Age Group Sales Dominance

Age Group Sales Dominance

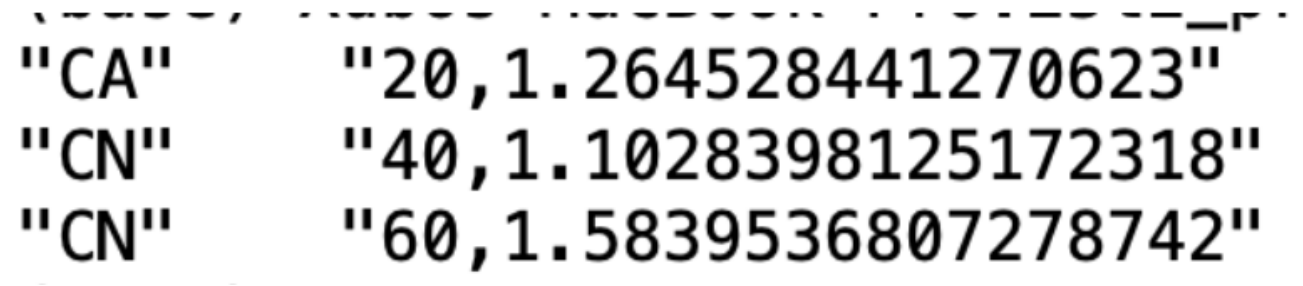
Background: Segment-dominated spending refers to situations where a particular demographic group—such as an age bracket, gender, or region—contributes disproportionately to total sales. Rather than consumer spending being evenly distributed, one segment often stands out as the primary revenue driver. Identifying these dominant segments enables businesses to optimize marketing strategies, personalize offerings, and allocate resources more effectively by focusing on the groups that generate the most value.

Problem Definition: You are given an E-commerce website logs data (*sales.csv*) created to practice exploratory data analysis. Each record in the dataset consists of a country code, user's language, user's age, and the sales value. A sample input file (*sample.txt*) has been provided. Your task is to utilize MRJob to find out the top-k age-sales for each country based on the following steps:

- For each country, calculate the country average sales (**country_avg_sales**).
- For each country, calculate the average sales in each age group (**country_age_avg_sales**).
- For each country, calculate the age dominance score (**ads**) for each age group where **ads** = **country_age_avg_sales / country_avg_sales**
- Report the age groups in each country whose **ads** value is not smaller than a given threshold **τ**.
- Records with invalid age column values should be ignored when computing average sales.
- You don't need to round anything during the computation.

An **age group** refers to a 10-year interval defined by rounding down the age to the nearest multiple of 10. For example, ages 0–9 are in group **0**, 10–19 in group **10**, 20–29 in group **20**, and so on.

Output Format: The output should contain three fields: the country code, the age group, and the ads value, in the format of `<the country code>**\t** <the age group>, < the ads value>`. The results should be sorted by country code alphabetically first and then by age group value in ascending order. Given the sample input file and the threshold **τ**=1.1, the result should be like:



Code Format: The code template has been provided. Your code should take three parameters: the input file, the output folder on HDFS, and the threshold value k. We will also use more than 1 reducer to test your code. Assuming **τ**=1.2 and using 2 reducers, you need to use the command below to run your code:

```
$ python3 project1.py -r hadoop sample.txt -o hdfs_output --jobconf
myjob.settings.tau=1.2 --jobconf mapreduce.job.reduces=2
```

Note: You can access the value of τ in your program like `τ = jobconf_from_env('myjob.settings.tau')`, and you need to import `jobconf_from_env` by `from mrjob.compat import jobconf_from_env` (see the code template)

Marking Criteria

- You must complete this assignment based on MRjob and Hadoop. Submissions only contain regular Python techniques will be marked as 0.
- You cannot simply emit all key-value pairs from mappers and buffer them in memory on reducers to do the task, and such a method will receive no more than 4 marks
- Submissions that cannot be compiled and run on Hadoop in the Ed environment will receive no more than 4 marks
- Submissions can be compiled on ED and run on Hadoop. => +4
- All the difference values in the output are correct. =>+1
- The order in the output is correct. =>+1 (**Note:** You only need to guarantee the order within each reducer output)
- The output format is correct. => +1
- Submissions correctly implement the combiner or in-mapper combing. => +1
- Submissions correctly implement order inversion (i.e., using special keys). => +1
- Submissions correctly implement secondary sort. => +1
- Submissions can produce the correct result using one MRStep. => +1
- Submissions can produce the correct result with multiple reducers. => +1 (**Note:** You do not need to include 'mapreduce.job.reduces' in JOBCONF since the number of reducers will be received from the command)

Solution

```
import sys
import csv
import re
```

```
from mrjob.job import MRJob
from mrjob.step import MRStep
from mrjob.compat import jobconf_from_env


class proj1(MRJob):


    SORT_VALUES = True

    JOBCONF = {
      'mapreduce.map.output.key.field.separator':' ',
      'mapreduce.partition.keypartitioner.options':'-k1,1',

'mapreduce.job.output.key.comparator.class':'org.apache.hadoop.mapreduce.lib
.partition.KeyFieldBasedComparator',
      'stream.num.map.output.key.fields': '2',
    }



    def configure_args(self):
        super(proj1, self).configure_args()


    def steps(self):
        return [ MRStep(
            mapper=self.mapper,
            combiner=self.combiner_count,
            reducer_init=self.reducer_init,
            reducer=self.reducer,
            reducer_final=self.reducer_final
        ) ]

    def mapper(self, _, value):
        value = value.strip()
        if not value:
            return

        line = None
        try:
            row = next(csv.reader([value]))
            if len(row) == 4:
                line = row
        except:
            pass

        if not line:
            line = value.split(',')
            if len(line) != 4:
```

```
                return

        # country, _, age_str, sales_str = line
        country = line[0].strip().strip('"')

        try:
            age = int(line[2].strip('"'))
            sales = float(line[3].strip('"'))
        except:
            return

        age_group = age - (age % 10)
        # yield country, (sales, 1)
        yield (country, '000_ALL'), (sales, 1)
        yield (country, f'100_{age_group:03d}'), (sales, 1)

    def combiner_count(self, key, values):
        country, tag = key
        total = 0.0
        count = 0
        for s, c in values:
            total += s
            count += c
        yield key, (total, count)

    def reducer_init(self):
        self.tau = float(jobconf_from_env('myjob.settings.tau') or
self.options.tau)

        self.current_country = None
        self.country_avg = None
        self.age_records = []

    def reducer(self, key, values):
        country, tag = key
        total = 0.0
        count = 0
        for s, c in values:
            total += s
            count += c
        avg = total / count

        if self.current_country and self.current_country != country:

            for age_avg, group_avg in sorted(self.age_records, key=lambda x:
x[0]):
                ads = group_avg / self.country_avg
                if ads >= self.tau:
                    yield self.current_country, f"{age_avg},{ads}"
            self.country_avg = None
```

```
            self.age_records = []

        self.current_country = country
        if tag == '000_ALL':
            self.country_avg = avg
        else:
            age_avg = int(tag.split('_',1)[1])
            self.age_records.append((age_avg, avg))

    def reducer_final(self):

        if self.current_country:
            for age_avg, group_avg in sorted(self.age_records, key=lambda x:
x[0]):
                ads = group_avg / self.country_avg
                if ads >= self.tau:
                    yield self.current_country, f"{age_avg},{ads}"


if __name__ == '__main__':
    proj1.run()
```