Project #2

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)
```

Output:

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

```
24 "41" 268.55555555554
25 "42" 383.5
26 "43" 230.57142857142858
27 "44" 282.166666666666
28 "45" 380.53846153846155
30 "47" 223.69238769239768
30 "47" 233.22222222223
31 "48" 281.4
281.4
281.4
281.4
381.52222222222223
32 "58" 254.6
33 "58" 254.6
33 "58" 254.6
33 "58" 254.6
36 "52" 380.63636363636
37 "58" 275.871428571428571
38 "55" 380.6363636363636
39 "56" 278.07869230769231
38 "55" 295.53846153846155
39 "56" 386.666666666666666
40 "57" 258.833333333333
41 "58" 116.94545454545455
42 "59" 220.0
43 "68" 202.71428571428572
44 "61" 256.222222222222
45 "62" 220.76923076923077
46 "63" 384.0
47 "64" 281.333333333333
48 "65" 298.2
49 "66" 276.44444444444446
50 "67" 214.625
51 "68" 269.6
```

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
"GM000010962" 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
"beigan." 1
"beigan." 1
"being" 6
"believe" 1
"below" 2
"besides" 1
"between" 1
"bit" 4
"bit" 1
"bite" 1
"bites" 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

 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,

```
('This', 4)
('eBook', 2)
('is', 13)
('for', 29)
('the', 212)
('use', 1)
('anyone', 1)
('anyone', 1)
('anyone', 1)
('ant, 40)
('ost', 1)
('and', 140)
('with', 38)
('almost', 1)
('restrictions', 1)
('whatsoever.', 1)
('you', 1)
('san', 3)
('copy', 1)
('it,', 5)
('give', 5)
('it', 22)
('away', 4)
('or', 6)
('re-use', 1)
```

```
('May', 1)
('3,', 1)
('3,', 1)
('language:', 1)
('tengish', 1)
('ere', 2)
('START', 1)
('O'', 1)
('HIS', 1)
('PROJECT', 1)
('GUTENBERG', 1)
('BOOK', 1)
('HON', 1)
('HON', 1)
('PROJECT', 1)
('MAN', 1)
('Produced', 1)
('Andrew', 1)
('SIOW', 1)
('CAPTEP', 2)
('T, 19)
('STRANGE', 1)
('MAN'S', 1)
```

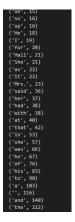
Task #3 - Sort the Word Frequency in a Book

 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

```
5 ('7,', 1)
4 ('ARRIVAL', 1)
5 ('Accidents,', 1)
6 ('After', 1)
7 ('Although', 1)
8 ('Altogether', 1)
9 ('Andrew', 1)
10 ('Being', 1)
11 ('Being', 1)
12 ('Bramblehurst.', 1)
13 ('Chirk,', 1)
14 ('Beook', 1)
15 ('EBook', 1)
16 ('EBook', 1)
17 ('Espoit,' 1)
18 ('Everything', 1)
19 ('FIRST', 1)
19 ('FIRST', 1)
10 ('Gleson's", 1)
11 ('Gleson's", 1)
12 ('Gutentay', 1)
13 ('Gleson's", 1)
14 ('HenrRev's', 1)
15 ('Hastings', 1)
16 ('He's', 1)
```



Task #4 - Obtain the Total Amount Spent by Customer

- Nesta task foi utilizado o mesmo método que ma task 1: foi feito um *parseLine(line)* das colunas mais importantes: (costumer_id,amount)
- De seguida, fizemos um reduceByKey, para obtermos a quantidade de amout que cada costumer gastou

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

lines = sc.textFile("customer-orders.csv")

rdd = lines.map(parseLine)
```

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

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

```
('44', 4756.88999999999)
('25', 594.59)
('27', 594.59)
('27', 594.59)
('29', 5032.5299999999)
('29', 5032.5299999999)
('70', 4642.2599999999)
('70', 5368.2499999999)
('85', 5503.43)
('35', 4945.2999999999)
('14', 4735.830000000001)
('51', 4975.22)
('42', 5696.84000000001)
('50', 4517.27)
('20', 4836.8599999999)
('15', 5413.5100000000001)
('55', 4561.06999999999)
('15', 4562.06)
('44', 4815.0500000000002)
('36', 4278.04999999999)
('48', 4384.433)
('31', 4765.05)
('44', 4664.84)
('12', 4664.83999999999)
('54', 5619.44999999999)
('54', 5619.44999999999)
('54', 56524.94999999999)
('88', 4830.549999999999)
```

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 = totalsbycostumer.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 costumer com mais amount. O output como é extenso não

está todo apresentado

('35', 5155.41999999999)
('2', 5994.59)
('2', 5994.59)
('47', 4316.2999999999)
('91', 4642.2599999999)
('78', 5368.2499999999)
('85', 5503.43)
('53', 4945.2999999999)
('14', 4735.03000000001)
('51', 4975.22)
('42', 5696.84000000003)
('79', 3790.570000000001)
('56', 4517.27)
('26', 4836.8599999999)
('15', 5413.510000000001)
('5', 4561.0699999999)
('48', 4384.33)
('31', 4765.05)
('4', 4815.050000000002)
('36', 4278.049999999997)
('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 SpartSession
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

Output:

Task #2 - Obtain the Word Frequency in a Book

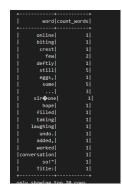
Para obter o dataframe, inicialmente fizemos um split das linhas para obter os RDDs (palavras únicas).
 Seguidamente para ser possível converter em dataframe foi necessário converter o RDD numa lista de (key,values)

```
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:



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:

Task #4 - Obtain the Total Amount Spent by Customer

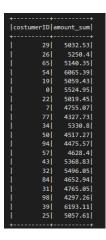
• Criamos um objeto Row () com as colunas importantes: costumerID e o amount

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

• Seguidamente agrupámos por costumerID e fizemos a soma do amount

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

Output:



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:

```
|costumerID|amount_sum|
                3309.38
                3924.23
                4042.65
         99|
75|
                4172.29
                4178.5
                4278.05
         47|
77|
13|
48|
               4327.73
                 4394.6
         94
                4475.57
         67|
50|
                4505.79
                4517.27
         78|
5|
                4524.51
                 4628.4
                 4635.8
only showing top 20 rows
```