', 4735.0300000000001)
('51', 4975.22)
('42', 5696.8400000000003)
('79', 3790.5700000000001)
('50', 4517.27)
('20', 4836.8599999999999)
('15', 5413.5100000000001)
('5', 4561.0699999999999)
('48', 4384.33)
('31', 4765.05)
('4', 4815.0500000000002)
('36', 4278.0499999999997)
('57', 4628.4)
```

## Assignment #3 - Data Processing with Spark SQL

### Task #1 – Minimum Temperature Per Capital

- Criar a Spark Session e fazer load dos dados e criar um RDD

```
#Create a SparkSession
spark = SparkSession.builder.appName("MinimumTemperaturePerCapitalSQL").getOrCreate()
```

```
lines = spark.sparkContext.textFile("1800.csv")
```

- Criar objetos Row() utilizando uma função map(), com as colunas importantes

```
def mapper(line):
    fields = line.split(',')
    return Row(city = str(fields[0]), observation_type=str(fields[2]), temperature = float(fields[3])/10)
```

- Criar um dataframe, transformando os row objects RDD
- Filtrar o dataframe para obter apenas dados correspondentes a temperaturas mínimas
- Agrupar por cidade e obter a menor temperatura

```
#Filter the schema and register the DataFrame as a table
schemaTemperature = spark.createDataFrame(temp).cache()

mintemp = schemaTemperature.filter(schemaTemperature.observation_type == 'TMIN')
mintemp.groupBy("city")\
    .agg(func.round(func.min("temperature"),2).alias("min_temperature")).show()
spark.stop()
```

- Output:

```
+-----+-----+
|      city|min_temperature|
+-----+-----+
|ITE00100554|      -14.8|
|EZE00100082|      -13.5|
+-----+-----+
```

## Task #2 – Obtain the Word Frequency in a Book

- Para obter o dataframe, inicialmente fizemos um split das linhas para obter os RDDs (palavras únicas). Seguidamente para ser possível converter em dataframe foi necessário converter o RDD numa lista de (key,values)

```
lines = lines.flatMap(lambda x: x.split(" "))\
               .map(lambda x: (x,0))

data = []
for i in lines.collect():
    data.append(i)

df_schema= spark.createDataFrame(data).toDF("word","aux_value")
```

- Output:

```
-----+-----+
| word|count_words|
-----+-----+
| online|      1|
| biting|      1|
| crest|      1|
| few|      2|
| deftly|      1|
| still|      5|
| eggs,|      1|
| some|      5|
| ...|      3|
| sir-one|      1|
| hope|      1|
| filled|      1|
| taking|      1|
| laughing|      1|
| undo.|      1|
| added,|      1|
| worked|      1|
| conversation|      1|
| so!"|      1|
| Title:|      1|
-----+-----+
only showing top 20 rows
```

## Task #3 – Sort the Word Frequency in a Book

- Para ordenar a frequência com que as palavras aparecem no dataset, foi acrescentada a seguinte linha no código da task 2

```
count_qt_words.sort("count_words").show()
```

- Output:

```
-----+-----+
| word|count_words|
-----+-----+
| appliances,|      1|
| station,|      1|
| approached|      1|
| biting|      1|
| muffler,|      1|
| afraid,|      1|
| Sty|      1|
| drowsy|      1|
| sir-one|      1|
| shut|      1|
| filled|      1|
| slow|      1|
| laughing|      1|
| utmost|      1|
| added,|      1|
| thick|      1|
| conversation|      1|
| leave|      1|
| Title:|      1|
| If|      1|
-----+-----+
only showing top 20 rows
```

## Task #4 – Obtain the Total Amount Spent by Customer

- Criamos um objeto Row () com as colunas importantes: *customerID* e o *amount*

```
def mapper(line):
    fields = line.split(',')
    return Row(customerID = int(fields[0]),amount = float(fields[2]))
```

- Seguidamente agrupámos por *customerID* e fizemos a soma do amount

```
schemaAmount.groupBy("customerID")\
  .agg(func.round(func.sum("amount"),2).alias("amount_sum")).show()
spark.stop()
```

- Output:

customerID	amount_sum
29	5032.53
26	5250.4
65	5140.35
54	6065.39
19	5059.43
0	5524.95
22	5019.45
7	4755.07
77	4327.73
34	5330.8
50	4517.27
94	4475.57
57	4628.4
43	5368.83
32	5496.05
84	4652.94
31	4765.05
98	4297.26
39	6193.11
25	5057.61

#### Task #4 – Sort the Total Amount Spent by Customer

- Utilizámos o código anterior e acrescentámos um linha para ordenar por *amount\_sum*

```
total_amount.sort("amount_sum").show()
```

- Output:

customerID	amount_sum
45	3309.38
79	3790.57
96	3924.23
23	4042.65
99	4172.29
75	4178.5
36	4278.05
98	4297.26
47	4316.3
77	4327.73
13	4367.62
48	4384.33
49	4394.6
94	4475.57
67	4505.79
50	4517.27
78	4524.51
5	4561.07
57	4628.4
83	4635.8

only showing top 20 rows