Part I of Project of Spark

Student_ID:16307130214 Name:Liu Baichuan

I. Introduction

- Tasks of this part of Spark Project is using RDD operation to implement easy statistical work. In this part, I learn how to use RDD and use RDD to accomplish MR.
- My Project runs on Ubuntu16.04 + Spark2.4 + Hadoop2.7 as platform.
- My code firstly writes and runs on Ipython notebook, so it is more convenient for me to paste printscreens in this report. More important, it's my fault to forget to record my code. So in my commit files, you have to run my code in notebook line by line and do not run it as python script in pyspark. I also know if I want to use python script, I have to use SparkConf and SparkContext which are loaded automatically in Ipython.
- Potential application: Have the ability to use large scale distributed system.
- Spark is a great platform to process big data. According to its document[1], Apache Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming.

II. data preprocessing

Although data is provided as a sql file, but I still use normal plain text style to preprocess this dataset due to my unfamiliar to postgresql.

At first, I have to eliminate extra text. I find first 75 lines and last 49 lines are irrelevant information. I use command line: `tail -n +76 install.log > install.log.tmp`

[1] http://spark.apache.org/docs/latest/

to eliminate first 75 lines and tac 'file name' | sed '1,49d' | tac to delete last 49 lines.

However, the latter command line will create a new file as it is so slow. Finally, I delete last 49 lines manually.

Interestingly, after preprocessing the dateset, I find the number of valid citizens is 49,611,708, not 49,611,709. There is still a chance that something wrong happens during my process. I hope not.

III. Q and A

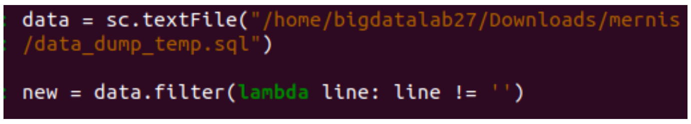
E1. 统计土尔其公民中所有人中年龄最大的男人

Algorithm:

- Transform time to standard datetime type
- Transform RDD data to data table in database
- Use SQL to find min datetime of men

Process:

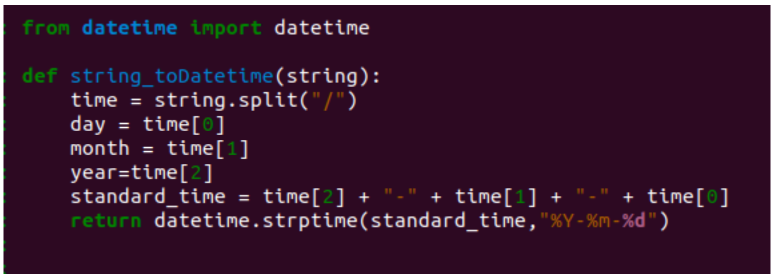
obtain data:



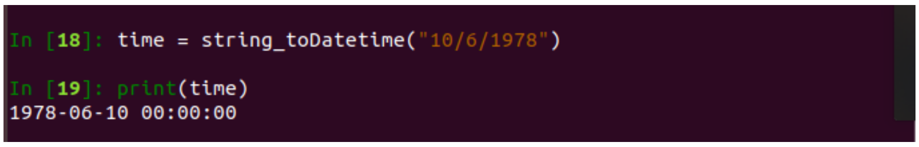
```
data = sc.textFile("/home/bigdatalab27/Downloads/mernis
/data_dump_temp.sql")

new = data.filter(lambda line: line != '')
```

The two lines will be used in every question, so in other questions, I won't show it again.

get standard datetime

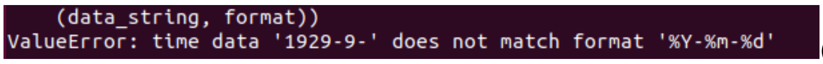


```
from datetime import datetime

def string_toDatetime(string):
    time = string.split("/")
    day = time[0]
    month = time[1]
    year=time[2]
    standard_time = time[2] + "-" + time[1] + "-" + time[0]
    return datetime.strptime(standard_time,"%Y-%m-%d")
```



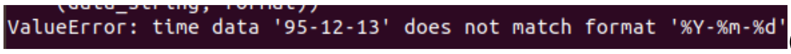
```
In [18]: time = string_toDatetime("10/6/1978")

In [19]: print(time)
1978-06-10 00:00:00
```

I use datetime type in python to transform text to datetime. However, when I try it on all date, there are some errors happening. All date has some irregularity and wrong date. For example:



```
    (data_string, format))
ValueError: time data '1929-9-' does not match format '%Y-%m-%d'
```

(lose day)

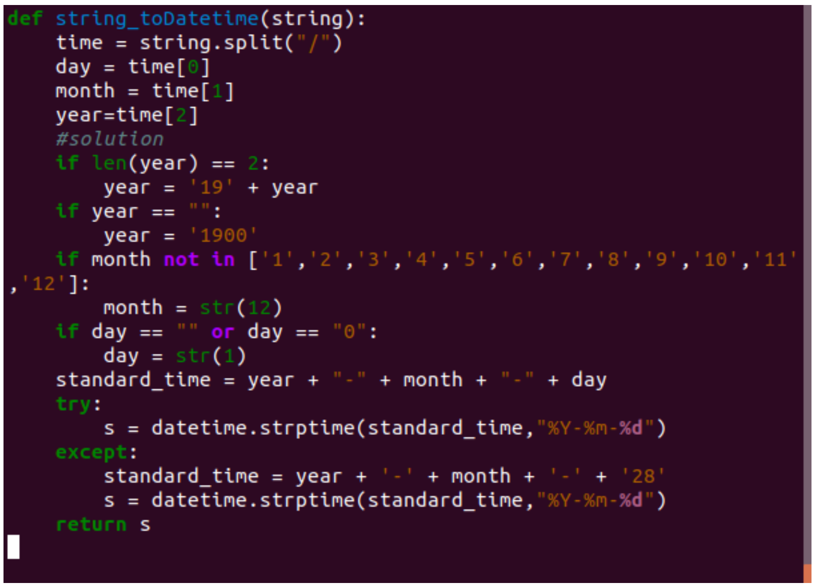


```
ValueError: time data '95-12-13' does not match format '%Y-%m-%d'
```

(irregularity year)

Also some wrong month like 13.

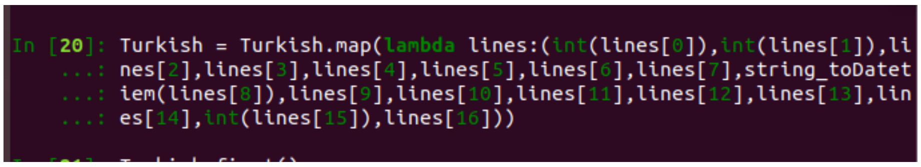
So I change my function to consider all situations: When year is 95, I change it to 1995. When month is invalid, I change it to 12. When day is missing, I add it as 1.



```
def string_toDatetime(string):
    time = string.split("/")
    day = time[0]
    month = time[1]
    year=time[2]
    #solution
    if len(year) == 2:
        year = '19' + year
    if year == "":
        year = '1900'
    if month not in ['1','2','3','4','5','6','7','8','9','10','11'
,'12']:
        month = str(12)
    if day == "" or day == "0":
        day = str(1)
    standard_time = year + "-" + month + "-" + day
    try:
        s = datetime.strptime(standard_time,"%Y-%m-%d")
    except:
        standard_time = year + '-' + month + '-' + '28'
        s = datetime.strptime(standard_time,"%Y-%m-%d")
    return s
```

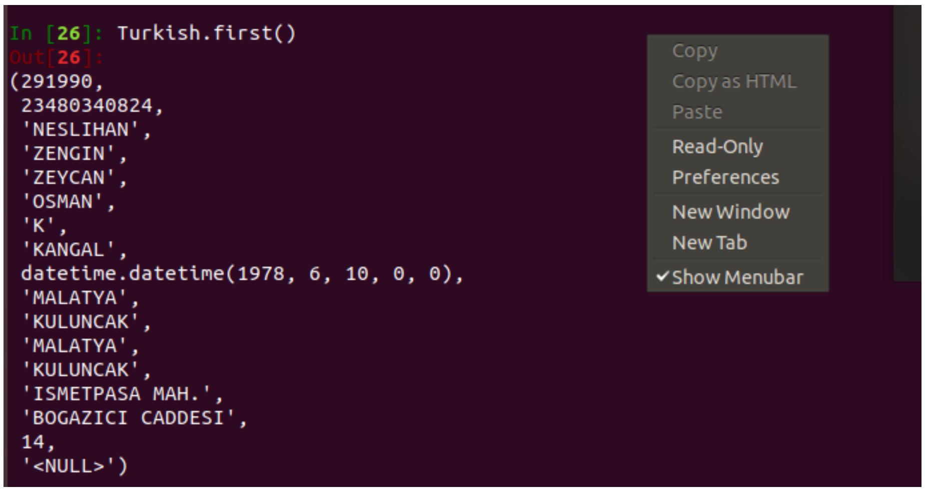
Then use this function on my RDD.(Some codes I forget to capture will be written manually)

Turkish = new.map(lambda line:line.split("\t"))



```
In [20]: Turkish = Turkish.map(lambda lines:(int(lines[0]),int(lines[1]),li
    ...: nes[2],lines[3],lines[4],lines[5],lines[6],lines[7],string_toDatet
    ...: iem(lines[8]),lines[9],lines[10],lines[11],lines[12],lines[13],lin
    ...: es[14],int(lines[15]),lines[16]))
```

#test data



```
In [26]: Turkish.first()
Out[26]:
(291990,
 23480340824,
 'NESLIHAN',
 'ZENGIN',
 'ZEYCAN',
 'OSMAN',
 'K',
 'KANGAL',
 datetime.datetime(1978, 6, 10, 0, 0),
 'MALATYA',
 'KULUNCAK',
 'MALATYA',
 'KULUNCAK',
 'ISMETPASA MAH.',
 'BOGAZICI CADDESI',
 14,
 '<NULL>')
```

#register data frame as a table in order to use sql

import pyspark.sql

list_dataframe =

sqlContext.createDataFrame(Turkish,['uid','national_identifier','first_name','last_name','mother_fir

st','father_first','gender','birth_city','date_of_birth','id_registration_city','

id_registration_district','address_city',' address_district','

address_neighborhood','stree_address','door_or_entrance_number','misc'])

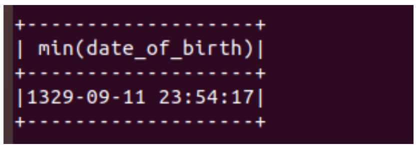
list_dataframe.registerTempTable("test")

top = sqlContext.sql('''SELECT MIN(date_time) FROM test WHERE gender = 'E'''')

#e stands for erkek(male) k stands for Kadin(female)

top.show()

Result:



```
+-------------------+
| min(date_of_birth)|
+-------------------+
|1329-09-11 23:54:17|
+-------------------+
```

To solve this question, I just use pysparksql. Construct table is the key part.

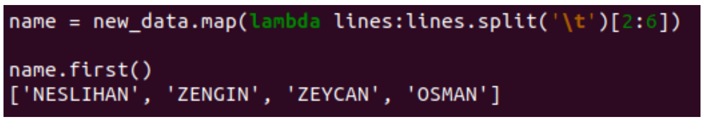
E2. 统计姓名中最长出现的字母

Algorithm：

- Get name date(first_name,last_name,mother_first,father_first)

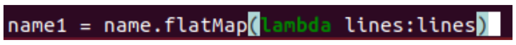
- Transform words to letters
- Create RDD pair (letter:1),count,sort

Process:

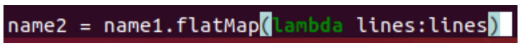


```
name = new_data.map(lambda lines:lines.split('\t')[2:6])

name.first()
['NESLIHAN', 'ZENGIN', 'ZEYCAN', 'OSMAN']
```

use flatMap to transform words to letters

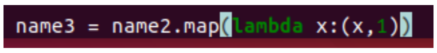


```
name1 = name.flatMap(lambda lines:lines)
```

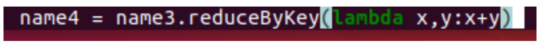


```
name2 = name1.flatMap(lambda lines:lines)
```

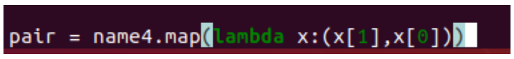
Create RDD pair, use reduceByKey operation to count and swap key and value in pair. Finally, use sortByKey to sort.



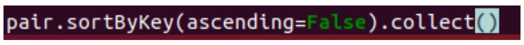
```
name3 = name2.map(lambda x:(x,1))
```



```
name4 = name3.reduceByKey(lambda x,y:x+y)
```

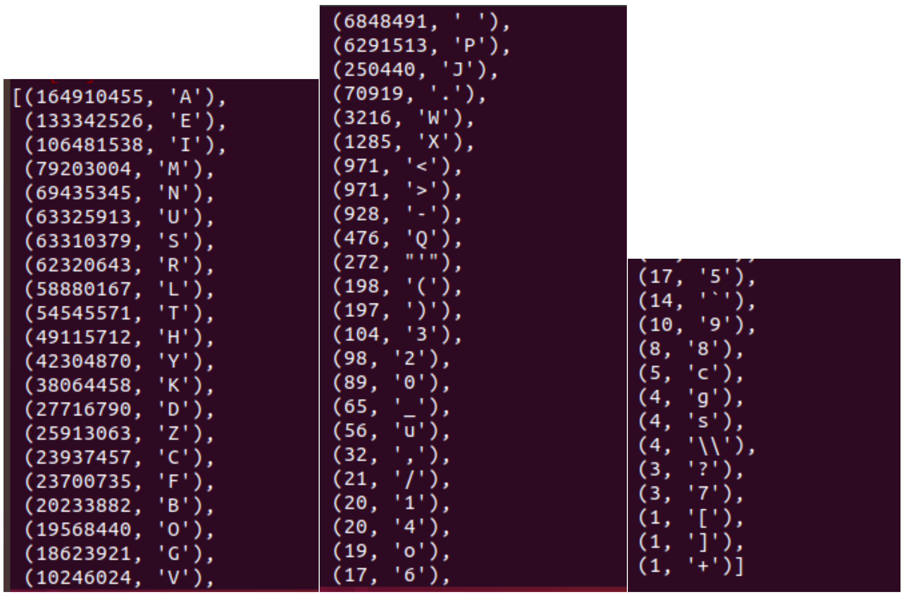


```
pair = name4.map(lambda x:(x[1],x[0]))
```



```
pair.sortByKey(ascending=False).collect()
```

Result:



```
[(164910455, 'A'),
 (133342526, 'E'),
 (106481538, 'I'),
 (79203004, 'M'),
 (69435345, 'N'),
 (63325913, 'U'),
 (63310379, 'S'),
 (62320643, 'R'),
 (58880167, 'L'),
 (54545571, 'T'),
 (49115712, 'H'),
 (42304870, 'Y'),
 (38064458, 'K'),
 (27716790, 'D'),
 (25913063, 'Z'),
 (23937457, 'C'),
 (23700735, 'F'),
 (20233882, 'B'),
 (19568440, 'O'),
 (18623921, 'G'),
 (10246024, 'V'),
```

```
 (6848491, ' '),
 (6291513, 'P'),
 (250440, 'J'),
 (70919, '.'),
 (3216, 'W'),
 (1285, 'X'),
 (971, '<'),
 (971, '>'),
 (928, '-'),
 (476, 'Q'),
 (272, "'"),
 (198, '('),
 (197, ')'),
 (104, '3'),
 (98, '2'),
 (89, '0'),
 (65, '_'),
 (56, 'u'),
 (32, ','),
 (21, '/'),
 (20, '1'),
 (20, '4'),
 (19, 'o'),
 (17, '6'),
```

```
 (17, '5'),
 (14, '`'),
 (10, '9'),
 (8, '8'),
 (5, 'c'),
 (4, 'g'),
 (4, 's'),
 (4, '\\'),
 (3, '?'),
 (3, '7'),
 (1, '['),
 (1, ']'),
 (1, '+')]
```

In this question, I use some basic RDD operations. We can find A is the most popular letter in Turkish. There are some strange characters, such as '/','\\'. There is a chance that Turkish use these characters in name besides dataset itself is not so clear.

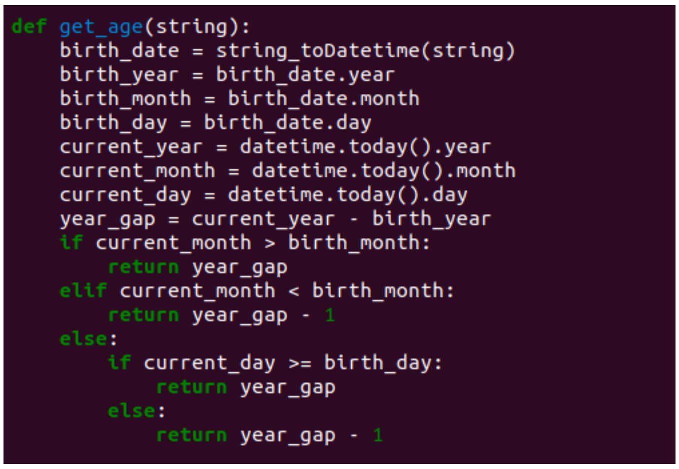
E3. 统计该国人的年龄分布，年龄段（0-18,19-28,29-38,39-48,49-59,>60）

Algorithm：

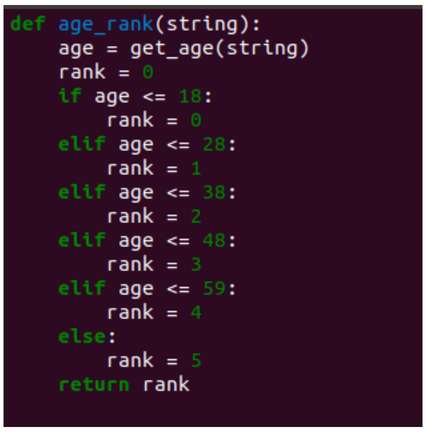
- Get accurate age and age_rank(0:0-18,1:19-28 and so on)
- Use countByValue directly to count

Process:

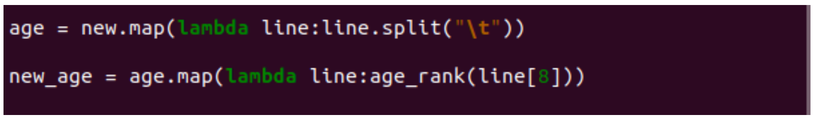
function for getting age_rank, use string_toDatetime function in E1.



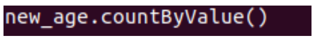
```
def get_age(string):
    birth_date = string_toDatetime(string)
    birth_year = birth_date.year
    birth_month = birth_date.month
    birth_day = birth_date.day
    current_year = datetime.today().year
    current_month = datetime.today().month
    current_day = datetime.today().day
    year_gap = current_year - birth_year
    if current_month > birth_month:
        return year_gap
    elif current_month < birth_month:
        return year_gap - 1
    else:
        if current_day >= birth_day:
            return year_gap
        else:
            return year_gap - 1
```



```
def age_rank(string):
    age = get_age(string)
    rank = 0
    if age <= 18:
        rank = 0
    elif age <= 28:
        rank = 1
    elif age <= 38:
        rank = 2
    elif age <= 48:
        rank = 3
    elif age <= 59:
        rank = 4
    else:
        rank = 5
    return rank
```

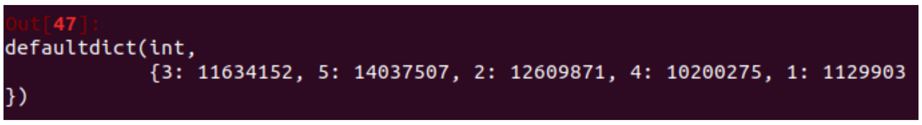


```
age = new.map(lambda line:line.split("\t"))

new_age = age.map(lambda line:age_rank(line[8]))
```



```
new_age.countByValue()
```

Result:



```
Out[47]:
defaultdict(int,
            {3: 11634152, 5: 14037507, 2: 12609871, 4: 10200275, 1: 1129903
})
```

It is interesting to find there is no 0-18 citizen. I think the reason may be this dataset is old. When I use current time(2019/5) to calculate their ages, mentioned situation happens. I also find the age distribution of Turkish is balanced.

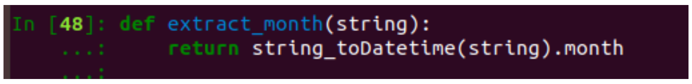
E4. 按月份，统计该国人的生日在每个月上的分布

Algorithm：

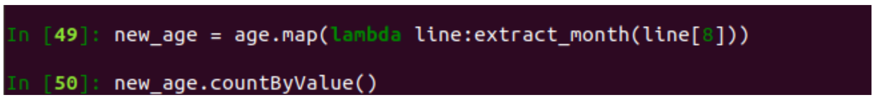
- Get accurate month
- Use countByValue directly to count

Process:

function for getting month, use string_toDatetime function in E1.

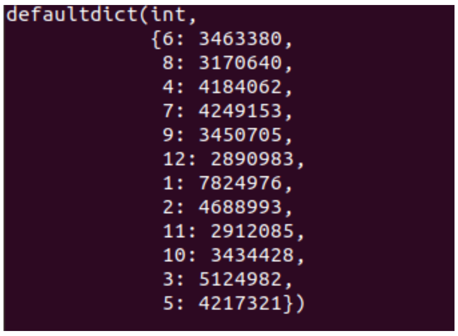


```
In [48]: def extract_month(string):
    ...:     return string_toDatetime(string).month
    ...:
```



```
In [49]: new_age = age.map(lambda line:extract_month(line[8]))

In [50]: new_age.countByValue()
```

Result:



```
defaultdict(int,
            {6: 3463380,
             8: 3170640,
             4: 4184062,
             7: 4249153,
             9: 3450705,
             12: 2890983,
             1: 7824976,
             2: 4688993,
             11: 2912085,
             10: 3434428,
             3: 5124982,
             5: 4217321})
```

We can find the number of births of January is relatively larger than others and the number of births of December is relatively smller than others. It can be deduced that February and March are the transition between winter and spring if we consider pregnant will take 10 months.

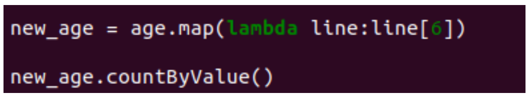
E4. 统计一下该国的男女比例，男女人数

Algorithm：

- Get directly gender information
- Use countByValue directly to count

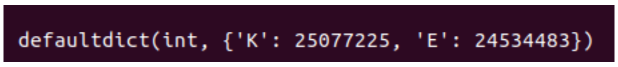
Process:

Gender field



```
new_age = age.map(lambda line:line[6])

new_age.countByValue()
```

Result:



```
defaultdict(int, {'K': 25077225, 'E': 24534483})
```

I also find the gender distribution of Turkish is balanced.

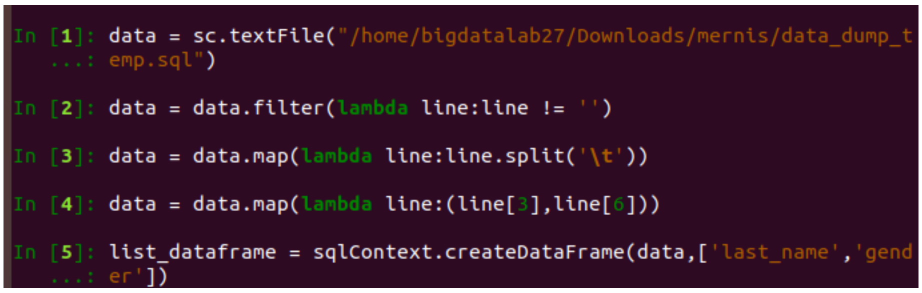
N1. 统计男性、女性中最常见的 10 个姓

Algorithm：

- Extract last_name and gender information
- Register a table
- Use sql GroupBy and OrderBy to find the most common last name respectively

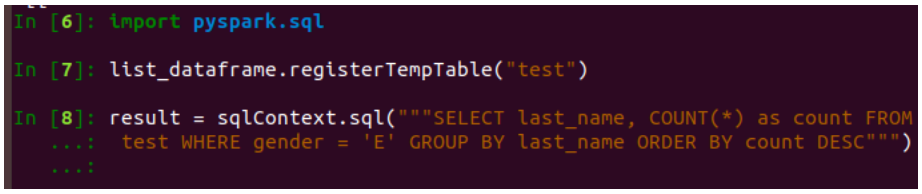
Process:

Last name and gender field

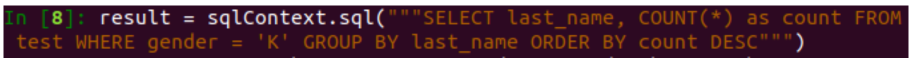


```
In [1]: data = sc.textFile("/home/bigdatalab27/Downloads/mernis/data_dump_t
   ...: emp.sql")

In [2]: data = data.filter(lambda line:line != '')

In [3]: data = data.map(lambda line:line.split('\t'))

In [4]: data = data.map(lambda line:(line[3],line[6]))

In [5]: list_dataframe = sqlContext.createDataFrame(data,['last_name','gend
   ...: er'])
```

Register table and use sql



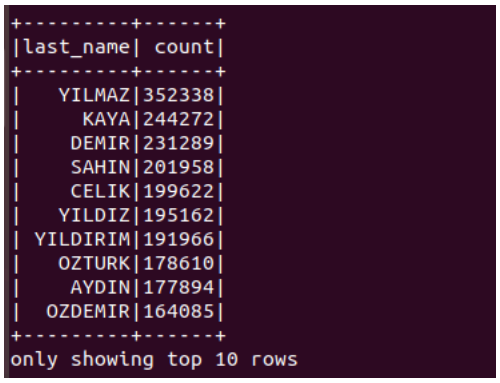
```
In [6]: import pyspark.sql

In [7]: list_dataframe.registerTempTable("test")

In [8]: result = sqlContext.sql("""SELECT last_name, COUNT(*) as count FROM
   ...:  test WHERE gender = 'E' GROUP BY last_name ORDER BY count DESC""")
   ...:
```



```
In [8]: result = sqlContext.sql("""SELECT last_name, COUNT(*) as count FROM
 test WHERE gender = 'K' GROUP BY last_name ORDER BY count DESC""")
```

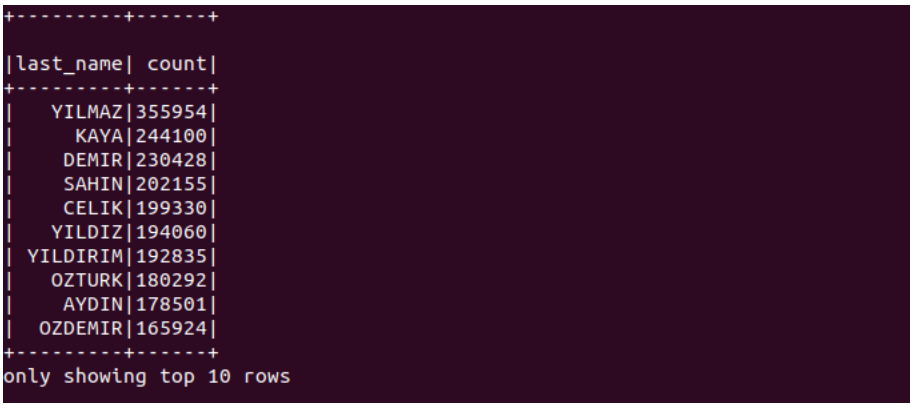
Result:

male



```
+---------+------+
|last_name| count|
+---------+------+
|   YILMAZ|352338|
|     KAYA|244272|
|    DEMIR|231289|
|    SAHIN|201958|
|    CELIK|199622|
|   YILDIZ|195162|
| YILDIRIM|191966|
|   OZTURK|178610|
|    AYDIN|177894|
|  OZDEMIR|164085|
+---------+------+
only showing top 10 rows
```

female



```
+---------+------+
|last_name| count|
+---------+------+
|   YILMAZ|355954|
|     KAYA|244100|
|    DEMIR|230428|
|    SAHIN|202155|
|    CELIK|199330|
|   YILDIZ|194060|
| YILDIRIM|192835|
|   OZTURK|180292|
|    AYDIN|178501|
|  OZDEMIR|165924|
+---------+------+
only showing top 10 rows
```

I can find top 10 last name in male and female is same.

YILMAZ,KAYA,DEMIR,SAHIN is to Turkish what zhao， qian， sun， li to China.

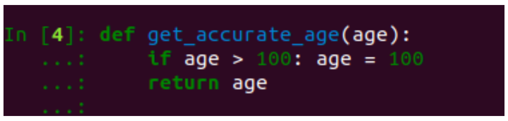
N2. 统计每个城市市民的平均年龄

Algorithm：

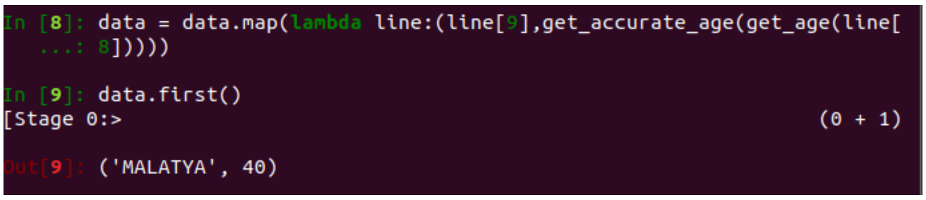
- Extract id_registration_city and age(use function I define before with little justification to solve age > 100)
- Use RDD pair operation combineByKey to calculate sum and count by key

Process:

id_registration_city and age field

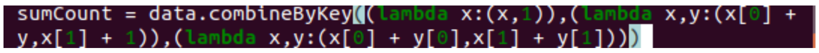


```
In [4]: def get_accurate_age(age):
   ...:     if age > 100: age = 100
   ...:     return age
   ...:
```

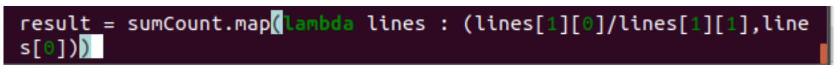


```
In [8]: data = data.map(lambda line:(line[9],get_accurate_age(get_age(line[
   ...: 8]))))

In [9]: data.first()
[Stage 0:>                                                          (0 + 1)
Out[9]: ('MALATYA', 40)
```

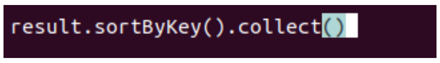
Use combineByKey



```
sumCount = data.combineByKey((lambda x:(x,1)),(lambda x,y:(x[0] +
y,x[1] + 1)),(lambda x,y:(x[0] + y[0],x[1] + y[1])))
```

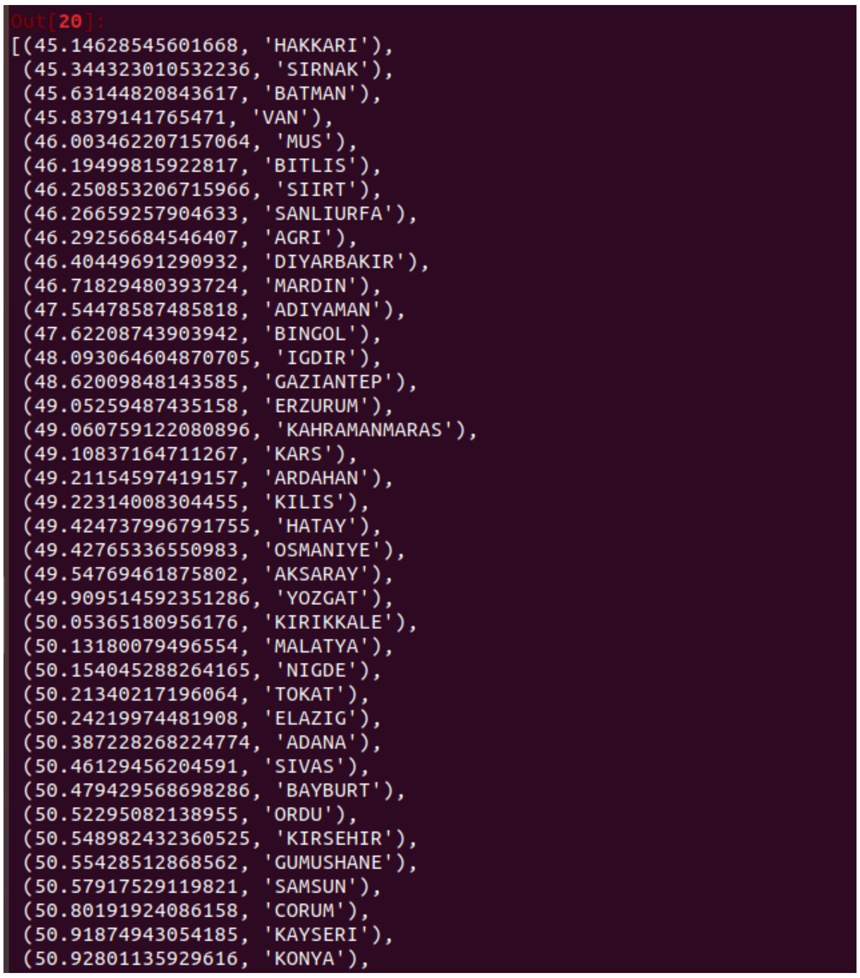


```
result = sumCount.map(lambda lines : (lines[1][0]/lines[1][1],line
s[0]))
```



```
result.sortByKey().collect()
```

Result:



```
Out[20]:
[(45.14628545601668, 'HAKKARI'),
 (45.344323010532236, 'SIRNAK'),
 (45.63144820843617, 'BATMAN'),
 (45.8379141765471, 'VAN'),
 (46.003462207157064, 'MUS'),
 (46.19499815922817, 'BITLIS'),
 (46.250853206715966, 'SIIRT'),
 (46.26659257904633, 'SANLIURFA'),
 (46.29256684546407, 'AGRI'),
 (46.40449691290932, 'DIYARBAKIR'),
 (46.71829480393724, 'MARDIN'),
 (47.54478587485818, 'ADIYAMAN'),
 (47.62208743903942, 'BINGOL'),
 (48.093064604870705, 'IGDIR'),
 (48.62009848143585, 'GAZIANTEP'),
 (49.05259487435158, 'ERZURUM'),
 (49.060759122080896, 'KAHRAMANMARAS'),
 (49.10837164711267, 'KARS'),
 (49.21154597419157, 'ARDAHAN'),
 (49.22314008304455, 'KILIS'),
 (49.424737996791755, 'HATAY'),
 (49.42765336550983, 'OSMANIYE'),
 (49.54769461875802, 'AKSARAY'),
 (49.909514592351286, 'YOZGAT'),
 (50.05365180956176, 'KIRIKKALE'),
 (50.13180079496554, 'MALATYA'),
 (50.154045288264165, 'NIGDE'),
 (50.21340217196064, 'TOKAT'),
 (50.24219974481908, 'ELAZIG'),
 (50.387228268224774, 'ADANA'),
 (50.46129456204591, 'SIVAS'),
 (50.479429568698286, 'BAYBURT'),
 (50.52295082138955, 'ORDU'),
 (50.548982432360525, 'KIRSEHIR'),
 (50.55428512868562, 'GUMUSHANE'),
 (50.57917529119821, 'SAMSUN'),
 (50.80191924086158, 'CORUM'),
 (50.91874943054185, 'KAYSERI'),
 (50.92801135929616, 'KONYA'),
```

I do not show all the answer for its length.

N3. 说出该国平均人就最年轻的五个城市

Use answer we get in N2.

HAKKARI,SIRNAK,BATMAN,VAN,MUS

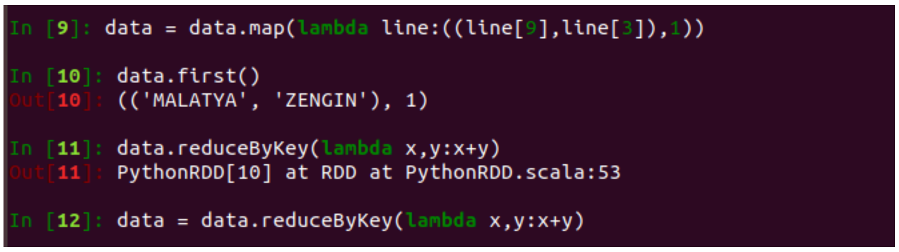
N4. 统计一下该国前 10 大人口城市中，每个城市的前 3 大姓氏

Algorithm：

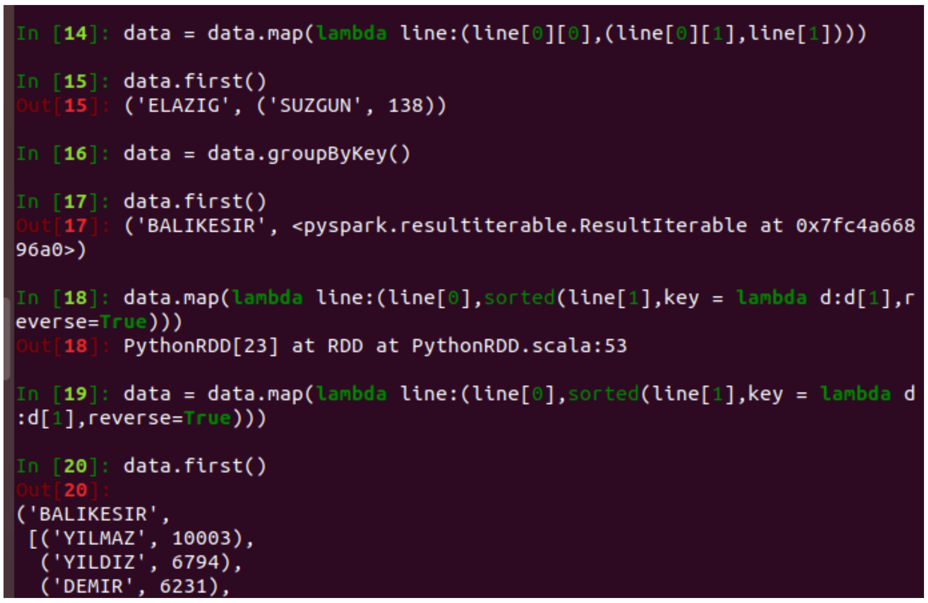
- Create RDD pair((city,last_name),1) and count by key
- Change RDD pair ((city,last_name),count) to (city,(last_name,count))
- Merge RDD pair by key and sort RDD pair by count in value
- Find top 10 population city
- Extract top3 last_name for top 10 population city

Process:

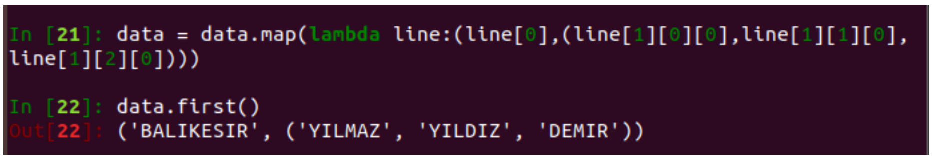
Get rdd (city,(last name,count))



```
In [9]: data = data.map(lambda line:((line[9],line[3]),1))

In [10]: data.first()
Out[10]: (('MALATYA', 'ZENGIN'), 1)

In [11]: data.reduceByKey(lambda x,y:x+y)
Out[11]: PythonRDD[10] at RDD at PythonRDD.scala:53

In [12]: data = data.reduceByKey(lambda x,y:x+y)
```

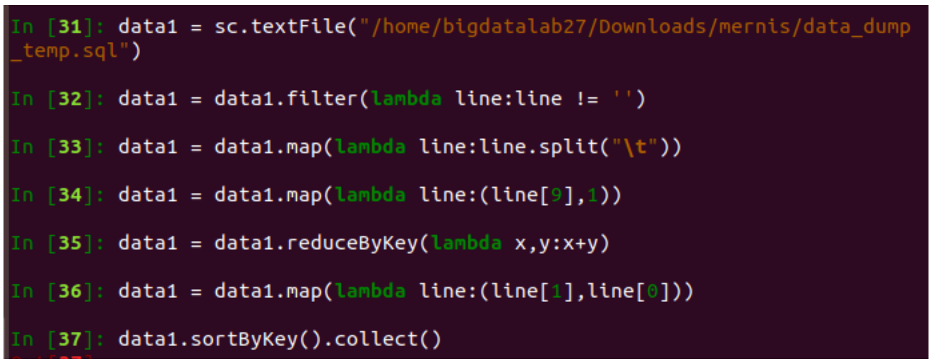


```
In [14]: data = data.map(lambda line:(line[0][0],(line[0][1],line[1])))

In [15]: data.first()
Out[15]: ('ELAZIG', ('SUZGUN', 138))

In [16]: data = data.groupByKey()

In [17]: data.first()
Out[17]: ('BALIKESIR', <pyspark.resultiterable.ResultIterable at 0x7fc4a668
96a0>)

In [18]: data.map(lambda line:(line[0],sorted(line[1],key = lambda d:d[1],r
everse=True)))
Out[18]: PythonRDD[23] at RDD at PythonRDD.scala:53

In [19]: data = data.map(lambda line:(line[0],sorted(line[1],key = lambda d
:d[1],reverse=True)))

In [20]: data.first()
Out[20]:
('BALIKESIR',
 [('YILMAZ', 10003),
  ('YILDIZ', 6794),
  ('DEMIR', 6231),
```

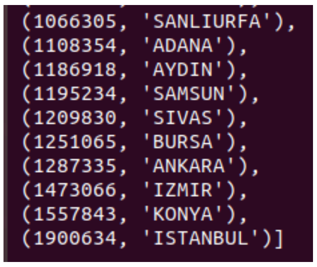


```
In [21]: data = data.map(lambda line:(line[0],(line[1][0][0],line[1][1][0],
line[1][2][0])))

In [22]: data.first()
Out[22]: ('BALIKESIR', ('YILMAZ', 'YILDIZ', 'DEMIR'))
```

Find top 10 city

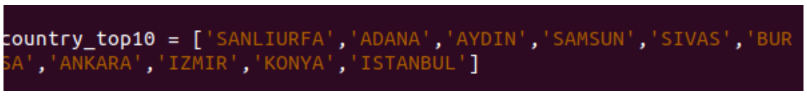


```
In [31]: data1 = sc.textFile("/home/bigdatalab27/Downloads/mernis/data_dump
_temp.sql")

In [32]: data1 = data1.filter(lambda line:line != '')

In [33]: data1 = data1.map(lambda line:line.split("\t"))

In [34]: data1 = data1.map(lambda line:(line[9],1))

In [35]: data1 = data1.reduceByKey(lambda x,y:x+y)

In [36]: data1 = data1.map(lambda line:(line[1],line[0]))

In [37]: data1.sortByKey().collect()
```

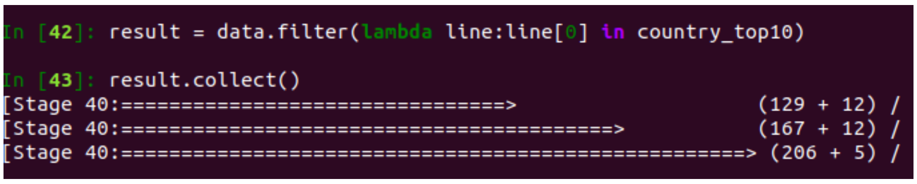


```
(1066305, 'SANLIURFA'),
(1108354, 'ADANA'),
(1186918, 'AYDIN'),
(1195234, 'SAMSUN'),
(1209830, 'SIVAS'),
(1251065, 'BURSA'),
(1287335, 'ANKARA'),
(1473066, 'IZMIR'),
(1557843, 'KONYA'),
(1900634, 'ISTANBUL')]
```

Only top 10 city need result

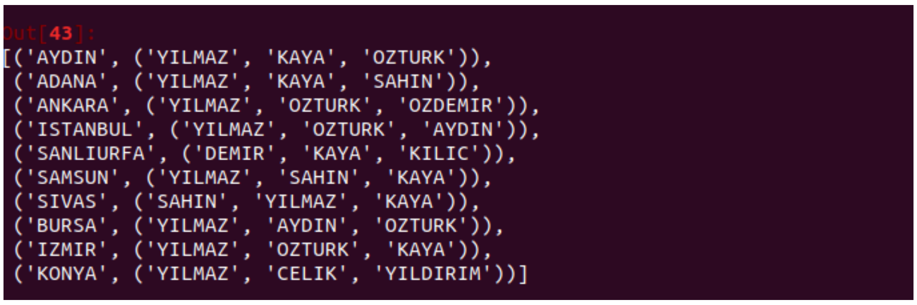


```
ountry_top10 = ['SANLIURFA','ADANA','AYDIN','SAMSUN','SIVAS','BUR
SA','ANKARA','IZMIR','KONYA','ISTANBUL']
```



```
n [42]: result = data.filter(lambda line:line[0] in country_top10)

n [43]: result.collect()
[Stage 40:================================>                  (129 + 12) /
[Stage 40:=========================================>         (167 + 12) /
[Stage 40:====================================================> (206 + 5) /
```

Result:



```
ut[43]:
[('AYDIN', ('YILMAZ', 'KAYA', 'OZTURK')),
 ('ADANA', ('YILMAZ', 'KAYA', 'SAHIN')),
 ('ANKARA', ('YILMAZ', 'OZTURK', 'OZDEMIR')),
 ('ISTANBUL', ('YILMAZ', 'OZTURK', 'AYDIN')),
 ('SANLIURFA', ('DEMIR', 'KAYA', 'KILIC')),
 ('SAMSUN', ('YILMAZ', 'SAHIN', 'KAYA')),
 ('SIVAS', ('SAHIN', 'YILMAZ', 'KAYA')),
 ('BURSA', ('YILMAZ', 'AYDIN', 'OZTURK')),
 ('IZMIR', ('YILMAZ', 'OZTURK', 'KAYA')),
 ('KONYA', ('YILMAZ', 'CELIK', 'YILDIRIM'))]
```

We can find only KILIC from SANLIURFA does not show up in most common last name refers to N1. Maybe we can infer The last name of Turkish citizens have little to do with the city

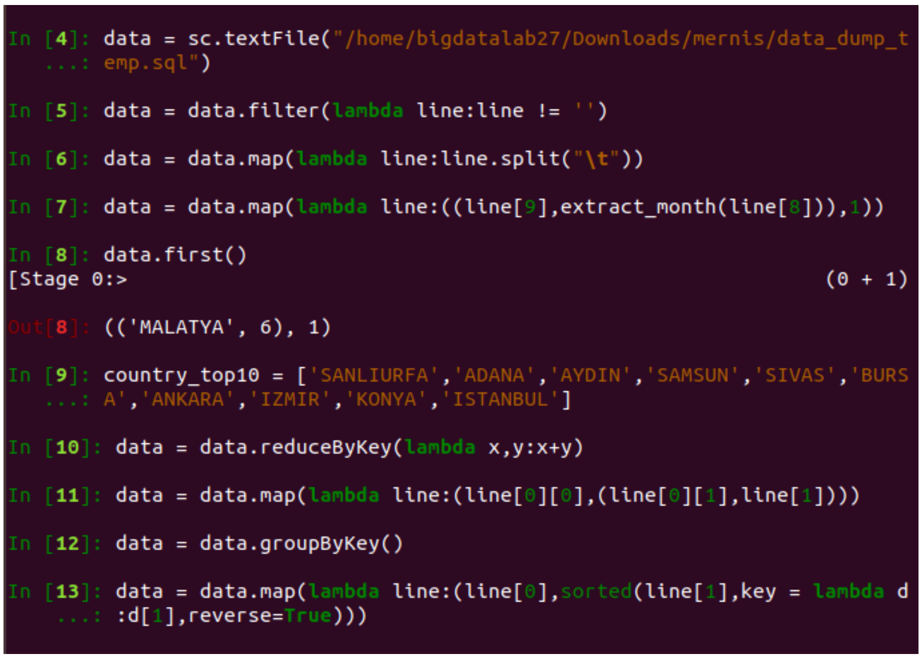
N5. 统计一下该国前 10 大人口城市中，每个城市的人口生日最集中分布的是哪两个月

Algorithm：

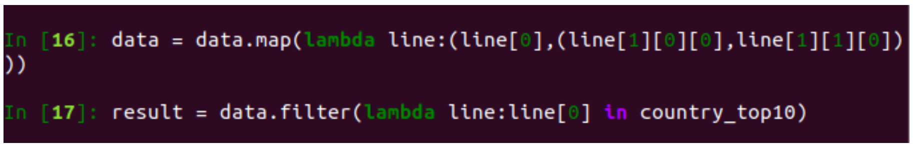
- Create RDD pair((city,birth_month),1) and count by key
- Change RDD pair ((city, birth_month),count) to (city,(birth_month,count))
- Merge RDD pair by key and sort RDD pair by count in value
- Extract top3 last_name for top 10 population city(find in N4)

Process:

Get rdd (city,(last name,count))

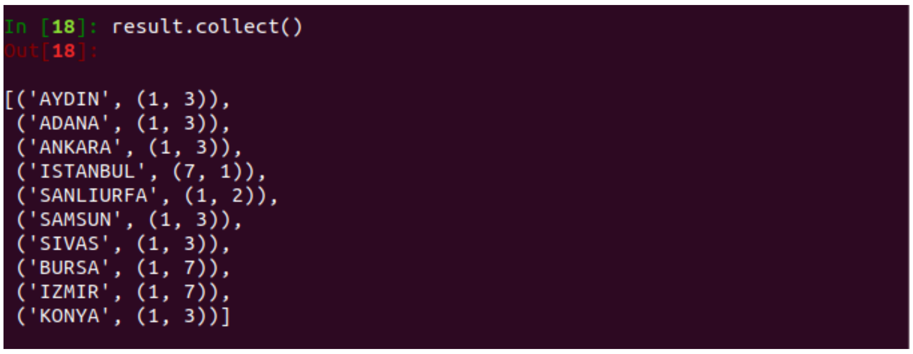


```
In [4]: data = sc.textFile("/home/bigdatalab27/Downloads/mernis/data_dump_t
   ...: emp.sql")

In [5]: data = data.filter(lambda line:line != '')

In [6]: data = data.map(lambda line:line.split("\t"))

In [7]: data = data.map(lambda line:((line[9],extract_month(line[8])),1))

In [8]: data.first()
[Stage 0:>                                                          (0 + 1)

Out[8]: (('MALATYA', 6), 1)

In [9]: country_top10 = ['SANLIURFA','ADANA','AYDIN','SAMSUN','SIVAS','BURS
   ...: A','ANKARA','IZMIR','KONYA','ISTANBUL']

In [10]: data = data.reduceByKey(lambda x,y:x+y)

In [11]: data = data.map(lambda line:(line[0][0],(line[0][1],line[1])))

In [12]: data = data.groupByKey()

In [13]: data = data.map(lambda line:(line[0],sorted(line[1],key = lambda d
    ...: :d[1],reverse=True)))
```



```
In [16]: data = data.map(lambda line:(line[0],(line[1][0][0],line[1][1][0])
))

In [17]: result = data.filter(lambda line:line[0] in country_top10)
```

Result:



```
In [18]: result.collect()
Out[18]:

[('AYDIN', (1, 3)),
 ('ADANA', (1, 3)),
 ('ANKARA', (1, 3)),
 ('ISTANBUL', (7, 1)),
 ('SANLIURFA', (1, 2)),
 ('SAMSUN', (1, 3)),
 ('SIVAS', (1, 3)),
 ('BURSA', (1, 7)),
 ('IZMIR', (1, 7)),
 ('KONYA', (1, 3))]
```

This result has no conflict with the result which is got in E4.