SPARK TP1 Soufiane MOUTEI

October 10, 2018

1 Let's start with initializing Spark:

```
In [1]: from pyspark import SparkContext
        sc = SparkContext()

In [2]: sc

Out[2]: <SparkContext master=local[*] appName=pyspark-shell>
```

2 Average monthly income of the shop in France:

```
In [3]: # A function to get: city, month, income, 1; it will be used for mapping
        def f(x):
            key = x[0]
            line = x[1].split()
            return key, line[0], int(line[1]), 1

In [4]: whole_file_using_stores = (
            # read the text files of the directory "input1"
            sc.wholeTextFiles("input1/")

            # Only the key is changed here: we split on "/" to get the textfile name
            # and then delete the file extension
            .map(lambda x: (x[0].split("/")[-1][:-4], x[1]))

            # Split the values to get each line on the file linked to the city name
            .flatMapValues(lambda x: x.split("\r\n"))

            # Map the function f to get: city, month, income, 1. 1 will be used for averaging
            .map(f)

        )

        # Take only the city name
        whole_file = whole_file_using_stores.map(
            lambda x: (x[0].split("_")[0], x[1], x[2], x[3])
        )
```

```
In [5]: # Answer to the question
        monthly_income_france = (
            whole_file

            # To get the key-value pair: month, (income, 1)
            .map(lambda x: (x[1], (x[2], x[3])))

            # reduce by key: sum of the revenues of the same month, and of "1"s
            .reduceByKey(lambda a, b: (a[0] + b[0], a[1] + b[1]))

            # Map each key to: total_income / n where n is the number of stores
            .mapValues(lambda x: x[0] / x[1])
        )

In [6]: monthly_income_france.collect()

Out[6]: [('APR', 20.23076923076923),
         ('MAY', 22.46153846153846),
         ('AUG', 23.076923076923077),
         ('JAN', 20.76923076923077),
         ('FEB', 19.153846153846153),
         ('MAR', 17.53846153846154),
         ('JUN', 27.846153846153847),
         ('JUL', 21.692307692307693),
         ('SEP', 25.53846153846154),
         ('OCT', 26.53846153846154),
         ('NOV', 24.53846153846154),
         ('DEC', 29.0)]
```

3 Average monthly income of the shop in each city:

```
In [7]: monthly_income_per_city = (
            whole_file

            # To get the key-value pair: (city, month), (income, 1)
            .map(lambda x: ((x[0], x[1]), (x[2], x[3])))

            # reduce by key: sum of the revenues of the same month, and of "1"s
            .reduceByKey(lambda a, b: (a[0] + b[0], a[1] + b[1]))

            # Map each key to: total_income / n
            .mapValues(lambda x: x[0] / x[1])

            # Change the key-value pair to: city, (month, average_monthly_income)
            .map(lambda x: (x[0][0], (x[0][1], x[1])))

            # Sort by key to get the values of each city together
```

```
            .sortByKey()
        )
```

In [8]:
```
monthly_income_per_city.collect()
```

Out[8]:
```
[('anger', ('APR', 15.0)),
 ('anger', ('MAY', 12.0)),
 ('anger', ('AUG', 15.0)),
 ('anger', ('JAN', 13.0)),
 ('anger', ('FEB', 12.0)),
 ('anger', ('MAR', 14.0)),
 ('anger', ('JUN', 15.0)),
 ('anger', ('JUL', 19.0)),
 ('anger', ('SEP', 13.0)),
 ('anger', ('OCT', 8.0)),
 ('anger', ('NOV', 14.0)),
 ('anger', ('DEC', 16.0)),
 ('lyon', ('APR', 15.0)),
 ('lyon', ('MAY', 12.0)),
 ('lyon', ('AUG', 25.0)),
 ('lyon', ('JAN', 13.0)),
 ('lyon', ('FEB', 12.0)),
 ('lyon', ('MAR', 14.0)),
 ('lyon', ('JUN', 15.0)),
 ('lyon', ('JUL', 19.0)),
 ('lyon', ('SEP', 13.0)),
 ('lyon', ('OCT', 11.0)),
 ('lyon', ('NOV', 22.0)),
 ('lyon', ('DEC', 22.0)),
 ('marseilles', ('JAN', 16.0)),
 ('marseilles', ('FEB', 16.0)),
 ('marseilles', ('MAR', 16.0)),
 ('marseilles', ('JUN', 25.0)),
 ('marseilles', ('JUL', 21.0)),
 ('marseilles', ('SEP', 23.0)),
 ('marseilles', ('OCT', 28.0)),
 ('marseilles', ('NOV', 24.0)),
 ('marseilles', ('DEC', 26.0)),
 ('marseilles', ('APR', 22.0)),
 ('marseilles', ('MAY', 18.5)),
 ('marseilles', ('AUG', 22.0)),
 ('nantes', ('JAN', 16.0)),
 ('nantes', ('FEB', 15.0)),
 ('nantes', ('MAR', 20.0)),
 ('nantes', ('JUN', 28.0)),
 ('nantes', ('JUL', 19.0)),
 ('nantes', ('SEP', 13.0)),
 ('nantes', ('OCT', 14.0)),
```

```
('nantes', ('NOV', 14.0)),
('nantes', ('DEC', 24.0)),
('nantes', ('APR', 12.0)),
('nantes', ('MAY', 21.0)),
('nantes', ('AUG', 11.0)),
('nice', ('APR', 9.0)),
('nice', ('MAY', 11.0)),
('nice', ('AUG', 11.0)),
('nice', ('JAN', 16.0)),
('nice', ('FEB', 15.0)),
('nice', ('MAR', 20.0)),
('nice', ('JUN', 18.0)),
('nice', ('JUL', 19.0)),
('nice', ('SEP', 23.0)),
('nice', ('OCT', 18.0)),
('nice', ('NOV', 14.0)),
('nice', ('DEC', 29.0)),
('orlean', ('JAN', 13.0)),
('orlean', ('FEB', 12.0)),
('orlean', ('MAR', 14.0)),
('orlean', ('JUN', 15.0)),
('orlean', ('JUL', 19.0)),
('orlean', ('SEP', 13.0)),
('orlean', ('OCT', 8.0)),
('orlean', ('NOV', 24.0)),
('orlean', ('DEC', 26.0)),
('orlean', ('APR', 15.0)),
('orlean', ('MAY', 12.0)),
('orlean', ('AUG', 25.0)),
('paris', ('APR', 38.666666666666664)),
('paris', ('MAY', 50.0)),
('paris', ('AUG', 41.666666666666664)),
('paris', ('JAN', 38.333333333333336)),
('paris', ('FEB', 33.0)),
('paris', ('MAR', 26.333333333333332)),
('paris', ('JUN', 55.0)),
('paris', ('JUL', 33.666666666666664)),
('paris', ('SEP', 48.0)),
('paris', ('OCT', 56.666666666666664)),
('paris', ('NOV', 48.666666666666664)),
('paris', ('DEC', 52.666666666666664)),
('rennes', ('JAN', 19.0)),
('rennes', ('FEB', 18.0)),
('rennes', ('MAR', 10.0)),
('rennes', ('JUN', 13.0)),
('rennes', ('JUL', 14.0)),
('rennes', ('SEP', 23.0)),
('rennes', ('OCT', 18.0)),
```

```
('rennes', ('NOV', 14.0)),
('rennes', ('DEC', 20.0)),
('rennes', ('APR', 9.0)),
('rennes', ('MAY', 11.0)),
('rennes', ('AUG', 11.0)),
('toulouse', ('JAN', 12.0)),
('toulouse', ('FEB', 13.0)),
('toulouse', ('MAR', 14.0)),
('toulouse', ('JUN', 18.0)),
('toulouse', ('JUL', 19.0)),
('toulouse', ('SEP', 23.0)),
('toulouse', ('OCT', 14.0)),
('toulouse', ('NOV', 12.0)),
('toulouse', ('DEC', 19.0)),
('toulouse', ('APR', 11.0)),
('toulouse', ('MAY', 11.0)),
('toulouse', ('AUG', 11.0)),
('troyes', ('APR', 17.0)),
('troyes', ('MAY', 15.0)),
('troyes', ('AUG', 22.0)),
('troyes', ('JAN', 21.0)),
('troyes', ('FEB', 21.0)),
('troyes', ('MAR', 11.0)),
('troyes', ('JUN', 25.0)),
('troyes', ('JUL', 11.0)),
('troyes', ('SEP', 21.0)),
('troyes', ('OCT', 28.0)),
('troyes', ('NOV', 11.0)),
('troyes', ('DEC', 11.0))]
```

4 Total revenue per city per year:

```
In [9]: yearly_income_city = (
            whole_file

            # To get the key-value pair: city, income
            .map(lambda x: (x[0], x[2]))

            # reduce by key: sum of the revenues
            .reduceByKey(lambda a, b: a + b)
        )

In [10]: yearly_income_city.collect()

Out[10]: [('troyes', 214),
          ('nice', 203),
          ('paris', 1568),
          ('anger', 166),
```

```
 ('lyon', 193),
 ('marseilles', 515),
 ('rennes', 180),
 ('orlean', 196),
 ('nantes', 207),
 ('toulouse', 177)]
```

5 Total revenue per store per year:

```
In [11]: # Here we need to change our structure
         yearly_income_store = (
             whole_file_using_stores

             # To get the key-value pair: store, income
             .map(lambda x: (x[0], x[2]))

             # reduce by key: sum of the revenues
             .reduceByKey(lambda a, b: a + b)

         )

In [12]: yearly_income_store.collect()

Out[12]: [('troyes', 214),
          ('marseilles_1', 284),
          ('nice', 203),
          ('paris_2', 642),
          ('anger', 166),
          ('paris_3', 330),
          ('lyon', 193),
          ('marseilles_2', 231),
          ('rennes', 180),
          ('orlean', 196),
          ('nantes', 207),
          ('paris_1', 596),
          ('toulouse', 177)]
```

6 The store that achieves the best performance in each month:

```
In [13]: # Method 1: Using sortBy
         best_performance_store = (
             whole_file_using_stores

             # To get the key-value pair: month, (store, income)
             .map(lambda x: (x[1], (x[0], x[2])))

             # Sort by (month, -income) to get the values sorted by the month,
```

```
        # and in case we have the same month, we sort in a descending order the income
        .sortBy(lambda x: (x[0], - x[1][1]))

        # we get only the first value that is the store with maximum revenue
        .reduceByKey(lambda x, y: x)

        # map the key to the store only since we don't care about the income
        .mapValues(lambda x: x[0])
    )
```

```
In [14]: best_performance_store.collect()
```

```
Out[14]: [('APR', 'paris_1'),
          ('AUG', 'paris_2'),
          ('MAY', 'paris_2'),
          ('DEC', 'paris_1'),
          ('FEB', 'paris_2'),
          ('JAN', 'paris_1'),
          ('JUL', 'paris_1'),
          ('JUN', 'paris_2'),
          ('MAR', 'paris_2'),
          ('NOV', 'paris_2'),
          ('OCT', 'paris_1'),
          ('SEP', 'paris_2')]
```

```
In [15]: # Method 2: Without sortBy
         best_performance_store_2 = (
             whole_file_using_stores

             # To get the key-value pair: month, (store, income)
             .map(lambda x: (x[1], (x[0], x[2])))

             # reduce by key: if the revenue of the 1st is bigger, we return x, otherwise, y
             .reduceByKey(lambda x, y: (x if x[1] > y[1] else y))

             # map the key to the store only since we don't care about the income
             .mapValues(lambda x: x[0])
         )
```

```
In [16]: best_performance_store_2.collect()
```

```
Out[16]: [('APR', 'paris_1'),
          ('MAY', 'paris_2'),
          ('AUG', 'paris_2'),
          ('JAN', 'paris_1'),
          ('FEB', 'paris_2'),
          ('MAR', 'paris_2'),
          ('JUN', 'paris_2'),
          ('JUL', 'paris_1'),
```

```
('SEP', 'paris_2'),
('OCT', 'paris_1'),
('NOV', 'paris_2'),
('DEC', 'paris_1')]
```