Exercise1 TABAAI_AYOUB & BOUTHER_OUMAIMA

January 2, 2022

[1]:
```
# Import all the necessary packages
import os
import findspark
findspark.init()
import pyspark
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession, functions as F
from statistics import mean
spark = SparkSession.builder.appName("Spark_project").getOrCreate()
```

In this Exercise we will use the 'WholeTextFiles' method to upload the data from our local machine into our notebook. This method takes as input the path to the folder that contains the text files and returns a key-value pairs, where the key is the path of each file and the value is the content of each file.

[2]:
```
sc = spark.sparkContext
path = r"C:\Users\acer\Desktop\M2_DS\Big data Frameworks"
files = "input"
dirPath = os.path.join(path,files)
data = sc.wholeTextFiles(dirPath)
textfiles = data.collect()
```

[3]:
```
data.take(3)
```

[3]:
```
[('file:/C:/Users/acer/Desktop/M2_DS/Big data Frameworks/input/anger.txt',
  'JAN 13\r\nFEB 12\r\nMAR 14\r\nAPR 15\r\nMAY 12\r\nJUN 15\r\nJUL 19\r\nAUG
15\r\nSEP 13\r\nOCT 8\r\nNOV 14\r\nDEC 16'),
 ('file:/C:/Users/acer/Desktop/M2_DS/Big data Frameworks/input/lyon.txt',
  'JAN 13\r\nFEB 12\r\nMAR 14\r\nAPR 15\r\nMAY 12\r\nJUN 15\r\nJUL 19\r\nAUG
25\r\nSEP 13\r\nOCT 11\r\nNOV 22\r\nDEC 22'),
 ('file:/C:/Users/acer/Desktop/M2_DS/Big data
Frameworks/input/marseilles_1.txt',
  'JAN 21\r\nFEB 21\r\nMAR 21\r\nAPR 27\r\nMAY 25\r\nJUN 25\r\nJUL 21\r\nAUG
22\r\nSEP 23\r\nOCT 28\r\nNOV 24\r\nDEC 26')]
```

Now we can made one single rdd or Dataframe from all the files that contains the information about the city, the store, the month and the income. To do this we will create the function ***toRDD()*** that maps each record in the files as tuple of respective informations previouselly mentioned.

[4]:
```
def toRDD(files):
    rdd = []
    for d in files:
        pairs = d[1].split('\r\n')
        store = d[0].split('/')[-1][:-4]
        city = store.split('_')[0]
        for p in pairs:
            rdd.append((city,store,p[:3],int(p[4:])))
    return sc.parallelize(rdd)
```

[5]:
```
# Lets create the rdd for our data files
# This table will contain the following infos:
# the city, the store, the month, the income.
rdd = toRDD(textfiles)
rdd.take(5)
```

[5]:
```
[('anger', 'anger', 'JAN', 13),
 ('anger', 'anger', 'FEB', 12),
 ('anger', 'anger', 'MAR', 14),
 ('anger', 'anger', 'APR', 15),
 ('anger', 'anger', 'MAY', 12)]
```

[6]:
```
# Let's create our datafram from the rdd created
schema = ["City","Store","Month","Income"]
dataframe = spark.createDataFrame(rdd, schema)
dataframe.show(5)
```

```
+-----+-----+-----+------+
| City|Store|Month|Income|
+-----+-----+-----+------+
|anger|anger|  JAN|    13|
|anger|anger|  FEB|    12|
|anger|anger|  MAR|    14|
|anger|anger|  APR|    15|
|anger|anger|  MAY|    12|
+-----+-----+-----+------+
only showing top 5 rows
```

0.0.1 1.Average monthly income of the shop in France

To have the average income of the shop in France, we **group by** month and **we caculate the average income** for each month on all the cities in FRANCE. We can add an **order by** to order by average income.

[7]:
```
average_rdd = rdd.map(lambda x:(x[2],
                                x[3])).groupByKey().mapValues(mean).sortBy(lambda x:
 ↪x[1])
```

```
average_rdd.collect()
```

[7]:
```
[('MAR', 17.53846153846154),
 ('FEB', 19.153846153846153),
 ('APR', 20.23076923076923),
 ('JAN', 20.76923076923077),
 ('JUL', 21.692307692307693),
 ('MAY', 22.46153846153846),
 ('AUG', 23.076923076923077),
 ('NOV', 24.53846153846154),
 ('SEP', 25.53846153846154),
 ('OCT', 26.53846153846154),
 ('JUN', 27.846153846153847),
 ('DEC', 29)]
```

[8]:
```
average_frame = dataframe.groupBy('Month').agg(
    F.avg('Income').alias('average_income')).orderBy('average_income',␣
 ↪ascending = True)
average_frame.show(12)
```

```
+-----+------------------+
|Month|    average_income|
+-----+------------------+
|  MAR| 17.53846153846154|
|  FEB|19.153846153846153|
|  APR| 20.23076923076923|
|  JAN| 20.76923076923077|
|  JUL|21.692307692307693|
|  MAY| 22.46153846153846|
|  AUG|23.076923076923077|
|  NOV| 24.53846153846154|
|  SEP| 25.53846153846154|
|  OCT| 26.53846153846154|
|  JUN|27.846153846153847|
|  DEC|              29.0|
+-----+------------------+
```

0.0.2 2. Average monthly income of the shop in each city

For both the rdd and the dataframe, we **group by the city and the month** and then **we calculate the average on each tuple**. We added a **sort by** to see the order.

[9]:
```
average_city_month_rdd = rdd.map(lambda x:((x[2],x[0]),
                            x[3])).groupByKey().mapValues(mean).sortBy(lambda x:
 ↪x[1])
average_city_month_rdd.take(10)
```

[9]:
```
[(('OCT', 'orlean'), 8),
 (('OCT', 'anger'), 8),
 (('APR', 'rennes'), 9),
 (('APR', 'nice'), 9),
 (('MAR', 'rennes'), 10),
 (('AUG', 'nantes'), 11),
 (('MAY', 'rennes'), 11),
 (('AUG', 'rennes'), 11),
 (('MAR', 'troyes'), 11),
 (('NOV', 'troyes'), 11)]
```

[10]:
```
average_city_month_frame = dataframe.groupBy('Month','City').agg(F.
                avg('Income').alias('average_income')).
 ↪orderBy('average_income',ascending = True)
average_city_month_frame.show(10)
```

```
+-----+--------+--------------+
|Month|    City|average_income|
+-----+--------+--------------+
|  OCT|   anger|           8.0|
|  OCT|  orlean|           8.0|
|  APR|    nice|           9.0|
|  APR|  rennes|           9.0|
|  MAR|  rennes|          10.0|
|  AUG|toulouse|          11.0|
|  NOV|  troyes|          11.0|
|  DEC|  troyes|          11.0|
|  APR|toulouse|          11.0|
|  AUG|  nantes|          11.0|
+-----+--------+--------------+
only showing top 10 rows
```

0.0.3 3. Total revenue per city per year

To have the total revenue for each city per year, we **group by the city** and **we calculate the same** over the cities.

[11]:
```
total_city_rdd = rdd.map(lambda x:(x[0],x[3])).groupByKey().mapValues(sum).
 ↪sortBy(lambda x:x[1])
total_city_rdd.take(10)
```

[11]:
```
[('anger', 166),
 ('toulouse', 177),
 ('rennes', 180),
 ('lyon', 193),
 ('orlean', 196),
 ('nice', 203),
```

```
 ('nantes', 207),
 ('troyes', 214),
 ('marseilles', 515),
 ('paris', 1568)]
```

[12]:
```
total_city_frame = dataframe.groupBy('City').agg(F.
                    sum('Income').alias('Total_income')).
 ↪orderBy('Total_income',ascending = True)
total_city_frame.show(10)
```

```
+----------+------------+
|      City|Total_income|
+----------+------------+
|     anger|         166|
|  toulouse|         177|
|    rennes|         180|
|      lyon|         193|
|    orlean|         196|
|      nice|         203|
|    nantes|         207|
|    troyes|         214|
|marseilles|         515|
|     paris|        1568|
+----------+------------+
```

0.0.4 4. Total revenue per store per year

As the previous question, **we group by the store** and then **we calculate the sum** over all the stores.

[13]:
```
total_store_rdd = rdd.map(lambda x:(x[1],x[3])).groupByKey().mapValues(sum).
 ↪sortBy(lambda
                                                                    x:
 ↪x[1])
total_store_rdd.collect()
```

[13]:
```
[('anger', 166),
 ('toulouse', 177),
 ('rennes', 180),
 ('lyon', 193),
 ('orlean', 196),
 ('nice', 203),
 ('nantes', 207),
 ('troyes', 214),
 ('marseilles_2', 231),
 ('marseilles_1', 284),
 ('paris_3', 330),
```

```
 ('paris_1', 596),
 ('paris_2', 642)]
```

[14]:
```
total_store_frame = dataframe.groupBy('Store').agg(F.
        sum('Income').alias('Total_income_year')).
 ↪orderBy('Total_income_year',ascending = True)
total_store_frame.show(13)
```

```
+-----------+-----------------+
|      Store|Total_income_year|
+-----------+-----------------+
|      anger|              166|
|   toulouse|              177|
|     rennes|              180|
|       lyon|              193|
|     orlean|              196|
|       nice|              203|
|     nantes|              207|
|     troyes|              214|
|marseilles_2|              231|
|marseilles_1|              284|
|    paris_3|              330|
|    paris_1|              596|
|    paris_2|              642|
+-----------+-----------------+
```

0.0.5 5.The store that achieves the best performance in each month

To have the store that achives the store that achieves the best performance, we need to have the best income in each month to pick all the stores that achived that income in that month. Thus, we create a RDD or DataFrame that contains **for each month the best (maximum) income** and **we use a left join on the income and the month where the left table is the best performane per month and the right one is the whole table**. Finally, we will get a table that contains all the stores per month that achieved the best performance (maximum income)

[15]:
```
# We create rdd with the income, the month and the store
# we will see why we group the month and the income
mon_city_income_rdd = rdd.map(lambda x:((x[3],x[2]),x[1]))
mon_city_income_rdd.take(10)
```

[15]:
```
[((13, 'JAN'), 'anger'),
 ((12, 'FEB'), 'anger'),
 ((14, 'MAR'), 'anger'),
 ((15, 'APR'), 'anger'),
 ((12, 'MAY'), 'anger'),
 ((15, 'JUN'), 'anger'),
 ((19, 'JUL'), 'anger'),
```

```
 ((15, 'AUG'), 'anger'),
 ((13, 'SEP'), 'anger'),
 ((8, 'OCT'), 'anger')]
```

[16]:
```
# We group the best income in each month as a key and empty value to made a␣
 ↪left join on the best income and the month.

best_performance_income_rdd = rdd.map(lambda x:(x[2],
        x[3])).groupByKey().mapValues(max).map(lambda x:((x[1],x[0]),''))
best_performance_income_rdd.collect()
```

[16]:
```
[((51, 'JAN'), ''),
 ((61, 'JUL'), ''),
 ((63, 'SEP'), ''),
 ((68, 'OCT'), ''),
 ((71, 'DEC'), ''),
 ((57, 'APR'), ''),
 ((72, 'MAY'), ''),
 ((45, 'AUG'), ''),
 ((42, 'FEB'), ''),
 ((44, 'MAR'), ''),
 ((85, 'JUN'), ''),
 ((64, 'NOV'), '')]
```

[17]:
```
# No we made the left join on the best income per month and the month so that␣
 ↪we get the store that has achieved
#the best performance in each month, and for each month we list the stores that␣
 ↪has achieved the best performance as a list.
best_performance_achieved_rdd = best_performance_income_rdd.
 ↪leftOuterJoin(mon_city_income_rdd).map(lambda x:
                            (x[0][1],x[1][1])).groupByKey().map(lambda x:(x[0],␣
 ↪list(x[1])))
print('The stores with the best performance per month are :...')
best_performance_achieved_rdd.collect()
```

```
The stores with the best performance per month are :…
```

[17]:
```
[('JUL', ['paris_1']),
 ('OCT', ['paris_1']),
 ('APR', ['paris_1']),
 ('AUG', ['paris_2']),
 ('NOV', ['paris_2']),
 ('SEP', ['paris_2']),
 ('DEC', ['paris_1']),
 ('JAN', ['paris_1']),
 ('MAY', ['paris_2']),
 ('FEB', ['paris_2']),
```

```
 ('JUN', ['paris_2']),
 ('MAR', ['paris_2'])]
```

The treatment for the DataFrame would be the same as the **RDD**

[18]:
```
best_income_frame = dataframe.groupBy('Month').agg(F.max('Income').
 ↪alias('Revenue'))
best_income_frame = best_income_frame.withColumnRenamed('Month','M')
best_income_frame.show()
```

```
+---+-------+
|  M|Revenue|
+---+-------+
|APR|     57|
|OCT|     68|
|NOV|     64|
|FEB|     42|
|SEP|     63|
|JAN|     51|
|AUG|     45|
|MAR|     44|
|DEC|     71|
|JUN|     85|
|JUL|     61|
|MAY|     72|
+---+-------+
```

[19]:
```
best_performance_frame = best_income_frame.join(dataframe,
        (best_income_frame.Revenue == dataframe.Income)&(best_income_frame.M ==␣
 ↪dataframe.Month),
    how='left_outer').select('Month','Income','Store').
 ↪withColumnRenamed('Income','best_income')

print('The stores with the best performance per month are :...')
best_performance_frame.show()
```

```
The stores with the best performance per month are :…
+-----+-----------+-------+
|Month|best_income|  Store|
+-----+-----------+-------+
|  DEC|         71|paris_1|
|  APR|         57|paris_1|
|  FEB|         42|paris_2|
|  JUN|         85|paris_2|
|  JUL|         61|paris_1|
|  MAY|         72|paris_2|
|  AUG|         45|paris_2|
|  OCT|         68|paris_1|
```

```
|  MAR|         44|paris_2|
|  SEP|         63|paris_2|
|  NOV|         64|paris_2|
|  JAN|         51|paris_1|
+-----+-----------+-------+
```